Faculdades Integradas de Taquara - Faccat
Av. Oscar Martins Rangel, 4.500
Taquara, RS, CEP 95600-000

Curso de Sistemas de Informação

UTILIZAÇÃO DA ENGINE DE JOGOS CRYENGINE PARA O DESENVOLVIMENTO DE APLICATIVOS DE ENTRETENIMENTO (JOGOS)

Leonardo Dalmina
Faculdades Integradas de Taquara – Faccat – Taquara – RS – Brasil
ldalmina@faccat.br

Eurico Jardim Antunes
Professor Orientador
Faculdades Integradas de Taquara – Faccat – Taquara – RS – Brasil
euricoja@gmail.com

Resumo

Este artigo apresenta um trabalho de pesquisa que tem como objetivo fornecer documentação sobre a *engine* de jogos *CryENGINE*. A documentação foi criada através da implementação de um conjunto de funcionalidades selecionadas desta *engine*. Estas funcionalidades foram utilizadas para o desenvolvimento de um aplicativo de entretenimento intitulado *Cryshooter*. Os resultados descrevem como cada uma das funcionalidades selecionadas está implementada no aplicativo de entretenimento, ilustrando sua utilização e apresentando uma definição descritiva e analítica sobre estas funcionalidades.

Palavras-chave: jogos, entretenimento, *engine*, *cryengine*.

UTILIZATION OF THE GAME ENGINE CRYENGINE FOR DEVELOPMENT OF ENTERTAINMENT APPLICATIONS (GAMES)

Abstract

This paper presents a research work which aims to provide documentation about the game engine CryENGINE. The documentation was created by implementing a set of features selected from this engine. These features were used for the development of an entertainment application titled Cryshooter. The results describe how each selected feature is implemented in the entertainment application, illustrating its use and presenting a descriptive and analytical definition about these features.

Key-words: *games, entertainment, engine, cryengine.*

1. Introdução

Dentro do conceito de engenharia de *software*, *engine* trata-se da parte do projeto que executa certas funcionalidades para um programa. Dentro da área de jogos, um *engine* se encarregará por entender-se com o hardware gráfico, irá controlar os modelos para serem renderizados, tratará das entradas de dados do jogador, tratará de todo o processamento de baixo nível e outras coisas que o desenvolvedor de jogos normalmente não deseja fazer ou não tem tempo para se preocupar (CLUA; BITTENCOURT, 2005).

De acordo com o portal de entretenimento de jogos IGN (2009), a *CryENGINE* está entre uma das dez melhores *engines* desta geração. Ao contrário de muitos dos seus concorrentes, esta *engine* não necessita de suporte adicional de *middlewares* e pode lidar com sua própria física, som e animações, bem como produzir o visual excepcional visto em jogos consagrados da *Crytek*.

Como pode ser visto em Crydev (2011), na data de 28 de Agosto de 2011 a empresa *Crytek* lançou a versão sem custos e para uso não comercial da sua *engine* de jogos *CryENGINE 3*, a mais recente solução para desenvolvimento de jogos.

Atualmente, o principal problema da *CryENGINE* é a existência de pouquíssima documentação sobre esta ferramenta, quando comparamos com as demais *engines* presentes no mercado. Somado a isso, hoje em dia não é possível encontrar referências bibliográficas consistentes que subsidiem a implementações baseadas na *CryENGINE*, dado o fato da ferramenta ter sido aberta ao público muito recentemente.

Para resolver esses problemas, se propôs a realização de uma pesquisa sobre o desenvolvimento de um aplicativo de entretenimento através da utilização da *engine* de jogos *CryENGINE*. Tal aplicativo de entretenimento foi denominado *Cryshooter*, fazendo uma analogia às palavras *Cry* (presente no início do nome da *engine*) e *shooter* (tradução do idioma inglês para atirador). O aplicativo em questão utiliza quinze funcionalidades selecionadas dentre cinco recursos específicos da *CryENGINE*. Os resultados da pesquisa descrevem como cada uma das quinze funcionalidades está sendo utilizada no aplicativo de entretenimento e apresentam uma análise destas ferramentas, o que aborda o problema de documentação anteriormente citado em um contexto prático de desenvolvimento.

O trabalho possui a seguinte estrutura: a seção 2 o referencial teórico, a seção 3 apresenta a metodologia, a seção 4 os resultados obtidos através da utilização das funcionalidades selecionadas e a seção 5 traz as conclusões do estudo.

2. Referencial teórico

2.1 Game

De acordo com Crawford (1982), *game* é um sistema formal fechado que representa subjetivamente um subconjunto da realidade, uma coleção de partes que interagem entre si, frequentemente de maneiras complexas.

Segundo Curti (2006), uma equipe de desenvolvimento multidisciplinar contempla as áreas de Artes Gráficas, *Game Design*, Gerência de Projeto, *Level Design*, Programação e Sonorização.

De acordo com Azevedo *et al*. (2005), existem 12 gêneros de *games*, que são: Aventura, Estratégia, Esportes, Luta, *RPG* (*Role Playing Game*), Ação, Simuladores, *God Game*, Casual, *Puzzle*, Educacional e *Online*/*Massive Multiplayer*.

A história dos *games* iniciou-se em 1952, quando Alexander Sandy Douglas escreveu uma tese na Universidade de *Cambridge* sobre a interação entre o homem e o computador (IHM), e criou o primeiro *game* de computador, uma versão eletrônica do "Jogo da Velha" chamado *Noughts and Crosses*. Douglas programou o *game* no *EDSAC* (*Electronic Delay Storage Automatic Computer*) que possuía um monitor de tubo de raios catódicos com uma tela que exibia apenas 35 por 16 pixels (BARIFOUSE, 2007).

Em março de 1978, a empresa *Nintendo* produziu seu primeiro *arcade*, chamado de *Computer Othello*, uma versão do jogo de mesa *Othello*. No mesmo ano, dois lançamentos aumentaram ainda mais a popularidade dos *games*: *Football* da *Atari* e o lendário *Space Invaders* para *arcade*, o primeiro *game* em cores da história, importado pela *Midway* e desenvolvido pela *Taito*. Esses dois títulos para *arcade* bateram todos os recordes de vendas (UOL JOGOS, 2009).

Em 2006, *Sony Playstation 3* e o revolucionário *Nintendo Wii*, ambos consoles de sétima geração, foram lançados. O *Xbox 360* recebeu o *game* com os melhores gráficos já vistos em um console até então, o *Gears of War*. A *Microsoft* se tornou o primeiro fabricante a diretamente trazer ao Brasil um console, no caso, o *Xbox 360* (UOL JOGOS, 2009).

2.2 Computação Gráfica

Como pode ser observado em PUCRS (2011), a Computação Gráfica é um conjunto de métodos e técnicas utilizados para converter dados para um dispositivo gráfico, via computador.

A Síntese de Imagens ou Computação Gráfica é uma das principais disciplinas relacionadas ao desenvolvimento de jogos. De acordo com Persiano e Oliveira (1989), Síntese de Imagens é a área que se preocupa com a produção de representações visuais a partir das especificações

geométricas e visuais de seus componentes. A qualidade das imagens sintetizadas e os modelos gráficos aplicados são decisivos para a qualidade visual dos *games*. Em diversas situações os aplicativos de entretenimento, assim como as animações em geral, são responsáveis pela pesquisa e evolução dos conceitos e métodos relacionados à Computação Gráfica enquanto área de pesquisa. Segundo TecMundo (2011), a Pixar (empresa de animações cinematográficas) iniciou suas atividades dedicando-se à pesquisa e ao desenvolvimento de *softwares* de computação gráfica a serem utilizados na produção de filmes.

A Computação Gráfica possui uma ampla variedade de conceitos, como por exemplo: polígono, atores e *cut-scenes*.

Os polígonos formam tudo o que é desenhado em uma cena 3D, desde um cubo ou uma partícula que faz parte de um efeito de fogo a um personagem complexo formado por milhares de triângulos, possuindo uma face com três ou mais lados e vértices (SANTEE, 2005).

Segundo Santee (2005), quase todos os *games* 3D utilizam como padrão os polígonos triangulares, e que, portanto, contêm três lados e três vértices.

Durante o desenvolvimento de um *game*, ator é um termo muito usado nas fases de modelagem e animação de personagens, significando qualquer objeto no mundo virtual de um *game* tridimensional. Personagens controlados pelo jogador, monstros controlados por inteligência artificial, objetos como armas e árvores são exemplos de atores que serão animados ou estáticos de acordo com a necessidade do *game* (SILVA; AGUIAR, 2005).

As *cut-scenes* (cenas pré-renderizadas) são pequenos filmes inseridos em pontos cruciais da maioria dos *games* em 3D, para ajudar a ilustrar a história e despertar interesse no *game*. Essas apresentações também são disponibilizadas na *Internet* para fins de divulgação do *game*, e algumas são, ainda, exibidas em anúncios de TV ou do cinema (SILVA; AGUIAR, 2005).

2.3 Animação de um Personagem

A animação se baseia no princípio da visão humana. Se você olhar uma série de imagens se movimentando rapidamente, vai perceber um movimento contínuo. Cada imagem individual se refere a um *frame* (SILVA; AGUIAR, 2005).

A animação dos personagens pode ser feita através de *softwares* como *Maya*, *3D Game Studio*, *3Ds Max*, além do *Adobe Flash*, ou então através de técnicas como *Motion Capture*, uma tecnologia de captura de movimentos de pessoas ou animais que se utiliza de roupas e equipamentos especiais. Outra técnica é a de Rotoscopia, que por meio da filmagem de um ator ou um animal é possível capturar os movimentos e transformá-los em animação (AZEVEDO *et al*., 2005).

2.4 Level Design

A construção do cenário de um *game* envolve três áreas: artes gráficas, programação e *game design*. Envolve artes porque lida com imagens, programação porque depende da implementação de uma linguagem de programação para "dar vida" ao *game*, fazê-lo acontecer e do *game design* por causa das ideias do *game* (AZEVEDO *et al*., 2005).

Os mundos virtuais em 3D são sempre finitos e quase sempre são fechados dentro de uma "caixa", de modo a dar a sensação de um cenário o qual se amplia no horizonte. Imagens 3D estáticas que combinam com o ambiente são coladas nas paredes internas dessa caixa e circundam o perímetro do *game* (SILVA; AGUIAR, 2005).

2.5 Áudio

Segundo Netmúsicos (2011), a trilha sonora é de extrema importância em um jogo e possui várias funções. Em *games* como *Super Mario Bros.* diversos sinais são dados através do áudio. A trilha sonora dos games não serve apenas como "plano de fundo". Cada canção, cada efeito sonoro, está em seu lugar, cumprindo sua função de ambientar o jogo e tornar a experiência mais realista.

Ao implementar o áudio em *games* deve-se utilizar uma série de elementos fundamentais para que o *game* e o áudio estejam sempre em harmonia (AZEVEDO *et al*., 2005), a saber: i) Atmosfera ou Ambiente: um *game* é na verdade um mergulho em outro mundo, um ambiente diferente de onde o jogador está; ii) Sons Externos: são todos aqueles que ocorrem fora do ambiente em que está acontecendo o *game*. Se o jogo acontece em uma grande cidade, por exemplo, buzinas e sirenes são sons frequentes; iii) Sons Elementares: são elementos realistas que não podem faltar ao *game*. Por exemplo, quando o personagem anda, deve-se colocar o som dos seus passos de acordo com a velocidade, calçado utilizado e o piso do local; iv) Sons de Atrito: colisões, choques, empurrões, carga de arma, galho quebrando ou portas batendo.

2.6 Programação

Programação se refere ao código que está em qualquer *game* ou *software* (UNIDEV, 2009).

O código do *game* contém as definições e os comportamentos para todos os elementos que o compõem, além de todas as suas regras (SILVA; AGUIAR, 2005).

Nos primórdios do desenvolvimento de *games*, a programação era realizada de maneira muito precária porque os computadores da época tinham uma capacidade de processamento muito limitada. Em virtude disso, a linguagem utilizada era o *Assembler* (PERUCIA *et al*., 2005).

Hoje em dia, a linguagem *Assembler* de um modo geral não é muito utilizada, somente em alguns casos em que partes críticas do código exigem performance (PERUCIA *et al*., 2005). Atualmente os programadores em geral costumam trabalhar com a linguagem *C++*, embora outras linguagens como *C#*, *Java*, *Ruby* ou *Python* também sejam empregadas (UNIDEV, 2009).

2.7 Game Engine

A *engine* é um componente de software que interage com os dispositivos de entrada (como *joystick*, *mouse* e teclado) recebendo um conjunto de dados; processando esses dados de entrada, calculando a lógica do *game* incluindo a movimentação de personagens controlados pelos usuários e pelo computador; e apresentando a imagem do estado atual do *game* em um dispositivo de vídeo. Uma *engine* pode também receber informações remotas, enviar mensagens para um servidor e reproduzir sons do *game* (CLUA; BITTENCOURT, 2005).

O uso de *engines* proporciona certas vantagens: o desenvolvimento de diversos *games* através da mesma *engine*, facilidade de reaproveitamento do código desenvolvido em projetos anteriores e integração entre código e modelagem 3D de forma facilitada (CLUA; BITTENCOURT, 2005).

A escolha da *engine* a ser utilizada pela equipe de desenvolvimento é muito importante e há diversos fatores que precisam ser observados (CLUA; BITTENCOURT, 2005), a saber: i) Gênero de *game* a ser desenvolvido: a maioria das *engines* possui uma série de características que favorecem o desenvolvimento de um gênero específico de *game*; ii) Documentação oferecida: uma documentação eficiente é um requisito para que o programador possa utilizar adequadamente os recursos oferecidos pela *engine*; iii) Ferramentas disponíveis: os programas usados pela equipe de desenvolvimento devem ser compatíveis com a *engine* a ser adotada.

Um dos benefícios das *engines* é o conceito WYSIWYP. Segundo GameReporter (2011), o significado desta sigla é "o que você vê é o que você joga". Deste modo, o que o desenvolvedor vê em ambiente de desenvolvimento de um *game* é o que ele irá jogar em ambiente de execução.

2.8 Inteligência Artificial

De acordo com TecMundo (2008), Inteligência Artificial (IA) é um ramo da ciência da computação que se propõe a elaborar dispositivos que simulem a capacidade humana de raciocinar, perceber, tomar decisões e resolver problemas, enfim, a capacidade de ser inteligente.

Para os desenvolvedores de jogos eletrônicos, as aplicações computacionais de IA e o significado do termo IA são diferente dos encontrados no meio acadêmico. Para distinguir a IA

utilizada em jogos e no meio acadêmico, os desenvolvedores adotaram o termo *Game AI* (FUNGE, 2004). A *Game AI*, no jogo *Cryshooter*, permite que este responda a tarefas complexas, como por exemplo, a simulação de um oponente sem a necessária intervenção de um humano.

A principal diferença entre a IA acadêmica e a IA para jogos é o objetivo que cada uma busca. No primeiro caso, o objetivo é buscar a solução para problemas extremamente difíceis, como imitar o reconhecimento que os humanos são capazes de realizar (reconhecimento facial e de imagens e objetos, por exemplo), entender e construir agentes inteligentes. No segundo caso, o objetivo é a diversão. Sua importância é quanto aos resultados que o sistema irá gerar, e não como o sistema chega até os resultados (SCHWAB, 2004).

3. Metodologia

A partir da problematização, foi desenvolvido um aplicativo de entretenimento através da *engine* de jogos *CryENGINE* que utiliza quinze funcionalidades específicas da ferramenta. Tal aplicativo de entretenimento denominou-se *Cryshooter*.

Para organização da etapa de desenvolvimento do jogo *Cryshooter*, adotou-se o método de desenvolvimento ágil e a metodologia (*Framework*) de gerenciamento de projetos Scrum. Segundo Schwaber e Sutherland (2011), Scrum é fundamentado nas teorias empíricas de controle de processo, ou empirismo. O empirismo afirma que o conhecimento vem da experiência e de tomada de decisões baseadas no que é conhecido. O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos. O desenvolvimento foi organizado seguindo os conceitos do Scrum, a saber: i) Foi criado um produto *Cryshooter* e este foi vinculado a uma equipe; ii) Foram criados dez *sprints*, cada um com duração de 30 dias; iii) Nos cinco primeiros *sprints* foi criado um item de *backlog* (um para cada recurso da *CryENGINE*); iv) Os demais *sprints* receberam os itens de *backlog* que não tiveram todas as tarefas concluídas; v) Cada funcionalidade selecionada da *CryENGINE* correspondeu a uma tarefa distinta, sendo esta criada abaixo do item de *backlog* que faz referência ao respectivo recurso da ferramenta.

A arquitetura de *software* utilizada foi a Cliente X Servidor, pois é a abordagem sugerida pela *CryENGINE*. Atualmente, diversos *games* adotam esta arquitetura como forma de permitir a pesquisa de informações dos usuários de *games* e a constante sugestão de atualizações, além de proporcionar que estes interajam entre si nos cenários oferecidos pelos *games*. Segundo Pressman (2006), a arquitetura do software fornece uma visão holística do sistema a ser construído. Mostra a estrutura e a organização dos componentes de *software*, suas propriedades e as conexões entre elas.

Para criar os artefatos (diagramas e documentos) gerados durante a fase de análise e projeto do *Cryshooter*, foi utilizada uma ferramenta chamada Astah Community (CHANGE VISION,

2012), que possibilita a modelagem UML. Os artefatos escolhidos para este tipo de modelagem foram os diagramas de Caso de Uso e de Atividades, por serem mais adequados para a organização e documentação do desenvolvimento do aplicativo de entretenimento. O diagrama de Caso de Uso descreve todas as interações do jogador (ator) com o sistema (jogo). Já o diagrama de Atividades descreve os diversos eventos existentes e as atividades executadas pelo sistema, bem como seus estados e transições de uma atividade para outra.

3.1 Análise

A análise deste aplicativo de entretenimento se iniciou com a análise de requisitos. Como o *Cryshooter* é um jogo de tiro, a análise de requisitos se iniciou com um levantamento (baseado em experiências práticas) de comandos comuns que normalmente um jogador executa ao jogar um jogo deste gênero. A partir dos requisitos, foram identificadas quais as interações que o jogador tem com o jogo. A Figura 1 exibe o diagrama de caso de uso do aplicativo de entretenimento.

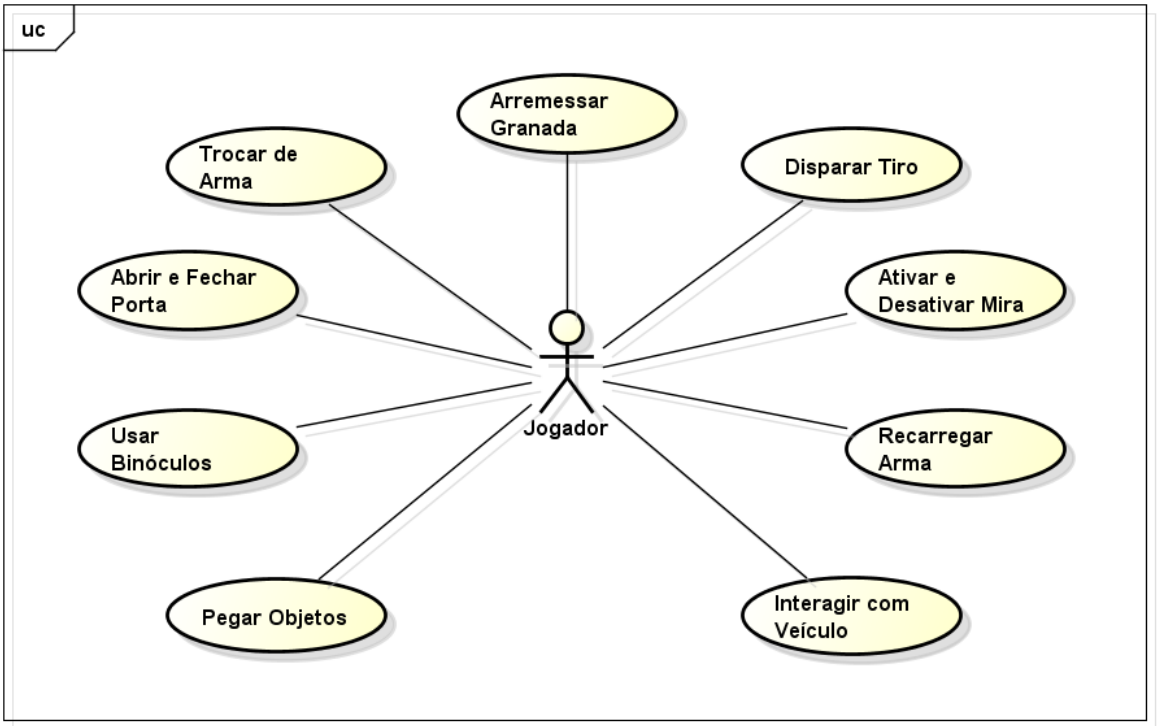

FIGURA 1 – Diagrama de Caso de Uso (Ações do Jogador)

3.2 Modelagem

A modelagem deste jogo contempla informações relativas à criação das regras, objetivos e elementos que definem, por exemplo, a dinâmica de interação do jogo.

O jogo *Cryshooter* tem apenas um único objetivo: eliminar 90 dos 100 inimigos espalhados pelos dois *levels* na menor quantidade de tempo possível. Em razão disso, não existe quantidade de vidas, portanto o jogador pode morrer quantas vezes forem necessárias sem que haja penalizações.

Dos 100 inimigos existentes no jogo, 38 estão presentes no *Level* 01 e 62 estão no *Level* 02. Em cada *level*, o jogador possui uma ajuda especial para eliminar a grande quantidade de inimigos que o cerca. No *Level* 01, ele conta com 24 aliados para combater os inimigos. Já no *Level* 02, está à disposição um Veículo que ajuda a danificar os oponentes.

Para controlar o objetivo principal do jogo, foram adicionados a ele dois elementos de vital importância: um cronômetro para marcar o tempo transcorrido e um contador de mortes. O contador de mortes só é incrementado quando um jogador da equipe inimiga é morto. A Figura 2 exibe o diagrama de atividades, focado nas principais atividades do jogo *Cryshooter*.

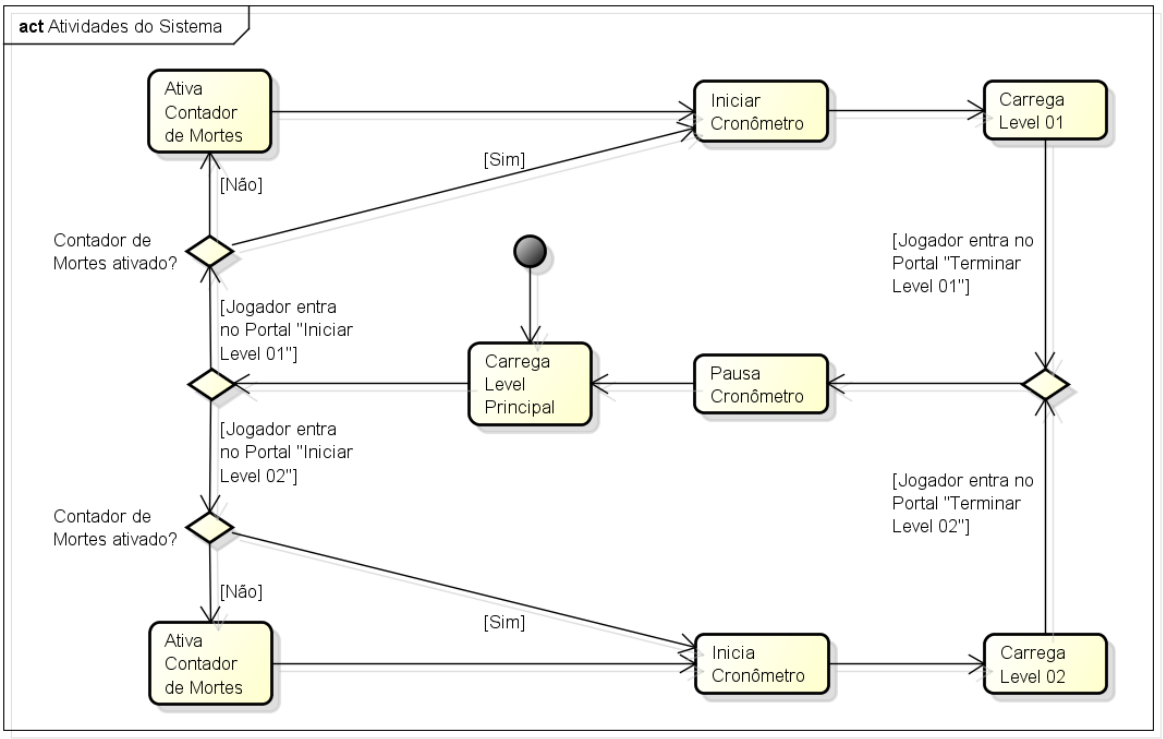

FIGURA 2 – Diagrama de Atividades (Atividades do Sistema)

Pode-se observar na figura acima que o jogador pode entrar e sair de um *level* quantas vezes quiser, porém seus dados só são carregados uma única vez, e desta forma, quando o jogador entrar pela segunda vez no *level* não precisará eliminar os mesmos inimigos que eliminou anteriormente, apenas os que permaneceram vivos.

Uma diferença importante entre o contador de mortes e o cronômetro é que o contador de mortes só é ativado uma única vez durante todo o jogo, enquanto que o cronômetro é ativado e desativado toda vez que o jogador entra e sai dos *Levels* 01 ou 02, respectivamente.

3.3 Desenvolvimento

O desenvolvimento deste aplicativo de entretenimento foi realizado inteiramente através da *engine* de jogos *CryENGINE*. Para isso, foram utilizadas 15 funcionalidades específicas pertencentes a cinco recursos distintos da *engine*.

A estrutura do jogo *Cryshooter* está dividida em duas partes fundamentais: a parte lógica e a parte visual. Para a parte lógica, foi utilizada apenas a funcionalidade correspondente à subseção 4.8, que oferece um sistema para criação de fluxogramas. Para a parte visual, foram utilizadas todas as outras 14 funcionalidades correspondentes às demais subseções da seção 4.

4. Resultados

O desenvolvimento teve como resultado um aplicativo de entretenimento, denominado *Cryshooter*. Nesta seção está descrito como cada uma das quinze funcionalidades está sendo utilizada no aplicativo de entretenimento.

4.1 Vegetação Integrada e Sistema de Geração de Cobertura de Terreno

Toda a vegetação do jogo foi feita utilizando esta funcionalidade. Através dela, foi possível cobrir diversas partes do mapa do jogo com vegetações variadas. A Figura 3 ilustra o resultado da funcionalidade aplicada no cenário principal do jogo.

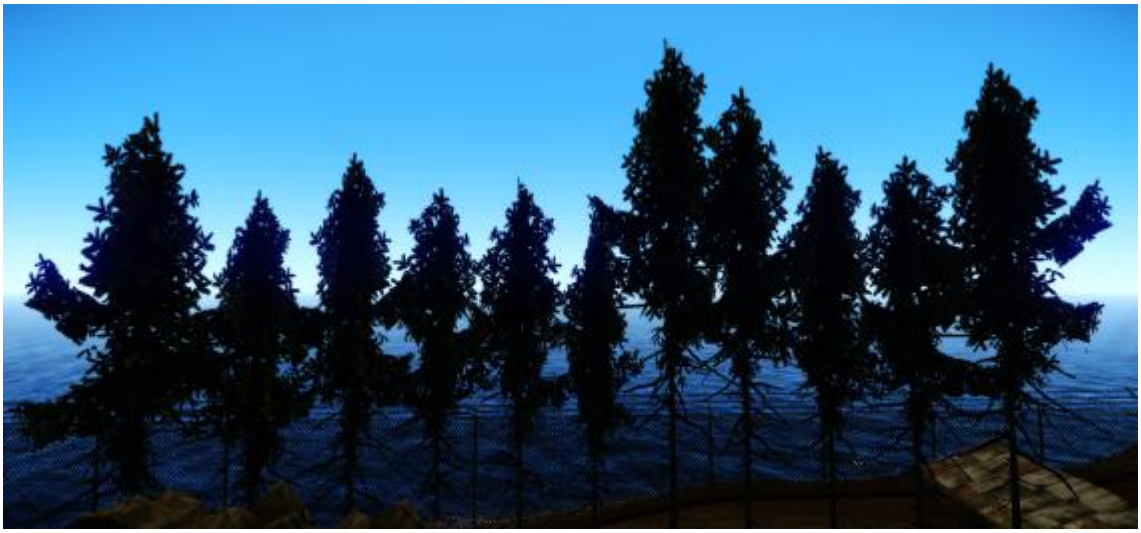

FIGURA 3 – Vegetação Integrada e Sistema de Geração de Cobertura de Terreno

Pode-se observar na figura acima que é possível controlar a rotação dos objetos que serão adicionados ao terreno, bem como a variação dos seus tamanhos. Isso pode ser feito através do módulo *Vegetation*, atribuindo um valor para o "*Brush Radius*" (tamanho do círculo que representa

o local da superfície que receberá a vegetação) e selecionando qual vegetação será utilizada em uma lista de objetos (do tipo vegetação) estruturalmente organizada.

Outros dois pontos importantes a serem considerados são os atributos *Bending* e *Density*. O primeiro faz o objeto balançar, simulando um efeito de vento passando por ele. Quanto maior o valor *bending*, maior será a intensidade com que o objeto irá balançar. O segundo corresponde à densidade do objeto no momento em que ele for adicionado ao terreno, tratando-se neste caso do tamanho do espaço que ele ocupará no terreno (independente do seu tamanho visual). Quanto maior for o valor *density*, mais espaço o objeto irá ocupar. Como os objetos não se sobrepõem, se o valor *density* for bastante elevado, haverá muito espaço entre os objetos de vegetação (chegando ao limite de nem serem adicionados caso a densidade ocupe um espaço maior que o valor "*Brush Radius*"), todavia se o valor for muito baixo, os objetos ficarão extremamente próximos, portanto definir um valor adequado se faz necessário para um cenário mais realista.

Agilidade no processo de criação, fácil parametrização e alto realismo podem ser considerados os pontos positivos desta funcionalidade. A falta de um aviso ou mensagem de erro ao utilizar um valor inadequado nos atributos como, por exemplo, o atributo *density*, é um ponto onde a ferramenta poderia melhorar.

4.2 Sistema de Partículas Suaves em Tempo Real e Editor de FX Integrado

O sistema de partículas foi responsável pela criação de efeitos como fogo, água e fumaça no jogo. Através dele, é possível controlar como a partícula irá se comportar, qual a quantidade de partículas que serão propagadas pelo objeto, e também a intensidade destas. Há ainda as partículas que emanam sons, de acordo com seu respectivo efeito. A Figura 4 ilustra o resultado da funcionalidade aplicada no *Level* 02 do jogo.

FIGURA 4 – Sistema de Partículas Suaves em Tempo Real e Editor de FX Integrado

A figura 4 exibe uma cachoeira. A cachoeira, a primeira vista, parece ser um elemento único. Todavia, no jogo *Cryshooter*, ela foi criada utilizando três tipos de partículas diferentes, que são: *middle_rapids*, que faz o efeito da água respingando acima das pedras, *fall_Wide*, que faz o efeito de água fluindo para um determinado ponto, e *bottom_splash*, que dá o efeito da água colidindo com uma superfície.

O alto controle sobre as características das partículas bem como a simplicidade de uso podem ser considerados os pontos fortes desta funcionalidade.

4.3 Ferramentas Dedicadas para Estrada e Rios

Todas as estradas e rios criados no *Cryshooter* foram feitos utilizando esta funcionalidade. Seus princípios de criação são muito similares. Quando se cria uma estrada utiliza-se o objeto *Road* desenhando uma linha que corresponde ao trajeto da estrada. Para a criação de rios, utiliza-se o objeto *WaterVolume* desenhando uma figura poligonal 2D. A Figura 5 ilustra o resultado da funcionalidade aplicada no *Level* 02 do jogo.

FIGURA 5 – Ferramentas Dedicadas para Estrada e Rios

A figura acima exibe a utilização de uma estrada e um rio em um só cenário. Ambos os objetos, depois de desenhados, precisam receber um material. A parte mais escura da estrada é apenas o terreno do mapa levemente elevado, a parte mais clara é o material aplicado na estrada, que segue a linha que foi desenhada na criação desta. Em um nível um pouco mais abaixo está o rio, que recebeu o material que lhe dá o efeito de água.

No jogo *Cryshooter* existem dois tipos de água: um é o oceano, que está presente em todo o mapa. O outro são os objetos *WaterVolume*, identificados aqui como rios. Estes rios estão presentes no *Level* 02 do jogo e também atrás do topo da cachoeira (também presente no *Level* 02), que embora não seja uma parte navegável do mapa, aparece na cena final do jogo.

Um ponto importante que deve ser observado ao se utilizar objetos *WaterVolume* é a extremidade do objeto. Uma vez que este objeto é uma figura poligonal 2D, suas extremidades determinam o ponto onde a água acaba, e portanto, devem ser escondidas do jogador para aumentar o realismo do cenário. Uma boa maneira para esconder estas extremidades é posicioná-las dentro de outro objeto. Na Figura 5, as extremidades do rio estão posicionadas dentro do muro do cenário e também dentro das montanhas.

O baixo custo para edição, tanto para a estrada como para o rio, é a característica mais formidável desta funcionalidade, pois mesmo depois do objeto estar pronto, com o material aplicado e já posicionado no cenário, ainda é possível modificá-lo em vários aspectos sem resultar em incompatibilidades no objeto final.

4.4 Criador Dedicado de Veículos

O criador dedicado de veículos permite que um veículo seja modificado e até mesmo criado desde o início. Estas modificações são próprias do comportamento do veículo, diferenciando-se das propriedades que ele possui como muitos outros objetos da *CryENGINE*. No *Cryshooter* foram utilizados dois veículos. O primeiro é o "*HMMWV*", um carro tanque que já vem com a ferramenta. O segundo, apesar de ser visualmente igual ao primeiro, é diferente nas configurações, pois consiste em uma modificação do carro tanque original, tratando-se de um novo veículo, denominado "*WaterproofHMMWV*", uma vez que é um carro tanque impermeável. A Figura 6 ilustra o resultado da funcionalidade aplicada no cenário principal do jogo.

FIGURA 6 – Criador Dedicado de Veículos

A figura acima exibe o carro tanque impermeável. Diferente do modelo original, este modelo não afunda quando está em contato com a água. A figura ilustra um momento quando o jogo já está sendo executado, e o veículo permanece boiando na superfície líquida. O que difere o

carro tanque original (que afunda) do carro tanque impermeável é a configuração de sua densidade. Enquanto o carro tanque original possui 50 unidades de densidade, o carro tanque impermeável possui 10 mil unidades, um valor extremamente elevado que faz com que a estrutura principal do veículo não afunde (isso não se aplica às rodas).

A principal qualidade desta funcionalidade é a maneira como ela separa bem as partes do Veículo em diferentes categorias, o que facilita a organização tanto para edição deste, como para criação de um Veículo totalmente novo, desde o início.

4.5 Iluminação Global Dinâmica em Tempo Real

A iluminação global é uma parte crucial para o realismo do cenário, pois calcula como a luz irá se comportar, de acordo com os objetos que foram adicionados, alterados ou removidos do ambiente. Os objetos responsáveis por fazer este cálculo de luz são conhecidos como *EnvironmentProbes*. A Figura 7 ilustra o resultado da funcionalidade aplicada no cenário *Level* 02 do jogo.

FIGURA 7 – Iluminação Global Dinâmica em Tempo Real

A figura acima exibe a funcionalidade antes e depois de ser aplicada. Para calcular o local de incidência da luz, o objeto *EnvironmentProbe* deve cobrir toda a área que se deseja aplicar a luz. Como o objeto é uma esfera, deve se levar em conta até objetos que estão posicionados no céu do ambiente, uma vez que estes também influenciam na distribuição da luz.

No jogo *Cryshooter* esta funcionalidade foi aplicada no cenário principal e no *Level* 02. Como pode ser visto na Figura 7, na parte da esquerda está o *Level* 02 com o objeto *EnvironmentProbe* desativado. Já na parte da direita, encontra-se ativado, e todos os elementos no cenário agora recebem a quantidade de luz apropriada, assim como a intensidade de sombra projetada por outros objetos é corrigida.

O alto realismo gerado no cenário através desta funcionalidade torna o seu uso indispensável, podendo ser classificado como sua melhor característica. Todavia, por causar um grande impacto na luz, alguma reconfiguração de luminosidade no cenário poderá ser necessária após utilizá-la, para se conservar o equilíbrio e a proporção de luz adequada.

4.6 Sistema de Animação de Personagens

Esta funcionalidade permite que tanto a animação como a criação do personagem seja manipulada. No jogo *Cryshooter* existem quatro personagens diferentes. O primeiro deles é o *agent*, identificado como o próprio jogador. O segundo é o *neutral_male*, um inimigo localizado no *Level* 01. O terceiro personagem é o *allied*, um aliado localizado no *Level* 01. O quarto é o *humanoid*, outro inimigo, porém localizado no *Level* 02. A Figura 8 ilustra os três últimos personagens citados anteriormente.

FIGURA 8 – Sistema de Animação de Personagens

A figura acima exibe, da esquerda para a direita, os personagens *neutral_male*, *allied* e *humanoid*, respectivamente. Os personagens *neutral_male* e *humanoid* já vêm com a ferramenta, já o personagem *allied* é exclusivo do jogo *Cryshooter*, pois foi feito utilizando esta funcionalidade, através da combinação de partes do personagem *neutral_male* com partes do personagem *agent*.

A similaridade de como esta funcionalidade trata os objetos bípedes em relação a outros aplicativos de modelagem de personagens 3D é notável. Uma vez concebida a estrutura de um bípede pela funcionalidade, torna-se extremamente simples manuseá-lo e incorporar a ele novos elementos, bem como alterar elementos já existentes.

4.7 Espalhamento de sub-superfície

Esta funcionalidade permite a utilização de objetos translúcidos dentro do jogo. A luz age de uma forma diferente nesses objetos, pois ao invés de iluminar somente sua superfície, ela o atravessa. A Figura 9 ilustra o resultado da funcionalidade aplicada no cenário principal do jogo.

FIGURA 9 – Espalhamento de sub-superfície

A figura acima exibe uma pedra de gelo muito próxima. Pode-se observar na figura que a luz atravessou o objeto, pois é possível enxergar através dele, ou seja, podemos ver todos os elementos do ambiente que se encontram atrás da pedra de gelo.

A facilidade de uso é o ponto forte desta funcionalidade, bastando-se apenas selecionar um objeto que possua essa camada translúcida e adicioná-lo ao cenário, uma vez que a ferramenta automaticamente exibe o que está além do objeto em suas partes translúcidas.

4.8 Sistema de Edição de IA Amigável e Desenhável

A *CryENGINE* possui um estilo de programação visual por fluxograma, conhecido como *FlowGraph*. Os *FlowGraphs* são criados através desta funcionalidade, sendo responsáveis por conduzir boa parte da lógica do jogo. No jogo *Cryshooter*, os *FlowGraphs* são responsáveis por controlar o cronômetro, ativação e desativação das cenas de introdução e fim, entrada e saída de fases, contagem de mortes etc. A Figura 10 ilustra o resultado da funcionalidade aplicada no jogo *Cryshooter*.

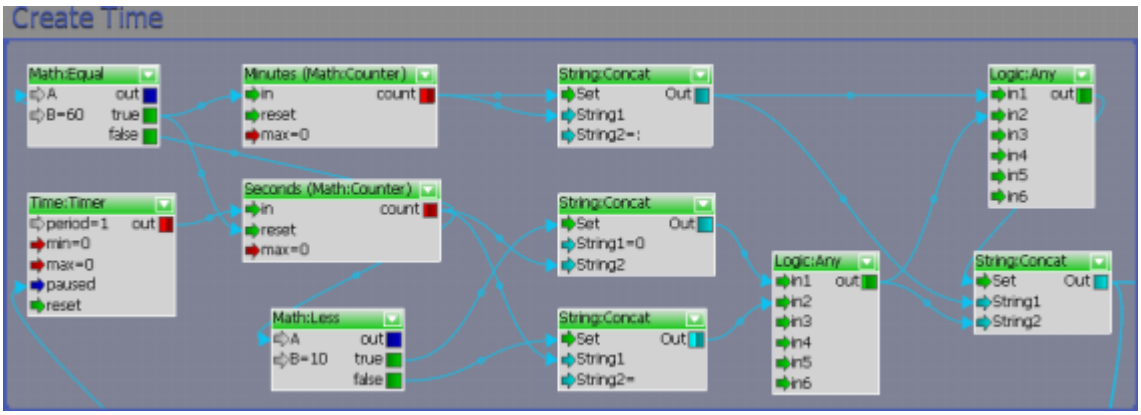

FIGURA 10 – Sistema de Edição de IA Amigável e Desenhável

A figura acima exibe um trecho do fluxograma *ScoreTrigger*. O fluxograma recebe este nome, pois foi criado a partir de um objeto com este mesmo nome, cuja utilização foi somente para a criação do fluxograma. Neste trecho, estão os diversos nodos que compõem a caixa de comentário "*Create Time*". Dentro desta caixa está a lógica responsável pela criação do cronômetro.

Para um melhor esclarecimento do funcionamento do Sistema de Edição de IA, a estrutura e a comunicação dos nodos exibidos na Figura 10 são identificadas da seguinte maneira: um evento externo altera o estado do nodo *Time* (do tipo cronômetro) para parado ou não. Quando não está parado, a cada período de tempo em segundos especificado no atributo *period*, um evento será gerado na sua saída *out*. A partir daí, o nodo *Seconds* (do tipo contador) começa a contagem dos segundos. Toda vez que os segundos são alterados, quatro outros nodos são atualizados. Desses quatro nodos, um serve para verificar se a contagem de segundos é menor que dez, e outro para verificar se é igual a sessenta. Os outros dois nodos são *Strings* (do tipo concatenar), e um exibe os segundos quando forem menores que dez (incluindo um zero na frente), e o outro exibe quando forem maiores, sempre ativados de acordo com o resultado do nodo que verifica se os segundos são menores que 10. Por fim, o nodo que verifica se os segundos atingiram sessenta ativa o nodo *Minutes* (do tipo contador) para começar a contagem dos minutos e também reinicia a contagem do nodo *Seconds*. Este nodo *Minutes* ativa outro nodo *String* para exibir os minutos, que por sua vez vai disparando novos eventos (paralelamente aos outros nodos) para ir concatenando o tempo até um último nodo *String* que irá enviar o valor para nodos fora do "*Create Time*" que irão formatar a hora e atualizá-la na tela para o jogador.

Um cuidado extra deve ser tomado quando se trabalha com os fluxogramas, pois uma vez que estes mudam o comportamento do sistema, dependendo do tipo de alteração realizada (e também da quantidade de uma só vez), há o risco dos eventos no sistema não corresponderem com o que foi programado no fluxograma. Nesse caso, deve-se acessar o item de menu *Generate All AI* (dentro do menu *AI*), que fará com que toda a IA do jogo seja compilada novamente para o ambiente de execução, incluindo qualquer tipo de alteração realizada nos fluxogramas.

Sistema altamente intuitivo, mecanismo de busca por nodos e abstração de codificação em uma linguagem específica são as características mais interessantes desta funcionalidade, pois possuindo apenas a lógica de programação e conhecendo seus elementos básicos, qualquer desenvolvedor pode criar grandes sistemas de lógica através da programação em alto nível oferecida pela ligação de nodos desta funcionalidade.

4.9 Sistema de Tempo do Dia

Esta funcionalidade da *CryENGINE* permite que toda a atmosfera do jogo seja configurável, desde a posição do sol até mesmo seu efeito no ambiente, como sua intensidade, cor refletida no solo, cor do céu, neblina etc. A Figura 11 ilustra o resultado da funcionalidade aplicada no cenário principal do jogo.

FIGURA 11 – Sistema de Tempo do Dia

A figura acima exibe o sol configurado para o jogo *Cryshooter*. Embora a funcionalidade permita que haja dia e noite no jogo, com transições graduais e velocidade parametrizável, neste jogo foi utilizado o horário fixo das onze horas da manhã. Assim sendo, o horário atual, horário de início e horário de fim sempre estão posicionados às onze horas da manhã.

Fácil utilização é o ponto forte desta funcionalidade, uma vez que qualquer projeto já inicia com esta funcionalidade ativada, incluindo a existência de dia e noite, cabendo ao desenvolvedor personalizar, sem muita complexidade, o sistema de tempo do dia conforme sua necessidade.

4.10 Ambientes Interativos e Destrutíveis

Objetos destrutíveis estão presentes no decorrer do jogo *Cryshooter*. Alguns objetos podem se destruir em dois estágios, como por exemplo, o Veículo, que primeiro pega fogo e depois explode. A Figura 12 ilustra o resultado da funcionalidade aplicada no cenário principal do jogo.

FIGURA 12 – Ambientes Interativos e Destrutíveis

A figura acima exibe outro tipo de destruição de objetos. Nela, a vegetação foi destruída, mais especificamente algumas árvores. Pode-se observar que este tipo de objeto não se destrói em estágios, mas sim em partes. Toda vez que o jogador acerta uma grande quantidade de tiros em um mesmo local desse objeto, ele se divide a partir desse local em duas partes. Se uma dessas partes for acertada novamente por uma quantidade de tiros em um determinado ponto, esta parte se dividirá em mais duas, tornando agora o objeto original que era uma árvore única em três partes distintas.

A simplicidade de uso é o ponto positivo desta funcionalidade, uma vez que os objetos destrutíveis tem toda sua lógica de destruição implementada pela *CryENGINE*.

4.11 Ferramentas de Análise de Desempenho

Como ferramenta de análise de desempenho, a *CryENGINE* fornece a geração de arquivos XML com dados estatísticos extremamente detalhados sobre o jogo, para que o desenvolvedor possa visualizar em um único local diversas informações úteis sobre o seu projeto. Esses arquivos XML seguem um padrão de formatação, e podem ser lidos em programas como o *Microsoft Excel*. A Tabela 1 descreve alguns dados referentes à geometria estática do jogo *Cryshooter*.

TABELA 1 – Dados estatísticos do jogo

Filename	Refs	Mesh Size (KB)	Texture Size (KB)
characters/humanoid/plasma_rifle.cgf	62	130	49.338
structures/walls/farm_wall/farm_wall_6m_high.cgf	38	97	9.333
structures/buildings/fishing_houses/fishing_house_door_a.cgf	2	21	9.599
structures/ladders/mp_ladder1.cgf	1	29	127

Todos os objetos presentes no jogo são considerados "*Static Geometry*", ou Geometria Estática. Nos dados apresentados na Tabela 1, pode-se observar quatro tipo de informações diferentes, que são: nome do arquivo que contém o objeto, quantidade de vezes que ele é referenciado no cenário, tamanho em *kilobytes* da malha do objeto e tamanho em *kilobytes* da textura do objeto.

A tabela acima contém a informação de quatro objetos inseridos no jogo *Cryshooter*, que seguindo a ordem apresentada de cima para baixo são: arma acoplada no inimigo (que possui aparência alienígena) e parede de pedra (objetos localizados no *Level* 02), porta e escada para acesso ao telhado da casa de pescador (objetos localizados no cenário principal).

Gerar esses relatórios é extremamente simples, bastando apenas acessar o item de menu "*Save Level Statistics*", dentro do menu *Tools*. Somado a isso, o alto nível de organização dos dados contidos nesses relatórios é o que torna essa funcionalidade essencial para qualquer jogo que necessite ter um desempenho otimizado.

4.12 Renderização Offline

A Renderização Offline contempla um elemento muito notável para os jogos: as *cut-scenes* (cenas pré-renderizadas). Através do módulo "*Track View*" da *CryENGINE* é possível criar quantas *cus-scenes* forem necessárias para o jogo. A Figura 13 ilustra as informações contidas dentro de uma das cenas do jogo *Cryshooter*.

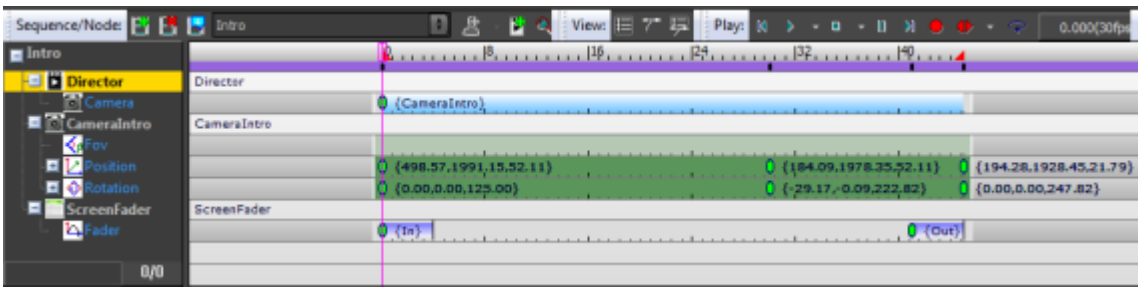

FIGURA 13 – Renderização Offline

A figura acima resume toda a lógica de execução da cena inicial do jogo, que inicia assim que o jogo é carregado. Na *engine*, essas cenas são identificadas como sequências, portanto, a cena

inicial do jogo é a sequência “Intro”. Para cada sequência, deve ser criada uma câmera, que irá “filmar” a cena em si. A trajetória da câmera é baseada em uma linha de tempo (em segundos), cuja duração é configurada nas propriedades da sequência. Para fazer a câmera se movimentar, deve-se gravar em determinadas posições da linha do tempo a posição e a rotação da câmera, de modo que a *engine* irá, no decorrer do tempo, gradualmente posicionar a câmera de uma posição para outra. Estas posições são obtidas dinamicamente pela *Viewport* (tela de edição do ambiente do jogo) da *engine*, através das coordenadas X, Y e Z atuais do objeto câmera.

No jogo *Cryshooter*, além das posições inicial e final da câmera, ainda há a posição intermediária, que ocorre no segundo número trinta. Foram adicionados também efeitos de transição de imagem, através do *ScreenFader*. Quando o tempo está passando sobre o trecho *In*, a tela está escura e vai clareando gradualmente, quando passa pelo trecho *Out*, a tela está clara e vai escurecendo, também gradualmente, ambas configuradas para duração de quatro segundos.

Uma ótima prática para agilizar o processo de criação das *cut-scenes* é visualizar em tempo real o que está sendo exibido na câmera vinculada a *cut-scene*. A câmera padrão da *CryENGINE* sempre exibe a visão de perspectiva do jogo. Uma maneira de alterar a câmera a ser utilizada é selecionar o seu objeto correspondente no jogo, ir até o item de menu *Switch Camera* (dentro do menu *Display*) e selecionar o item *Selected Camera Object*. Para voltar a utilizar a câmera de perspectiva, basta seguir o mesmo caminho de menus, apenas trocando a seleção final pelo item *Default Camera*.

A alta facilidade de uso e a similaridade desta funcionalidade com outros editores de vídeos são magníficas. Isso se dá pelo fato de que a *cut-scene* fica armazenada em uma linha do tempo, que assim como em outros editores de vídeos, possui diferentes camadas, uma para cada tipo de elemento a ser nela inserido.

4.13 Sons Dinâmicos e Música Interativa

A música interativa é uma parte em destaque no jogo *Cryshooter*, uma vez que todo o jogo possui música de fundo. Esta funcionalidade permite que, de uma forma clara e intuitiva, sejam manipuladas as diferentes músicas que irão tocar no decorrer do jogo, ativadas por determinados eventos e seguindo uma programação lógica. A Figura 14 ilustra a utilização da funcionalidade aplicada dentro de um fluxograma.

FIGURA 14 – Sons Dinâmicos e Música Interativa

A figura acima exibe um nodo "*Music:PlayPattern*" interagindo com outros nodos através de eventos, responsáveis por deslocar o jogador de um ponto do mapa para outro ao entrar no *Level* 01 e também ativar a música de fundo deste *Level*. No jogo *Cryshooter*, este nodo "*Music:PlayPattern*" teve apenas alguns atributos utilizados, que são: *Play* (responsável por iniciar a música), *Stop* (responsável por parar a música), *Pattern* (responsável por guardar a música que irá ser reproduzida) e *Finished* (que dispara um evento quando a música chega no fim). O atributo *Play* é ativado sempre que um evento for recebido, que pode ser tanto o evento do jogador ter entrado no *Level* 01 quanto o evento da música ter chegado ao fim. O atributo *Stop* também é ativado sempre que um evento for recebido, porém estes eventos são externos à caixa de comentário "*Warp To Level 01*", podendo ser tanto um deslocamento para fora do *Level* 01 quanto a execução da cena final do jogo.

Um atributo que deve ser utilizado com cuidado é o atributo *Done*. Este atributo é ativado não apenas quando a música chega ao fim, mas também quando é abortada. Portanto, a menos que seja necessário disparar o mesmo evento para ambas as situações, o correto é utilizar ou o atributo *Finished* ou o atributo *Aborted*.

4.14 Áudio Ambiental

O Áudio Ambiental é uma funcionalidade que, além de ter um uso simples, objetivo e bastante visual, aumenta ainda mais o realismo do jogo *Cryshooter*, caracterizando determinados pontos do jogo com sons ambientais, como por exemplo, o som da chuva presente no *Level* 01 que está coberto por nuvens e com uma relativa quantidade de água caindo do céu sobre todo o *Level*. A Figura 15 ilustra o resultado da funcionalidade aplicada no cenário *Level* 02 do jogo.

FIGURA 15 – Áudio Ambiental

A figura acima exibe o áudio ambiental aplicado em uma cachoeira pequena. A cachoeira é feita com o sistema de partículas, porém como citado anteriormente, algumas partículas não emitem som, o que se aplica a esta partícula em questão. Para dar ao jogador o realismo do som da cachoeira, devemos incluir um objeto no jogo do tipo *AmbientVolume*, que armazenará o som ambiente a ser reproduzido. Todavia, este objeto sozinho não irá emitir um som, ele precisa de uma área para a *engine* saber em que parte do cenário ela deve reproduzir o som. Esta área pode ser tanto um objeto *Shape* (que cria um segmento de linhas) quanto um objeto *AreaBox* (que cria uma caixa com dimensões configuráveis). Ambos os objetos servirão para delimitar o alcance do som, porém como o objeto *Shape* não possui altura, o som não possui limites para esta direção, o que já não acontece com o objeto *AreaBox*, que o som fica limitado a altura da caixa. Uma vez decidido qual objeto será utilizado para emitir o som, basta ligá-lo ao objeto *AmbientVolume* desejado, adicionando ao atributo *Target* do objeto *Shape* ou *AreaBox* o objeto em questão.

Conforme a Figura 15, podemos observar que foi utilizado um objeto *Shape* (sem limites de altura), ou seja, quando o jogador entrar no objeto, gradualmente começará a escutar o som da cachoeira, e quando se afastar, gradualmente notará o som ficando mais baixo, até sumir.

4.15 Modos de Som

Os modos de som adicionam ao jogo *Cryshooter*, de uma maneira simplificada e altamente eficiente, equalizações específicas para um determinado local do jogo. Essas equalizações não são músicas nem sons específicos, apenas tratam-se de ajustes feitos no áudio nativo do jogo, como barulho dos passos do jogador, por exemplo. A Figura 16 ilustra o resultado da funcionalidade aplicada no cenário principal do jogo.

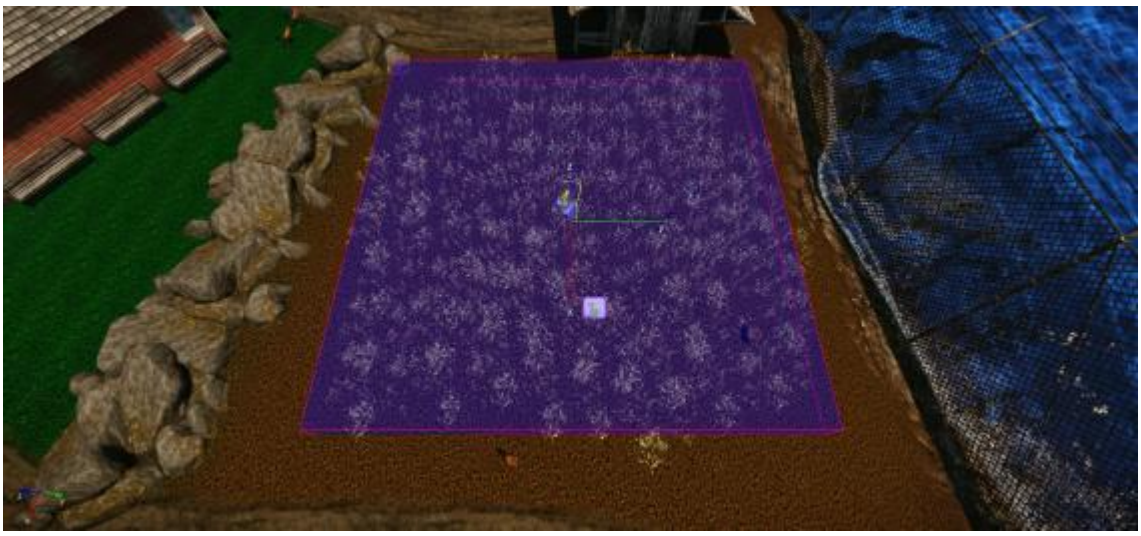

FIGURA 16 – Modos de Som

A figura acima exibe a funcionalidade aplicada sobre uma vegetação específica. Seu funcionamento é muito similar ao da funcionalidade "Áudio Ambiental", a única diferença é que ao invés de ligarmos um objeto *AmbientVolume* ao objeto *Shape* ou *AreaBox*, agora ligamos um objeto *SoundMoodVolume*, que é responsável por armazenar a configuração dos equalizadores do jogo.

Conforme ilustrado na Figura 16, temos um tipo específico de vegetação posicionada em um local onde o jogador pode pisar. Esta vegetação é repleta de pequenos galhos e folhas, portanto uma equalização que represente um tipo de atrito com esta vegetação se faz necessária. Para isso, foi selecionada a equalização *in_cover*. Todavia, para que essa equalização seja utilizada, é preciso armazená-la no atributo *SoundMoodName* do objeto *SoundMoodVolume*. Uma vez armazenada a equalização, basta ligar o objeto *SoundMoodVolume* ao objeto que delimitará os limites do novo áudio, que neste caso, trata-se de um objeto *AreaBox*.

5. Conclusão

Com o presente trabalho pode-se concluir que a *CryENGINE* fornece uma solução ampla para o desenvolvimento de um jogo, indo muito além dos recursos básicos que um *game* contém, oferecendo por exemplo, a utilização de cenas pré-renderizadas, um elemento adicional que muitas vezes é incluído em jogos de maneira textual, ocultando elementos de importância para o *game*.

Através das metodologias aplicadas no desenvolvimento deste trabalho, foi possível gerar uma documentação que até o presente momento não existe e que oferece diversas contribuições, fornecendo informações sobre o desenvolvimento de um *game*, extraindo seus elementos e mapeando-os para as respectivas funcionalidades da *engine*, descrevendo detalhadamente sua utilização e implementação, e possibilitando que o leitor (seja ele um desenvolvedor ou apenas um interessado na área) tenha uma concepção descritiva e analítica destas funcionalidades.

Por fim, a documentação gerada neste trabalho serve não apenas para o entendimento da *engine* e o possível desenvolvimento de um *game*, como também oferece um modelo que pode ser utilizado por outras pessoas para realizar trabalhos similares com outras *engines*.

Referências

AZEVEDO, Eduardo et al. **Desenvolvimento de Jogos 3D e Aplicações em Realidade Virtual**. Rio de Janeiro: Elsevier, 2005. 319 p.

BARIFOUSE, R. **Da Guerra Fria à Batalha dos Consoles**. 2007. Disponível em: <http://epocanegocios.globo.com/Revista/Epocanegocios/0,,EDG77957-8377-5,00.html>. Acesso em: 02 out. 2012.

CHANGE VISION. **Astah Community**. 2012. Disponível em: <http://www.astah.net>. Acesso em: 29 ago. 2012.

CLUA, E. W. G.; BITTENCOURT J. R. 2005. **Desenvolvimento de Jogos 3D: Concepção, Design e Programação**. Disponível em: <http://www.unisinos.br/_diversos/congresso/sbc2005/_dados/anais/pdf/arq0286.pdf>. Acesso em: 28 nov. 2011.

CRAWFORD, C. **The Art of Computer Game Design**. 1982. Disponível em: <http://www.stanford.edu/class/sts145/Library/Crawford%20on%20Game%20Design.pdf>. Acesso em: 28 nov. 2011.

Crydev. **News - Crytek Releases CryENGINE 3 SDK Free-of-Charge**. Disponível em: <http://www.crydev.net/newspage.php?news=72255>. Acesso em 24 nov. 2011.

CURTI, M. M. **Conceitos e Tecnologias no Desenvolvimento de Jogos Eletrônicos**. 2006. 84 p. Trabalho de Conclusão do Curso (Sistemas de Informação) – UNIFEV, Centro Universitário de Votuporanga, Votuporanga, 2006.

FUNGE, John David. **Artificial Intelligence for Computer Games: An Introduction**. Natick: AK Peters. 2004.

GameReporter. **CryENGINE 3, da Crytek, pode ser baixado gratuitamente**. Disponível em: <http://gamereporter.uol.com.br/cryengine-3-da-crytek-pode-ser-baixado-gratuitamente/>. Acesso em: 29 nov. 2011.

IGN. **The 10 Best Game Engines of This Generation**. Disponível em: <http://pc.ign.com/articles/100/1003725p1.html>. Acesso em: 21 ago. 2011.

Netmúsicos. **A importância da música nos games**. Disponível em: <http://www.netmusicos.com.br/a-importancia-da-musica-nos-games/>. Acesso em: 08 dez. 2011.

PERSIANO, R. C. M.; OLIVEIRA, A. A. F. de. **Introdução à computação gráfica**. Rio de Janeiro: Livros Técnicos e Científicos, 1989.

PERUCIA, A. S. et al. **Desenvolvimento de Jogos Eletrônicos: Teoria e Prática**. São Paulo: Novatec, 2005. 320 p.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. Rio de Janeiro: McGraw-Hill, 2006.

PUCRS. **Origens da Computação Gráfica**. Disponível em: <http://www.inf.pucrs.br/~pinho/CG/Aulas/Intro/intro.htm>. Acesso em: 08 dez. 2011.

SANTEE, A. **Programação de Jogos com C++ e DirectX**. São Paulo: Novatec, 2005. 399 p.

SCHWAB, Brian. **AI Game Engine Programming**. Hingham: Charles River Media. 2004.

SCHWABER, K.; SUTHERLAND J. **Guia do Scrum**. 2011. Disponível em: <http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20Portuguese%20BR.pdf>. Acesso em: 29 ago. 2012.

SILVA, J. C.; AGUIAR, F. C. **Modelagem de Personagem para Jogos com 3ds Max 8**. São Paulo: Érica, 2005.

TecMundo. **O que é Inteligência Artificial?**. 2008. Disponível em: <http://www.tecmundo.com.br/1039-o-que-e-inteligencia-artificial-.htm>. Acesso em: 08 dez. 2011.

TecMundo. **Pixar e a reinvenção da computação gráfica**. 2011. Disponível em: <http://www.tecmundo.com.br/cinema/7545-pixar-e-a-reinvencao-da-computacao-grafica.htm>. Acesso em: 04 out. 2012.

UNIDEV. **Guia de Especialidades**. 2009. Disponível em: <http://www.unidev.com.br/phpbb3/viewtopic.php?f=86&t=49131>. Acesso em: 28 nov. 2011.

UOL JOGOS. **A História do videogame**. 2009. Disponível em: <http://jogos.uol.com.br/reportagens/historia/>. Acesso em: 02 out. 2012.