Construindo um framework para jogos FPS com Microsoft XNA

Edson Angelo Mattos Bruno Rabello de Oliveira Esteban Clua

Universidade Federal Fluminense, Instituto de Computação, Niterói, Brasil

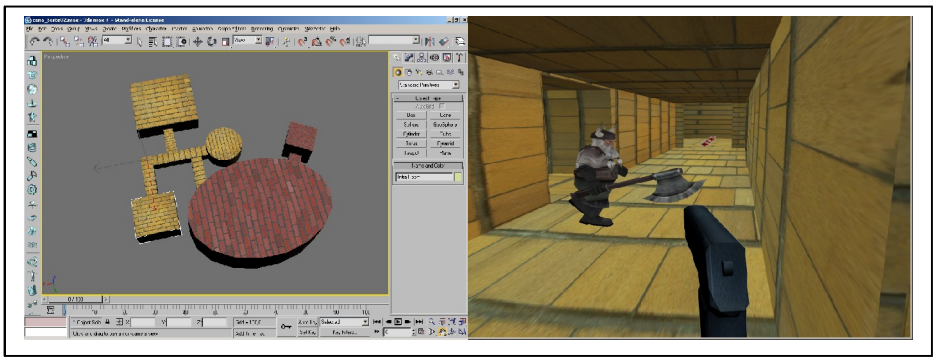

Figura 1: Cenário modelado no 3DS Max, visto de cima (esquerda). O mesmo cenário usando o framework.

Resumo

Este trabalho apresenta alguns aspectos da construção de um framework para o desenvolvimento de jogos de tiro em primeira pessoa (*First-Person Shooter* - FPS) utilizando o Microsoft XNA.

O objetivo deste framework é abstrair certos aspectos que surgem durante a construção de jogos deste tipo quando se utiliza a plataforma XNA, tais como: gerenciamento de cena e simulação de física em tempo real.

Keywords: framework, jogos de computador, física em jogos, first-person shooter, editor de fases, microsoft xna

Authors' contact:
{edsonmattos, uffbruno}@gmail.com
esteban@ic.uff.br

1. Introdução

O gênero FPS constitui um dos tipos mais populares no mundo dos jogos digitais, desde o lançamento do título *Doom* em 1993, pela empresa id Software. Títulos neste gênero ainda são lançados atualmente, para a 7ª geração de videogames [Wikipedia 2007]. Tem-se como exemplos os títulos *Metroid Prime 3: Corruption* (Retro Studios, 2007) para o console Wii (Nintendo), *Resistance: Fall of Man* (Insomniac Games, 2006) para o console PlayStation 3 (Sony), e *Halo 3* (Bungie Studios, 2007) para o console Xbox 360 (Microsoft).

Nesta 7ª geração de videogames, a Microsoft disponibilizou uma plataforma de desenvolvimento gratuita, tendo como público-alvo estudantes, amadores e pequenos grupos de desenvolvimento independentes: o Microsoft XNA. Com o XNA é possível criar jogos digitais, tanto para PC como para o console Xbox 360 [2007]. Ela permite o desenvolvimento de jogos de gêneros diversos, com gráficos bidimensionais ou tridimensionais.

Por permitir a criação de um jogo de praticamente qualquer gênero, o framework do XNA tende a ser genérico o suficiente para tal.

O público-alvo que a Microsoft objetivou para utilizar o XNA, em grande parte, não tem experiência profissional para o desenvolvimento de um jogo FPS. Conhecimento sobre algoritmos de gerenciamento de cena, técnicas de *culling*, simulação da física em tempo real, inteligência artificial para os inimigos e etc. são extremamente necessários para compor um jogo deste tipo.

O framework apresentado neste trabalho visa abstrair estes aspectos, dando mais liberdade ao desenvolvedor para se concentrar na lógica e dinâmica do jogo, além de permitir que *designers* possam construir cenários e personagens e utilizar um aplicativo de modelagem 3D como um editor de fases, como exemplificado na figura 1.

2. Trabalhos relacionados

Não se conhece um framework ou componente para a criação de jogos FPS usando XNA. Existem, no entanto, componentes que podem auxiliar nesta tarefa.

2.1 XNA Quake

Implementação em XNA de uma aplicação que carrega e renderiza mapas de fases no formato MD2, utilizado pelo jogo *Quake* (id Software, 1996). Pode ser utilizado como base de desenvolvimento, porém não tem suporte à *shaders* e não poder ser executado no Xbox 360.

2.2 XNA Box Collider

Cria uma malha de colisão a partir de um cenário 3D, porém não tem suporte a entidades dinâmicas, como armas, inimigos, luzes, munição. Foi utilizado neste framework e será explicado na seção 4.3.

2.3 XNA First-Person Camera

Componente escrito para o XNA que abstrai o posicionamento e rotação de câmera em primeira pessoa. A movimentação da câmera é baseada em eventos do mouse, dispositivo não suportado pelo Xbox 360.

2.4 XNA Animation Component

Permite a utilização de modelos 3D animados em um jogo XNA. Por ser um componente específico para animação, não contém a simulação de física em tempo real, como a gravidade. Também foi utilizado neste framework.

3. Motivação

3.1 Câmera em primeira pessoa

Utilizando o XNA, é possível simular a visualização do ambiente 3D em primeira pessoa. Todavia, requer programação adicional, conhecimento sobre matrizes de rotação e o conceito de cone de câmera (*frustum*), além da gerência da entrada de dados, para responder aos movimentos. Essa simulação corresponde à visão do jogador dentro do mundo virtual apresentado.

3.2 Física

A Microsoft não disponibilizou no framework do XNA algo relativo à simulação de física em tempo real, necessária a vários tipos de jogos. Pode-se utilizar bibliotecas desenvolvidas por terceiros para o XNA, como é o caso da BulletX [2007].

3.3 Interpretador de scripts

Descrever a inteligência artificial do jogo (IA) via *script* é uma prática muito comum [DeLoura 2000], porém com pouco suporte no framework do XNA. É possível utilizar a biblioteca XNuA [2007] que permite traduzir *scripts* escritos em Lua [2007] para código gerenciado em tempo de compilação.

O presente framework optou por usar arquivos de configuração no formato XML, pois não há suporte no .NET Compact Framework for Xbox 360 para interpretação e execução de *scripts* em tempo real.

4. Desenvolvimento

Foram desenvolvidos e integrados ao presente framework módulos que visam suprir os aspectos apresentados na seção 3. Elaborou-se um diagrama de componentes (ver figura 2), que ajuda a compreender como cada módulo se integra ao projeto. Elementos como renderização, reprodução de efeitos sonoros, manipulação de entrada/saída de dados, persistência e biblioteca matemática são providos pelo framework do XNA.

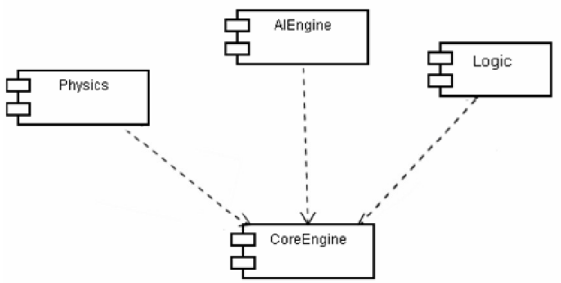

Figura 2: Diagrama de componentes do framework.

4.1 Componente AIEngine

Este componente é responsável pela parte de inteligência artificial do jogo. Contém uma arquitetura para lidar com eventos pré-definidos que possam acontecer no cenário e prover um sistema de mensagens que permita notificar os interessados sobre a ocorrência destes eventos [DeLoura 2000]. Sua utilização se dá, principalmente, para a relação entre jogador e os objetos dinâmicos autônomos (inimigos).

Para simular a autonomia, o comportamento dos inimigos é baseado em estados. Esses estados são definidos pelo *game designer*, e o framework apresentado contêm alguns estados básicos. Quando notificado sobre algum evento, o elemento deve tomar uma decisão, mudando de estado ou não.

Para fazer a transição entre um estado e outro, faz-se uso de uma máquina de estado finita (*Finite State Machine* – FSM).

Os estados pré-definidos neste componente são:

- *Idle State*: quando o elemento está em posição de sentinela;
- *Attack State*: quando o elemento foi atacado ou tem visão do objeto de interesse;
- *Follow State*: quando o elemento tem ciência da existência do objeto de interesse e este objeto ainda está no raio de alcance, o elemento vai ao seu encontro;
- *Die State*: quando a quantidade de vida foi esgotada em função do ataque do inimigo.

Um exemplo de notificação destes eventos é dado a seguir: se uma bala da arma do jogador acerta um inimigo, ocorre uma notificação ao inimigo de que ele foi acertado. Este evento provoca uma mudança de estado do inimigo. Assim, se ele estava no estado de sentinela, a partir de então ele muda para o estado de ataque.

A implementação deste tipo de sistema é simples, porém eficaz, útil e extensível. Há um tipo abstrato presente neste framework, chamado *IGameState*, que fornece um modelo de estado a ser utilizado no jogo.

4.2 Componente Logic

Um jogo FPS contém alguns elementos considerados padrões, como repositor de munição (*ammopack*), reparador de vida com medicamentos (*medpack*), inimigos ou NPCs (*Non-playable character*), armas, chaves para portas. Este componente contém classes concretas que descrevem a parte lógica. A Figura 3 contém um diagrama de classes representando a relação entre as classes do componente e as classes abstratas, pertencentes ao *CoreEngine*.

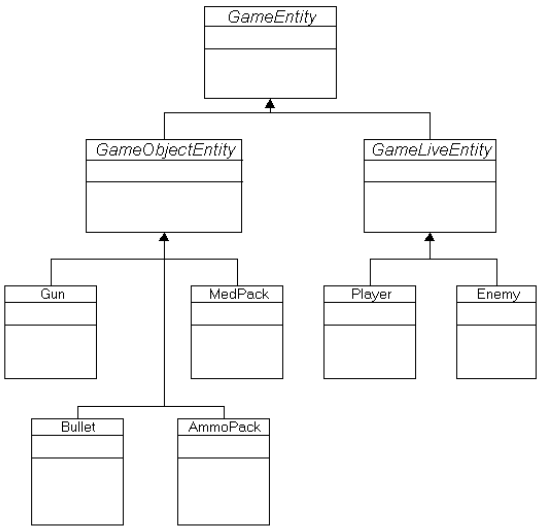

Figura 3: Diagrama de classes simplificado.

GameEntity é uma classe abstrata que contém atributos e métodos comuns a todos os elementos dinâmicos de um jogo FPS, como posição, dados sobre a caixa de colisão (*Axis-Aligned Bounding Box* - AABB) [Lander 2000] e a matriz de transformação de objetos 3D. Dela são derivadas duas classes, também abstratas: *GameObjectEntity* e *GameLiveEntity*, que contêm dados comuns tanto para objetos não-autônomos como para autônomos, tais como vida, saúde e armas.

Com as classes concretas *Gun*, *MedPack*, *Bullet*, *AmmoPack*, *Player* e *Enemy* é possível fazer um jogo FPS básico. A seguir uma breve descrição de cada uma delas:

- **Gun**: representa uma arma com munição de um determinado tipo.
- **AmmoPack** e **MedPack**: classes que representam pacotes (*packs*) com munição de um determinado tipo e medicina, respectivamente.
- **Player**: representa o jogador. Contém uma lista de armas disponíveis, uma câmera de colisão e dados como altura e largura.
- **Enemy**: classe que representa os inimigos. Contém os estados definidos pelo *game designer* através do arquivo de configuração.

4.3 Componente Physics

Este componente é responsável pela detecção de colisão dos elementos do cenário e pela simulação da gravidade. O XNA não provê uma biblioteca de física para este fim, para isso foi utilizado um componente externo, chamado XNA Box Collider, que permite gerar uma malha de colisão utilizando o algoritmo *Octree* [DeLoura 2000], a partir de um cenário 3D. No 3DS Max exporta-se o cenário através do *plugin* Panda X Exporter, para gerar arquivos do tipo X, entendidos pelo Box Collider. Além disso, a câmera FPS foi abstraída para rotacionar e se mover pelo cenário através de comandos do teclado ou gamepad, sempre respeitando a malha de colisão e a gravidade.

O Box Collider utiliza funções providas pelo XNA para determinar caixas de colisão do tipo AABB para cada extensão continua de malha presente no cenário.

O algoritmo de particionamento do espaço 3D *Octree* é eficiente para uso em tempo real porém para elementos estáticos. Para isso, o componente *Physics* contém uma seção dedicada para tratar de objetos dinâmicos: a biblioteca BulletX aplica gravidade, dinâmica de colisão e etc.

4.4 Componente CoreEngine

Componente principal e a base para a construção de um jogo FPS. Ele contém as classes abstratas descritas na seção 4.2, e mais 2 classes concretas: *Scene* e *Log*.

A classe *Scene* é a representação de uma fase do jogo, que contém um cenário estático, os elementos dinâmicos e o personagem principal. Além disso, as fases são independentes entre si, facilitando a gerência de múltiplos níveis. A classe *Log* possibilita a geração de arquivos contendo *logs* de erros, informações e mensagens de depuração.

5. Exemplo de utilização

A figura 1 mostra um cenário estático modelado no 3DS Max e em execução usando o framework. Este cenário foi exportado para um arquivo no formato X.

Cada modelo 3D presente neste cenário recebeu um nome e os objetos dinâmicos são diferenciados por conter um o símbolo "_" à frente. Por exemplo: o anão presente na figura 1 recebeu o nome de "*_Dwarf01*". Esse nome é usado no arquivo XML de configuração para definir os atributos pertinentes ao objeto. No caso do inimigo é necessário definir qual arquivo do projeto contém o modelo animado que o representará no jogo. A figura 4 mostra o relacionamento entre os elementos.

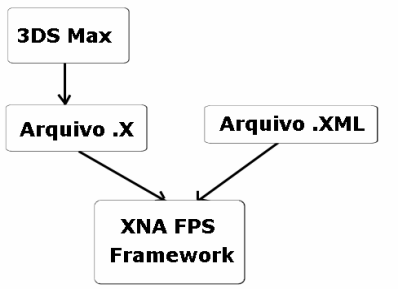

Figura 4: Relação entre os elementos.

É necessário criar um projeto simples no XNA Game Studio Express e importar o arquivo do modelo 3D que representa o cenário, os arquivos dos modelos animados e o arquivo XML para referência no código fonte. Na classe principal, cria-se um objeto do tipo *Scene*, passando como parâmetro o arquivo de configuração e o modelo 3D do cenário.

O objeto *Scene* recém criado é do tipo *GameComponent*, tipo este pertencente ao framework do XNA, e que foi estendida para uso no presente framework. Além disso, é preciso registrá-lo junto à classe principal do projeto como componente da aplicação. Assim, a lógica e a renderização da fase serão atualizadas na mesma freqüência de tempo que o código da classe principal (em média 60 vezes por segundo).

Esta fase de exemplo contém 6 inimigos, 12 *medpacks* e 1 arma. O arquivo XML descreve, para cada um destes elementos, respectivamente: a quantidade de vida inicial; quantidade de medicamento; quantidade de munição inicial, e, quando localizada no cenário, se faz uma rotação e qual a velocidade angular desta rotação. O posicionamento de cada um é definido no modelo 3D do cenário e entendido pelo framework.

6. Conclusão

O Microsoft XNA é uma ferramenta de grande utilidade para desenvolvedores de jogos independentes, provendo mecanismos facilitadores na concepção de diferentes gêneros. O propósito do XNA FPS Framework, aqui apresentado, é adicionar uma camada a mais de abstração para quem deseja desenvolver um jogo do gênero FPS, tornando mais simples a sua concepção através do gerenciamento automático de vários aspectos inerentes, tais como matemática (rotação, translação, escala) e detecção de colisão.

7. Trabalhos Futuros

O framework se encontra em estágio de desenvolvimento e pretende-se continuar o seu desenvolvimento. Pretende-se escrever um *plugin* para 3DS Max que permita gerar, automaticamente um arquivo XML baseado no cenário; escrever um interpretador de scripts simples; expandir e refatorar a arquitetura de classes do componente *Logic*.

Este trabalho se baseou na versão Refresh 1.0 do XNA Framework.

Agradecimentos

Agradecemos ao prof. Esteban Clua pela orientação durante o projeto e ao Fábio Policarpo pela atenção dada.

Referências

WIKIPEDIA, 2007 *History of video game consoles (seventh generation)*. [online] Disponível em: http://en.wikipedia.org/wiki/History_of_video_game_consoles_(seventh_generation) [Acessado em 14 Out. 2007].

XNA QUAKE. Disponível em: http://www.codeplex.com/q3libxna/Release/ProjectReleases.aspx?ReleaseId=3478

XNA BULLETX. Disponível em: http://www.codeplex.com/xnadevru/Wiki/View.aspx?title=Managed%20Bullet%20Physics%20Library.

XNUA. Disponível em: http://www.xnua.com

XNA BOX COLLIDER. Disponível em: http://fabio.policarpo.nom.br/xna/index.htm.

XNA FIRST PERSON CAMERA. Disponível em: http://www.dhpoware.com/demos/xnaFirstPersonCamera.html.

XNA ANIMATION COMPONENT LIBRARY. Disponível em: http://www.codeplex.com/animationcomponents.

XBOX 360. *Especificações técnicas do Xbox 360.* [online] Microsoft Corporation. Disponível em: http://www.xbox.com/pt-BR/support/getstarted/manual/xbox360/xbox360specs.htm [Acessado em 28 Ago. 2007].

DELOURA, M., 2000. *Game Programming Gems 1*. Charles River Media.

LUA, THE PROGRAMMING LANGUAGE. Disponível em: http://www.lua.org/

WIKIPEDIA 2, 2007 *First-person Shooter*. [online] Disponível em: http://en.wikipedia.org/wiki/First-person_shooter [Acessado em 5 Set. 2007].

LANDER, J., 2000. *When Two Hearts Collide: Axis-Aligned Bounding Boxes* [online] Gamasutra. Disponível em: http://www.gamasutra.com/features/20000203/lander_01.htm [Acessado em 1 Sep. 2007].