O Desenvolvimento de um Motor Multiplataforma para Jogos 3D

Aline Bessa	Caio T. Sousa	Carlos E. Bezerra	Ivan Monteiro	Humberto Bandeira	Rodrigo Souza
Indigente	Indigente	Indigente	Indigente	Indigente	Indigente
UFBA	UFBA	UFRGS	UFRGS	UFBA	UFBA

Resumo

O desenvolvimento de jogos modernos é uma atividade que requer conhecimentos multidisciplinares, envolvendo diversas áreas da computação, matemática, física e outras ciências. Os motores de jogos visam a suprir grande parte da demanda computacional, integrando diversas técnicas freqüentemente necessárias nos jogos, simplificando muito a criação de um novo. A implementação desses motores, no entanto, é uma tarefa árdua, visto que uma gama muito grande de funções precisa ser desenvolvida de forma integrada. Este artigo apresenta o motor InGE e analisa algumas das dificuldades enfrentadas e decisões tomadas na implementação deste *software* multiplataforma.

Keywords:: Motor de Jogos, Multiplataforma, Desenvolvimento de Jogos

Author's Contact:

{aline061, caio, nkbeto, rodrigo}@dcc.ufba.br
{kdubezerra, fehler}@gmail.com

1 Introdução

O objetivo de um motor de jogos é agrupar funções fundamentais para o desenvolvimento de jogos, que podem se estender da interação com os periféricos de entrada até a renderização dos cenários e personagens. Assim, várias aplicações podem ser desenvolvidas utilizando como base de código este componente central. Isto certamente reduz o tempo total de produção, à medida que concentra a equipe de trabalho em atividades de mais alto nível. Por mais genéricos que sejam, entretanto, os motores costumam ser projetados tendo em vista uma classe particular de jogos, como 2D ou 3D.

A partir do estudo de alguns motores 3D de código aberto disponíveis no mercado, como Panda3D [Carnegie Mellon University 2007] e Ogre 3D [Torus Knot Software 2007], o grupo de pesquisa Indigente decide, em 2004, desenvolver seu próprio motor: o InGE. Para o grupo de pesquisa, a experiência de desenvolver uma base genérica de código e, em seguida, aplicações para ela, parecia mais enriquecedora do que utilizar um motor consolidado.

O InGE é um motor voltado para jogos 3D em geral, mas otimizado para jogos de ação em primeira pessoa. Ele é implementado na linguagem C++ e serve de base para o desenvolvimento de jogos multiplataforma, sejam eles livres ou comerciais. O InGE foi testado nos sistemas operacionais Linux e Windows e está sob a licença LGPL. Projetado para atender a requisitos de rede, áudio, matemática, física, controle e gráficos, ele se diferencia de alguns motores disponíveis no mercado, que se ocupam exclusivamente com funções de algumas dessas áreas.

O InGE passa por um processo de constante aprimoramento, e, desde 2006, está apto a servir como base de jogos 3D. Neste mesmo ano, também foi realizado o primeiro jogo que utiliza exclusivamente as funções providas por este motor: o *We Are The Champignons* [InDigEnte 2006], premiado com a 3ª posição no *Festival de jogos independentes* de 2006.

Nas próximas seções, o InGE será detalhado sob a perspectiva de objeto de pesquisa. Na seção 2, será dada uma visão geral do *software*; nas seções 3, 4, 5 e 6 serão discutidos seus subsistemas gráfico, de física, de rede e de áudio, respectivamente; na seção 7, serão apresentadas algumas conclusões e planos de desenvolvimento.

2 InGE: visão geral

Nesta seção, serão descritos alguns aspectos que caracterizam o motor InGE: suas funções, as principais bibliotecas e APIs utilizadas e alguns conceitos básicos a ele relacionados.

2.1 Funções e principais bibliotecas e APIs

O motor InGE implementa funções diversas, como detecção de colisão, carregamento de modelos, reprodução de áudio e comunicação via rede. Para auxiliar o desenvolvimento destas funções, este motor utiliza algumas bibliotecas especializadas e consolidadas, como OpenGL [Silicon Graphics Inc. 2006], OpenAL [Loki Software 2007], ODE [Smith 2007] e SDL_net [Lantinga et al. 2007]. As interações entre elas e os subsistemas do InGE serão detalhadas nas próximas seções.

Estas bibliotecas ou APIs permitiram que fosse poupado muito trabalho de baixo nível. Entretanto, integrá-las ao motor não foi uma tarefa trivial: muitas vezes, as bibliotecas ou APIs apresentaram certas incompatibilidades entre si ou não suportaram algumas características desejadas. Portanto, dentre as etapas de desenvolvimento do InGE, esteve incluso o trabalho de adaptá-las às necessidades.

As interfaces do InGE foram definidas visando à utilização do motor sem a necessidade de conhecimento de detalhes do mesmo, permitindo também o uso de funções mais específicas sem inconsistências.

2.2 Conceitos Básicos

O ambiente virtual é representado pelo InGE através de um cenário e diversas entidades. O cenário é a parte estática do ambiente, como chão, paredes ou montanhas. Por fim, uma entidade é qualquer coisa que possui pelo menos uma posição no espaço, uma orientação e uma velocidade. Um exemplo de entidade são os objetos móveis na cena.

Objetos 3D, no InGE, são entidades que estão associadas a um modelo 3D e possuem massa e volume. Com isso, eles podem colidir com outros objetos e são visíveis pelo jogador. Um exemplo típico são itens espalhados pelo cenário. O avatar é um tipo particular de objeto 3D: ele é a representação de um jogador no mundo virtual. O avatar pode ser controlado por uma pessoa ou pela inteligência artificial do jogo.

Entidade é um conceito fundamental no domínio de jogos: jogar é manipular entidades [Zagal et al. 2005]. Por esta razão, elas usam diversas funções do motor. O próximo parágrafo ilustra este fato através da análise de como algumas ações recorrentes em jogos são implementadas com o InGE.

Ação: jogador caminha pelo cenário; **implementação**: avatar tem atributos posição, orientação e velocidade alterados. **Ação**: jogador pega item "kit médico"; **implementação**: o avatar colide com o item; o item é destruído; uma entidade fonte sonora é posicionada próxima ao avatar e toca o efeito "pegar item"; o atributo saúde do avatar é incrementado. **Ação**: jogador atira granada; **implementação**: a entidade granada é criada e posicionada na frente do avatar, com uma velocidade inicial; sob ação da gravidade, o vetor velocidade é alterado a cada instante.

3 Gráficos

O subsistema gráfico do InGE agrega duas grandes áreas da computação gráfica: modelagem geométrica e síntese de imagem. Com

isto, ele oferece funções que se estendem do carregamento de modelos geométricos à preparação da cena para sua renderização. Os principais módulos deste subsistema são: *Base*, *Camera*, *Model* e *World*.

3.1 O módulo Base

O módulo Base cria uma camada de abstração sobre componentes básicos presentes em uma cena, como vértices, malhas, texturas, tipos de material e volumes de envoltória. Como o próprio nome indica, este módulo serve de alicerce para a construção dos demais módulos gráficos.

Está presente também no módulo Base uma funcionalidade para aplicação dinâmica de efeitos aos elementos básicos de uma cena. Ela é obtida através do uso do padrão de projeto *Decorator* [Gamma et al. 1995], que permite que uma malha combine, em tempo de execução, os efeitos a ela aplicados. Muitos desses efeitos estão associados a texturas.

3.2 O módulo Camera

No módulo Camera, destaca-se a interface ICamera. Ela permite o tratamento unificado para implementações de câmeras distintas. Ainda neste módulo, são fornecidas implementações de câmeras freqüentemente utilizadas em jogos de ação, como as câmeras de primeira pessoa, em que o ponto de visão simula o do personagem. Outro tipo de câmera provido pelo InGE se baseia em *Spline* e é útil para tomadas de cenas. A representação através de uma interface única simplifica a utilização, em tempo de execução, de câmeras distintas para a visualização de uma mesma cena.

3.3 O módulo Model

No módulo Model, é feito o carregamento de modelos geométricos, bem como sua renderização. Os modelos geométricos carregados neste módulo são apenas animados por quadro-chave, ou seja, um mesmo modelo pode ser armazenado em diferentes posições, como uma fotografia tridimensional do seu movimento. Isto permite a composição de uma animação através da interpolação destes quadros-chave. Atualmente, o InGE suporta o carregamento do formato MD3, que permite anexar mais de um modelo para compor, por exemplo, um personagem de um jogo. Desta forma, são carregadas estruturas distintas para cabeça, tronco e pernas do personagem que, quando anexadas, formam um único modelo.

3.4 O módulo Render

Este é o módulo responsável pela síntese das imagens. A classe principal deste módulo é a *RenderManager*, que gerencia toda a renderização na tela e ordena os comandos que deverão ser passados para OpenGL[1]. A *RenderManager* controla a iluminação da cena, a renderização da GUI[2], do cenário e dos modelos. A fim de, futuramente, habilitar InGE a oferecer suporte a outras APIs gráficas, foi criada uma camada de abstração na classe *Drawer*. Todos os comandos de acesso à API gráfica são feitos através da interface de *Drawer*, que atualmente usa OpenGL através da classe *DrawerOpenGL*. Deste modo, é possível criar, por exemplo, a classe *DrawerDirectX*, fazendo com que *Drawer* interaja também com DirectX [Microsoft 2007].

3.5 O módulo World

O módulo World é responsável pela organização da hierarquia de cena. Sua principal função é fornecer um estrutura que permita a renderização de cena em tempo real, eliminando ao máximo a renderização de polígonos não visíveis. Para atingir tal objetivo, o módulo World implementa uma combinação de árvore BSP[3] e grafo de visibilidade. As duas técnicas combinadas trazem grandes vantagens para jogos que possuem ambientes fechados[4], permitindo uma melhor seleção do que deve ser renderizado na cena. No BSP, é feito o particionamento de um espaço em dois sub-espaços através de um plano. Cada sub-espaço recebe uma envoltória convexa e vai sendo dividido em dois novos sub-espaços até um critério de parada. Isso permite que, no momento da renderização, seja feito um percurso na árvore, onde são descartados os ramos em que a envoltória convexa do sub-espaço está fora do volume de visualização. Para complementar esse percurso, cada sub-espaço presente nas folhas dessa árvore forma um nó de um grafo de visibilidade[5]. Assim, uma vez definido em qual sub-espaço está a câmera, é possível checar quais sub-espaços são significativos para a renderização.

4 Física

O subsistema de Física do InGE agrega, principalmente, funções relativas à Mecânica. A biblioteca adotada para servir de suporte às implementações é a ODE. Basicamente, este subsistema está dividido em dois módulos: um voltado para simulação de corpos rígidos e outro voltado para cinemática e dinâmica.

4.1 Biblioteca adotada

A ODE, desenvolvida em C++, possui suporte a desenvolvimento 3D e é distribuída sob as licenças BSD e LGPL. Esta compatibilidade de licenças foi um fator importante para a sua escolha como dependência. Além disso, dentre as alternativas livres, ODE é uma das mais populares no mercado. O jogo *Call of Juarez* [Ubisoft 2007] é um exemplo de aplicação desta biblioteca.

4.2 Cinemática e Dinâmica

Apesar de muito utilizada em grande parte das tarefas pertinentes à cinemática e à dinâmica, a ODE é sub-aproveitada no InGE, pois existem incompatibilidades entre ela e a maneira como o motor lida com colisão. O InGE não calcula todos os pontos de contato requeridos pela biblioteca. Não dispor destes dados impede, em determinados momentos, que a biblioteca calcule as forças resultantes corretas.

Esta incompatibilidade entre informações requeridas pela ODE e providas por InGE possui solução de alto custo. No que diz respeito a mapas em formato BSP, seria especialmente trabalhoso mapear mais pontos de contato através do motor, dada a forma como ele se desenvolveu. Utilizar o vetor velocidade linear de cada objeto envolvido em uma colisão, algo que não requer muitos pontos, é uma alternativa viável ao cálculo da força resultante provido pela ODE, apesar de não ser tão realista. Atualmente, esta é a solução provida por InGE para determinar a dinâmica do sistema. Gravidade e impulso, por exemplo, são simulados através da aplicação de velocidade linear ao objeto.

A detecção de colisão no motor utiliza apenas um ponto de contato e árvores BSP. Além de aplicações para o sistema gráfico, esta estrutura de dados serve para averigurar qual é a posição relativa entre um dado ponto de contato e uma série de planos do mundo, definidos no decorrer do percurso na árvore, da raiz às folhas.

4.3 Simulação de objetos

A interface de física do InGE utiliza bastante a ODE para simulação de objetos, diferentemente do que ocorre com o tratamento de colisão. Neste ponto, o motor funciona principalmente como uma camada de abstração entre o usuário e a ODE, a fim de aumentar a sua usabilidade. Assim, os conceitos de corpo rígido[6], articulação[7]

[1] A escolha por OpenGL se deve ao fato de o InGE ser projetado para ser multiplataforma, além de ser desenvolvido em Software Livre. OpenGL é compatível com os maiores sistemas operacionais de computadores pessoais e consoles.

[2] Graphic User Interface — Interface Gráfica com Usuário.

[3] *Binary Space Partition.* - Partição Binária do Espaço

[4] Ambientes que possuem uma envoltória convexa.

[5] Grafo em que os nós representam regiões e os arcos representam que um nó é visível a partir de outro.

[6] Um corpo rígido possui propriedades como os vetores posição, ajustado ao centro de massa, e velocidade linear, além de matriz de inércia.

[7] Articulações conectam corpos rígidos e permitem que eles assumam apenas algumas posições relativas entre si.

e geometria[8] são abordados no InGE de maneira similar à apresentada pela biblioteca.

5 Rede

Jogos multijogador têm se popularizado bastante. Surgiram, inclusive, jogos *online* massivamente multijogador, que fazem uso intensivo da rede devido ao grande número de participantes envolvidos. Um exemplo desta categoria de jogos é *World of Warcraft* [Blizzard 2004]. Além deles, há também jogos que podem ser jogados de maneira solitária, mas que foram criados preferencialmente para funcionarem em rede, tais como *Counter-Strike* [Valve Software 1999]. Neste contexto, o InGE provê suporte a rede em um de seus principais subsistemas.

5.1 Biblioteca

Existem diversas bibliotecas para suporte a rede em jogos, tais como HawkNL [Frisbie 2004], SDL_net e enet [Salzman 2002]. Para implementação do suporte a rede no InGE, foi escolhida a biblioteca SDL_net, por ter uma interface de programação simples, se comparada com a interface dos sockets POSIX, e suficiente para a implementação das funções desejadas. Além disso, faz parte da SDL [Lantinga 2007], que é multiplataforma e um padrão no motor.

5.2 Arquitetura

Após a escolha da biblioteca a ser utilizada, foi feita a escolha da arquitetura de rede suportada pelo motor, que foi a de cliente-servidor. Os critérios de escolha foram consistência, segurança e simplicidade de implementação. Segurança refere-se a resistência contra trapaça dos jogadores. É mais simples verificar se o estado dos jogadores está consistente, ou se houve trapaça, se há um ponto central que faça tal julgamento. Além disso, em uma rede descentralizada seria necessário algum protocolo mais complexo que fizesse os dados serem trocados entre todos os participantes do jogo de forma consistente [Hu and Liao 2004].

5.3 Protocolo de transporte

Outra escolha foi a do protocolo de transporte a ser utilizado. O protocolo TCP impõe um sobrecusto para que as mensagens sejam ordenadas e para garantir sua entrega, o que pode tornar as atualizações de estado mais lentas [Honeycutt et al. 2003]. Foi escolhido o UDP, que apesar de não oferecer tais garantias, é mais interessante para jogos em rede. Sobre o UDP, foi implementada uma conexão virtual entre cliente e servidor. O ordenamento de mensagens foi implementado de forma simples sobre o UDP, e a perda de pacotes foi mascarada por *dead-reckoning* [Smed et al. 2001]. Foi garantido também que as mensagens estivessem em ordem, para que a seqüência de estados de determinada entidade do jogo fosse coerente com as ações efetuadas pelo jogador remoto que controla aquela entidade.

No InGE, cada mensagem de atualização de estado de um jogador tem um número de seqüência, de acordo com a ordem em que as mesmas são geradas na origem. Quando uma mensagem chega ao destino, este verifica se o número de seqüência da mesma é maior que o da última mensagem recebida anteriormente. Se for maior, é atualizado o estado com os dados contidos naquela mensagem. Caso contrário, ela é descartada.

5.4 Modelo

O subsistema de rede do InGE possui duas classes principais: servidor e cliente. A primeira possui métodos que permitem autenticação de um cliente que tente se conectar a ele, assim como métodos para receber e efetuar *broadcast* de mensagens para todos que estiverem a ele conectados ou forçar a desconexão de um cliente. A classe do cliente é responsável por enviar atualizações de estado de entidades controladas localmente e receber e processar atualizações de estado de jogadores remotos. Além desses métodos principais, o subsistema de rede do InGE também verifica *timeout* de clientes e do servidor, através de mensagens *dummy*.

Além das classes de cliente e servidor, o InGE possui uma classe de controle, que possui parâmetros de inicialização e finalização do subsistema de rede. Esta classe possui também parâmetros de *timeout*, periodicidade de envio de atualizações de estado e sincronização das *threads* de envio e recebimento. Por fim, há também uma classe que permite o envio de mensagens da aplicação através da rede.

5.5 Implementação

Cada entidade pode ser controlada por algum jogador ou não - como objetos que meramente façam parte do cenário, por exemplo. Para que os jogadores interajam, cada entidade controlada por cada jogador em um cliente é transformada em um pacote de rede. Os atributos daquela entidade são codificados em XML e então transmitidos em um pacote UDP para o servidor, que o verifica e encaminha para todos os outros clientes. Cada entidade tem um identificador único no cliente onde está o jogador que a controla, e cada cliente tem um identificador único na rede. Assim sendo, cria-se um identificador único na rede para cada entidade no jogo, que é derivado do identificador de cada cliente juntamente com o identificador local daquela entidade na máquina de onde veio.

Quando é recebida uma mensagem de atualização de estado de uma entidade, o receptor verifica, através do identificador único de rede da mesma, se aquela entidade já foi instanciada. Em caso positivo, são atualizados seus atributos. Caso contrário, aquela entidade é instanciada com aqueles valores.

6 Áudio

Músicas e efeitos sonoros conferem maior realismo aos jogos. Usar gravações de alta qualidade não é suficiente: é importante simular a maneira como uma pessoa percebe o som em um ambiente 3D. O InGE provê funções relativas à reprodução de áudio 3D, promovendo interface com a biblioteca OpenAL. Optou-se por utilizar esta biblioteca por ela ser multiplataforma e distribuída sob licença LGPL, diferentemente de DirectX Audio, que só está disponível para plataformas da Microsoft.

6.1 OpenAL: funções e integração

OpenAL possui funções que manipulam três elementos básicos: fontes sonoras, *buffers* de áudio e um ouvinte. *Buffers* são regiões na memória que armazenam áudio. O ouvinte e as fontes sonoras possuem uma posição e uma orientação; cada fonte sonora é associada a um *buffer* e pode reproduzir o áudio do *buffer*. A posição do ouvinte em relação às fontes determina como o som chega ao jogador: o volume do som é atenuado com a distância entre ouvinte e fonte; uma fonte posicionada à esquerda do ouvinte é reproduzida com maior intensidade na caixa de som da esquerda. A quantidade de fontes sonoras disponíveis é limitada de acordo com o dispositivo de áudio.

A integração entre a biblioteca e o motor é relativamente simples. O ouvinte deve compartilhar a posição e a orientação do jogador; em casos simples, cada entidade que produz som deve estar associada a uma fonte sonora de OpenAL. Essa abordagem falha em dois casos: quando o número de entidades excede a quantidade de fontes sonoras disponíveis e quando uma entidade precisa emitir dois ou mais sons simultaneamente - por exemplo, falar e disparar uma arma. A solução mais comum nesses casos é manter um *pool* com todas as fontes disponíveis. Quando uma entidade precisa emitir som, uma fonte do *pool* é associada à entidade e ao *buffer* correspondente, reproduz o *buffer* e então é devolvida ao *pool* para que possa ser reaproveitada. O *pool* de fontes ainda não foi implementado no InGE.

[8] Geometrias podem assumir diversas formas, como esfera ou caixa. Juntamente com corpos rígidos, representam completamente um objeto.

6.2 Formato de áudio

OpenAL não está vinculada a nenhum formato de áudio particular. Qualquer formato de áudio precisa ser convertido em áudio digital sem compressão antes de ser armazenado em um *buffer* de OpenAL. O motor InGE trabalha com o formato de compressão de áudio Ogg Vorbis, alternativa livre ao MP3, e utiliza funções da biblioteca libvorbis [Monty 2007] para descomprimir o arquivo.

6.3 Streaming

Como áudio com qualidade de CD ocupa cerca de 10MB de memória por minuto, é indesejável carregar uma música inteira em um *buffer*. É mais vantajoso usar a técnica de *streaming*: a música é carregada aos poucos, enquanto é tocada. São alocados dois *buffer*s de tamanho fixo. Inicialmente dois trechos consecutivos do início da música são carregados nos *buffer*s e o primeiro *buffer* começa a ser reproduzido. Então, cada vez que um dos *buffer*s acaba, o outro *buffer* é selecionado e um novo trecho da música é carregado no primeiro *buffer*.

7 Considerações finais e trabalhos futuros

O motor InGE ainda está em fase de desenvolvimento, embora seja possível implementar jogos utilizando suas funções. O tempo de desenvolvimento de um motor pode ser bastante longo; no caso do InGE, foram mais de dois anos de implementações genéricas e rascunhos de projeto até a criação do jogo *We Are The Champignons*. Isso se deve ao porte do *software* e à quantidade considerável de escolhas que precisam ser feitas durante seu desenvolvimento. Manter o InGE suficientemente genérico e bem integrado com suas dependências também não é trivial. O conhecimento adquirido na experiência de sua produção, entretanto, o justifica.

Após mais de três anos, é possível delinear alguns trabalhos futuros para o InGE, a serem desenvolvidos em médio prazo. No subsistema gráfico, será dada continuidade às implementações de interfaces gráficas utilizando a biblioteca CEGUI [CrazyEddie 2006]. No subsistema de física, estudam-se métodos de aperfeiçoar as implementações de dinâmica - especialmente no que diz respeito à detecção de colisão; para isto, investiga-se a possibilidade de se utilizar a biblioteca Bullet [Coumans 2007] ou alterar a integração entre o motor e a ODE. No subsistema de rede, está prevista a implementação de um protocolo de comunicação que possibilite a execução de comandos remotamente. Por fim, no subsistema de áudio, planeja-se a implementação de um *pool* de fontes sonoras.

Agradecimentos

A Christina von Flach e a Adolfo Duran, pelas sugestões para a melhoria do artigo.

Referências

BLIZZARD. 2004. World of warcraft. Nota: World of Warcraft é um jogo de computador. Disponível em http://www.worldofwarcraft.com.

CARNEGIE MELLON UNIVERSITY. 2007. Panda 3d. Nota: Panda 3D é um motor de jogos de código aberto. Disponível em http://www.panda3d.org/.

COUMANS, E., 2007. Bullet. Bullet é uma biblioteca para simulação de dinâmica de corpos rígidos e detecção de colisão. Disponível em http://www.continuousphysics.com/Bullet.

CRAZYEDDIE, 2006. Cegui. CEGUI é uma biblioteca multiplataforma para interfaces gráficas. Disponível em http://www.cegui.org.uk.

FRISBIE, P., 2004. Hawknl™ (hawk network library). Nota: HawkNL™ (Hawk Network Library) é uma biblioteca de rede. Disponível em http://www.hawksoft.com/hawknl.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns*. Addison-Wesley Professional, January.

HONEYCUTT, M., KASLIKOWSKI, J., AND RAMASWAMY, S., 2003. On the design and development of 3d multiplayer role playing games: Lessons learnt.

HU, S. Y., AND LIAO, G. M. 2004. Scalable peer-to-peer networked virtual environment. *ACM SIGCOMM 2004 Workshop on NetGames*.

INDIGENTE. 2006. We are the champignons. Nota: We Are The Champignons é um jogo de computador. Disponível em http://indigente.dcc.ufba.br/watc.

LANTINGA, S., WOOD, R., AND MINAMI, M., 2007. Sdl_net. Nota: SDL_net é uma biblioteca de rede. Disponível em http://www.libsdl.org/projects/SDL_net.

LANTINGA, S., 2007. Sdl (simple directmedia layer). Nota: SDL é um conjunto de bibliotecas. Disponível em http://www.libsdl.org.

LOKI SOFTWARE, 2007. Openal. OpenAL é uma API multi-plataforma para renderização de áudio. Disponível em http://www.openal.org.

MICROSOFT, 2007. Directx. DirectX é uma API e uma especificação de hardware de gráficos e áudio voltada para o desenvolvimento de jogos. Disponível em http://msdn.microsoft.com/directx/.

MONTY, 2007. libvorbis. Libvorbis é o codificador e decodificador padrão para o formato de áudio Ogg vorbis. Disponível em http://xiph.org/vorbis/.

SALZMAN, L., 2002. Enet. Nota: enet é uma biblioteca de rede. Disponível em http://enet.cubik.org.

SILICON GRAPHICS INC., 2006. Opengl. OpenGL é uma especificação definindo uma API multi-plataforma e multi-linguagem para desenvolvimento gráficos computacionais 2D e 3D. Disponível em http://www.opengl.org.

SMED, J., KAUKORANTA, T., AND HAKONEN, H. 2001. Aspects of networking in multiplayer computer games. In *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*, L. W. Sing, W. H. Man, and W. Wai, Eds., 74–81.

SMITH, R., 2007. Ode. ODE é uma biblioteca para simulação de dinâmica de corpos rígidos. Disponível em http://www.ode.org.

TORUS KNOT SOFTWARE. 2007. Ogre3d. Nota: Ogre3D é um motor de jogos de código aberto. Disponível em http://www.ogre3d.org/.

UBISOFT. 2007. Call of juarez. Nota: Call of Juarez é um jogo de computador. Disponível em http://www.callofjuarez.com.

VALVE SOFTWARE, 1999. Counter-strike. Nota: Counter-Strike é um jogo de computador. Disponível em http://www.counter-strike.net.

ZAGAL, J., MATEAS, M., FERNÁNDEZ-VARA, C., HOCHHALTER, B., AND LICHTI, N. 2005. Towards an ontological language for game analysis. In *DIGRA Conf.*