

FORGE V8: Um Framework para Jogos de Computador e Aplicações Multimídia

**Charles A. G. Madeira, Mauro F. Vieira, Talita R. Menezes, Danielle R. D. Silva,
Geber L. Ramalho, Carlos A. G. Ferraz**

Centro de Informática – Universidade Federal de Pernambuco (UFPE)

Caixa Postal 7851 – 50732-970 – Recife – PE – Brasil

{cagm,mfv,trm,drds,glr,cagf}@cin.ufpe.br

Abstract. *Until the end of this last decade, computer games were developed in an ad hoc way, disregarding reusability and focusing basically on performance. Recently, the games' industry realized the importance of using software engineering tools, and sought for new solutions by means of the techniques and methodologies of object-oriented development. However, a lot of work remains to be done. Unfortunately, there is not yet a clear and broad definition of a framework for games, nor a study of the applicability of design patterns in such framework. In this article, we propose an original and comprehensive framework for the development of games and multimedia applications, denominated FORGE V8, in order to make easier game development, as well as guarantee a better quality of these types of application.*

Resumo. *Até o final desta última década, jogos de computador eram desenvolvidos de maneira ad hoc, desconsiderando reusabilidade e objetivando apenas o bom desempenho. Recentemente, a indústria de jogos despertou para a utilização das ferramentas oferecidas pela engenharia de software, tentando buscar novas soluções através das técnicas e metodologias de desenvolvimento de projetos orientados a objetos. No entanto, muito trabalho resta a ser feito. Ainda não há uma definição clara e abrangente de um framework para jogos, nem tão pouco um estudo da aplicabilidade de padrões de projeto a tal framework. Neste artigo, propomos um framework original e abrangente para o desenvolvimento de jogos e aplicações multimídia, denominado FORGE V8, na intenção de facilitar a implementação e garantir uma melhor qualidade destes tipos de aplicação.*

1. Introdução

A indústria em geral, seja ela de software ou qualquer outro gênero, tem buscado mecanismos cada vez mais eficazes de divulgar seus produtos, treinar seus funcionários e executar suas tarefas procurando sempre alcançar uma maior eficiência e eficácia na realização de seus objetivos. Ultimamente, na indústria de software, um dos meios mais utilizados para estes fins tem sido o desenvolvimento de sistemas multimídias, principalmente para a Internet. Esses sistemas consistem de aplicações que reúnem diversas mídias como textos, sons, imagens e vídeos. O âmbito desses sistemas tem aumentado bastante indo desde a educação à divulgação e entretenimento. Esse crescimento se deve graças à exigência dos clientes que requisitam os sistemas buscando sempre o melhor, bem como os seus futuros usuários que desejam aplicações

eficientes, legíveis e de fácil aprendizado, além da concorrência de um mercado acirrado que segue na conquista de novos espaços. Tudo isso tem impulsionado o surgimento de novas ferramentas que possibilitam o desenvolvimento de aplicações de melhor qualidade e em menor espaço de tempo, no entanto exigindo bons níveis de qualificação de seus desenvolvedores.

Contudo nem sempre essas ferramentas fornecem flexibilidade suficiente para todas as aplicações dessa categoria, um bom exemplo são os jogos de computador - um dos representantes mais legítimos das aplicações multimídia. Esses sistemas requerem a manipulação de sons, vídeos, textos, gráficos e outras rotinas de processamento que são quase que impossíveis de serem manipuladas por uma ferramenta de forma a exigir, em grande parte, um trabalho árduo de codificação para prover todos os requisitos exigidos por um determinado jogo. Hoje em dia esse caminho é bastante custoso, uma vez que compromete bastante o tempo de desenvolvimento, dado a concorrência da indústria de jogos.

Para resolver esses problemas, especialistas em projeto de jogos têm tentado buscar soluções em diversas áreas de conhecimento, dentre elas, a engenharia de software, a qual por muito tempo teve seus princípios ignorados no desenvolvimento destas aplicações. Como consequência, *frameworks*, bibliotecas e novas tecnologias com características específicas para renderização¹ gráfica, sonorização, entre outros, estão sendo desenvolvidas e evoluem muito rapidamente [Rollings & Morris 2000]. Esses mecanismos trabalham com diferentes níveis de complexidade e granularidade de abstração fornecendo basicamente três direções hierárquicas de trabalho: a construção do jogo propriamente dita, a construção de *Application Program Interfaces* (APIs)/bibliotecas e a construção de *frameworks* de desenvolvimento.

Atualmente, a construção de *frameworks* é uma direção pouco explorada para o desenvolvimento de jogos, contudo esse caminho tende a dar uma maior reusabilidade e flexibilidade no desenvolvimento de aplicações desta natureza, além de fornecer uma maior facilidade e estabilidade aos desenvolvedores.

O presente artigo apresenta um *framework* original, denominado *FORGE V8*, para o desenvolvimento de aplicações multimídias, em especial, jogos de computador. Esse *framework* tem como objetivo reduzir o tempo de desenvolvimento, riscos e complexidades exigidos no projeto dessas aplicações. A intenção é fornecer aos desenvolvedores de jogos facilidade de uso, reusabilidade e modularidade - características essenciais para aplicações de boa qualidade.

Nesse cenário, foi constatado que vários problemas de projeto encontrados no desenvolvimento desse *framework* poderiam ser resolvidos com a aplicação de padrões de projeto já consolidados. Para isso, houve um grande esforço em pesquisas e exploração de estudos de casos, já que não é do nosso conhecimento publicações que resumam os principais padrões de projeto aplicáveis ao desenvolvimento de jogos. Como estudo de caso do *framework* proposto, foi desenvolvido um jogo simples a fim de avaliar o seu grau de facilidade de uso e complexidade de desenvolvimento.

¹ Processo do sistema de computação gráfica que é responsável pela projeção no monitor (2D) de um ambiente 3D.

A organização deste artigo é realizada da seguinte forma. Na seção 2 é descrito o estado da arte do desenvolvimento de aplicações multimídias especificamente jogos de computador exemplificando algumas APIs de desenvolvimento e as vantagens de construir um *framework*. A seção seguinte mostra a estruturação do *framework* proposto, o *FORGE V8*, e a descrição dos módulos que o compõem. Na seção 4 são relacionados os padrões de projetos utilizados no desenvolvimento do *FORGE V8*, bem como a descrição dos problemas solucionados através deles. Na seção 5 são apresentados os resultados obtidos com a utilização do *FORGE V8* em um estudo de caso correspondente ao desenvolvimento de um jogo demo. Finalmente, algumas conclusões sobre o presente trabalho e propostas futuras são apresentadas na seção 6.

2. Desenvolvimento de Aplicações Multimídia

Como já mencionada toda aplicação que requer recursos como som, vídeo, texto e gráficos pode ser classificada como uma aplicação multimídia. Essas aplicações são extremamente multidisciplinares abrangendo diversas áreas de conhecimento como artes gráficas, sonoplastia, cinema, psicologia, inteligência artificial, computação gráfica, engenharia de software, algoritmos e estruturas de dados, redes de computadores, entre outras e compreendem o entendimento e resolução de diversos problemas. Dentro dessa categoria existe um tipo de aplicação que reúne todos esses elementos: os jogos de computador.

2.1. Jogos de Computador e Aplicações Multimídia

Jogos de computador são aplicações que requerem realismo, altos requisitos gráficos e sonoros, algoritmos complexos, um vasto uso de memória e rede sob fortes restrições temporais. Geralmente essas aplicações demandam um grande tempo de desenvolvimento, aproximadamente de dois a três anos para serem concluídas por uma equipe, e a complexidade de desenvolvimento cresce em função da área de atuação a qual o programa pertence.

Outro ponto importante é que na criação de novos jogos surge sempre um novo conjunto de problemas a ser resolvido, principalmente, devido à introdução de novas tecnologias e propriedades, como por exemplo, novas bibliotecas de renderização gráfica ou novas formas de modelagem física dos elementos de um jogo. De uma maneira geral, esses sistemas devem fornecer aos usuários, no mínimo, o mesmo desempenho e qualidade oferecida pelos últimos aplicativos lançados no mercado, além de novas características. Naturalmente, isso tudo com a pretensão de atrair um maior público de usuários.

Por estas razões, é extremamente relevante possuir ferramentas adequadas a fim de agilizar o processo de desenvolvimento de aplicações multimídia principalmente dado ao avanço tecnológico e concorrência do mercado. Muitas dessas ferramentas oferecem flexibilidade e facilidades suficientes para construir aplicações de alta qualidade, no entanto para sistemas específicos como jogos se tornam insuficientes uma vez que eles necessitam de uma flexibilidade muito maior em sua produção. Nesse contexto se inserem as bibliotecas multimídia.

2.2. Bibliotecas Multimídia

As bibliotecas multimídia englobam rotinas e algoritmos complexos bastante utilizados na construção de jogos e aplicações multimídia em geral. As bibliotecas padrão, mais utilizadas atualmente, são a *Microsoft DirectX* [Kovach 2000] e a *OpenGL* [OpenGL 2001].

DirectX é um conjunto de APIs de baixo nível que apresenta funcionalidades com acesso direto a hardware, ideal para a criação de aplicações multimídia que necessitam de alto desempenho como os jogos de computador. Inclui suporte a gráficos bidimensionais (2D) e tridimensionais (3D), execução de músicas e efeitos sonoros, controle de dispositivos de entrada e suporte para aplicações em rede como jogos multiusuários. *DirectX* é composto pelos módulos *DirectX Graphics*, *DirectX Audio*, *DirectInput*, *DirectShow* e *DirectPlay*.

OpenGL é uma biblioteca multiplataforma para renderização de gráficos 3D que utiliza aceleração por hardware. Esta biblioteca é o padrão para computação gráfica nas áreas de simulação e pesquisas acadêmicas. Ela disponibiliza versões para *Windows*, *MacOS*, *Linux/Unix*, entre outras plataformas.

Embora essas ferramentas ofereçam implementações de rotinas de baixo nível para tratamentos de hardware e todo o pipeline gráfico (iluminação, transformação e rasterização de polígonos) necessários para o desenvolvimento de aplicações multimídia, elas não são de fácil utilização para desenvolvedores de alto nível. Portanto, se faz necessário a disponibilização de um melhor suporte ao uso destas ferramentas através da construção de gerenciadores das diversas mídias que adicionem todas as suas propriedades apresentando os princípios de orientação a objetos para garantir robustez, e gerar software modular e reutilizável, requisitos exigidos pelo grande dinamismo dessa área de software. Pensando nesse contexto, a introdução de um *framework* se faz ideal.

2.3. Framework para Jogos

Um *framework* representa o esqueleto de uma aplicação orientada a objetos – projeto de todo ou parte de um sistema que é representado por um conjunto de classes abstratas e uma maneira de interação com instâncias destas classes – que pode ser especializado para produzir aplicações personalizadas, a fim de reduzir o custo e aumentar a qualidade [Fayad & Schmidt 1997] [Johnson 1997].

Devido à alta complexidade que o desenvolvimento de aplicações multimídia demanda, o uso de *frameworks* se torna o ferramental ideal para estas aplicações. Nesta seção será explanada a necessidade dessa tecnologia para a referida indústria de software.

Frameworks, na área de desenvolvimento de jogos, são mais comumente conhecidos como motores. Geralmente, esses sistemas são responsáveis por oferecer as funcionalidades básicas de uma aplicação multimídia, fornecendo abstrações das características de mais baixo nível como as bibliotecas e APIs utilizadas no desenvolvimento de uma aplicação específica. A principal função desses motores é então concentrar os processamentos básicos necessários para o controle das mídias envolvidas nessas aplicações. Isto inclui serviços para o gerenciamento de renderização de gráficos, objetos do jogo, janelas da aplicação, de sons, suporte a rotinas

matemáticas, dentre outros. Logo, uma vez projetado, um motor pode ser reutilizado em uma enorme variedade de aplicações correlacionadas. Para isso, o motor deve apresentar a flexibilidade de ser personalizado para cada projeto específico a fim de atender as suas necessidades particulares sem grandes alterações. Desta forma, o desenvolvimento de aplicações torna-se mais fácil, deixando os desenvolvedores voltados para os objetivos característicos da própria aplicação – lógica, arte, inteligência, entre outros – e não mais para os processamentos de baixo nível.

No mercado existem alguns motores *freeware* genéricos em desenvolvimento, como por exemplo, o *Genesis3D* [Genesis3D 2001], o *Crystal Space* [CrystalSpace 2001], o *Golgotha* [Golgotha 2001] e alguns outros comerciais voltados exclusivamente a atender funcionalidades específicas – gráficos, personagens, sons ou modelagem física – de um jogo. No entanto, segundo nossas avaliações, sobretudo nos três primeiros motores, há uma série de problemas: arquitetura não modular, paradigma de orientação a objetos ignorado em muitos aspectos, pouca documentação, pouca reusabilidade e baixo desempenho – a maior parte desses motores foram projetados para trabalhar com plataformas e bibliotecas específicas. Essa avaliação serviu também para chegar a um fato muito importante: praticamente é inexistente o uso de padrões de projetos nesses motores mesmo sabendo que os mesmos apresentam situações próprias para seu uso.

Pensando em eliminar alguns desses problemas e também verificar a viabilidade do uso de padrões de projetos consolidados na resolução de problemas encontrados no desenvolvimento de um motor para aplicações multimídia, projetamos o *FORGE V8* descrito na próxima seção.

3. O Framework Proposto: FORGE V8

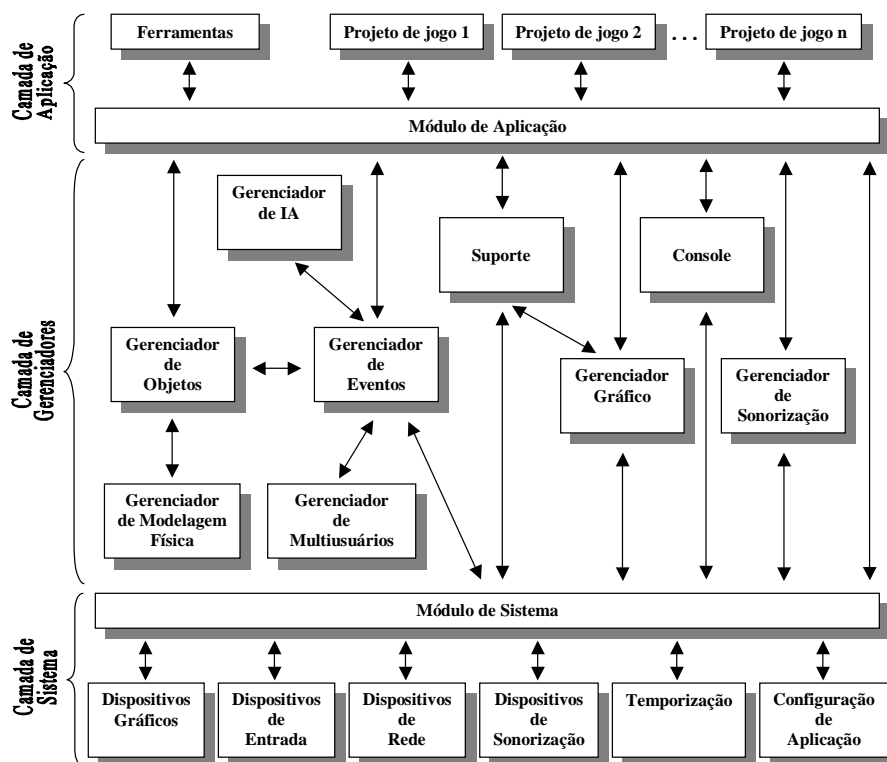


Figura 1. Estrutura do FORGE V8

FORGE V8 é o *framework* (motor) para jogos de computador que está sendo desenvolvido no Centro de Informática da UFPE. Esse motor possibilitará o desenvolvimento de jogos 2D, 2D ½ e 3D, diferentemente dos motores já existentes no mercado, que permitem apenas uma das três possibilidades.

O *framework* é composto de diversos módulos, os quais desempenham funções específicas. Nesta seção, são apresentados esses módulos básicos e suas respectivas funcionalidades e serviços (figura 1).

3.1. Camada de Sistema

A camada de sistema é a camada de mais baixo nível, responsável por realizar toda comunicação com o hardware. Essa camada é a única que necessita de extensões caso se deseje portar o motor para diferentes plataformas. Ela contém diversos subsistemas que provêm funcionalidades específicas e gerais. Os subsistemas de funcionalidade específica são: dispositivos gráficos, dispositivos de entrada, dispositivos de sonorização e dispositivos de rede. Os subsistemas de funcionalidades gerais são: temporização e configuração, responsáveis pelo acesso ao sistema operacional. A camada de sistema contém ainda um módulo geral de acesso, chamado módulo de sistema, responsável pela inicialização, configuração e finalização de todos os seus subsistemas. Esta camada gera uma biblioteca que tem o objetivo de executar todas as operações dependentes de plataforma para as aplicações. Desta maneira, as aplicações poderão facilmente portar para alguma das plataformas suportadas.

3.1.1. Subsistema de Dispositivos Gráficos

É o subsistema de mais alta complexidade da camada de sistema, responsável pelo acesso ao dispositivo de vídeo, leitura e gerenciamento dos modelos gráficos e imagens, e pelo processamento de renderização de cenários. Atualmente, no desenvolvimento de motores para jogos em geral, este subsistema utiliza-se de bibliotecas gráficas padrão (*OpenGL* e *DirectX Graphics*) e renderização por software proprietária. Para que mais de uma biblioteca possa ser alocada a esse subsistema, interfaces são projetadas de modo que diferentes implementações possam ser realizadas, disponibilizando aos usuários a possibilidade de escolha ou a realização de uma seleção automática. Dessa forma, provê uma maior compatibilidade e desempenho com o hardware apresentado. Esta é uma tarefa um tanto trabalhosa, especialmente devido às bibliotecas gráficas não possuírem um mesmo conjunto de características.

Este subsistema realiza a configuração e inicialização dos dispositivos gráficos e utiliza as bibliotecas gráficas produzindo um bom resultado devido à utilização das características inerentes aos hardwares gráficos, apresentando componentes otimizados para renderização por software em casos onde uma placa aceleradora não esteja presente.

3.1.2. Subsistema de Dispositivos de Entrada

Esse subsistema disponibiliza serviços de teclado, *mouse*, *gamepad*, *joystick* e efeitos *force feedback*² permitindo abstração do controle real usado pelo jogo. Ele deve realizar

² Tecnologia utilizada nos novos dispositivos de entrada para percepção física de ações sofridas por personagens no ambiente de um jogo

transparentemente checagem de todos os dispositivos instalados e identificar os eventos destes dispositivos, além de apresentar facilidade de configuração para os usuários, possibilitando vários dispositivos executarem uma mesma ação.

3.1.3. Subsistema de Dispositivos de Sonorização

Esse subsistema é responsável pela leitura e gerenciamento de sons e músicas através de *streams* ou *buffers* de memória, e a reprodução destes no amplificador. Um requisito bastante interessante adicionado a esse subsistema é o suporte a sons 3D³.

3.1.4. Subsistema de Dispositivos de Rede

É o subsistema da camada de sistema responsável por realizar comunicação (conexões, formação de sessões de grupos, e envio e recepção de mensagens) através da rede. Esse subsistema pode prover o suporte à comunicação por voz, e conexão via Internet através de modem ou placa de rede.

3.1.5. Subsistema de Temporização

O subsistema de temporização é responsável por tratar todas as informações ligadas ao tempo. É utilizado principalmente devido a atualmente os jogos serem na sua maioria em tempo real, o que caracteriza a necessidade de rotinas de gerenciamento de temporização. Este é o subsistema mais simples da camada de sistema, desde que o tempo seja a base para o que necessitar de movimentação.

3.1.6. Subsistema de Configuração de Aplicação

O subsistema de configuração é responsável pelos ajustes necessários à aplicação do jogo de acordo com o sistema operacional. Ele provê leitura de arquivos de configuração específicos para as aplicações, acesso a parâmetros de linha de comando, tratamento de mensagens do sistema operacional, e outros métodos quaisquer que sejam necessários acesso ao sistema operacional. Todos os outros subsistemas devem consultar este quando da inicialização e execução, proporcionando mais facilidade nas suas configurações (seleção de bibliotecas a serem utilizadas para renderização gráfica, renderização sonora, acesso aos dispositivos de entrada, e acesso ao dispositivo de rede). Desenvolver um motor com um bom suporte à configuração facilita os testes e permite que os usuários ajustem seus parâmetros de acordo com as suas necessidades.

3.1.7. Módulo de Sistema

O módulo de sistema é uma fachada que realiza a comunicação entre as camadas de níveis acima com os subsistemas da camada de sistemas. Todos os comandos são controlados por este módulo.

3.2. Camada de Gerenciadores

A camada de gerenciadores é uma camada de nível médio, responsável pelo controle da execução da aplicação, ou seja, trata os processos “disparados” pela lógica do jogo. Ela

³ Sons que utilizam um espaço tridimensional para serem posicionados e são percebidos com uma maior amplitude por objetos mais próximos.

contém diversos módulos que provêm funcionalidades específicas e gerais, do mesmo modo que a camada de sistema. Os módulos de funcionalidades específicas são: gerenciador gráfico, gerenciador sonoro, gerenciador de objetos, gerenciador de eventos, gerenciador de Inteligência Artificial (IA), gerenciador de modelagem física e gerenciador de multiusuários. Os módulos de funcionalidades gerais são: console e suporte, que são utilizados pelos outros diversos módulos. O ideal é que os módulos desta camada possam ser utilizados de forma independente por outras aplicações correlacionadas que necessitem de apenas um ou poucos gerenciadores.

3.2.1. Console

O console é o módulo do motor responsável por modificar as configurações do jogo e/ou do motor sem a necessidade de reiniciá-lo, como também é uma maneira eficiente para apresentar a saída de informações de *debug* no estágio de desenvolvimento. Frequentemente, se faz necessário examinar algumas variáveis e dados de saída, operação que um console realiza de maneira mais rápida e algumas vezes melhor que o “debugador”. No caso da ocorrência de erros em tempo de execução, não se deve haver a necessidade de sair da aplicação, mas sim tratar este erro cuidadosamente e imprimir uma mensagem explicativa. Um console caracteriza o desenvolvimento de um mecanismo de verificação de exceções bastante robusto. Se não for de interesse que ele seja apresentado ao usuário final, facilmente poderá ser escondido.

3.2.2. Suporte

O módulo de suporte é utilizado pelos muitos outros módulos do motor e tem a função de oferecer serviços comuns e utilitários. Esse módulo inclui todas as rotinas matemáticas necessárias (vetores, planos, matrizes, transformações, etc.) para os processamentos gráficos, gerenciadores de memória, leitores de arquivos, repositórios (STL⁴ ou proprietário), componentes de interface de usuário, ferramentas para compactação e descompactação de arquivos multimídia proprietários, entre outros.

3.2.3. Gerenciador Gráfico

Este módulo é responsável controlar e processar os componentes utilizados na construção de um cenário, enviando os resultados para a renderização pelo subsistema de dispositivos gráficos da camada de sistema.

Os componentes que compõem o gerenciador gráfico são: visibilidade, detecção de colisão e resposta, animação, câmera, geometria estática, geometria dinâmica, sistemas de partículas, *billboarding*, malhas, iluminação, neblina, espelhamento, texturização, etc [Eberly 2001]. Cada uma destas subseções necessita ter uma interface que facilmente permita modificações nas suas configurações, posição, orientação, ou alguma outra característica que esteja associada ao sistema.

3.2.4. Gerenciador de Sonorização

Este módulo é responsável por controlar o processo de execução de sons e músicas utilizadas na composição de cenários mais realistas, enviando os resultados para a

⁴ Standard Template Language

renderização pelo subsistema de dispositivos de sonorização da camada de sistema. Permite a mixagem de sons e músicas, e controla a execução de efeitos, volume e posicionamento (sons 3D).

3.2.5. Gerenciador de Objetos

O gerenciador de objetos é responsável pelo controle do ambiente (mundo) e entidades (personagens) de um jogo. Estas entidades devem ser armazenadas em estruturas de dados e controladas de acordo com o ciclo de vida específico de cada uma. Em geral, um jogo contém várias instâncias de diferentes entidades e todas as informações necessárias para que sejam gerenciadas. Esses dados envolvem, posicionamento, velocidade, dimensão, etc. Além disso, devem possuir métodos necessários para detecção de colisão com outros objetos.

Em geral, um editor de cenário se associa a este gerenciador para descrever o estado do mundo em cada nível do jogo. Tal associação resulta em delegações de tarefas para este gerenciador a fim de que ele inicie os objetos do mundo de acordo com a especificação do cenário.

Mundos podem ser formados de diversas formas: em jogos 3D, o mundo se refere a um ambiente 3D construído por polígonos; em jogos 2D, grids 2D são representados por vetores bidimensionais; e em jogos isométricos, os grids 2D são apresentados numa visão em perspectiva. O mundo é uma etapa de desenvolvimento bastante importante dos jogos de computador, pois dele pode se extrair a “mágica” do sucesso dos jogos.

3.2.6. Gerenciador de Eventos

O gerenciador de eventos é responsável por rotear todos os eventos relacionados à aplicação e ao motor: eventos do sistema operacional, eventos de janela, eventos dos dispositivos de entrada, eventos do motor de inferência⁵ (IA), eventos do processamento multiusuário e eventos do controle de objetos do ambiente do jogo. Tem como função simplificar o manuseamento de eventos evitando que vários objetos tratem eventos diferentes separadamente.

Para evitar o acoplamento dentro deste gerenciador, deve-se utilizar manipuladores de eventos (*handlers*). Alguns componentes do *framework*, tanto aqueles criados durante a execução do jogo quanto elementos que existem desde o início da aplicação, devem ser notificados da ocorrência de eventos que os interessem. Os componentes devem se inscrever para poder passar a observar os eventos de interesse.

3.2.7. Gerenciador de IA

Gerencia o comportamento de entidades autônomas. Para isso, pode utilizar diversas tecnologias (redes neurais, algoritmos genéticos, lógica *fuzzy*, motor de inferência, máquina de estados) que podem receber regras baseadas em *scripts* pré-estabelecidos pelas aplicações. Executa as ações de acordo com o estado atual da aplicação e as condições apresentadas nos *scripts*.

⁵ Aplicação responsável por executar ações referentes ao comportamento inteligente do jogo.

3.2.8. Gerenciador de Modelagem Física

O gerenciador de modelagem física é utilizado para modelagem de simulação dinâmica. O foco maior é dado para entidades de um jogo, e tem o objetivo de resolver equações de movimentação para estas entidades. Os modelos físicos são baseados em física Newtoniana, que trabalha razoavelmente bem em movimentação de objetos que possuem limites racionais de dimensão e massa.

3.2.9. Gerenciador de Multiusuários

Este módulo tem o objetivo de permitir interação entre os vários usuários de uma aplicação. Controla o estado dos usuários referentes às sessões estabelecidas pelo subsistema de dispositivos de rede da camada de sistema. O grau de importância da quantidade de jogadores pode variar de acordo com a categoria do jogo.

3.3. Camada de Aplicação

A camada de aplicação é a camada de mais alto nível do motor, responsável pela ligação entre este e os projetos de jogos e ferramentas necessárias. O ideal no desenvolvimento de um motor reusável é que as aplicações nunca tenham seus componentes inseridos a este motor. Portanto, esta camada contém o módulo de aplicação para atingir este desligamento. Esta camada é a responsável por oferecer todas as interfaces necessárias a construção de aplicações que utilizem o *FORGE V8* como base.

3.3.1. Módulo de Aplicação

O módulo de aplicação é uma fachada de todo o motor que expõe toda a sua funcionalidade em alto nível para o desenvolvimento de jogos e aplicações multimídia. Representa o caminho de comunicação e acesso de projetos com todo o *framework*. Um importante objetivo deste módulo é abstrair ao máximo o processamento do motor e conseqüentemente facilitar o desenvolvimento de aplicações. Além de que, se uma determinada aplicação deve ser desenvolvida baseada neste motor, então esta aplicação deve implementar as interfaces e padrões oferecidos pelo módulo de aplicação, caracterizando uma padronização à aplicação. Para cada gerenciador do motor que tem uma propriedade dinâmica, o motor/módulo de aplicação provê uma interface para modificá-la.

3.3.2. Ferramentas

Durante o desenvolvimento de um motor/jogo, diversos tipos de informações são necessárias. Por exemplo, não é fácil escrever arquivos que definam modelos de objetos 3D. Para isto, é necessário utilizar modeladores, e programas de texturização gráfica. Certamente, muitas ferramentas são estritamente necessárias e algumas delas podem/devem ser construídas num formato proprietário, como editores de níveis e editores de personagens, para realizar as tarefas específicas do projeto. Neste caso, precisa-se desenvolver conversores ou *plug-ins* do formato utilizado pelas ferramentas de terceiros para o formato utilizado pelas ferramentas proprietárias. A maior certeza é que provavelmente se faz necessário tão quanto ou mais código de ferramentas que do próprio código do jogo.

4. Padrões Aplicados ao *FORGE V8*

Nas próximas seções, veremos como padrões de projeto foram aplicados no desenvolvimento dos diversos módulos do *FORGE V8*.

4.1. Camada de Sistema – Módulo de Sistema

4.1.1. Problemas

1. Deve existir uma única instância da camada de sistema, sendo ela acessível por qualquer parte do *framework* a partir de um ponto conhecido.
2. Ela deve oferecer todas as funcionalidades do motor, servindo de fachada para os subsistemas da camada de sistema.

4.1.2. Soluções

Pode ser observado que o item (1) é um caso típico de aplicabilidade do padrão **Singleton**, o qual garante que uma classe possui uma única instância e provê um único ponto de acesso global a ela [Gamma 1995]. Dessa forma, foi utilizado tal padrão a fim de solucionar o respectivo problema.

O padrão mais adequado para resolver o problema (2) foi o **Facade** [Gamma 1995], pois ele permitiu a centralização das funcionalidades oferecidas pelo motor na camada de sistema, além de unir as funcionalidades dos subsistemas desta camada.

4.2. Camada de Sistema – Dispositivos Gráficos

4.2.1. Problemas

1. Deve existir uma única instância do dispositivo gráfico, sendo ela acessível por qualquer parte do *framework* a partir de um ponto conhecido.
2. Devem existir abstrações que permitam que objetos como texturas, câmeras e *sprites* se tornem independentes não só da biblioteca gráfica utilizada como OpenGL e DirectX Graphics, mas também da forma de visualização do jogo (2D ou 3D).

4.2.2. Soluções

Tal como no módulo de sistema, o padrão **Singleton** foi utilizado na solução do item(1).

A solução da situação (2) pode ser alcançada utilizando-se o Padrão **Factory Method** [Gamma 1995], que prevê a criação de uma estrutura que é responsável pela manufatura dos objetos gráficos do jogo. Todos estes objetos seriam criados baseado na biblioteca gráfica, abordagem utilizada sem o conhecimento de um cliente dessa *factory*.

4.3. Camada de Gerenciadores – Gerenciador Gráfico

4.3.1. Problemas

1. O gerenciador deve possuir um repositório (coleção) de instâncias de texturas, *sprites* no caso de jogos em 2D, e de malhas (representações de

objetos 3D) em jogos 3D. No entanto, deve-se evitar que mudança na implementação desse repositório cause impacto no gerenciador gráfico.

4.3.2. Soluções

O item acima pode ser solucionado pela utilização do padrão **Bridge** [Gamma 1995]. O repositório em questão poderia ser uma abstração de qualquer tipo de estrutura de dados para armazenamento, como por exemplo *hash tables*, listas, árvores, entre outros, e o gerenciador gráfico ficaria sempre independente da estrutura de dados utilizada.

4.4. Camada de Sistema – Dispositivos de Sonorização

4.4.1. Problemas

1. O dispositivo de sonorização funciona baseado num único identificador. Esse identificador deve ser acessível por qualquer parte do *framework* a partir de um ponto conhecido.
2. A camada de som deve ser uma abstração da placa de som fornecendo funcionalidades para processamento sonoro ao longo do jogo.
3. Deve-se ser possível reutilizar diferentes bibliotecas para processamento de som em jogos eletrônicos como OpenAL [OpenAL 2001] e DirectX Audio [Kovach 2000].

4.4.2. Soluções

Pode ser observado, que, assim como foi aplicado no módulo de dispositivo gráfico, o padrão **Singleton** pode ser aplicado no item (1) da lista acima.

A fim de solucionar os itens (2) e (3) foi projetada uma fachada de acordo com o padrão **Facade**. Essa fachada tem como finalidade fornecer as funcionalidades do dispositivo de som da camada de sistema além de abstrair como a mesma é implementada pelas diversas bibliotecas de processamento de som.

4.5. Camada de Sistema – Dispositivos de Entrada

4.5.1. Problemas

1. Deve existir uma única instância para cada dispositivo de entrada, sendo cada uma acessível por qualquer parte do *framework* a partir de um ponto conhecido.
2. Este subsistema deve ser uma abstração dos dispositivos de entrada disponíveis, como teclado e *joystick*, fornecendo funcionalidades para esse processamento ao longo do jogo.
3. Deve-se ser possível reutilizar diferentes bibliotecas para processamento de entrada para jogos eletrônicos como GLUT [OpenGL 2001] e DirectInput [Kovach 2000].

4.5.2. Soluções

Tal como no dispositivo de som, o padrão **Singleton** foi utilizado na solução do item(1).

Os problemas (2) e (3) são bastante similares aos problemas (2) e (3) do dispositivo de som. Dessa forma, nada mais natural que resolvê-los também através do padrão **Facade**.

4.6. Camada de Gerenciadores – Gerenciador de Eventos

4.6.1. Problemas

1. O gerenciador de eventos deve rotear todos os eventos possíveis necessários à aplicação: eventos do sistema operacional, eventos de janela, eventos dos dispositivos de entrada, eventos do motor de inferência, e eventos de processamento multiusuário.
2. Tem como função simplificar o manuseamento de eventos evitando que vários objetos tratem eventos diferentes separadamente. Tornando a aplicação pouco flexível.
3. Utiliza manipuladores de eventos (handlers) no intuito evitar o acoplamento dentro deste gerenciador.
4. Alguns componentes do *framework*, tanto aqueles criados durante a execução do jogo quanto elementos que existem desde o início da aplicação, devem ser notificados da ocorrência de eventos que os interessam.
5. Os componentes devem se inscrever para poder passar a observar os eventos de interesse.

4.6.2. Soluções

Os problemas descritos nos itens (1) a (3) podem ser resolvidos com a utilização do padrão **Chain of Responsibility** [Gamma 1995]. Este padrão trata da construção de uma estrutura hierárquica que, no nosso caso, repassa a responsabilidade do tratamento de um determinado evento para o nível mais baixo da hierarquia até chegar a ser tratado ou por um dispositivo dos citados acima.

Em relação aos dois últimos problemas, pode-se notar uma dependência entre alguns componentes do *framework* e este gerenciador. Quando os dados armazenados no gerenciador de eventos são modificados devido a um novo evento, todos os seus observadores devem ser notificados. O padrão que resolve esse problema naturalmente é o **Observer** [Gamma 1995]. No caso do *FORGE V8*, alguns subsistemas e gerenciadores correspondem aos **Subjects** [Gamma 1995] e os observadores deles aos **Observers**.

4.7. Camada de Aplicação – Módulo de Aplicação

4.7.1. Problemas

1. O gerenciador de aplicação deve possuir uma estrutura genérica do laço principal de um jogo (leitura e execução dos níveis do jogo, atualização dos eventos de entrada, atualização de estado e posicionamento dos objetos, e apresentação na tela do usuário).

4.7.2. Soluções

Para resolver esse problema, o padrão **Template Method** [Gamma 1995] foi utilizado. O laço principal de um jogo executa diversas funções, sendo cada uma delas representada por um **Template Method** que deve ser implementado de acordo com o jogo em questão.

5. Implementação e Resultados

No desenvolvimento de qualquer projeto, algumas decisões críticas devem ser tomadas de acordo com o seu grau de complexidade, com o ambiente e as ferramentas que serão utilizadas, e com o tempo disponibilizado para sua conclusão. A respeito do *FORGE V8*, estão discriminadas abaixo algumas destas decisões que influenciaram no seu desenvolvimento.

5.1. Configurações

Apenas duas linguagens de programação, Java e C++, foram seriamente consideradas para a implementação deste projeto. Isto se deve a estas linguagens apresentarem robustez, compatibilidade com diversas bibliotecas gráficas, e portabilidade para inúmeras plataformas. Mas dentre estas, escolhemos C++ devido principalmente ao seu desempenho, que atualmente é bem mais adequado que o de Java para aplicações como jogos [Lamothe 1999].

A respeito das bibliotecas multimídia, inicialmente pensou-se trabalhar em duas versões de código fonte, uma utilizando *DirectX* em ambiente *Windows*, e outra utilizando *OpenGL* em ambiente *Linux*. Mas, para a implementação da versão inicial do motor, decidimos por utilizar a primeira abordagem, devido a 90% dos usuários de computador utilizarem o *Windows* como plataforma básica [Hendricks 1998] e também, pelo motivo de *DirectX* implementar diversas outras funcionalidades com acesso direto a hardware (controle de dispositivos de entrada, sonorização, renderização de vídeo por *streaming* e controle de mensagens de rede) que *OpenGL* não implementa. Em versões futuras, poderão ser realizadas contribuições para com outros sistemas operacionais e bibliotecas multimídia, já que o *FORGE V8* está preparado para este tipo de incremento através da sua camada de sistema.

5.2. Implementação do Projeto de um Jogo

Devido à complexidade do projeto foi determinado que, num primeiro momento, seriam implementados apenas os requisitos básicos do *FORGE V8*. Isto inclui alguns subsistemas da camada de sistema, da camada de gerenciadores e camada de aplicação que fornecem serviços essenciais para execução de um jogo. Essa primeira fase de desenvolvimento durou cerca de 4 meses com o esforço de 4 engenheiros de software.

Uma vez definido o protótipo do *FORGE V8*, passou-se a desenvolver um jogo simples em versão demo com a finalidade de testar o referido *framework*. Esse jogo foi denominado *SuperTank*, pertence à categoria ação e utiliza gráficos isométricos. O jogador é representado por um tanque de guerra que tem o objetivo de destruir os inimigos que detém a princesa *Tiffanus* sequestrada.

O *SuperTank* havia sido desenvolvido algum tempo atrás sem utilizar qualquer *framework* como base, e teve a pretensão de estudar alguns problemas técnicos pontuais

do desenvolvimento de jogos. Uma vez o *FORGE V8* parcialmente desenvolvido, o mesmo jogo foi re-implementado, permitindo uma comparação entre os dois processos de desenvolvimento. Esse estudo de caso, pôde mostrar a facilidade de uso, por parte dos desenvolvedores, e o desempenho final da aplicação gerada, além de verificar a correteza das funcionalidades oferecidas. Alguns resultados são descritos na tabela 1.

Versão	Linhas de código	Período (homens/mês)	Recursos envolvidos (engenheiros de software)	Performance (frames/seg)
<i>SuperTank</i> 1.0	6.204	8	5	35
<i>SuperTank</i> 2.0	3.597	3	3	32

Tabela 1 – Comparação dos resultados obtidos

6. Conclusões e Trabalhos Futuros

Este trabalho apresenta o *FORGE V8*, um *framework* para suporte ao desenvolvimento de aplicações multimídia, em especial jogos eletrônicos, que fornece os serviços básicos requisitados por aplicações dessa natureza. O objetivo desse projeto é facilitar o trabalho desempenhado pelos desenvolvedores reduzindo tempo, complexidade e custo envolvidos nesse processo. Além disso, devido à alta modularidade apresentada no *FORGE V8*, diversos gerenciadores podem ser utilizados independentemente para compor aplicações multimídias específicas, sem a necessidade da incorporação de todo o motor no projeto envolvido.

Este trabalho serve também como base ou referência para aplicabilidade de padrões de projeto em motores de jogos, dado que não é do nosso conhecimento a existência de trabalhos relacionados.

Como continuação deste projeto, a etapa seguinte será a implementação de algumas funcionalidades que ainda não foram adicionadas ao *FORGE V8*. Uma vez completada essa etapa, passaremos para uma fase de otimização de código visando atingir o melhor desempenho possível. Enfim, pretendemos implementar uma outra versão do *FORGE V8* para outros sistemas operacionais como *Unix* e *Linux*, utilizando outras bibliotecas gráficas como *OpenGL*.

7. Referências Bibliográficas

- Beck, K., Johnson, R. (1994) "Patterns Generate Architectures", ECOOP.
- Booch, G., Rumbaugh, J., Jacobson, I. (1999) "The Unified Modeling Language User Guide", Addison-Wesley.
- D'Souza, D., Wills, A. (1999) "Objects, Components, and Frameworks with UML: The Catalysis Approach", Addison-Wesley.
- Fayad, M., Schmidt, D. (1997) "Object-Oriented Application Frameworks", Communications of the ACM, Vol.40, N°10, October.
- Gamma, Erich, et al. (1995) "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley.

- Johnson, Ralph E. (1997) "Frameworks = (Components + Patterns)", Communications of the ACM, Vol.40, N°10, October.
- Kovach, Peter J. (2000) "Inside Direct3D", Microsoft Press.
- Lajoie, R., Keller, R. (1994) "Design and Reuse in Object-Oriented Frameworks: Patterns, Contracts, and Motifs in Concert", Proceedings of the 62nd Congress of the Association Canadienne Française pour l'Avancement des Sciences (ACFAS), Montreal, Canada.
- Lamothe, A. (1999) "Tricks of the Windows Game Programming Gurus - Fundamentals of 2D and 3D Game Programming", Sams, United States.
- Hodorowicz, L. (2001) "Elements Of A Game Engine". Disponível em <http://www.flipcode.com/>.
- Meyer, B. (1997) "Object-Oriented Software Construction", ISE Inc., California, USA.
- Petzold, C. (1998) "Programming Windows, The Definitive Guide to the Win32 API", Microsoft Press.
- Perez, A. (2000) "Advanced 3-D Game Programming Using DirectX 7.0", Wordware Publishing Inc.
- Posnak, E., Lavender, R., Vin, H. (1997) "An Adaptive Framework for Developing Multimedia Software Components", Communications of the ACM, Vol.40, N°10, October.
- Rollings A., Morris D. (2000) "Game Architecture and Design", The Coriolis Group LLC.
- Salkin, S. (1998) "Design Patterns for Game Development", Game Developer Magazine, July.
- Schmidt, H. (1997) "Systematic Framework Design by Generalization", Communications of the ACM, Vol.40, N°10, October.
- Eberly, D. H. (2001) "3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics", Morgan Kaufmann Publishers.
- Genesis3D (2001) Disponível em <http://www.genesis3d.com>.
- Crystal Space (2001) Disponível em <http://crystal.linuxgames.com>.
- Golgotha (2001) Disponível em <http://www.planetquake.com/golgotha>.
- OpenGL (2001) Disponível em <http://www.opengl.org>.
- OpenAL (2001) Disponível em <http://www.openal.org>.
- Hendricks, S. (1998) "IDSA Report Video and Computer Games Industry's \$16 Billion Contribution to the U.S. Economy". IDSA On-line. Disponível em <http://www.idsa.com/releases/econ.html>.