Game Patterns: Patterns for Game Programming - Catálogo de Padrões para Desenvolvimento de Jogos

DAVID DE ALMEIDA FERREIRA, Universidade Estadual do Ceará
CIDCLEY TEIXEIRA DE SOUZA, Instituto Federal de Educação, Ciência e Tecnológica do Ceará

This paper presents an initial cataloging design patterns for electronic games, that aims at the dissemination, usage and encouraging the collection of new patterns enabling a greater professionalism to game development. Patterns can be used to build 2D/3D games, specialized engines, and frameworks, focusing mainly on issues of structure, organization, user interface, management and optimization. Patterns can be classified in two ways: first categorization based on the types of problems they solve, such as: Structuring, Functioning and Management. The second is based on the necessary experience for its perception, in other words, we have categories of patterns: Newbies Developers and Experienced Programmers.

Este trabalho apresenta uma catalogação inicial de padrões de projetos para jogos eletrônicos, visando assim a difusão, a utilização e o incentivo a catalogação de novos padrões permitindo um maior profissionalismo ao desenvolvimento de jogos. Os padrões podem ser empregados na construção de jogos 2D/3D, motores especialistas e frameworks, atuando principalmente nas questões de estruturação, organização, interface com usuário, gerenciamento e otimização. Os padrões podem ser classificados de duas formas: A primeira baseando-se em uma categorização dos tipos de problemas que resolvem, tais como: Estruturação, Funcionalidade e Gerenciamento. A segunda baseando-se na experiência necessária para a sua percepção, ou seja, assim temos as categorias de Padrões para: Programadores Novatos e Programadores Experientes.

Categories and Subject Descriptors: **D.2.10 [Software]**: Software Engineering—*Design*; **D.1.5 [Software]**: Programming Techniques—Object-Oriented Programming

General Terms: Software Engineering

Additional Key Words and Phrases: Design Patterns, Game Development, Game Patterns, Source, UML, Software Engineering, Desenvolvimento de Jogos, Exemplos, Código Fonte, Engenharia de Software

ACM Reference Format:

Ferreira, D. and Souza, C. 2010. Game Patterns: Patterns for Game Programming – Catálogo de Padrões para Desenvolvimento de Jogos. ACM Trans. Appl. Percept. 2, 3, Article 1 (September 2010), 49 pages.

1 INTRODUÇÃO

Com o amadurecimento da produção de software, percebemos cada vez mais a aplicação das diretrizes da Engenharia de Software, o que traz reconhecidamente melhores resultados do que os software montados de forma caótica.

No universo de desenvolvimento de jogos, podemos perceber, ainda um certo grau de complexidade para aplicação de práticas comuns no desenvolvimento de softwares convencionais [SANCHES,2009] [ARAUJO et al.,2006]. De fato, ao participarmos de listas de discussão, fóruns e comunidades [UNIDEV,2002] [PDJ,2003] [GDJBR,2009] [JOGOSPRO,2002], podemos observar a existência de um certo sentimento de rejeição às diretivas da engenharia de software. Em geral, tais diretivas são fortemente encontras no desenvolvimento de softwares tradicionais, porém quando observamos o desenvolvimento de jogos termos como “metodologia, processo, disciplinas, qualidade, conformidade e boas práticas”, não estão bem aplicadas ou aceitas, talvez por termos um alto grau de dinamismo entre os projetos.

Focando na etapa de programação, ou seja, nas questões arquiteturais e de implementação, podemos observar mais claramente uma ausência de informações ou referências sobre a utilização de “Design Patterns” em jogos. Indo além do código [DUKITAN,2007], observa-se a falta de literatura especializada, de congressos e discussões focando no assunto.

Author's address: David Ferreira; email: davidferreira.fz@gmail.com; site: www.davidferreira.com.br; Cidcley Teixeira; email: cidcley@gmail.com

Com isso em mente estamos propondo um trabalho, focando em Design Patterns para Jogos, ou seja, estamos documentando alguns padrões percebidos durante o desenvolvimento de jogos, com o objetivo de auxiliar na resolução de problemas encontrados durante o desenvolvimento, além de estimular a aplicação e catalogação de padrões de projeto focados especificamente em jogos.
Com isso devemos ter em mente que este trabalho por si só, é apenas a primeira parte de um longo processo de catalogação de padrões de projeto para jogos.

1.1 Objetivos

Para elaboração do catálogo desejado, estamos adotando a estratégia de divisão do trabalho em pequenas partes, as quais desejamos validá-las com: as comunidades acadêmicas, as comunidades de desenvolvedores de jogos e a comunidade de padrões.
Assim poderemos assegurar que os padrões propostos estejam bem escritos, que sejam de fato padrões utilizados pelos desenvolvedores de jogos e que estejam de acordo com as prerrogativas do movimento internacional de padrões.
Para cumprir tais objetivos este documento apresenta os cinco primeiro padrões de projeto para jogos. Onde com base na nossa estratégia de validação, já obtivemos aprovações da comunidade acadêmica e de diversos grupos de desenvolvedores de jogos, restando agora a adequação dos padrões propostos ao movimento internacional de padrões, por meio da Conferência Latino-Americana em Linguagens de Padrões para Programação [SUGAR,2010].

1.2 Audiência

Este trabalho é baseado na perspectiva inicial da resolução de problemas comuns encontrados durante a construção de jogos 2D, porém entendemos que tais padrões podem ser implementados em outros tipos de jogos tais como 2.5D [WIKIPEDIA,2010], ou seja, jogos 2D com percepção de profundidade e jogos 3D, podendo ser necessárias algumas pequenas variações, porém sem descaracterização do padrão em si.
Os padrões aqui documentados devem auxiliar principalmente programadores com pouca ou média experiência em desenvolvimento de jogos, pois observando os cinco padrões, poderíamos tentar categorizá-los de acordo com suas finalidades. Sendo assim, poderíamos agrupá-los em:

Padrões de Estruturação: os padrões responsáveis por delimitar um formato arquitetural para o programa, no caso o jogo, como por exemplo o Game Skeleton;

Padrões de Funcionalidades: os padrões responsáveis por auxiliar ou prover características desejáveis aos jogos, como por exemplo Font Mapping e o Resource Language;

Padrões de Gerenciamento: os padrões responsáveis por incluir características essenciais ou melhorar questões relacionadas a performance do jogos, como por exemplo o Resource Manager e o Animated Game Elements;

Poderíamos ainda tentar classificar os padrões baseando-se na experiência necessária para sua identificação e utilização. Sendo assim, teríamos:

Padrões para Programadores Novatos: são padrões básicos encontrados na maioria dos jogos, ou seja, são padrões essenciais para que o programador possa desenvolver seus primeiros jogos. Como exemplo teríamos:

Game Skeleton: é o ponto de partida inicial para que o programador saiba como organizar seu projeto de jogo.

Font Mapping: é um padrão para permitir a escrita de texto na tela, apesar de ser um padrão básico para os jogos, nem sempre sua principal motivação (a qual é a ausência de mecanismo para escrita na tela) é encontrada pelos programadores novatos, pois dependendo do conjunto de API e bibliotecas utilizadas o problema pode não estar em manifestação.

Animated Game Elements: é o padrão básico para dar movimento ao elementos do jogo, ou seja, permite que programadores novatos aprendam como estruturar e realizar o mecanismo de animação de seus personagens, objetos e cenários do jogo.

Padrões para Programadores Experientes: são padrões de nível avançado encontrados em projetos de jogos, que visam não apenas a resolução de problema imediatos, mas sim, a prevenção de problemas futuros, bem como a busca de uma solução com qualidade, performance, robustez e extensibilidade, ou seja, são padrões encontrados em jogos desenvolvidos por programadores com grande conhecimento técnico e teórico. Como exemplo teríamos:

Resource Manager: é o padrão que permite aos programadores o gerenciamento de recursos computacionais nos jogos, ou seja, permite controlar e gerenciar os elementos utilizados nos jogos, tais como imagens, sons, personagens e etc.

Language Manager: é o padrão que permite aos programadores uma abstração do texto utilizado na interface do jogo, evitando misturar texto informativo com o código fonte do jogo.

2 CATÁLOGO DE PADRÕES

Os padrões de projetos apresentados neste catálogo, possuem um grande inter-relacionamento, como podemos observar na Figura 1, a qual demonstra não só a composição e o inter-relacionamento dos padrões propostos, mas como seus relacionamentos com padrões de catálogos mundialmente conhecidos. Tais relacionamentos nos permitem entender melhor o funcionamento dos padrões, com algumas das características desejadas e esperadas, assim poderemos entender como os novos padrões foram embasados e o que podemos esperar de sua utilização.
Para um melhor entendimento explicaremos a seguir um pouco sobre a notação utilizada para demonstrar o relacionamento entre os padrões.

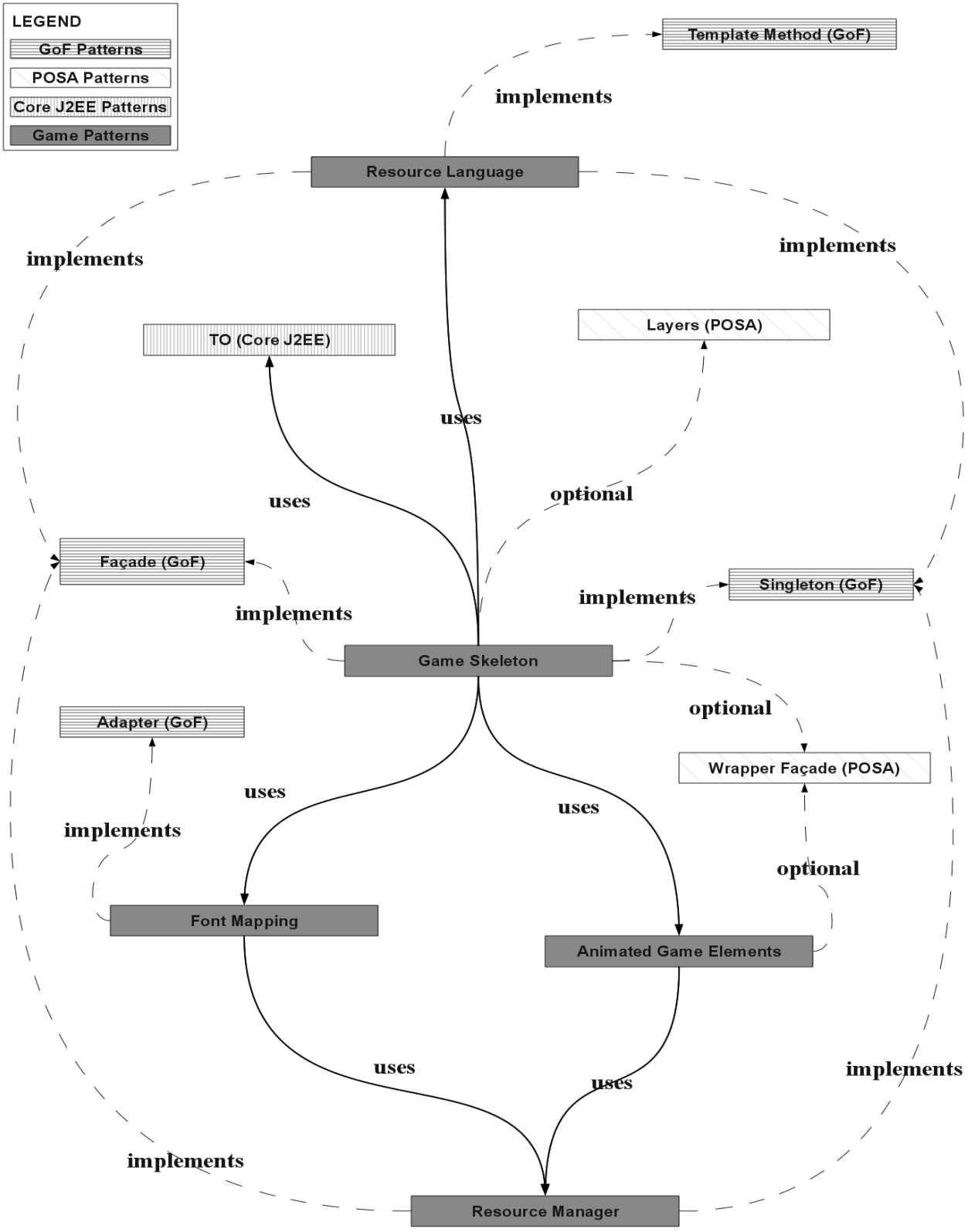

Figura. 1. Mapa de Relacionamento dos Padrões.

Iniciando nosso mapa de relacionamentos, podemos observar o destaque de alguns elementos, sendo eles:

GoF Patterns: Gang of Four - Design Patterns: Elements of Reusable Object-Oriented Software [GOF,1995];
POSA Patterns: POSA - Pattern-Oriented Software Architecture [POSA,1996];
Core J2EE Patterns: Core J2EE Patterns - Best Pratices and Design Strategies [J2EE,2002];
Game Patterns: Game Patterns – Patterns for Game Programming, ou seja, o qual estamos propondo;

Assim podemos rapidamente entender quais são os padrões necessários para aplicação dos padrões propostos, inclusive identificando a qual catálogo pertencem.
Em um segundo momento podemos observar que as setas possuem um único sentido, ou seja, elas iniciam-se do padrão origem indo em direção ao padrão que será referenciado ou utilizado. Além disso, observamos que existem apenas dois estilos de linhas, sendo elas:

Linha Contínua, a qual tem como finalidade identificar que um padrão utiliza-se de um outro padrão para seu pleno funcionamento, ou seja, durante a implementação de um padrão do nosso catálogo, mais precisamente, quando o padrão for concretizado em código, em algum momento existirá a utilização de uma instância do padrão relacionado;
Linha Pontilhada, tem como finalidade identificar que um padrão implementa um outro padrão para seu funcionamento, ou seja, durante a implementação de um padrão do nosso catálogo, o padrão apresentará características do padrão ao qual está relacionado, podemos imaginar um evento semelhante a uma herança, porém não necessariamente haverá utilização de todas as características do padrão herdado.

Em um terceiro momento podemos observar a existência de rótulos ou label próximo aos relacionamentos, os quais nos auxiliam a entender melhor o significado proposto, sendo assim temos:

uses, utilizado apenas no caso de linhas contínuas, onde reforça a ideia de que um padrão utiliza uma instância do padrão relacionado;
implements, utilizado apenas em linhas pontilhadas, indicando que um padrão implementa características do outro padrão;
optional, utilizado apenas em linhas pontilhadas, é semelhante ao implements, porém reforça que devido algumas características da aplicação do padrão origem, poderá não ocorrer o relacionamento com o padrão citado, como por exemplo, o caso do padrão Wrapper Façade [POSA,1996], onde existe a necessidade de utilizá-lo apenas quando tivermos elementos de código orientado a objeto combinado com código procedural, como no caso da utilização de uma API de baixo nível;

2.1 Game Skeleton

2.1.1 Intenção

Organizar e estruturar o funcionamento interno de um jogo eletrônico.

2.1.2 Outro Nome

Estrutura do Jogo

2.1.3 Motivação

Quando pretendemos criar um jogo eletrônico, devemos ter em mente que um jogo é um software altamente especializado, e por consequência, é bem mais complexo que a maioria dos softwares

tradicionais, pois possui um alto grau de interação com seu usuário, ou seja, o jogador, ao mesmo tempo que integra-se com diversos subsistemas e seus componentes, para assim poder executar de forma ordenada, controlada e prevista.

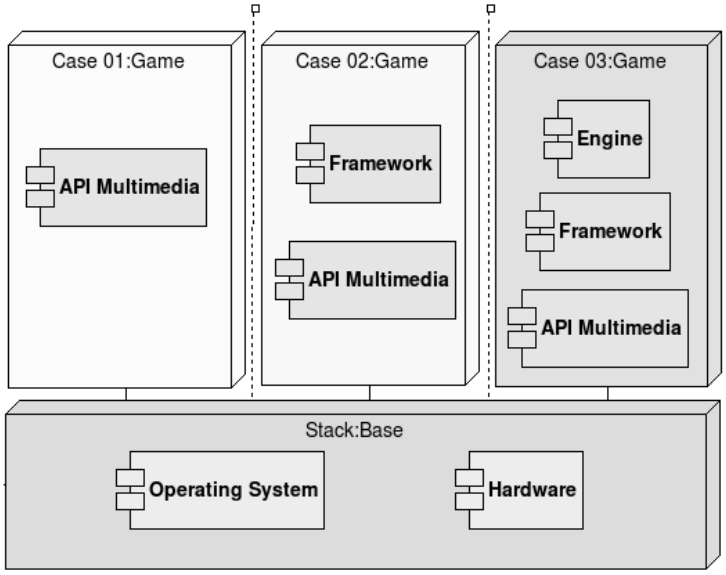

Figura 2. Visualização das construções dos jogos mais comuns.

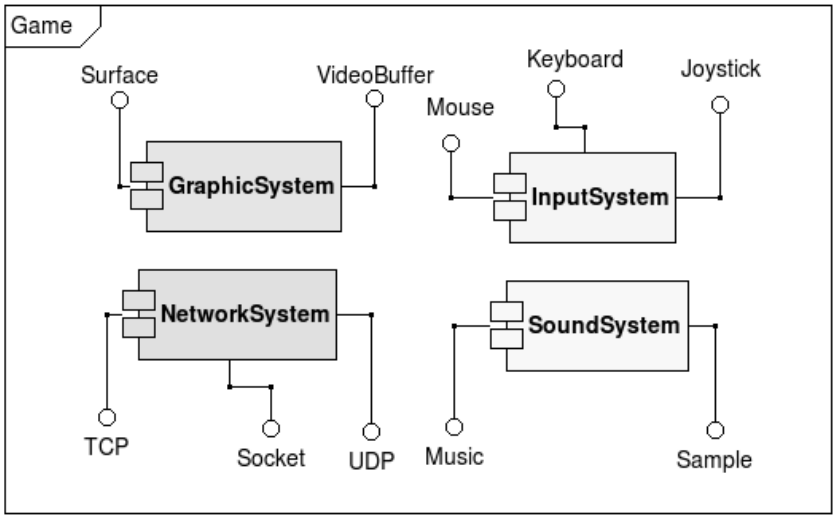

Figura 3. Composição dos subsistemas básicos de um jogo.

Ao se construir um jogo, é necessário termos em mente quais serão os “macro componentes” utilizados para montá-lo como pode ser visto na Figura 2, onde temos questões de integração que vão:

de baixo nível com integrações com o Sistema Operacional e /ou Hardware;
de médio nível com integrações como API e Frameworks;
de alto nível com um foco bem especializado na Engine;

Porém a certeza é que independente dos componentes selecionados construídos ou por construir, um jogo vai requerer alguns subsistemas básicos, como podemos ver na Figura 3, e justamente uma das grandes dúvidas é como orquestrar a comunicação destes subsistemas de modo que o jogo aconteça de forma fluída aos olhos e ações do jogador.

2.1.4 Aplicabilidade

Use o padrão Game Skeleton quando:

for necessário desenvolver componentes (api, frameworks, engine) para jogos;

for desejável organizar como seus subsistemas interagem;

for desenvolver um jogo eletrônico;

2.1.5 Estrutura

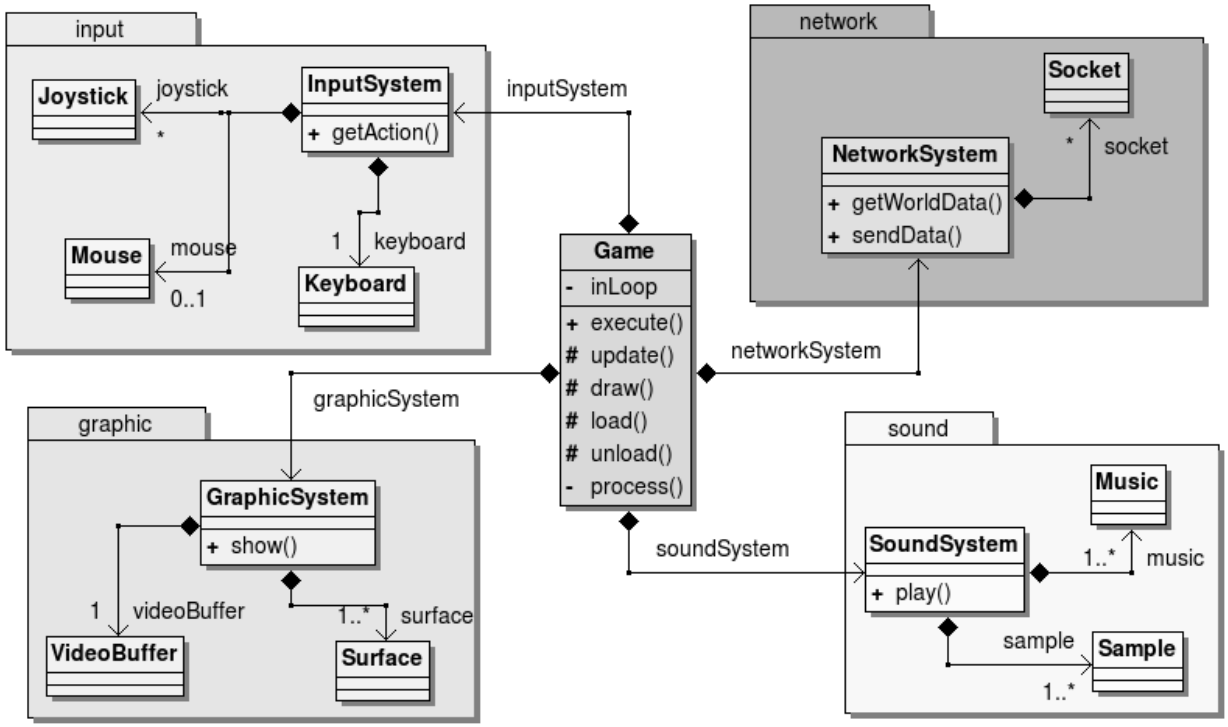

Figura 4. Estrutura do padrão Game Skeleton e suas colaborações.

2.1.6 Participantes

Pacote Input: classes pertencentes a este pacote são responsáveis por prover as interfaces e controladores para os dispositivos de entrada do jogo como teclado, mouse e joystick;

Pacote Graphic: classes pertencentes a este pacote são responsáveis por prover as interfaces e controladores para o dispositivo gráfico;

Pacote Sound: classes pertencentes a este pacote são responsáveis por prover as interfaces e controladores para os dispositivos de som do jogo como músicas e efeitos sonoros;

Pacote Network: classes pertencentes a este pacote são responsáveis por prover as interfaces e controladores para os dispositivos de comunicação com outros jogadores ou com o servidor;

Game

é a classe responsável por coordenar a integração entre os diversos subsistemas e o processo de execução do jogo;

2.1.7 Colaborações

A classe Game ao ser carregada, inicializa uma instância de cada controlador de subsistema;

Game requisita ao controlador de InputSystem a verificação do estado do(s) dispositivo(s) utilizado no jogo;

Game requisita ao controlador de NetworkSystem, os dados de outros jogadores no servidor, assim como também envia para os mesmos as informações sobre o jogador local;

Game requisita ao controlador de SoundSystem, que toque a música de fundo e os efeitos sonoros;

Game requisita ao controlador de GraphicSystem, que exiba as imagens armazenadas no buffer secundário de vídeo para o monitor;

2.1.8 Consequências

O Game Skeleton tem como consequências:

Arquitetura em Camadas, quando é necessário construir os subsistemas, é feito um encapsulamento das API de baixo e médio nível, o que leva comumente a construção de aplicações em camadas;

Fácil Manutenção, por haver uma padronização na composição do jogo, a manutenção é simplificada, devido a separação das etapas de Carregamento (load), Atualização (update), Desenho (draw) e Descarregamento (unload) dos componentes do jogo, ou seja, o ludu (load-update-draw-unload) é bem definido na implementação;

Facilidade de Uso, por haver uma organização em subsistemas, os quais são comumente organizados e acessados por classes de controle, a utilização de fachadas abstrai como os elementos internos do componente colaboram e interagem entre si;

Complexidade de Implementação, por haver uma grande separação em camadas e subsistemas e um inúmeras integrações com alto grau de iteração, temos em contra partida um aumento na complexidade da solução, indo desde o uso de herança a composição e agregação de classes para haver o perfeito funcionamento;

Perda de Performance, por ser uma solução baseada em uma grande orquestração de classes, objetos e subsistemas insere-se neste contexto uma perda de performance se compararmos aos modelos monolíticos e caóticos, ou seja, jogos semelhantes criados sem um modelo de estruturação baseado em responsabilidades, classes e camadas, os quais podem apresentar uma grande performance, porém inserem grandes problemas relacionados a manutenção, uso e expansão;

2.1.9 Implementação

Para implementar o padrão Game Skeleton, devemos ter em mente:

1. Definir os elementos que irão compor as interfaces de integração e como elas irão interagir com as classes mais internas do subsistema;
2. Ao criar os componentes ou o jogo diretamente sobre uma API, precisaremos separar bem as etapas de “ludu” do jogo, ou seja, load-update-draw-unload;
3. No caso da utilização de engines ou frameworks, precisamos saber quais são os métodos que devemos implementar que correspondem as mecanismos desejados;

Considere os seguintes aspectos de implementação quando utilizar o padrão Game Skeleton:

1. Alguns jogos podem não requerer um subsistema de network, ou o mesmo pode ser utilizado de forma opcional em alguns outros estados do jogo, como uma galeria de recordes;
2. Os componentes podem ser montados em uma arquitetura em camadas, mas em algumas ocasiões, é necessário permitir acesso direto às camadas inferiores, principalmente quando não é possível determinar todas as funcionalidades possíveis para a camada, ou quando pode ser exigida uma extrema otimização dos recursos das camadas inferiores;
3. O controlador de subsistema pode ser implementado utilizando-se de padrões como Singleton [GOF,1995] e Façade [GOF,1995];
4. O padrão define uma estrutura minimalista de como deve ser coordenada as etapas de execução do jogo, porém deve-se observar que é necessário manter estas características nos demais componentes embutidos como por exemplo em personagens ou itens, onde os mesmos devem possuir etapas de update-draw bem definidas;

2.1.10 Exemplo de Código

Para exemplificação da aplicação do padrão Game Skeleton, considere a Figura 5.

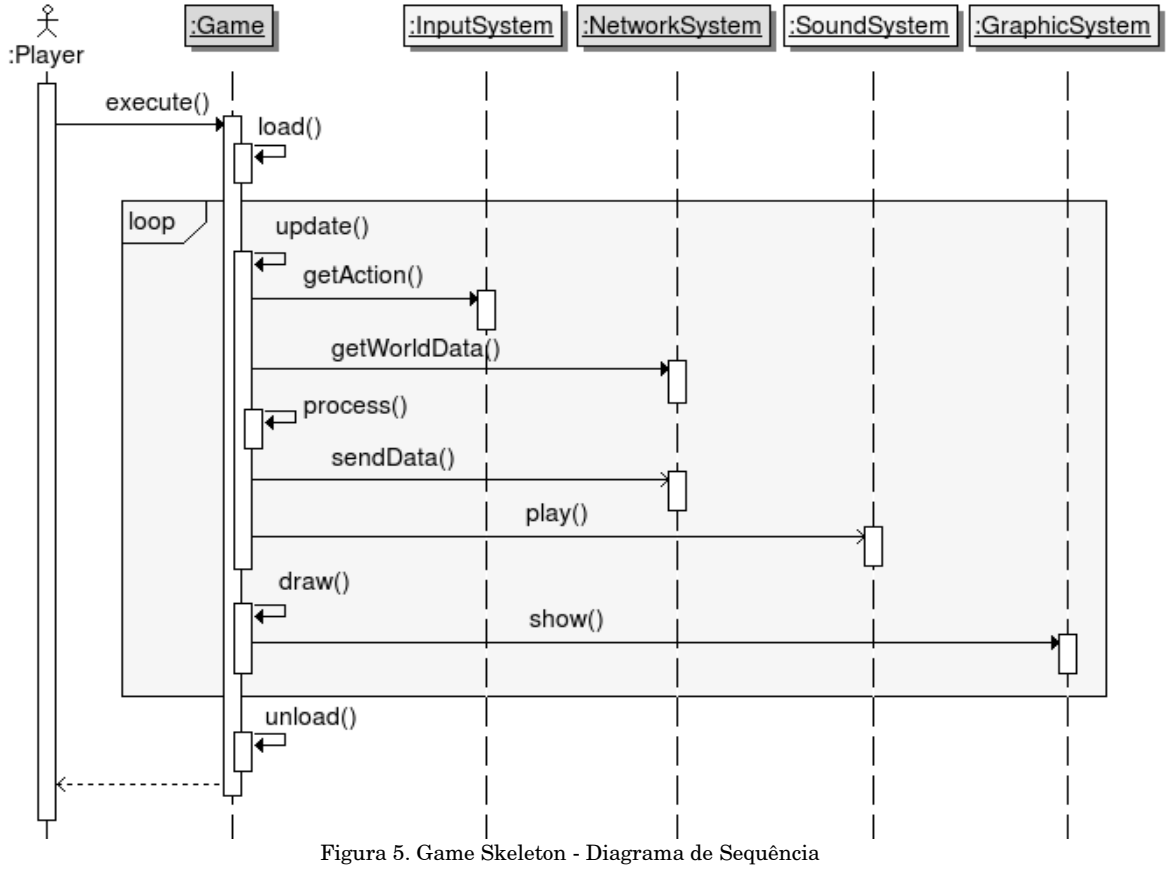

Figura 5. Game Skeleton - Diagrama de Sequência

Onde:

Player é a representação de um jogador acionando a execução do jogo;

Game é a representação do executável do jogo, orquestrando as etapas de processamento e a orquestração da comunicação entre os subsistemas;

InputSystem é o controlador dos dispositivos de entrada como o teclado, o mouse e o joystick;

NetworkSystem é o controlador dos dispositivos de comunicação em rede, responsável por obter informações do mundo do jogo, como por exemplo a localização de itens e outros jogadores, bem como é responsável por enviar assincronamente ao servidor os dados sobre o jogador local;

SoundSystem é o controlador do dispositivo sonoro responsável por processar a música de fundo e os efeitos sonoros de forma que o jogo possa prosseguir em paralelo, ou seja, assincronamente sem esperar seu término;

GraphicSystem é o controlador do dispositivo gráfico responsável por fornecer memória de armazenamento de imagens (surface) e o controle sobre as técnicas de bufferização do vídeo (double buffer, page flip);

Observando o código de exemplo da classe Game (Código 1), destacamos alguns detalhes:

1. Os métodos load e unload são executados respectivamente no início e no término do jogo, permitindo alocar e desalocar todos os recursos e objetos utilizados;

2. O método update é responsável pela canalização do processamento principal do jogo, ou seja, é ele que irá processar ou delegar as tarefas para:
    1. Obter as ações do Jogador (getAction);
    2. Obter os dados do jogo no servidor (getWorldData);
    3. Analisar e verificar as condições sobre as ações do jogador e seus efeitos no jogo (process);
    4. Enviar os dados sobre o jogo local para o servidor (sendData);
    5. Solicitar para o subsistema de som (SoundSystem) que toque a música de fundo e os efeitos sonoros;
3. O método process é utilizado para fazer as verificações das ações do jogador, com os dados do servidor, permitindo assim que o jogador possa participar de batalhas em grupos e enfrentar monstros ou apenas outros jogadores, ou seja, é o responsável real pela lógica do jogo;
4. O método draw é utilizado para dar a resposta visual ao jogador, ou seja, exibindo as imagens no monitor do jogador;

Código 1. Código da classe Game.

```
class Game {
  public:
    Game();
    void execute();

  protected:
    void update();
    void draw();
    void load();
    void unload();
    InputSystem inputSystem;
    NetworkSystem networkSystem;
    GraphicSystem graphicSystem;
    SoundSystem soundSystem;

  private:
    bool inLoop;
    void process();
};

Game::Game() {
    inLoop = true;
}

void Game::load() {
    //inicialização dos subsistemas e recursos utilizados
}

void Game::unload() {
    //desalocação de objetos e memória utilizados
}

void Game::process() {
    //processamento da lógica principal do jogo
}

void Game::execute() {
```

```
    load();
    while(inLoop){
       update();
       draw();
    }
    unload();
}

void Game::update() {
    if (inputSystem.getAction(VKCode::ESC)){
       inLoop = false;
    }

    if (inputSystem.getAction(VKCode::UP)){
       //mover para cima
    } else if (inputSystem.getAction(VKCode::DOWN)){
       //mover para baixo
    }

    if (inputSystem.getAction(VKCode::LEFT)){
       //mover para esquerda
    } else if (inputSystem.getAction(VKCode::RIGHT)){
       //mover para direita
    }

    TOWorld * to = networkSystem.getWorldData();
    process();
    networkSystem.sendData(to);
    soundSystem.play();
}

void Game::draw() {
    graphicSystem.show();
}
```

2.1.11 Usos Conhecidos

O framework Microsoft XNA Game Studio, em sua estrutura e em diversos tutoriais publicados no site MSDN [XNADC,2009], aplica o padrão, onde o método é responsável por delegar para diversos métodos privados as chamadas equivalentes a interação do usuário, obtenção e envio de dados da rede, além do processamento da lógica do jogo.

O livro OpenGL Game Programming [HAWKINGS e ASTLE,2001] de Kevin Hawkings e Dave Astle (CEO e COO da GameDev.net [GAMEDEV,1999]), publicado em 2001 pela editora Prima Tech's, sugere uma arquitetura básica, baseada em subsistemas, ilustrando seus relacionamentos, e uma proposta do ciclo de vida do jogo, onde encontramos as principais tarefas do "ludu", e algumas especificidades para a game engine proposta pelo livro, como por exemplo o uso de um subsistema para ambientes tridimensionais, física e inteligência artificial.

O GBFramework [GBF,2005], em seu pacote de GAT (GBF Application Template), aplica o padrão para que os jogos possam de fato apresentarem uma estrutura simples, direta e organizada. Com o padrão aplicado os jogos conseguem facilmente implementar diversos mecanismos, tais como exibição em modo pause, onde neste momento apenas a parte de draw é acionada, permitindo assim que a tela seja desenhada com as ultimas ações do jogador, ou ainda pode-se cortar o controle do jogador, para

que ações automáticas sejam executadas, as quais são muito comumente utilizadas nas transições de fases ou na finalização do jogo, como por exemplo a comemoração do jogador ao derrotar um chefe de fase.

2.1.12 Padrões Relacionados

Façade [GOF,1995], é empregado nos controladores de subsistemas, pois permite que a classe Game possa interagir com eles, sem de fato conhecer como internamente as classes cooperam;

Singleton [GOF,1995], é empregado nos controladores de subsistema, para garantir que existirá apenas um controlador por subsistema, permitindo assim facilmente o acesso aos recursos necessários por outras partes do jogo;

Wrapper Façade [POSA,1996], é empregado para isolar o código orientado a objetos de uma API procedural, isso corre normalmente quando trabalhamos com API de baixo e médio nível como as API de Multimídia ou do Sistema Operacional;

Layers [POSA,1996], ao criarmos jogos com os diversos "macro componentes" (ver Figura 2), é comum isolarmos suas dependências em uma arquitetura de camadas, permitindo que as camadas só interajam com a primeira camada abaixo mais próxima;

Transfer Object [J2EE,2002], é utilizado para transferir os dados do jogo para o servidor e vice versa, permitindo assim enviar apenas os dados importantes, como por exemplo processar a obtenção de itens, golpes de ataque e defesa;

2.2 Resource Manager

2.2.1 Intenção

Permitir que uma aplicação gerencie os recursos multimídia necessários para seu funcionamento.

2.2.2 Outro Nome

Gerenciador de Recursos

2.2.3 Motivação

Muitas vezes, estamos desenvolvendo uma aplicação ou um jogo eletrônico, que fará uso de recursos externos ao código fonte, como por exemplo: arquivos de imagens, arquivos de vídeos e arquivos de som.

Sendo que diversas vezes uma mesma imagem ou som é utilizado por diversos elementos do jogo, como por exemplo: A animação de uma explosão, que contém 6 quadros (imagens), se em um certo momento do jogo, encontramos 7 aviões na tela, teríamos então que ter em memória o equivalente a 42 imagens, apenas para representar a possível explosão dos aviões.

Tendo isso como base, observamos que é necessário um gerenciamento de recursos que além de carregar o que for necessário, quando necessário, possa prover o compartilhamento para que diversos elementos do jogo façam uso.

Figura 6. AeroTarget, imagem do jogo.

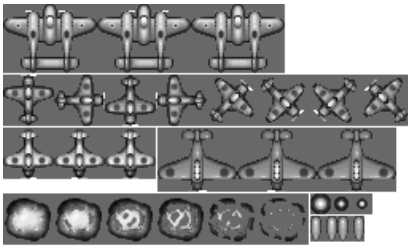
Figura 7. Imagens utilizadas no jogo AeroTarget.

2.2.4 Aplicabilidade

Use o padrão Resource Manager quando:

é necessário carregar recursos externos ao seu código;
é desejável carregar os recursos sob demanda;
é necessário o compartilhamento de recursos entre diversos elementos da aplicação;

2.2.5 Estrutura

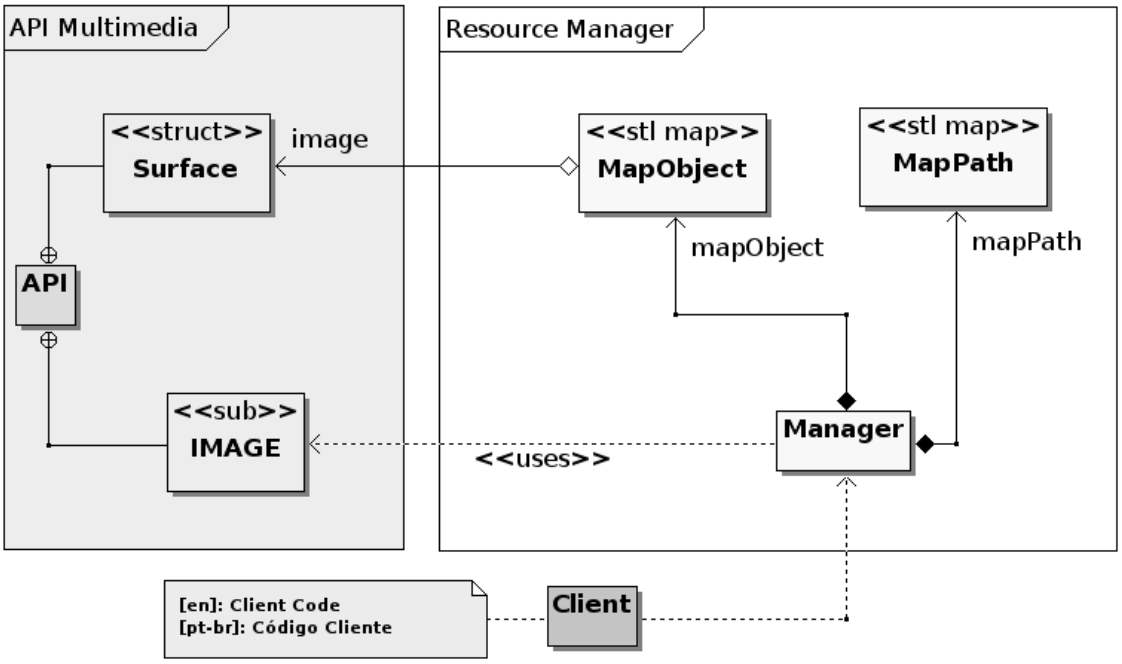

Figura 8. Estrutura do padrão Resource Manager e suas colaborações.

2.2.6 Participantes

Surface <<API Multimedia>>: estrutura de dados para representar o arquivo de imagem armazenado em memória, é um componente ou estrutura de dados provida pela biblioteca multimídia;

IMAGE <<API Multimedia>>: representação do manipulador de vídeo, provê operações ou funções de acesso a baixo nível disponibilizado pela biblioteca multimídia (I/O);

Manager: representação e implementação do padrão ResourceManager, gerencia o carregamento dos recursos necessários para o jogo;

MapPath: contêiner STL (Standard Template Language) para criação de um atributo capaz de armazenar um par "chave" e "objeto", responsável por mapear uma chave a um arquivo em disco;

MapObject: contêiner STL (Standard Template Language) para criação de um atributo capaz de armazenar um par "chave" e "objeto", responsável por mapear uma chave a uma Surface;

Client: colabora utilizando o Manager;

2.2.7 Colaborações

O Client inicia adicionando um recurso ao Manager, que por sua vez armazena as informações referentes a chave (identificador do recurso) e o seu caminho no disco;

O Client solicita a obtenção de um recurso ao Manager, que por sua vez verifica se o mesmo já encontra-se em memória, caso contrário verifica sua tabela de armazenamento e carrega-o;

O Client solicita que um recurso seja descarregado do Manager, que por sua vez, descarrega-o liberando memória até que o mesmo seja novamente requerido;

O Client solicita que um recurso seja removido do Manager, o qual por sua vez, descarrega-o completamente da memória, inclusive remove toda informação referente ao recurso;

2.2.8 Consequências

O Resource Manager tem como consequências:

Otimização de Memória, por permitir o compartilhamento de recursos dentro da aplicação é necessário uma quantidade menor de memória;

Tempo de Carga Menor, por permitir que os recursos sejam carregados apenas quando necessário, isso permite que aplicações com dezenas de recursos não precisem carregá-los ao mesmo tempo, possibilitando assim o carregamento inicial dos recursos mais utilizados;

Abstração em Alto Nível, evitando que o Client tenha o conhecimento sobre a API Multimedia, visto que a classe Manager funciona como um Façade [GOF,1995], simplificando e isolando as operações de carga de recursos do disco;

Aumento de Complexidade, por ser uma solução baseada na modelagem de classes e suas responsabilidades, exige-se uma maior capacidade mental de visualização e entendimento dos programadores, se compararmos a uma implementação baseada em programação estruturada e monolítica, na qual não existe uma separação clara de responsabilidades;

Perda de Performance, por ser uma solução baseada em uma orquestração de objetos insere-se neste contexto uma perda de performance se compararmos a uma implementação monolítica em programação estrutura, ou seja, que não apresente um modelo de classes baseado em responsabilidades, ou seja, que não esteja baseada nas boas práticas de análise e projeto de sistemas;

2.2.9 Implementação

Para implementar o padrão Resource Manager, devemos ter em mente:

1. Precisaremos de um arquivo de recursos a ser carregado;
2. Precisaremos armazenar os pares chave-caminhoArquivo e chave-objetoEmMemoria, para o carregamento dos recursos;

Considere os seguintes aspectos de implementação quando utilizar o padrão Resource Manager:

1. Manager não necessariamente precisa conter um tipo de dado da API Multimedia, ele pode conter qualquer estrutura ou objeto, que possa encapsular o recurso em memória;
2. Manager deve armazenar os pares “chave-caminhoArquivo” e “chave-objetoEmMemoria” em algum tipo de contêiner, para que possa de fato utilizar-se de todos os benefícios providos pelo padrão;
3. Deve-se criar um Manager, para cada tipo de recurso que se deseja manipular, exemplo: MusicManager, ImageManager, FontManager;
4. Pode-se implementar o Manager como um Singleton [GOF,1995], garantindo assim que existirá apenas um gerenciador para cada tipo de recurso;
5. O Manager é o único responsável por desalocar os recursos por ele criados;
6. O Manager ao ser removido deve liberar todos os recursos por ele gerenciados;

2.2.10 Exemplo de Código

Para exemplificação da aplicação do padrão Resource Manager, considere a Figura 9.

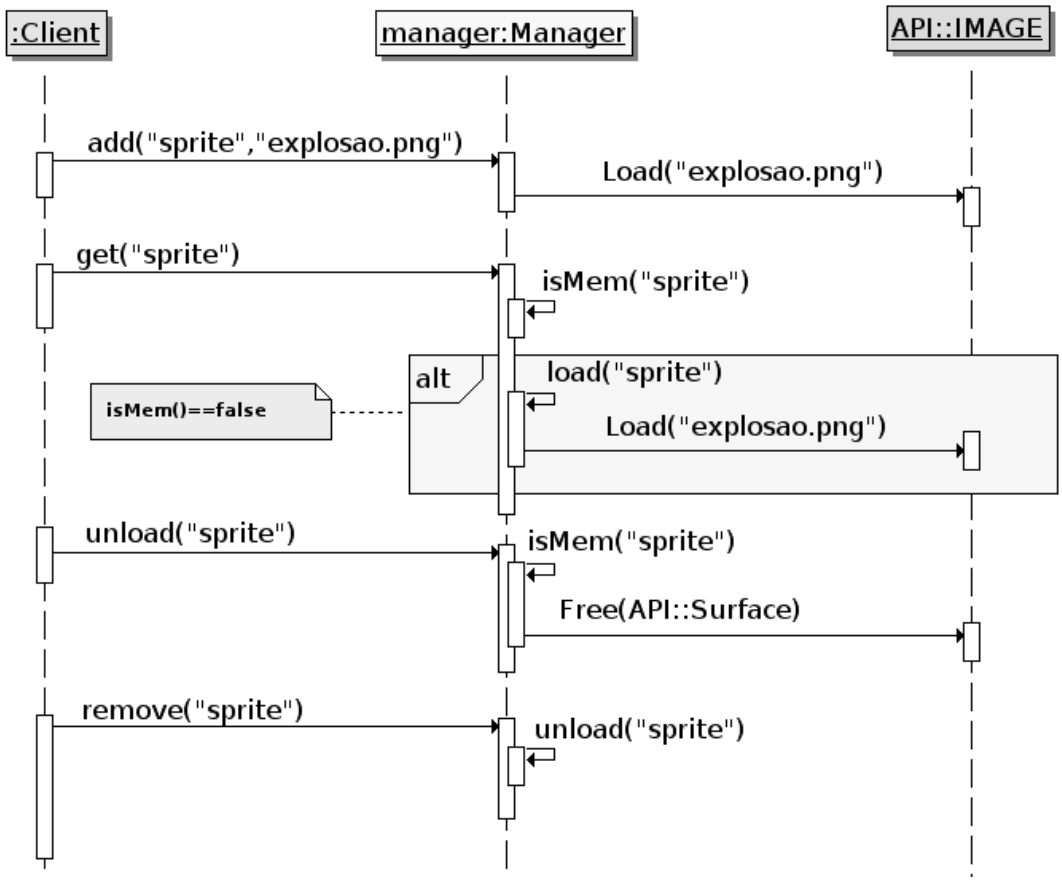

Figura 9. Resource Manager - Diagrama de Sequência.

Onde:

Client é a representação de um trecho de código que faz uso de Manager;
Manager é a classe que representa a implementação do padrão;
API::IMAGE é a representação do manipulador de gráficos e vídeo de uma API Multimedia.

Observando o código de exemplo da classe Manager (Código 2), destacamos alguns detalhes:

1. Para armazenar cada um dos mapeamentos de pares, foi utilizado um objeto do tipo map (STL);
2. O tipo API::Surface e a funções API::IMAGE::Load são definições da API Multimedia;
3. O método load, armazena no mapObject um ponteiro para a estrutura da API::Surface em memória;
4. A manipulação de adição, pesquisa, remoção nos objetos mapPath e mapObject, são providas pelo contêiner da STL (C++);

Observando o código de exemplo da classe Manager (Código 2), destacamos alguns detalhes:

1. Para armazenar cada um dos mapeamentos de pares, foi utilizado um objeto do tipo map (STL);
2. O tipo API::Surface e a funções API::IMAGE::Load são definições da API Multimedia;
3. O método load, armazena no mapObject um ponteiro para a estrutura da API::Surface em memória;
4. A manipulação de adição, pesquisa, remoção nos objetos mapPath e mapObject, são providas pelo contêiner da STL (C++);

Código 2. Código da classe Manager

```
class Manager {
  public:
    void add(std::string key, std::string filename);
    bool load(std::string key);
    bool unload(std::string key);
    bool remove(std::string key);
    API::Surface * get(std::string key);

  protected:
    std::map<std::string,std::string> mapPath;
    std::map<std::string,API::Surface*> mapObject;

  private:
    bool isMem(std::string key);
};

bool Manager::load(std::string key) {
    bool result = false;

    if (mapPath.find(chave)!=mapPath.end()){
        API::Surface * image = API::IMAGE::Load(mapPath[key].c_str());

        if (image!=NULL){
            mapObject[key] = image;
            result = true;
        }
    }

    return result;
}

bool Manager::unload(std::string key) {
    if (isMem(key)==true){
        API::IMAGE::Free(mapObject[key]);
        mapObject[key]=NULL;
    }
}

void Manager::add(std::string key, std::string filename) {
    mapPath[key]=filename;
    mapObject[key]=NULL;
}

bool Manager::remove(std::string key) {
    if (mapPath[key]){
        unload(key);
        mapObject.erase(key);
        mapPath.erase(key);
    }
}

API::Surface * Manager::get(std::string key) {
```

```
    if (isMem(key)==false){
        if (load(key)==false){
            //Error!
        }
    }
    return mapObject[key];
}

bool Manager::isMem(std::string key) {
    bool result = false;

    if ((mapObject.find(key)!=mapObject.end())&&
        (mapObject[key]!=NULL)){
        result = true;
    }

    return result;
}
```

Observando o código de exemplo do Client (Código 3), destacamos alguns detalhes:

1. É necessário obter uma instancia de Resource Manager;
2. De posse da instância solicitamos, o carregamento de um recurso;
3. Com o recurso podemos utilizá-lo normalmente;
4. Ao final podemos solicitar ao Resource Manager que o recurso seja removido;

Código 3. ICódigo do Client, exemplo de utilização do padrão Resource Manager.

```
void main() {

    bool inLoop = true;

    API::Start("Resource Manager",640,480);

    Manager manager;
    manager.add("sprite","data//imagem//explosao.png");

    API::Surface * explosao1 = manager.get("sprite");
    API::Surface * explocao2 = manager.get("sprite");

    //Loop do Jogo
    while(inLoop) {

        //Atualiza disposivitos de Entrada
        API::INPUT::Update();

        //Sair do jogo
        if (API::INPUT::Key(KEY_ESCAPE)){
            inLoop = false;
        }

        //Desenha a surface no backbuffer na posição 100,100
        API::VIDEO::Blit(explosao1,100,100);
```

```
        //Desenha a surface no backbuffer na posição 200,200
        API::VIDEO::Blit(explosao2,200,200);

        API::VIDEO::Flip();
    }

    manager.remove("sprite");

    API::Finish();
}
```

2.2.11 Usos Conhecidos

O artigo “A Simple Fast Resource Manager using C++ and STL” [MAHTAB e WALI,2000], publicado na GameDev.net em maio de 2008, apresenta como implementar rapidamente um gerenciador de recursos. No artigo, é apresentado o foco de implementação no uso da STL, juntamente com o suporte de templates de C++.

O blog “Game Programming Snippets”, apresenta uma postagem intitulado “Resource Manager Snippet” [MORVICK,2008], onde são demonstrados fragmento de código, escrito na linguagem Java, armazenando recursos como imagens e sons para utilização em jogos.

O framework GBF [GBF,2005], implementa o padrão Resource Manager para permitir que jogos desenvolvidos com o mesmo possam beneficiar-se das características fornecidas pelo uso do padrão. Onde em sua implementação podemos observar que o framework disponibiliza alguns gerenciadores especializados como por exemplo: FXManager (gerenciamento de efeitos sonoros), MusicManager (gerenciamento de músicas) e ImageBufferManager (gerenciamento de imagens). Os quais permitem assim a possibilidade de fornecer elementos básicos amplamente utilizados durante o jogo, sem que haja duplicidade na alocação de recursos em memória.

2.2.12 Padrões Relacionados

Façade [GOF,1995], é empregado no padrão, pois permite que as operações da API Multimídia fiquem ocultas das classes usuárias dos gerenciadores;

Singleton [GOF,1995], deve ser empregado no padrão para que exista apenas um gerenciador de recursos por tipo, permitindo que o mesmo seja acessível por todas as partes da aplicação;

2.3 Animated Game Elements

2.3.1 Intenção

Permitir a criação de animação para jogos, baseado no uso de imagens estáticas.

2.3.2 Outro Nome

Elementos Animado do Jogo

2.3.3 Motivação

Quando estamos diante de um jogo eletrônico, nossa primeira percepção recaí sobre os gráficos, principalmente sobre o movimento hipinótico exercido por eles, ou seja, nosso foco visual é rapidamente atraído para os objetos que se movem na cena, ou seja, as animações. Desta forma é imprescindível ao desenvolvermos um jogo, elaborarmos a capacidade de dar movimento as imagens, fazendo uso extensivo deste recurso conhecido como animação, o qual é o responsável pela completude da vida do personagem e objetos na cena.

Tendo isso em mente, necessitamos da capacidade de unir as diversas imagens estáticas (ver Figura 10) de forma padronizada e ordenada (ver Figura 11) para que possamos efetuar as mudanças de quadro de forma controlada, causando o efeito desejado, ou seja, o movimento fluído do personagem ou objeto (ver Figura 12).

Para melhorarmos o entendimento sobre o padrão, utilizaremos algumas convenções:

Sprite é como são conhecidas as animações nos jogos;

SpriteSheet é o conjunto de imagens colocadas lado a lado no mesmo arquivo de imagem;

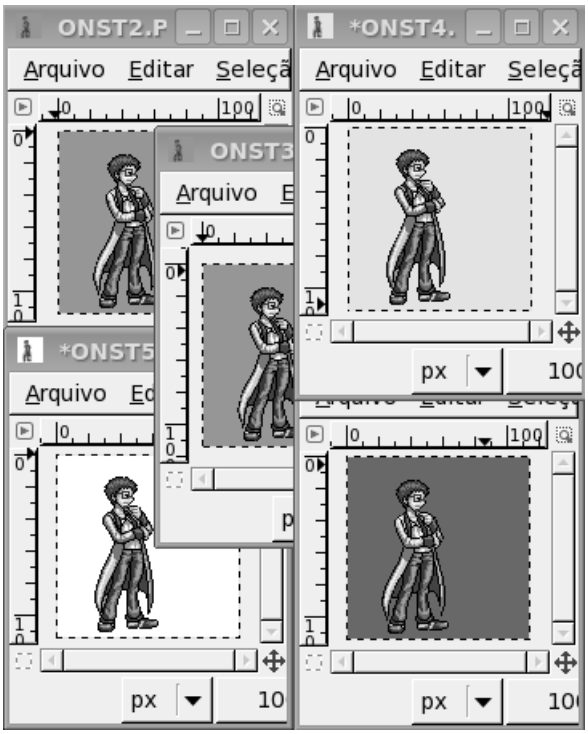

Figura 10. Imagens estáticas, desordenadas e não padronizadas.

Figura 11. Imagem agrupada, padronizada e ordenada.

Figura 12. Animação sendo exibida.

2.3.4 Aplicabilidade

Use o padrão Animated Game Elements quando:

é necessário reunir diversas imagens, porém apenas uma é exibida por vez;
é desejável padronizar e ordenar sequencialmente diversas imagens;
é necessário criar a ideia de movimentação;

2.3.5 Estrutura

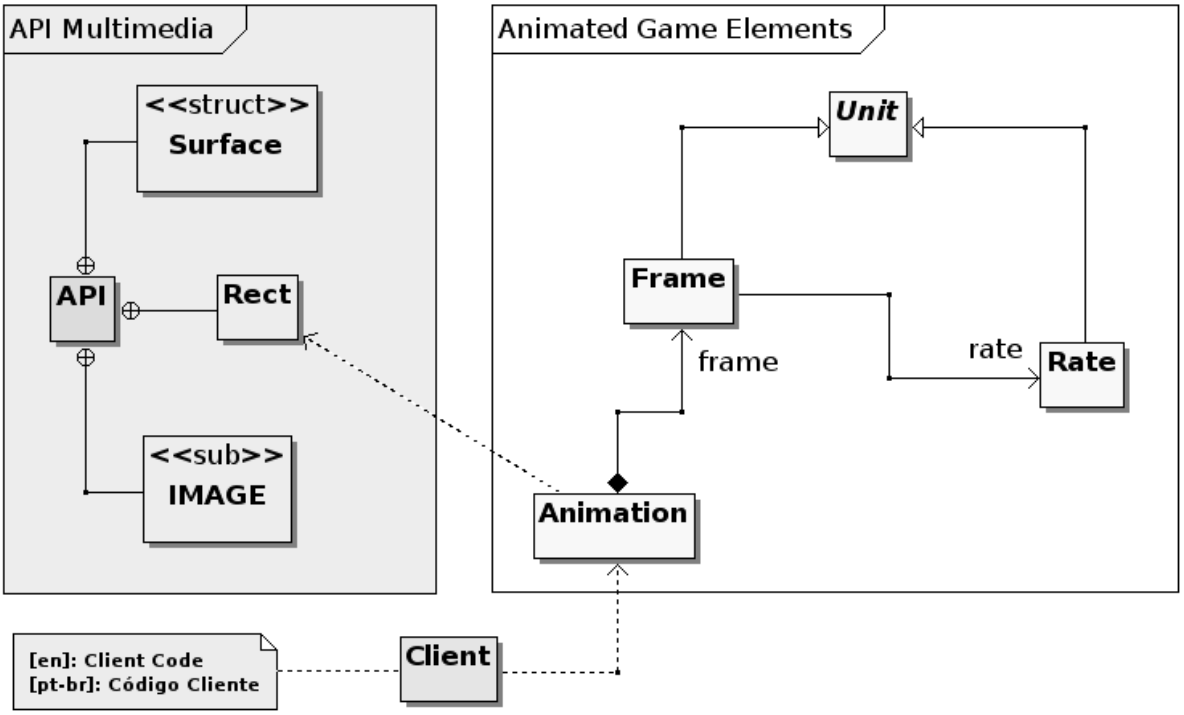

Figura 13. Estrutura do padrão Animated Game Elements e suas colaborações.

2.3.6 Participantes

Surface << API Multimedia>>: estrutura de dados para representar o arquivo de imagem armazenado em memória, é um componente ou estrutura de dados provida pela biblioteca multimídia;

Rect << API Multimedia>>: estrutura de dados para representar uma região no espaço, é dada pelo par ordenado (x,y) e pelo complemento de x (largura) e y (altura);

Unit: tipo abstrato que delimita as operações básicas para processamento de contadores;

Rate: implementação concreta do super tipo Unit, especializando o contador para processar a passagem de tempo;

Frame: implementação concreta do super tipo Unit, especializando o contador para processar a mudança de imagem, de acordo com a passagem de tempo informada pelo Rate correspondente;

Animation: classe responsável por encapsular e gerenciar o ciclo de controle dos frames do jogo;

Client: colabora utilizando um Animation;

2.3.7 Colaborações

O Client inicia uma instância de Animation, configurando os aspectos sobre a dimensão de um quadro, quantidade de quadros e duração dos mesmos;

O Client solicita de Animation, as informações (x, y, largura, altura) sobre a área da SpriteSheet a ser desenhada;

Animation, invoca o processamento de Frame, que por sua vez solicita a Rate que seja processado, o qual determina até quando o quadro atual será exibido e quando haverá a mudança para o próximo quadro;

O Client invoca da API Multimedia, a função para realizar o desenho do Sprite na tela, informando a Surface contendo a SpriteSheet em memória, Rect com as dimensões da área a ser desenhada e a posição do Sprite na tela;

2.3.8 Consequências

O Animated Game Elements tem como consequências:

Precisão de Controle, permitindo o controle preciso da duração de cada quadro do frame, assim como a mudança do quadro;

Otimização de Memória, por permitir o desacoplamento entre a imagem (Surface) e o controle de quadros;

Simplificação do Carregamento, por exigir a unificação dos diversos arquivos de imagens que compõem a mesma animação;

Padronização, por exigir o agrupamento, ordenação e a definição de uma paleta de cores em comum, principalmente para determinação da cor de transparência, também conhecida como colorkey;

Aumento de Complexidade, por ser uma solução baseada na modelagem de classes e suas responsabilidades, exige-se uma maior capacidade mental de visualização e entendimento dos programadores, se compararmos a uma implementação baseada em programação estruturada e monolítica, na qual não existe uma separação clara de responsabilidades;

Perda de Performance, por ser uma solução baseada em uma orquestração de objetos insere-se neste contexto uma perda de performance se compararmos a uma implementação monolítica em programação estrutura, ou seja, que não apresente um modelo de classes baseado em responsabilidades, ou seja, que não esteja baseada nas boas práticas de análise e projeto de sistemas;

2.3.9 Implementação

Para implementar o padrão Animated Game Elements, devemos ter em mente:

1. Precisaremos de um arquivo contendo as imagens para a animação;
2. Precisaremos conhecer as dimensões exatas dos quadros de forma unificada, assim como a duração da exibição de cada quadro;
3. Precisaremos armazenar em memória o arquivo de imagem;

Considere os seguintes aspectos de implementação quando utilizar o padrão Animated Game Elements:

1. Animated Game Elements, pode obter seu elemento Surface de um Resource Manager especializado em gráficos, permitindo assim que a mesma imagem seja utilizada em várias animações simultaneamente, sem interferência;
2. Algumas vezes é interessante que animação possua dois controles de mudança de quadros, uma automática onde os quadros são processados durante a chamada para obtenção da área a ser desenhada e outra manual, onde a mudança de quadro só pode ocorrer enquanto uma certa ação está sendo executada, como por exemplo a mudança de quadros enquanto o personagem esta andando, o que significa que neste momento uma determinada tecla está sendo pressionada e deve-se então haver o processamento da contagem de quadros;
3. Cada Frame pode possuir um Rate com tempo diferenciado, permitindo que um Frame demore mais que os outros em exibição ou seja mais rápido;
4. A classe Frame, funciona como uma coleção de instâncias de Rate, onde a classe é responsável apenas por controlar a mudança de quadros e determinar a dimensão do quadro a ser desenhado;
5. Em ambientes com altas restrições de memória, é indicado que as dimensões dos quadros do Sprite sejam otimizados para conter apenas áreas preenchidas, ou seja, cada quadro possui um tamanho diferente. Neste caso é necessário armazenar na estrutura de controle da animação a localização de cada hotspot, ou seja, do ponto central para desenho de cada quadro, para evitar mudanças bruscas na animação;
6. O Padrão Animated Game Elements, pode ser encapsulado em uma classe de Personagem ou de Itens, para melhorar o desacoplamento, isolando as configurações para a animação e seu controle das classes clientes, criando uma interface unificada e centralizada como no padrão Facade [GOF,1995];

2.3.10 Exemplo de Código

Para exemplificação da aplicação do padrão Animated Game Elements, considere a Figura 14.

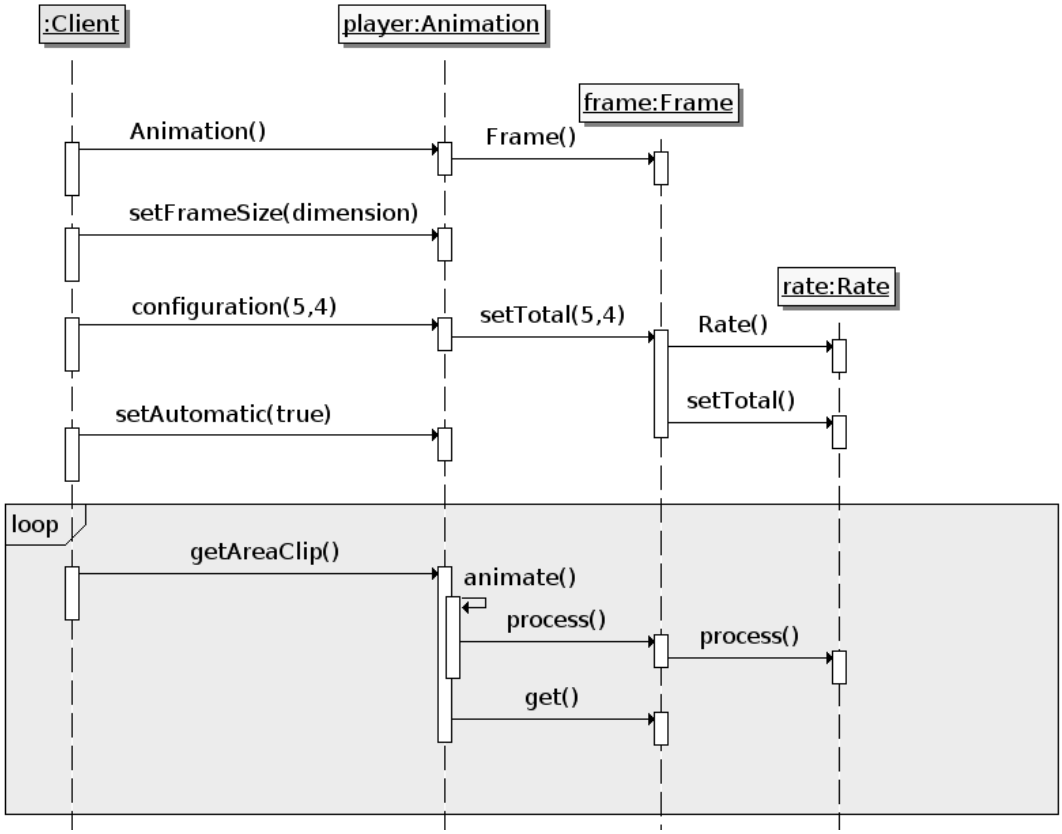

Figura 14. Animated Game Elements - Diagrama de Sequência.

Onde:

Client é a representação de um trecho de código que faz uso de Animation;

Animation é a classe de fachada que coordena a implementação e execução do padrão;

Rate é uma classe interna ao padrão, sendo especializada no processamento e contagem de tempo;

Frame é uma classe interna ao padrão, sendo especializada na mudança de imagem, baseada na informação passada pela classe Rate;

Observando o código de exemplo da classe Unit (Código 4), destacamos alguns detalhes:

1. É o super tipo para classes contadoras, mantendo o estado da contagem atual, e o total máximo de sua contagem;
2. O método isEnd, é abstrato pois suas implementações concretas que definirão quando a contagem termina;

Código 4. Código da classe Unit.

```
class Unit {
  public:
    Unit();
    virtual ~Unit();
    virtual int get();
    virtual void setBegin();
    virtual void setEnd();
    virtual bool isEnd() = 0;

  protected:
```

```
    int current;
    int total;
};

Unit::Unit() {
    total=0;
    current=0;
}

Unit::~Unit() {
}

int Unit::get() {
    return current;
}

void Unit::setBegin() {
    current=0;
}

void Unit::setEnd() {
    current=total-1;
}
```

Observando o código de exemplo da classe Rate (Código 5), destacamos alguns detalhes:

1. Implementa a classe Unit, especializando a contagem para a duração referente a quadro;
2. O método processar, o qual além de proceder com a contabilidade da duração do quadro, irá auxiliará a classe Frame a determinar quando haverá a mudança para o próximo quadro;

Código 5. Código da classe Rate.

```
class Rate : public Unit {
  public:
    Rate();
    virtual ~Rate();
    void setTotal(int total);
    bool isEnd();
    bool process();
};

Rate::Rate():Unit() {
}

Rate::~Rate() {
}

void Rate::setTotal(int total) {
    this->total=total;
}

bool Rate::isEnd() {
    if (current==total){
```

```
        return true;
    } else {
        return false;
    }
}

bool Rate::process() {
    current++;

    if (current>total){
        current = 0;
        return true;
    } else {
        return false;
    }
}
```

Observando o código de exemplo da classe Frame (Código 6), destacamos alguns detalhes:

1. Implementa a classe Unit, especializando a contagem para a quantidade de quadros;
2. O método process, irá determinar quando haverá a mudança de quadro, contando com o auxilio do objeto Rate correspondente;
3. O método isEnd, é implementando informando que está no último quadro e que o tempo de exibição do mesmo foi encerrado;

Código 6. Código da classe Frame.

```
struct Frame : public Unit {
  public:
    Frame();
    virtual ~Frame();
    void setTotal(int total, int rate);
    void setTotal(int total, int rate[]);
    bool isBegin();
    bool isEnd();
    void process();

  protected:
    Rate * rate;
};

Frame::Frame():Unit() {
}

Frame::~Frame() {
    delete[](rate);
}

void Frame::process() {
    if (rate[current].process()){
        current++;
        if (current>=total){
            current = 0;
```

```
        }
    }
}

void Frame::setTotal(int total, int rate) {
    this->total=total;
    this->rate = new Rate[total];

    for (int i=0; i<total; i++){
        this->rate[i].setTotal(rate);
    }
}

void Frame::setTotal(int total, int rate[]) {
    this->total=total;
    this->rate = new Rate[total];

    for (int i=0; i<total; i++){
        this->rate[i].setTotal(rate[i]);
    }
}

bool Frame::isBegin() {
    if (current==0){
        return true;
    } else {
        return false;
    }
}

bool Frame::isEnd() {
    if ((current==total-1) && (rate[current].isEnd())){
        return true;
    } else {
        return false;
    }
}
```

Observando o código de exemplo da classe Animation (Código 7), destacamos alguns detalhes:

1. A classe é responsável por determinar se a mudança de quadros é continua, ou seja em loop ou se é finita;
2. O método getAreaClip, é responsável por determinar a área da SpriteSheet a ser desenhada;
3. O método configuration foi sobrecarregado para permitir Rate com o mesmo valor ou com valores diferenciados informado em formato de array;

Código 7. Código da classe Animation.

```
class Animation {
  public:
    Animation();
    virtual ~Animation();
    bool isBegin();
```

```
    bool isEnd();
    API::Rect getFrameSize();
    API::Rect getAreaClip();
    void setAutomatic(bool automatic);
    void setInLoop(bool inLoop);
    void setEnd();
    void setFrameSize(const API::Rect & area);
    void setBegin();
    void configuration(int numberOfFrames, int repetitionRate);
    void configuration(int numberOfFrames, int repetitionRate[]);
    void processManual();

  protected:
    Frame frame;
    bool automatic;
    API::Rect frameSize;
    bool inLoop;

  private:
    void animate();

};

Animation::Animation() {
    automatic = false;
    inLoop    = true;
}

Animation::~Animation() {
}

bool Animation::isBegin() {
    return frame.isBegin();
}

bool Animation::isEnd() {
    return frame.isEnd();
}

API::Rect Animation::getFrameSize() {
    return  frameSize;
}

API::Rect Animation::getAreaClip() {
    if (automatic){
        animate();
    }

    API::Rect area = frameSize;
    area.x = frameSize.x + (frameSize.w * frame.get());
    return area;
}
```

```
void Animation::setAutomatic(bool automatic) {
   this->automatic=automatic;
}

void Animation::setInLoop(bool inLoop) {
   this->inLoop=inLoop;
}

void Animation::setEnd() {
   frame.setEnd();
}

void Animation::setFrameSize(const API::Rect & area) {
   frameSize=area;
}

void Animation::setBegin() {
   frame.setBegin();
}

void Animation::configuration(int numberOfFrames, int repetitionRate) {
   frame.setTotal(numberOfFrames,repetitionRate);
}

void Animation::configuration(int numberOfFrames, int repetitionRate[]) {
   frame.setTotal(numberOfFrames,repetitionRate);
}

void Animation::processManual() {
   animate();
}

void Animation::animate() {
   if((inLoop)||(!frame.isEnd())){
      frame.process();
   }
}
```

Observando o código de exemplo do Client (Código 8), destacamos alguns detalhes:

1. É carregada para uma instância de API::Surface o arquivo contendo a SpriteSheet do personagem (Figura 11);
2. É configurado um API::Rect (dimension), baseado na SpriteSheet informando a posição inicial do primeiro quadro (x,y), bem como sua largura(w) e altura(h);
3. Uma instância de Animation é configurada com as informações sobre a dimensão, a quantidade de quadros, a duração e se é automática;
4. É configurado um API::Rect (positionClip), para determinar a posição do Sprite na tela;
5. O método getAreaClip, é o responsável por determinar a área da SpriteSheet a ser desenhada;
6. O método API::IMAGE::Blit da API, é invocado passando os parâmetros necessários para que possa ocorrer o desenho do Sprite na tela;

Código 8. Código do Client, exemplo de utilização do padrão Animated Game Elements.

```
int main(int argc, char * argv[]) {
   bool inLoop = true;

   API::Start("Animated Game Elements",400,200);
   API::Surface * surface = API::IMAGE::Load("personagem.png");

   API::Rect dimension;
   dimension.x=0;  dimension.y=0;
   dimension.w=42; dimension.h=98;

   Animation player;
   player.setFrameSize(dimension);
   player.configuration(5,4);
   player.setAutomatic(true);

   API::Rect positionClip;
   positionClip.x=200;
   positionClip.y=60;

   while(inLoop) {

      API::INPUT_Update();

      if (API::INPUT_Key(KEY_ESCAPE)){
         inLoop = false;
      }

      //Determina a área da imagem a ser desenhada
      API::Rect areaClip=player.getAreaClip();

      //Desenha a surface no backbuffer
      API::IMAGE::Blit(surface, areaClip, positionClip);

      API::IMAGE::Flip();
   }
   delete(surface);
   API::Finish();
   return 0;
}
```

2.3.11 Usos Conhecidos

Animated Game Elements foi amplamente utilizado em arquiteturas de hardware dos consoles da era 8 bits e 16 bits, respectivamente consoles como o Nes (Nintendo), o Mega Drive (SEGA) e SNes (Super Nintendo), onde seu hardware já provia uma memória de vídeo maior e dedicada ao armazenamento de gráficos para exibição;

A produtora Techfront [TECHFRONT,2009], especializada em jogos casuais para diversas plataformas, possui sua própria engine, a qual permite que os mesmos recursos sejam utilizados nos mais diversos hardwares, para isso ela conta com a aplicação de alguns padrões, dentre eles temos o Animated Game Elements, pois assim permite-se que os jogos possuam animações de forma controlada e fluída para rodar em diferentes dispositivos de vídeo. Como exemplo podemos citar os jogos Burguer Island para Nintendo Wii/DS e PC, e o Puzzle City para Nintendo Wii/DS e PC.

A produtora virtual DukItan Software [DUKITAN,2007] disponibiliza dois frameworks que fazem uso do padrão Animated Game Elements, são eles o GBFramework [GBF,2005] e o TuGA Game API [TUGA,2008], onde o primeiro é um framework em C++ multiplataforma para a criação de jogos 2D e o segundo é um middleware java para jogos em TV Digital.

2.3.12 Padrões Relacionados

Façade [GOF,1995], é empregado no padrão, pois permite que as classes cliente de Animacao, não conheçam como é o funcionamento da duração e mudança de quadros;

Wrapper Façade [POSA,1996], é empregado para isolar o código orientado a objetos de uma API Multimídia procedural, as quais são as mais comumente encontradas, por exemplo a Simple DirectMedia Layer [SDL,1999];

Resource Manager, é empregado para permitir o compartilhamento de surfaces (imagens) de forma a otimizar os recursos utilizados para criação das animações;

2.4 Language Manager

2.4.1 Intenção

Permitir que uma aplicação ou jogo possa suportar múltiplos idiomas.

2.4.2 Outro Nome

Gerenciador de Idioma

2.4.3 Motivação

Ao se desenvolver um jogo, uma das primeiras decisões que se toma é a definição de qual será o idioma de seu público alvo, onde geralmente acaba-se optando para que o jogo esteja apenas em um único idioma. Porém, com isso eliminamos a possibilidade do mesmo entrar em contato com outros públicos, simplesmente pela barreira da língua.
Sendo assim, um jogo em um mundo globalizado deve permitir ou se adaptar ao idioma nativo de seus jogadores (ver Figura 15 e Figura 16), sem comprometer sua estrutura interna, não sendo aceitável que seja mantido duas ou três versões do mesmo código para gerar versões em português, inglês ou espanhol.

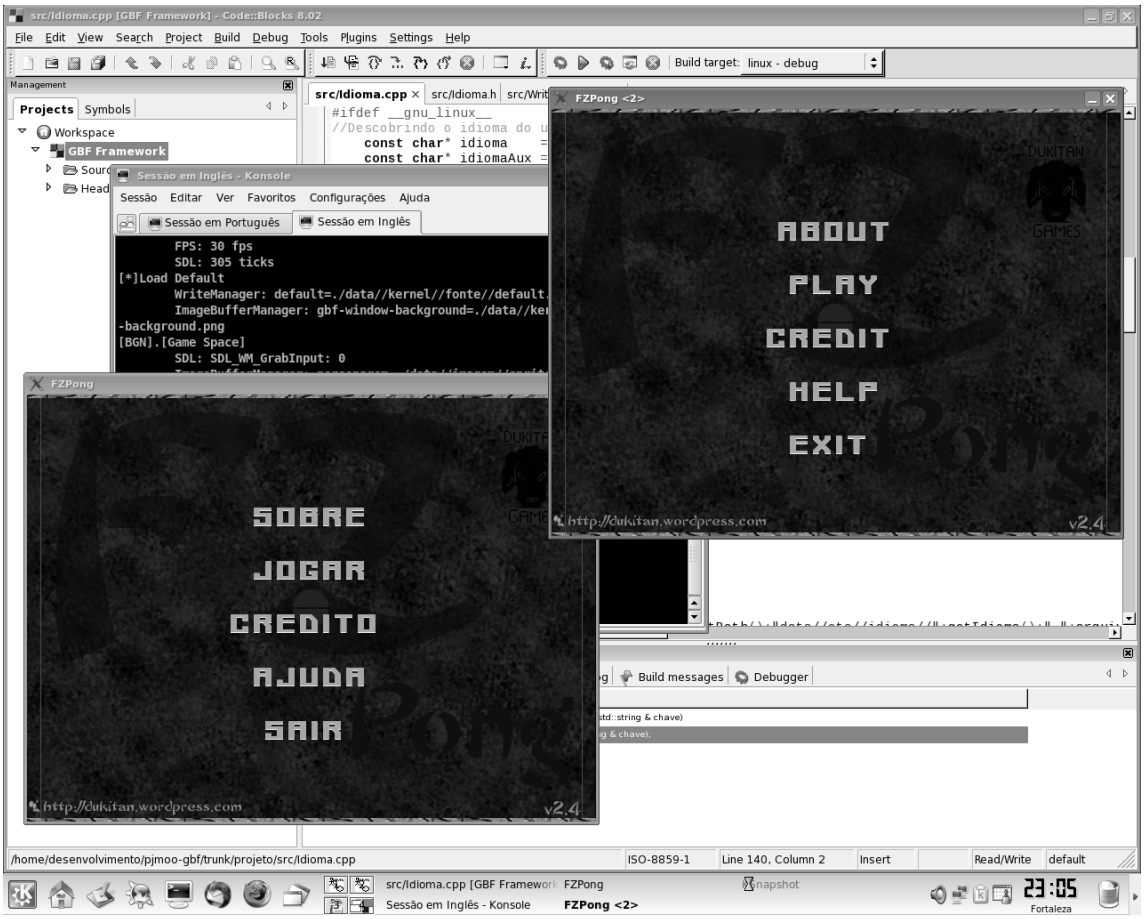

Figura 15. Imagem do jogo FZPong. O jogo a esquerda está utilizado o idioma Português, e o da direita está em Inglês.

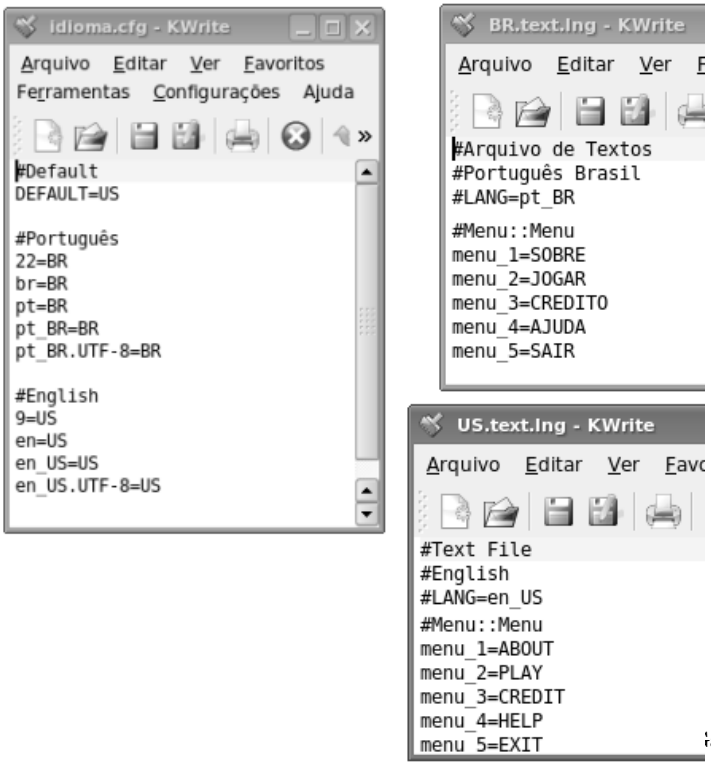

Figura 16. Arquivos de linguagens e mapeamento de idioma do jogo.

2.4.4 Aplicabilidade

Use o padrão Language Manager quando:
é desejável que os textos do jogo sejam facilmente atualizáveis, sem a necessidade de gerar-se uma nova versão executável;
é necessário suportar múltiplos idiomas;

2.4.5 Estrutura

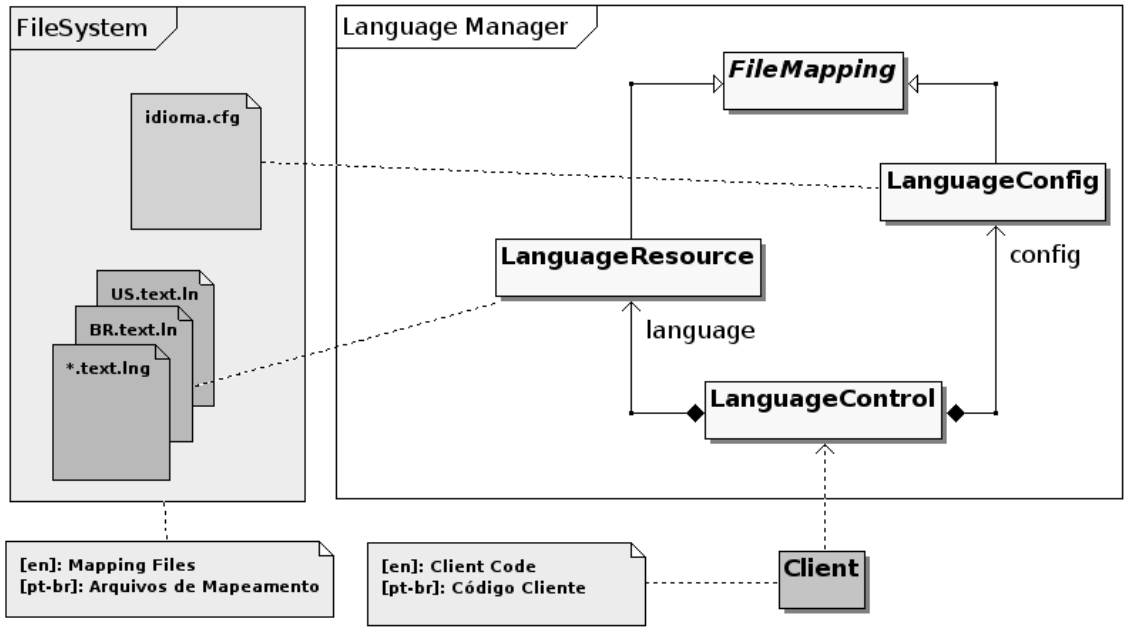

Figura 17. Estrutura do padrão Language Manager e suas colaborações.

2.4.6 Participantes

idioma.cfg: mapeamento de idiomas, arquivo de dados contendo as informações sobre os idiomas suportados;

*.text.lng: arquivos de linguagem, contendo os textos traduzidos para cada idioma suportado;

LanguageControl: gerencia o suporte a idiomas no jogo;

LanguageConfig: representação em memória do mapeamento de idiomas;

LanguageResource: representação em memória de um arquivo de linguagem selecionado;

FileMapping: classe abstrata que possibilita o carregamento das informações dos arquivos para uma tabela (map) em memória;

Client: colabora utilizando o LanguageControl;

2.4.7 Colaborações

O Client inicia acionando o método load de LanguageControl, que por sua vez aciona load de LanguageConfig, em seguida LanguageControl auto invoca o método autodetect, que se encarrega de selecionar o idioma, que por sua vez, invocação o método load de LanguageResource;

O Client invoca do LanguageControl a obtenção de um recurso de texto, por meio do método getText, onde LanguageControl retorna o texto de acordo com o idioma do usuário;

2.4.8 Consequências

O Language Manager tem como consequências:

Manutenibilidade das Informações, por externalizar da aplicação os textos de informação, é possível a correção ou alteração sem o comprometimento da estrutura da aplicação;

Adaptação ao Usuário, por permitir que a aplicação se adapte ao idioma do usuário, de forma automática ou seletiva;

Abstração em Alto Nível, aplicando o padrão Façade [GOF,1995], evita-se que Client tenha o conhecimento sobre o funcionamento interno do padrão e seus relacionamentos, assim como sobre como obter do sistema operacional as informações do usuário, e o carregamento dos arquivos de configuração e linguagem;

Aumento de Complexidade, por ser uma solução baseada na modelagem de classes e suas responsabilidades, exige-se uma maior capacidade mental de visualização e entendimento dos programadores, se compararmos a uma implementação baseada em programação estruturada e monolítica, na qual não existe uma separação clara de responsabilidades;

Perda de Performance, por ser uma solução baseada em uma orquestração de objetos insere-se neste contexto uma perda de performance se compararmos a uma implementação monolítica em programação estrutura, ou seja, que não apresente um modelo de classes baseado em responsabilidades, ou seja, que não esteja baseada nas boas práticas de análise e projeto de sistemas;

2.4.9 Implementação

Para implementar o padrão Language Manager, devemos ter em mente:

1. Precisaremos do arquivo de configuração de idiomas;
2. Precisaremos dos arquivos de linguagens de idiomas;
3. Precisaremos encapsular o conteúdo dos arquivos em objetos, os quais armazenarão as informações em pares chave-texto, para a obtenção das informações;

Considere os seguintes aspectos de implementação quando utilizar o padrão Language Manager:

1. LanguageControl, pode ser implementado como um Singleton [GOF,1995], para mantermos apenas uma única instância na aplicação;
2. LanguageControl, aplica o padrão Façade [GOF,1995], encapsulando a colaboração com as demais classes envolvidas no padrão;
3. FileMapping, armazena o conteúdo de um arquivo texto em map (Contêiner STL), similar ao padrão Resource Manager;
4. FileMapping, aplica o padrão Template Method [GOF,1995], para permitir que as classes que implementem o padrão informem o arquivo a ser carregado;
5. LanguageConfig e LanguageResource, herdam de FileMapping para poder encapsular o conteúdo de um arquivo texto em objeto;
6. O método autodetect de LanguageControl, é dependente da plataforma alvo, pois necessita interagir com as APIs do Sistema Operacional, em C/C++ para isolarmos esse comportamento especifico fazemos uso das diretivas de compilação #ifdef e #ifndef permitindo assim que código seja compilado apenas na plataforma compatível;

2.4.10 Exemplo de Código

Para exemplificação da aplicação do padrão Language Manager, considere a Figura 18.

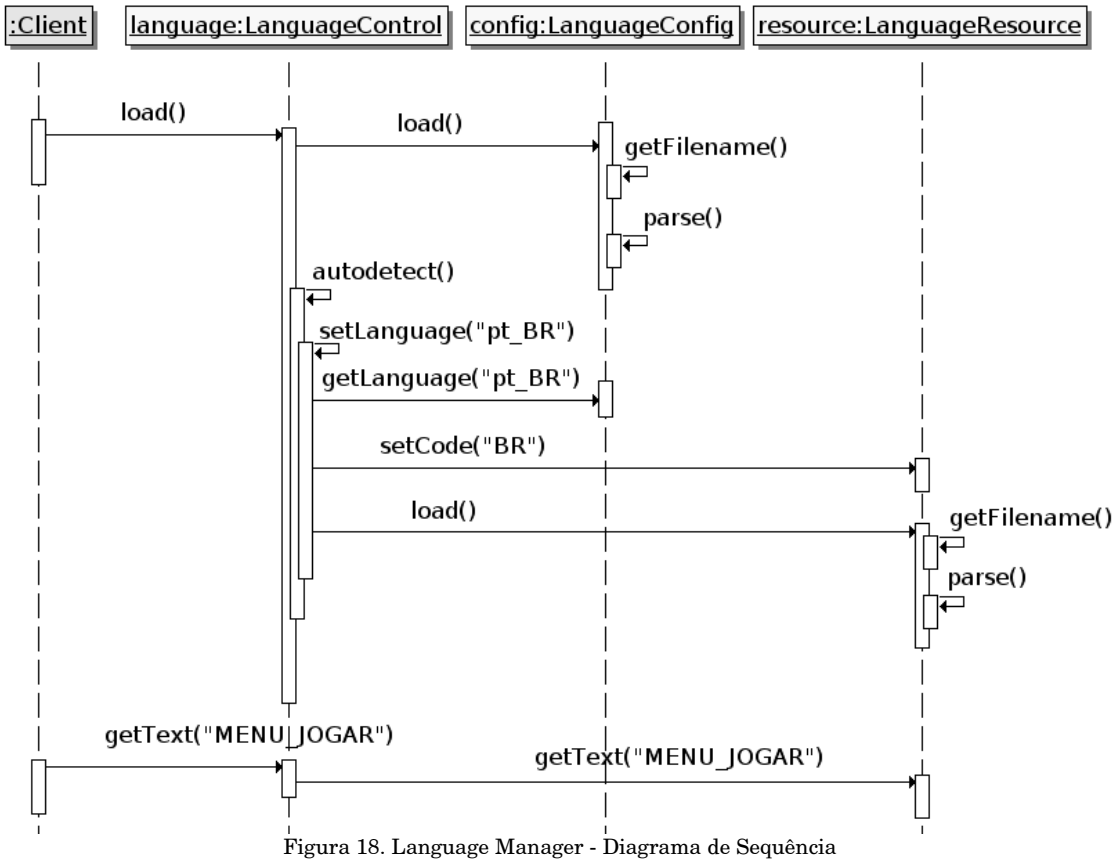

Figura 18. Language Manager - Diagrama de Sequência

Onde:

Client representação de um trecho de código que faz uso de LanguageControl;

LanguageControl é a classe de fachada que coordena a implementação e execução do padrão;

LanguageConfig é uma classe interna ao padrão, sendo responsável por informar quais os idiomas suportados;

LanguageResource é uma classe interna ao padrão, sendo responsável por armazenar o idioma utilizado;

Observando o código de exemplo da classe FileMapping (Código 9), destacamos alguns detalhes:

1. Para armazenar um mapeamento chave-valor, foi utilizado um objeto do tipo map (STL);
2. É utilizado na classe o padrão Template Method [GOF,1995], no método getFilename();
3. A classe tem como objetivo transportar para si o conteúdo de um arquivo, respeitando as regras de interpretação (método parser);

Código 9. Código da classe FileMapping

```
class FileMapping {
  public:
    bool load();
    void unload();

  protected:
```

```
    std::map<std::string,std::string> content;
    virtual void parse(char * info);
    virtual std::string getFilename() = 0;
};

void FileMapping::parse(char * info) {
    int i = 0;
    std::string line = info;
    int size = line.length();

    if (line[0]!='#'){
        for (i=0;i<size;i++){
            if (line[i]=='=') {
                break;
            }
        }
    }
    if ((i>0)&&((i+1)<size)){
        content[line.substr(0,i)]=line.substr(i+1,size);
    }
}

bool FileMapping::load() {
    char str[256];
    bool result = false;

    std::string fullpath=getFilename();

    if (!fullpath.isEmpety()){
        std::fstream file(fullpath.c_str(),std::ios::in);
        if (file!=NULL){
            unload();
            while(!file.eof()){
                file.getline(str,256);
                parser(str);
            }
            file.close();
            result = true;
        } else {
            std::cerr << "[ERROR] File Not Found : " << fullpath << std::endl;
        }
    } else {
        std::cerr << "[ERROR] File undefined :" << fullpath <<  std::endl;
    }

    return result;
}

void FileMapping::unload() {
    content.clear();
}
```

Observando o código de exemplo da classe LanguageConfig (Código 10), destacamos alguns detalhes:

1. A classe por estender FileMapping e seguindo o padrão Template Method [GOF,1995] é obrigada a implementar o método getFilename;
2. O método getLanguage é o responsável por traduzir o código de idioma, por seu prefixo mapeado no arquivo;

Código 10. Código da classe LanguageConfig.

```
class LanguageConfig : public FileMapping {
  public:
    std::string getLanguage(std::string code);

  protected:
    virtual std::string getFilename();
};

std::string LanguageConfig::getLanguage(std::string code) {
    if ((!code.empty()) && (content.find(code)!=content.end())){
        return content[code]
    }
}

std::string LanguageConfig::getFilename() {
    return Kernel::Util::Path::getPath()+"//data//etc//idioma.cfg";
}
```

Observando o código de exemplo da classe LanguageResource (Código 11), destacamos alguns detalhes:

1. A classe por estender FileMapping e seguindo o padrão Template Method [GOF,1995] é obrigada a implementar o método getFilename;
2. O método setCode armazena o prefixo do idioma a ser utilizado;
3. O método getFilename fornece o nome do arquivo de linguagem a ser carregado, com base no prefixo do idioma;
4. O método getText com base em uma chave retorna o texto correspondente de acordo com o idioma pré-selecionado;

Código 11. Código da classe LanguageResource.

```
class LanguageResource : public FileMapping {
  public:
    void setCode(std::string code);
    std::string getText(const std::string & key);

  protected:
    virtual std::string getFilename();

  private:
    string code;
```

```
};

void LanguageResource::setCode(std::string code) {
   this->code=code;
}

std::string LanguageResource::getText(const std::string & key) {
   if (content.find(key)!=content.end()){
      return content[key];
   } else {
      return "ERRO";
   }
}

std::string LanguageResource::getFilename() {
   return Kernel::Util::Path::getPath()
      + "data//etc//idioma//"+code+".text.lng";
}
```

Observando o código de exemplo da classe LanguageControl (Código 12), destacamos alguns detalhes:

1. A classe é uma Façade [GOF,1995], por abstrair e centralizar a interação entre as classes do padrão (LanguageConfig e LanguageResource);
2. O método autodetect invoca a execução de um método de acordo com o Sistema Operacional. Devido ao suporte da linguagem C++, a decisão de qual método deve ser invocado, é feita em tempo de compilação, já que a detecção para o sistema MS-Windows requer uma integração com a WinAPI, por isso apresenta-se o uso de #ifdef;

Código 12. Código da classe LanguageControl.

```
class LanguageControl {
  public:
    bool load();
    void setLanguage(std::string code);
    std::string getText(std::string key);

  protected:
    LanguageConfig config;
    LanguageResource resource;
    void autodetect();

  private:
    void detectWindows();
    void detectLinux();
};

bool LanguageControl::load(){
   config.load();
   autodetect();
}
```

```
void LanguageControl::autodetect(){
#ifdef __gnu_linux__
   detectLinux();
#else
   detectWindows();
#endif
}

void LanguageControl::setLanguage(std::string code) {
   std::string localCode = config.getLanguage(code);
   resource.setCode(localCode);
   resource.load();
}

void LanguageControl::detectWindows() {
#ifndef __gnu_linux__
   char code[6];
   int id = GetUserDefaultLangID();
   sprintf(code,"%d",id & 0x3ff);
   if (!setLanguage(code)){
      setLanguage("DEFAULT");
   }
#endif
}

void LanguageControl::detectLinux() {
#ifdef __gnu_linux__
   const char* code    = getenv("LC_ALL");
   const char* codeExt = getenv("LANG");
   if (code==NULL){
      if (codeExt==NULL){
         setLanguage("DEFAULT");
      } else {
         if (!setLanguage(codeExt)){
            setLanguage("DEFAULT");
         }
      }
   } else {
      if (!setLanguage(code)){
         setLanguage("DEFAULT");
      }
   }
#endif
}

std::string LanguageControl::getText(std::string key) {
   return language.getText(key);
}
```

Observando o código de exemplo do Client (Código 13), destacamos alguns detalhes:

1. É necessário obter uma instância de LanguageControl;
2. De posse da instância podemos solicitar o carregamento automático de idiomas;

3. Podemos solicitar a instância de LanguageControl à obtenção do texto desejado, passando para o método getText uma chave de localização;

Código 13. Código do Client, exemplo de utilização do padrão LanguageManager.

```
void main() {
   LanguageControl language;

   language.load();

   Font font;
   font.loadFromFile("arial.png");

   font.write(0,0,language.getText("MENSAGEM_ENTRADA"));

   font.write(320,100,language.getText("MENU_JOGAR"));
   font.write(320,120,language.getText("MENU_SOBRE"));
   font.write(320,140,language.getText("MENU_SAIR"));
}
```

2.4.11 Usos Conhecidos

O jogo SpaceShooter [SPACESHOOTER,2005], um shooting'up clássico com temática de Star Trek, está disponível em três idiomas, sendo eles: português, espanhol e inglês, onde a implementação do padrão oferece ao jogador a possibilidade de mudança de idioma. Um idioma inicial é carregado de acordo com o idioma usado pelo sistema operacional, permitindo neste primeiro momento uma facilidade maior para os jogadores mais leigos, porém em seu menu principal, pelo acionamento do item de opções, é exibido para o jogador as três opções de idiomas.

O jogo FZPong [FZPONG,2007], produzido pela DukItan Games, é um clássico jogo de pong, o qual faz uso do padrão, porém a seleção de idioma é transparente para o jogador, onde apenas durante a primeira execução da aplicação o idioma do jogo é selecionado e assim mantido até a próxima execução. Está estratégia pode apresentar alguns inconvenientes, como por exemplo, a falta de opção do jogador em mudar o idioma. Por exemplo, é comum no Brasil e em outros países onde a língua nativa não é o inglês de encontrarmos pessoas utilizando sistema operacional configurado no idioma estrangeiro para permitir um treino na linguagem, o que de fato pelo emprego da estratégia de seleção de idiomas transparente não permite o jogador, selecionar o seu idioma nativo ou muito menos conhecer que existia a possibilidade, e assim recomendar o jogo por possuir esta característica.

O jogo Dark Colony [DARKCOLONY,1997], produzido em 1997 pela Strategic Simulations, é um jogo de estratégia em tempo real, com forte apelo sci-fi, já que o mesmo retrata um futuro onde Humanos e Grays (raça extraterrestre) disputam os recursos naturais energéticos do planeta Marte. Neste jogo, o foco da implementação foi dado na capacidade de se externalizar os textos permitindo assim uma fácil localização, inclusive contando com uma maior aproximação da comunidade de fãs, estreitando o relacionamento entre produtor e comunidade. Porém o mecanismo utilizado para seleção do idioma ocorre apenas durante a instalação do jogo, e que pode ser parte da estratégia de vendas da produtora.

2.4.12 Padrões Relacionados

Façade [GOF,1995], é empregado no padrão, permitindo que as colaborações entre LanguageConfig e LanguageResource fiquem ocultas dos utilizadores de LanguageManager;

Singleton [GOF,1995], deve ser empregado no padrão para que exista apenas um gerenciador de idioma, permitindo que o mesmo seja acessível por todas as partes da aplicação;

Template Method [GOF,1995], é utilizado para permitir que as classes que herdam de FileMapping, possam informa qual o arquivo que deverá ser carregado e analisado para preencher o map (Contêiner STL);

Resource Manager, a classe FileMapping, apresenta uma similaridade com o padrão, visto que o mesmo trabalha armazenando objetos em um contêiner do tipo map, porém com a diferença de que as informações armazenadas em FileMapping, dizem respeito apenas a um mapeamento direto, e não a utilização ou otimização de recursos externos;

2.5 Font Mapping

2.5.1 Intenção

Permitir que aplicações multimídias possam escrever informações(textos) na tela para o usuário.

2.5.2 Outro Nome

Mapeamento de Fonte

2.5.3 Motivação

Estamos desenvolvendo uma aplicação multimídia como por exemplo um player de vídeo ou um jogo eletrônico, porém necessitamos exibir informações para o usuário, geralmente essas informações são textos composto por caracteres alfa numéricos, que representam informações sobre o estado do jogo ou do filme. Nestes casos, na maioria das vezes, desejamos utilizar textos de fontes estilizadas, quer seja pelo seu formato ou pelo estilo utilizado, como por exemplo o uso de sombras, bordas ou degradê.

Para tanto, faz-se uso de uma imagem contendo todos os caracteres desejados, incluindo a formatação e o estilo, como podemos observar na Figura 19 e Figura 20

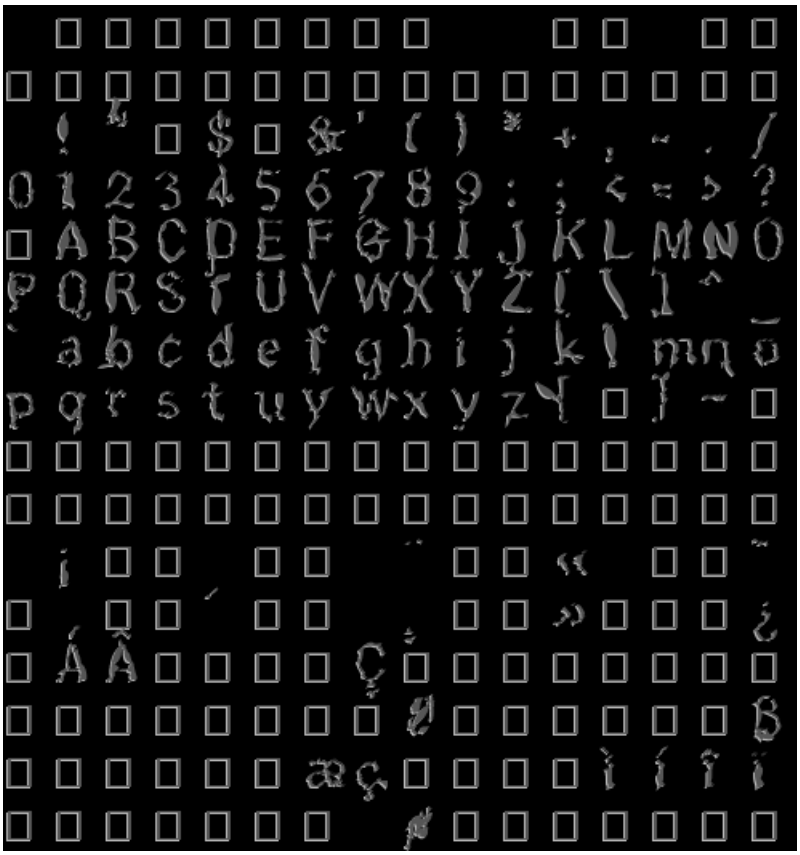

Figura 19. Imagem contendo caracteres

Figura 20. Imagem ampliada para melhor visualizarmos o estilo utilizado.

2.5.4 Aplicabilidade

Use o padrão Font Mapping quando:

é necessário escrever utilizando uma fonte estilizada (cores e formatos personalizados);

não é possível garantir ou não é desejável que o usuário da aplicação tenha a fonte instalada;

é necessário escrever no vídeo (tela) e não existe a funcionalidade nativa da linguagem ou na biblioteca multimídia utilizada;

2.5.5 Estrutura

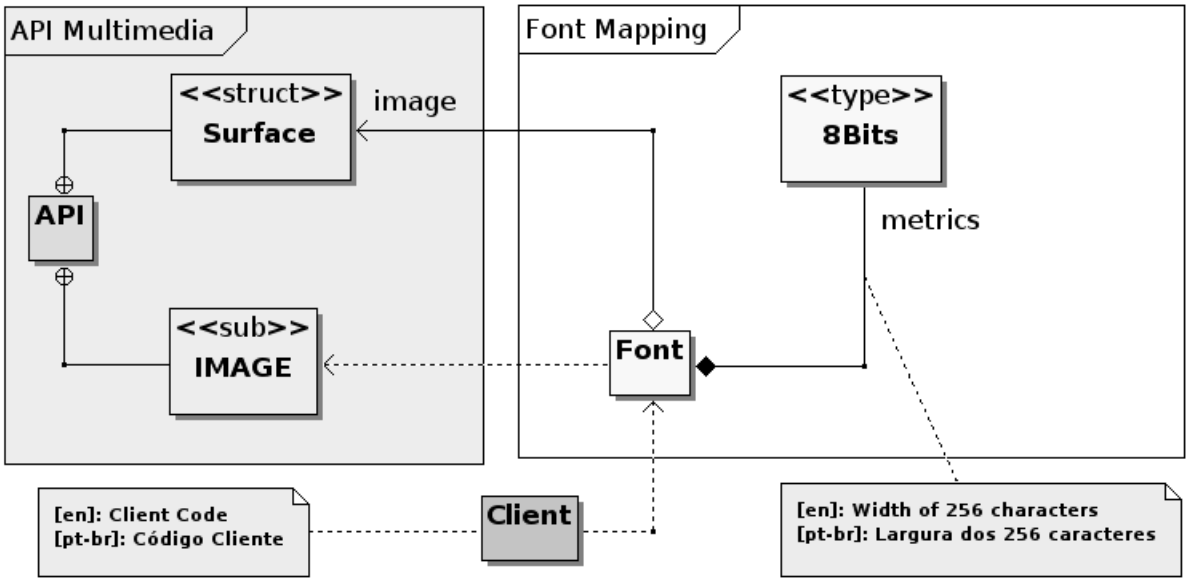

Figura 21. Estrutura do padrão Font Mapping e suas colaborações.

2.5.6 Participantes

Surface <<API Multimedia>>: estrutura de dados para representar o arquivo de imagem armazenado em memória, é um componente ou estrutura de dados provida pela biblioteca multimídia;

IMAGE <<API Multimedia>>: representação do manipulador de vídeo, provê operações ou funções de acesso a baixo nível disponibilizado pela biblioteca multimídia (I/O);

Font: define o mapeamento entre o texto e a imagem;

8Bits <<type>>: tipo de dados de 8 bits, utilizado para armazenar a métrica, tamanho de cada letra, dos 256 caracteres;

Client: colabora utilizando Font;

2.5.7 Colaborações

A Rotina (ou fragmento de código) inicia solicitando a instanciação e o carregamento de uma Fonte, a qual por sua vez, comunica-se com a API Multimídia, para o carregamento da imagem e das informações de métricas;

A Rotina (ou fragmento de código) solicita a execução do método escrever de Fonte, que por sua vez interage com a API Multimídia para realizar a operação de exibição da imagem na tela;

2.5.8 Consequências

O Font Mapping tem como consequências:

Abstração em alto nível, evitando que Rotina tenha o conhecimento sobre a API Multimídia, visto que a classe Fonte funciona como um adaptador (Adapter [GOF,1995]) entre o os dados (parâmetros) passados pela Rotina, ao que de fato a API Multimídia realmente necessita;

Otimização no Processamento, por utilizar-se de uma imagem pré-renderizada e armazenada, evita-se o processamento decorrente da conversão do formato de arquivo de fonte para a memória de vídeo em tempo de execução;

Independência de Dispositivo, por utilizar-se da pré-renderização, qualquer dispositivo que possua suporte gráfico para imagens pode fazer uso dos caracteres desejados, sem necessariamente possuir suporte ao formato de fonte original, permitindo assim a utilização em dispositivos portáteis, computadores e consoles de vídeo games.

Aumento de Complexidade, por ser uma solução baseada na modelagem de classes e suas responsabilidades, exige-se uma maior capacidade mental de visualização e entendimento dos programadores, se compararmos a uma implementação baseada em programação estruturada e monolítica, na qual não existe uma separação clara de responsabilidades;

Perda de Performance, por ser uma solução baseada em uma orquestração de objetos insere-se neste contexto uma perda de performance se compararmos a uma implementação monolítica em programação estrutura, ou seja, que não apresente um modelo de classes baseado em responsabilidades, ou seja, que não esteja baseada nas boas práticas de análise e projeto de sistemas;

2.5.9 Implementação

Para implementar o padrão Font Mapping, devemos ter em mente:

1. Precisaremos de um arquivo de imagem com os caracteres a serem utilizados;
2. Precisaremos armazenar as informações de cada letra representada na imagem, como por exemplo largura;
2. Precisaremos criar o mapeamento entre cada letra a ser escrita(desenhada) na tela com sua respectiva representação dentro da imagem;

Considere os seguintes aspectos de implementação quando utilizar o padrão Font Mapping:

1. Fonte não necessariamente precisa interagir diretamente com a API Multimídia, ela pode interagir com uma outra classe que funcione como uma Façade [GOF,1995] geral para a API Multimídia.
2. Fonte não necessariamente deve ser responsável pelo gerenciamento da surface, em casos mais robustos implementa-se um GraphicManager (ver padrão ResourceManager) para controle, otimização e compartilhamento de imagens na aplicação.
3. Fonte deve armazenar as métricas (largura) do conjunto de caracteres utilizados, geralmente utiliza-se um tipo de dado que armazene 8bits em array.
4. Pode-se implementar um gerenciador de fontes WriteManager (ver padrão ResourceManager), que deve permitir o armazenamento e compartilhamento das fontes carregadas por toda aplicação.

2.5.10 Exemplo de Código

Para exemplificação da aplicação do padrão Font Mapping, considere a Figura 22.

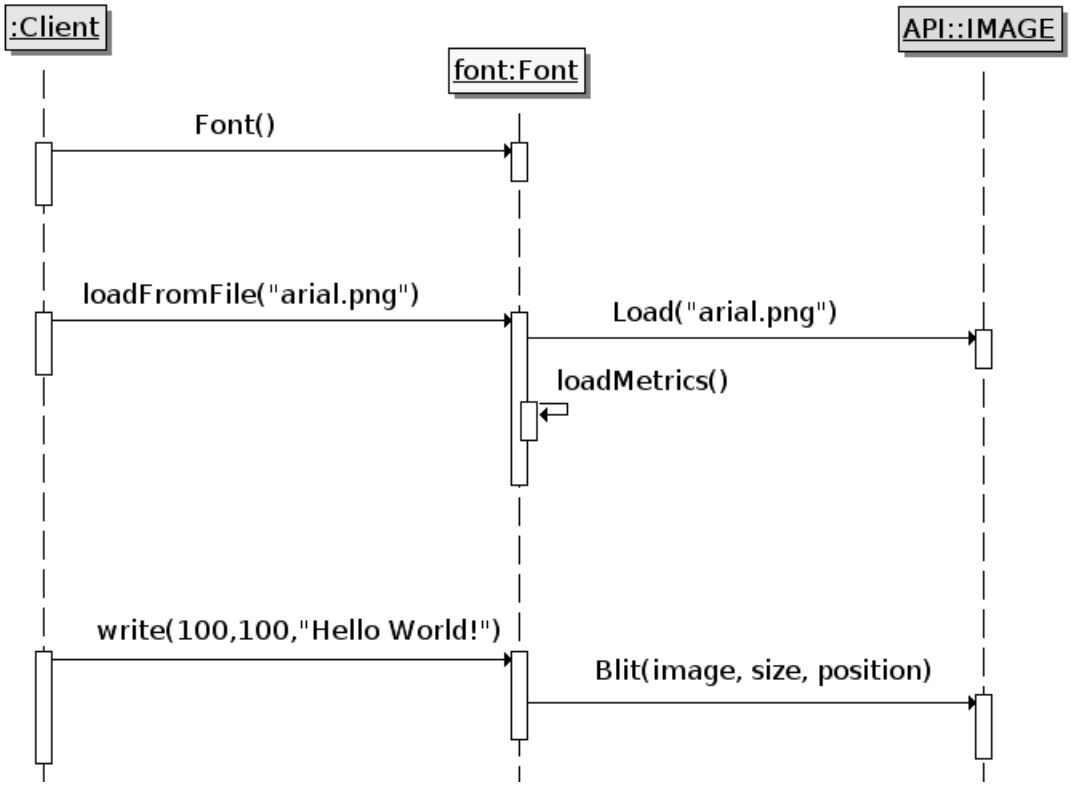

Figura 22. Font Mapping - Diagrama de Sequência

Onde:

Client é a representação de um trecho de código que faz uso de Font;
Font é a classe que representa a implementação do padrão;
API::IMAGE é a representação do manipulador de gráficos e vídeo de uma API Multimedia;

Observando o código de exemplo da classe Fonte (Código 14), destacamos alguns detalhes:

1. Para armazenar a métrica de cada letra é utilizado um array de char;
2. Os tipos API::Surface e API::Rect com as funções API::IMAGE::Load e API::IMAGE::Blit são definições da API Multimedia;
3. O tipo FILE e as funções fopen, fread e fclose são definições da linguagem C (ANSI C) para manipulação de arquivos;
4. A API::Surface disponibiliza por meio dos atributos w (weight) e h (height) informações sobre o tamanho da imagem;
5. As imagens utilizadas armazenam os caracteres sempre em 16 linhas por 16 colunas, com isso dividindo o tamanho da imagem por 16, podemos calcular a área da imagem em que cada letra está desenhada;
6. Para escrever na tela, a string de texto é percorrida e individualmente cada letra é desenhada;

Código 14. Código da classe Font.

```
class Font {
  public:
    Font();
    virtual ~Font();
```

```
    bool loadFromFile(std::string filename);
    void write(int x, int y, const char text);
    void loadMetrics(std::string filename);

  protected:
    char metrics[256];
    API::Surface * image;
    API::Rect letterFrame;
};

bool Font::loadFromFile(std::string filename) {
    image = API::IMAGE::Load(filename.c_str());
    if (image!=NULL){
        loadMetrics(filename)
        letterFrame.w=image->w/16;
        letterFrame.h=image->h/16;
        return true;
    } else {
        return false;
    }
}

void Font::write(int x, int y, const char text) {
    int i,t=strlen(text);
    unsigned char l;
    API::Rect position, size;
    position.x=x;
    position.y=y;
    for (i=0; i<t; i++){
        l=text[i];
        size.x=(l%16)*letterFrame.w; size.w=metricas[l];
        size.y=(l/16)*letterFrame.h; size.h=letterFrame.h;

        API::IMAGE::Blit(image, size, position);

        position.y=y;
        position.x=position.x+metrics[l];
    }
}

void Font::loadMetrics(std::string filename) {
    FILE *file;
    std::string txt="";
    txt=filename.substr(0,filename.length()-4);
    txt+=".dat";

    file = fopen(txt.c_str(),"rb");
    if (file!=NULL){
        fread(&metrics, 256, 1, file);
        fclose(file);
    } else {
        for (int l=0;l<256;l++){
            metrics[l] = image->w/16;
```

```
      }
   }
}
```

Observando o código de exemplo do Client (Código 15), destacamos alguns detalhes:

1. É necessário obter uma instancia de Font;
2. De posse da instância solicitamos o carregamento de uma imagem com as fontes;
3. De posse da fonte, solicitamos a escrita (desenho) na tela;
4. Desalocamos a fonte quando não for mais necessário;

Código 15. Código do Client, exemplo de utilização do padrão Font Mapping.

```
void main(){
   Font * font = new Font();
   font->loadFromFile("arial.png");
   font->write(100,100,"Hello World!");
   delete(font);
}
```

2.5.11 Usos Conhecidos

A produtora de jogos Dynamic Games [DYNAMIC,2008], é uma das empresas que fazem extenso uso de Font Mapping, onde podemos destacar os jogos War Operations (Estratégia em Tempo Real), Dynamic Parkint (Puzzle), Advergame Ocesp (Criado para a Ocesp utilizar em suas feiras) e Os Conquistadores (Épico de Ação 3D) os quais fazem a integração desse padrão com a API Gráfica do DirectX.

O jogo de corrida 3D, Penguin Racer [PENGUIN,2004] da produtora ICON Games [ICON,2006], é outro exemplo de aplicação do padrão Font Mapping aplicando-se de fato, fontes estilizadas (com degrades), integrando o padrão a API OpenGL.

A produtora virtual DukItan Software [DUKITAN,2007] disponibiliza diversos demos e jogos que fazem uso do padrão Font Mapping, os quais são disponibilizados gratuitamente junto com seus códigos fontes. Também disponibiliza a ferramenta F2IBuilder [F2IBUILDER,2007], a qual é a responsável por gerar a imagem contendo as letras a serem utilizadas.

A engine Easy2D [EASY2D,2009], que foca no desenvolvimento de jogos 2D com orientação a objetos, permite a utilização de fontes de duas formas, sendo a principal por meio do padrão Font Mapping.

A biblioteca Libwiisprite [LIBWIISPRITE,2008], utilizada para manipulação de imagens em jogos caseiros (homebrew) do console wii, utiliza para escrita de texto o padrão Font Mapping.

2.5.12 Padrões Relacionados

Adapter [GOF,1995], é empregado na classe “Fonte”, pois adapta as informações provenientes da escrita de texto, para a interface disponibilizada pela biblioteca multimídia.

Resource Manager, pode ser empregado na criação de um gerenciador para controlar e facilitar o uso de um conjunto de fontes, previamente carregado ou que se deseje compartilhar com toda aplicação.

3 AGRADECIMENTOS

Gostaríamos de agradecer as inúmeras pessoas que acreditaram na realização deste trabalho, nos incentivando e auxiliando na elaboração.

Em especial, gostaríamos de destacar a participação:

Das comunidades nacionais de desenvolvedores de jogos, tais como: Unidev [UNIDEV,2002], PDJ [PDJ,2003], JogosPro [JOGOSPRO,2002];

Da comunidade local GDJCE [GDJBR,2009], por oferecer espaço para apresentação e discussão sobre os padrões documentados;

Dos amigos por atuarem como consultores e revisores: Alexandre Ribeiro [RIBEIRO,2010], Ricardo Bittencourt [BITTENCOURT R.,2010], Fernando Bittencourt [BITTENCOURT F.,2010], Felipe Lira [LIRA,2010], Daniel Leite [LEITE,2010], Bruno Evangelista [EVANGELISTA,2010], Gustavo Bastos [BASTOS,2010], Karine Roberta [MENDES,2010], Daniel Marques [MARQUES,2010] e Mike Moreira [MOREIRA,2010];

Dos Professores: Dr. Jerffeson Teixeira de Souza [SOUZA J.,2010], Dr. Cidcley Teixeira de Souza [SOUZA C.,2010] e Msc. Milton Escóssia Barbosa Neto [BARBOSA,2010] por atuarem como revisores técnicos e acadêmicos nas áreas de Engenharia de Software, Documentação e Aplicação de Padrões de Software, TV Digital Interativa, Sistemas Hipermídias, Computação Gráfica e Jogos Digitais;

De Federico Balaguer [BALAGUER,2010], por atuar como shepherd no processo de submissão ao SugarLoafPlop [SUGAR,2010], orientando para que este trabalho pudesse atingir o grau de maturidade necessária para ser avaliado pela comunidade de padrões;

Dos participantes da 12ª sessão de Writers Workshop do SugarLoafPlop [SUGAR,2010]: Jerffeson Teixeira de Souza [SOUZA J.,2010] e Daniel Cukier [CUKIER,2010] por colaborarem discutindo os pontos positivos e sugestões de melhorias deste trabalho durante a conferência;

4 REFERÊNCIAS

ARAUJO et al.,2006: Allan R. S. Araujo, Juliana M. Silva, Artur F. Mittelbach, Scrum: Novas Regras do Jogo, 2006

BALAGUER,2010: Federico Balaguer, Linkedin, 2010. Disponível em: <http://ar.linkedin.com/pub/federico-balaguer/13/4b0/132>, Acesso em: 10 ago. 2010

BARBOSA,2010: Milton Escóssia Barbosa Neto, Currículo Lattes, 2010. Disponível em: <http://buscatextual.cnpq.br/buscatextual/visualizacv.jsp?id=K4739143Y7>, Acesso em: 10 ago. 2010

BASTOS,2010: Gustavo Bastos, Linkedin, 2010. Disponível em: <http://br.linkedin.com/pub/gustavo-bastos/7/816/290>, Acesso em: 10 ago. 2010

BITTENCOURT F.,2010: Fernando Bittencourt, Linkedin, 2010. Disponível em: <http://br.linkedin.com/in/frbitten>, Acesso em: 10 ago. 2010

BITTENCOURT R.,2010: Ricardo Bittencourt, Linkedin, 2010. Disponível em: <http://br.linkedin.com/pub/ricardo-bittencourt/5/392/423>, Acesso em: 10 ago. 2010

CUKIER,2010: Daniel Cukier, Linkedin, 2010. Disponível em: <http://br.linkedin.com/in/danielcukier>, Acesso em: 10 nov. 2010

DARKCOLONY,1997: Strategic Simulations, Dark Colony, 1997. Disponível em: <http://pt.wikipedia.org/wiki/Dark_Colony>, Acesso em: 13 mai. 2009

DUKITAN,2007: DukItan Software & Games, Web Site, 2007. Disponível em: <http://www.dukitan.com>, Acesso em: 12 jul. 2009

DYNAMIC,2008: Dynamic Games, Web Site, 2008. Disponível em: <http://www.dynamicgames.com.br/jogos.html>, Acesso em: 20 dez. 2008

EASY2D,2009: Jonatas de Moraes Junior, Easy2D Game Library, 2009. Disponível em: <http://easy2d.sourceforge.net/>, Acesso em: 20 nov. 2009

EVANGELISTA,2010: Bruno Evangelista, Linkedin, 2010. Disponível em: <http://www.linkedin.com/in/brunoevangelista>, Acesso em: 10 ago. 2010

F2IBUILDER,2007: DukItan Software & Games, F2IBuilder, Font To Image Builder, 2007. Disponível em: <http://f2ibuilder.sourceforge.net>, Acesso em: 05 out. 2009

FZPONG,2007: DukItan Software & Games, FZ Pong, 2007. Disponível em: <http://portal.dukitan.com/fzpong>, Acesso em: 13 mai. 2009

GAMEDEV,1999: GameDev.net, Web Site, 1999. Disponível em: <http://www.gamedev.net/>, Acesso em: 10 jun. 2009

GBF,2005: DukItan Software & Games, GBFramework, 2005. Disponível em: <http://portal.dukitan.com/gbframework>, Acesso em: 10 mai. 2009
GDJBR,2009: GDJBR, Grupo de Desenvolvedores de Jogos, 2009. Disponível em: <http://www.gdjbr.com>, Acesso em: 10 ago. 2010
GOF,1995: Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, Padrão de projeto de software, 1995. Disponível em: <http://pt.wikipedia.org/wiki/Padr%C3%A3o_de_projeto_de_software>, Acesso em: 20 nov. 2009
HAWKINGS e ASTLE,2001: Kevin Hawkings, Dave Astle, OpenGL Game Programming, 2001
ICON,2006: ICON Games, Web Site, 2006. Disponível em: <http://www.icongames.com.br/>, Acesso em: 10 dez. 2009
J2EE,2002: Oracle / Sun Microsystems, Core J2EE Patterns, 2002. Disponível em: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html>, Acesso em: 20 nov. 2009
JOGOSPRO,2002: JogosPro, Lista JogosPro, 2002. Disponível em: <http://tech.groups.yahoo.com/group/jogospro/>, Acesso em: 10 ago. 2010
LEITE,2010: Daniel Frederico Leite, Linkedin, 2010. Disponível em: <http://br.linkedin.com/pub/daniel-frederico-lins-leite/23/ab9/173>, Acesso em: 11 ago. 2010
LIBWIISPRITE,2008: Wii Brew, libwiisprite is a C++ sprite library written for the Wii , 2008. Disponível em: <http://wiibrew.org/wiki/Libwiisprite>, Acesso em: 10 dez. 2009
LIRA,2010: Felipe Lira, Linkedin, 2010. Disponível em: <http://br.linkedin.com/in/feliperlira>, Acesso em: 10 ago. 2010
MAHTAB e WALI,2000: Ashic Mahtab; Zinat Wali, A Simple Fast Resource Manager using C++ and STL, 2000. Disponível em: <http://www.gamedev.net/reference/programming/features/resourceMngtCppStl>, Acesso em: 10 mai. 2009
MARQUES,2010: Daniel de Albuquerque Marques, Currículo Lattes, 2010. Disponível em: <http://buscatextual.cnpq.br/buscatextual/visualizacv.jsp?id=K4209709D8>, Acesso em: 11 ago. 2010
MENDES,2010: Karine Roberta Vieira Mendes, Facebook, 2010. Disponível em: <http://pt-br.facebook.com/profile.php?id=100000819386683>, Acesso em: 11 ago. 201
MOREIRA,2010: Mike Moreira, Linkedin, 2010. Disponível em: <http://br.linkedin.com/in/mikemoreira>, Acesso em: 10 ago. 2010
MORVICK,2008: Morbid Morvick, Resource Manager Snippet, 2008. Disponível em: <http://gpsnippets.blogspot.com/2008/07/resource-manager-snippet.html>, Acesso em: 10 mai. 2009
PDJ,2003: PDJ, Programadores e Desenvolvedores de Jogos, 2003. Disponível em: <http://www.pdj.com.br>, Acesso em: 10 ago. 2010
PENGUIN,2004: Icon Games, Penguin Racer, 2004. Disponível em: <http://www.icongames.com.br/pracer>, Acesso em: 10 dez. 2009
POSA,1996: Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal , Pattern-Oriented Software Architecture, 1996. Disponível em: <http://www.hillside.net/component/content/article/53-architecture-requirements-patterns-books/178-pattern-oriented-software-architecture-a-system-of-patterns?directory=127>, Acesso em: 19 nov. 2009
RIBEIRO,2010: Alexandre Ribeiro de Sá, Perfil Linkedin, 2010. Disponível em: <http://br.linkedin.com/in/ardes>, Acesso em: 10 ago. 2010
SANCHES,2009: Bruno Crivelari Sanches, Os softwares de um jogo, 2009. Disponível em: <http://www.pontov.com.br/site/index.php?view=article&id=108>, Acesso em: 18 dez. 2009.
SDL,1999: Simple DirectMedia Layer, Web Site, 1999. Disponível em: <http://www.libsdl.org>, Acesso em: 12 jun. 2009
SOUZA C.,2010: Cidcley Teixeira de Souza, Currículo Lattes, 2010. Disponível em: <http://buscatextual.cnpq.br/buscatextual/visualizacv.jsp?id=K4795182D7>, Acesso em: 10 ago. 2010
SOUZA J.,2010: Jerffeson Teixeira de Souza, Currículo Lattes, 2010. Disponível em: <http://buscatextual.cnpq.br/buscatextual/visualizacv.jsp?id=K4794205D4>, Acesso em: 10 ago. 2010
SPACESHOOTER,2005: DukItan Software & Games, SpaceShooter, 2005. Disponível em: <http://spaceshooter.dukitan.com>, Acesso em: 12 mai. 2009
SUGAR,2010: SugarLoaf Plop, Conferência Latino-Americana em Linguagens de Padrões para Programação, 2010. Disponível em: <http://hillside.net/conferences/sugarloaf-plop>, Acesso em: 29 jul. 2010
TECHFRONT,2009: TechFront - Play it Forward, Web Site, 2009. Disponível em: <http://www.techfront.com.br>, Acesso em: 20 mai. 2009
TUGA,2008: DukItan Software & Games, TuGA Game API, 2008. Disponível em: <http://tuga-sdk.googlecode.com>, Acesso em: 10 jun. 2009
UNIDEV,2002: Unidev, Programação de Jogos, 2002. Disponível em: <http://www.unidev.com.br>, Acesso em: 10 ago. 2010
WIKIPEDIA,2010: Wikipédia, Duas Dimensões e Meia – 2.5D, 2010. Disponível em: <http://pt.wikipedia.org/wiki/2.5D>, Acesso em: 04 ago. 2010
XNADC,2009: XNA Developer Center, Tutorial 4: Make a Game in 60 Minutes., 2009. Disponível em: <http://msdn.microsoft.com/en-us/library/bb975644.aspx>, Acesso em: 10 out. 2009