Curso: **Engenharia de Software** com Ênfase em **Padrões de Software (UECE – Universidade Estadual do Ceará)**

Nome: David de Almeida Ferreira E-mail: davidferreira.fz@gmail.com

Nome: Karine Roberta Vieira E-mail: karinerobertavieira@gmail.com

Font Mapping

Intenção

Permitir a escrita de textos em uma aplicação multimídia.

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 tanta, faz-se uso de uma imagem contendo todos os caracteres desejados, incluindo a formatação e o estilo, como podemos observar na Figura 1.

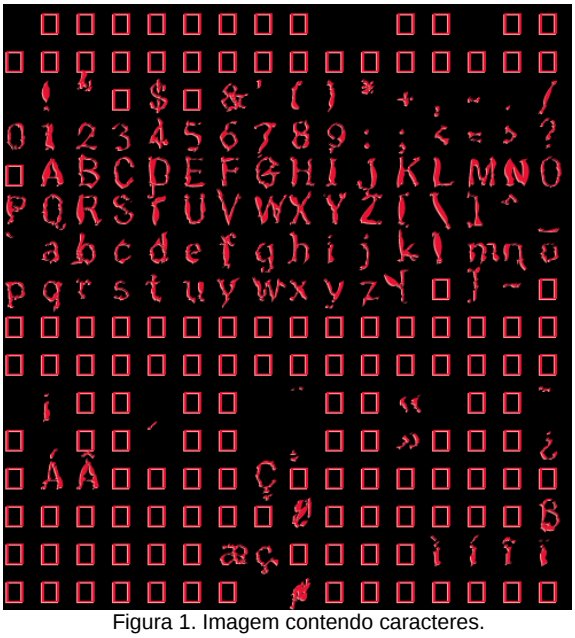

Figura 1. Imagem contendo caracteres.

Figura 2. Imagem ampliada para melhor visualizarmos o estilo utilizado.

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;

Nome: David de Almeida Ferreira E-mail: davidferreira.fz@gmail.com
Nome: Karine Roberta Vieira E-mail: karinerobertavieira@gmail.com

Estrutura

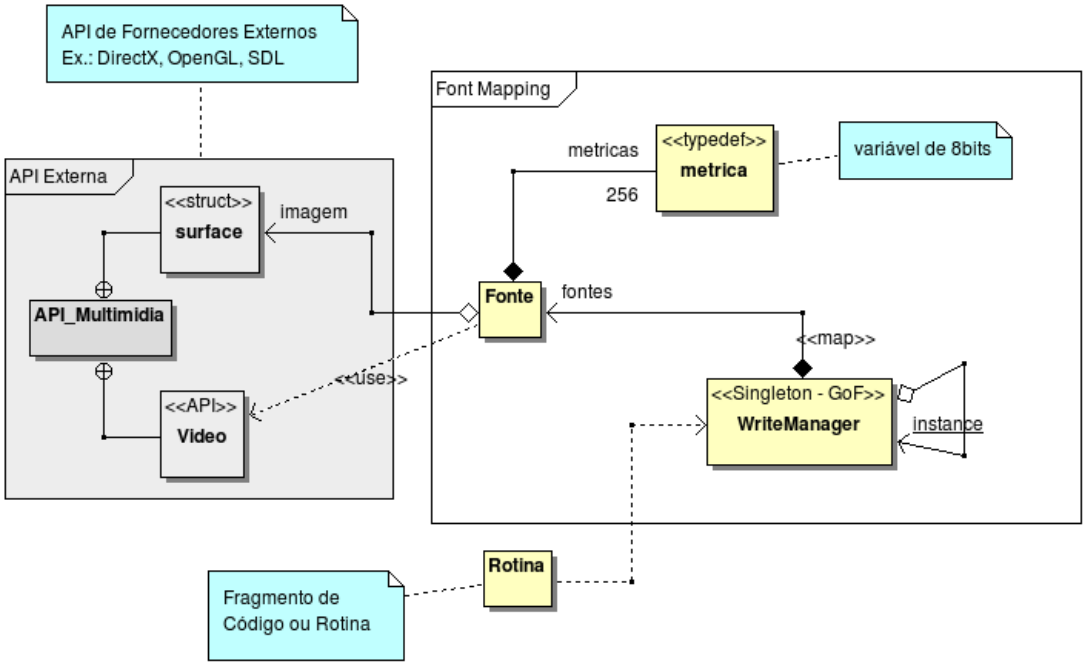

Figura 3. Estrutura do padrão Font Mapping e suas colaborações.

Participantes

- surface <<struct>>
  - estrutura de dados para representar o arquivo de imagem armazenado em memória, é um componente ou estrutura de dados provida pela biblioteca multimídia;
- Video <<API>>
  - representação do manipulador de vídeo, provê operações ou funções de acesso a baixo nível disponibilizado pela biblioteca multimídia;
- Fonte
  - define o mapeamento entre o texto e a imagem(surface);
- metrica <<typedef>>
  - tipo para armazenamento das métricas(tamanho) de cada letra;
- WriteManager
  - armazena as Fontes disponíveis em um pool para futura utilização;
- Rotina
  - colabora utilizando uma fonte previamente armazenada em WriteManager;

Colaborações

- A Rotina (ou fragmento de código) inicia solicitando ao *WriteManager* 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 *WriteManager*, o qual delega a execução para a *Fonte*, que por sua vez interage com a *API Multimídia* para realizar a operação de exibição da imagem na tela;

Nome: David de Almeida Ferreira E-mail: davidferreira.fz@gmail.com
Nome: Karine Roberta Vieira E-mail: karinerobertavieira@gmail.com

Conseqüências

O Font Mapping tem como conseqüências:

- 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) entre o os dados (parâmetros) passados pela *Rotina*, ao que de fato a *API Multimídia* realmente necessita;
- Simplificação, *WriteManager* simplifica o processo de criação e utilização de uma *Fonte*, por meio da aplicação do padrão *Façade* em conjunto com o padrão *Singleton*, tornando-se um gerenciador para escrita de textos;
- Otimização, *WriteManager* internamente controla um pool de objetos *Fontes*, para ser reutilizada por toda a aplicação, minimizando a árdua tarefa de alocação inline das *Fontes*;

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;
3. 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* geral para a API Multimídia.
2. *Fonte* não necessariamente deve ser responsável pelo gerenciamento da *surface*, em casos mais robustos implementa-se um *GraphicManager* (similar ao *WriteManager*) 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. *WriteManager*, deve ser robusto para evitar que quando uma *Fonte* inexistente for solicitada, evite que a aplicação como um todo seja abortada (exceção fatal) ou que não exiba as informações desejada, para isso pode-se:
    1. utilizar-se de uma *Fonte* default, a qual deve ser carregada na inicialização da aplicação;
    2. criar um mecanismo inteligente que possa verificar a possibilidade de carregar a *Fonte* desejada e então disponibiliza-la;

Nome: David de Almeida Ferreira E-mail: davidferreira.fz@gmail.com
Nome: Karine Roberta Vieira E-mail: karinerobertavieira@gmail.com

Exemplo de código

Para exemplificação da aplicação do padrão *Font Mapping*, considere a Figura 4.

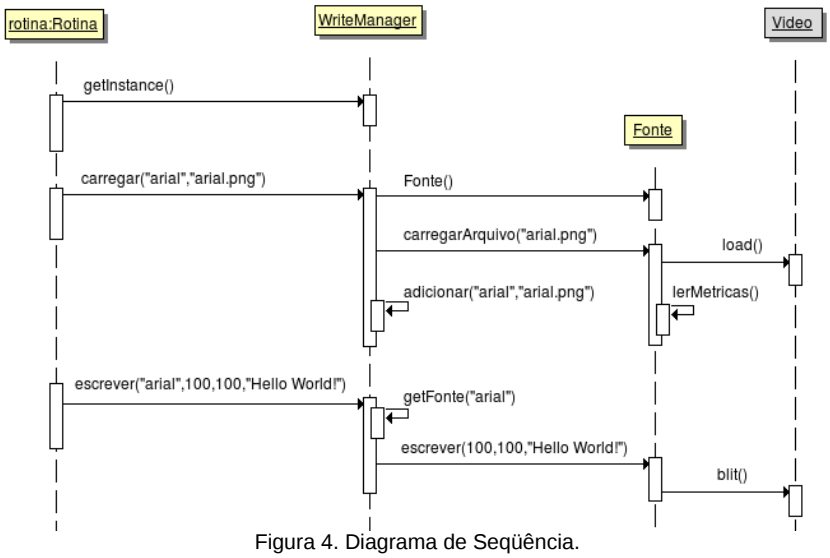

Figura 4. Diagrama de Seqüência.

Onde:

- *rotina* é a representação de um fragmento de código que faz uso de *WriteManager*;
- Video é a representação de uma API Multimídia, nos exemplos a seguir utilizaremos a Simple DirectMedia Layer (SDL) para os exemplos de código.

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

1. Para armazenar a métrica de cada letra é utilizado um array de char;
2. Os tipos *SDL_Surface* e *SDL_Rect* com as funções *IMG_Load* e *SDL_BlitSurface* são definições da SDL;
3. O tipo *FILE* e as funções *fopen*, *fread* e *fclose* são definições da linguagem C (ANSI C) para manipulação de arquivos;
4. A *SDL_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;

```
class Fonte
{
  protected:
    char metricas[256];
    SDL_Surface * imagem;
    SDL_Rect dimensaoQuadro;

  public:
    Fonte();
    virtual ~Fonte();
    bool carregarArquivo(string arquivo);
    void escrever(int x, int y, const char texto);
    void lerMetricas(string arquivo);
};


bool Fonte::carregarArquivo(string arquivo)
{
      imagem = IMG_Load(arquivo.c_str());

      if (imagem!=NULL){
          lerMetricas(arquivo)
```

```
void Fonte::escrever(int x, int y, const char texto)
{
      int i,t=strlen(texto);
      unsigned char l;
      SDL_Rect posicao, tamanho;
      posicao.x=X;
      posicao.y=Y;

      for (i=0; i<t; i++){
          l=texto[i];
          tamanho.x=(l%16)*dimensaoQuadro.w; tamanho.w=metricas[l];
          tamanho.y=(l/16)*dimensaoQuadro.h; tamanho.h=dimensaoQuadro.h;

          SDL_BlitSurface(imagem, &tamanho, surfacePrimaria, &posicao);

          posicao.y=Y;
          posicao.x=posicao.x+metricas[l];
      }
}

void Fonte::lerMetricas(string arquivo)
{
        FILE *arquivoMetricas;
```

Nome: David de Almeida Ferreira E-mail: davidferreira.fz@gmail.com
Nome: Karine Roberta Vieira E-mail: karinerobertavieira@gmail.com

```
            dimensaoQuadro.w=imagem->w/16;
            dimensaoQuadro.h=imagem->h/16;

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

```
        std::string txt="";

        txt=arquivo.substr(0,arquivo.length()-4);
        txt+=".dat";

        arquivoMetricas = fopen(txt.c_str(),"rb");

        if (arquivoMetricas!=NULL){
            fread(&metricas, 256, 1, arquivoMetricas);
            fclose(arquivoMetricas);
        } else {
            for (int l=0;l<256;l++){
                metricas[l] = imagem->w/16;
            }
        }
}
```

Código 1. Código da classe Fonte.

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

1. O pool de *Fontes* é armazenado em um contêiner (Standart Template Library) do tipo *map*, onde cara elemento do pool é formado pelo conjunto chave-objeto, onde a chave é uma *string* e o objeto é uma instância da classe *Fonte*;
2. *WriteManager* é um *Façade* simplificando a utilização de algumas funcionalidades, como por exemplo no carregamento da Fonte, onde a classe já encarrega-se de armazena-la para uso posterior;
3. *WriteManager* delega para *Fonte* algumas operações para de fato serem executadas;

```
class WriteManager
{
  private:
    static WriteManager * instance;
    bool adicionar(string nome, Fonte * fonte);

  protected:
    std::map<std::string,Fonte*> fontes;

  public:
    bool carregar(string nome, string arquivo);
    void escrever();
    static WriteManager * getInstance();
    Fonte * getFonte(string nome);

};

WriteManager * WriteManager::instance;

void WriteManager::adicionar(string nome, Fonte * fonte)
{
     fontes[nome]=fonte;
}

void WriteManager::escrever()
{
     getFonte(fonte)->escrever(texto,x,y);
}
```

```
bool WriteManager::carregar(string nome, string arquivo)
{
      bool status = false;
      Fonte *fonte = new Fonte();

      if (fonte->carregarArquivo(arquivo)){
          if (adicionar(nome,fonte)){
              status = true;
          }
      }

      return status;
}

Fonte WriteManager::getFonte(string nome)
{
      if (fontes.find(nome)!=fontes.end()){
          return fontes[nome];
      } else {
          return NULL;
      }
}
```

Código 2. Código da classe WriteManager.

Observando o código de exemplo da Rotina (Código 3), destacamos alguns detalhes:

1. É necessário obter uma instancia de *WriteManager*;
2. Por meio de *WriteManager*, solicitamos o carregamento de *Fonte* e sua utilização;

```
void main()
{
      WriteManager *writeManager = WriteManager::getInstance();
      writeManager->carregar("arial","arial.png");


      writeManager->escrever("arial",100,100,"Hello World!");
}
```

Código 3. Fragmento de Código baseado na rotina para utilização de WriteManager

Nome: David de Almeida Ferreira E-mail: davidferreira.fz@gmail.com

Nome: Karine Roberta Vieira E-mail: karinerobertavieira@gmail.com

Usos conhecidos

A produtora de jogos Dynamic Games, 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, mais informações sobre os jogos da Dynamic Games podem ser visto no site http://www.dynamicgames.com.br/jogos.html

O jogo de corrida 3D, Penguin Racer da produtora ICON Games, é outro exemplo de aplicação do padrão Font Mapping aplicando-se de fato, fontes estilizadas (com degrades), integrando o padrão a API OpenGL, mais informações sobre o Penguin Racer pode ser visto no site http://www.icongames.com.br/pracer/

A produtora virtual DukItan Software 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, a qual é a responsável por gerar a imagem contendo as letras a serem utilizadas. Mais informações podem ser vistas no site http://dukitan.wordpress.com e http://f2ibuilder.sourceforge.net

Padrões relacionados

- Singleton (GoF)
- Façade (GoF)
- Adapter (GoF)