Desenvolvimento de Jogos de Computadores Usando Software Livre

RICARDO DA SILVEIRA COELHO
VLADIMIR OLIVEIRA DI IORIO

DPI – Departamento de Informática
UFV – Universidade Federal de Viçosa – MG
ricardoscoelho@ig.com.br , vladimir@dpi.ufv.br

Resumo: Este artigo apresenta uma breve descrição sobre desenvolvimento de jogos de computadores e discute a utilização de ferramentas de código livre para simplificar o seu desenvolvimento. Uma dessas ferramentas, a **ClanLib**, é amplamente discutida e um exemplo de criação da base de um jogo de ação é apresentado como estudo de caso do uso dessa biblioteca.

Palavras-chave: Desenvolvimento de Jogos, ClanLib, C++, Software Livre.

Development of Computer Games Using Open Source Software

Abstract: This paper presents a brief description of the development of computer games, and analyses the use of open source tools as an option to simplify it. One of these tools, the ClanLib library, is deeply discussed throughout the paper and an example of its use is shown, demonstrating the creation of an action game's base.

Keywords: Game Development, ClanLib, C++, Open Source Software.

(Received October 22, 2004 / Accepted April 13, 2005)

1. Introdução

Com a popularização dos jogos de computadores, tem sido crescente o número de pessoas que buscam ferramentas que simplifiquem seu desenvolvimento.

As diversas linguagens existentes, como Assembly [5], C++ [3], JAVA [4], entre outras, oferecem recursos para desenvolvimento de jogos. Entretanto, a incorporação de ferramentas a essas linguagens facilita o desenvolvimento, uma vez que oferecem funcionalidades que possibilitam um maior nível de abstração e, principalmente, permitem a criação de códigos portáveis.

Dentre essas ferramentas, destacam-se as de código livre, como **OpenGL** (Open Graphic Library) e **SDL** (Simple DirectMedia Layer), cujas descrições são apresentadas na **Seção 2**.

Oferecendo recursos com um escopo delimitado, tais ferramentas são, muitas vezes, usadas em conjunto para a criação de jogos, o que as tornam complementares entre si.

Este trabalho discute a biblioteca **ClanLib,** que foi escolhida para aparecer como destaque nesse artigo devido ao fato de que, incorporando as bibliotecas citadas e algumas outras, apresenta os recursos necessários para a criação de um jogo completo, construindo uma interface de nível mais alto para o acesso às funções implementadas por essas bibliotecas.

Dessa forma, para criar um jogo que utiliza recursos gráficos e de som, o desenvolvedor não precisará chamar diretamente funções da OpenGL, para exibição de gráficos e nem de outra biblioteca de recursos de som. Bastará apenas utilizar as funções predefinidas da ClanLib que encapsulam essas chamadas e os recursos serão disponibilizados. Um

detalhamento das funções oferecidas pela ClanLib é apresentado na **Seção 3**.

A aplicação das principais funções da biblioteca é demonstrada na **Seção 4**, que apresenta um breve exemplo, demonstrando os principais passos para o desenvolvimento da base de um jogo de ação em 2D totalmente feito com a ClanLib.

Com este exemplo, será possível visualizar que um jogo completo pode ser desenvolvido com a ClanLib, sem a necessidade de fazer chamadas a funções de diferentes bibliotecas.

2. Ferramenta de Desenvolvimento de Jogos

Os jogos de computadores compreendem desde jogos simples, como o **Tetris** [16], até jogos complexos que exigem aplicações de técnicas de inteligência artificial, como o **Doom II** [9].

Apesar de existirem diversas ferramentas que facilitam o desenvolvimento de tais jogos, a escolha da linguagem de programação a ser usada é fundamental [1], [6]. O desenvolvedor deve atentar para fatores como recursos oferecidos, nível de fluência que possui na linguagem, facilidade de uso e também, quais linguagens são suportadas pela ferramenta escolhida.

Abaixo é apresentada uma breve descrição de algumas ferramentas de código livre existentes, bem como as linguagens suportadas por cada uma delas:

- **OpenGL** [15]: **Open Graphic Library** é uma biblioteca de rotinas gráficas de modelagem, manipulação de objetos e exibição tridimensional que permite a criação de aplicações que usam computação gráfica. Seus recursos permitem criar objetos gráficos, bem como tratar animações, imagens e texturas. Suporta nativamente as seguintes linguagens de programação: Ada, C, C++, Fortran, Python, Perl e Java. É licenciada como Free Software License B [14];
- **SDL** [13]: **Simple DirectMedia Layer** é uma biblioteca multiplataforma criada para prover acesso a dispositivos de áudio, teclado, mouse, joystick e vídeo. Foi desenvolvida na linguagem C, mas suporta também nativamente C++ e, através de ligações (bindings), outras linguagens como Ada, Eiffel, Java, Lua, ML, Perl, PHP, Pike, Python, e Ruby. É licenciada como Lesser General Public Licence [10];
- **ClanLib** [11]: será amplamente discutida na **Seção 3**. Desenvolvida por [11], é um projeto de código aberto que suporta a linguagem C++. É licenciada como Lesser General Public Licence [10];
- **Outras Ferramentas**: existem ainda diversas outras ferramentas que auxiliam a criação de jogos, como a biblioteca **HERMES** [7], para conversão de imagens; a biblioteca **OpenAL** - **Open Audio Library** [8], para manipulação de dispositivos de áudio e a biblioteca **ZLIB** [17], para compressão de dados.

3. Biblioteca ClanLib

ClanLib é uma biblioteca multiplataforma de código livre desenvolvida em C++, para o desenvolvimento de jogos.

O objetivo principal da biblioteca é prover um nível de abstração que permita que seus usuários criem jogos sem ter que se preocupar com detalhes de baixo nível, como mixagem de som, leitura de arquivos de imagem, etc.

Sua natureza orientada a objetos permite ao usuário programar em alto ou baixo nível, eliminando redundância de código e ainda permitindo a programação de funcionalidades avançadas.

Dessa forma, as classes da biblioteca disponibilizam interfaces simples que podem ser customizadas e expandidas. Isso possibilita que o código do jogo seja claro e simples, mesmo se houver utilização de recursos avançados.

Como exemplo de sua facilidade de uso, pode-se citar a reprodução de som. No código abaixo, simplesmente criando um objeto da classe **CL_SoundBuffer** (linha 1) e chamando uma função dessa classe (linha 2), é possível reproduzir um arquivo de som:

```
1  CL_SoundBuffer som("TIRO.wav");
2  som.play();
```

Estando disponível atualmente na versão 0.7, a ClanLib oferece diversas classes agrupadas por funcionalidade. Abaixo são demonstrados os principais grupos e suas classes e características associadas.

- **ClanLib Application**: todo e qualquer jogo criado com a ClanLib precisará utilizar a classe **CL_ClanApplication**, que é responsável por inicializar a biblioteca e a aplicação. Dessa forma, os jogos criados com a ClanLib devem ser derivados dessa classe (linha 1) e redefinir sua função `main` (linha 3), como no exemplo a seguir:

```
1  class MeuJogo : public CL_ClanApplication{
2  public:
3  virtual int main(int argc, char **argv){
4    CL_SetupCore::init();
5    // Código do jogo
6    CL_SetupCore::deinit();
7    return 0;}
8  } app;
```

Na Linha 8, `app` já é uma instância da classe `MeuJogo` e as funções `init()` (linha 4) e `deinit()` (linha 6) permitem inicializar e finalizar respectivamente os componentes da biblioteca. Isso foi feito no exemplo anterior com a classe `CL_SetupCore`, que inicializou as classes **Core**, responsáveis pela entrada/saída.

Todos os grupos de classes na ClanLib possuem uma classe respectiva para incialização de seus componentes, nomeada da seguinte forma: **CL_Setup<NomeGrupo>**. Um exemplo é a interface gráfica do usuário (GUI), cuja classe de inicialização é **CL_SetupGUI**.

- **ClanLib Display**: as classes desse grupo são necessárias para utilização de recursos gráficos em um jogo. Seu objetivo é oferecer um nível de abstração suficiente para que não haja necessidade de preocupação com os detalhes dos dispositivos que estejam sendo utilizados. Para oferecer tal abstração, a ClanLib utiliza as bibliotecas **OpenGL** e **SDL**.

A principal classe desse grupo é **CL_DisplayWindow**. Essa classe permite a criação de uma janela, que será necessária em todo o jogo para possibilitar a exibição de objetos gráficos na tela.

O código de criação de uma janela utilizando-se a ClanLib poderia ser simplesmente:

```
1    CL_DisplayWindow window("Jogo ", 640, 480);
```

É também no grupo ClanLib Display que existem as classes que permitem a manipulação de dispositivos de entrada. Uma delas é **CL_InputDevice**, que é uma abstração de todos os dispositivos de entrada. O exemplo abaixo mostra a criação de objetos representando alguns desses dispositivos:

```
1  CL_InputContext *ic = window.get_ic();
2  CL_InputDevice keybd = ic->get_keyboard();
3  CL_InputDevice joyst = ic->get_joystick();
```

Ainda dentro do grupo ClanLib Display, encontra-se **CL_InputEvent**, que, em conjunto com **CL_InputDevice**, permite capturar e tratar os diversos sinais emitidos pelos dispositivos de entrada.

Outra classe não menos importante desse grupo é **CL_Surface**. Essa classe permite desenhar os objetos gráficos na tela, como no exemplo abaixo, onde é criado um objeto a partir de um arquivo de imagem (linha 1) e desenhado nas coordenadas informadas (linha 2);

```
1  CL_Surface desenho ("Tiro.png");
2  desenho.draw(10, 10);
```

Existem ainda diversas outras classes em ClanLib Display, como **CL_Sprite**, que permite a animação de imagens e **CL_Font,** que permite a manipulação de fontes de textos na aplicação.

- **ClanLib Sound**: esse grupo de classes permite a utilização de recursos de som.

Para reproduzir um arquivo de som, o usuário precisará de apenas duas classes: **CL_SoundOutput** (linha 1), que cria a interface para o dispositivo de som e **CL_SoundBuffer** (linha 2), que cria um objeto a partir de um arquivo de som, conforme exemplo abaixo:

```
1  CL_SoundOutput output();
2  CL_SoundBuffer som("EXPLOSAO.wav");
3  som.play();
```

- **ClanLib Input:** esse grupo oferece classes, como **CL_Keyboard** e **CL_Mouse**, para manipulação dos dispositivos de entrada. Isso torna possível processar teclas pressionadas no teclado e cliques do mouse, respectivamente.
- **ClanLib Slot:** a principal classe desse grupo é **CL_Slot**, que permite a criação de objetos para capturarem os sinais emitidos pelos dispositivos de entrada.
- **ClanLib GUI**: apesar de não ter aplicação em todos os jogos, as classes desse grupo permitem criar objetos para manipulação da interface gráfica do usuário, como menus, botões, barras de status, caixas de texto, etc.
- **ClanLib Network**: a ClanLib também oferece classes para utilização de recursos de rede, possibilitando a criação de jogos que podem ser compartilhados por vários jogadores através de uma rede de computadores.

Tais classes oferecem facilidades para utilização de soquetes, sessões de rede, replicação de objetos e etc. Dentre as principais, destacam-se:

CL_NetSession: cria um recipiente dos computadores conectados à aplicação;

CL_NetComputer: representa cada computador conectado à sessão;

CL_Socket: encapsula funções do sistema de soquetes em uma plataforma independente de versão;

CL_EventTrigger e **CL_EventListener:** permitem a criação de objetos para esperar e manipular dados recebidos via soquete.

4. Criação da base de Jogo com a ClanLib

Para demonstrar a aplicação de algumas das classes da biblioteca, a codificação da base de um jogo de ação em C++ será seguida passo a passo.

Trata-se de um jogo 2D, denominado **Fireball** [2], onde o jogador controla um tanque de guerra que deve atirar em meteoros caindo e destruí-los.

Neste breve exemplo, será apresentada apenas a aplicação das classes da biblioteca e não a lógica do jogo em si. O código completo, bem como os executáveis do jogo podem ser obtidos em [2] ou no próprio site da ClanLib [11].

Este exemplo assume que o leitor tenha conhecimentos de C++ e que a ClanLib já esteja instalada e configurada. Os procedimentos de instalação da biblioteca para o Microsoft Visual C++ 6.0, podem ser obtidos em [2], e para outras plataformas em [11].

Para facilitar o entendimento do código, as linhas serão numeradas e referenciadas durante o texto.

4.1. Definição do Corpo do Jogo

A base do jogo consistirá de apenas uma classe denominada Fireball, apresentada a seguir:

```
1  #include <ClanLib/core.h>
2  #include <ClanLib/display.h>
3  #include <ClanLib/gl.h>
4  #include <ClanLib/application.h>
5  class Fireball: public CL_ClanApplication {
6  public:
7    Fireball(){};
8    ~Fireball(){};
9    int coordX, coordY;
10   void pTecl(const CL_InputEvent &);
11 virtual int main(int, char **){
12 CL_ConsoleWindow console("Fireb Console");
13 console.redirect_stdio();
14  try {
15  CL_SetupCore::init();
16  CL_SetupDisplay::init();
17  CL_SetupGL::init();
18 CL_DisplayWindow window("Jogo", 500, 450);
19  CL_Slot teclaPres =
20  CL_Keyboard::sig_key_down().connect(this,
21  &Fireball::pTecl);

22  //Loop do Jogo
23  CL_SetupGL::deinit();
24  CL_SetupDisplay::deinit();
25  CL_SetupCore::deinit();
26  } catch (CL_Error err) {
27   std::cout<<"Erro:"<<err.message.c_str();
28   console.display_close_message();}
29   return 0;}
30  } game;
31  //Implementação da função pTecl}
```

O código anterior é a base para o jogo, pois deriva a classe `Fireball` (linha 5), redefine a função `main` (linha 11) e já cria uma instância da classe (linha 30). As diretivas `#include` (linhas 1 a 4), incluem os quatro arquivos de cabeçalho obrigatórios em qualquer aplicação ClanLib.

Verifica-se que tanto o construtor quanto o destrutor não possuem código (linhas 7 e 8), pois o objetivo deste exemplo é apenas demonstrar as funções da ClanLib e dessa forma, para diminuir a quantidade de código, não serão utilizadas as facilidades da orientação a objetos.

O código `"int coordX, coordY"` (linha 9) será usado na **Seção 4.3** para desenho dos gráficos. Da mesma forma, a função `pTecl` (linha 10) e o código `"CL_Slot teclaPres..."` (linhas 19 a 21), responsáveis pela captura e processamento de eventos, serão demonstrados na **Seção 4.4**.

Nota-se também a criação do objeto `console`, da classe **CL_ConsoleWindow** (linha 12), que será usado para exibir mensagens de erro na aplicação.

Como a chamada de funções da ClanLib pode gerar exceções, as mesmas devem ser colocadas dentro de um bloco `try/catch` (linha 14). Dentro desse bloco estão as chamadas de funções para inicialização das classes principais da biblioteca (linhas 15 a 17).

O comentário `"//Loop do Jogo"` (linha 22) deverá ser substituído pelo código na próxima seção.

Finalmente observam-se as chamadas a funções de finalização das classes (linhas 23 a 25), onde **`CL_SetupCore`** é a primeira a ser inicializada e a última a ser finalizada.

4.2. O Loop do Jogo

Um loop será necessário para manter o jogo em processamento.

Nas aplicações feitas com a ClanLib, o loop do jogo necessita de três chamadas de funções:

- **CL_Display::flip()**: responsável por atualizar os gráficos na tela;
- **CL_System::sleep(10)**: faz com que o controle da CPU seja devolvido ao sistema operacional;
- **CL_System::keep_alive()**: permite que a ClanLib seja atualizada e execute tarefas como gerenciar mensagens, processar as entradas, etc.

Com isso, o loop do jogo terá o seguinte código:

```
1  While(!CL_Keyboard::get_keycode(CL_KEY_ESCAPE))
2  { // Exibição de Gráficos
3  CL_Display::flip();
4  CL_System::sleep(10);
5  CL_System::keep_alive();}
```

A função `get_keycode` (linha 1), da classe **`CL_Keyboard`**, que será melhor detalhada na **Seção 4.4**, foi utilizada para tornar possível a saída do loop. Assim, o código verifica se a tecla `ESCAPE` foi pressionada e sai do loop. Na ClanLib, a constante `CL_KEY_ESCAPE` representa a tecla `ESCAPE`.

O comentário `"//Exibição de Gráficos"` (linha 2) deverá ser substituído pelo código na próxima seção.

4.3. Exibição de Gráficos

O presente exemplo exibe um tanque de guerra na tela e o move horizontalmente.

O arquivo usado para representar o tanque possui formato PNG; no entanto, a ClanLib aceita também outros formatos, como JPG, TGA, etc.

Segue o código para exibição do tanque na tela.

```
1  CL_Display::clear(CL_Color(0, 0, 0));
2  CL_Surface tanque("TANQUE.png");
3  tanque.draw(coordX,coordY);
```

Onde a linha 1 chama a função `clear` para limpar a tela, utilizando como parâmetro a cor preta, que é definida em `CL_Color` como `"0,0,0"`.

Um objeto tanque foi criado a partir da classe `CL_Surface`**,** informando como parâmetro, o caminho no disco do arquivo de imagem (linha 2). Esse código poderia também ser colocado fora do loop, pois é necessário carregar a figura apenas uma vez.

Para exibir o tanque na tela, basta chamar a função `draw` da classe `CL_Surface`, informando as coordenadas da janela do jogo (linha 3). As coordenadas, nesse caso, são as duas variáveis declaradas na **Seção 4.1**. Dessa forma, para mover o tanque na tela, basta alterar o valor das variáveis `coordX` e/ou `coordY`.

4.4. Processamento de Eventos

Na ClanLib, atividades como o clique do mouse ou o pressionar de uma tecla disparam sinais que podem ser capturados e processados.

Quando uma tecla é pressionada, a ClanLib envia um sinal através de **CL_Keyboard::sig_key_down.** Isso permite a criação de **Slots** para recebimento e processamento do sinal, como no código abaixo:

```
1  CL_Slot teclaPres =
2  CL_Keyboard::sig_key_down().connect(this,
3  &Fireball::pTecl);
```

Esse código, exibido na **Seção 4.1**, cria um objeto `CL_Slot` e o conecta ao evento de pressionar tecla (`sig_key_down`) da função `CL_Keyboard` (linhas 1 a 3). Tal evento será processado pela função `pTecl` (código a seguir) e deve substituir o comentário da linha 31 da **Seção 4.2**.

```
1 void Fireball::pTecl(const CL_InputEvent &tecla)
2 { if (tecla.id == CL_KEY_LEFT){coordX--;}
3 else if(tecla.id==CL_KEY_RIGHT){coordX++;}}
```

No código anterior, caso seja pressionada a tecla `<SETA ESQUERDA>`, a variável `coordX` será diminuída em 1 unidade (linha 2) e, caso seja a tecla `<SETA DIREITA>`, a mesma será aumentada em 1 unidade (linha 3). Isso fará com que o tanque se mova na tela, pois na próxima passagem pelo código do loop,

demonstrado na **Seção 4.2**, o objeto tanque será desenhado na tela com a nova coordenada.

Apesar do código apresentado na **Seção 4** não criar um jogo completo, o mesmo utiliza os principais recursos da ClanLib necessários para tal.

A **Figura 1** a seguir demonstra a tela do jogo Fireball.

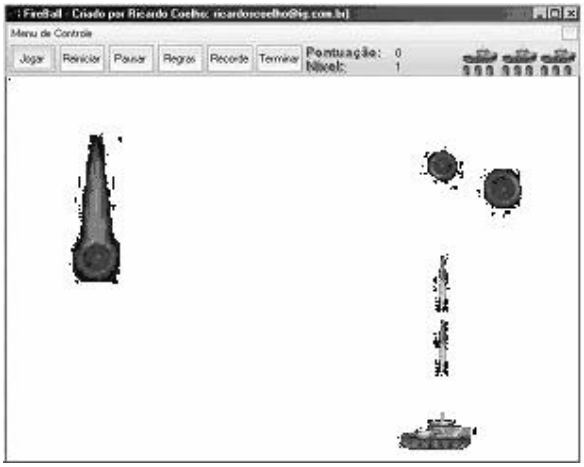

Figura 1: Jogo Fireball completo

5. Conclusão

As bibliotecas de código livre são uma opção real para a criação de jogos, pois além de oferecerem um alto nível de abstração, permitem que a comunidade de desenvolvedores adquira cada vez mais conhecimentos sobre a lógica de sua implementação.

Neste artigo foi discutida a aplicação de uma dessas bibliotecas, a ClanLib, que foi escolhida para aparecer como destaque no texto exatamente porque engloba algumas das principais bibliotecas de código livre para produção de jogos – como OpenGL e SDL – construindo uma interface de nível mais alto para o acesso às funções implementadas por essas bibliotecas

Foi também verificado que as bibliotecas de código livre suportam diversas linguagens de programação e portanto, facilitam o desenvolvimento de códigos portáveis.

A principal dificuldade encontrada no desenvolvimento do jogo proposto é o fato da ClanLib suportar atualmente apenas a linguagem C++.

Como projeto futuro está o estudo mais aprofundado do suporte 3D que a ClanLib oferece, o que possibilitaria a criação de jogos de computadores tridimensionais.

6. Referências Bibliográficas

[1] Brown, Damon. *Choosing a Language*. (recuperado em 17 de novembro de 2004 de http://www.gamedev.net/reference/articles/article713.asp)

[2] R. Coelho. *Fireball Game.* (recuperado em 16 de outubro de 2004 de http://fireball-game.sourceforge.net).

[3] H. M. Deitel, P. J. Deitel. *C++ Como Programar: 3ª Ed.* Bookman, 2001.

[4] H. M. Deitel, P. J. Deitel. *Java Como Programar: 4ª Ed.* Bookman, 2002.

[5] J. Duntemann. *Assembly Language Step-By-Step*. J. Wiley & Sons, 1992.

[6] J. Hattan. *What Language Do I Use?* (recuperado em 17 de novembro de 2004 de http://www.gamedev.net/reference/articles/article895.asp).

[7] Hermes. *Hermes Graphic Library*. (recuperado em outubro de 2004 de http://www.clanlib.org/hermes).

[8] G. Hiebert. *OpenAL Cross-Platform 3D Audio*. (recuperado em 16 de outubro de 2004 de http://www.openal.org).

[9] Id Software. *Doom II* (recuperado em 17 de novembro de 2004 de http://www.idsoftware.com/games/doom/doom2)

[10] LGPL. *Lesser General Public Licence* (recuperado em 17 de novembro de 2004 de http://www.gnu.org/copyleft/lesser.html).

[11] M. Norddahl, K. Gangstoe. *ClanLib Game SDK*. (recuperado em 16 de outubro de 2004 de http://www.clanlib.org).

[12] Ogg Vorbis. *The Ogg Vorbis CODEC Project.* (recuperado em 16 de outubro de 2004 de http://www.xiph.org/ogg/vorbis).

[13] E. Pazera. *Focus On SDL: First Edition.* Muska & Lipman/Premier-Trade, 2002.

[14] SGI. *OpenGL Licensing Programs.* (recuperado em 17 de novembro de 2004 de http://www.sgi.com/products/software/opengl/license.html).

[15] D. Shreiner et al. *OpenGL Programming Guide, Version 1.4: Fourth Edition.* Pearson, 2003.

[16] Tetris Company. *Tetris.* (recuperado em 17 de novembro de 2004 de http://www.tetris.com/index_front.html

[17] ZLib. *ZLib Library*. (recuperado em 16 de outubro de 2004 de http://www.gzip.org/zlib).