Aprendendo a Programar em C com PlayLib

Biblioteca Gráfica e de Funções de Jogo para Aprender C

Versão 1.4.1

Tutorial e Manual do Usuário

Edirlei Soares de Lima
Bruno Feijó
Augusto Baffa

VisionLab/ICAD PUC-Rio
Abril/2014

Dept. de Informática, PUC-Rio Rua Marquês de São Vicente, 225 - Gávea, CEP 22451-900, Rio de Janeiro, RJ
Tel. 3527-1500 r.4534/4308 Fax 3527-1530

APRESENTAÇÃO

PlayLib é uma biblioteca gráfica e de funções básicas para aplicações interativas que tem como objetivo simplificar o processo de desenvolvimento de aplicativos gráficos e auxiliar, de forma lúdica, o aprendizado da linguagem C, além de ensinar técnicas de programação aplicadas na criação de jogos.

A biblioteca consiste de um conjunto de funções para criação/manipulação de formas geométricas 2D, manipulação de imagens, controle de áudio e interação com o usuário via teclado e mouse. Destina-se à linguagem de programação C/C++ e suas funcionalidade gráficas são baseadas na API gráfica OpenGL.

PlayLib foi projetada para ser muito leve e fácil de usar, que tanto serve para suportar atividades em um curso de introdução à programação como para construir programas mais elaborados. Esta biblioteca possibilita o aluno criar jogos 2D, animações e outros aplicativos gráficos.

A facilidade de uso da PlayLib é confirmada pela formação de várias turmas de alunos do 2o. ano do Ensino Médio em programação C, nas quais os alunos criaram pequenos jogos de excelente qualidade.

Com a PlayLib, o aluno estará, na realidade, usando C++ como se fosse C. Isto não comprometerá o ensino da linguagem C, porque o aluno estará usando muito pouco das características do C++, em prol de uma facilidade muito grande para o desenvolvimento de aplicações gráficas e interativas.

Este documento introduz o uso da PlayLib passo-a-passo e também documenta todas as suas funções. Desta maneira, este documento é, ao mesmo tempo, um tutorial e um manual de referência.

Todos os recursos e informações relativos à biblioteca PlayLib, incluindo este manual, podem ser encontrados no site do projeto:

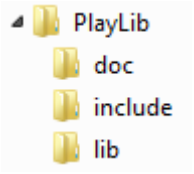
www.inf.puc-rio.br/~playlib

A biblioteca gráfica PlayLib foi criada por Edirlei Soares de Lima e Bruno Feijó, no laboratório ICAD/IGames/VisionLab, do Departamento de Informática da PUC-Rio, em 2012. Aprimoramentos foram acrescentados por Augusto Baffa em 2014.

1 INSTALAÇÃO E CONFIGURAÇÃO

1.1 Arquivos

Decompacte o arquivo PlayLib_v1_4.zip disponível no site do projeto PlayLib e copie as pastas **doc**, **include** e **lib** para um local de sua preferência. Por exemplo, você mesmo pode criar a pasta C:\PlayLib e copiar as pastas para ter a seguinte organização:



Se o site do projeto tiver um instalador automático, você pode optar por usá-lo. Neste caso, o instalador vai criar as pastas acima automaticamente. Além disto, o instalador vai criar um projeto e uma solução (arquivo .sln) que você pode alterar à vontade. Caso contrário, siga as instruções da seção 1.2.

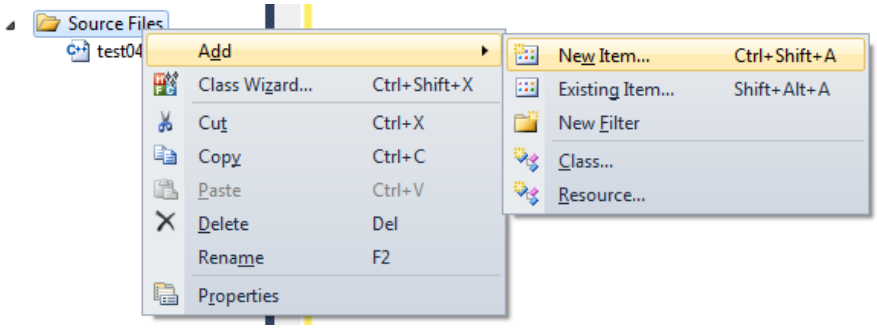
1.2 Configuração

A configuração depende do ambiente de programação que você utiliza. As instruções a seguir são para o Microsoft Visual Studio 2015. Instruções para outros compiladores e ambientes podem ser encontradas no site do projeto PlayLib.

Se os passos descritos nesta seção ainda forem complicados para você, veja uma versão ilustrada no site da PlayLib. Os passos (I) e (II) a seguir são iguais aos já utilizados por você nos programas em C que não usam a PlayLib.

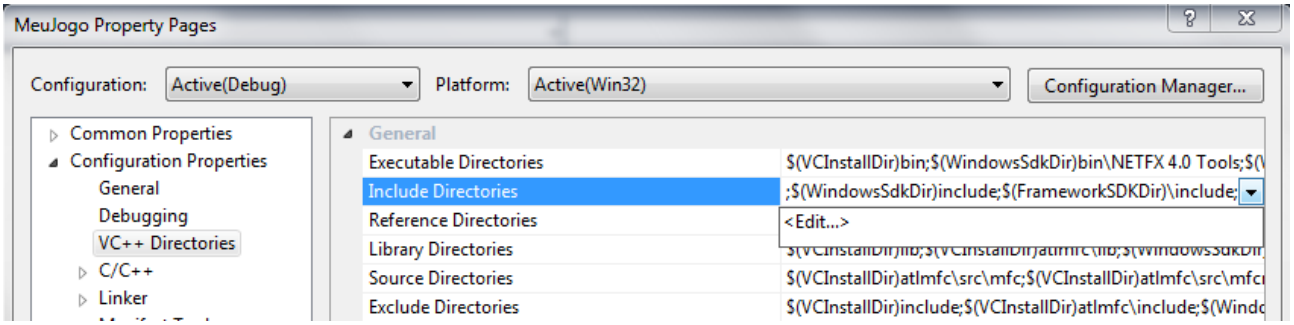
(I) Crie o seu projeto (File > New > Project), para **Win 32 Console Application**, preenchendo os dados do projeto (*e.g.* Name: MeuJogo) e pressionando **Next** para depois marcar "**Empty Project**", desmarcar ***SDL checks***, e pressionar **Finish**.

(II) Crie um módulo **.cpp** clicando com botão direito em Source Files e depois escolhendo **Add** e **New Item…** . Na janela que surge, escolha **Code** e **C++ File (.cpp)** e entre com o nome no campo **Name:** (*e.g.* MeuJogo).



(II) Crie as referências para **include** e **lib** e forneça o arquivo **PlayLib.lib** como dependência, fazendo o seguinte:

Project > *nome_do_projeto* Properties
Configuration Properties > **VC++ Directories**
Clique em **Include Directories**, clique no botão seta e depois em <Edit…>:

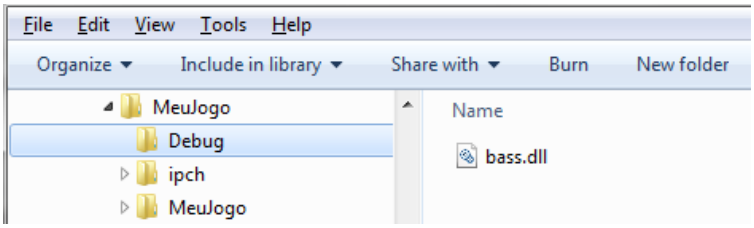


Na janela que se abre, clique no botão New Line  e digite (ou busque) nome do diretório), *e.g.* : c:\PlayLib\include

Faça o mesmo para **Library Directories** > Edit…, fornecendo o caminho para o diretório lib, *e.g.* : c:\ PlayLib \lib

Agora, para definir dependências, faça o mesmo para **Linker** > **Input** > Additional Dependencies > Edit…, digitando **PlayLib.lib** na janela que se abre.

(III) Copie o arquivo **bass.dll** (que está na pasta lib) para a pasta onde estará o arquivo **.exe**. No modo Debug, o arquivo .exe de seu programa será colocado na pasta Debug. Portanto, crie a pasta Debug e copie o arquivo bass.dll para dentro dela (atenção: é copiar e não mover; *i.e.* deixe a bass.dll original onde estava). BASS é uma biblioteca de áudio (gratuita, se usada para fins não-comerciais) e disponível em uma forma bem compacta e leve através do arquivo bass.dll. Seu Windows Explorer ficará como se segue:



(IV) Copie as DLLs para 32-bits e para 64-bits, da seguinte maneira:

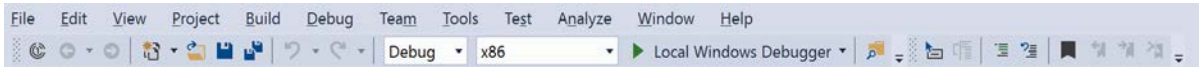
Vá à pasta PlayLib\lib\System32 e copie o arquivo **msvcr100d.dll** para C:\Windows**System32**.

Depois vá à pasta PlayLib/lib/SysWOW64 e copie o arquivo **msvcr100d.dll** para C:\Windows\ **SysWOW64**.

Apesar de terem o mesmo nome, estas DLLs são totalmente diferentes. A razão para este passo IV é porque queremos ter a mesma PlayLib rodando em versões diferentes do Visual Studio. Numa futura versão, vamos otimizar e simplificar esta compatibilidade.

Você precisa ser o dono de sua máquina (i.e. ter os privilégios de administrador) para gravar arquivos nestas duas pastas do sistema Windows.

Normalmente você constrói e testa o seu programa (*i.e.* o seu “projeto” ou “solução”) para console (Win32 Console Application) numa configuração de depuração (**Debug**) e para arquitetura 32-bit (Solution Platform = **x86**), mesmo que sua máquina seja 64-bits. Veja a figura abaixo:



Neste caso, a execução do programa (arquivo .exe) sempre precisa do Visual Studio para resolver todas as referências. Se você quiser gerar um arquivo executável (.exe) otimizado e que possa

ser rodado em qualquer outro computador (com Windows) de maneira independente do Visual Studio, você deve fazer um **DEPLOYMENT** (i.e. ao invés de **Debug**, você usa **Release**). Veja mais instruções no site da PlayLib.

2 ESTRUTURA DOS PROGRAMAS

Aplicativos gráficos e jogos usam orientação a objetos, que é um paradigma não tratado por cursos de introdução à programação baseados em C. Aliás C, não é uma linguagem orientada a objetos, como são C++, C# e java. Entretanto, o uso da PlayLib requer o uso de funções pertencentes a objetos. Isto não traz grandes dificuldades e diferenças para o iniciante na programação em C, mas requer uma rápida explicação sobre objetos – o que pode ser encontrada na seção 2.1.

2.1 Funções da PlayLib

Objetos são entidades mais abrangentes do que as variáveis e as estruturas (struct) que você já usou na linguagem C. Objetos podem ser vistos como entidades concretas criadas a partir de classes genéricas, num processo que chamamos de instanciação. Em outras palavras, um objeto concreto é uma instância de alguma classe. Por exemplo, o objeto bola1 pode ser uma instância da classe círculo e você é uma instância única e concreta da classe dos humanos. Neste contexto, uma classe é um molde (*template*, em inglês) que define, de maneira genérica, como serão os objetos. Este molde contém propriedades (definidas por variáveis) e comportamentos (definidos por funções) que caracterizam a classe em questão. O raio de um círculo (a variável `float raio`) e a área de um círculo (a função `float calculaArea(float raio)`) são exemplos de propriedade e comportamento inerentes à classe círculo. As funções inerentes a uma classe são chamadas de **funções membro** ou de **métodos**.

Na versão corrente da PlayLib, você usará três classes (Graphics, Image e Audio) e não se preocupará com o processo de criação de objetos. Você apenas vai declarar variáveis que representarão objetos de uma determinada classe. Por exemplo:

```
Graphics graphics;
Image minha_imagem1;
Audio musica1;
```

Como a classe Graphics é usada de uma maneira bastante genérica (criando e manipulando elementos gráficos), sugerimos você dar um nome genérico, *e.g.* `Graphics graphics;` ou `Graphics grafico;`. Nos exemplos deste manual, usamos o nome em inglês (`graphics`) para declarar um objeto tipo Graphics.

Uma vez tendo um objeto, podemos invocar uma de suas funções membro através do operador ponto. Por exemplo, para o objeto `graphics`, podemos invocar a função que desenha uma linha saindo da posição (100,100) e indo até a posição (200,100) através do seguinte comando

```
graphics.DrawLine2D(100,100,200,100);
```

Na PARTE 2 deste documento estão especificadas todas as funções da classe Graphics.

Na versão corrente da PlayLib, você não usará as propriedades dos objetos, *i.e.*: você apenas usará as funções membro.

2.2 Estrutura Geral

Você provavelmente foi instruído a sempre criar programas divididos em módulos (onde cada módulo está em um arquivo em separado). Entretanto, para usar diretamente a biblioteca PlayLib, sem precisar ter conhecimentos de orientação a objetos maiores do que a breve introdução da

Seção 2.1, você deverá construir programas com um único módulo contendo a função main e as demais funções auxiliares.

A estrutura básica de um programa gráfico usando a PlayLib é a seguinte:

```
#include "Graphics.h"

Graphics graphics;

void MainLoop()
{


}

int main(void)
{
  graphics.CreateMainWindow(800, 600, "Teste");
  graphics.SetMainLoop(MainLoop);
  graphics.StartMainLoop();
  return 0;
}
```

Inclusão da biblioteca gráfica e de imagem da PlayLib

Instância de um objeto do tipo **Graphics** que permite o uso das funções gráficas.

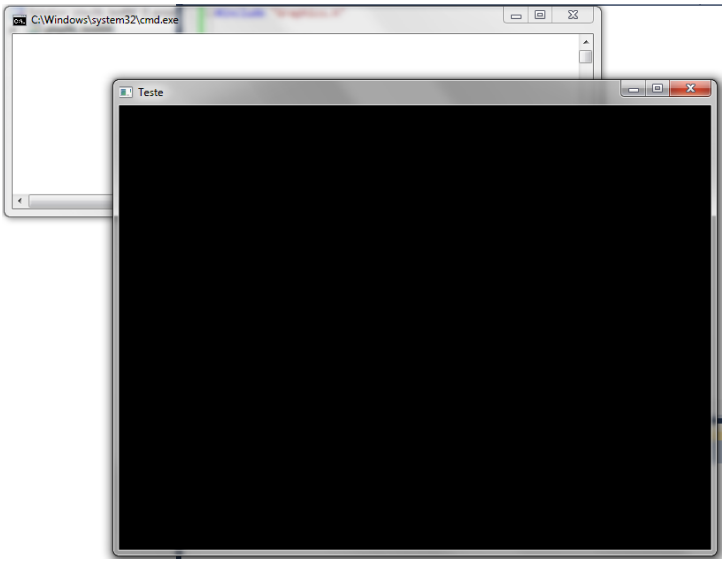
Este é o corpo da função que fica sendo executado repetidamente. Coloque aqui as funções gráficas e/ou as chamadas às suas funções auxiliares (que também podem conter funções gráficas).

Cria uma **janela** de tamanho 800x600 com o título "Teste"

Define a função **MainLoop** como o **Loop Principal** do programa, *i.e.* define a função que a PlayLib ficará executando repetidamente.

Inicia o **Loop principal** do programa

Se o programa acima for executado, você obterá apenas uma janela de 800x600, com fundo preto:



Na realidade são duas janelas: a janela usual de comando e a janela gráfica (também chamada de **canvas**). Atenção: se você fechar apenas a janela gráfica e rodar novamente o programa no Visual Studio, você obterá uma mensagem de erro de Debug. … Portanto, **sempre feche as duas janelas!**

Para entender melhor a estrutura de um programa usando PlayLib, rode o seguinte programa que desenha um círculo vermelho numa janela de fundo branco (neste programa, os comentários explicam o que cada função gráfica faz):

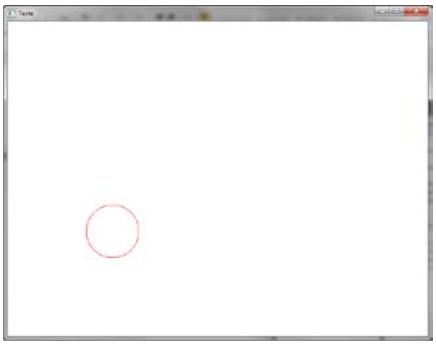
```
#include "Graphics.h"

Graphics graphics;

void MainLoop()
{
   graphics.SetColor(255,0,0); // define vermelho como a cor dos objetos a serem desenhados
   graphics.DrawCircle2D(200,200,50);    // desenha circulo de raio 50 no ponto (200,200)
}

int main(void)
{
  graphics.CreateMainWindow(800, 600, "Teste");
  graphics.SetBackgroundColor(255, 255, 255); // define branco como cor de fundo
  graphics.SetMainLoop(MainLoop);
  graphics.StartMainLoop();
  return 0;
}
```

A janela obtida é a seguinte:



O Loop Principal fica desenhando a figura repetidamente (mas o seu olho não percebe isto). Se você encontrasse uma maneira de atualizar o valor do ponto central do círculo a cada execução da função MainLoop, você estaria criando uma animação do círculo !!! Entretanto, você nem sabe (por enquanto) qual é o intervalo de tempo que a função MainLoop fica se repetindo. Você aprenderá a fazer animações mais adiante.

Faça o exercício a seguir e depois continue lendo as seções que apresentam mais detalhes sobre a estrutura e o funcionamento dos programas que usam a PlayLib.

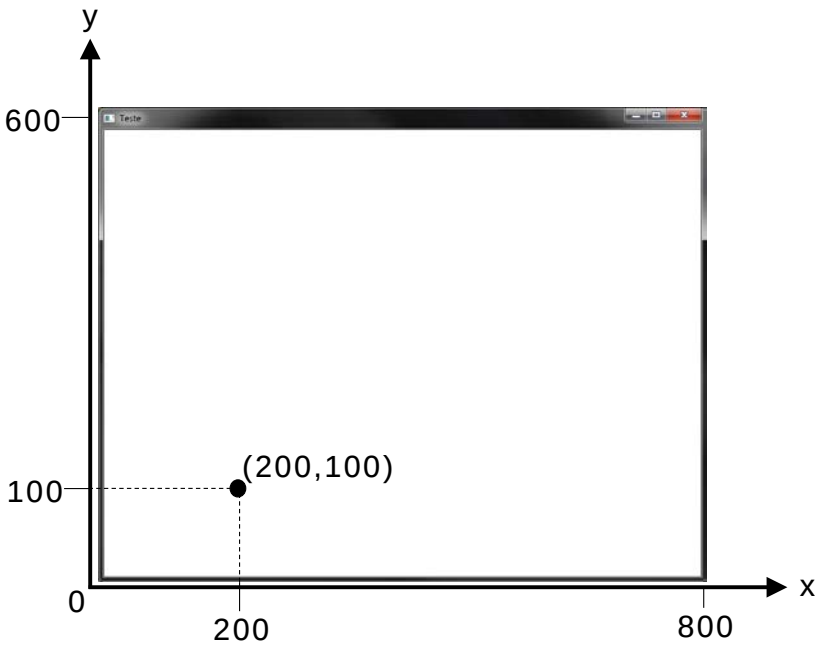
EXERCÍCIO E2.2-1
Troque a cor dos objetos para azul com SetColor(0,0,255) e desenhe mais de um círculo em posições diferentes. Também desenhe alguns retângulos azuis que vão do ponto (x1,y1) ao ponto (x2,y2), utilizando a função DrawRectangle2D(x1,y1,x2,y2). Por fim acrescente um círculo vermelho, chamando novamente a SetColor seguida de uma DrawCircle2D.

Considerações: [1] a cor é definida pelo modelo RGB (Red-Green-Blue), onde cada componente de cor varia de 0 a 255. O máximo das 3 cores (255,255,255) cria uma luz branca. Tente (0,0,0) e veja o que acontece. [2] todos os valores de R,G,B,x1,y1,x2 e y2 são inteiros.

Sempre que possível documente o seu programa em inglês (nomes de variáveis e comentários), porque códigos devem ser entendidos por qualquer programador (e inglês é a *lingua franca* em programação). Isto é especialmente válido no mundo dos games. Nos exemplos de código do presente documento, algumas vezes escrevemos comentários em português para facilitar o entendimento dos conceitos.

2.3 A Janela de Desenho (Canvas)

A janela de desenho (denominada “canvas”) contém pixels que são localizados por suas coordenadas de tela. No programa da seção 2.2, você definiu o canvas com a função `CreateMainWindow(800, 600, "Teste")`. As coordenadas de tela são definidas no sistema de coordenadas cartesiano, onde o canto inferior esquerdo da tela do programa é definido na coordenada X=0 e Y=0 e qualquer ponto é representado por números inteiros de pixels. Esse sistema de coordenadas é ilustrado na figura abaixo:



O protótipo da função que define a janela de desenho é o seguinte:

```
void CreateMainWindow(int sizeX, int sizeY, char * title);
```

O título da janela é exibido no canto esquerdo da barra da janela.

2.4 Modelo de Cor RGB

O modelo de cor RGB representa as cores que são possíveis de serem produzidas no dispositivo gráfico de saída através da combinação das luzes vermelha (Red), verde (Green) e azul (Blue). Há cores do espectro visível que não podem ser representadas pela adição destas três cores (entretanto, você dificilmente notará esta limitação).

Convencionamos indicar a intensidade de cada componente de cor por um número de 0 a 255. Desta maneira, a combinação destas três cores com o valor máximo de 255 gera a luz branca. Os valores (0,0,0) indicam ausência dos 3 componentes, ou seja: o preto. Qualquer combinação com igual proporção dos três componentes geram várias intensidades de cinza, *e.g.* (128,128,128) é um cinza médio. As outras combinações típicas são:

R + G = amarelo
R + B = magenta
G + B = ciano

2.5 Callbacks

Podemos passar uma função como argumento de uma outra função. Ao fazer isto, estamos, na realidade, passando uma referência a uma função e não uma função em si. A função SetMainLoop, usada no programa da seção 2.2, tem apenas um argumento que é uma função com o seguinte protótipo:

```
void func(void)
```

Na seção 2.2, escolhemos chamar esta função de MainLoop:

```
void MainLoop(void)
```

Quando passamos uma referência a uma função como argumento de uma outra função, dizemos que uma "chamada de volta" (*call back*) foi feita. Por esta razão chamamos estas funções de ***callbacks***. É muito útil termos a total liberdade de definir as funções que são argumentos de outras. No caso da seção 2.2, a função SetMainLoop determina que a função MainLoop é a função que a PlayLib ficará executando repetidamente. Ficamos com a liberdade de dar o nome que quisermos e de escrevermos o que esta função deve fazer.

No exemplo da SetMainLoop, a *callback* não tem argumentos e nem retorna valor. Mas há callbacks que têm argumentos. Por exemplo, a PlayLib tem a função SetKeyboardInput que define a função que lidará com eventos vindo do teclado. A SetKeyboardInput tem, como único argumento, uma função que, por sua vez, tem três argumentos: o primeiro é uma variável int contendo o código da tecla que foi pressionada, o segundo é uma variável int que representa o código do estado da tecla (pressionada ou liberada) e os dois outros são variáveis int que contêm as posições x e y do mouse no momento em que a tecla foi pressionada. O protótipo desta função é o seguinte:

```
void func(int key, int state, int x, int y)
```

Você deve fornecer esta função e determinar o que deve ser feito com os valores destes 3 argumentos. O nome da função e dos argumentos ficam a seu encargo. Por exemplo, para `graphics.SetKeyboardInput(meuTeclado)` podemos definir a função `void meuTeclado(int tecla, int estado, int x, int y)`.

Execute o programa abaixo e verifique que toda a vez que o usuário pressiona a tecla d do teclado, o círculo avança 50 unidades (pixels) na horizontal e na vertical. Neste programa, precisamos definir uma variável global chamada `posicao`, para que o seu valor seja acessado pela função MainLoop toda vez que a MailLoop é executada.

```
#include "Graphics.h"

Graphics graphics;
int posicao = 0;

void MainLoop(void)
{
   graphics.SetColor(255,0,0);
   graphics.DrawCircle2D(posicao,posicao,50);
}

void meuTeclado(int key, int state, int x, int y)
{
   if ((key == 'd')&&(state == KEY_STATE_DOWN))
      posicao += 50;
```

```
}

int main(void)
{
  graphics.CreateMainWindow(800, 600, "Teste");
  graphics.SetBackgroundColor(255, 255, 255);
  graphics.SetKeyboardInput(meuTeclado);
  graphics.SetMainLoop(MainLoop);
  graphics.StartMainLoop();
  return 0;
}
```

EXERCÍCIO E2.5-1
Altere o programa acima de maneira que a tecla 'd' continue provocando movimentos para a frente, mas que, se o usuário pressionar a tecla 'a', o círculo se moverá para trás. Mude também o nome da função `meuTeclado` para `KeyboardInput`.

2.6 Interações via Teclado e Mouse

Na seção anterior (2.5), usamos *callbacks* para trabalhar com o teclado. Outras funções para teclado podem ser encontradas na PARTE 2 deste documento (Seção 8) e os seus usos podem ser testados em pequenos programas que você mesmo pode criar.

2.7 Animações

A *callback* MainLoop dos programas implementados nas seções anteriores fica sendo repetidamente chamada pela PlayLib. Na primeira chamada da MainLoop, os comandos de desenho que foram executados dentro dela são enviados para a placa de vídeo do computador e a placa começa a renderizar o quadro (*frame*). Quando a renderização termina, o *frame* é exibido na tela e a função MainLoop é chamada novamente. Nesta segunda chamada, se algo mudar na MainLoop, o novo *frame* será diferente. No primeiro programa deste documento (seção 2.2), os *frames* ficam se repetindo sem qualquer mudança.

Execute o programa a seguir e verifique que o círculo automaticamente se desloca e para quando ultrapassa o ponto (500,200).

```
#include "Graphics.h"

Graphics graphics;

void MainLoop(void)
{
   static int posx = 0; // posicao do circulo no eixo x
   if (posx <= 500)
      posx += 2;
   graphics.SetColor(255,0,0);
   graphics.DrawCircle2D(posx,200,50);
}

int main(void)
{
  graphics.CreateMainWindow(800, 600, "Teste");
  graphics.SetBackgroundColor(255, 255, 255);
  graphics.SetMainLoop(MainLoop);
  graphics.StartMainLoop();
```

```
  return 0;
}
```

A grande novidade neste programa é a definição da variável `posx` como sendo estática (***static***). Uma variável local declarada como static significa que o valor atribuído a ela numa execução da função é recuperado quando a função for novamente executada. Se você retirar o atributo *static* da variável `posx`, o círculo ficará parado na posição do primeiro desenho que é definido por `DrawCircle2D(posx,200,50)` com `posx` = 2., visto que o valor de `posx` será sempre calculado como 2.

A função MainLoop só é chamada novamente quando a exibição do *frame* termina. Quanto tempo isto leva ? A resposta é:

depende do computador que você estiver usando!

Então a velocidade do círculo depende da máquina ? A resposta é: *sim, a velocidade do deslocamento do círculo do programa acima depende do processador central e da placa de vídeo do computador*. Um computador mais rápido calculará e exibirá *frames* numa taxa mais alta do que um computador (ou um smartphone) mais lento. Esta taxa é denominada de **FPS** (*Frames per Second*).

Uma maneira de fazer o movimento ficar independente do computador é definir deslocamentos em função do valor do tempo que se passou desde a geração do último frame. Este valor de tempo é denominado de **Elapsed Time**. A PlayLib disponibiliza a seguinte função que retorna o Elapsed Time:

```
float GetElapsedTime();
```

Desta maneira, calculamos a posição x de um objeto gráfico como sendo:

```
posX = posX + (speed * graphics.GetElapsedTime());
```

onde `speed` é a velocidade. Isto é, estamos usando a equação cinemática de deslocamento:

$$\Delta x = v\Delta t$$

onde v é a velocidade e Δt é o intervalo de tempo decorrido.

O programa abaixo desloca o círculo de forma independente da taxa de frames do seu computador:

```
#include "Graphics.h"

Graphics graphics;

void MainLoop(void)
{
   static float posx = 0; // posicao do circulo no eixo x
   float posy = 200;
   float speed = 150;

   if (posx <= 500)
      posx += speed * graphics.GetElapsedTime();
   graphics.SetColor(255,0,0);
   graphics.DrawCircle2D(posx,posy,50);
}

int main(void)
{
  graphics.CreateMainWindow(800, 600, "Teste");
```

```
  graphics.SetBackgroundColor(255, 255, 255);
  graphics.SetMainLoop(MainLoop);
  graphics.StartMainLoop();
  return 0;
}
```

A independência da taxa de frames do computador é essencial em jogos e simulações.

Observe que o programa acima poderia ser implementado com as variáveis posx e posy sendo variáveis globais. Entretanto, **devemos usar variáveis globais apenas quando não houver outra alternativa**. As variáveis globais tornam os códigos difíceis de serem entendidos e documentados. Nos programas em C deste documento, definimos variáveis globais quando usamos funções de interação (como a meuTeclado da seção 2.5), porque precisamos que outras funções tenham acesso a estas variáveis.

2.8 Imagens

Para desenhar uma imagem na tela são necessários os seguintes passos:

(1) Criar uma variável (*i.e.* objeto) do tipo Image.

```
Image nome_da_variável_imagem;
```

OBS1: Sempre declare as variáveis Image como **variáveis globais**.

Exemplo:

```
#include "Graphics.h"

Graphics graphics;
Image minha_imagem1;
Image minha_imagem2;

int main(void)
{
...
```

OBS2: As variáveis globais Image devem ser declaradas no início do programa, antes e fora da função principal ou outras funções.

(2) Carregar a imagem do arquivo usando a função LoadPNGImage do objeto tipo Image.

```
nome_da_variável_imagem.LoadPNGImage("path e nome do arquivo");
```

Exemplo:

```
int main(void)
{
...
  minha_imagem1.LoadPNGImage("C:\\Imagens\\Mario.png");
...
}
```

OBS1: Cada imagem deve ser carregada **apenas uma vez**. Por isso, nunca carregue a imagem diretamente de dentro do Loop Principal.

OBS2: observe a necessidade de duas barras invertidas na cadeia de caracteres quando há um *path*.

OBS3: se o aquivo da imagem estiver na mesma pasta do projeto, o *path*. não será necessário, *e.g.*: `minha_imagem1.LoadPNGImage("Mario.png");`.

(3) Desenhar a imagem na tela, usando a função DrawImage2D do objeto tipo Graphics.

```
graphics.DrawImage2D(x, y, width, height, nome_da_variável_imagem);
```

onde (x,y) é a posição da imagem na tela e (width,height) é o tamanho da imagem (em pixels). Uma alternativa é primeiro definir posição e tamanho da imagem com o comando SetPosition e depois desenhar:

```
nome_da_variavel_imagem.SetPosition(x, y, width, height);
graphics.DrawImage2D(nome_da_variável_imagem);
```

Observe que a função DrawImage2D tem dois protótipos. Isto não é permitido na linguagem C. Mas, lembre que você está, na realidade, usando C++ como se fosse C. Isto não compromete o ensino da linguagem C, porque estamos usando muito pouco das características do C++ em prol de uma facilidade muito grande para o desenvolvimento de aplicações gráficas e interativas.

Exemplo:

```
void MainLoop()
{
...
  graphics.DrawImage2D(200, 200, 256, 256, minha_imagem1);
...
}
```

ou

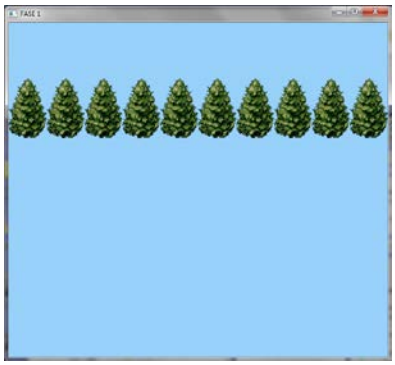
```
void MainLoop()
{
..minha_imagem1.SetPosition(200,200,256,256);
  graphics.DrawImage2D(minha_imagem1);
...
}
```

As seguintes observações gerais devem sempre ser verificadas:

- Somente são aceitas imagens no formato **PNG**. Mas isso não é uma limitação, o formato PNG é um dos melhores formatos para esse tipo de aplicação. A principal vantagem é que ele permite o **uso de transparência** nas imagens.

- Se a sua imagem estiver em outro formado (JPG, GIF, BMP, ...) você deve convertê-la para o formato PNG (e especificando que o *background* é transparente) antes de carregá-la. Você mesmo pode produzir suas imagens usando programas como o Photoshop para salvar imagens PNG com fundo transparente.

O exemplo a seguir prepara um cenário com várias árvores em linha, conforme a seguinte ilustração:



Este cenário é criado a partir de um *tile* PNG de 68x108 pixels (arquivo tree_68x108.png, disponível no site da PlayLib) que foi repetido várias vezes até preencher todo o tamanho X da tela. O tamanho X da tela foi definido como SIZE_X = 680, para conter exatamente 10 tiles. A medida 68x108 pixels não é usual, sendo a preferência por múltiplos de 32 (e, muitas vezes formando um quadrado), e.g.: 32x32, 128x128 e 256x256.

O programa que gera o cenário acima é o seguinte:

```
#include "Graphics.h"

#define SIZE_X 680
#define SIZE_Y 600

Graphics graphics;
Image treeTile;

void MainLoop()
{
   int x, y;
   y = 392;
   for (x = 0; x < SIZE_X; x+=68)
      graphics.DrawImage2D(x, y, 68, 108, treeTile);  // Draw tree tiles
}

int main(void)
{
   graphics.CreateMainWindow(SIZE_X, SIZE_Y, "FASE 1");
   graphics.SetBackgroundColor(152,209,250); // color like sky
   // Load image files
   treeTile.LoadPNGImage("c:\\PlayLib_Resources\\images\\tree_68x108.png");
   // Set up loop
   graphics.SetMainLoop(MainLoop);
   graphics.StartMainLoop();
   return 0;
}
```

A Playlib também permite a utilização de tiras de imagens contendo diversos objetos ou animações de um personagem em uma única imagem. Para isso, você deve utilizar outra versão do comando DrawImage2D:

```
graphics.DrawImage2D(x, y, width, height, crop_x, crop_y, crop_width, crop_height, nome_da_variável_imagem);
```

Neste caso, a imagem será desenhada com o tamanho width x height na posição (x,y) da tela, sendo desenhado somente a região da imagem que se inicia na posição (crop_x, crop_y) e possui tamanho crop_width x crop_height.

Um exemplo de tira de imagens é ilustrado a seguir:

O exemplo a seguir ilustra como uma animação pode ser realizada a partir de uma tira de imagens:

```
#include "Graphics.h"

#define SPRITE_SIZE 128   // tamanho de cada imagem da tira de imagens

Graphics graphics;

Image sprite_sheet;

struct Sprite{  // estrutura para armazenar a posição do recorte de cada frame da animação
    int x;
    int y;
    int width;
    int height;
};

Sprite sprite[4][4];

float player_x = 100;
float player_y = 250;
float player_speed = 80;
int animation_clip = 1;  // índice para controlar a animação
int animation_index = 0; // índice para controlar o frame da animação
float time_next_frame = 0;

bool key_states[256];  // vetor para armazenar as teclas pressionadas

void KeyboardInput(int key, int state, int x, int y)
{
   if (state == KEY_STATE_DOWN)
      key_states[key] = true;
   else if (state == KEY_STATE_UP)
      key_states[key] = false;
}

void MainLoop()
{
   bool next_frame = false;
   time_next_frame = time_next_frame + graphics.GetElapsedTime();

   if (key_states[KEY_RIGHT])
   {
     next_frame = true;
```

```
      player_x += player_speed * graphics.GetElapsedTime();
   }

   if (next_frame)
   {
      if (time_next_frame > 0.1) // controle da velocidade da mudança de frames da animação
      {
         animation_index++;
         time_next_frame = 0;
      }
      if (animation_index > 3)
         animation_index = 0;
   }

   // desenha a imagem na tela
   graphics.DrawImage2D(player_x, player_y, SPRITE_SIZE, SPRITE_SIZE,
                        sprite[animation_clip][animation_index].x,
                        sprite[animation_clip][animation_index].y,
                        sprite[animation_clip][animation_index].width,
                        sprite[animation_clip][animation_index].height,
                        sprite_sheet);
}

void InitSprites()  // função que calcula a posição do recorte e armazena na matriz sprite
{
    int x = 0, y = 0;
    int spritex = 0;
    int spritey = 0;
    for (int w = 0; w < 16; w++)
    {
        sprite[x][y].x = spritex;
        sprite[x][y].y = spritey;
        sprite[x][y].width = SPRITE_SIZE;
        sprite[x][y].height = SPRITE_SIZE;
        spritex += SPRITE_SIZE;
        y++;
        if (spritex >= 512)
        {
            spritex = 0;
            spritey += SPRITE_SIZE;
            x++;
            y = 0;
        }
    }
}

int main(void)
{
    graphics.CreateMainWindow(800, 600, "Exemplo 11 - Sprite Sheet");
    graphics.SetBackgroundColor(152,209,250);
    graphics.SetKeyboardInput(KeyboardInput);

    sprite_sheet.LoadPNGImage("Char_Sprite.png");
    InitSprites();

    graphics.SetMainLoop(MainLoop);
    graphics.StartMainLoop();
    return 0;
}
```

EXERCÍCIO E2.8-1

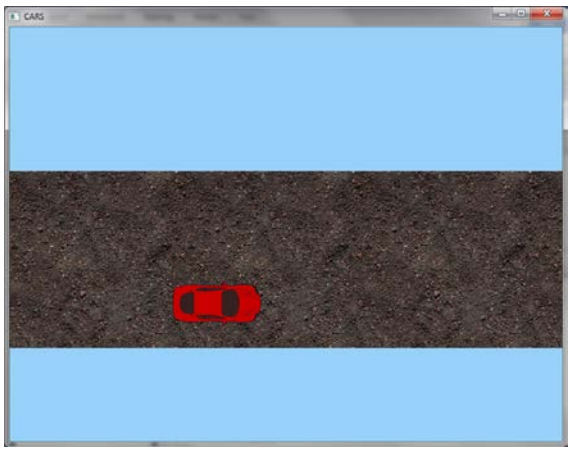
Altere o programa acima de maneira que a tecla da seta direcional da esquerda mova o personagem para a esquerda e a animação correspondendo ao personagem caminhando nessa direção seja executada.

Considerações:
[1] O código da tecla da seta direcional da esquerda é `KEY_RIGHT`.
[1] O índice (`animation_clip`) da animação do personagem andando para a esquerda é 0.

2.9 Áudio

A PlayLib suporta arquivos de áudio no formato **MP3** e **WAV**. Para executar um áudio (música ou efeito sonoro) com a PlayLib, o processo e os cuidados são muito semelhantes ao do caso de imagens. Uma diferença importante é que podemos ordenar que um um áudio comece a tocar (ou parar) em qualquer função (na main, no loop principal, ...).

O exemplo a seguir apresenta as principais funções de áudio e os cuidados mais importantes no uso de áudio. O programa abaixo desenha uma estrada e comanda o movimento de um carro (para frente e para trás) com as setas do teclado. Neste programa, a título de abertura, o som de um motor de carro sendo ligado é executado tão logo o programa é executado. Além disto, o som de um motor em aceleração acompanha os movimentos do carro. A janela tem a seguinte cena:



```
#include "Graphics.h"
#include "Audio.h"

#define SIZE_X 800
#define SIZE_Y 600

Graphics graphics;
Image roadTile;
Image car;

Audio motor;
Audio startEng;

int car_x = 236;   // car X position

void MainLoop()
{
   int x, y;
   int car_y = 136; // car Y position is fixed
   for (x = 0; x < SIZE_X; x+=256)
      graphics.DrawImage2D(x, 136, 256, 256, roadTile);  // Draw road tiles
   graphics.DrawImage2D(car_x, car_y, 128, 128, car);    // Draw car
}
```

```
void KeyboardInput(int key, int state, int x, int y)
{
   if ((key == KEY_LEFT)&&(state == KEY_STATE_DOWN))
   {
      car_x = car_x - 4;
      if (motor.IsPlaying()==false)
      {
         if (startEng.IsPlaying() == true)
            startEng.Stop();
         motor.Play();
      }
   }
   if ((key == KEY_RIGHT)&&(state == KEY_STATE_DOWN))
   {
      car_x = car_x + 4;
      if (motor.IsPlaying()==false)
      {
         if (startEng.IsPlaying() == true)
            startEng.Stop();
         motor.Play();
      }
   }
   if (state == KEY_STATE_UP)
   {
      if (motor.IsPlaying() == true)
         motor.Stop();
   }
}

int main(void)
{
   graphics.CreateMainWindow(SIZE_X, SIZE_Y, "CARS");
   graphics.SetBackgroundColor(152,209,250);
   // Load image files
   roadTile.LoadPNGImage("c:\\PlayLib_Resources\\images\\car_ground.png");
   car.LoadPNGImage("c:\\PlayLib_Resources\\images\\car_right.png");
   // Load Audio files
   motor.LoadAudio("c:\\PlayLib_Resources\\sounds\\car_accelerating.mp3");
   startEng.LoadAudio("c:\\PlayLib_Resources\\sounds\\start_engine.mp3");
   // Play audio that game starts with
   startEng.Play();  // sound of a car engine starting
   // Set up interaction and loop
   graphics.SetKeyboardInput(KeyboardInput);
   graphics.SetMainLoop(MainLoop);
   graphics.StartMainLoop();
   return 0;
}
```

As imagem da estrada (um tile de 256x256) e do carro (128x128) estão disponíveis no site da PlayLib.

Algumas observações sobre o programa acima são muito instrutivas. Por exemplo, você deve notar que ao pressionar a seta para a direita, a posição corrente do carro é atualizada e o som de abertura (um motor sendo ligado) é interrompido e o som de um motor acelerando é tocado. Observe também que tão logo a tecla for liberada há uma ordem de parar o áudio do motor acelerando. Note também que dentro do if de cada tecla pressionada há a repetição do código que controla os áudios startEng e motor. Se, ao invés de ficar repetindo este código, tivéssemos o escrito apenas uma vez no final da função KeyboardInput, teríamos introduzido um erro sutil: qualquer tecla especial (*e.g.* Esc, Ctrl, ...) faria o áudio do motor acelerando ser tocado. Lembre-se de que quando qualquer tecla é pressionada a função KeyboardInput é executada.

3 PRATICANDO COM PlayLib

Se você absorveu o conhecimento das seções anteriores, agora só resta praticar. Primeiro faça alguns exercícios que sejam variantes dos exemplos e exercícios da Seção 2. Você mesmo pode inventar vários destes exercícios. Mas, se preferir, faça os exercícios a seguir:

EXERCÍCIO E3-1
Escreva um programa onde um círculo preenchido com a cor vermelha se desloca na horizontal até bater numa linha vertical e ao mesmo tempo faça com que toda a vez que o usuário pressionar a tecla 'p' o círculo para (se estava se movimentando) ou reinicia o movimento (se estava parado). Implemente os movimentos independentes da taxa de frames do computador.

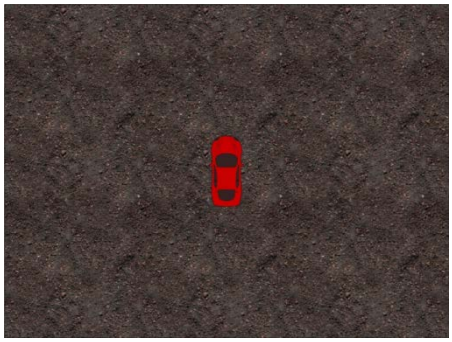
Considerações:
[1] para criar um círculo preenchido com uma cor use `void FillCircle2D(int x, int y, int radius, int segments)`, onde quanto mais segmentos houver, mais perfeitamente redondo fica o círculo (30 é suficiente);
[2] uma linha que começa em (x1,y1) e termina em (x2,y2) é criada com `void DrawLine2D(int x1, int y1, int x2, int y2)`;
[3] na função MainLoop use uma função auxiliar chamada animaBola() e leve a animação do círculo para ela:

```
void MainLoop()
{
   animaBola();
   graphics.SetColor(...);
   graphics.FillCircle2D(...);
}
```

[4] Na animaBola faça um grande if, logo no início, onde a variável global `pause` é testada. Na função KeyboardInput faça com que se a tecla 'p' é pressionada, então: `pause = !pause`. Desta maneira você facilmente fica chaveando a situação.

EXERCÍCIO E3-2
Faça as seguintes alterações no programa da seção 2.9: coloque árvores ao longo da estrada; crie carros que automaticamente passam pela estrada em alta velocidade; e implemente as teclas para cima e para baixo para deslocar o carro principal na vertical (como estivesse se desviando de obstáculos). Depois melhore o seu pequeno jogo redefinindo as teclas: barra de espaço para acelerar e setas para mudar de direção.

Depois estude o código a seguir, onde, com as setas do teclado, você controla um carro sobre um terreno:



As imagens do carro e do terreno estão disponíveis no site da PlayLib, mas você pode usar quaisquer imagens (basta buscá-las na internet).

```
#include "Graphics.h"
#include "Audio.h"

#define SIZE_X 800
#define SIZE_Y 600

Graphics graphics;
Image roadTile;
Image car[4];

Audio motor;
Audio startEng;

int car_dir = 2;   // car direction (initially, 2)
int car_x = 236;   // car position X
int car_y = 136;   // car position Y

void MainLoop()
{
   int x, y;
   for (x = 0; x < SIZE_X; x+=256)
   {
      for (y = 0; y < SIZE_Y; y+=256)
      {
         graphics.DrawImage2D(x, y, 256, 256, roadTile);
      }
   }
   graphics.DrawImage2D(car_x, car_y, 128, 128, car[car_dir]);
}
void playMotorAudio(void)
{
   if (motor.IsPlaying()==false)
   {
      if (startEng.IsPlaying() == true)
         startEng.Stop();
      motor.Play();
   }
}

void KeyboardInput(int key, int state, int x, int y)
{
   int step = 4;
   if ((key == KEY_LEFT)&&(state == KEY_STATE_DOWN))
   {
      car_dir = 1;
      car_x -= step;
      playMotorAudio();
   }
   if ((key == KEY_RIGHT)&&(state == KEY_STATE_DOWN))
   {
      car_dir = 3;
      car_x += step;
      playMotorAudio();
   }
   if ((key == KEY_UP)&&(state == KEY_STATE_DOWN))
   {
      car_dir = 2;
      car_y += step;
      playMotorAudio();
   }
   if ((key == KEY_DOWN)&&(state == KEY_STATE_DOWN))
   {
```

```
      car_dir = 0;
      car_y -= step;
      playMotorAudio();
   }

   if (state == KEY_STATE_UP)
   {
      if (motor.IsPlaying() == true)
         motor.Stop();
   }
}

int main(void)
{
   graphics.CreateMainWindow(SIZE_X, SIZE_Y, "CARROS");
   graphics.SetBackgroundColor(152,209,250);  // sky color
   // Load image files
   roadTile.LoadPNGImage("c:\\PlayLib_Resources\\images\\car_ground.png");
   car[0].LoadPNGImage("c:\\PlayLib_Resources\\images\\car_down.png");
   car[1].LoadPNGImage("c:\\PlayLib_Resources\\images\\car_left.png");
   car[2].LoadPNGImage("c:\\PlayLib_Resources\\images\\car_up.png");
   car[3].LoadPNGImage("c:\\PlayLib_Resources\\images\\car_right.png");
   // Load audio files
   motor.LoadAudio("c:\\PlayLib_Resources\\sounds\\car_accelerating.mp3");
   startEng.LoadAudio("c:\\PlayLib_Resources\\sounds\\start_engine.mp3");
   // Play audio that game starts with
   startEng.Play();  // sound of a car engine starting
   // Set up interaction and loop
   graphics.SetKeyboardInput(KeyboardInput);
   graphics.SetMainLoop(MainLoop);
   graphics.StartMainLoop();
   return 0;
}
```

Como continuação de sua prática, faça pequenos programas que implementem as funções apresentadas na PARTE 2 deste documento.

Finalmente, estude os programas e jogos feitos por alunos no site da PlayLib.

PARTE 2 : FUNÇÕES

4 PRIMITIVAS GEOMÉTRICAS

Ponto

```
void DrawPoint2D(int x, int y);
```

Desenha um ponto na posição (x,y) da tela.
Exemplo: `graphics.DrawPoint2D(200, 200);`

Linha

```
void DrawLine2D(int x1, int y1, int x2, int y2);
```

Desenha uma linha começando no ponto (x1,y1) e terminando no ponto (x2,y2).
Exemplo: `graphics.DrawLine2D(100, 100, 200, 100);`

Círculo

```
void DrawCircle2D(int x, int y, int r);
```

Desenha um círculo com origem no ponto (x,y) e raio igual a r.
Exemplo: `graphics.DrawCircle2D(200, 200, 20);`

Círculo Preenchido

```
void FillCircle2D(int x, int y, int r, int n);
```

Desenha um círculo totalmente preenchido pela cor corrente, com origem no ponto (x,y), de raio r e com n segmentos definindo a borda.
Exemplo: `graphics.FillCircle2D(200, 200, 20, 30);`

Retângulo

```
void DrawRectangle2D(int x1, int y1, int x2, int y2);
```

Desenha um retângulo cuja diagonal é definida pelos pontos (x1,y1) e (x2,y2).
Exemplo: `graphics.DrawRectangle2D(100,100,200,200);`

Retângulo Preenchido

```
void FillRectangle2D(int x1, int y1, int x2, int y2);
```

Desenha um retângulo cuja diagonal é definida pelos pontos (x1,y1) e (x2,y2), totalmente preenchido pela cor corrente.
Exemplo: `graphics.FillRectangle2D(100,100,200,200);`

Triângulo

```
void DrawTriangle2D(int x1, int y1, int x2, int y2, int x3, int y3);
```

Desenha um triângulo definido pelos pontos (x1,y1), (x2,y2) e (x3,y3).
Exemplo: `graphics.DrawTriangle2D(100,100,200,100,150,200);`

Triângulo Preenchido

```
void FillTriangle2D(int x1, int y1, int x2, int y2, int x3, int y3);
```

Desenha um triângulo definido pelos pontos (x1,y1), (x2,y2) e (x3,y3), totalmente preenchido pela cor corrente.
Exemplo: `graphics.FillTriangle2D(100,100,200,100,150,200);`

Texto

```
void DrawText2D(int x, int y, const char * format, ...);
```

Desenha o texto da cadeia de caracteres format contendo valores de variável (opcional) a partir da posição (x,y), da esquerda para a direita, semelhante ao printf.
Exemplo:

```
char * s = "Vidas Suficientes";
int pontos = 120;
graphics.DrawText2D(100,350,s);
graphics.DrawText2D(100,300,"Pontos: %d",pontos);
```

Texto – Fonte

```
void SetTextFont(const char * font_name, int size, int weight, bool italic, bool underline);
```

Altera a fonte dos textos a serem desenhados.

Valores parâmetro weight:

- FONT_WEIGHT_NONE
- FONT_WEIGHT_THIN
- FONT_WEIGHT_EXTRALIGHT
- FONT_WEIGHT_LIGHT
- FONT_WEIGHT_NORMAL
- FONT_WEIGHT_MEDIUM
- FONT_WEIGHT_SEMIBOLD
- FONT_WEIGHT_BOLD
- FONT_WEIGHT_EXTRABOLD
- FONT_WEIGHT_HEAVY

Exemplo:

```
char * s = "Vidas Suficientes";
int pontos = 120;
graphics.SetTextFont("Arial",28,FONT_WEIGHT_HEAVY,false,false);
graphics.DrawText2D(100,350,s);
graphics.SetTextFont("Times New Roman",42,FONT_WEIGHT_BOLD,true,false);
graphics.DrawText2D(100,300,"Pontos: %d",pontos);
```

Cor dos Objetos

```
void SetColor(float r, float g, float b);
```

Altera a cor, que será usada para desenhar os objetos, para o valor RGB (r,g,b), onde as componentes são: r para o vermelho (red), g para o verde (green) e b para o azul (blue). O valor de cada componente fica na faixa de 0 a 255.

Exemplo: `graphics.SetColor(255, 255, 0);  //max r + max g = amarelo puro`

Cor do Funda de Tela

```
void SetBackgroundColor(float r, float g, float b);
```

Altera a cor de fundo de tela para o valor RGB (r,g,b), onde as componentes são: r para o vermelho (red), g para o verde (green) e b para o azul (blue). O valor de cada componente fica na faixa de 0 a 255.

Exemplo: `graphics.SetBackgroundColor(255, 255, 255); //max r+ max g+ max b = branco`

Largura da Linha

```
void SetLineWidth(float w);
```

Altera a largura da linha dos objetos para w, incluindo a primitiva ponto.

Exemplo: `graphics.SetLineWidth(12);`

Rotacionar Objetos

```
void RotateBegin(float angle);
...
void RotateEnd();
```

O comando RotateBegin inicia o processo de rotação e o comando RotateEnd finaliza o processo de rotação. Todos os objetos desenhados depois do RotateBegin e antes do RotateEnd sofreram a rotação indicada pelo ângulo.

Exemplo:

```
graphics.RotateBegin(30);
graphics.FillRectangle2D(100,100,200,200);
graphics.RotateEnd();
```

Observação: Texto não é afetado pela rotação.

5 OUTRAS FUNÇÕES GRÁFICAS

Janela Principal

```
void CreateMainWindow(int sizeX, int sizeY, char * s);
```

Cria a janela principal com tamanho sizeX × sizeY e com título s.

Exemplo: `graphics.CreateMainWindow(800, 600, "Nome da Janela");`

Tela Cheia

```
void SetFullscreen(bool enable);
```

Coloca o programa em tela cheia, quando enable for verdade (true) e remove o programa quando enable for falso (false).

Exemplo:
```
graphics.SetFullscreen(true);   // full screen
graphics.SetFullscreen(false);  // sai de full screen
```

Largura da Janela

```
int GetScreenWidth();
```

Retorna a largura da tela.

Exemplo: `width = graphics.GetScreenWidth();`

Altura da Janela

```
int GetScreenHeight();
```

Retorna a altura da tela.

Exemplo: `height = graphics.GetScreenHeight();`

FPS – Frames per Second

```
float GetFPS();
```

Retorna o FPS corrente, isto é: o número de quadros por segundo.

Exemplo: `fps = graphics.GetFPS();`

Tempo Decorrido

```
float GetElapsedTime();
```

Retorna o tempo decorrido desde a geração do último frame.

Exemplo: `posX += 150 * graphics.GetElapsedTime());`

6 IMAGEM

Carregar Imagem

```
void LoadPNGImage(char * filename);
```

Carrega a imagem do arquivo filename para um objeto tipo Image.

Exemplo:

```
Image car1;
Image car2;
...
int main(void)
{
...
   car1.LoadPNGImage("c:\\resources\\BMW.png");
   car2.LoadPNGImage("Ferrari.png");
...
}
```

OBS: imagens devem ser carregadas apenas uma vez, por isto nunca devem ser carregadas dentro do loop principal. Apenas imagens PNG são aceitas.

Posicionar Imagem

```
void SetPosition(int x, int y, int width, int height);
```

Define a posição (x,y) da tela para uma imagem de tamanho width x height ser desenhada, como uma função de um objeto tipo Image.

Exemplo:

```
Image car;
...
```

```
void MainLoop(void)
{
...
car.SetPosition(200, 200, 256, 256);
...
}
```

Desenhar Imagem

```
void DrawImage2D(int x, int y, int width, int height, Image image);
void DrawImage2D(Image image);
void DrawImage2D(int x, int y, int width, int height, int crop_x, int
crop_y, int crop_width, int crop_height, Image image);
```

Desenha a imagem image de tamanho width x height na posição (x,y) da tela, como um objeto gráfico. A segunda opção requer o uso anterior da função SetPosition que posiciona a imagem.

Exemplo:

```
Graphics graphics;
Image car1;
Image car2;
...
void MainLoop(void)
{
...
graphics.DrawImage2D(100, 100, 256, 256, car1);
car2.SetPosition(200, 200, 256, 256);
graphics.DrawImage2D(car2);
...
}
```

A terceira opção é utilizada para desenhar somente uma determinada região da imagem. A imagem image será desenhada com o tamanho width x height na posição (x,y) da tela, sendo desenhado somente a região da imagem image que se inicia na posição (crop_x, crop_y) e possui tamanho crop_width x crop_height.

Exemplo:

```
Graphics graphics;
Image sprite1;
...
void MainLoop(void)
{
...
graphics.DrawImage2D(100, 100, 128, 128, 256, 256, 128, 128, sprite1);
...
}
```

7 ÁUDIO

Carregar Áudio

```
void LoadAudio(char * filename);
```

Carrega o áudio MP3 ou WAV do arquivo filename para um objeto tipo Audio.

Exemplo:

```
Audio explosion;
...
int main(void)
{
...
   explosion.LoadPNGImage("c:\\resources\\explosion.mp3");
```

```
...
}
```

OBS: áudios devem ser carregados apenas uma vez, por isto nunca devem ser carregados dentro do loop principal.

Executar Áudio

```
void Play(void);
```

Toca o áudio de um objeto tipo Audio que já foi previamente carregado pela LoadAudio.

Exemplo:

```
Audio explosion;
...
   explosion.Play();
```

OBS: a função Play() pode ser invocada a partir de qualquer função (main, MainLoop, …).

Parar Execução de Áudio

```
void Stop(void);
```

Para a execução do áudio de um objeto tipo Audio.

Exemplo:

```
Audio explosion;
...
   explosion.Stop();
```

Pausar Execução de Áudio

```
void Pause(void);
```

Pausa a execução do áudio de um objeto tipo Audio.

Exemplo:

```
Audio explosion;
...
   explosion.Pause();
```

Verificar se Áudio está sendo Executado

```
bool IsPlaying(void);
```

Verifica se o áudio de um objeto tipo Audio está sendo executado, retornando `true` ou `false`.

Exemplo:

```
Audio explosion;
...

   if (explosao.IsPlaying() == false)
      explosao.Play();
```

OBS: usar IsPlaying como no exemplo acima e dentro de um Loop Principal é uma maneira de fazer um áudio ficar permanentemente tocando.

8 INTERAÇÃO – TECLADO

Eventos de Teclado

```
void SetKeyboardInput(func);
```

Indicação da função que tratará de eventos de teclado, cujo protótipo deve ser o seguinte:

```
void func(int key, int state, int x, int y)
```

onde key é o código da tecla que foi pressionada/liberada, state indica o estado da tecla (se ela foi pressionado ou librada), e (x,y) é a posição do mouse quando a tecla foi pressionada/liberada.

Exemplo:

```
void KeyboardInput(int key, int state, int x, int y);
```

```
{
   if ((key == 'a') &&(state == KEY_STATE_DOWN)))
   {
     ...
   }
   if ((key == KEY_RIGHT) &&(state == KEY_STATE_DOWN)))
   {
     ...
   }
   if ((key == KEY_LEFT) &&(state == KEY_STATE_UP)))
   {
     ...
   }

}
int main(void)
{
   graphics.SetKeyboardInput(KeyboardInput);
   ...
}
```

Para verificar o pressionamento das teclas referentes a letras, números e outros símbolos é possível comparar o valor da variável key com um char contendo o caractere desejado. Exemplo: `'a'`

Algumas teclas, como por exemplo, as setas direcionais, requerem códigos especiais. Os códigos das **teclas especiais** são os seguintes:

KEY_LEFT
KEY_UP
KEY_RIGHT
KEY_DOWN
KEY_PAGE_UP
KEY_PAGE_DOWN
KEY_HOME
KEY_END
KEY_INSERT
KEY_ESC
KEY_ENTER
KEY_BACKSPACE
KEY_LEFTCTRL
KEY_RIGHTCTRL
KEY_LEFTSHIFT
KEY_RIGHTSHIFT
KEY_LEFTALT
KEY_RIGHTALT
KEY_TAB
KEY_F1
KEY_F2
KEY_F3
KEY_F4
KEY_F5
KEY_F6
KEY_F7
KEY_F8
KEY_F9
KEY_F10
KEY_F11
KEY_F12

A variável state pode assumir dois valos dependendo do estado da tecla. Enquanto a tecla estiver sendo pressionada a variável state terá o valor KEY_STATE_DOWN. Quando a tecla for liberada, a variável state terá o valor KEY_STATE_UP.

9 INTERAÇÃO – MOUSE

Eventos de Cliques de Mouse

```
void SetMouseClickInput(func);
```

Indicação da função que tratará de eventos de clique do mouse, cujo protótipo deve ser o seguinte:

```
void func(int button, int state, int x, int y);
```

onde button é o botão que foi pressionado, (x,y) é a posição do mouse quando o clique foi realizado e state é o estado que o botão pode assumir – *i.e.* para baixo (down) ou para cima (up).
Os códigos dos botões do mouse são os seguintes:

MOUSE_LEFT_BUTTON
MOUSE_MIDDLE_BUTTON
MOUSE_RIGHT_BUTTON

Os códigos para os estados que os botões podem assumir são os seguintes:

MOUSE_STATE_DOWN
MOUSE_STATE_UP

Exemplo:

```
void MouseClickInput(int button, int state, int x, int y)
{
   if (button==MOUSE_LEFT_BUTTON && state==MOUSE_STATE_DOWN)
   {
      destino_x = x;  // variavel global destino_x guarda posicao x do clique
      destino_y = y;  // variavel global destino_y guarda posicao y do clique
   }
}
int main(void)
{
   graphics.SetMouseClickInput(MouseClickInput);
   ...
}
```

Movimento Livre do Mouse

```
void SetMouseMotionInput(func);
```

Indicação da função que tratará de eventos de movimento do mouse, cujo protótipo deve ser o seguinte:

```
void func(int x, int y)
```

onde (x,y) é a posição corrente do cursor do mouse na tela.
Exemplo:

```
void MouseMotionInput(int x, int y)
{
   mouse_x = x;
   mouse_y = y;
}
int main(void)
{
   graphics.SetMouseMotionInput(MouseMotionInput);
   ...
}
```

Eventos de Clique de Mouse sobre uma Imagem

```
void SetOnClick(func);
```

Indicação da função que tratará de eventos de clique do mouse sobre uma imagem (i.e. sobre um específico objeto Image), cujo protótipo deve ser o seguinte:

```
void func(int button, int state, int x, int y)
```

onde button é o botão que foi pressionado, (x,y) é a posição do mouse relativa à imagem que o mouse estava quando o clique foi realizado e state é o estado em que o botão se encontra.

OBS: Para poder usar este evento é necessário que a posição da imagem tenha sido definida com o comando **SetPosition**.

Exemplo:

```
Image yoda;
...

void MouseClickYoda(int button, int state, int x, int y)
{
   clicou_em_yoda= true;
   estadoBotaoSobreYoda = state;
   ...
}

int main(void)
{
   yoda.LoadPNGImage("yoda.png");
   yoda.SetPosition(0,100,256,256);
   yoda.SetOnClick(MouseClickYoda);
   ...
}
```