PlayLib

Educational Game Programming Library

Documentação

Edirlei Soares de Lima

elima@inf.puc-rio.br

Sumário

1 Instalação e Configuração no Visual Studio 2010

1) Faça o download da última versão da PlayLib: http://www.inf.puc-rio.br/~elima/playlib/

2) Se você optar pelo instalador automático, basta executar o arquivo "**PlayLib_V1.0.exe**" e seguir as instruções de instalação. Por padrão, os arquivos instalados ficaram na pasta: "**C:\Arquivos de Programas\PlayLib**". Lembre-se que é necessário ter direitos de administrador do computador para poder usar o instalador automático.

3) Se você optar pela versão compactada, você deve descompactar o arquivo "**PlayLib_V1.0.zip**" para a pasta de sua preferencia.

4) Após a instalação você encontrará as seguintes pastas:

5) A pasta "**doc**" contém a documentação da PlayLib.

6) A pasta "**include**" contém os arquivos para serem incluídos nos programas criados.

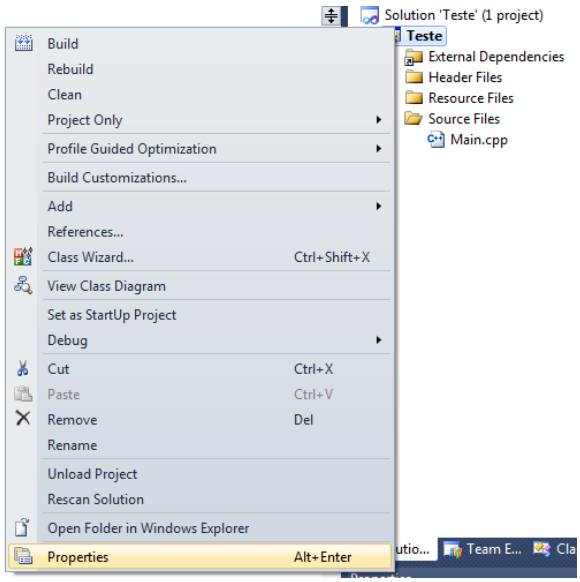
7) A pasta "**lib**" contém algumas dlls necessárias e a biblioteca para ser utilizada nos programas criados.

8) Para utilizar a PlayLib, crie um novo projeto no **Microsoft Visual Studio 2010**. Este projeto deve ser do tipo **Win32 Console Application** na linguagem **C++.**

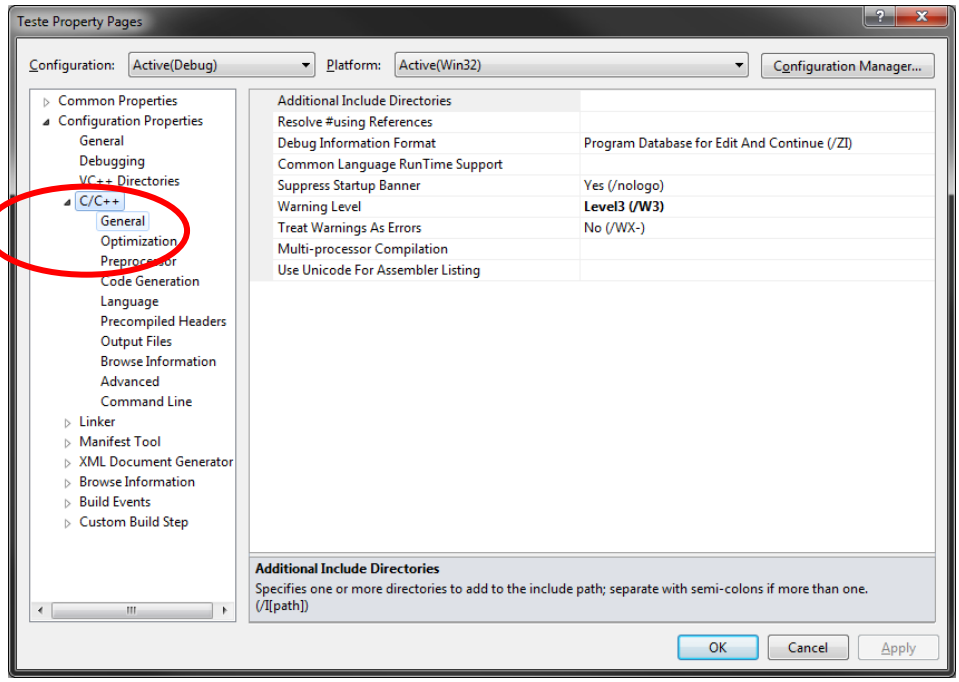
   Para mais informação sobre a criação de um projeto siga as instruções deste tutorial de utilização do Visual Studio 2010:

   http://www.inf.puc-rio.br/~elima/intro-prog/IntroProg_Aula_04_Introducao_Visual_Studio.pdf

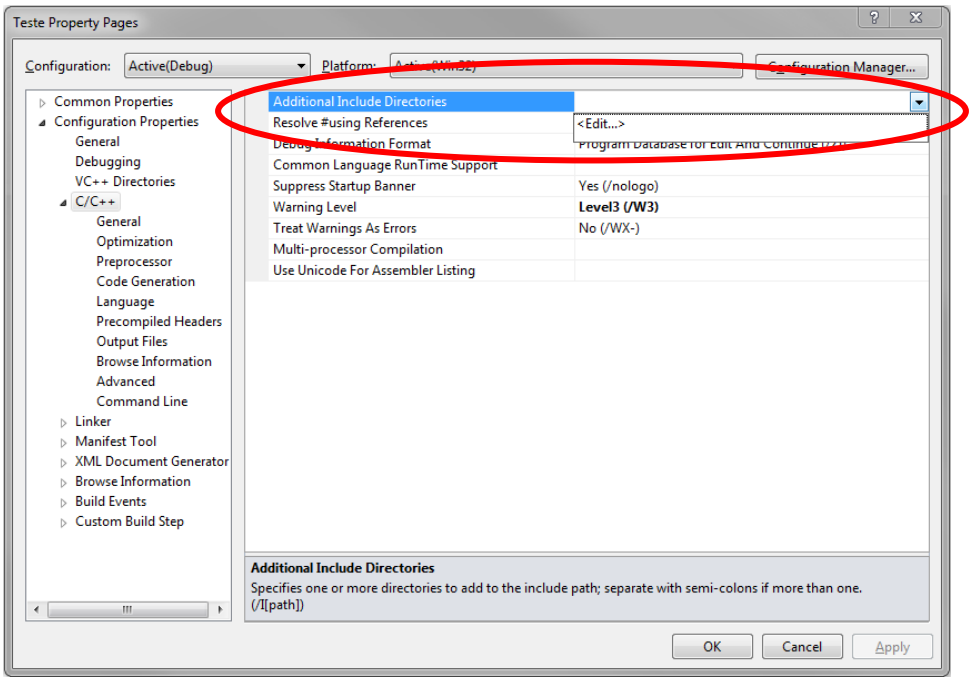
9) Acesse as **propriedades do projeto** clicando com o botão da direita no nome do seu projeto.



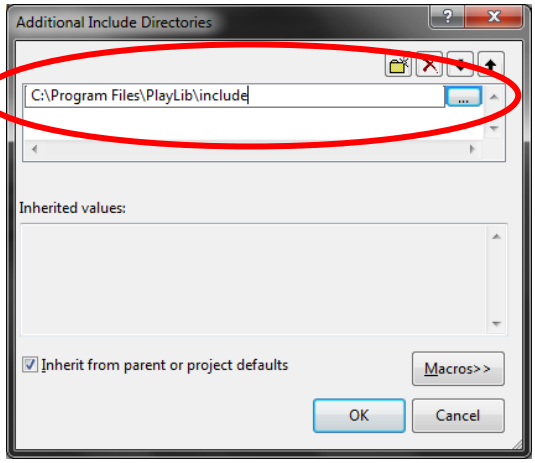
10) Selecione a opção **C/C++** e a sub-opção **General**.



11) Selecione a opção **Additional Include Directories** e clique em **<Edit...>**

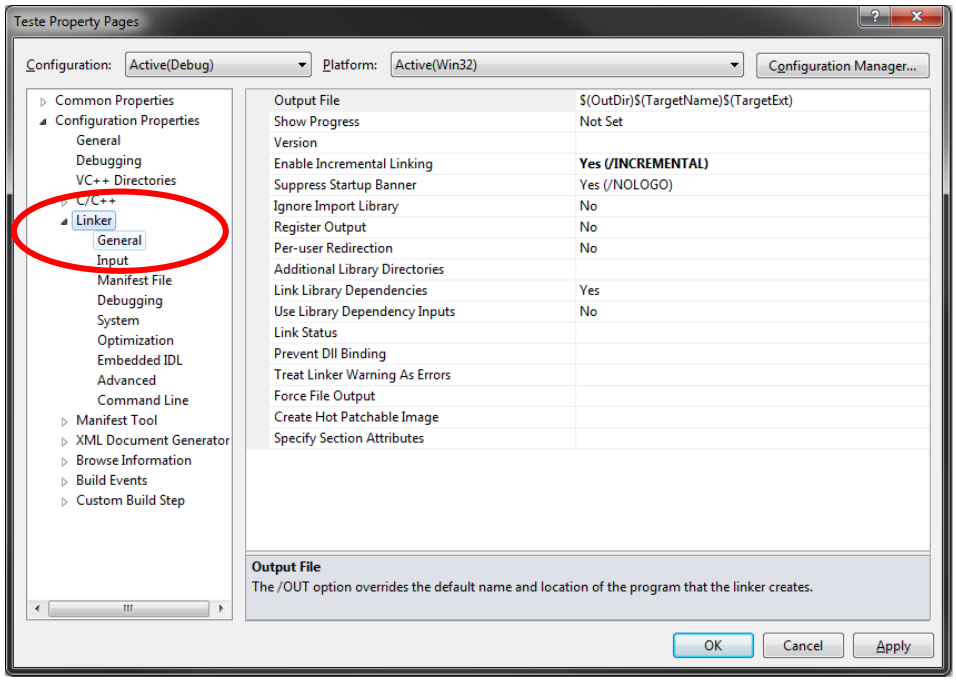


12) Selecione ou digite o caminho completo para a pasta **include** que está dento da pasta **PlayLib**.

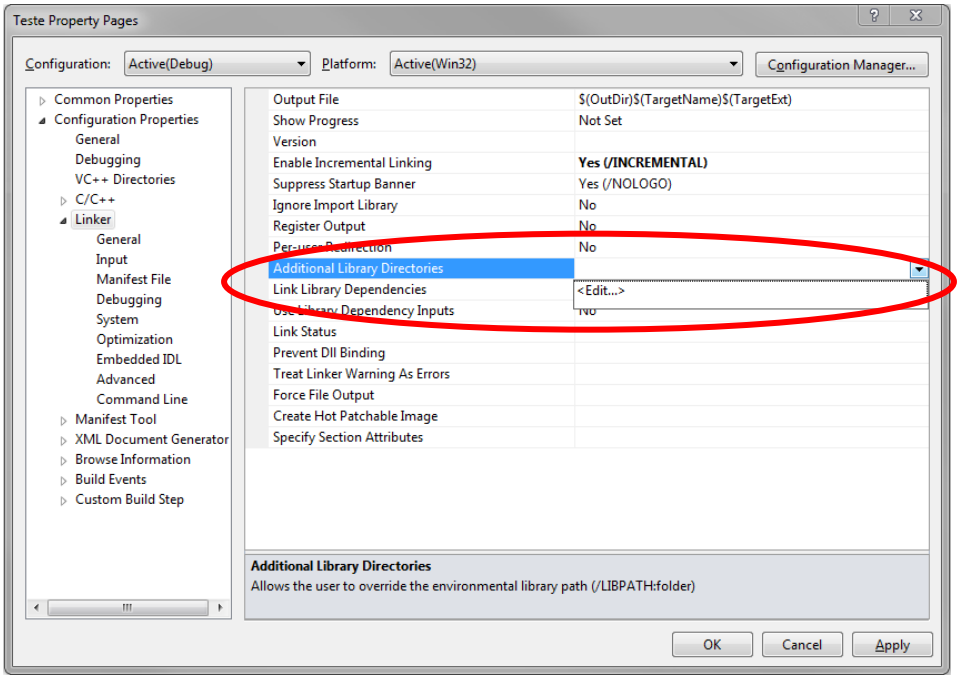


13) Clique em **OK**.

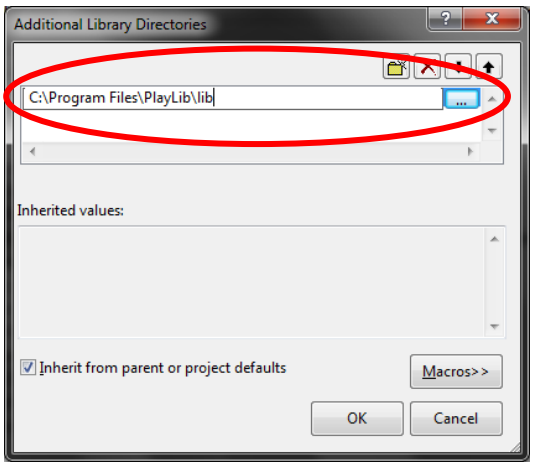
14) Selecione a opção **Linker** e a sub-opção **General**.



15) Selecione a opção **Additional Library Directories** e clique em **<Edit...>**

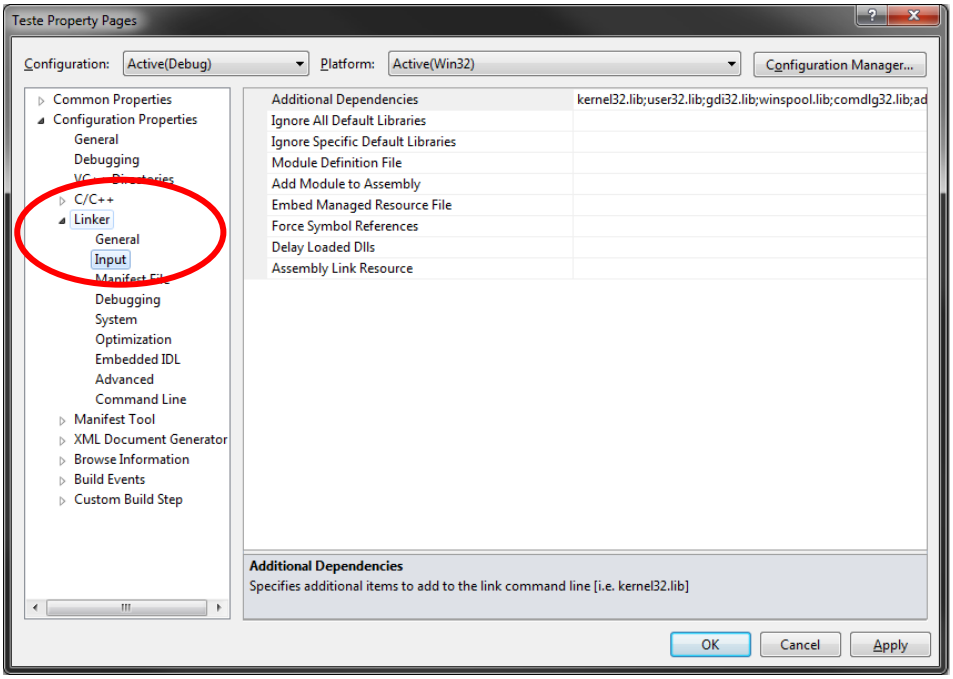


16) Selecione ou digite o caminho completo para a pasta **lib** que está dento da pasta **PlayLib**.

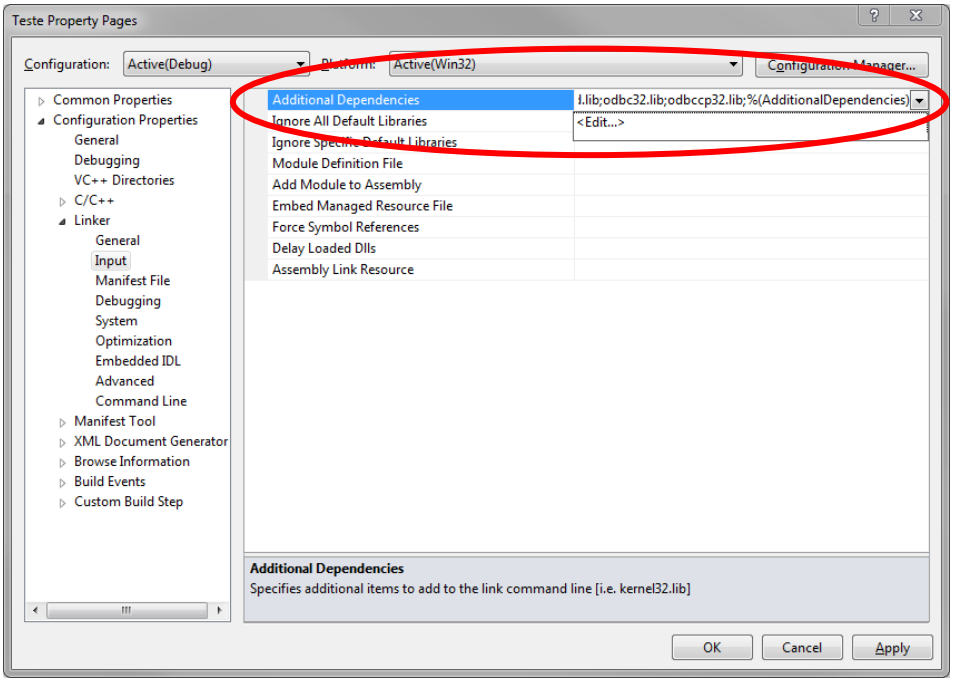


17) Clique em OK.

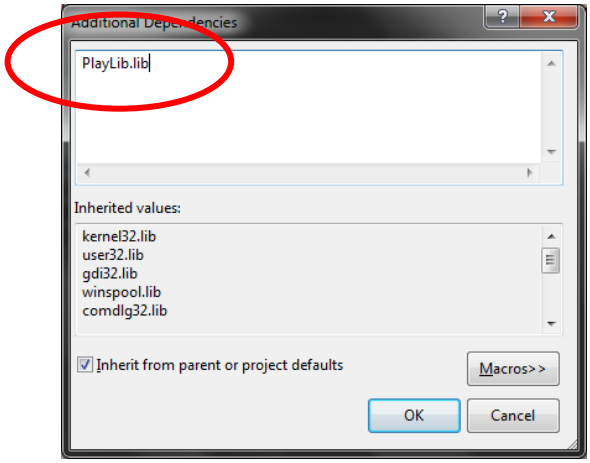
18) Selecione a opção **Linker** e a sub-opção **Input**.



19) Selecione a opção **Additional Dependencies** e clique em **<Edit...>**

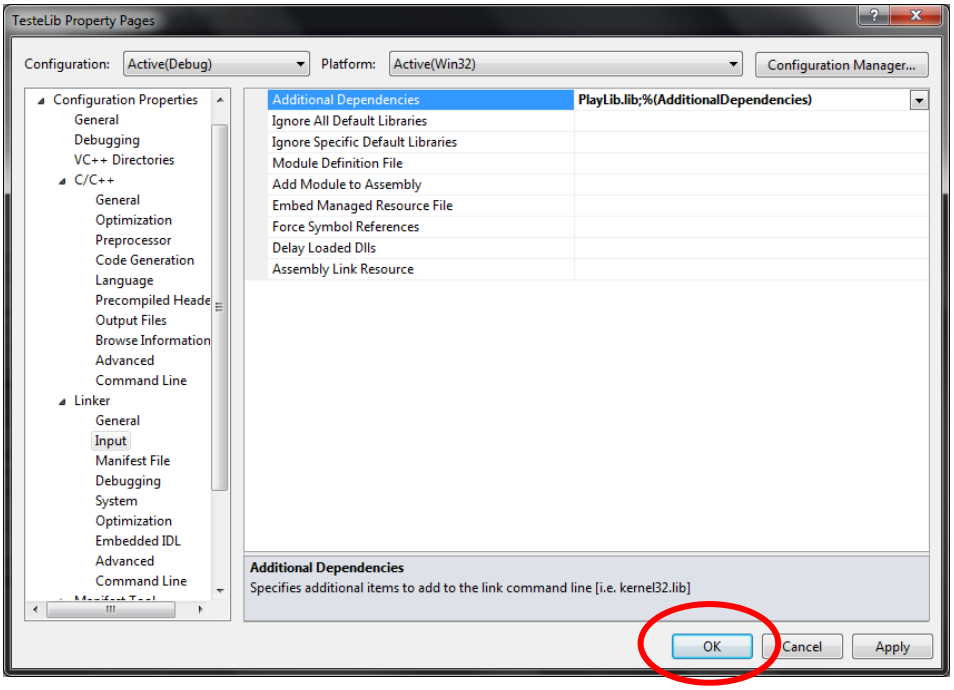


20) Digite **PlayLib.lib**



21) Clique em **OK**.

22) Clique em **OK** para concluir a configuração do projeto.


TesteLib Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
General
Debugging
VC++ Directories
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Heade
Output Files
Browse Information
Advanced
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
Additional Dependencies
PlayLib.lib;%(AdditionalDependencies)
Ignore All Default Libraries
Ignore Specific Default Libraries
Module Definition File
Add Module to Assembly
Embed Managed Resource File
Force Symbol References
Delay Loaded Dlls
Assembly Link Resource
Additional Dependencies
Specifies additional items to add to the link command line [i.e. kernel32.lib]
OK
Cancel
Apply

2 Estrutura de um Programa

A PlayLib é uma biblioteca educacional de programação de jogos que tem como objetivo simplificar o processo de desenvolvimento de aplicativos gráficos e auxiliar, de forma lúdica, o aprendizado de técnicas de programação aplicadas na criação de jogos.

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

Com a PlayLib é possível criar jogos 2D, simulações cientificas, animações e outros aplicativos gráficos.

O código a seguir ilustra a estrutura básica de um programa criado com a PlayLib:

```
#include "Graphics.h"

Graphics graphics;

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

  graphics.StartMainLoop();

  return 0;
}
```

Inclusão da biblioteca gráfica da PlayLib

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

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

Inicia o **Loop principal** do programa

O programa anterior simplesmente cria uma janela de tamanho 800x600 com o título “Teste” como ilustrado na figura abaixo:

3 Loop Principal

O **Loop Principal** consiste de uma função que é repetida enquanto o programa não for fechado pelo usuário. Todo processamento realizado pelo programa gráfico está de alguma forma ligado ao Loop Principal.

No Loop Principal deve ser programado:

- Os objetos que serão desenhados na tela e como eles serão apresentados;
- Quais animações e movimentos os objetos terão.
- Toda a lógica do programa.

Para criar o Loop Principal do programa é necessário criar uma função que será utilizada como Loop Principal. Em seguida é necessário indicar que a função criada será o Loop Principal do programa.

Exemplo:

```
#include "Graphics.h"

Graphics graphics;

void MainLoop()
{

  graphics.SetColor(0,255,0);



  graphics.FillRectangle2D(100, 100, 400, 200);

}

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

  graphics.SetMainLoop(MainLoop);

  graphics.StartMainLoop();

  return 0;
}
```

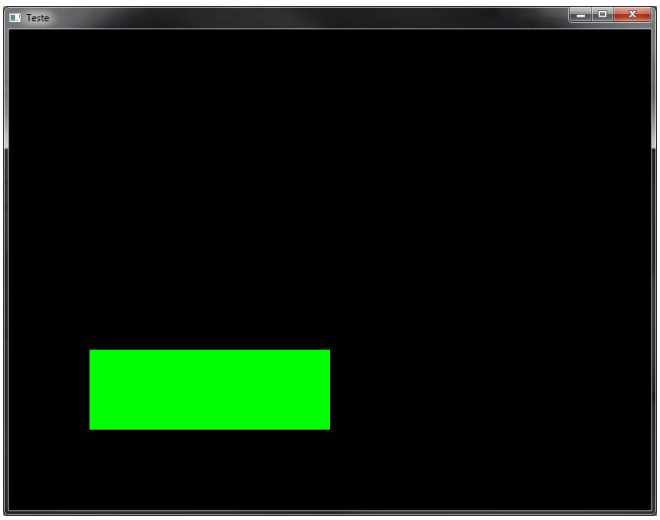
Função que será usada como **Loop Principal** do programa

Define a **cor** que será utilizada para desenhar objetos na tela (Formato RGB)

Desenha um **retângulo** preenchido iniciando na posição (100,100) e indo até (200,400)

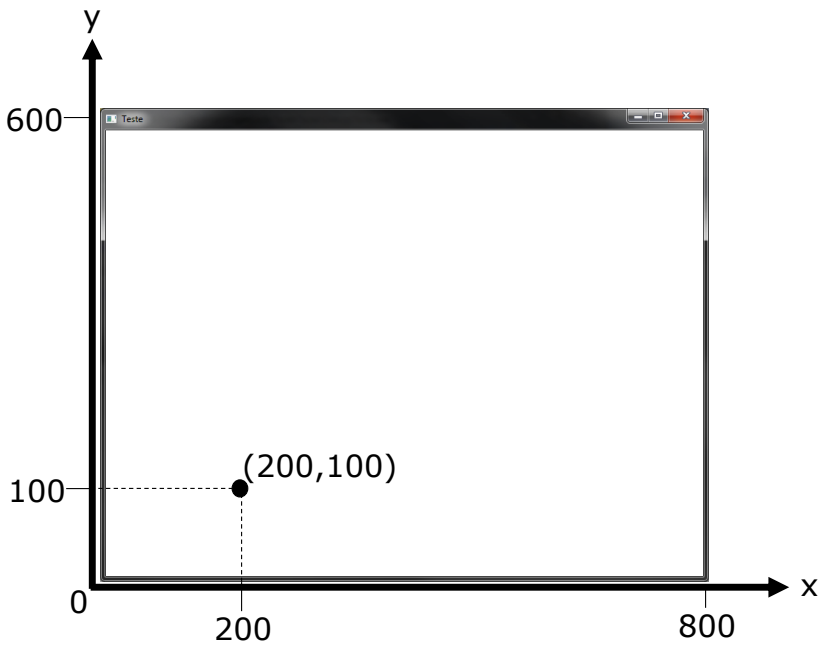
Define que a função **MainLoop** será o **Loop Principal** do programa

O programa anterior desenha na tela um retângulo preenchido iniciando na posição (100,100) e indo até (200,400) na cor verde como ilustrado na figura abaixo:



4 Coordenadas de Tela

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. Esse sistema de coordenadas é ilustrado na figura abaixo:



5 Desenho de Primitivas Geométricas

A PlayLib fornece um conjunto de funções para o desenho de primitivas geométricas básicas. As próximas seções detalham essas funções.

5.1 Ponto

Sintaxe:

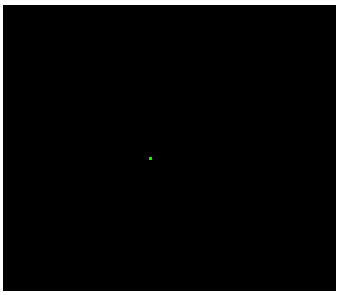
```
void DrawPoint2D(int x, int y);
```

Exemplo:

```
graphics.DrawPoint2D(200, 200);
```

Desenha um ponto na posição (200, 200) da tela.

Ilustração:



5.2 Linha

Sintaxe:

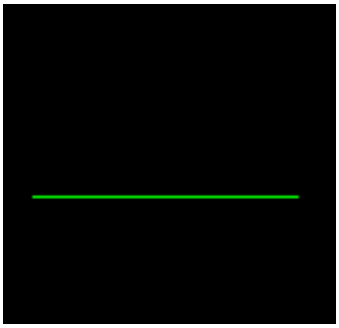
```
void DrawLine2D(int x1, int y1, int x2, int y2);
```

Exemplo:

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

Desenha uma linha saindo da posição (100, 100) e indo até a posição (200, 100)

Ilustração:



5.3 Círculo

Sintaxe:

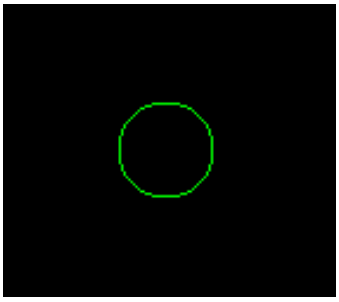
```
void DrawCircle2D(int x, int y, int radius);
```

Exemplo:

```
graphics.DrawCircle2D(200, 200, 20);
```

Desenha um círculo de raio 20 na posição (200, 200) da tela.

Ilustração:



5.4 Círculo Preenchido

Sintaxe:

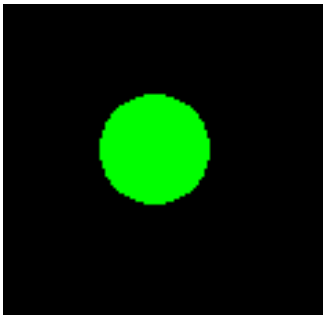
```
void FillCircle2D(int x, int y, int radius, int segments);
```

Exemplo:

```
graphics.FillCircle2D(200, 200, 20, 30);
```

Desenha um círculo preenchido de raio 20 com 30 segmentos na posição (200, 200) da tela. Quantos mais segmentos, mais redondo o circulo será.

Ilustração:



5.5 Retângulo

Sintaxe:

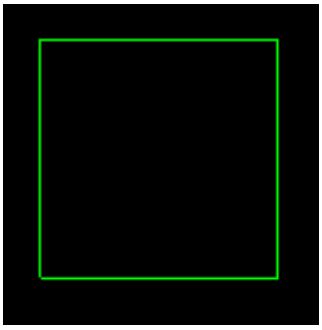
```
void DrawRectangle2D(int x1, int y1, int x2, int y2);
```

Exemplo:

```
graphics.DrawRectangle2D(100,100,200,200);
```

Desenha um retângulo iniciando na posição (100, 100) e indo até a posição (200, 200).

Ilustração:



5.6 Retângulo Preenchido

Sintaxe:

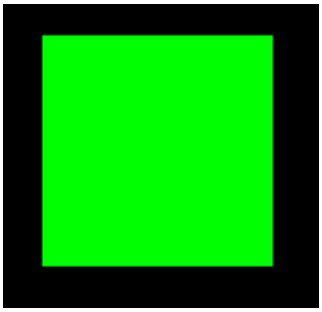
```
void FillRectangle2D(int x1, int y1, int x2, int y2);
```

Exemplo:

```
graphics.FillRectangle2D(100,100,200,200);
```

Desenha um retângulo preenchido iniciando na posição (100, 100) e indo até a posição (200, 200).

Ilustração:



5.7 Triângulo

Sintaxe:

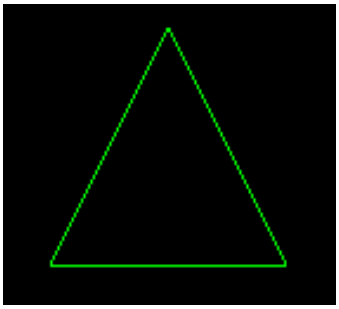
```
void DrawTriangle2D(int x1, int y1, int x2, int y2, int x3, int y3);
```

Exemplo:

```
graphics.DrawTriangle2D(100,100,200,100,150,200);
```

Desenha um triângulo com o primeiro ponto na posição (100, 100), segundo ponto na posição (200, 100) e terceiro ponto na posição (150, 200).

Ilustração:



5.8 Triângulo Preenchido

Sintaxe:

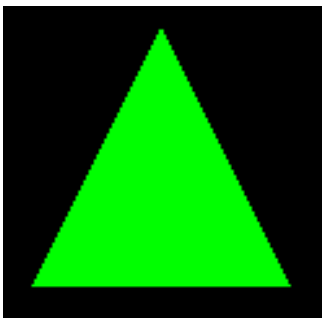
```
void FillTriangle2D(int x1, int y1, int x2, int y2, int x3, int y3);
```

Exemplo:

```
graphics.FillTriangle2D(100,100,200,100,150,200);
```

Desenha um triângulo preenchido com o primeiro ponto na posição (100, 100), segundo ponto na posição (200, 100) e terceiro ponto na posição (150, 200).

Ilustração:



5.9 Texto

Sintaxe:

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

Exemplo:

```
graphics.DrawText2D(100, 100, "Pontos: %d", pontos);
```

Escreve “Pontos: ” seguido do valor da variáveis “pontos” na posição (100, 100) da tela.

Ilustração:

5.10 Modificando a Fonte do Texto

Sintaxe:

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

Exemplo:

```
graphics.SetTextFont("Times New Roman", 42, FONT_WEIGHT_BOLD, false, false);
```

Altera a fonte utilizada no programa para “Times New Roman”, com tamanho 42, bold.

Ilustração:

Os possíveis valores para o parâmetro “weight” são:

- 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

5.11 Modificando a Cor

Sintaxe:

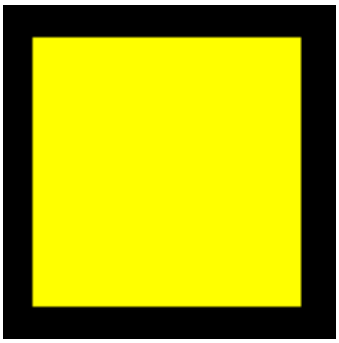
```
void SetColor(float r, float g, float b);
```

Exemplo:

```
graphics.SetColor(255, 255, 0);
```

Altera a cor que será usada para desenhar os objetos para o valor RGB (255,255,0). Ou seja, mistura o máximo de vermelho com o máximo de verde, o que resulta em amarelo.

Ilustração:



5.12 Modificando a Cor de Fundo da Tela

Sintaxe:

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

Exemplo:

```
graphics.SetBackgroundColor(255, 255, 255);
```

Altera a cor do fundo da tela para o valor RGB (255,255,255). Ou seja, mistura o máximo de todas as cores, o que resulta em branco.

Ilustração:

5.12 Modificando a Largura das Linhas

Sintaxe:

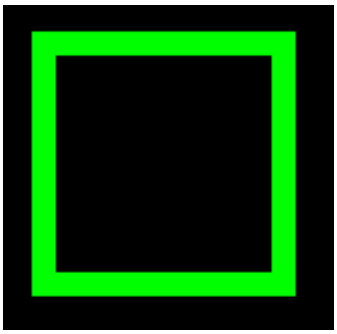
```
void SetLineWidth(float width);
```

Exemplo:

```
graphics.SetLineWidth(12);
```

Altera para 12 a largura das linhas usadas para desenhar as formas geométricas.

Ilustração:



5.13 Rotacionar Objetos

Sintaxe:

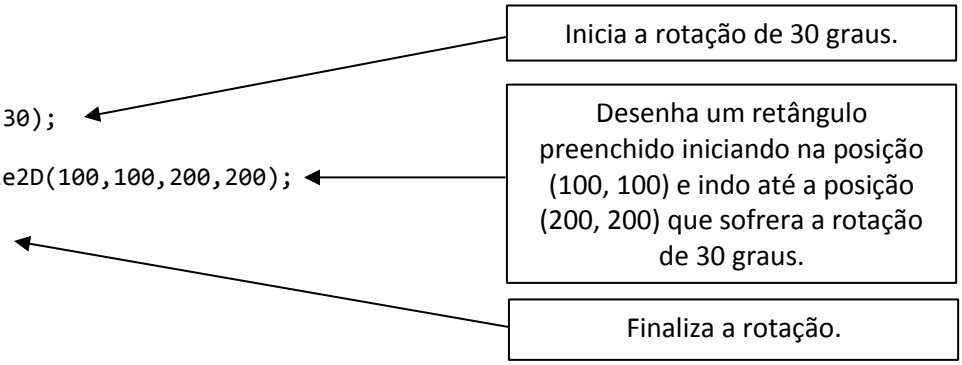
```
void RotateBegin(float angle);
```

e

```
void RotateEnd();
```

Exemplo:

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



Todos os objetos desenhados depois do **RotateBegin** e antes do **RotateEnd** sofreram a rotação indicada pelo ângulo.

Observação: O texto não é afetado pela rotação.

6 Outras Funções

6.1 Criando a Janela do Programa

Sintaxe:

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

Exemplo:

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

Cria a janela principal do programa com o tamanho 800x600 e com o título “Nome da Janela”

Ilustração:

6.2 Executando o Programa em Tela Cheia

Sintaxe:

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

Exemplo:

```
graphics.SetFullscreen(true);
```

Coloca o programa em tela cheia

```
graphics.SetFullscreen(false);
```

Remove o programa da tela cheia

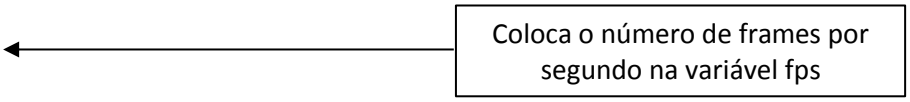
6.3 Velocidade de Execução do Programa (FPS)

Sintaxe:

```
float GetFPS();
```

Exemplo:

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



FPS (Frames per Second): Medida que nos indica quantos frames (imagens) consecutivos a placa de vídeo está conseguindo gerar por segundo.

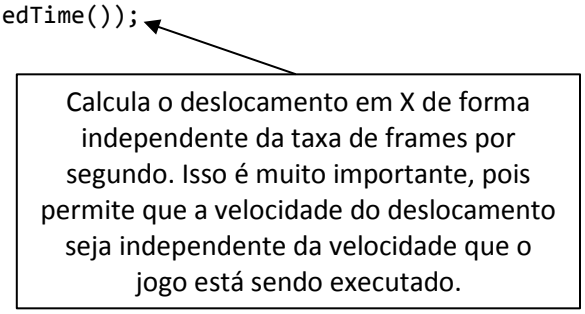
6.4 Velocidade de Execução do Programa (ElapsedTime)

Sintaxe:

```
float GetElapsedTime();
```

Exemplo:

```
PosicaoX = PosicaoX + (Speed * graphics.GetElapsedTime());
```



6.5 Largura e Altura da Janela

Sintaxe:

```
int GetScreenWidth();

int GetScreenHeight();
```

Exemplo:

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

Coloca a largura da tela na variável width

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

Coloca a altura da tela na variável height

7 Imagem

Para desenhar uma imagem na tela é necessário:

1) Criar uma variável do tipo Image.

```
Image minha_imagem;
```

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

Exemplo:

```
#include "Graphics.h"

Graphics graphics;
Image minha_imagem1;
Image minha_imagem2;

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

Variáveis Image declaradas no inicio do programa. Antes e fora da função principal ou outras funções.

2) Carregar a imagem do arquivo usando o comando LoadPNGImage.

```
minha_imagem.LoadPNGImage("Mario.png");
```

Lembre-se de colocar o arquivo **Mario.png** na pasta do seu projeto. Ou passe como parâmetro o caminho completo para a imagem, exemplo: "C:\\Imagens\\Mario.png"

Exemplo:

```
int main(void)
{
...
  minha_imagem.LoadPNGImage("Mario.png");
...
}
```

Carrega a imagem do arquivo **Mario.png** para a variável minha_imagem.

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

3) Desenhar efetivamente a imagem na tela usando o comando DrawImage2D.

```
graphics.DrawImage2D(200, 200, 256, 256, minha_imagem);
```

Exemplo:

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

Desenha a imagem "minha_imagem" na posição (200, 200) com tamanho (256, 256) na tela.

Também é possível definir a posição e tamanho das imagens em variáveis armazenadas dentro do objeto Image. Para isso, deve-se:

1) **Criar uma variável do tipo Image.**

```
Image minha_imagem;
```

2) **Carregar a imagem do arquivo usando o comando LoadPNGImage.**

```
minha_imagem.LoadPNGImage("Mario.png");
```

3) **Definir a posição da imagem com o comando SetPosition.**

```
minha_imagem.SetPosition(100,100,256,256);
```

Posiciona a imagem "minha_imagem" na posição (100, 100) com tamanho (256, 256) na tela.

4) **Desenhar a imagem na tela com o comando DrawImage2D.**

```
graphics.DrawImage2D(minha_imagem);
```

Note que não é necessário passar a posição e o tamanho da imagem como parâmetro. Ela será desenhada na posição definida pelo comando SetPosition

7.1 Carregando uma Imagem

Sintaxe:

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

Exemplo:

```
Image mario;
```

Declaração da variável do tipo Image que vai armazenar a imagem

```
mario.LoadPNGImage("Mario.png");
```

Carrega o arquivo “Mario.png” para a variável “mario”

Ilustração:

7.2 Desenhando uma Imagem

Sintaxe:

```
void DrawImage2D(int x, int y, int width, int height, Image image);

ou

void DrawImage2D(Image image);

ou

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

Exemplo:

```
Image mario;
```

Declaração da variável do tipo Image que vai armazenar a imagem

```
graphics.DrawImage2D(200, 200, 256, 256, mario);
```

Desenha a imagem “mario” na posição (200, 200) com tamanho (256, 256) na tela.

ou

```
graphics.DrawImage2D(mario);
```

Desenha a imagem “mario” na posição definida anteriormente pelo comando SetPosition.

ou

```
graphics.DrawImage2D(200,200,128,128,0,0,128,128,sprite);
```

Desenha a região de posição (0, 0) e tamanho (128, 128) da imagem “sprite” na posição (200, 200) com tamanho (128, 128) na tela.

Ilustração:

7.3 Definindo a Posição uma Imagem

Sintaxe:

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

Exemplo:

```
mario.SetPosition(200, 200, 256, 256);
```

Define a posição da imagem “mario” na posição (200, 200) com tamanho (256, 256) na tela.

Ilustração:

7.4 Observações importantes sobre imagens

- **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.

- Cerifique-se de que as imagens que serão lidas estão **dentro da pasta do seu projeto do Visual Studio**. Se preferir armazena-las em outras pastas você deve fornecer o caminho completo para o diretório onde as imagens estão para o comando LoadPNGImage.

- Se a sua imagem estiver em **outro formado** (JPG, GIF, BMP...) você deve convertê-la para o formato PNG antes de carrega-la.

8 Áudio

A PlayLib suporta arquivos de áudio no formato **MP3** e **WAV**. Para executar um áudio (música ou efeito sonoro) com a PlayLib é necessário:

1) Incluir a biblioteca de áudio.

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

2) Criar uma variável do tipo Audio.

```
Audio minha_musica;
```

OBS: Sempre declare as variáveis Audio como **variáveis globais**.

Exemplo:

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

Graphics graphics;
Audio musica1;
Audio musica2;

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

Variáveis Audio declaradas no inicio do programa. Antes e fora da função principal ou outras funções.

3) Carregar o áudio do arquivo usando o comando LoadAudio.

```
musica1.LoadAudio("Musica.mp3");
```

Lembre-se de colocar o arquivo **Musica.mp3** na pasta do seu projeto. Ou passe como parâmetro o caminho completo para a imagem, exemplo: "C:\\Imagens\\Musica.mp3"

Exemplo:

```
int main(void)
{
...
  musica1.LoadAudio("Musica.mp3");
...
}
```

Carrega a imagem do arquivo **Musica.mp3** para a variável minha_imagem.

OBS: Cada áudio deve ser carregado **apenas uma vez**. Por isso, nunca carregue o áudio diretamente de dentro do Loop Principal.

4) Executar efetivamente o áudio usando o comando Play.

```
musica1.Play();
```

Exemplo:

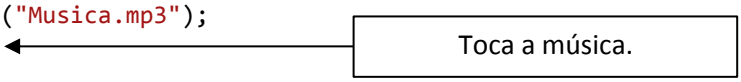
```
#include "Graphics.h"
#include "Audio.h"

Graphics graphics;

Audio musica1;

int main(void)
{
  musica1.LoadAudio("Musica.mp3");
  musica1.Play();
  ...

}
```



8.1 Carregando um Áudio

Sintaxe:

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

Exemplo:

```
Audio explosao;
```

Declaração da variável do tipo Audio que vai armazenar o efeito sonoro

```
explosao.LoadAudio("Explosao.mp3");
```

Carrega o arquivo “Explosao.mp3” para a variável “explosao”

8.2 Executando um Áudio

Sintaxe:

```
void Audio.Play();
```

Exemplo:

```
Audio explosao;
```

Declaração da variável do tipo Audio que vai armazenar o efeito sonoro

```
explosao.LoadAudio("Explosao.mp3");
```

Carrega o arquivo “Explosao.mp3” para a variável “explosao”

```
explosao.Play();
```

Toca o áudio “explosao”.

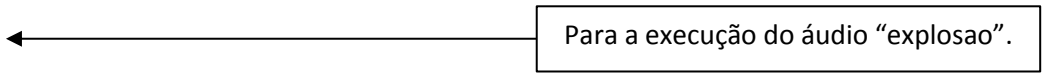
8.3 Parando a Execução de um Áudio

Sintaxe:

```
void Audio.Stop();
```

Exemplo:

```
explosao.Stop();
```

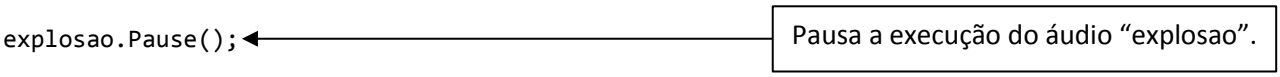


8.4 Pausando a Execução de um Áudio

Sintaxe:

```
void Audio.Pause();
```

Exemplo:



8.5 Verificando se um Áudio está sendo Executando

Sintaxe:

```
bool Audio.IsPlaying();
```

Exemplo:

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

Verifica se áudio “explosao” não está sendo executado.

Toca o áudio “explosao”.

9 Interação

A **PlayLib** oferece suporte para a interação pelo **teclado e mouse**.

9.1 Tratando Entradas do Teclado

Para poder tratar os eventos gerados pelo teclado (**teclas sendo pressionadas**) é necessário criar uma função para essa tarefa. Essa função deve ter a seguinte sintaxe:

```
void KeyboardInput(int key, int state, int x, int y)
{
       /* Bloco de Comandos */
}
```

Também é **necessário indicar** que essa é a sua função para tratar eventos de teclado usando a função SetKeyboardInput:

```
graphics.SetKeyboardInput(KeyboardInput);
```

Dessa forma, sempre que uma tecla do teclado for pressionada a função **KeyboardInput** será executada e o parâmetro **key** indicará qual tecla foi pressionada. O parâmetro **state** indicará se a tecla foi pressionada ou liberada. E os parâmetros **x** e **y** indicam a posição do mouse quando a tecla foi pressionada.

Exemplo:

```
void KeyboardInput(int key, int state, int x, int y)
{
  if ((key == 'f')&&(state == KEY_STATE_DOWN))
  {
    graphics.SetFullscreen(true);
  }
  if ((key == KEY_RIGHT)&&(state == KEY_STATE_DOWN))
  {
    posicao_personagem_x = posicao_personagem_x + 2;
  }
  if ((key == KEY_ESC)&&(state == KEY_STATE_DOWN))
  {
    exit(0);
  }
}
```

Se a letra f for pressionada

Coloca o programa em tela cheia

Se a letra a seta direcional direita for pressionada

Incrementa em +2 uma variável que representa a posição de um personagem

Se a tecla Esc for pressionada

Fecha o programa

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

Os códigos dos estados das teclas são os seguintes:

- KEY_STATE_DOWN
- KEY_STATE_UP

9.2 Tratando Cliques do Mouse

Para poder tratar os eventos gerados pelo mouse (**cliques do mouse**) é necessário criar uma função para essa tarefa. Essa função deve ter a seguinte sintaxe:

```
void MouseClickInput(int button, int state, int x, int y)
{
       /* Bloco de Comandos */
}
```

Também é necessário indicar que essa é a sua função para tratar eventos de clique do mouse usando a função SetMouseClickInput:

```
graphics.SetMouseClickInput(MouseClickInput);
```

Dessa forma, sempre que um botão do mouse for pressionado a função **MouseClickInput** será executada e o parâmetro **button** indicará qual botão foi pressionado. Os parâmetros **x** e **y** indicam a posição na tela em que mouse estava quando o clique foi realizado.

Exemplo:

```
void MouseClickInput(int button, int state, int x, int y)
{
  if ((button == MOUSE_LEFT_BUTTON)&&(state == MOUSE_STATE_DOWN))
  {

     destino_x = x;
     destino_y = y;

  }
}
```

Se o botão esquerdo do mouse foi pressionado

As variáveis destino_x e destino_y recebem a posição x e y do mouse no momento do clique, ou seja, onde o usuário clicou.

Os códigos dos **botões do mouse** são os seguintes:

- MOUSE_LEFT_BUTTON
- MOUSE_MIDDLE_BUTTON
- MOUSE_RIGHT_BUTTON

Os **estados** que estes botões podem assumir são os seguintes:

- MOUSE_STATE_DOWN
- MOUSE_STATE_UP

9.3 Tratando o Movimento do Mouse

Para poder tratar os eventos de movimento gerados pelo mouse é necessário criar uma função para essa tarefa. Essa função deve ter a seguinte sintaxe:

```
void MouseMotionInput(int x, int y)
{
       /* Bloco de Comandos */
}
```

Também é **necessário indicar** que essa é a sua função para tratar eventos de movimento do mouse usando a função SetMouseClickInput:

```
graphics.SetMouseMotionInput(MouseMotionInput);
```

Dessa forma, sempre que o mouse for movimentado pelo usuário a função **MouseClickInput** será executada e os parâmetros **x** e **y** indicaram a posição do mouse na tela.

Exemplo:

```
void MouseMotionInput(int x, int y)
{
  mouse_x = x;
  mouse_y = y;
}
```

As variáveis mouse_x e mouse_y recebem a posição x e y do mouse, ou seja, o local onde o usuário está com o cursor do mouse.

9.5 Tratando Cliques do Mouse Sobre uma Imagem

Para poder tratar os eventos de clique do mouse sobre uma determinada imagem é necessário definir uma função para essa tarefa. A função para tratar esse evento deve ter a seguinte sintaxe:

```
void MouseClickMinhaImagem(int button, int state, int x, int y)
{
       /* Bloco de Comandos */
}
```

Também é **necessário indicar** que essa é a sua função para tratar eventos de clique do mouse sobre a imagem em questão usando o comando SetOnClick:

```
MinhaImagem.SetOnClick(MouseClickMinhaImagem);
```

Dessa forma, sempre que o usuário clicar sobre a imagem "MinhaImagem", a função **MouseClickMinhaImagem** será executada e o parâmetro **button** indicará qual botão foi pressionado. Os parâmetros **x** e **y** indicam a posição na tela relativa a imagem em que mouse estava quando o clique foi realizado.

Exemplo:

```
void MouseClickMinhaImagem(int button, int state, int x, int y)
{
  carregando_imagem = true;
}
```

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

```
Image minha_imagem;

void MouseClickMinhaImagem(int button, int state, int x, int y)
{
  clicou_na_imagem = true;
}

int main(void)
{
  minha_imagem.LoadPNGImage("Marvin.png");
  minha_imagem.SetPosition(0,100,256,256);
  minha_imagem.SetOnClick(MouseClickMarvin);
}
```

PlayLib

Educational Game Programming Library

CREATED BY

Edirlei Soares de Lima

elima@inf.puc-rio.br