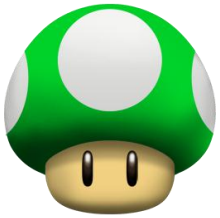
Introdução a Computação

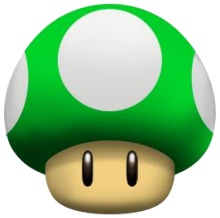
Aula 11 – Interface Gráfica

Edirlei Soares de Lima
<elima@inf.puc-rio.br>



Biblioteca Gráfica

- **Conjunto de funções** para criação e manipulação de formas geométricas, imagens, janelas...
- Baseada na API **OpenGL**.
- Pode ser usada para criação de **jogos 2D**, **simulações**, **animações** e outros aplicativos.
- **Desenvolvida especialmente para esse curso!**



Estrutura de um Programa

```
#include "Graphics.h"

using namespace GraphicsLib;

Graphics graphics;

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

  graphics.StartMainLoop();

  return 0;
}
```

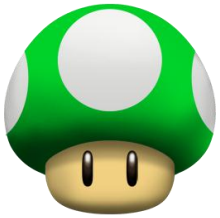
Inclusão da Biblioteca gráfica

Indicação de que as funções da biblioteca gráfica serão usadas no contexto desse programa

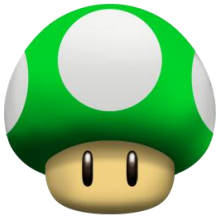
Instância de um objeto do tipo **Graphics** que permite o uso das funções gráficas.

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

Inicia o **Loop principal** do programa



Resultado do Programa Anterior?

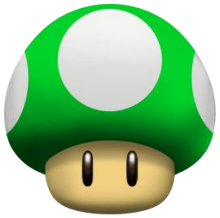


Loop Principal

- 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.



Definindo um Loop Principal

```
void MainLoop()
{
  graphics.SetColor(0,255,0);

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

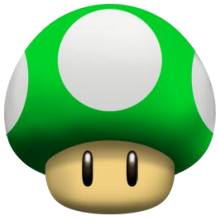
Função que será usada como **Loop Principal** do programa

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

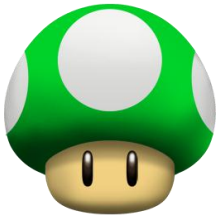
Desenha um **retângulo** preenchido iniciando na posição (100,100) e indo até (200,400)

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

  graphics.SetMainLoop(MainLoop);

  graphics.StartMainLoop();

  return 0;
}
```

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

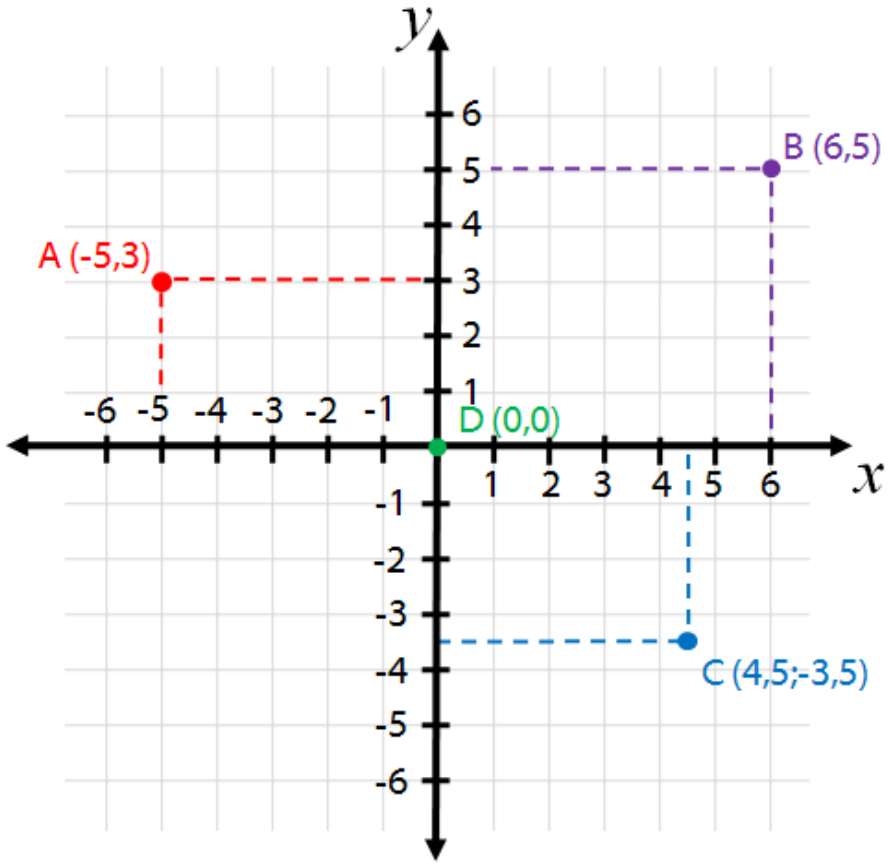


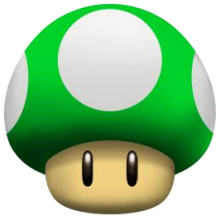
Resultado do Programa Anterior?



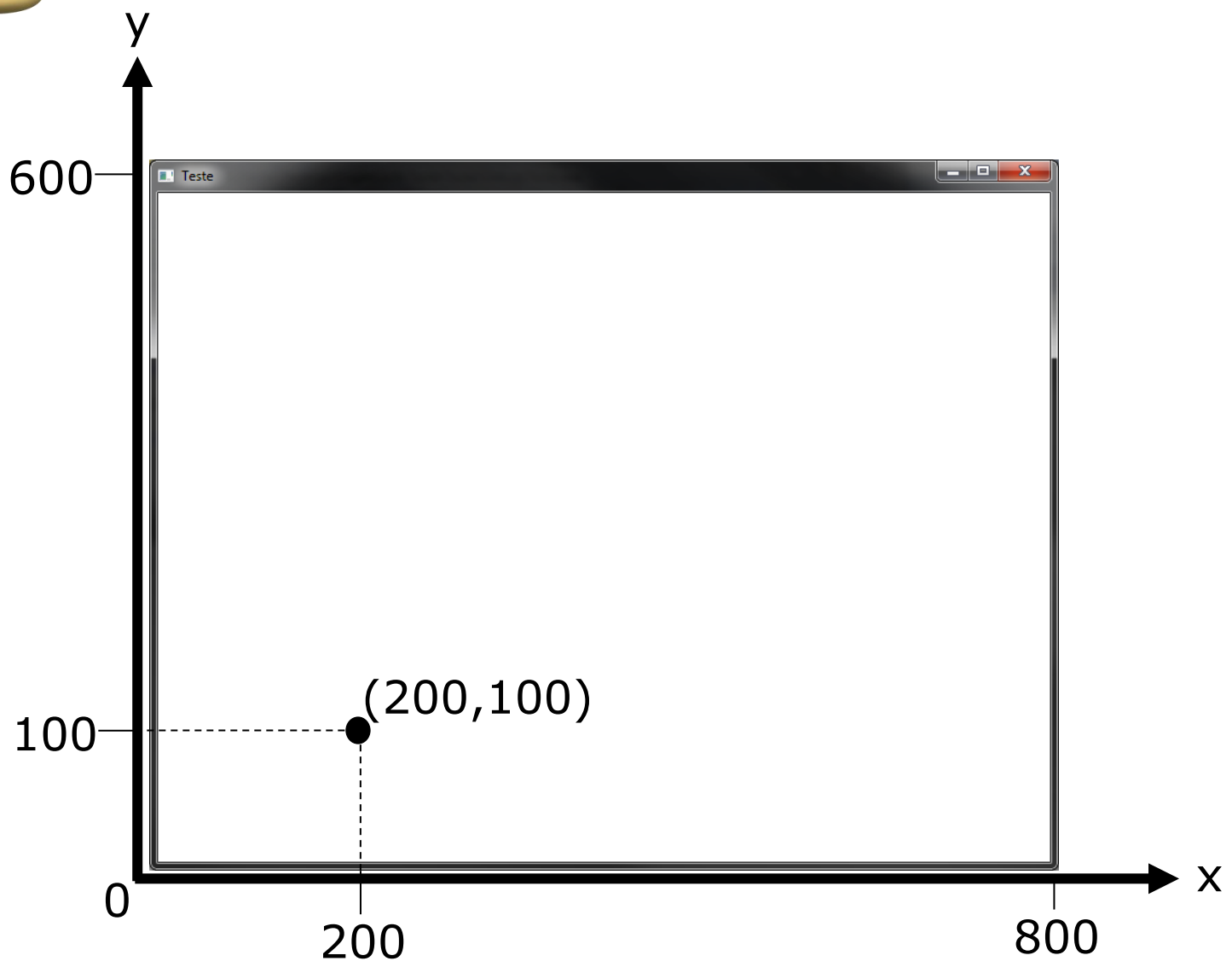
Coordenadas de Tela

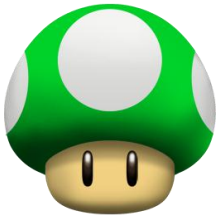
- Sistema de Coordenadas Cartesiano
- Duas dimensões (2D)
- Coordenas X e Y





Coordenadas de Tela





Desenho de Primitivas Geométricas

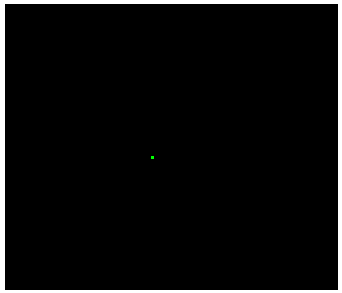
Ponto:

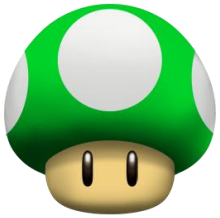
```
void DrawPoint2D(int x, int y);
```

Exemplo:

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

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





Desenho de Primitivas Geométricas

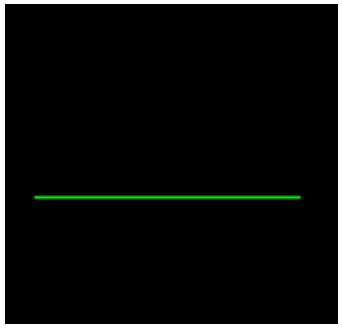
Linha:

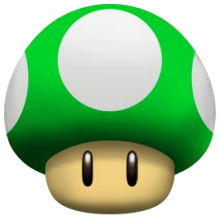
```
void DrawLine2D(int x1, int y1, int x2, int y2);
```

Exemplo:

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

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





Desenho de Primitivas Geométricas

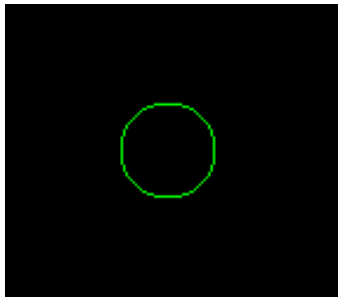
Círculo:

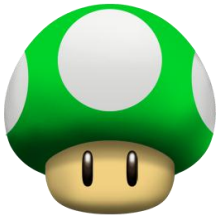
```
void DrawCircle2D(int x, int y, int radius);
```

Exemplo:

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

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





Desenho de Primitivas Geométricas

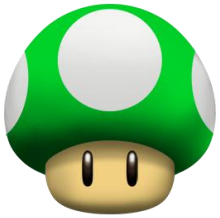
Círculo Preenchido:

```
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.



Desenho de Primitivas Geométricas

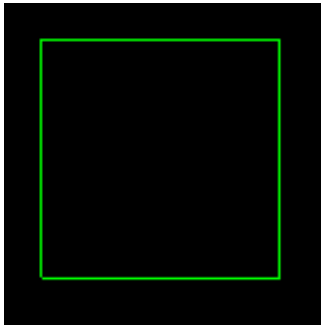
Retângulo:

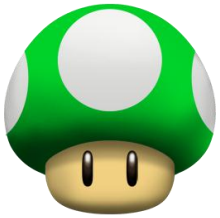
```
void DrawRectangle2D(int x1, int y1, int x2, int y2);
```

Exemplo:

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

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





Desenho de Primitivas Geométricas

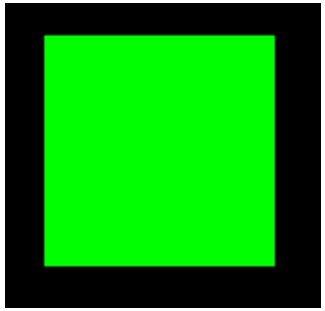
Retângulo Preenchido:

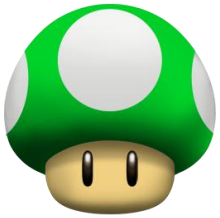
```
void FillRectangle2D(int x1, int y1, int x2, int y2);
```

Exemplo:

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

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





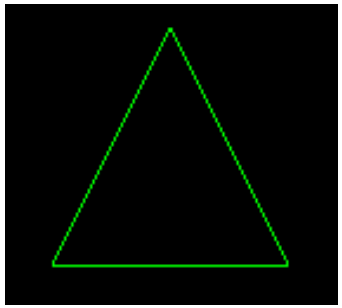
Desenho de Primitivas Geométricas

Triângulo:

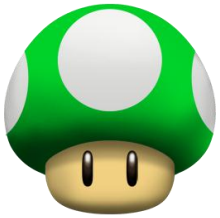
```
void DrawTriangle2D(int x1, int y1, int x2, int y2, int x3, int y3);
```

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)



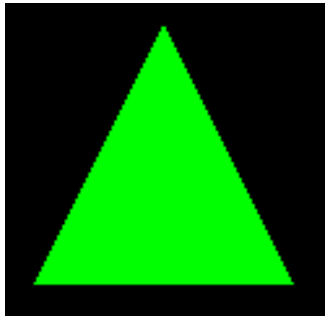
Desenho de Primitivas Geométricas

Triângulo Preenchido:

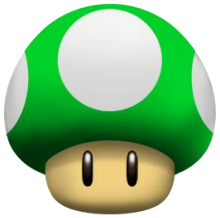
```
void FillTriangle2D(int x1, int y1, int x2, int y2, int x3, int y3);
```

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)



Desenho de Primitivas Geométricas

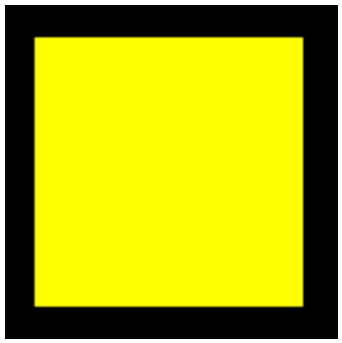
Modificando a Cor:

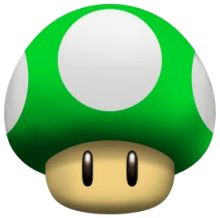
```
void SetColor(float r, float g, float b);
```

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.

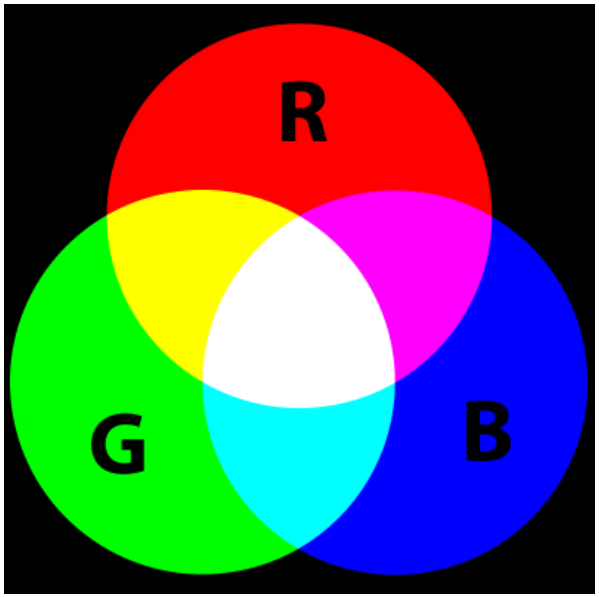




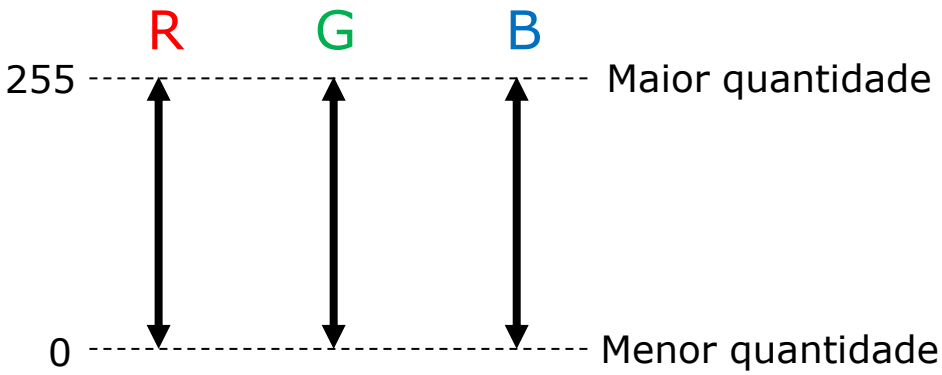
Desenho de Primitivas Geométricas

- **Formato de cor RGB:**

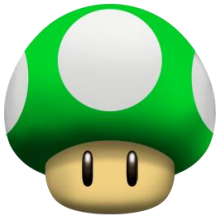
R = Red (Vermelho)
G = Green (Verde)
B = Blue (Azul)



- **Escala RGB:**



Não sabe o valor RGB da cor que você quer?
http://www.colorpicker.com/



Desenho de Primitivas Geométricas

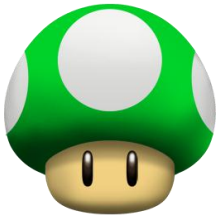
Modificando a Cor do Fundo da Tela:

```
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.



Desenho de Primitivas Geométricas

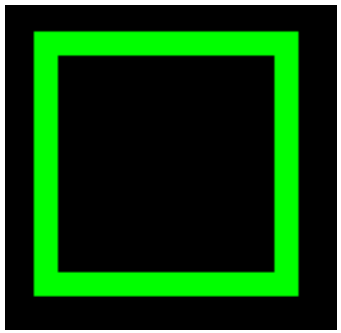
Modificando a Largura das Linhas:

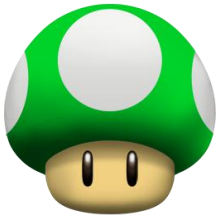
```
void SetLineWidth(float width);
```

Exemplo:

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

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





Desenho de Primitivas Geométricas

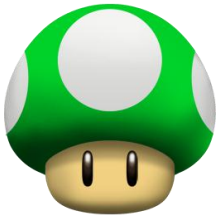
- **Escrevendo um Texto na Tela:**

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

Exemplo:

```
graphics.DrawText2D(100, 100, "Hello World!");
```

Escreve "Hello World!" na posição (100, 100) da tela



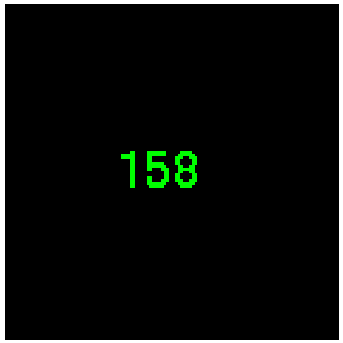
Desenho de Primitivas Geométricas

- **Escrevendo o valor de uma variável inteira na Tela:**

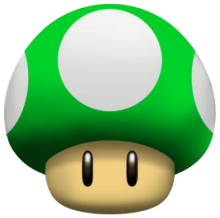
```
void DrawTextInt2D(int x, int y, int value);
```

Exemplo:

```
graphics.DrawTextInt2D(100, 100, MinhaVariavel);
```



Escreve o valor atual armazenado na variável inteira "MinhaVariavel" na posição (100, 100) da tela



Desenho de Primitivas Geométricas

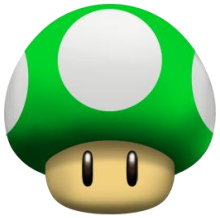
- **Escrevendo o valor de uma variável float na Tela:**

```
void DrawTextFloat2D(int x, int y, float value);
```

Exemplo:

```
graphics.DrawTextFloat2D(100, 100, Valor3);
```

Escreve o valor atual armazenado na variável float "Valor3" na posição (100, 100) da tela



Outras Funções

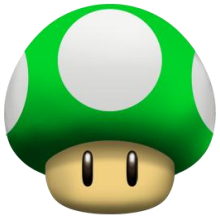
- **Criar a Janela do Programa:**

```
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”



Outras Funções

- **Executando o programa em tela cheia:**

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

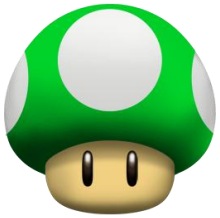
Exemplo:

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

Coloca o programa em tela cheia

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

Remove o programa da tela cheia



Outras Funções

- **Verificando a Velocidade de Execução do Programa:**

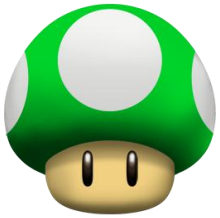
```
float GetFPS();
```

Exemplo:

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

Coloca o número de frames por segundo na variável fps

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



Outras Funções

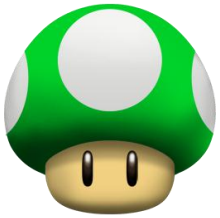
- **Verificando a Velocidade de Execução do Programa:**

```
float GetElapsedTime();
```

Exemplo:

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

Calcula o deslocamento em X de forma independente da taxa de frames por segundo. Isso é muito importante, pois permite que a velocidade do deslocamento seja independente da velocidade que o jogo está sendo executado.



Outras Funções

- **Verificando a Largura e a Altura da Tela:**

```
int GetScreenWidth();

int GetScreenHeight();
```

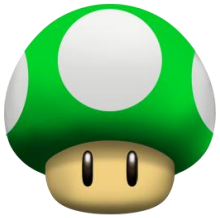
Exemplo:

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

Coloca a largura da tela na variável width

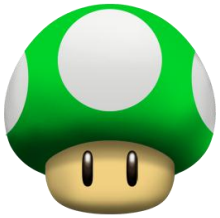
```
height = graphics.GetScreenHeight();
```

Coloca a altura da tela na variável height



Desenhando Imagens

- **Para desenhar uma imagem na tela é necessário:**
  - **(1)** Criar uma variável do tipo **Image**.
  - **(2)** Carregar a imagem do arquivo usando o comando **LoadPNGImage**.
  - **(3)** Desenhar efetivamente a imagem na tela usando o comando **DrawImage2D**.



Desenhando Imagens

- **(1) Criar uma variável do tipo Image:**

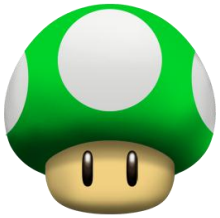
```
Image minha_imagem;
```

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

Exemplo:

```
#include "Graphics.h"
using namespace GraphicsLib;

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.



Desenhando Imagens

- **(2) Carregar a imagem do arquivo usando o comando LoadPNGImage:**

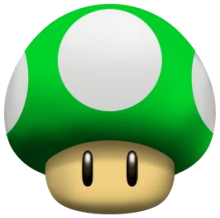
```
minha_imagem = graphics.LoadPNGImage("Mario.png");
```

Exemplo:

```
int main(void)
{
...
  minha_imagem = graphics.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.



Desenhando Imagens

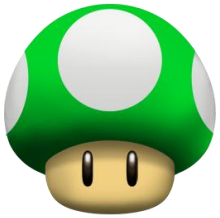
- **(3) Desenhar efetivamente a imagem na tela usando o comando DrawImage2D.**

```
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.



Desenhando Imagens

- **Carregando uma Imagem:**

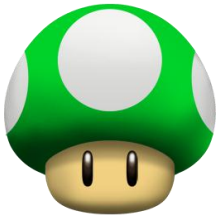
```
Image LoadPNGImage(char *filename);
```

Exemplo:

```
Image mario;
mario = graphics.LoadPNGImage("Mario.png");
```

Declaração da variável do tipo Image que vai armazenar a imagem

Carrega o arquivo "Mario.png" para a variável "mario"



Desenhando Imagens

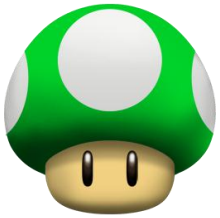
- **Desenhando Imagens na Tela:**

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

Exemplo:

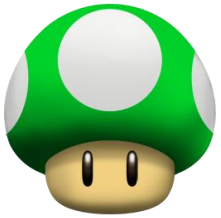
```
graphics.DrawImage2D(200, 200, 256, 256, mario);
```

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



Desenhando Imagens

- **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.



Tratando Entradas do Teclado

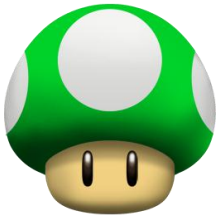
- Para poder tratar os eventos gerados pelo teclado (**teclas sendo pressionadas**) é necessário criar uma função para essa tarefa.

- Essa função deve ter a seguinte sintaxe:

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

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

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



Tratando Entradas do Teclado

- Dessa forma, sempre que uma tecla normal do teclado for pressionada a função **KeyboardInput** será executada e o parâmetro key indicará qual tecla foi pressionada.
- **Exemplo:**

```
void KeyboardInput(unsigned char key, int x, int y)
{
  if (key == 'f')
  {
    graphics.SetFullscreen(true);
  }
  if (key == 'w')
  {
    posicao_personagem_x = posicao_personagem_x + 2;
  }
  if (key == KEY_ESC)
  {
    exit(0);
  }
}
```

Se a letra f for pressionada

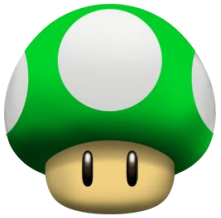
Coloca o programa em tela cheia

Se a letra w for pressionada

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

Se a tecla esc for pressionada

Fecha o programa



Tratando Entradas do Teclado

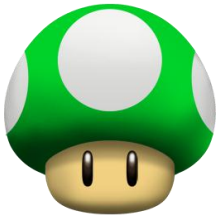
- Algumas **teclas especiais**, como por exemplo as setas direcionais do teclado, requerem o uso de outra função especifica para elas.

- Essa função deve ter a seguinte sintaxe:

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

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

```
graphics.SetKeyboardSpecialInput(KeyboardSpecialInput);
```



Tratando Entradas do Teclado

- Dessa forma, sempre que uma tecla especiais do teclado for pressionada a função **KeyboardSpecialInput** será executada e o parâmetro key indicará qual tecla foi pressionada.

- **Exemplo:**

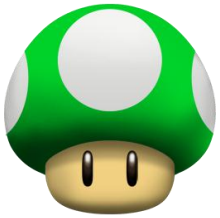
```
void KeyboardSpecialInput(int key, int x, int y)
{
  if (key == KEY_LEFT)
  {
    posicao_personagem_x = posicao_personagem_x - 2;
  }
  if (key == KEY_RIGHT)
  {
    posicao_personagem_x = posicao_personagem_x + 2;
  }
}
```

Se a tecla direcional esquerda for pressionada

Decrementa em -2 uma variável que representa a posição de um personagem

Se a tecla direcional direita for pressionada

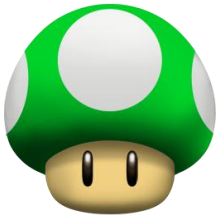
Incrementa em +2 uma variável que representa a posição de um personagem



Códigos das Teclas Especiais

- KEY_LEFT
- KEY_UP
- KEY_RIGHT
- KEY_DOWN
- KEY_PAGE_UP
- KEY_PAGE_DOWN
- KEY_HOME
- KEY_END
- KEY_INSERT
- KEY_ESC

- KEY_F1
- KEY_F2
- KEY_F3
- KEY_F4
- KEY_F5
- KEY_F6
- KEY_F7
- KEY_F8
- KEY_F9
- KEY_F10
- KEY_F11
- KEY_F12



Tratando Cliques do Mouse

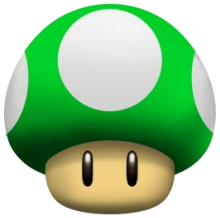
- Para poder tratar os eventos gerados pelo mouse (**cliques do mouse**) é necessário criar uma função para essa tarefa.

- 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:

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



Tratando Cliques do Mouse

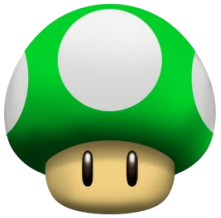
- Dessa forma, sempre que um botão do mouse for pressionado a função **MouseClickInput** será executada e o parâmetro button indicará qual botão foi pressionado. Os parâmetros x e y indicam a posição na tela em que mouse estava quando o clique foi realizado.

- **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 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.



Tratando o Movimento do Mouse

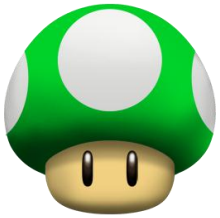
- Para poder tratar os eventos de movimento gerados pelo mouse é necessário criar uma função para essa tarefa.

- 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:

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



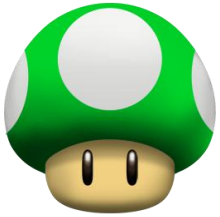
Tratando Cliques do Mouse

- 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.



Códigos da Teclas do Mouse

Botões:

- MOUSE_LEFT_BUTTON
- MOUSE_MIDDLE_BUTTON
- MOUSE_RIGHT_BUTTON

Estados:

- MOUSE_STATE_DOWN
- MOUSE_STATE_UP