Ficha nº 8 – Programação gráfica orientada a eventos

Âmbito da matéria

Os exercícios desta ficha abordam:

- Criação da janela principal de uma aplicação gráfica e o seu tratamento de eventos em Win32.
- Tratamento de eventos relacionados com diversos meios de interação com o utilizador tais como teclado e rato.
- Criação e carregamento de recursos tais como: ícones, cursor, menu, bitmaps, caixas de diálogo.
- Tratamento de eventos relacionados com os menus e caixas de diálogo (caixas de texto, etiquetas, botões, imagens e lista de *strings*).
- Conceito e aplicação de *double buffering*.

Pressupostos:

- Conhecimento de algoritmia e de programação em C / C++.
- Conhecimentos de conceitos de base em SO (1º semestre) e das aulas anteriores.

Referências bibliográficas

- Material das aulas teóricas
- Capítulos 2-5 e 7-9 do Livro Windows NT 4 Programming
- MSDN:

  Introduction to Windows Programming in C++ .... https://msdn.microsoft.com/en-us/library/ff381398(v=vs.85).aspx

  Mouse Input .....https://msdn.microsoft.com/en-us/library/gg153549(v=vs.85).aspx

  Keyboard Input .....https://msdn.microsoft.com/en-us/library/gg153546(v=vs.85).aspx

  Mouse Movement .....https://msdn.microsoft.com/en-us/library/gg153550(v=vs.85).aspx

  Dialog Boxes .... https://msdn.microsoft.com/en-us/library/windows/desktop/ms632588(v=vs.85).aspx

  Menus ..... https://msdn.microsoft.com/en-us/library/windows/desktop/ms646977(v=vs.85).aspx

  Windows Graphics Device Interface (GDI) .....

  ..... https://msdn.microsoft.com/en-us/library/dd145203(v=vs.85).aspx

  Tutorial online: Win32 Fundamentals http://www.functionx.com/win32/Lesson01.htm

  Events http://www.functionx.com/win32/Lesson05.htm

  Object-Oriented Win32 ..... http://www.functionx.com/win32/Lesson06.htm

Introdução e contexto

A interface gráfica em Win32 é baseada no conceito de **evento**: o sistema envia às aplicações ***mensagens*** relativas à ocorrência de um determinado acontecimento. As aplicações, por sua vez, reencaminham o evento à janela a que dizem respeito.
Os acontecimentos (eventos) estão relacionados com aquilo que pode ocorrer no sistema, em particular, de interesse para uma janela ou componente gráfico: movimento de rato, operação de botões do rato, operação de uma tecla (pressionar, libertar), operações sobre janelas (deslocar, minimizar, redimensionar), operações de gestão da janela (reatualizar, repor o fundo), ativação de controlos (botão, *scroll bars*, menus). A lista de eventos possíveis é muito grande.

O processamento de eventos segue as seguintes linhas gerais:

- A receção de um evento corresponde à invocação de uma função *callback* designada por *função da janela*.
- A função da janela tem um protótipo específico e é identificada na criação da janela.
- O atendimento do evento corresponde à execução da função *callback* da janela. Os parâmetros desta função descrevem o evento e os pormenores acerca desse evento (exemplo: o rato foi deslocado – os pormenores são as coordenadas).
- Cada evento é identificado por um código. Esse código é um dos parâmetros da função da janela.
- A aplicação deverá ter um ciclo de processamento de eventos. Neste ciclo a aplicação recebe o evento e reencaminha-o para a janela (para a função de tratamento de eventos dessa janela). Este ciclo deve existir na mesma *thread* que criou a janela.
- A execução da função *callback* é feita no contexto da *thread* que tem o ciclo de processamento de eventos. Durante a execução da função mais nenhum evento é processado. Por essa razão o atendimento do evento deve ser o mais breve possível para permitir que a execução regresse ao ciclo de processamento de eventos para atender outro evento.

A lógica de uma interface gráfica orientada a eventos tem uma lógica diferente das interfaces em modo texto do paradigma das consolas:

- Não existem compassos de espera específicos. Considerar que se aguarda que o utilizador escreva um conjunto de caracteres apenas faz sentido em sentido abstrato.
- Qualquer *input* pode surgir a qualquer momento desde que haja um evento capaz de o encapsular e uma função de janela com código para reagir a ele. Enquanto o programa regista teclas podem surgir outros acontecimentos (ativação de botões, menus, etc.) que podem ser do interesse do programa.
- O programa deixa de impor uma ordem específica de eventos relacionados com o utilizador. Em vez disso vai registando os vários acontecimentos e quando reúne a informação que precisa então permite avançar.

Construção das interfaces:
Apesar de possível detetar e reagir a todos os eventos do rato e teclado e construir de raiz uma interface, normalmente não se faz dessa forma. Em vez disso usam-se componentes já existentes:

- São específicos a uma determinada funcionalidade: edição de texto, botões, menus, etc.
- São razoavelmente prontos a usar, sendo necessária apenas alguma configuração (por exemplo, texto a apresentar).
- Possuem comportamento autónomo e reagem a determinados eventos consoante a sua natureza e finalidade: teclado para edição de texto, rato para botões, etc.

Existem também alguns tipos específicos de janela com alguma funcionalidade autónoma já semi-pronta a usar: por exemplo: *dialog boxes* e *message boxes*. As janelas são genéricas ao passo que *dialog boxes* e *message boxes* tem um uso muito mais específico. Os componentes são normalmente adicionados a *dialog boxes*, mas também a janelas genéricas.
Existem diversas técnicas para ocultar este funcionamento através de uma *framework* orientada a objetos. A forma como se constroem poderá ser abordada nas aulas teórica, mas ultrapassa o objetivo das aulas práticas.

-> Esta introdução (resumo) não dispensa as aulas teóricas sobre este assunto.

Exercícios

Os exercícios devem ser resolvidos recorrendo a projetos do tipo Win32 Application, inicialmente vazios. Os ficheiros podem ter extensão *.cpp*, mas deve optar por *.c* (a não ser que sejam mesmo necessários mecanismos de C++).

1. Crie um novo projeto para uma aplicação gráfica (Win32 Application) no Visual Studio.

   a) Acrescente o código da listagem 1 que permite criar uma janela simples.

   b) Localize no código o ponto onde são indicados: o ícone, título, cor de fundo, localização e tamanho da janela criada. Altere esses dados para outros valores e experimente o programa.

2. Altere o fim do programa do exercício anterior de modo a obrigar o utilizador a confirmar que deseja mesmo fechar a janela através de uma Caixa de Mensagens (*message box*).

3. Memorize a letra correspondente à última tecla premida e apresente-a nas coordenadas em que o rato é premido.

4. Simule o comportamento de uma *edit box* através de uma janela. De cada vez que se escreve um novo caracter a palavra deve ser acrescentada e aparecer maior na janela. À medida que é premido "Backspace" os últimos caracteres vão sendo eliminados

5. O que é necessário fazer ao programa anterior para que, de cada vez que se minimize a janela, o fundo da janela não seja perdido? Aplique-o no seu código.

6. Escreva um programa que carregue uma imagem bitmap e apresente o seu conteúdo na janela principal. Pode usar o ficheiro bitmap qualquer. Sugerem-se as seguintes dimensões: Bitmap entre 100 e 200 pixels de largura (quadrado, se quiser), e uma janela bastante maior, por exemplo, 600x600. A imagem deverá permanecer na janela mesmo que a oculte o redimensione. Deve minimizar os acessos ao disco e o esforço computacional.

   a) Escreva o programa fazendo a imagem aparecer no centro (aproximado) da janela. Caso redimensione a janela, a imagem deverá ser reposicionada no novo centro.

   b) Acrescente a funcionalidade de movimento ao programa: a imagem desloca-se na horizontal, pixel a pixel, invertendo o sentido do movimento sempre que toca na extremidade da janela (deve ter em atenção que a janela pode ser redimensionada, mas não permita que fique mais pequena que o tamanho da imagem).
   A velocidade inicial é de 2 pixels a cada 30 milissegundos (meras sugestões). A velocidade pode ser aumentada com a tecla '+' e diminuída com a tecla '-'. O movimento pode ser pausado / recomeçado com a tecla espaço. Ao recomeçar, o movimento continua na posição, sentido e velocidade que tinha. Deve minimizar o esforço computacional.

   c) Nas velocidades mais elevadas deve acontecer algum efeito de cintilação. Resolva esse problema recorrendo à técnica de *double buffer*. É importante que minimize o esforço computacional.

7. (TPC / Revisão de conceitos) Construa uma aplicação que permita desenhar círculos de diâmetro 20 nas coordenadas em que o rato é pressionado. Resolva o exercício através das seguintes etapas:

   a) Crie uma aplicação com uma janela que não exibe nenhum comportamento. A janela tem um título (à sua escolha) e uma cor de fundo que não é a cor branca (à sua escolha).

   b) Acrescente a funcionalidade de deteção do movimento do rato. Desenhe uma circunferência (oca) de diâmetro 20 centrado no rato que acompanha sempre o movimento do rato.

   c) Acrescente a deteção do pressionar do rato, desenhando um círculo (com cor sólida) nas coordenadas em que o rato é pressionado com o botão esquerdo. O círculo fica definitivamente desenhado na janela.

   d) Faça com que a dimensão do círculo seja definida pelo utilizador: depois de pressionar o rato, o utilizador arrasta o rato com o botão premido. Quando larga o botão, o círculo fica com a dimensão correspondente ao movimento percorrido pelo rato. O círculo é visível durante o movimento.

   e) Verifique o que acontece se ocultar/redimensionar a janela e depois a restaurar. O círculo desenhado volta a aparecer? Garanta que sim.

   f) Substitua a cor sólida por uma textura baseada num bitmap. Para esse efeito, use um ficheiro .bmp qualquer.

   g) Acrescente o mecanismo que permite memorizar todos os círculos. Assuma o máximo de 500 círculos.

   h) Depois de desenhar muitos círculos, começa a notar-se um efeito de cintilação. Resolva isso com *double buffering* usando *MemoryDC*.

   i) Acrescente outra janela que veja também os mesmos círculos que estão a ser desenhados na outra. Deve ser possível desenhar novos círculos em qualquer das janelas, e o novo círculo aparece imediatamente em ambas.

   j) Modifique o programa de forma que cada janela tenha os seus próprios dados, ou seja, conjuntos de círculos independentes. Nota: a solução não passa por acrescentar uma nova classe de janela (se fizer isso, neste cenário, estará errado).

   k) Generalize a sua solução para qualquer número de janelas. Pode acontecer que o seu programa já esteja nessa situação e, se for esse o caso, passe para os exercícios seguintes.

8. Considere um simulador de multibanco com interface gráfica. Os dados do banco são fixos e correspondem ao código apresentado na listagem 2. Pretende-se criar uma interface para agir sobre esses dados usando recursos definidos no Visual Studio: *dialog boxes*, *list boxes*, menus, etc.

   a) De forma a preparar a interface gráfica deste exercício, crie o seguinte conjunto de recursos que serão carregados na janela principal (Fig.1-3): ícone pequeno e médio, menu e 2 caixas de diálogo tais como mostram as figuras seguintes.

Fig.1 Janela Principal

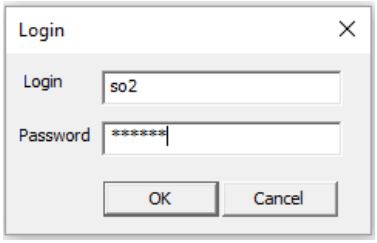

Fig.2 Caixa de diálogo de login

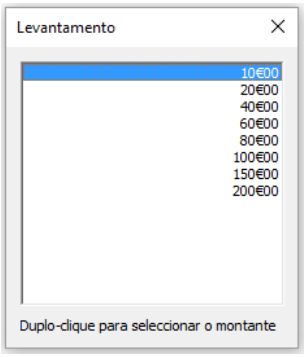

Fig.3 Caixa de diálogo de levantamento

b) Construa a janela principal, e todo o código associado a essa janela e aos recursos definidos na alínea anterior, e o restante código para suportar as seguintes operações: login, consulta de saldo, levantamento.

c) Acrescente opções e código para depósito, *logout*, saída do programa, consulta de operações efetuadas. O valor a depositar deverá ser introduzido escrevendo o número (ao contrário do que acontece com o levantamento, que é feito usando uma lista de valores). Se for preciso defina mais uma caixa de diálogo.

d) Acrescente ao seu programa os seguintes recursos: imagem e fundo na janela principal, cursor com o símbolo "€" quando o rato anda em cima da janela principal.

e) Acrescente ao seu programa os seguintes recursos: atalhos de teclas (aceleradores) e tabela de *strings* com todas as mensagens apresentadas pelo programa.

9. (Revisão) Construa uma calculadora com interface gráfica. Este exercício dispensa explicações alongadas e os requisitos são: deve suportar as operações básicas, números com ponto decimal, deve permitir a introdução de números por teclas (botões) ou escrevendo diretamente com o teclado. Sugestão: neste exercício vai ter que considerar a questão do *focus*.

Listagem de Programas

-> Estas listagens são fornecidas também em ficheiros independentes de texto, de forma a facilitar a cópia do código para o projeto a desenvolver na aula.
É recomendado que se usem os ficheiros txt em vez do código que está no ficheiro pdf.

Listagem 1

```
#include <windows.h>
#include <tchar.h>

/* ================================================= */
/* Programa base (esqueleto) para aplicações Windows     */
/* ================================================= */

// Cria uma janela de nome "Janela Principal" e pinta fundo de branco
// Modelo para programas Windows:
//  Composto por 2 funções:
//   WinMain()     = Ponto de entrada dos programas windows
//              1) Define, cria e mostra a janela
//              2) Loop de recepção de mensagens provenientes do Windows
//     TrataEventos()= Processamentos da janela (pode ter outro nome)
//              1) É chamada pelo Windows (callback)
//              2) Executa código em função da mensagem recebida

LRESULT CALLBACK TrataEventos(HWND, UINT, WPARAM, LPARAM);

// Nome da classe da janela (para programas de uma só janela, normalmente este nome é
// igual ao do próprio programa) "szprogName" é usado mais abaixo na definição das
// propriedades do objecto janela

TCHAR szProgName[] = TEXT("Base");

// ================================================================================
// FUNÇÃO DE INÍCIO DO PROGRAMA: WinMain()
// ================================================================================
// Em Windows, o programa começa sempre a sua execução na função WinMain()que desempenha
// o papel da função main() do C em modo consola WINAPI indica o "tipo da função" (WINAPI
// para todas as declaradas nos headers do Windows e CALLBACK para as funções de
// processamento da janela)
// Parâmetros:
//   hInst: Gerado pelo Windows, é o handle (número) da instância deste programa
//   hPrevInst: Gerado pelo Windows, é sempre NULL para o NT (era usado no Windows 3.1)
//   lpCmdLine: Gerado pelo Windows, é um ponteiro para uma string terminada por 0
//              destinada a conter parâmetros para o programa
//   nCmdShow:  Parâmetro que especifica o modo de exibição da janela (usado em
//            ShowWindow()

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmdLine, int nCmdShow) {
  HWND hWnd;          // hWnd é o handler da janela, gerado mais abaixo por CreateWindow()
  MSG lpMsg;          // MSG é uma estrutura definida no Windows para as mensagens
  WNDCLASSEX wcApp;   // WNDCLASSEX é uma estrutura cujos membros servem para
                     // definir as características da classe da janela
```

```
// ================================================================================
// 1. Definição das características da janela "wcApp"
//    (Valores dos elementos da estrutura "wcApp" do tipo WNDCLASSEX)
// ================================================================================
  wcApp.cbSize = sizeof(WNDCLASSEX);      // Tamanho da estrutura WNDCLASSEX
  wcApp.hInstance = hInst;                // Instância da janela actualmente exibida
                                          // ("hInst" é parâmetro de WinMain e vem
                                          // inicializada daí)
  wcApp.lpszClassName = szProgName;       // Nome da janela (neste caso = nome do programa)
  wcApp.lpfnWndProc = TrataEventos;       // Endereço da função de processamento da janela
                                          // ("TrataEventos" foi declarada no início e
                                          // encontra-se mais abaixo)
  wcApp.style = CS_HREDRAW | CS_VREDRAW;  // Estilo da janela: Fazer o redraw se for
                                          // modificada horizontal ou verticalmente

  wcApp.hIcon = LoadIcon(NULL, IDI_APPLICATION);   // "hIcon" = handler do ícon normal
                                                   // "NULL" = Icon definido no Windows
                                                   // "IDI_AP..." Ícone "aplicação"
  wcApp.hIconSm = LoadIcon(NULL, IDI_INFORMATION); // "hIconSm" = handler do ícon pequeno
                                                   // "NULL" = Icon definido no Windows
                                                   // "IDI_INF..." Ícon de informação
  wcApp.hCursor = LoadCursor(NULL, IDC_ARROW);     // "hCursor" = handler do cursor (rato)
                                      // "NULL" = Forma definida no Windows
                                      // "IDC_ARROW" Aspecto "seta"
  wcApp.lpszMenuName = NULL;               // Classe do menu que a janela pode ter
                                      // (NULL = não tem menu)
  wcApp.cbClsExtra = 0;                    // Livre, para uso particular
  wcApp.cbWndExtra = 0;                    // Livre, para uso particular
  wcApp.hbrBackground =(HBRUSH)GetStockObject(WHITE_BRUSH);
// "hbrBackground" = handler para "brush" de pintura do fundo da janela. Devolvido por
// "GetStockObject".Neste caso o fundo será branco

// ================================================================================
// 2. Registar a classe "wcApp" no Windows
// ================================================================================
  if (!RegisterClassEx(&wcApp))
    return(0);

// ================================================================================
// 3. Criar a janela
// ================================================================================
  hWnd = CreateWindow(
     szProgName,               // Nome da janela (programa) definido acima
     TEXT("Exemplo de Janela Principal em C"), // Texto que figura na barra do título
     WS_OVERLAPPEDWINDOW,  // Estilo da janela (WS_OVERLAPPED= normal)
     CW_USEDEFAULT,        // Posição x pixels (default=à direita da última)
     CW_USEDEFAULT,        // Posição y pixels (default=abaixo da última)
     CW_USEDEFAULT,        // Largura da janela (em pixels)
     CW_USEDEFAULT,        // Altura da janela (em pixels)
     (HWND) HWND_DESKTOP,  // handle da janela pai (se se criar uma a partir de
                           // outra) ou HWND_DESKTOP se a janela for a primeira,
                           // criada a partir do "desktop"
     (HMENU) NULL,              // handle do menu da janela (se tiver menu)
     (HINSTANCE) hInst,         // handle da instância do programa actual ("hInst" é
                           // passado num dos parâmetros de WinMain()
     0);                   // Não há parâmetros adicionais para a janela
```

```
// ========================================================================
// 4. Mostrar a janela
// ========================================================================
  ShowWindow(hWnd, nCmdShow);    // "hWnd"= handler da janela, devolvido por
                            // "CreateWindow"; "nCmdShow"= modo de exibição (p.e.
                            // normal/modal); é passado como parâmetro de WinMain()
  UpdateWindow(hWnd);       // Refrescar a janela (Windows envia à janela uma
                            // mensagem para pintar, mostrar dados, (refrescar)…

// ========================================================================
// 5. Loop de Mensagens
// ========================================================================
// O Windows envia mensagens às janelas (programas). Estas mensagens ficam numa fila de
// espera até que GetMessage(...) possa ler "a mensagem seguinte"
// Parâmetros de "getMessage":
// 1)"&lpMsg"=Endereço de uma estrutura do tipo MSG ("MSG lpMsg" ja foi declarada no
//   início de WinMain()):
//               HWND hwnd       handler da janela a que se destina a mensagem
//               UINT message          Identificador da mensagem
//               WPARAM wParam         Parâmetro, p.e. código da tecla premida
//               LPARAM lParam         Parâmetro, p.e. se ALT também estava premida
//               DWORD time      Hora a que a mensagem foi enviada pelo Windows
//               POINT pt        Localização do mouse (x, y)
// 2)handle da window para a qual se pretendem receber mensagens (=NULL se se pretendem
//   receber as mensagens para todas as janelas pertencentes à thread actual)
// 3)Código limite inferior das mensagens que se pretendem receber
// 4)Código limite superior das mensagens que se pretendem receber

// NOTA: GetMessage() devolve 0 quando for recebida a mensagem de fecho da janela,
//      terminando então o loop de recepção de mensagens, e o programa

  while (GetMessage(&lpMsg,NULL,0,0)) {
    TranslateMessage(&lpMsg);    // Pré-processamento da mensagem (p.e. obter código
                            // ASCII da tecla premida)
    DispatchMessage(&lpMsg);     // Enviar a mensagem traduzida de volta ao Windows, que
                            // aguarda até que a possa reenviar à função de
                            // tratamento da janela, CALLBACK TrataEventos (abaixo)
  }

// ========================================================================
// 6. Fim do programa
// ========================================================================
  return((int)lpMsg.wParam);     // Retorna sempre o parâmetro wParam da estrutura lpMsg
}

// ========================================================================
// FUNÇÃO DE PROCESSAMENTO DA JANELA
// Esta função pode ter um nome qualquer: Apenas é necesário que na inicialização da
// estrutura "wcApp", feita no início de WinMain(), se identifique essa função. Neste
// caso "wcApp.lpfnWndProc = WndProc"
//
// WndProc recebe as mensagens enviadas pelo Windows (depois de lidas e pré-processadas
// no loop "while" da função WinMain()
// Parâmetros:
//         hWnd  O handler da janela, obtido no CreateWindow()
//         messg Ponteiro para a estrutura mensagem (ver estrutura em 5. Loop...
//         wParam     O parâmetro wParam da estrutura messg (a mensagem)
//         lParam     O parâmetro lParam desta mesma estrutura
//
// NOTA:Estes parâmetros estão aqui acessíveis o que simplifica o acesso aos seus valores
```

```
//
// A função EndProc existe um "switch..." com "cases" que descriminam a mensagem
// recebida e a tratar. Estas mensagens são identificadas por constantes (p.e.
// WM_DESTROY, WM_CHAR, WM_KEYDOWN, WM_PAINT...) definidas em windows.h
// ========================================================================

LRESULT CALLBACK TrataEventos(HWND hWnd, UINT messg, WPARAM wParam, LPARAM lParam) {
  switch (messg) {
    case WM_DESTROY:  // Destruir a janela e terminar o programa
                      // "PostQuitMessage(Exit Status)"
     PostQuitMessage(0);
     break;
    default:
      // Neste exemplo, para qualquer outra mensagem (p.e. "minimizar","maximizar","restaurar")
      // não é efectuado nenhum processamento, apenas se segue o "default" do Windows
     return(DefWindowProc(hWnd,messg,wParam,lParam));
     break;  // break tecnicamente desnecessário por causa do return
  }
  return(0);
}
```

Listagem 2

```
typedef struct {
    unsigned int ID;
    TCHAR username[16];
    TCHAR password[16];
    unsigned int saldo;
} cliente;

cliente clientes[3];  // são sempre três clientes.

// inserir dados fixos iniciais para os 3 clientes.

typedef struct {
   unsigned int tipo;       // 1 = depósito, 2 = levantamento
   unsigned int quantia;    // quanto foi levantado / depositado
   usigned int ID;          // quem fez a operação
} operacao;

operação historico[200];
unsigned int numOperacoes;

// este código é apenas uma sugestão para simplificar o problema
```