

GUI grafických aplikací

GUI pro Othello

Aleš Keprt

Ústav informatiky MVŠO

duben 2006, březen 2007, duben 2008, březen 2009, březen 2010, listopad 2010

Windows – Událostmi řízené aplikace

- Každá aplikace má tzv. „frontu zpráv“
- Funkce main() vykonává kód:

```
Message msg;  
while(GetMessage(&msg)) {  
    DispatchMessage(&msg);  
}
```
- GetMessage blokuje až do příchodu zprávy
- DispatchMessage předá zprávu do WndProc
 - WndProc zpracuje zprávu

Jak vypadá WndProc?

```
long* WndProc(HWND hwnd, UINT msg,  
    UINT wparam, ULONG lparam) {  
    switch(msg) {  
        case WM_CLOSE:  
        case WM_PAINT:  
        case WM_KEYDOWN:  
        ...  
    }
```

- hwnd (okno), msg = zpráva + 2 parametry
- Návratová hodnota = odpověď na zprávu

Překreslování grafiky ve Windows

- Když chce aplikace něco překreslit, tak zavolá systémovou funkci `Invalidate()` = zneplatnění
 - Jako parametr určí oblast okna k překreslení
- Systém pošle oknu zprávu `WM_PAINT`
 - Systém invalidované oblasti „sčítá“ a optimalizuje tak překreslování
 - Dokud se okno nepřekreslí, žádné další `WM_PAINT` zprávy nepřijdou
- Zpracování zprávy `WM_PAINT`:
 - Pomocí funkcí GDI si okno překreslí svůj obsah
 - Případně použijeme OpenGL nebo DirectX

Proč je to tak složité?

- Systém má šanci, aby ignoroval invalidace, když má důležitější práci
- Aplikace volá Invalidate při každé změně obsahu – když to dělá moc často, systém nechá překreslit okno jen „občas“ – záleží na rychlosti počítače

Jak se dá kreslit

- **GDI** – standardní rozhraní Windows
 - Výhody: úplně jednoduché
 - Nevýhody: téměř bez HW akcelerace, závislé na rozlišení
- **GDI+** – nahrazuje GDI v .NETu
 - Výhody: nativně objektové, částečně akcelerované
 - Nevýhody: Stále daleko od DX, stále závislé na rozlišení
- **DirectX**
 - Výhody: Nejlepší funkcionality, nejrychlejší běh
 - Plná HW akcelerace, nezávislost na rozlišení
 - Nevýhody: Extrémně složité, neumí formulářové prvky
- **OpenGL** – obdoba DirectX, ale je jednodušší
 - Výhody: multiplatformní, jednodušší než DX
 - Nevýhody: Užší funkcionality, jen 3D grafika, neobjektové
- **WPF (Windows Presentation foundation)** – nástupce GDI+ (.NET 3.0)
 - Výhody: lepší objektový model, již nezávislé na rozlišení

Jaké mohou být problémy

- Nelze ovlivnit, jak často se obraz překresluje
 - Proto přehrávače filmů obcházejí princip invalidace přímým kreslením na obrazovku
- Při rychlém překreslování grafika bliká
 - Double buffering existuje jen v OpenGL a DirectX
 - Obecně se může stát, že hardware zobrazí obraz částečně starý a částečně nový, protože „monitor nepočká“, až připravíme celý nový obraz
 - V GDI řešíme pomocí „kreslení přes bitmapu“
 - GDI+ toto řeší nativně → snadný život programátora
 - Ve Windows Vista a 7 nikdy nic nebliká (Aero UI)

Jakých chyb se vyvarovat?

- Grafiku překreslujte výhradně v obsluze zprávy WM_PAINT (v .NETu je to událost Paint)
- Nikdy nevynucujte překreslení pomocí Update(), ale jen invalidujte okno pomocí Invalidate()
- Grafiku přenášejte do okna najednou; v případě potřeby použijte kreslení do bitmapy
 - PictureBox kreslí vše přes Bitmapu automaticky 😊
- Nepoužívejte OpenGL a DirectX pro triviální programy s jednoduchou grafikou
- Dávejte přednost GDI+ (případně WPF), protože je nativně objektové (Windows.Forms používá GDI+)

Náš případ: GUI pro Othello

- Původně to byla přednáška o GUI pro Dámu 😊
- Máme čtvercovou „šachovnicová“ plochu
- Nové kameny klademe klikáním myši
 - Tj. řešíme i vstup, ale je to velmi jednoduché 😊
- Příklad realizace v GDI+
 - Čtvercová grafická plocha jako jeden objekt
 - Čili plochu nijak nedělíme na políčka
 - Připravíme si obrázky políček a kamenů
 - Překreslujeme vždy celou plochu
 - Zachytáváme klikání myši

Příprava (1.) – typ Barva, obrázky

Enum Barva

Žádná

Černá

Bílá



End Enum

Tyto obrázky jsou od studenta Sebastiana Buška, který projekt Othello implementoval v roce 2010 na KMI PŘF UP Olomouc

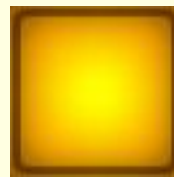
Obrázky políček:

■ Prázdné políčko

■ Minulý tah

■ Možný tah

■ Chybný tah



Příprava (2.) – třídy Hra a Deska

- Okno musí odněkud zjistit, co má kreslit – k tomu máme třídu Hra
 - Hra je definována ve Visual Basicu jako „modul“ = jen jedna instance v celém programu
- Potřebné součásti ve třídě Hra:
 - Sub PoložKámen(x, y, barva)
 - Function ZjistiKámen(x, y) as Barva
 - Const Velikost = 8
 - Všechno to jsou věci z desky, takže je možno místo nich rovnou předat celou desku
 - Function DejDesku() as Deska

Jdeme na věc 😊

- Aby toho nebylo moc najednou, neřešíme objektový návrh
 - Takže celé kreslení a ovládání nacpeme do třídy okna
- Visual Studio 2008, založíme nový projekt
- Nastavíme černé pozadí oknu
- Vložíme `PictureBox`, pojmenujeme `pad`
- Velikost obrázku je 64x64 → `Const VelikostKostky = 64`
 - upravíme velikost padu a okna (v konstruktoru)
 - `pad.Width = pad.Height = Deska.Velikost * VelikostKostky`
 - `ClientSize =`
`new Size(pad.Width+pad.Left*2, pad.Height+pad.Top*2)`

Načtení obrázků

- Obrázky načteme do objektů typu `Bitmap`
- Vytvoříme si pole pro barvy a další bitmapy samostatně
 - Deklarace: `Dim kameny(2) as Bitmap`
 - Naplnění: `kameny(1) = new Bitmap("dark.png")`
- Poznámka: Obrázky kamenů mají průhledné pozadí
 - Takže při kreslení pod ně budeme dávat políčko

Kreslení

- Aby to hned „něco dělalo“, přidáme kreslení
- pad → Properties → událost **Paint**
 - Toto lze dělat i programově, ale přes Visual Studio je to snazší (stačí párkrát kliknout 😊)
- Projdeme všechna políčka a nakreslíme

```
Sub pad_Paint(object sender, PaintEventArgs e)
    e.Graphics.DrawImage(obrázek, x, y, VelikostKostky,
        VelikostKostky)
End Sub
```
- Poznámka:
 - Počátek souřadnic GDI+ je vlevo nahoře
 - Je ale dobré mít počátek desky vlevo dole
 - Není dobré vázat dohromady GUI a logiku hry

Ovládání myši

- Potřebujeme převod souřadnic myši na číslo políčka:
 $x / \text{VelikostKostky}, \text{Deska.Velikost} - 1 - y / \text{tilesize}$
- Událost `MouseClicked`
 - Je-li políčko prázdné, necháme tam položit kámen
- Ošetření chyby
 - Při neplatném tahu zobrazíme okraj desky na chvíli červeně
 - Realizujeme to pomocí časovače
 - Založíme `System.Windows.Forms.Timer`
 - Zapneme jej vždy při chybě a vypneme při prvním tiku
 - Pomocí `Form.BackColor` změníme barvu pozadí okna

Doladění objektové struktury

- Třída Deska se může po přidání kamene sama nechat překreslit – nemusíme na to pak pořád myslet
 - Přidáme vlastní událost do třídy Deska
 - Metoda Deska.PoložKámen() ji aktivuje po změně desky
 - Program.Main() napojí obsluhu této události na Form1

© Mgr. Aleš Kepřt, Ph.D., 2006-2010

Obrázky © Sebastian Bušek 2010

Vytvořeno pro potřeby výuky na MVŠO. Tento text není určen pro samostudium, ale jen jako vodítka pro přednášku, takže jeho obsah se může čtenáři zdát stručný, nekompletní či možná i chybný. Použití je povoleno dle vlastní libosti, ale jen na vlastní nebezpečí. ☺ V případě dalšího šíření je NUTNO uvádět původního autora a odkaz na původní dokument. Komentáře můžete posílat emailem autorovi (adresu najdete přes Google).