

GUI grafických aplikací

– verze s komentářem –

Aleš Kepřt

Katedra informatiky UP

duben 2006, aktualizace březen 2007, duben 2008

Windows – Událostmi řízené aplikace

- Každá aplikace má 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
- Návrátová 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** – nástupce GDI+ (.NET 3.0)
 - Výhody: lepší objektový model, již nezávislé na rozlišení

Nejzajímavější je otázka závislosti na rozlišení. Moderní grafické systémy nejsou závislé na rozlišení monitoru. Výhodou je, že programy pak vypadají pořád pěkně, bez ohledu na to, na jakém monitoru a v jakém rozlišení je používáte. Ve Windows zatím tento princip používá jen Media Player a samozřejmě většina počítačových her (bohužel ne všechny). Uvidíme, jaký bude Internet Explorer 7 – protože nějaký pokrok právě zde lze tušit... Microsoft před nedávnem uvolnil další verzi knihovny WPF, také známé jako Avalon, která by měla „vymoženosti“ dříve známé jen z DirectX přinést do klasických „okenních“ aplikací ve Windows.

Ve vašich programech se však vždy řiďte heslem: Nepoužívat lopatu na komáry. Jsou-li grafické požadavky programu jednoduché, použijte jednoduchou knihovnu (např. GDI).

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 prý nikdy nic neblíká (Aero UI)

Jakých chyb se vyvarovat?

- Grafiku překreslujte výhradně v obsluze zprávy WM_PAINT
- Nikdy nevynucujte překreslení pomocí Update(), ale jen invalidujte okno pomocí Invalidate()
- Grafiku přenášejte do okna najednou; v případě potřeby použijte kreslení do bitmapy
- Nepoužívejte OpenGL a DirectX pro triviální programy s jednoduchou grafikou
- Dávejte přednost GDI+ (případně WPF), protože je nativně objektové

Potřebujete-li animovanou grafiku, pak se vám může stát, že v některých specifických případech budete muset ustoupit od obvyklých principů a kreslit grafiku přímo. Zatímco běžný systém pomocí `Invalidate` a `WM_PAINT` umožní třeba 10 nebo 30 překreslení za sekundu, pomocí přímého kreslení lze dosáhnout mnohem větší „rychlosti“. Jedna z mých her třeba kdysi na procesoru Pentium 60 překreslovala 11000 obrázků za sekundu – ano, měla jednoduchou grafiku a kreslil jsem ji pomocí DirectX. Otázkou opět je, zda je to u daného konkrétního programu nutné.

Následuje modelový příklad řešení GUI pro dámu. Příklad je v GDI+. Je to nativně objektová knihovna, která je součástí .NETu. Na rozdíl od toho, co bylo řečeno v úvodu této přednášky, programy v .NETu nemají žádnou smyčku zpráv a `WndProc`. Namísto toho jsou založeny na událostech, jde o tzv. event driven programming. Událostem přiřadíte obslužné funkce/metody, čímž zajistíte, aby při příchodu určité zprávy, které způsobí právě vznik oné události, byl vykonán kód vaší metody.

Příklad: Chceme v .NETu udělat nějakou činnost při stisku tlačítka. Pak stačí na formulář přidat tlačítko a do kódu (třeba do konstruktoru) napsat `button1.Click += a 2x stisknout Tab`. Dvojitý stisk klávesy Tab vloží do kódu vše potřebné a zbývá jen dopsat kód, který se má vykonat při stisku toho vašeho tlačítka.

Konkrétně pro překreslování grafiky samozřejmě využijeme událost `Paint`.

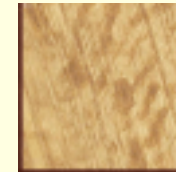
Příklad: GUI pro dámu

- Čtvercová „šachovnicová“ plocha
- Myší lze posouvat kameny (tj. řešíme i vstup)
- Příklad realizace v GDI+
 - Čtvercová grafická plocha, nijak ji nedělíme
 - Připravíme si obrázky políček a kamenů
 - Překreslujeme vždy celou plochu
 - Zachytáváme pohyb a klikání myší

Příprava (1.) – typ Piece, obrázky

```
enum Piece {  
  None,  
  DarkMan,   
  DarkKing,   
  LightMan,   
  LightKing,   
}
```

+ grafika prázdných políček



Příprava (2.) – typ Board

```
class InvalidCoordinatesException : Exception;
```

```
class Board {
```

```
    public const int boardsize = 10;
```

```
    // Zkontroluje, že pozice je platná (tj. v mezích velikosti šachovnice)
```

```
    // Pokud není, vyhodí výjimku InvalidCoordinatesException
```

```
    public void CheckPosition(int x, int y); // Toto je jen pro kontrolu
```

```
    // Vrací typ figurky na dané pozici
```

```
    public Piece GetPiece(int x, int y);
```

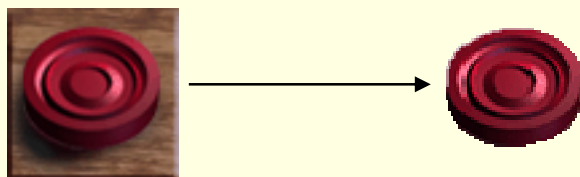
```
    // Pohne figurkou
```

```
    public void MovePiece(int sx, int sy, int dx, int dy);
```

```
}
```

Příprava (3.) – grafika přesunu kamenů

- Chceme mít funkci „uchop kámen myší a přetáhni jej na novou pozici“
- Potřebujeme grafiku „kámen bez políčka“
 - Ořežeme jeden kámen na oblý tvar
 - Ostatní tři kameny dopočítáme programem
 - Máme tak zajištěno, že všechny vypadají stejně



Jdeme na věc 😊

- Aby toho nebylo moc najednou, neřešíme objektový návrh → vše nacpeme do třídy okna
- Visual Studio 2005, založíme nový projekt
- Nastavíme černé pozadí oknu
- Vložíme PictureBox, pojmenujeme pad
- Velikost obrázku je 64x64 → `const tilesize = 64`
 - upravíme velikost padu a okna (v konstruktoru)
 - `pad.Width = pad.Height = Board.boardsize * tilesize`
 - `ClientSize = new Size(pad.Width + pad.Left*2, pad.Height+pad.Top*2)`

Nutno upozornit, že místo PictureBox tady lze použít jakýkoliv jiný objekt typu odvozeného od třídy Control. My totiž ve skutečnosti z toho picture boxu použijeme jen to, že sám nijak nepřekresluje ani nemaže své pozadí. Proč je opravdu možné tam dát úplně cokoliv, nejlépe úplně prázdný objekt Control, najdete v nějaké knize podrobněji popisující principy GUI. (Mělo by to platit i mimo Windows, nicméně nechci se v tom vrtat.)

Načtení obrázků

- Obrázky načteme do objektů typu Bitmap
- Použijeme 3 pole
 - Deklarace: `Bitmap[] tiles, pieces, moving;`
 - Naplnění: `tiles[0] = new Bitmap("dark.png");`
- Obrázky pro pohyb musíme vypočítat
 - Jeden použijeme jako vzor a ostatní ořežeme
`Bitmap output = (Bitmap)source.Clone();`
`Color transparent = pattern.GetPixel(0, 0);`
`output.MakeTransparent(transparent);`
`if(pattern.GetPixel(x, y) == transparent)`
`output.SetPixel(x, y, transparent);`

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

```
void pad_Paint(object sender, PaintEventArgs e) {  
    e.Graphics.DrawImage(img, x, y, tilesize, tilesize);  
}
```
- Poznámka:
 - Počátek souřadnic GDI+ je vlevo nahoře
 - Je ale dobré mít počátek desky vlevo dole
 - Není dobré vázat dohromady GUI a logiku hry

Zde je zajímavé podotknout, že metoda `Graphics.DrawImage` umí i zmenšovat a zvětšovat kreslené bitmapy. Čili při troše snahy můžete nakonec i pomocí GDI+ dosáhnout toho, že váš program bude víceméně nezávislý na rozlišení monitoru. Prostě podle toho, jak bude uživatel myši měnit velikost okna, budete upravovat velikosti políček na šachovnici. Řešení už je na vás, jistě to zvládnete. 😊

Následující slidy se zabývají řešením ovládní pomocí myši. Nutno upozornit, že myš, ačkoliv je to vstup, přímo souvisí s oknem, do kterého kreslíme výstup. Teoreticky se sice vždy doporučuje, abyste ve vašich programech nějak rozumně vstup a výstup oddělili – pro přehlednost kódu – ale vždy nakonec budete muset obojí napojit do jedné třídy, neboť vstup a výstup se vždy týkají nějakého okna. Zkuste to napřed napsat dohromady, pak jistě přijdete na to, jak to vhodně oddělit a zpřehlednit.

Ovládní myši bude vyžadovat mj. metodu `PointToClient`, která přepočítává souřadnice myši z celoobrazovkových (myš má vždy počátek vlevo nahoře na primárním monitoru) do souřadnic objektu (v našem případě pad). Tak se toho nelekněte! 😊

Ovládání myši (1.)

- Převod souřadnic myši na číslo políčka:
 $x / \text{tilesize}, \text{Board.boardsize} - 1 - y / \text{tilesize}$
- Událost **MouseDown**
 - Je-li stisk na pozici s kamenem, zapamatujeme si pozici stisku a začneme přetahovat kámen
 - $\text{pickedpos} \leftarrow$ souřadnice na desce
 - Když je tam kámen, tak $\text{mousedown} = \text{true}$, $\text{Cursor} = \text{Cursors.Hand}$ a invalidujeme pad
 - Musíme doplnit kód do Paint, aby se nekreslil kámen, který držíme myší: $\text{testujeme mousedown \&\& pickedpos}$
 - Musíme invalidovat pad, jinak se to nepřekreslí

Ovládání myši (2.)

■ Událost **MouseMove**

- Kreslíme kámen během tažení myši

- `if(mousedown) pad.Invalidate();`

- Vlastní kreslení musí být opět v Paint (!)

- Souřadnice: `pad.PointToClient(MousePosition)`

■ Událost **MouseUp**

- Obsluhujeme jen při mousedown

- Pokusíme se pohnout kamenem

- `board.MovePiece(pickedpos.X, pickedpos.Y, c.X, c.Y);`

- Vrátime zpět podobu myši a invalidujeme pad

- `Cursor = Cursors.Default`

Ovládání myši (3.)

- Událost **MouseLeave**

- Když myš opustí okno během tažení kamene, vrátíme kámen zpět na původní místo

- Ošetření chyby

- Při neplatném tahu zobrazíme pozadí okna na chvíli v červené barvě

- 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

- Není vhodné nacpat vše do třídy Form1
- Oddělíme jednotlivé části
 - Třída hry (Game – nyní tam žádná není)
 - Vstupy (události myši) sleduje jiná třída (Input)
 - Zachycené události předáme třídě Game
 - Výstup (kreslení) necháme ve Form1
 - Třída Board se může při změně stavu desky sama nechat překreslit
 - Přidáme vlastní událost do třídy Board
 - Metoda MoveTile ji aktivuje při změně desky
 - Program.Main napojí obsluhu této události na Form1

Poslední (dvacátý) slide jsem z této verze záměrně odstranil. Tak po něm nepátrejte. ☺

Celá tato přednáška byla zaměřena jen na řešení grafické části GUI, mnohé jiné věci byly tedy zjednodušeny nebo úplně opomenuty. Navíc cílem samozřejmě je, abyste si dámu naprogramovali sami – ne abyste ji opsali ode mne. ☺

Obrázky k dámě pocházejí z programu guichesters © Jon Kreuzer, Josh Hess.
Použito se svolením autora pro výukové účely na KMI.

© Mgr. Aleš Kepřt, Ph.D., 2006,2007,2008

Vytvořeno pro potřeby přednášky na UP Olomouc. Tento text není určen pro samostudium, ale jen jako vodítko 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 e-mailem autorovi (adresu najdete přes Google).