DUM č. 8 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 30.06.2014

Ročník: studenti semináře

Anotace DUMu: Úvodní lekce druhého projektu, simulace zajíců v jazyce C s knihovnou Allegro. Nainstalování IDE DEV-C++ a knihovny Allegro. Jednoduchá demo aplikace.

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Projekt: Simulace života zajíce v C/Allegro

Úvod

Druhý projekt sady je zaměřen na strukturované programování v jazyce C s herní knihovnou Allegro ve verzi 4 (má zásadně odlišnou stavbu a funkce oproti novější verzi 5, kterou nepoužívám kvůli menšímu množství studijních materiálů a vzorového kódu). Zadání je stejné, ale budeme využívat jiné programovací paradigma, jiný jazyk a nebude mít k dispozici žádné běhové prostředí, takže si musíme celou aplikaci naprogramovat od začátku.

Projekt má pro studenty největší přínos ve srovnání objektového a strukturovaného paradigmatu. Jelikož je poměrně jasné, že u projektu zaměřeného na jednoznačně identifikovatelné entity – objekty a navíc v takovém rozsahu kódu, je volba OOP jednoznačně výhodnější, mohou studenti snadno nabýt dojmu, že „Java a Greenfoot je lepší než C“. To je poněkud ukvapené rozhodnutí a je důležité ho korigovat a vysvětlit, že ono „je lepší“ platí jen v tomto případě, protože Greenfoot je pro tento typ úkolu vhodnějším (spíš než lepším) nástrojem. Strukturovaný přístup a jazyk C se mnohem víc hodí na jiné typy projektů (např. mikroprocesory, rychlé a výkonné knihovny, programování modulů do jádra OS apod.). Je pochopitelně možné implementovat celý projekt v C++, ale tím by odpadla možnost porovnat neobjektový a objektový přístup.

Zadání úkolů se bude v této sadě DUM z velké části překrývat s první sadou v Greenfootu. Rozdíl je ale v samotné implementaci (řešení) a v nutnosti vytvořit si vlastní běhové prostředí. Aplikaci nebudeme komplikovat dělením do více souborů i když by to bylo velmi vhodné. Studenty by to odvádělo od úvah nad prací s knihovnou Allegro a tento přístup si vyzkouší v třetím projektu.

Instalace knihoven a IDE

Projekt budeme vytvářet v IDE Dev-C++ v prostředí MS Windows nebo v libovolném editoru v Linuxu. Dev-C++ je dnes již zastaralý projekt, občas se chová nesprávně (problémy s kompilací upraveného kódu nebo spouštění starší verze), takže je nutné program zavřít a znovu spustit. Přesto má dvě velké výhody: prostředí je velice jednoduché a přehledné a má správce a instalátor knihoven. Studenti se v něm snadno a rychle zorientují a instalace knihoven probíhá také jednoduše.

IDE stáhneme a nainstalujeme ze stránek http://www.bloodshed.net/devcpp.html. Po spuštění programu doinstalujeme knihovnu Allegro. To se udělá pomocí správce balíků v menu Nástroje → Zjistit updaty… a v nově otevřeném okně zvolíme jako devpak server „devpaks.org“ a klikneme na „Check for updates“. Poté zvolíme jako Groups jen „Allegro“ a ve výběru balíčků zaškrtneme balík „Allegro 4.2.1“ a zvolíme „Download selected“. Po stažení je nutné v instalačním okně několikerým kliknutím na „Next“ a pak „Finish“ provést samotnou instalaci.

Pak už lze založit nový projekt (Soubor → Nový → Projekt…), jako typ je pod záložkou „Multimedia“ k výběru „Allegro application (DLL)“ nebo „Allegro application (static)“. První varianta používá tzv. dynamické linkování, což znamená, že výsledná binárka (exe soubor ve Windows) neobsahuje funkce knihovny Allegro, ale pouze jejich volání a samotné funkce se při běhu programu načítají z knihovny (soubory s koncovkou .dll). Pokud knihovní soubor (dll) není ani v systému ani v adresáři s aplikací, program nelze spustit. Výhodou ale je, že binárka je o knihovní funkce menší a

máme-li v systému více aplikací, které tuto knihovnu sdílí, pak je její kód v systému přítomen pouze jednou společně pro všechny programy v knihovním souboru dll. Tento knihovní soubor je vlastně zkompilovaný program, který ale nemá žádný vstupní bod (funkce main).

Varianta statického linkování pak znamená, že všechny knihovní funkce se budou vkládat přímo do výsledného spustitelného souboru, který bude proto větší, ale už nebude potřebovat žádné dodatečné soubory k běhu. Nevýhoda je tedy větší obsazené místo, což se projeví, pokud více různých aplikací sdílí kód knihovny. Každá pak má svou vlastní verzi ve své binárce. Výhodou naopak je, že není nutné dodávat žádné dll soubory. My si proto zvolíme variantu statického linkování. Projekt pojmenujeme Rabbit a nezapomeneme zvolit, že se jedná o projekt v jazyce C, nikoliv C++.

Nyní máme uložit projektový soubor, který pojmenujeme Rabbit.dev. Tento soubor není pro vznik programu nijak potřebný. Je to soubor s nastavením Dev-C pro tento projekt. Až projekt poprvé překompilujeme (uložíme), můžeme si soubor Rabbit.dev prohlédnout v textovém editoru. Je to obyčejný textový soubor formátu INI.

IDE nám vygenerovalo do projektu nový soubor „main.c“ se vzorovou ukázkou v Allegru. Soubor uložíme jako „rabbit.c“, což můžeme udělat přímo při kompilaci: klávesa F9 nebo ikona s barvami Windows v rámečku se záhlavím (hned vedle čtyř šedých čtverečků). Po spuštění aplikace nic nedělá, pouze čeká na stisk ESC.

Následující kód je citací vzorového kódu aplikace, který vygeneruje prostředí Dev-C++.

```
#include <allegro.h>

void init();
void deinit();

int main() {
    init();

    while (!key[KEY_ESC]) {
          /* put your code here */
    }

    deinit();
    return 0;
}
END_OF_MAIN()

void init() {
    int depth, res;
    allegro_init();
    depth = desktop_color_depth();
    if (depth == 0) depth = 32;
    set_color_depth(depth);
    res = set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
    if (res != 0) {
          allegro_message(allegro_error);
```

```
            exit(-1);
        }

        install_timer();
        install_keyboard();
        install_mouse();
        /* add other initializations here */
}

void deinit() {
        clear_keybuf();
        /* add other deinitializations here */
}
```

Kód se studenty projdeme řádek za řádkem a vysvětlíme si jednotlivé konstrukce. Na dotaz, co která funkce dělá a jak, odkazujeme studenty na dokumentaci na stránkách http://alleg.sourceforge.net/stabledocs/en/index.html a pod odkazem „The allegro manual“. Zde ať si sami dohledají, jak která funkce funguje. Hledíme si také na správné používání terminologie strukturovaného paradigmatu. Zde nemáme ani metody ani atributy, pouze funkce a proměnné.

Studentům také doporučím stránky https://www.allegro.cc/, kde lze najít mimo různých informací především vzorové projekty. Z nich si řada zaslouží pozornost, ale obzvláště zajímavá je hra Alex, the Allegator 4 mající samostatné stránky http://allegator.sourceforge.net/. Tato hra je velice podobná (i grafikou) známé dosové hře Prehistorik. Je naprogramovaná v Allegro 4 a jsou k dispozici kompletní zdrojové kódy, takže si studenti mohou zkusit hru poupravit, překompilovat, pročíst.

Úkoly s řešeními

1. Vytvořte nový projekt v Allegro 4 v prostředí Dev-C++. Pojmenujte jej Rabbit, nechť je psán v čistém jazyce C a je to staticky linkovaná aplikace s knihovnou Allegro. Ponechejte v projektu výchozí demonstrační kód, který prostudujte s použitím manuálu knihovny Allegro.

   Řešení je podle úvodního textu na předcházejících stránkách.
2. Upravte demonstrační kód tak, aby se spouštěl přes celou obrazovku (fullscreen), v rozlišení 800×600 px a aby se aplikace vypnula nejen při stisku ESC, ale také při stisku klávesy Q.

   Řešení vychází z pochopení demonstračního kódu a úprav jedné podmínky a tří parametrů funkce, která nastavuje grafický mód. Každý ji musí zvládnout individuálně dohledat v manuálu.
3. Zařiďte, aby pozadí okna (herní plán) byl vyplněný obrázkem trávy z úlohy v Greenfootu. Nejprve na zkoušku můžete použít jeden velký obrázek, pak ale vyplňte plán dlaždicovitě pomocí menšího čtvercového obrázku. Pro vykreslení pozadí napište samostatnou funkci s deklarací `int drawBackground(BITMAP* buffer)` – přebíraný parametr je odkaz na bitmapu, do které se má kreslit (bude to obvykle obrazovka), funkce vrací 1 při úspěchu, 0 pokud nelze obrázek načíst. Tip pro hledání v dokumentaci: pro překreslování bitmap se používá termín „blitting“.

   Nejprve je nutné v dokumentaci najít funkci, která slouží k vykreslení zdrojové bitmapy do cílové. (Viz tip.) Dále je potřeba umět načíst existující soubor s obrázkem – Allegro na to má funkci v dokumentaci v sekci „Loading image files“… Při vykreslování obrázku rozhodně nepoužíváme pevně nastavené rozměry číslem, ale dotazujeme se na výšku a šířku plánu i obrázku funkcí nebo proměnnou. Funkci uvedenou v řešení voláme v mainu s parametrem screen, což je globální proměnná knihovny Allegro, která reprezentuje obrazovku.

```
int drawBackground(BITMAP* buffer) {
   int x, y;
   □ palette;
   BITMAP* image = □("grass.bmp",palette);
   if (!□) return 0;

   for(y=0;y<SCREEN_□/image->□;y++) {
     for(x=0;x<SCREEN_□/image->□;x++) {
        blit(□, □, 0, 0, □*image->□, □*image->□, image->□, image->□);
     }
   }
   return 1;
}
```

4. Přepište funkci drawBackground na `int loadBackground(BITMAP** bgBuffer)` a upravte ji tak, aby do bgBuffer načetla kompletní obrázek pozadí, abychom pak už jen vykreslovali obsah bgBuffer. Nelze-li obrázek vytvořit, vrátí funkce 0, jinak 1. Proč je toto výhodnější než předchozí řešení? Jaká je nevýhoda? Proč má bgBuffer nyní dvě hvězdičky?

   Toto řešení zabírá mnohem více paměti, protože celý velký obrázek pozadí musí být načtený v paměti. Podstatnou výhodou ale je, že se s každým překreslováním nemusí znovu načítat soubor s obrázkem a znova cyklicky vykreslovat dlaždice, pouze se rovnou vykreslí obsah

bgBuffer. Jen je kvůli tomu nutné zavést novou proměnnou v main, která bude nést vygenerovaný obrázek pozadí. Důvod pro dvě hvězdičky je ten, že ve funkci loadBackground budeme obrázek alokovat a proto se změní hodnota odkazu na bgBuffer. Musíme tedy tento odkaz předat odkazem, proto použijeme ukazatel na ukazatel. Na závěr funkce main nezapomeneme alokovanou paměť pro bgBuffer dealokovat!

```
int loadBackground(BITMAP** bgBuffer) {
   int x, y;
   □ palette;
   BITMAP* image = □("grass.bmp",palette);
   if (!□) return 0;

   □bgBuffer = □(SCREEN_□,SCREEN_□);
   if (!*□) return 0;

   for(y=0;y<SCREEN_□/image->□;y++) {
     for(x=0;x<SCREEN_□/image->□;x++) {
        blit(□, *□, 0, 0, □*image->□, □*image->□, image->□, image->□);
     }
   }
   return 1;
}

int main() {
    BITMAP * bgBuffer;

    init();
    if (!loadBackground(□)) return 1;

    while (!key[KEY_ESC] □ !key[□]) {
         blit(□, □, 0, 0, 0, 0, SCREEN_□, SCREEN_□);
    }

    deinit();
    □_bitmap(bgBuffer);
    return 0;
}
END_OF_MAIN()
```

Úkoly

1. Vytvořte nový projekt v Allegro 4 v prostředí Dev-C++. Pojmenujte jej Rabbit, nechť je psán v čistém jazyce C a je to staticky linkovaná aplikace s knihovnou Allegro. Ponechejte v projektu výchozí demonstrační kód, který prostudujte s použitím manuálu knihovny Allegro.
2. Upravte demonstrační kód tak, aby se spouštěl přes celou obrazovku (fullscreen), v rozlišení 800×600 px a aby se aplikace vypnula nejen při stisku ESC, ale také při stisku klávesy Q.
3. Zařiďte, aby pozadí okna (herní plán) byl vyplněný obrázkem trávy z úlohy v Greenfootu. Nejprve na zkoušku můžete použít jeden velký obrázek, pak ale vyplňte plán dlaždicovitě pomocí menšího čtvercového obrázku. Pro vykreslení pozadí napište samostatnou funkci s deklarací `int drawBackground(BITMAP* buffer)` – přebíraný parametr je odkaz na bitmapu, do které se má kreslit (bude to obvykle obrazovka), funkce vrací 1 při úspěchu, 0 pokud nelze obrázek načíst. Tip pro hledání v dokumentaci: pro překreslování bitmap se používá termín „blitting“.
4. Přepište funkci drawBackground na `int loadBackground(BITMAP** bgBuffer)` a upravte ji tak, aby do bgBuffer načetla kompletní obrázek pozadí, abychom pak už jen vykreslovali obsah bgBuffer. Nelze-li obrázek vytvořit, vrátí funkce 0, jinak 1. Proč je toto výhodnější než předchozí řešení? Jaká je nevýhoda? Proč má bgBuffer nyní dvě hvězdičky?