DUM č. 9 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 30.06.2014

Ročník: studenti semináře

Anotace DUMu: Simulace zajíce v Allegru, vytvoření herní scény, jednoduchého prostředí, implementace hráče a jeho základního pohybu a vykreslování ve scéně.

esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
2007-13 OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Projekt: Simulace života zajíce v C/Allegro
Hráči – vytvoření a pohyb postav

Úvod

V této lekci implementujeme hráče: zajíce, lišku a částečně mrkev. Oproti řešení v Greenfoot nemáme možnost používat objekty a jejich atributy a metody, které úlohu pěkně rozčlenily a zpřehlednily. Ve strukturovaném přístupu nezbývá, než atributy zavést jako proměnné, ideálně pomocí heterogenních datových struktur (struct) nebo polí. Místo metod pak budeme mít společné funkce. Tím se řešení mírně znepřehlední.

Další problém nastává v tom, že nemáme žádné běhové prostředí, od kterého bychom dědili již implementovaný kód (není zde žádná třída Actor), proto nezbývá než vše implementovat od začátku. Pro srovnání s objektovým řešením vytvoříme strukturu sActor, která bude reprezentovat libovolný objekt v herním plánu (zajíce, lišku, mrkev) a ponese potřebné informace jako je pozice, rychlost pohybu, typ objektu apod. Tuto strukturu budeme generovat funkcemi pro vytvoření hráčů, další funkce bude sloužit k jejich vykreslování, k rušení, aktualizaci pohybu apod.

Netriviální úkol bude spočívat později ve vytváření a odstraňování nových hráčů, k čemuž budeme používat komplikovanější datové struktury. Tuto funkcionalitu ale zavedeme až v příští lekci.

Úkoly s řešeními

1. Vytvořte strukturu sActor, která bude sloužit k uchování potřebných informací o hráči. Nechť má následující složky: souřadnice x a y, rychlost v počtu pixelů za krok hry (speed), natočení ve vhodné a hlavně dostatečně přesné jednotce (direction), typ hráče (type). Doplňte také direktivami konstanty pro jednotlivé typy hráčů: AT_FOX (liška), AT_RABBIT (zajíc), AT_FRABBIT (zaječka), AT_CARROT (mrkev).

   Datové typy souřadnic i rychlosti jsou nenulová celá čísla. Pro typ stačí zvolit menší (jednobytový) číselný datový typ. Natočení lze uchovávat buď v radiánech (desetinná čísla) nebo stupních (celá čísla), eventuálně v hodnotách 0 až 256, jak zpracovávají úhly funkce Allegra. Kvůli udržení přesnosti bude nejvýhodnější zvolit stupně a při výpočtech posunutí přepočítávat pro goniometrické funkce na radiány, než uchovávat přímo nepřesné údaje v radiánech. Řešení jen přibližně:

```
#□ AT_FOX     0
#□ AT_RABBIT  1
#□ AT_FRABBIT 2
#□ AT_CARROT  3

struct sActor {
   □ type;
   □ □ x;
   □ □ y;
   □ □ speed;
   □ □ direction;
};
```

2. Zadefinujte direktivu preprocesoru CHECK_TYPE(t), která bude sloužit pro kontrolování, zda je zadaný správný typ hráče, tj. bude vkládat logický výraz vracející 1, má-li t hodnotu některé z AT_* konstant, jinak 0.

```
#□ CHECK_TYPE(t) (t==AT_FOX □ t==AT_RABBIT □ t==AT_FRABBIT □
t==AT_CARROT)
```

3. Vytvořte funkci `struct sActor* createActor(□ type, □ x, □ y, □ speed, □ direction)`, která bude sloužit k alokaci nového hráče (místo čtverečků doplňte správné datové typy, jak jste je určili v 1. úkolu). Testujte, zda jsou všechny předané parametry korektní. Pokud ne nebo pokud se nepodaří alokovat prostor pro nového hráče, vraťte NULL. Je-li alokace úspěšná a parametry v pořádku, nastavte hodnoty hráče na předané parametry a vraťte ukazatel na nového hráče. Ve funkci main přidejte ukazatel na hráče „actor“ a přiřaďte do něj ihned funkcí createActor nového hráče. Zkoušejte správné i nesprávné parametry, zda se funkce chová korektně.

   Funkce musí pomocí malloc správně alokovat jedno pole v paměti. Vzorové řešení vypad takto:

```
struct sActor* createActor(□ type, □ x, □ y, □ speed, □ direction) {
    □ actor = (struct sActor *) malloc(sizeof(□));
   if (!□ || !(□(type)) || x□SCREEN_□ || y□SCREEN_□) {
     return NULL;
   }
```

```
    actor□type      = type;
    actor□x         = x;
    actor□y         = y;
    actor□speed     = speed;
    actor□direction = direction;
    return actor;
}
```

4. Vytvořte funkci `void drawActor(□ actor, □ buffer, □ images)`, která bude vykreslovat předaného hráče do obrazového bufferu buffer (může být předán např. screen) a jako poslední parametr má pole ukazatelů (tedy ukazatel na ukazatel) na obrázky hráčů podle typu. Toto pole obsahuje již načtené obrázky a to tak, že index obrázku je jedna z konstant AT_*. Nejprve zkontrolujte předané parametry (jsou-li špatné, opusťte funkci). Pak vykreslujte: liška, zajíc a zaječka se kreslí v natočení podle actor->direction. Použijete-li funkci rotate_sprite, všimněte si v dokumentaci, že poslední parametr otočení je typ fix a je nutné předané číslo na fix vhodnou funkcí zkonvertovat. Navíc jsou úhly v rozsahu 0 až 256, nikoliv 0 až 360. Podobně jako v řešení pro Greenfoot i zde budeme obrázek překlápět, aby nikdy nebyl hřbetem dolů. Použijeme na to vhodnou alternativní funkci k rotate_sprite, která navíc překlápí. Mrkev se pouze vykreslí (jako sprite – částečně průhledný obrázek hráče) bez otáčení. Nezapomeňte, že ve všech případech vykreslujeme hráče tak, aby na jeho souřadnicích byl střed obrázku, nikoliv okraj.

   K správnému vyřešení je potřeba pouze důkladně pročíst dokumentaci, zvážit správné datové typy parametrů a najít funkci pro konverzi celočíselných stupňů na fix hodnotu v intervalu 0 až 256. Během převodu je potřeba počítat co nejpřesněji, proto dělení musí proběhnout nad desetinnými čísly nikoliv celými. Protože dělíme celočíselnou hodnotu, musí být dělitel konstanta desetinná, aby dělení proběhlo desetinně a ne celočíselně! Je vhodné ve funkci místo „ifů“ používat switch, jelikož kód dělíme podle konstantních voleb a je rozumné implementovat i volbu „default“ pro případ chyby, kdy se změní typ hráče na neexistující volbu. V příkladu se pak vykreslí červené kolečko.

```
void drawActor(□ actor, □ buffer, □ images) {
   if (!□ || !□ || !□) return;
   switch(actor□□) {
      case □:
      case □:
      case □:
         if (actor□direction<=90 □ actor□direction>=270) {
            rotate_sprite(buffer, images[□], actor->□-images[□]->□/2,
                  actor->□-images[□]->□/2, □(actor□direction/□.0*□));
         } else {
            rotate_sprite□(buffer, images[□], actor->□-images[□]->□/2,
                  actor->□-images[□]->□/2, □(actor□direction/□.0*□));
         }
         break;
      case □:
         □_sprite(buffer, images[□], actor->□-images[□]->□/2,
                  actor->□-images[□]->□/2);
         break;
```

```
      default:
         circlefill(buffer,actor->x,actor->y,5,make□(255,0,0));
         break;
   }
}
```

5. Napište funkci `int loadActorImages(□ images)`, která bude načítat pro vykreslování hráčů potřebné bitmapy do pole ukazatelů (ukazatel na ukazatel) na obrázek. Ve funkci načítejte do pole pod příslušným indexem daným direktivami AT_* obrázky hráčů. Vždy zkontrolujte, zda byl úspěšně načten, pokud ne, oznamte který obrázek nešlo načíst voláním allegro_message a navrácením hodnoty 0. Pokud se vše načetlo správně, vraťte 1. Ve funkci main vytvořte pole ukazatelů na BITMAP o správném počtu polí a pojmenujte actorImages. Načítejte do něj funkcí loadActorImages potřebné obrázky.

   Funkce pro načtení obrázků je podobná už dříve řešenému příkladu vykreslení pozadí. Neobsahuje žádnou záludnost snad vyjma možných problému s formátem souboru (není-li předaná bitmapa úplně jednoduchý 24bitový BMP obrázek, načítání selže bez chybové hlášky). Další možný problém představuje volání této funkce dříve, než proběhne „init“, kde se inicializuje prostředí a důležité globální proměnné.

```
int loadActorImages(□ images) {
   □ pal;
   images[□]=load_bitmap("fox.bmp",pal);
   if(!images[□]) {
      allegro_message("Cannot found fox.bmp.");
      return 0;
   }
   images[□]=load_bitmap("rabbit.bmp",pal);
   if(!images[□]) {
      allegro_message("Cannot found rabbit.bmp.");
      return 0;
   }
   images[□]=load_bitmap("frabbit.bmp",pal);
   if(!images[□]) {
      allegro_message("Cannot found frabbit.bmp.");
      return 0;
   }
   images[□]=load_bitmap("carrot.bmp",pal);
   if(!images[□]) {
      allegro_message("Cannot found carrot.bmp.");
      return 0;
   }
   return 1;
}
```

6. Volejte ve funkci main funkci drawActor (a drawBackground z minulé lekce, obě ve vykreslovacím cyklu) a kontrolujte, zda vykresluje správně při různém otočení, umístění a typu hráče. Abyste zamezili problikávání scény kvůli neustálému překreslování přímo na obrazovku, vytvořte v mainu pomocný obrázek „buffer“ o rozměrech celé obrazovky a kreslete do něj. Teprve poté jedním příkazem přepište celý buffer do obrazovky (screen).

   Nezapomeňte nejprve načíst bitmapy – stačí jednou mimo cyklus. Kontrolujte, zda se podařilo pomocný buffer korektně vytvořit. Ne-li, ukončete program. Můžete nejprve kreslit přímo do screen a podívat se, jak obrázek problikává kvůli překreslování prázdného pozadí. Pak ale řešení opravte. Pro kontrolu správné pozice obrázku vykreslujeme ve vzorovém řešení červené kolečko.

```
init();

□ bgBuffer;
□ actorImages[3];
□ buffer = □_bitmap(SCREEN_□,SCREEN_□);
if (!□) return 1;

□ actor = createActor(AT_FOX,400,300,1,0);

if (!□(&bgBuffer)) return 2;
if (!□(actorImages)) return 3;

while (!key[KEY_ESC] □ !key[□]) {
   □(bgBuffer, buffer, □, □, □, □, SCREEN_□, SCREEN_□);
   drawActor(actor, buffer, actorImages);
   circlefill(buffer,actor->x,actor->y,2,make□(255,0,0));
   □(buffer, screen, □, □, □, □, SCREEN_□, SCREEN_□);
}
```

7. Napište funkci void moveActor(actor). Pomocí směru a rychlosti hráče vypočítejte jeho nové souřadnice. Jsou-li uvnitř herního plánu, nastavte je, jinak ponechejte původní. K výpočtu budete potřebovat goniometrické funkce, které se nachází v matematické knihovně math.h. Includujte hlavičkový soubor a v nastavení projektu je nutné předat linkeru informaci o nové knihovně: v menu Projekt → Vlastnosti projektu, záložka Parametry, sloupec Linker se musí přidat parametr „-lm“.

   Tento úkol může způsobit, že program nepůjde překompilovat, pokud se špatně nastaví parametr linkeru, zkontrolujte proto nejprve, že se program opravdu bez chyb kompiluje. Nezapomeňte includovat na začátku programu hlavičkový soubor math.h, jinak nebude možné používat sinus, kosinus a zaokrouhlování. Souřadnice počítejte s desetinnou přesností, před konverzí na integer zaokrouhlujte funkcí pro zaokrouhlení z knihovny math.h.

```
void moveActor(□ actor) {
   int dx = (□) □(actor->□*□((actor->□)/□.0*□.□));
   int dy = (□) □(actor->□*□((actor->□)/□.0*□.□));
   if (actor->x+dx□0 □ actor->x+dx□SCREEN_□ □
       actor->y+dy□0 □ actor->y+dy□SCREEN_□) {
       actor->x□=dx;
       actor->y□=dy;
```

```
    }
}
```

8. Do herního cyklu doplňte vstup z klávesnice (rozhodněte se sami, zda použijete globální pole key nebo funkci readkey() a zdůvodněte). Pomocí stisku různých kláves umožněte měnit načtenému hráči typ, o jedna zvýšit nebo snížit rychlost, přidávat po 10 ° k natočení a vykonat funkci moveActor. Zkuste, zda se váš program (a hráč v poli) chová korektně. Tzn. nevznikají chyby, správně se otáčí i pohybuje.

   Při řešení tohoto úkolu je vhodnější použít readkey, ačkoliv je „blokující“, tzn. pokud není v bufferu klávesnice žádný znak, čeká se, dokud uživatel nějakou klávesu nestiskne. Oproti tomu pole key nijak běh programu nezablokuje, ale jelikož smyčka probíhá velmi rychle, bude jediný stisk klávesy nejspíš identifikován jako několik stisků po sobě, což by u funkce moveActor nebo při změně rychlosti a směru způsobilo velké problémy. Nezapomeneme zamezit hodnotě natočení opustit interval 0 až 360. Do smyčky ve funkci main stačí doplnit pár podmínek (případně by bylo možné použít i switch):

```
int k = readkey()□0x□;
if (k=='f') actor->type=AT_FOX;
if (k=='r') actor->type=AT_RABBIT;
if (k=='a') actor->type=AT_FRABBIT;
if (k=='c') actor->type=AT_CARROT;
if (k=='+') actor->speed□;
if (k=='-') actor->speed□;
if (k=='m') moveActor(□);
if (k=='o') {actor->direction□=10;actor->direction□=360;}
```

9. Vytvořte funkci removeActor, která převezme ukazatel na hráče a dealokuje ho. Na konci běhu aplikace tuto funkci volejte, abyste alokovaný prostor dealokovali.

   Úkol je triviální, jednořádkové řešení spočívající pouze v dealokaci zvládne student sám.

Úkoly

1. Vytvořte strukturu sActor, která bude sloužit k uchování potřebných informací o hráči. Nechť má následující složky: souřadnice x a y, rychlost v počtu pixelů za krok hry (speed), natočení ve vhodné a hlavně dostatečně přesné jednotce (direction), typ hráče (type). Doplňte také direktivami konstanty pro jednotlivé typy hráčů: AT_FOX (liška), AT_RABBIT (zajíc), AT_FRABBIT (zaječka), AT_CARROT (mrkev).

2. Zadefinujte direktivu preprocesoru CHECK_TYPE(t), která bude sloužit pro kontrolování, zda je zadaný správný typ hráče, tj. bude vkládat logický výraz vracející 1, má-li t hodnotu některé z AT_* konstant, jinak 0.

3. Vytvořte funkci `struct sActor* createActor(☐ type, ☐ x, ☐ y, ☐ speed, ☐ direction)`, která bude sloužit k alokaci nového hráče (místo čtverečků doplňte správné datové typy, jak jste je určili v 1. úkolu). Testujte, zda jsou všechny předané parametry korektní. Pokud ne nebo pokud se nepodaří alokovat prostor pro nového hráče, vraťte NULL. Je-li alokace úspěšná a parametry v pořádku, nastavte hodnoty hráče na předané parametry a vraťte ukazatel na nového hráče. Ve funkci main přidejte ukazatel na hráče „actor“ a přiřaďte do něj ihned funkcí createActor nového hráče. Zkoušejte správné i nesprávné parametry, zda se funkce chová korektně.

4. Vytvořte funkci `void drawActor(☐ actor, ☐ buffer, ☐ images)`, která bude vykreslovat předaného hráče do obrazového bufferu buffer (může být předán např. screen) a jako poslední parametr má pole ukazatelů (tedy ukazatel na ukazatel) na obrázky hráčů podle typu. Toto pole obsahuje již načtené obrázky a to tak, že index obrázku je jedna z konstant AT_*. Nejprve zkontrolujte předané parametry (jsou-li špatné, opusťte funkci). Pak vykreslujte: liška, zajíc a zaječka se kreslí v natočení podle actor->direction. Použijete-li funkci rotate_sprite, všimněte si v dokumentaci, že poslední parametr otočení je typ fix a je nutné předané číslo na fix vhodnou funkcí zkonvertovat. Navíc jsou úhly v rozsahu 0 až 256, nikoliv 0 až 360. Podobně jako v řešení pro Greenfoot i zde budeme obrázek překlápět, aby nikdy nebyl hřbetem dolů. Použijeme na to vhodnou alternativní funkci k rotate_sprite, která navíc překlápí. Mrkev se pouze vykreslí (jako sprite – částečně průhledný obrázek hráče) bez otáčení. Nezapomeňte, že ve všech případech vykreslujeme hráče tak, aby na jeho souřadnicích byl střed obrázku, nikoliv okraj.

5. Napište funkci `int loadActorImages(☐ images)`, která bude načítat pro vykreslování hráčů potřebné bitmapy do pole ukazatelů (ukazatel na ukazatel) na obrázek. Ve funkci načítejte do pole pod příslušným indexem daným direktivami AT_* obrázky hráčů. Vždy zkontrolujte, zda byl úspěšně načten, pokud ne, oznamte který obrázek nešlo načíst voláním allegro_message a navrácením hodnoty 0. Pokud se vše načetlo správně, vraťte 1. Ve funkci main vytvořte pole ukazatelů na BITMAP o správném počtu polí a pojmenujte actorImages. Načítejte do něj funkcí loadActorImages potřebné obrázky.

6. Volejte ve funkci main funkci drawActor (a drawBackground z minulé lekce, obě ve vykreslovacím cyklu) a kontrolujte, zda vykresluje správně při různém otočení, umístění a typu hráče. Abyste zamezili problikávání scény kvůli neustálému překreslování přímo na obrazovku, vytvořte v mainu pomocný obrázek „buffer“ o rozměrech celé obrazovky a kreslete do něj. Teprve poté jedním příkazem přepište celý buffer do obrazovky (screen).
7. Napište funkci void moveActor(actor). Pomocí směru a rychlosti hráče vypočítejte jeho nové souřadnice. Jsou-li uvnitř herního plánu, nastavte je, jinak ponechejte původní. K výpočtu budete potřebovat goniometrické funkce, které se nachází v matematické knihovně math.h. Includujte hlavičkový soubor a v nastavení projektu je nutné předat linkeru informaci o nové knihovně: v menu Projekt → Vlastnosti projektu, záložka Parametry, sloupec Linker se musí přidat parametr „-lm“.
8. Do herního cyklu doplňte vstup z klávesnice (rozhodněte se sami, zda použijete globální pole key nebo funkci readkey() a zdůvodněte). Pomocí stisku různých kláves umožněte měnit načtenému hráči typ, o jedna zvýšit nebo snížit rychlost, přidávat po 10 ° k natočení a vykonat funkci moveActor. Zkuste, zda se váš program (a hráč v poli) chová korektně. Tzn. nevznikají chyby, správně se otáčí i pohybuje.
9. Vytvořte funkci removeActor, která převezme ukazatel na hráče a dealokuje ho. Na konci běhu aplikace tuto funkci volejte, abyste alokovaný prostor dealokovali.