DUM č. 11 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 30.06.2014

Ročník: studenti semináře

Anotace DUMu: Simulace pohybu zajíce a lišky, růstové fáze mrkve. Náhodné chování pomocí generátoru pseudonáhodných čísel.

Materiály jsou určeny pro bezplatné používání pro potřeby výuky a vzdělávání na všech typech škol a školských zařízení. Jakékoliv další využití podléhá autorskému zákonu.

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
Pohyb a růst, náhoda

Úvod

V této lekci se konečně dostáváme k tomu, abychom začali implementovat hráčům jejich chování. Zavedeme nahodilý pohyb a pro mrkev její fáze růstu.

Zadání zůstává stejné jako tomu bylo u projektu v Greenfootu, řešení se ale bude lišit. Jelikož nemáme objektový přístup, bude pohyb všech hráčů řešen pomocí bloku switch ve společné funkci oneLapActivity. Tato funkce bude udržovat chování všech hráčů. Vidíme zde zásadní rozdíl oproti OOP, kde chování hráče bylo formou metody svázané s jeho objektem a tedy oddělené od chování ostatních hráčů. Ve strukturovaném přístupu bychom sice také mohli pro každý typ hráče vytvořit samostatnou funkci, ale i tak by byla na stejné úrovni jako ostatní funkce. Navíc by to nedávalo smysl, když všichni hráči sdílí stejnou datovou strukturu a liší jen jednou její hodnotou (typem).

Zbývá ještě podotknout, že poměrně náročná práce a dlouhý čas strávený na předchozích třech lekcích, než jsme se vůbec dostali k popisu základního chování hráčů, není dán větší složitostí jazyka C oproti Javě, ale tím, že jsme si museli vytvořit kompletní běhové prostředí sami (a to stále ještě není hotové, protože oproti Greenfootu nám např. chybí snadný způsob pozastavení nebo restartování simulace či uložení výchozí podoby scény). Volba Greenfootu nám umožnila se nezabývat tvořením herní scény, ale přejít přímo k implementaci chování světa a hráčů (díky existenci tříd Actor a World). Pokud bychom projekt programovali v čisté Javě, pak bychom moseli také věnovat poměrně hodně času vytvoření okna s kreslící plochou, seznamu hráčů apod. Výhoda Javy oproti C by ovšem spočívala v tom, že řetězený seznam je v Javě v jejích knihovnách dávno naprogramovaný, podobně jako třeba zmiňovaná stromová struktura. Negativem je na druhou stranu to, že nemůžeme ovlivnit nízkoúrovňovou implementaci sami, používáme-li hotové struktury.

K implementaci růstu mrkve a automatizaci běhu simulace zavedeme časovač pomocí přerušení. Nechť si studenti nastudují v dokumentaci kapitolu Timer routines.

Úkoly s řešeními

1. Vytvořte funkci oneLapActivity tak, aby rozlišovala jednotlivé typy hráčů a implementovala jejich specifický pohyb, jak bylo popsáno v úkolech pro Greenfoot. Funkci pro generování (pseudo)náhodných čísel si dohledejte, generátor inicializujte jak je zvykem časem (hodnotou time()).

   Po nastudování funkcí pro náhodná čísla je řešení už řemeslné:

```
void oneLapActivity() {
   □ actor = firstActor;
   for (;actor!=□;actor = actor->□) {
      switch(□) {
         case AT_FOX:
            if (□()□6==0) {
               actor->direction+=□()□41-20;
            }
            actor->speed□=□()□5-2;

            actor->direction□=360; actor->direction□=360;
            actor->speed = (actor->speed□0?0:(actor->speed□10?10:□));
            break;
         case AT_FRABBIT:
         case AT_RABBIT:
            actor->direction+=(□()□7)*15-45;
            actor->speed=□()□11+4;

            actor->direction□=360; actor->direction□=360;
            actor->speed = (actor->speed□0?0:(actor->speed□10?10:□));
            break;
         case AT_CARROT:
            break;
         default:
            break;
      }
   }
}
```

2. Zaveďte novou globální proměnnou timeCounter a funkci timeInterrupt. Uvnitř této funkce se bude aktualizovat hodnota timeCounter a to tak, že s každým voláním se o jedna zvýší až do maximální hodnoty 1000, pak se bude počítat opět od 0. Funkci timeInterrupt zaregistrujte jako časovač s prodlevou 10 ms.

   Opět by tento úkol neměl činit potíže, pokud student správně pochopí dokumentaci a dohledá si příklady na internetu. Na tento příklad navazují následující dva úkoly, takže vzorové řešení bude přidáno až dále.

3. Jelikož funkce readkey blokuje vykreslovací smyčku, odstraňte její volání. Místo ní zaveďte funkci keyPressed, která bude vracet 1, pokud byla klávesa předaná jako argument stisknuta, ale jen za předpokladu, že vypršel timeout, který budete nastavovat po každém stisku klávesy do globální proměnné keyDelay a to na hodnotu direktivy KEY_DELAY. Hodnotu keyDelay snižujte o jedna v každém volání funkce timeInterrupt, hodnotu KEY_DELAY rozumně vyberte sami. Alternativní možností je zavést pole příznaků pro používané klávesy a je-li klávesa stisknuta, ihned nastavit její příznak tak, aby se zamizilo dalšímu vykonání činnosti svázané s touto klávesou. Příznak by se vymazal (ve smyčce) tehdy, když by se detekovalo, že klávesa byla puštěna. První řešení umožňuje držet klávesu a její činnost se pak vykonává opakovaně, ale jen tak často, jak její vykonání zdržuje nastavný keyDelay. Na druhou stranu znemožňuje stisknout více kláves naráz, protože timeout se nastaví ihned po detekci první klávesy pro všechny. Nicméně se zabere místo jen jedné proměnné. Druhé řešení vynucuje klávesu uvolnit, má-li být její činnost opakována. Zabírá více místa v paměti, ale umožňuje stisknout a rozpoznat více kláves naráz. Ideální je kombinované řešení, kdy se jako příznak uchovává timeOut pro každou klávesu zvlášť.

   Implementaci první varianty lze udělat jen jako jednoduchý logický výraz, ve kterém se využije líné vyhodnocování – vlastnost jazyka C, že vykonává v podmínce jen ty části, které jsou nutné k určen pravdivosti celého výroku. Ve vzorovém řešení si všimněte, že třetí část podmínky není podmínka, ale přidělení hodnoty, které bude vždy úspěšné, pokud KEY_DELAY není 0. Není tam kvůli ověření podmínky, ale proto, aby pokud uspějí první dva testy, aby se nastavila nová časová prodleva a příkaz celý vrátil 1.

```
int keyPressed(□ k) {
   return (keyDelay□0)□(key[k]==□)□(keyDelay=KEY_DELAY);
}
```

4. Nechejte ve vykreslovací smyčce ve funkci main kromě vykreslování všech hráčů i provádět pohyb všech hráčů a volat oneLapActivity. Všimněte si, že vykreslování a pohyb běží příliš rychle, protože cyklus se vykonává velmi rychle (nezávisle na přerušení časovače). Zaveďte globální proměnnou doActivity, kterou budete nastavovat v časovači na 1 jako signál podmiňující provedení aktivit a posunutí a po vykonání aktivit ji opět vynulujete. Jelikož i tak se budou zajíci pohybovat příliš rychle, zaveďte proměnnou globální timeDivider, která bude v časovači sloužit jako modulo timeCounteru a doActivity se nastaví jen bude-li aktuální timeCounter dělitelný timeDividerem.

   Pokuste se porozumět běhu času a vykonávání jednotlivých částí kódu. Do časovače je nevhodné dávat více kódu, než je nezbytné. Lepší je v něm přenastavit nějakou proměnnou, která pak aktivuje činnost přímo ve vykreslovací smyčce (jako to dělá doActivity). Časovač sám běží zcela nezávisle na ostatním kódu, spouští ho systém pomocí svých hodin v pravidelné časové úseky, bez ohledu na to, kde se zrovna nachází kód programu. Proto je také důležité dávat pozor, aby přenastavení nějaké hodnoty nemělo vliv na případný pád programu. Ze stejného důvodu používáme klíčové slovo volatile u proměnných a direktivy k zamykání proměnných a funkcí, které v přerušení vystupují: LOCK_VARIABLE a LOCK_FUNCTION. Řešení posledních tří úkolů vypadá společně takto (v místě □□□ je vynecháno několik řádků):

```
#define KEY_DELAY □
volatile □  timeCounter  = 0;
```

```
volatile □  keyDelay     = 0;
volatile □  doActivity   = 0;
volatile □  timeDivider  = 1;

void timeInterrupt() {
   timeCounter□;
   timeCounter□=1000;
   keyDelay□;
   if (keyDelay□0) keyDelay=□;
   if(□(□%□)) doActivity = 1;
}
END_OF_FUNCTION(timeInterrupt)

void init() {
        □□□
        □(time(□));

        LOCK_□(timeCounter);
        LOCK_□(timeDivider);
        LOCK_□(keyDelay);
        LOCK_□(doActivity);
        LOCK_□(timeInterrupt);
}

int main() {
    □□□
    □(timeInterrupt, 10);

    while (!key[KEY_ESC] □ !key[□]) {
       □(bgBuffer, buffer, □, □, 0, 0, SCREEN_□, SCREEN_□);
       drawAllActors(buffer, actorImages, 0);
       □(buffer, screen, □, □, 0, 0, SCREEN_□, SCREEN_□);
       if (keyPressed(□)) lastActor->type=AT_FOX;
       if (keyPressed(□)) lastActor->type=AT_RABBIT;
       if (keyPressed(□)) lastActor->type=AT_FRABBIT;
       if (keyPressed(□)) {
          lastActor->type=AT_CARROT;
          lastActor->speed=0;
          lastActor->liveTime=0;
       }

       if (keyPressed(□)) timeDivider++;
       if (keyPressed(□)&&□>1) timeDivider--;
       if (keyPressed(KEY_SLASH_PAD)&& lastActor->□!=□) lastActor = □;
       if (keyPressed(KEY_ASTERISK) && lastActor->□!=□) lastActor = □;
       if (keyPressed(□)) {
           removeActor(□);lastActor=□;
       };
       if (keyPressed(□)) {
```

```
            lastActor->direction□=10;lastActor->direction□=360;
        }
        if (keyPressed(□)) {
            lastActor = □(AT_RABBIT,SCREEN_□/□,SCREEN_□/□,10,0);
        }

        if(□) {
            oneLapActivity();
            moveAllActors();
            doActivity=□;
        }
    }

    □_int(timeInterrupt);

    deinit();
    destroy_bitmap(□);
    deallocActors();
    return 0;
}
```

5. Implementujte zrání mrkve. Ke změně barvy použijte opět podbarvení průhledného obrázku. Do struktury sActor přidejte položku liveTime, která bude určovat fázi zrání (0 až 800) a měnit se bude ve oneLapActivity. Zaveďte direktivy CARROT_EATABLE_TIME a CARROT_ROTTING_TIME, které budou obsahovat hodnoty 200 a 600 jako předěly jednotlivých fází zrání. Stačí, když implementujete tři různé barvy, každou pro jednu fázi zrání.

   Kromě zavedení direktiv je nutné jen drobně upravit oneLapActivity:

```
case AT_CARROT:
    actor->liveTime□;
    actor->liveTime□=800;
    break;
```

   a také truchu více upravit drawActor (řešení s třemi stabilními barvami):

```
case AT_CARROT:
   if (actor->□<□) {
      □fill(buffer, □-□/2+1, □-□/2+1,
                    □+□/2-1, □+□/2-1, □col(32,106,35));
   }
   if (actor->□>=□ □ actor->□<=□) {
      □fill(buffer, □-□/2+1, □-□/2+1,
                    □+□/2-1, □+□/2-1, □col(249,117,0));
   }
   if (actor->□>□) {
      □fill(buffer, □-□/2+1, □-□/2+1,
                    □+□/2-1, □+□/2-1, □col(94,73,34));
   }
   □_sprite(buffer, □, □-□/2, □-□/2);
   break;
```

Úkoly

1. Vytvořte funkci oneLapActivity tak, aby rozlišovala jednotlivé typy hráčů a implementovala jejich specifický pohyb, jak bylo popsáno v úkolech pro Greenfoot. Funkci pro generování (pseudo)náhodných čísel si dohledejte, generátor inicializujte jak je zvykem časem (hodnotou time()).

2. Zaveďte novou globální proměnnou timeCounter a funkci timeInterrupt. Uvnitř této funkce se bude aktualizovat hodnota timeCounter a to tak, že s každým voláním se o jedna zvýší až do maximální hodnoty 1000, pak se bude počítat opět od 0. Funkci timeInterrupt zaregistrujte jako časovač s prodlevou 10 ms.

3. Jelikož funkce readkey blokuje vykreslovací smyčku, odstraňte její volání. Místo ní zaveďte funkci keyPressed, která bude vracet 1, pokud byla klávesa předaná jako argument stisknuta, ale jen za předpokladu, že vypršel timeout, který budete nastavovat po každém stisku klávesy do globální proměnné keyDelay a to na hodnotu direktivy KEY_DELAY. Hodnotu keyDelay snižujte o jedna v každém volání funkce timeInterrupt, hodnotu KEY_DELAY rozumně vyberte sami. Alternativní možností je zavést pole příznaků pro používané klávesy a je-li klávesa stisknuta, ihned nastavit její příznak tak, aby se zamizilo dalšímu vykonání činnosti svázané s touto klávesou. Příznak by se vymazal (ve smyčce) tehdy, když by se detekovalo, že klávesa byla puštěna. První řešení umožňuje držet klávesu a její činnost se pak vykonává opakovaně, ale jen tak často, jak její vykonání zdržuje nastavný keyDelay. Na druhou stranu znemožňuje stisknout více kláves naráz, protože timeout se nastaví ihned po detekci první klávesy pro všechny. Nicméně se zabere místo jen jedné proměnné. Druhé řešení vynucuje klávesu uvolnit, má-li být její činnost opakována. Zabírá více místa v paměti, ale umožňuje stisknout a rozpoznat více kláves naráz. Ideální je kombinované řešení, kdy se jako příznak uchovává timeOut pro každou klávesu zvlášť.

4. Nechejte ve vykreslovací smyčce ve funkci main kromě vykreslování všech hráčů i provádět pohyb všech hráčů a volat oneLapActivity. Všimněte si, že vykreslování a pohyb běží příliš rychle, protože cyklus se vykonává velmi rychle (nezávisle na přerušení časovače). Zaveďte globální proměnnou doActivity, kterou budete nastavovat v časovači na 1 jako signál podmiňující provedení aktivit a posunutí a po vykonání aktivit ji opět vynulujete. Jelikož i tak se budou zajíci pohybovat příliš rychle, zaveďte proměnnou globální timeDivider, která bude v časovači sloužit jako modulo timeCounteru a doActivity se nastaví jen bude-li aktuální timeCounter dělitelný timeDividerem.

5. Implementujte zrání mrkve. Ke změně barvy použijte opět podbarvení průhledného obrázku. Do struktury sActor přidejte položku liveTime, která bude určovat fázi zrání (0 až 800) a měnit se bude ve oneLapActivity. Zaveďte direktivy CARROT_EATABLE_TIME a CARROT_ROTTING_TIME, které budou obsahovat hodnoty 200 a 600 jako předěly jednotlivých fází zrání. Stačí, když implementujete tři různé barvy, každou pro jednu fázi zrání.