DUM č. 2 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 08.06.2014

Ročník: studenti semináře

Anotace DUMu: Třídy Zajíc a Liška v simulaci života zajíce. Náhodný pohyb a otočení, simulace pohybu podle chování zvířete. Ovládání klávesnicí.

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 Greenfoot
Třídy zajíc a liška: pohyb

Úvod

V této lekci vytvoříme v prostředí Greenfoot třídy Rabbit (zajíc) a Fox (liška) a implementujeme jim pohyb. Pohyb bude daný zatím pravidly bez ohledu na okolí, což se jistě ukáže jako nevýhodné chování, protože zajíc nebude schopen reagovat na výskyt mrkve nebo lišky ve svém okolí, podobně jako liška nebude reagovat na výskyt zajíce. Toto chování doplníme až v 6. lekci.

Aby nebyl pohyb stále stejný, ale vykazoval prvek náhody, bude nutné používat generátor náhodných čísel. Měli bychom se studenty probrat pojem „náhody“, resp. pseudo-náhody na počítači a vysvětlit, že generování náhodného čísla je dáno matematicky číselnou řadou s rovnoměrným rozložením hodnot a volbou počáteční pozice v této řadě (seed). Je-li volena opakovaně stejná pozice, dostáváme při každém běhu stejná „náhodná“ čísla.

Studenty odkážeme na dokumentaci Javy:

http://docs.oracle.com/javase/6/docs/api/java/util/Random.html

V rámci této diskuse není na škodu naprogramovat jednoduchou aplikaci na generování náhodných čísel s pevnou a proměnlivou hodnotou seed:

```
import java.util.Random;

class RandomApp1 {   public static void main(String[] args) {
        Random rand = new Random(1);
        for (int i=0; i < 10; i++) {
           System.out.println(rand.nextInt(10));
        }
    }
}
```

Tento kód stačí upravit tak, že při vytváření nové instance třídy Random nepředáme jedničku (konstruktor sám zvolí seed). Příklad lze napsat v textovém editoru, uložit jako RandomApp1.java a zkompilovat příkazem `javac RandomApp1.java` a opakovaně spouštět `java RandomApp1` s tím, že dostáváme stále stejná „náhodná“ čísla. Někteří studenti pak třeba ocení komix http://dilbert.com/strips/comic/2001-10-25/.

Dále bude potřeba naučit se pracovat s pojmy setter a getter (metoda nastavující a vracející nějaký atribut). Za tímto účelem doporučíme studentům následující články (zdůrazníme, že na serveru Stackoverflow bývají velmi užitečné programátorské rady):

http://www.itnetwork.cz/java-tutorial-vlastnosti-gettery-settery

http://stackoverflow.com/questions/1568091/why-use-getters-and-setters

Úkoly s řešeními

1. Vytvořte třídy Rabbit a Fox. Nakreslete nebo z internetu stáhněte a upravte si vhodné obrázky. Je žádoucí, aby zajíc neměl rozměry větší než cca 50 px, liška může být proporcionálně větší. Vložte do herního pole lišku a pár zajíců. Uložte rozložení scénáře (svět) a podívejte se do třídy RabbitWorld, jak ji uložení světa ovlivnilo.

   Vkládání se provádí pravým tlačítkem a volbou „new Rabbit()“ resp. „new Fox()“, takže jakobychom vytvářeli instanci v kódu. Uložení přes Controls → Save the World, což má za následek vytvoření metody prepare ve třídě RabbitWorld a její volání v konstruktoru. Kód bychom mohli napsat přímo do konstruktoru světa ručně a trochu zjednodušeněji:

   ```
   this.addObject(new Fox(), 100, 150);
   ```

   Tím vložíme do světa nově vzniklý objekt na pozici 100, 150. Stojí za povšimnutí, že Greenfoot nevkládá „this“, což pro začátečníky kód mírně znepřehledňuje.

2. V obou třídách importujte třídu Random z balíku java.utils a vytvořte privátní atribut randomGenerator. Nechejte zajíce v každém kroku (volání act) poskočit o 4 až 14 polí a pootočit se o libovolný úhel z intervalu −45 až +45. Liška se pohybuje o 0 až 10 polí a otáčí se o čísla z intervalu −20 až +20.

   Má-li skákat či se posouvat o náhodné vzdálenosti, použijeme atribut randomGenerator:

   ```
   this.go(this.randomGenerato.nextInt(11));
   ```

   V příkladu je záměrná chyba – metoda „go“ neexistuje, pro pohyb je nutné dohledat jinou metodu, která je děděná od třídy Actor. Příklad generuje čísla mezi 0 a 10. Mají-li se generovat čísla −20 až +20, budeme generovat od 0 po 40 (argument nextInt bude 41) a odečteme od vygenerovaného čísla 20. Příklad skončí chybou (NullPointerException), pokud jsme atribut randomGenerator vytvořili, ale nepřiřadili jsme do něj novou instanci pomocí new Random();

3. Ve třídě Rabbit zajistěte, aby otočení bylo v intervalu −45 až +45 ale s krokem 15, tedy hodnoty: 0, 15, 30, 45 a stejně tak záporné.

   Náhodné číslo se generuje 0 až 6 (je 7 různých hodnot), násobí se 15 a odečítá 45.

4. Ve třídě Fox zajistěte, aby liška měnila směr náhodně často s pravděpodobností 1 ku 5 (mění směr průměrně jednou za 6 kroků)

   Změna směru musí být v podmínce. Jako podmínku využijeme generování čísla od 0 do 5 a pouze pro jedno z nich budeme lišku otáčet (v příkladu je záměrně neexistující metoda rotate, třída Actor má jinou metodu pro otočení – lze snadno najít napsáním `this.` a stiskem Ctrl-mezerník, které napovídá; nepoužívejte metodu setRotation, protože byste v následujícím kroku museli přičítat původní hodnotu získanou z getRotation, použijte raději metodu, která otáčí vůči aktuální pozici):

   ```
   if(this.randomGenerator.nextInt(6)==0) {
      this.rotate(this.randomGenerator.nextInt(41)-20);
   }
   ```

5. Liška by se měla pohybovat rovnoměrně a plynule. Proto zajistěte, aby její pohyb byl stále o 0 až 10 polí, aby se rychlost měnila mezi jednotlivými kroky nejvýše o 2. Budete asi potřebovat nový atribut pro aktuální rychlo, která by na začátku měla být 0.

   Rychlost pohybu bude určena atributem, ke kterému budeme přičítat náhodně generované číslo od −2 do +2. Nesmíme zapomenout po přičtení vygenerovaného čísla ještě dvě podmínky – první v případě, že rychlost je větší než 10 nastaví rychlost na 10, druhá nastaví 0, pokud byla menší než nula (jinak by liška couvala). Někdo ze studentů možná generování čísla v cyklu tak dlouho, dokud nevygeneruje vhodné číslo, aby rychlost nebyla mimo interval. To je krajně nevhodné řešení, protože se zbytečně čeká, až se náhodné číslo vygeneruje „správné“. Měli bychom upozornit studenty, že je-li žádoucí brát jen některá čísla, pak vždycky raději špatné vygenerované číslo nějak přepočítáme (zaokrouhlení na nejbližší povolenou hodnotu apod.) než generovat neustále další a další čísla.

6. Všimněte si, že otočí-li se zajíc nebo liška do pozice mezi 90º a 270º, pak má nohy vzhůru a hřbet dolů. Vytvořte jim převrácené obrázky a pokud je otočení v tomto intervalu, změňte obrázek (a naopak). Přemýšlejte, v jaké metodě by tato změna měla proběhnout a kdy. Podívejte se do zdrojového kódu na implementaci metod setRotation a turn a tu správnou předefinujte. Nezapomeňte, že musí dělat i to, co dělala dříve, což se udělá nejsnáze zavoláním původního kódu v předeklarovávané metodě ve vhodném místě pomocí `super.metodaPredka()` – super ukazuje na předka aktuální třídy (podobné jako this).

   Řešení je trochu těžší. Zcela jistě není vhodné, aby změna obrázku proběhla v metodě act, protože pokud bychom někdy zavolali metodu setRotation nebo turn jinde, obrázek by zůstal špatný. Otázkou zní, jestli umístit kontrolu do setRotation nebo turn nebo do obou. Odpověď nepomůže najít dokumentace, protože v ní chybí informace, že metoda turn využívá metodu setRotation. To zjistíme ve zdrojovém kódu třídy Actor. Proto změnu obrázku provedeme přepsáním metody setRotation ve své třídě, metoda turn ji užbude volat. Obrázky, které měníme, by měly být načtené při inicializaci do vhodného atributu, není dobré je neustále znovu načítat podle jména.

   Je vhodné poukázat na anotaci @Override, jejíž význam spočívá v tom, že kompilátor bude kontrolovat, že se nejedná o novou metodu, ale že opravdu přepisujeme metodu, kterou musí mít předek. Tím předejdeme chybám, pokud bychom udělali překlep.

   Řešení je tentokrát uvedené správně, ovšem pouze pro třídu Rabbit (obrázky zajicR1.png a zajicL1.png jsou umístěny v adresáři projektu v podadresáři images):

```
private GreenfootImage pictureR1 = new GreenfootImage("zajicR1.png");
private GreenfootImage pictureL1= new GreenfootImage("zajicL1.png");

@Override
public void setRotation(int rotation) {
    if ((this.getRotation()<90 || this.getRotation()>270) &&
        (rotation>=90 && rotation<=270)) {
        this.setImage(this.pictureL1);
    }
    if ((this.getRotation()>=90 && this.getRotation()<=270) &&
```

```
        (rotation<90 || rotation>270)) {
        this.setImage(this.pictureR1);
    }

    super.setRotation(rotation);
}
```

7. Vytvořte novou lišku – třída KeyboardFox, která se nebude pohybovat vůbec autonomně, ale bude se hýbat jen na základě ovládání klávesnicí. Vytvořte ji jako potomka původní lišky (třídy Fox), aby později při vytvoření dalších metod nebylo nutné je také doplňovat. KeyboardFox tedy bude mít přepsanou jen metodu act. Uprate novou lišku tak, aby šipka nahoru zvyšovala a šipka dolů snižovala její rychlost a šipky do stran ji natáčely o dva stupně. Dodržte omezení na maximální (a minimální) rychlost lišky nejlépe tak, že v původní třídě Fox implementujete setter a getter pro atribut „speed“. Uvědomte si, že poděděná třída KeyboardFox má atribut pro rychlost (speed) už od svého předka.

   Níže uvedené řešení předpokládá, že ve třídě Fox je implementovaný setter a getter pro atribut speed a to tak, že setter kontroluje (tak jako jsme nastavovali dříve) požadovanou rychlost a při hodnotě pod 0 nastaví 0 a nad 10 nastaví 10. Místo komentáře „zde získej klávesu“ je nutné opravdu získat string strisknuté klávesy, což dělá vhodná metoda jedné ze tříd prostředí Greenfoot, konkrétně třída, která s prostředím interaguje. Je dobré si všimnout, že od této třídy nemusíme vytvářet novou instanci (pomocí new) ale můžeme metodu pro stisk klávesy volat přímo nad názvem třídy, protože je statická… Bylo by hezké použít místo 4 podmínek if raději switch, ale ten lze aplikovat na řetězce až od Javy 7, kterou stávající Greenfoot nepodporuje.

```
public class KeyboardFox extends Fox
{
   public void act()
   {
      String key = Objekt.stisknutáKlávesa(); // zde získej klávesu
      if(key!=null) {
          if (key.equals("left"))  this.setSpeed(this.getSpeed()-1);
          if (key.equals("right")) this.setSpeed(this.getSpeed()+1);
          if (key.equals("up"))    this.turn(-2);
          if (key.equals("down"))  this.turn(+2);
      }

      this.move(this.getSpeed());
   }
}
```

Úkoly

1. Vytvořte třídy Rabbit a Fox. Nakreslete nebo z internetu stáhněte a upravte si vhodné obrázky. Je žádoucí, aby zajíc neměl rozměry větší než cca 50 px, liška může být proporcionálně větší. Vložte do herního pole lišku a pár zajíců. Uložte rozložení scénáře (svět) a podívejte se do třídy RabbitWorld, jak ji uložení světa ovlivnilo.

2. V obou třídách importujte třídu Random z balíku java.utils a vytvořte privátní atribut randomGenerator. Nechejte zajíce v každém kroku (volání act) poskočit o 4 až 14 polí a pootočit se o libovolný úhel z intervalu −45 až +45. Liška se pohybuje o 0 až 10 polí a otáčí se o čísla z intervalu −20 až +20.

3. Ve třídě Rabbit zajistěte, aby otočení bylo v intervalu −45 až +45 ale s krokem 15, tedy hodnoty: 0, 15, 30, 45 a stejně tak záporné.

4. Ve třídě Fox zajistěte, aby liška měnila směr náhodně často s pravděpodobností 1 ku 5 (mění směr průměrně jednou za 6 kroků)

5. Liška by se měla pohybovat rovnoměrně a plynule. Proto zajistěte, aby její pohyb byl stále o 0 až 10 polí, aby se rychlost měnila mezi jednotlivými kroky nejvýše o 2. Budete asi potřebovat nový atribut pro aktuální rychlo, která by na začátku měla být 0.

6. Všimněte si, že otočí-li se zajíc nebo liška do pozice mezi 90º a 270º, pak má nohy vzhůru a hřbet dolů. Vytvořte jim převrácené obrázky a pokud je otočení v tomto intervalu, změňte obrázek (a naopak). Přemýšlejte, v jaké metodě by tato změna měla proběhnout a kdy. Podívejte se do zdrojového kódu na implementaci metod setRotation a turn a tu správnou předefinujte. Nezapomeňte, že musí dělat i to, co dělala dříve, což se udělá nejsnáze zavoláním původního kódu v předeklarovávané metodě ve vhodném místě pomocí `super.metodaPredka()` – super ukazuje na předka aktuální třídy (podobné jako this).

7. Vytvořte novou lišku – třída KeyboardFox, která se nebude pohybovat vůbec autonomně, ale bude se hýbat jen na základě ovládání klávesnicí. Vytvořte ji jako potomka původní lišky (třídy Fox), aby později při vytvoření dalších metod nebylo nutné je také doplňovat. KeyboardFox tedy bude mít přepsanou jen metodu act. Uprate novou lišku tak, aby šipka nahoru zvyšovala a šipka dolů snižovala její rychlost a šipky do stran ji natáčely o dva stupně. Dodržte omezení na maximální (a minimální) rychlost lišky nejlépe tak, že v původní třídě Fox implementujete setter a getter pro atribut „speed“. Uvědomte si, že poděděná třída KeyboardFox má atribut pro rychlost (speed) už od svého předka.