DUM č. 5 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 27.06.2014

Ročník: studenti semináře

Anotace DUMu: Implementace rození nové generace zajíců, dělení podle pohlaví a fáze vývoje.

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
Rození zajíců

Úvod

V stávající verzi simulace se zatím neobjevuje podstatný prvek, kterým je rození nové generace zajíců, kteří jsou potravou pro lišky a tak je zřejmé, že simulace vždy po určitém čase přejde do stavu, kdy všichni zajíci i lišky vymřou. V této lekci proto doplníme funkcionalitu plození potomků. Bude k tomu potřeba (pro vyšší věrohodnost) rozlišit zajíce na samce a samice. Zavedeme také možnost ovlivnit pravděpodobnost zrození samce či samice (a tím sledovat vývoj následujících generací).

V podobných simulacích bývá zajímavé najít tzv. „rovnovážnou polohu“, tedy nastavení parametrů tak, aby populace nikdy nevymřela. Je snad zřejmé, že nalezení takové situace je tím jednodušší, čím jednodušší jsou vstupní parametry/proměnné a její nalezení komplikuje zavedení prvku náhody. Jelikož naše simulace obsahuje prvek náhody v pohybu zajíce, což má zásadní vliv na jeho schopnost najít potravu a nevyhladovět, stejně jako na nalezení partnera a zplození nové generace, můžeme buď tyto úvahy zcela vynechat nebo nechat simulaci proběhnout s různými vstupními parametry opakovaně a počítat s průměrnou hodnotou.

S nadanými nebo zvídavými studenty můžeme v této fázi projektu začít diskutovat o umělé inteligenci a učících se či genetických algoritmech. Zajímavější tato diskuse bude až v sedmé lekci, kdy zavedeme počítadla, která budou schopny počítat stav populace a případně implementovat vhodné statistické funkce. Můžeme pak zavést také tzv. hodnotící funkci, pomocí které budeme schopni vybrat z populace zajíců ty s nejlépe zvolenými parametry z hlediska přežití. Tím se už dostáváme právě ke genetickým algoritmům umělé inteligence.

Úkoly s řešeními

1. Vytvořte potomka třídy Rabbit se jménem RabbitF, který bude reprezentovat samici zajíce (zaječku). Této třídě přiřaďte barevně odlišný obrázek.

2. Vytvořte privátní metodu „bear“ ve třídě RabbitF, která nebude nic vracet ani brát žádné parametry a bude volaná metodou act. V metodě detekujte zda se objekt překrývá s objektem typu Rabbit. Pokud ano, stvořte novou instanci třídy Rabbit a vložte ji na stejné místo, jako se nachází aktuální instance RabbitF (this).

   Řešení je jednoduché a proto jej uvedeme až u další rozšiřující úlohy. Při hledání způsobu, jak vložit nový objekt do scény je nutné se zamyslet, že vkládání objektů do světa je záležitostí světa a proto bude příslušná metoda přítomna ve třídě aktuálního světa, který získáme vhodnou metodou zajíce. Pro detekci překrytí s jiným zajícem volíme stejně jako u žraní zajíce liškou raději metodu, která detekuje pouze jeden objekt, nikoliv všechny. Pokud se instance RabbitF nepohybuje, nejspíš jste přepsali metodu act – nezapomeňte, že se musí buď celá opakovaně implementovat, nebo lépe doplnit jen nové chování a pak volat `super.act()`*, tj. metodu act předka.*

3. Zvažte, kde je v konceptu z předchozího úkolu (samice jako potomek třídy Rabbit – samce) zásadní chyba, týkající se návrhu a potažmo toho, za jakých okolností samice rodí nové zajíce. Proto upravte třídu Rabbit tak, že doplníte privátní atribut „sex“. Vytvořte setter a getter. Vytvořte konstantní atributy MALE a FEMALE pro přiřazení pohlaví. Vytvořte jednoparametrický konstruktor, který umožní nastavit pohlaví, v bezparametrickém konstruktoru nastavte pohlaví na MALE. Nezapomeňte vyřešit zobrazení správného obrázku podle nastaveného pohlaví.

   Předchozí koncept má tu chybu, že zajíce rodí i dvě samičky (navíc obě). Je to dáno vlastností OOP, které se říká polymorfismus. Každá třída vystupuje i v roli libovolného svého předka, proto instance třídy RabbitF jsou zároveň i instancemi třídy Rabbit. Navíc pohlaví je pouze jedním ze znaků (atributem). Lze namítnout, že samec by vůbec neměl mít metodu „bear“ a mohli bychom raději mít abstraktní třídu AbstractRabbit se společnými atributy a metodami (bez ohledu na pohlaví) a její dva potomky RabbitM a RabbitF s implementací specifických atributů a metod daného pohlaví. Jelikož rozdíl je jen v obrázku a privátní metodě bear volané (podle pohlaví) přímo v act, není takové komplikování hierarchie tříd nutné. Při implementaci nezapomeňte podmínkovat rození samičím pohlavím. Také si uvědomte, že nastavení pohlaví v setteru nebo konstruktoru musí okamžitě změnit správný obrázek a to i vzhledem k otočení! (Proto v konstruktoru raději zavoláme setter, který to už za nás ošetří.) Řešení je komplikovanější a proto je níže uvedena třída Rabbit s vynecháním nepodstatných metod (které se neměnily) a některých podstatných slov nebo znaků symbolem □.

```
public class Rabbit extends Actor
{
    private Random randomGenerator = new □();
    private □ pictureRM= new GreenfootImage("zajicR1.png");
    private □ pictureLM= new GreenfootImage("zajicL1.png");
    private □ pictureRF= new GreenfootImage("zajicR2.png");
    private □ pictureLF= new GreenfootImage("zajicL2.png");
```

```
    private int sex;

    public □ int MALE = 0;
    public □ int FEMALE = 1;

    public Rabbit() {
        this.food = this.maxFood;
        this.setSex(□);
    }

    public Rabbit(int sex) {
        this.food = this.maxFood;
        this.□(sex);
    }

    public void setSex(int sex) {
        this.sex = sex;
        if ((this.getRotation()<90 □ this.getRotation()>270)) {
            this.setImage((□==this.MALE?this.picture□:this.picture□));
        }
        if ((this.getRotation()>=90 □ this.getRotation()<=270)) {
            this.setImage((□==this.MALE?this.picture□:this.picture□));
        }
    }

    public int getSex() {
        return this.sex;
    }

    @Override
    public void setRotation(int rotation) {
        if ((this.getRotation()<90 □ this.getRotation()>270) &&
            (rotation>=90 □ rotation<=270)) {
            this.setImage((□==this.MALE?this.picture□:this.picture□));
        }
        if ((this.getRotation()>=90 □ this.getRotation()<=270) &&
            (rotation<90 □ rotation>270)) {
            this.setImage((this.□()==this.□?this.□:this.□));
        }

        super.setRotation(rotation);
    }

    private void bear() {
        if (this.getSex()!=this.□) return;

        if (this.getOneIntersectingObject(□)□null) {
            this.getWorld().□(new Rabbit(), this.get□(), this.get□());
```

```
        }
    }

    public void act()
    {
        this.bear();
        this.dealWithFood();
        this.turn(this.randomGenerator.nextInt(7)*15-45);
        this.move(this.randomGenerator.nextInt(11)+4);
    }
}
```

4. Řešení stále obsahuje chybu v tom, že plodí i dvojice samice. Zajistěte, aby samice plodila pouze, pokud potká samce. Zajistěte pomocí generátoru pseudonáhodných čísel, aby samice zplodila potomka s pravděpodobností 1 ku 5 a aby plodila dva ku jedné samce ku samicím.

   Nejprve vyřešíme kontrolu, koho jsme potkali. Můžeme stále používat metodu beroucí pouze jediný (první) překrývající se objekt. Pokud budeme procházet všechny, je nutné nechat zplodit potomka jen při prvním nalezeném samci, nikoli všech. U počítání zda a jakého pohlaví porodit potomka je nutné napočítat, že 3 různé hodnoty potřebujeme pro případ kdy se rodí (2 pro samce a jednu pro samici) a pak ještě trojnásobek pro případy, kdy se nerodí. Proto je celkový počet možných hodnot 18. Přiložené řešení obsahuje opět vynechávky:

```
private void bear() {
   if (this.getSex()!=this.□) return;

   Rabbit partner = (□) this.getOneIntersectingObject(□);
   int    fortune = this.randomGenerator.nextInt(□);

   if (partner!=□ □ partner.getSex()==□ □ fortune<3) {
       this.getWorld().□(
                new Rabbit((fortune==□?this.FEMALE:this.MALE)),
                this.get□(), this.get□());
   }
}
```

5. Je neakceptovatelné, že konstruktor i setter pohlaví je ochoten nastavit libovolné číslo, přičemž validní pohlaví je pouze hodnota konstant MALE a FEMALE. Proto vytvořte novou třídu BadSexException dědící po Exception a vyvolejte ji v konstruktoru i setteru, pokud se bude nastavovat špatná hodnota pohlaví.

   Příklad je pouze na procvičení výjimek, které ale nesmíme zapomenout v kódu deklarovat a odchytávat všude, kam se mohou dostat… Samotná třída výjimky se vejde na jeden řádek: `public class BadSexException extends Exception{}`*. Vyvoláváme ji pouze v metodě setSex:* `if (sex!=this.MALE && sex!=this.FEMALE) throw new BadSexException();`*, ale deklarujeme klauzulí throws v záhlaví setteru i obou kontruktorů. Pak je nezbytné vytvořit try-catch bloky u všech metod, které konstruktor nebo setter používají (pozor ne u volání setteru v konstruktorech, kde se výjimka propaguje na vyšší úroveň skrze konstruktor).*

6. Vytvořte další možné pohlaví konstantou CHILD_MALE a CHILD_FEMALE, které bude reprezentovat pohlavně nedozrálé jedince, kteří nemohou plodit. Zajistěte, aby každý nově zplozený zajíc měl toto pohlaví a aby se po 500 cyklech volání metody act přenastavilo na příslušné „dospělé“ pohlaví.

   Musíme opravit setter pohlaví, aby akceptoval nové hodnoty a metodu bear, aby vytvářela nové zajíce s těmito hodnotami. Opravíme také bezparametrický konstruktor, aby nový zajíc byl CHILD_MALE a jak v setteru, tak v metodě setRotation opravíme volbu obrázku, aby nastavovala správné obrázky i pro child-varianty. Pak jen přidáme nový privátní atribut (třeba age), který bude na začátku nastavený na 0, s každým act se zvýší o jedna a jakmile dosáhne 500, přenastaví pohlaví. To vše lze uvnitř metody act nebo lépe v nové privátní metodě oneActOlder. Řešení s vynechávkami a samotnou metodou neobsahuje nikde typ int! Zvažte, jaký datový typ (lepší než int) je asi na místě vynechávky u atributu age.

```
private □ age=0;
private void oneActOlder() {
   this.age++;
   if (□>□) {
      if (this.getSex()==this.CHILD_MALE) this.setSex(this.MALE);
      if (this.getSex()==this.CHILD_FEMALE) this.setSex(this.FEMALE);
   }
}
public void act() {
   this.oneActOlder();
   this.bear();
```

7. BONUS: zajistěte, aby obrázek zajíce v dětské fázi byl poloviční velikosti než v dospělé. Použijte k tomu vhodnou metodu u obrázku, nikoliv nové obrázky.

   Řešení je na první pohled jednoduché. Do setSex a setRotation (ihned po změně obrázku) se přidá jednoduchý řádek:

```
if (this.getSex()==this.CHILD_MALE □
this.getSex()==this.CHILD_FEMALE)
this.getImage().scale(this.getImage().getWidth()/2,
this.getImage().getHeight()/2);
```

   Jenže to nebude fungovat a sice kvůli tomu, že se budou obrázky neustále zmenšovat, až bude jejich velikost nulová, což vyvolá výjimku. Důvod je v tom, jak obrázky přiřazujeme. Předáváme neustále jen odkaz na tutéž jedinou instanci. Přiřazením `this.setImage(this.pictureLM)` *se nepřiřadí kopie obrázku z* `this.pictureLM`*, ale přímo odkaz na tento obrázek. Jestliže pak zavolám metodu* `scale` *nad* `this.getPicture()`*, de facto ji volám nad obrázkem uloženým v* `this.pictureLM`*. A s každým novým voláním ho znovu zmenšuji. Abychom opravdu předali novou kopii obrázku, je nutné vytvořit nový GreenfootImage – naštěstí jeden jeho konstruktor umožňuje převzít jinou instanci GreenfootImage a vytvořit její kopii. Musíme tedy striktně začít používat variantu* `this.setImage(new GreenfootImage(this.pictureLM));`*, která nastaví obrázek na kopii* `this.pictureLM`*, nikoliv „originál“.*

Úkoly

1. Vytvořte potomka třídy Rabbit se jménem RabbitF, který bude reprezentovat samici zajíce (zaječku). Této třídě přiřaďte barevně odlišný obrázek.
2. Vytvořte privátní metodu „bear“ ve třídě RabbitF, která nebude nic vracet ani brát žádné parametry a bude volaná metodou act. V metodě detekujte zda se objekt překrývá s objektem typu Rabbit. Pokud ano, stvořte novou instanci třídy Rabbit a vložte ji na stejné místo, jako se nachází aktuální instance RabbitF (this).
3. Zvažte, kde je v konceptu z předchozího úkolu (samice jako potomek třídy Rabbit – samce) zásadní chyba, týkající se návrhu a potažmo toho, za jakých okolností samice rodí nové zajíce. Proto upravte třídu Rabbit tak, že doplníte privátní atribut „sex“. Vytvořte setter a getter. Vytvořte konstantní atributy MALE a FEMALE pro přiřazení pohlaví. Vytvořte jednoparametrický konstruktor, který umožní nastavit pohlaví, v bezparametrickém konstruktoru nastavte pohlaví na MALE. Nezapomeňte vyřešit zobrazení správného obrázku podle nastaveného pohlaví.
4. Řešení stále obsahuje chybu v tom, že plodí i dvojice samice. Zajistěte, aby samice plodila pouze, pokud potká samce. Zajistěte pomocí generátoru pseudonáhodných čísel, aby samice zplodila potomka s pravděpodobností 1 ku 5 a aby plodila dva ku jedné samce ku samicím.
5. Je neakceptovatelné, že konstruktor i setter pohlaví je ochoten nastavit libovolné číslo, přičemž validní pohlaví je pouze hodnota konstant MALE a FEMALE. Proto vytvořte novou třídu BadSexException dědící po Exception a vyvolejte ji v konstruktoru i setteru, pokud se bude nastavovat špatná hodnota pohlaví.
6. Vytvořte další možné pohlaví konstantou CHILD_MALE a CHILD_FEMALE, které bude reprezentovat pohlavně nedozrálé jedince, kteří nemohou plodit. Zajistěte, aby každý nově zplozený zajíc měl toto pohlaví a aby se po 500 cyklech volání metody act přenastavilo na příslušné „dospělé“ pohlaví.
7. BONUS: zajistěte, aby obrázek zajíce v dětské fázi byl poloviční velikosti než v dospělé. Použijte k tomu vhodnou metodu u obrázku, nikoliv nové obrázky.