DUM č. 4 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 08.06.2014

Ročník: studenti semináře

Anotace DUMu: Interakce mezi objekty. Požírání zajíce liškou, požírání mrkve zajícem. Implementace vyhladovění a úbytku energie pohybem.

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
Žraní mrkve a zajíců

Úvod

V této lekci se zaměříme na interakci instancí mezi sebou. Zatím se každý objekt vložený do herního plánu pohyboval volně a vzájemně se nijak neovlivňovaly. Nyní implementujeme funkcionalitu, která začne dělat simulátor zajímavější – zajíci budou žrát mrkev a lišky budou žrát zajíce.

Všechny potřebné metody budou tentokrát vycházet pouze z kódu tříd Greenfootu a to hlavně třídy Actor a třídy World. Studenty necháme najít si potřebné v dokumentaci metody samotné. Při té příležitosti zmíníme, že dokumentace, kterou čtou, je vygenerovaná JavaDocem automaticky z komentářů v kódu a proto je nejvyšší čas tyto komentáře doplnit, pokud tak ještě neučinili v předchozích lekcích. Jakmile komentáře doplní, mohou si v okně s kódem v pravém horním rohu přepnout na zobrazení dokumentace a prohlédnout si výsledek.

Dokumentaci k třídám Greenfootu najdeme jednak v instalaci programu v menu a také na webových stránkách http://www.greenfoot.org/doc v online HTML podobě nebo i stručně v PDF tabulce. Je důležité, aby si studenti navykli sami hledat v dokumentaci, sami volit vhodné metody a nespoléhali na předložené informace od učitele. Argumentujeme tím, že v praxi budou dostávat pod ruce stále nové a nové systémy a knihovny a budou nuceni se v nich sami zorientovat.

Úkoly s řešeními

1. Doplňte do třídy Rabbit privátní atributy food a maxFood typu integer. První určuje aktuální nasycenost zajíce, druhý určuje maximální možnou sytost (velikost žaludku). Inicializujte maxFood na vhodnou hodnotu (např. 2000 – s ohledem na délku cyklu růstu mrkve), doplňte konstruktor a v něm inicializujte i food na hodnotu maxFood. Vytvořte setter a getter pro food i maxFood, nedovolte zadávat čísla menší než 1 a pro food nedovolte číslo větší než maxFood. Vytvořte také metodu addFood(int amount), která bude k food přičítat množství dané parametrem amount. Pokud by hodnota food přesáhla maxFood, nastavte maxFood, pokud by byla méně než 0, nastavte 0. Přidejte také číselnou konstantu carrotFoodConstant, kterou nastavte na 200 a která bude určovat, kolik energie přidá jedna snědená mrkev.

   Vytvoření atributů je triviální úloha, která nevyžaduje návod, musí zvládnout každý student zcela sám. Atribut carrotFoodConstant musí být „final“ jakožto konstanta. V setterech a addFood použijeme podmínky pro omezení povolených hodnot, nejlépe tak, že nejprve nastavíme i špatnou hodnotu, ale pak pokud je menší než 0 opravíme na 0, když větší než maxFood, opravíme na maxFood. To lze udělat i ternárním operátorem (příklad by se hodil do addFood, lepší ale bude umístit test do setFood a v addFood volat `this.setFood(this.getFood()+amout)`*, čímž se vyhneme dvojí implementaci testování a z toho plynoucí hrozby nekonzistence):*

   ```
   this.food += amount;
   this.food = (this.food<0?0:this.food);
   this.food = (this.food>this.maxFood?this.maxFood:this.food);
   ```

2. Vytvořte v třídě Rabbit metodu dealWithFood, která nepřebírá žádné argumenty a nic nevrací a je protected, aby byla přístupná i potomkům, ale ne jiným třídám. V této metodě nejprve snižte food o 1, pak pomocí vhodné metody detekujte, zda zajíc nestojí v místě mrkve. Pokud ano, zavolejte na této instanci mrkve metodu eatOne a v případě, že vrátí true, zvyšte zajíci metodou addFood jeho sytost o carrotFoodConstant. Tuto metodu volejte v zajícově metodě act. Podívejte se do zdrojových kódů Greenfootu, jak je metoda, pomocí které se detekuje střet s jiným objektem, implementovaná.

   Metoda dealWithFood je nejzajímavější v detekci střetu s jiným objektem. Každý potomek třídy Actor k tomu má k dispozici dvě možné metody, z nichž jedna vrací celý seznam všech kolidujících objektů. Využívá se k tomu třída java.util.List, což je jedna z tzv. kolekcí Javy – složitější datová struktura, odpovídající v principu poli, ale na objektové úrovni a implementovanému různými způsoby (např. ArrayList pomocí pole nebo LinkedList jako zřetězený seznam). Umožňuje to kontrolovat více v jednom místě kolidujících objektů. Nám stačí vybrat pouze jeden „záhon mrkve“ a proto využijeme druhou možnou metodu, která vrací pouze jeden („první“) objekt, který se střetává. Za povšimnutí stojí, že vrací třídu Actor, nikoli Carrot. To ale nevadí, protože na základě OOP principu polymorfismu, je Carrot jakožto potomek také Actor. Abychom ale mohli zavolat metodu eatOne, bude nutné získaný Actor přetypovat na Carrot. Abychom měli jistotu, že protínající se objekt je opravdu instance třídy Carrot, musíme jako parametr nalezené metody pro hledání kolidujících objektů předat „class“ objektu, který chceme najít. Příslušný parameter je „Carrot.class“. Řešení bude uvedeno za dalším úkolem.

3. Doplňte do metody dealWithFood možnost vyhladovění k smrti. Pokud zajíc ani po testu na přítomnost mrkve nemá hodnotu food větší než 0, nechejte ho odstranit z herního plánu. Nalezněte k tomu vhodnou metodu v dokumentaci některé ze tříd Greenfootu.

   Objekt nedokáže odstranit sám sebe. Objekty z herního plánu lze odstraňovat pouze metodou herního plánu, které daný objekt předáme (pokud chceme aktuální instanci, tak „this“). Řešení s vynechanými některými metodami/slovy:

```
protected void dealWithFood() {
    this.addFood(-1);
    if(this.□(Carrot.class)!=null) {
        if (((Carrot) this.□(Carrot.class)).eatOne()) {
            this.addFood(this.carrotFoodConstant);
        }
    }

    if (this.getFood()<1) {
        this.get□().remove□(this);
    }
  }
```

4. Implementujte lišce totéž, co zajíci v prvním úkole, pouze konstanta se bude jmenovat rabbitFoodConstant a její hodnota bude 500. Liška bude mít také maxFood roven 4000.

   Použijeme kopírování kódu, jen je nutné dávat pozor, abychom opravili vše, co se mění.

5. Nechejte lišku sežrat zajíce, pokud se překrývají. Doplňte také vyhladovění (implementujte u lišky podobnou metodu jako je dealWithFood u zajíce).

   Metoda dealWithFood je u lišky prakticky stejná, pouze se nemusí přetypovávat a volat eatOne, ale nalezený zajíc se odstraní ze světa. Settery a gettery zůstávají stejné, musí se jen změnit název konstanty a konstruktoru. Nesmíme zapomenout přidat dealWithFood do metody act. Zkrácené řešení:

```
protected void dealWithFood() {
    this.addFood(-1);
    if(this.□(Rabbit.class)!=null) {
        this.get□().remove□(this.□(Rabbit.class));
        this.addFood(this.rabbitFoodConstant);
    }

    if (this.getFood()<1) {
        this.get□().remove□(this);
    }
}
```

6. Jistě jste zaregistrovali, že v herním plánu je někdy špatné pořadí zobrazování objektů. Zajíci jsou pod obrázkem mrkve apod. Opravte v konstruktoru světa RabbitWorld, aby bylo pořadí takové, že vespod je mrkev, nad ní se pohybují lišky a nad liškami králíci v nejvyšší vrstvě. Slouží k tomu vhodná metoda, které se také předává „class“ jako metodě pro detekci kolizí.

   Řešením je jediný příkaz: `this.set□(□.class,Fox.class,□.class);`

7. Ověřte, zda klávesnicí ovládaná liška také funguje jak má (žere zajíce). Pokud ne opravte.

   Důvodem, proč KeyboardFox nežere zajíce (a ani neumírá hlady) je, že přepisuje metodu act svého předka, třídy Fox a nikde metodu act předka nevolá (ačkoliv je v ní důležitý příkaz)…

Úkoly

1. Doplňte do třídy Rabbit privátní atributy food a maxFood typu integer. První určuje aktuální nasycenost zajíce, druhý určuje maximální možnou sytost (velikost žaludku). Inicializujte maxFood na vhodnou hodnotu (např. 2000 – s ohledem na délku cyklu růstu mrkve), doplňte konstruktor a v něm inicializujte i food na hodnotu maxFood. Vytvořte setter a getter pro food i maxFood, nedovolte zadávat čísla menší než 1 a pro food nedovolte číslo větší než maxFood. Vytvořte také metodu addFood(int amount), která bude k food přičítat množství dané parametrem amount. Pokud by hodnota food přesáhla maxFood, nastavte maxFood, pokud by byla méně než 0, nastavte 0. Přidejte také číselnou konstantu carrotFoodConstant, kterou nastavte na 200 a která bude určovat, kolik energie přidá jedna snědená mrkev.

2. Vytvořte v třídě Rabbit metodu dealWithFood, která nepřebírá žádné argumenty a nic nevrací a je protected, aby byla přístupná i potomkům, ale ne jiným třídám. V této metodě nejprve snižte food o 1, pak pomocí vhodné metody detekujte, zda zajíc nestojí v místě mrkve. Pokud ano, zavolejte na této instanci mrkve metodu eatOne a v případě, že vrátí true, zvyšte zajíci metodou addFood jeho sytost o carrotFoodConstant. Tuto metodu volejte v zajícově metodě act. Podívejte se do zdrojových kódů Greenfootu, jak je metoda, pomocí které se detekuje střet s jiným objektem, implementovaná.

3. Doplňte do metody dealWithFood možnost vyhladovění k smrti. Pokud zajíc ani po testu na přítomnost mrkve nemá hodnotu food větší než 0, nechejte ho odstranit z herního plánu. Nalezněte k tomu vhodnou metodu v dokumentaci některé ze tříd Greenfootu.

4. Implementujte lišce totéž, co zajíci v prvním úkole, pouze konstanta se bude jmenovat rabbitFoodConstant a její hodnota bude 500. Liška bude mít také maxFood roven 4000.

5. Nechejte lišku sežrat zajíce, pokud se překrývají. Doplňte také vyhladovění (implementujte u lišky podobnou metodu jako je dealWithFood u zajíce).

6. Jistě jste zaregistrovali, že v herním plánu je někdy špatné pořadí zobrazování objektů. Zajíci jsou pod obrázkem mrkve apod. Opravte v konstruktoru světa RabbitWorld, aby bylo pořadí takové, že vespod je mrkev, nad ní se pohybují lišky a nad liškami králíci v nejvyšší vrstvě. Slouží k tomu vhodná metoda, které se také předává „class“ jako metodě pro detekci kolizí.

7. Ověřte, zda klávesnicí ovládaná liška také funguje jak má (žere zajíce). Pokud ne opravte.