DUM č. 9 v sadě

28. Inf-4 Jednoduchá hra Had ve Flashi (ActionScript)

Autor: Robert Havlásek

Datum: 16.03.2013

Ročník: 5AV

Anotace DUMu: Flash - teorie: Syntaxe cyklů for, while, do-while. Přerušení pomocí continue a break.

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

Cyklus for

Cyklus for zajišťuje opakované provádění příkazů. Jeho zápis je:
`for (počáteční_výraz; koncová_podmínka; iterační_výraz)` *`{blok_příkazů}`*
případně
`for (počáteční_výraz; koncová_podmínka; iterační_výraz)` *`příkaz`*

Přesný postup provádění for cyklu je tento:
krok 0: provede se (poprvé a naposledy) počáteční_výraz
krok 1: otestuje se, zda je koncová_podmínka pravdivá, a pokud není, cyklus skončil (Flash pokračuje dalším příkazem za cyklem)
krok 2: provedou se příkazy uvnitř cyklu (tzv. tělo cyklu)
krok 3: provede se iterační_výraz a program pokračuje krokem 1.

Obvyklý postup je řízení for-cyklu pomocí nějaké číselné proměnné (označované jako řídící proměnná cyklu), která postupně nabývá hodnot od–do, například:

```
for (i=0; i<5; i++) {trace(i);}
                        // vypíše čísla 0, 1, 2, 3, 4, pro pětku už se příkaz trace neprovede
for (i=10; i>=0; i--) {trace(i);}
                       // vypíše 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```

Iter. výrazy zde používám ve formě `i++` (resp. `i--`), která znamená `i=i+1` (resp. `i=i-1`).

Studentům zadáváme následující kusy kódu s výzvou, aby řekli, co vypíše:

```
for (i=5; i<=5; i++) {trace(i);}
```
// vypíše jen 5

```
for (i=5; i>=5; i++) {trace(i);}
```
// vypíše 5, 6, 7, atd., nikdy neskončí
Pedagogická poznámka: Studenti, zkuste tento kód na PC spustit, ať víte, jak Flash zareaguje.

```
for (i=3; i!=5; i++) {trace(i);}
```
// vypíše 3, 4

```
for (i=3; i!=3; i++) {trace(i);}
```
// nevypíše nic, podmínka neprojde prvním testem

```
for (c='A'; c<='D'; c++) {trace(i);}
```
// vypíše undefined, undefined, undefined, undefined (vypisujeme nedefinované i, takový test pozornosti… Kdybychom vypisovali c, odpověď by byla: vypíše A, B, C, D.

```
for (i=0; i<3;) {trace(i++);}
```
// vypíše 0, 1, 2, iterační výraz není nutný, je ale vhodné, aby se řídící proměnná měnila v některém z příkazů uvnitř cyklu.

Všimněte si, že příkaz `i++` sice i zvýší, ale vrátí původní hodnotu. Tedy `i=0;trace(i++)` nejdřív vypíše 0 a pak do i přiřadí o jedno větší číslo. Naopak příkaz `++i` vrátí již zvýšenou hodnotu, tedy `i=0;trace(++i)` vypíše 1.

Cyklus for-in

Chceme-li procházet polem (které nemusí mít nutně indexy ve formě čísel), použijeme cyklus for-in. Jeho syntaxe je:

```
for (proměnná in objekt_typu_pole) {blok_příkazů}
```

či

```
for (proměnná in objekt_typu_pole) příkaz
```

For-in cyklus nemusí procházet položkami uvnitř pole, může též procházet vlastnostmi uvnitř objektu. Procházet též můžeme objekty uvnitř jiného objektu, například:

```
for (jmeno in _root) {trace(jmeno);}
```

vypíše jména všech objektů v ploše, tedy: všech proměnných, které jsou v Actions plochy zavedeny a dále všech pojmenovaných instancí všech objektů plochy.
Velmi zajímavý výstup získáme, pokud tento kód vložíme za poslední řádek Actions plochy v našem projektu hada:

```
m
obsazeno
k
i
klonzdi
zdiy
zdix
smrt
$version
klonzdi[124]
...
klonzdi[0]
vzorzdi
hlava
```

Nejprve jsou uvedeny běžné proměnné, pak vestavěná proměnná $version, pak položky pole, na konci výpisu jsou jména instancí objektů na ploše (vzorzdi, hlava).

Cyklus while

Cyklus s podmínkou na začátku (cyklus while) má tvar:

```
while (podmínka) {blok_příkazů}
```

či

```
while (podmínka) příkaz
```

Nejdříve program vyhodnotí podmínku a teprve je-li splněna, vykonají se příkazy cyklu. Poté je opět vyhodnocena podmínka, atd. Není-li podmínka splněna, pokračujeme dalším příkazem za cyklem.

Cyklus do-while

Cyklus s podmínkou na konci (cyklus do-while) má vždy stejný tvar:

```
do {blok_příkazů} while (podmínka);
```

Blok příkazů musí být vždy ve složených závorkách, i kdyby šlo o jeden příkaz.
Cyklus do-while provede příkazy ve svém těle, vyhodnotí podmínku; vše se opakuje tak dlouho, dokud je podmínka splněna. Končí, když podmínka splněna není.
Příkazy uvnitř cyklu do-while jsou vykonány vždy alespoň jednou, proto je vhodný například při interakci s uživatelem (zadá data, jsou-li špatně, zadá data, atd.)

Příklady na cyklus while a cyklus do-while v této fázi nedělám, zejména z časových důvodů.

V jistých případech je pohodlnější uvnitř těla cyklu ukončit jeden aktuální běh nebo celý cyklus. Taková ukončení fungují u všech zmiňovaných cyklů (for, for-in, while, do-while). Programátorsky to není úplně „čistá“ varianta, ale platí zde, že účel světí prostředky.

Násilné ukončení jednoho průběhu těla cyklu – příkaz continue

Příkaz continue ukončí daný (jeden) průběh těla. Typické použití:

```
for (počáteční_výraz; koncová_podmínka; iterační_výraz)
  {příkazy;
   if (podmínka) continue;
   další_příkazy }
```

Je-li v daném průběhu těla podmínka za if splněna, provede se continue a skočí se znovu na další průběh for-cyklu od začátku (část „další_příkazy“ se přeskočí).
Například:

```
s=0;
for (i=1; i<=100; i++)
  { if (random(10)>0) continue;
     trace(i);
     s=s+i;
  }
trace("Soucet je "+s);
```

Tento kus kódu pro každé číslo od 1 do 100 nejprve náhodně zvolí, zda je použije (random(10) nabývá celých čísel 0 až 9, podmínka ukončí tělo náhodně v cca 90% případů), pokud je použije, vypíše je na obrazovku a přidá do součtu.

Násilné ukončení celého cyklu – příkaz break

Příkaz break ukončí celý cyklus. Typicky, v případech, že něco hledáme a právě nalezneme, nemá smysl v hledacím cyklu pokračovat.
Například, psali-li jsme v minulém DUMu č. 8 na 3. straně kód pro zjišťování, zdali je místo, kam generujeme zeď, již obsazené:

```
var obsazeno=0;
for (m=0; m<zdix.length; m=m+1)
   if ((_root.vzorzdi._x==zdix[m])&&(_root.vzorzdi._y==zdiy[m]))
      obsazeno=1;   //najdeme-li v prubehu cyklu zed na stejnem miste
```

bude určitě rychlejší, když při nálezu cyklus ukončíme:

```
var obsazeno=0;
for (m=0; m<zdix.length; m=m+1)
   if ((_root.vzorzdi._x==zdix[m])&&(_root.vzorzdi._y==zdiy[m]))
      {obsazeno=1;break;} //najdeme-li v prubehu cyklu zed na stejnem miste
```

Studenti, kód ve svém programu (přibližně 22. řádek kódu plochy) můžete poupravit.
Podobně u zjišťování narážení do zdi poupravte přibližně 19. řádek kódu hlavy na

```
for (m=0; m<_root.zdix.length; m=m+1)
   if ((_root.hlava._x==_root.zdix[m])&&(_root.hlava._y==_root.zdiy[m]))
      {obsazeno=1;break;}
```