helyett a lekérdezést úgy alakította ki, hogy az adatbázisnak csak 2000 adatra kellett figyelnie. „Az Access buta – mondja Calabrese –, az összes rekordot kézbe veszi és minden egyes alkalommal az összeset átnézi. Ez rendkívül pazarló módszer. Jelenleg harminc alkalmazottunk van, és ha történetesen minden számítógéppel egyszerre próbálnák meg elérni az adatbázist, ez igen hamar nagyon komoly sebességcsökkenést okozna."

A PostgreSQL ellenőrzése

A változtatás következő lépése a lekérdezések hibaellenőrzése, ahol mindjárt két út közül választhatunk. Az egyik a PostgreSQL ODBC-meghajtójához tartozó ellenőrzőeszközök használata. A meghajtóval készíttethetünk egy naplófájlt, és amikor az Access SQL-parancsot küld, a PostgreSQL azt azonnal bevezeti a naplóba, mely a C meghajtó gyökérkönyvtárában található. Ezzel elcsíphetjük az Access olyan ügyetlen próbálkozásait, amikor mondjuk százezer sort próbál egyszerre behívni. Ebben az esetben például a lekérdezést ezer kisebbre bonthatjuk szét. Ez a napló igazából egy nyomkövetés, mely segítségével gyorsan kiszúrhatjuk, ha valami hiba lépett fel, mint ahogy itt is történt.

```
conn=86311032, query=' '

CONN ERROR: func=SQLDriverConnect,
desc='Error from CC_Connect',
errnum=105, errmsg='The database does not
exist on thet server or user authentication
failed.'
```

Azt is megtehetjük, hogyha az Access lekérdezést küld, és a rendszer leáll, a kiszolgálóoldal hibakezelési szintjét (***Debug level***) átállítva a kérelmet csak azért is kiolvassuk. A finomhangolás lényege, hogy minden egyes képernyőn végig kell haladnunk és a kérelmek egyszerűsítésével, összevonásával gyorsítanunk kell őket. Az SQL-t jól ismerők tudják, hogy a rendszer milyen összetett, így elmondhatjuk, hogy fáradságos munka elébe nézünk. Ha azonban a fejlesztésnek ezen a pontján elvégezzük a megfelelő ellenőrzéseket, rengeteg későbbi fejfájástól kímélhetjük meg magunkat.
Mielőtt a rendszer működését visszaállítanánk, a kipróbálás következik. Calabrese folyamatosan figyelte a Bike Friday adatbázisrendszerét, miközben az irodákban már használták a rendszert. „Nemcsak azt kell kipróbálnunk, hogy akadnak-e a kezelőfelületnek hibái, hanem azt is, hogy mekkorára kell a kiszolgálót terveznünk" – mondja Calabrese. Írt egy lekérdező héjprogramot is, amely a három fő gondot okozó részegység (processzor, merevlemezek, hálózat) terheltségét kíséri figyelemmel.
Calabrese programja a processzor kihasználtságát aszerint ellenőrizte, hogy a terhelés hány másodpercig maradt 100, 50 és 0 százalékos. A lemez adatátvitelének értékelését úgy végezte, hogy hány olvasási és írási művelet zajlik éppen, illetve mérte az ezek során átvitt kilobájtokat is. A hálózat terhelését a másodpercenkénti csomagszámmal és a másodpercenként átvitt bájtok számával írta le. Calabrese azt is javasolja, hogy a lezárt hálózati szakaszban végezzünk árasztásos pingelést (`ping -F`), így meghatározhatjuk, hogy a kiszolgáló mekkora terhelésnél akad meg. A memóriával egyszerű a helyzet: minél több van belőle, a PostgreSQL annál többet használ föl, és így annál gyorsabb lesz az adatbázis működése.
Természetesen az adatbázis sebességéről a felhasználók véleménye árulkodik leginkább. A lépésenkénti apró várakozási idők a valóságban hatalmas késésekké adódhatnak össze.

Minden vállalatnál, szervezetnél kialakul egy vélemény arról, mi számít lassúnak és mi elfogadhatóan gyorsnak. Így a rendszer főpróbája mindenképpen az lesz, amikor a felhasználók pár órás használat után kimondják a végítéletet: „Csigalassú" vagy „Hm, nem is olyan rossz".
Végül, miután a rendszer megfelelőnek találtatott, mi pedig a kezelőfelület minden hibáját kijavítottuk, máris nekiláthatunk egy nyílt forrású alapokra építkező e-üzleti rendszer kiépítésének.

Chris Volpe
(chris@macnet2.com)
New Hampshire-ben él és technológiai leírásokat készít.

mascTovábbi érdekességek

Bruce Momjian: PostgreSQL: Introduction and Concepts (ISBN: 0-201-70331-9, 44,95 dollár, 544 oldal)
➲ http://www.ca.postgresql.org/docs/awbook.html címen érhető el.
A gép és az alkatrészek, valamint a PostgreSQL összehangolásáról olvassuk el *Momjian* írását „PostgreSQL Performance Tuning" címmel
➲ http://www.linuxjournal.com/lj-issues/issue88/4791.html
F. Scott Barker: Microsoft Access 2000 Power Programming (ISBN: 0-672-31506-8, 49,99 dollár, 1332 oldal, CD-ROM)
PostgreSQL 7.1 kézikönyv
➲ http://www.ca.postgresql.org/users-lounge/docs/7.1/reference
Postgres GYK
➲ http://www.ca.postgresql.org/users-lounge/docs/#7.1
Postgres/Access GYK
➲ http://joelburton.com/resources/pgaccess
PostgreSQL-leírás és Data Migration Tools
➲ http://postgresql.crimelabs.net/users-lounge/docs
Migration Tools: a csomagban a *pgAdmin* (grafikus PostgreSQL-vezérlőfelület), a *phpPgAdmin* (webalapú eszköz a pgAdminhoz hasonló feladatokra) és a PsqlODBC Windows-meghajtó található. Ez utóbbi lehetővé teszi, hogy a PostgreSQL-adatbázist az ODBC-meghajtókon keresztül elérő Windows-alkalmazásokat írjunk.
Még egy érdekesség: az exSQL új, nyilvános változata is elérhető. Az exSQL nagyszerű PostgreSQL-átalakító eszköz. Az új változat, amely az indexeket és az idegen kulcsokat megbízhatóbban kezeli, a
➲ http://www.geocities.com/musica_6898/postgresaccess_home.html címről tölthető le. A PostgreSQL az idegen kulcsokat a táblázatok összekapcsolására és kapcsolataik kódolására használja. A *Michael Calabrese* által írt exSQL-változat módosítja azokat a szabályokat, melyek meghatározzák, hogy az egyes Access-mezőkből milyen PostgreSQL-mezőtípusok készíthetők. További egyszerűsítések és hibajavítások mellett ez a változat tartalmaz egy parancsfájlt, amely az Access egyik hibáját küszöböli ki: alapértelmezés szerint a program szövegessé alakítja a pénzmezőket. A parancsfájl felülbírálja ezt az alapértelmezést, így futás közben ebből hibák adódhatnak.

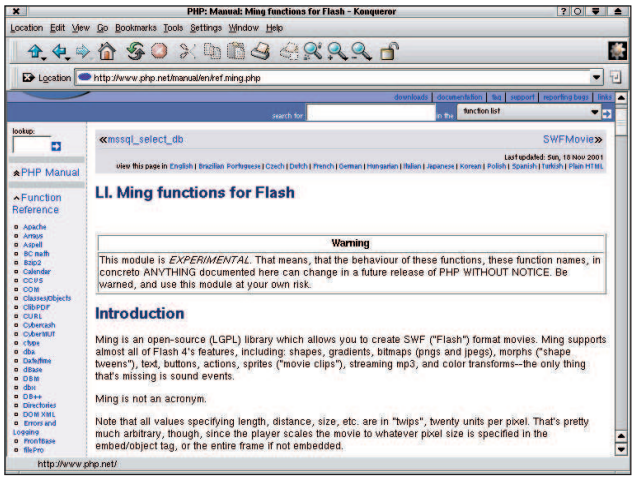
A PHP és a MING

Hogyan készítsünk weblapunkra röptében Flash-mozikat?

Napjainkban a Flash-féle pörgő-forgó csodát nehéz kikerülni a Világhálón. Nem is kell, hiszen a Flash-lejátszás az összes újabb linuxos böngészőben lehetséges. Kedvenc operációs rendszerünkön manapság az ilyen mozik létrehozása sem elérhetetlen cél. A ***MING*** könyvtár segítségével és PHP-támogatással weblapjainkat elláthatjuk röptében előállított animációkkal. Cikkünkben a MING Linuxra telepítéséről, valamint a PHP-vel történő összeházasításáról lesz szó. Emellett a cikk kínálta lehetőségekhez mérten igyekszem a MING működését is bemutatni.

A MING beszerzése és telepítése

A PHP MING kiegészítésének telepítését Debian Woody rendszeren a PHP 4.0.6-os változatához mutatom be. Kis módosításokkal természetesen más GNU/Linux-változatokon is üzembe helyezhető. Tekintettel arra, hogy a támogatás csak a 4.0.5-ös változattól került a PHP-ba, érdemes a gépünkön egy friss PHP-val próbálkozni.



1. kép A MING-függvények részletes leírása a PHP-kézikönyvben

A MING hivatalos weblapját megnyitva hamar szembesülhetünk vele, hogy nem vagyunk magunkra hagyatva. Innen a legfrissebb PHP-változatokhoz azonnal letölthetjük az előre lefordított ***php_ming.so*** modult. Jó esetben akár lusták is lehetünk, és a MING-kiegészítést fordítás nélkül beizzíthatjuk, csupán ezt a fájlt szükséges a PHP-kiterjesztéseket tartalmazó könyvtárba másolnunk. Ennek helyét könnyen megtudhatjuk, ha a parancssorban kiadjuk a `php-config --extension-dir` parancsot. A másolás mellett még a ***php.ini*** fájlba is bele kell nyúlnunk, és a következő bejegyzést kell beillesztenünk:

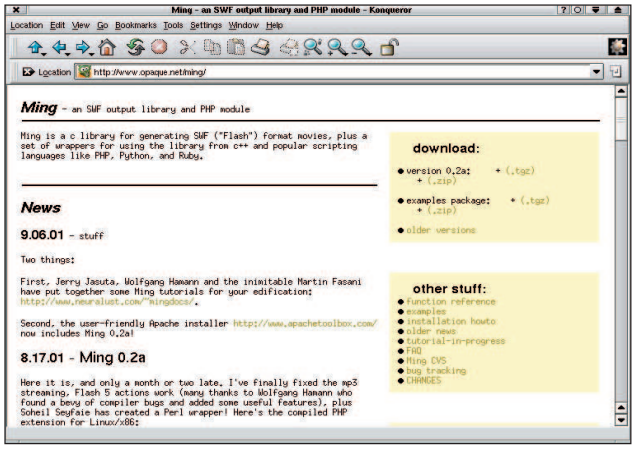
```
extension=php_ming.so
```

Amennyiben ezzel megvagyunk és minden egyezik, máris rendelkezünk a Flash-mozik létrehozásához szükséges eszköztárral. Ne feledkezzünk meg webkiszolgálónk újraindításáról, amennyiben az a beállításváltozások érvényre juttatásához szükséges.
Előfordulhat, hogy a folyamat nem ilyen egyszerűen zajlik le, ekkor sem kell kétségbe esni, a ***php_ming.so*** kiegészítést mi magunk is könnyen létrehozhatjuk. Ehhez először be kell szereznünk a MING forráskódját, majd le kell fordítanunk és telepítenünk (ehhez a parancsokat a MING forráskönyvtárából adjuk ki):

```
make
make install
```

Ezáltal a ***/usr/lib*** alá létrejön a ***limbing.so*** állomány, és egy ***ming.h*** is megfelelő helyére kerül a ***/usr/include*** könyvtárban. Ezáltal megteremtettük a feltételét, hogy a MING-támogatást a jelenlegi PHP-rendszerünkbe építsük. A további szükséges lépéseket már a PHP-forráskönyvtárából kell elvégeznünk:

```
./buildconf
./configure --with-ming <egyØb kapcsol k>
make
make install
```



2. kép A MING oldala ➲ http://www.opaque.net/ming/

A szükséges könyvtárat tehát létrehoztuk, és a helyére is került. A ***php.ini***-nek a fenti bejegyzéssel történő kiegészítése természetesen ekkor is szükséges.

Az ismerkedés

A nagy fejesugrás előtt nem árt néhány dolgot tisztázni: a pontosság és a jó nagyíthatóság érdekében bevezették a ***twip*** mértékegységet. Hogy értsük, mit is takar ez: húsz twip tesz ki egy képpontot. A mozi méretei, azon belül is minden elemnek a mérete, a távolságok mind ebben az egységben értelmezendők. Alapesetben tehát egy 200×200 képpontmérettel rendelkező mozihoz 4000×4000 twip méretű valódi munkafelület tartozik. A másik fontos tudnivaló, hogy a MING jelen pillanatban csak

az FDB-típusú betűkészletek megjelenítésére alkalmas. Ilyenek begyűjtésére a MING-forrás ***util*** alkönyvtárában található `makefdb` használható, először ezt sem árt lefordítanunk. Tekintsük át nagy léptékekben, hogyan is épül fel a MING-birodalom! A legtöbb PHP-kiegészítéssel ellentétben itt nem ömlesztett függvénykönyvtárat kapunk a nyakunkba, hanem 13 osztályt. Ezek mindegyike egy témát ölel fel, és a hozzá tartozó eljárásokat, függvényeket, tulajdonságokat hordozza magában. Lássuk, miből fogunk csipegetni!

`SWFShape()`	Ezzel hozhatjuk létre és rajzolhatjuk meg a különböző, a moziban megjelenő formákat.
`SWFBitmap()`	A moziba bevihető objektumokat JPEG-képekből hozhatjuk létre.
`SWFText()`	A szöveges elemek létrehozására való osztály.
`SWFTextField()`	Szöveges űrlapelemek létrehozásához.
`SWFSprite()`	Önálló animációs almozik hozhatók létre vele, amelyek saját időskálával rendelkeznek.
`SWFButton()`	Nyomógombok létrehozására szolgál.
`SWFFont()`	Különféle betűtípusokat tölthetünk be vele, valamint a szövegek megjelenítéséhez szükséges.
`SWFGradient()`	Színátmenetek létrehozására, továbbá a formák kifestésénél használható.
`SWFFill()`	Már meglévő kifestőobjektumok forgatására, mozgatására és átméretezésére szolgál.
`SWFDisplayItem()`	A moziban létrehozott, behúzott objektumokat itt tudjuk pörgetni, forgatni és nagyítgatni.
`SWFMorph()`	Alakjukat változtató látványelemek létrehozását teszi lehetővé.
`SWFAction()`	A Flash saját nyelvén írhatunk ActionScripteket.
`SWFMovie()`	A mozi maga: elemeket adhatunk hozzá, menthetjük vagy a kimenetre küldhetjük a tartalmát.

Az `SWFMovie()` osztályt mindig használni fogjuk, mert mind a mozi születésekor, mind a véglegesítésekor jelen van. Egy moziobjektumot a következő módon hozunk létre:

```
$mozi = new SWFMovie();
```

Innentől létezik is $mozi néven az objektumunk, ebbe szórjuk bele a mozgatnivaló elemeket. Ha létrehoztuk, nem árt néhány vele kapcsolatos dolgot beállítani:

```
$mozi->SetRate(20);
$mozi->SetDimension(4000,4000);
$mozi->SetBackground(0xff, 0xaa, 0x66);
```

A `SetRate()` által tudjuk megadni, hogy egy másodpercben hány képkockát játszunk le, ez lesz a teljes mozira érvényes beállítás. Megjegyzendő, hogy csak amolyan kívánatos értékről van szó, hiszen egy leterhelt gépen a képkockák megrajzolása a rendelkezésre álló időnél többet vehet igénybe. Ilyenkor egyszerű lassulásról van szó: a lejátszó nem hagy ki kockákat, csupán lassabban játssza le őket. Más eset áll fenn akkor, ha folyamatos MP3 zenei aláfestés is tartozik a mozinkhoz, mert

1. lista A gorillamozi.php – immár teljes pompájában

```
<?php

$mozi = new SWFMovie();

$mozi->SetRate(20.0);
$mozi->SetDimension(4000,4000);
$mozi->SetBackground(0xff, 0xaa, 0x66);

// A kimenetre k ldØs elıtt tudatnunk kell
// a b ngØszıvel az adathalmaz MIME-t pusÆt.

header('Content-type:
       ➥application/x-shockwave-flash');

$mozi->Output();

?>
```

2. lista A gorilla.html – az első mozinkat beágyazó HTML-oldal

```
<html>
<head>
    <title>GorillÆk - rendezıi
      ➥vÆltozat</title>
</head>
<body bgcolor=#ffffff>

    <embed src=gorillamozi.php width=200
      ➥height=200></embed>

</body>
</html>
```

ilyenkor a lejátszónak biztosítania kell, hogy a hanglejátszás lehetőleg folyamatos legyen. Ezt a gondot képkockakihagyással küszöböli ki.

A befoglaló méreteket, azaz megjelenő munkaterületünk méretét a `SetDimension()` által adhatjuk meg. Mint korábban már említettem, itt nem képpontokban, hanem twipekben kell gondolkodnunk, azaz a fenti példa egy 200x200 képpont méretű Flash-objektumot hoz létre.

A méretek közül először a szélességet, másodjára a magasságot kell megadni. A mozinak kell háttérszín is. Ennek beállításához használatos a `SetBackground()`. A háttérszínt az RGB- (vörös, zöld, kék) összetevők keverésével tudjuk létrehozni. A színeket 3x8 biten képezhetjük le, vagyis az egyes értékek értéktartománya 0-tól 255-ig terjed, és csakis egész számokban adhatók meg. A példában éltem a PHP nyelv adta lehetőséggel, és az értékeket tizenhatos számrendszerben ábrázoltam (hiába no, én már csak hexadecimálisokban látom a színeket).

Örvendezzünk, ugyanis elkészítettük életünk első egész estés animációs filmjét! A címe „Gorillák a narancsos ködben" lehetne, tekintve az események letisztult egyszerűségét. Egy apróság hiányzik még: mozinkat láthatóvá is kellene tenni a közönség (legalábbis a böngészőnk) számára. Ehhez az

3. lista A haromszog.php már valami, de még nem mozog…

```
<?php

// A befoglal  mozi.

$mozi = new SWFMovie();
$mozi->SetDimension(6000,6000);
$mozi->SetBackground(0xff, 0xff, 0xff);

// a hÆromsz g megrajzolÆsa-

$valaki = new SWFShape();
$valaki->setLine(5, 0xce, 0xce, 0xce);
$valaki->setRightFill($valaki->addFill(0xe0,
      ➥0xe4, 0xec, 50));
$valaki->movePenTo(0,1600);
$valaki->drawLineTo(-1000,-400);
$valaki->drawLineTo(1000,-400);
$valaki->drawLineTo(0,1600);

// tegy k a moziba, Øs k ldj k a helyØre

$haromszog = $mozi->add($valaki);
$haromszog->move(3000,3000);
$mozi->nextFrame();

// kØsz van, mehet a vilÆg elØ...

header('Content-type:
      ➥application/x-shockwave-flash');

$mozi->Output();

?>
```

`SWFMovie()` egy újabb fontos eljárását kell alkalmaznunk. Lássunk erre is példát!

```
$mozi->Output();
```

Munkánk eredményét ez a gyakorlatlanok számára így még eléggé emészthetetlen formában tálalja: krixkrax karakterek halmazaként. Mielőtt még e furcsa betűkben látni kezdenénk a jeleneteket, tegyük fogyaszthatóvá az adatokat. Ehhez több dologra is szükség lesz: először is tudatnunk kell a böngészővel, hogy a küldött adatfolyam Flash-mozit közvetít. Második lépésként pedig létre kell hoznunk egy, a művünket beágyazó HTML-oldalt.
Tapasztalat, hogy a Flash-mozik fejlesztése során böngészőnk gyorsítótárát érdemes kikapcsolni, ellenkező esetben hajlamos makacsul ragaszkodni egy korábbi állapothoz. Így pedig meglehetősen nehézkes a kódolás folyamán ellenőrizni, hogy az történik-e, amit valóban szeretnénk.

Lejátszó hiányában…

Előfordulhat, hogy jelenlegi böngészőnk nem alkalmas Flash-fájlok lejátszására, ilyenkor hamar átirányít a Macromedia letöltési oldalára. Amennyiben ez önműködően nem történne meg, magunknak kell ellátogatnunk oda (*3. kép*).

Forgatás előtt – a szereplők

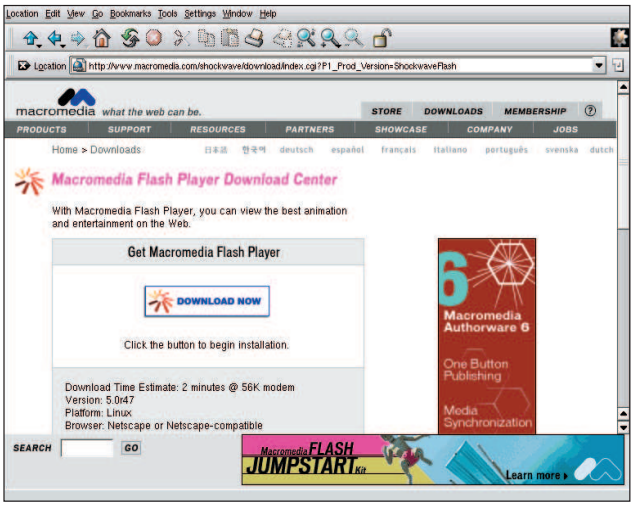
Most már feltehetően az összes szükséges eszközzel rendelkezünk a PHP-s Flash-fejlesztéshez. Beállítottuk a kiszolgálót és a böngészőnket is. Kíséreljünk meg összeállítani valami komolyabbat. Eddig csak az `SWFMovie()` osztály biztosította eszközkészlettel ügyeskedtünk. Itt az ideje megismerni egy másik, szintén elég alapvető osztályt, az `SWFShape()`-et.
A szemléltetést egy egyszerű háromszög alakjának megrajzolásával kezdem. Hozzuk létre a formát képviselő objektumot:

```
$valaki = new SWFShape();
```

Innentől $valaki néven létezik az objektum. Miután megrajzoltuk, az animációnkba számtalan példányban beilleszthetjük (így lehet fenyőfákból erdőt növeszteni). Természetesen minden ilyen egyed külön torzítható, forgatható, mozgatható. A forma megrajzolása előtt szükség lesz pár alapadat megadására, ilyen például a vonalvastagság és -szín. Tudatnunk kell azt is, ki akarjuk-e a rajzunkat festeni, és amennyiben igen, vajon mivel.

```
$valaki->setLine(5, 0xce, 0xce, 0xce);
$valaki->setRightFill($valaki->addFill(0xe0,
      ➥0xe4, 0xec, 50));
```

Az első sorral a rajzot megjelenítő vonal stílusát határozhatjuk meg. Először a vonal vastagságát, majd pedig a színét kell megszabnunk. A vastagság természetesen twipben értendő, ezért az 5-ös vastagság elég vékonynak számít. A vonal színének megadása a korábban megismerthez hasonló módon zajlik.



3. kép ➲ http://www.macromedia.com/shockwave/download

A vonalrajzolás beállítása után következik a kifestés meghatározása. Mint látható, egymásba ágyazott eljárásokról van szó. A belsővel, azaz az `SWFShape()` osztály `addFill()` függvényével különféle kifestési stílusokat hozhatunk létre. A jelenlegi egy puritán egyszínű festőstílus. Három adatot kell kötelezően megadnunk, amelyeket akár egy negyedikkel is kiegészíthetünk. A példára tekintve az első három talán kézenfekvő is, ezek a szokásos színösszetevők. A negyedik megadható adat pedig egy 0-tól 100-ig terjedő, áttetszőséget meghatározó érték. Ennek a `$valaki->addFill()` kifejezésnek a visszatérő értékét (egy azonosítót) kapja meg a `$valaki->setRightFill()`.
A `setRightFill()` parancsnak létezik egy társa, a

4. lista A mar_valami.php – mozgással ellátott mozi

```
<?php

// vØletlenszÆmgyÆrt  f ggvØny behœzÆsa

include 'random.inc';

// beÆll tÆsok egy helyen

define('NUM_TRIANGLES',20);
define('NUM_FRAMES',100);
define('FRAME_RATE',15);

// kezdı lØpØsek: mozi lØtrehozÆsa

$mozi = new SWFMovie();
$mozi->SetRate(FRAME_RATE);
$mozi->SetDimension(6000,6000);
$mozi->SetBackground(0xff, 0xff, 0xff);

// a mozi sorÆn rengetegszer felhasznÆlt
// hÆromsz g alakjÆnak megrajzolÆsa

$valaki = new SWFShape();
$valaki->setLine(5, 0xce, 0xce, 0xce);
$valaki->setRightFill($valaki->addFill(0xe0,
➥0xe4, 0xec, 50));
$valaki->movePenTo(0,1600);
$valaki->drawLineTo(-1000,-400);
$valaki->drawLineTo(1000,-400);
$valaki->drawLineTo(0,1600);

// A k vÆnt mennyisØgß hÆromsz g elhelyezØse,
// vØletlenszerß torz tÆsa, szØtsz rÆsa.
// Mindegyikhez k l nb zı,  sszevissza
// lØtrehozott forgÆsi sebessØg adÆsa.

for ($i=0; $i<NUM_TRIANGLES; $i++) {
    $hszog[$i] = $mozi->add($valaki);
    $hszog[$i]->move(2000+randomint(2000),
            ➥2000+randomint(2000));
    $hszog[$i]->scale(randomint(15)/10,
            ➥randomint(30)/10);
    $hszog[$i]->rotate(randomint(360));
    $hszogforg[$i] = randomint(15)-7.5;
}

// a k vÆnt mennyisØgß kØpkocka legyÆrtÆsa
// szØp sorban. Itt mÆr csak forgatni kell. :)

for ($j=0; $j<NUM_FRAMES; $j++) {
    for ($i=0; $i<NUM_TRIANGLES; $i++) {
      $hszog[$i]->rotate($hszogforg[$i]);
    }
    $mozi->nextFrame();
}

// kØsz van, mehet a vilÆg elØ...

header('Content-type:
            ➥application/x-shockwave-flash');

$mozi->Output();

?>
```

`setLeftFill()`. Ha formánk pontjait sorrendben úgy adjuk meg, hogy a körbejárási sorrendjük az óramutató járásával megegyező irányú, akkor van szükség az elsőre. Fordított irányban haladva a „befelé” balra esik. Érdemes erre figyelmet fordítani, mert a lejátszó esetleg bedobhatja miatta a törülközőt. Érdekes adat, hogy ezeket jelenleg valamiért az `SWFMorph()` osztályon belül felcserélve kell használnunk. Apró következetlenség, majd „kinövi” a program. No igen, a PHP/MING-leírás minden egyes oldalon kihangsúlyozza, hogy ez a modul igencsak kísérleti állapotban található, az egyes elemek bármikor gyökeresen megváltozhatnak. Így jelenleg senki sem garantálja, hogy a mostani MING-objektumaink a következő kiadással is ugyanúgy fognak működni, tehát bánjunk velük óvatosan. Innentől kezdődik a forma megrajzolása, amihez vonalakat és íveket kell sorra megadnunk. Emellett rajzeszközünket vonal rajzolása nélkül is arrébb tudjuk pakolni, amire mindjárt az elején szükségünk is lesz, hiszen a tárgyunkat nem a 0,0 pontból kezdjük rajzolni. Irány a kiindulópont!

```
$valaki->movePenTo(0,1600);
```

Ezzel az eljárással „ceruzánkat” vonal rajzolása nélkül mozgatni tudjuk. A koordináták a szokásos módon X,Y sorrendben adandók meg. Vízszintes irányban egyértelmű a helyzet, hiszen ritka eset, ha valahol nem balról jobbra nő a koordinátaérték; függőleges irányban pedig számításba kell vennünk a Besenyő Pista bácsi-féle biorobotelmélet(et): „Egyet kell kérdezni: hogy mekkora, és hogy leerű-fee vagy feerű le?” Nos, kedves Boborján, a második. Tehát Y irányban a koordinátaérték lefelé növekszik, ami pont a körbejárási irány meghatározásánál a legfontosabb (csak megemlítem, hogy a méreteket itt is twipsben kell érteni). A rendszer legalább ebben végig következetes. Lehetőségünk nyílik közvetlen vagy viszonylagos helyzetmegadásra is. Példánkban egész idő alatt a közvetlen módszert választottam, amire az eljárások nevének végén található „To” szócska utal. Viszonylagos elmozdulást egy `$valaki->movePen()`-nel lehetett volna megadni. Innentől kezdve a háromszöget az óramutató járásával megegyező irányban rajzolom körbe:

```
$valaki->drawLineTo(-1000,-400);
$valaki->drawLineTo(1000,-400);
$valaki->drawLineTo(0,1600);
```

Tárgyunk megrajzolásával ezennel végeztünk is. Háromszögünket helyezzük a munkatérbe, hogy láthatóvá váljon a kotyvasztásunk végeredménye. A befoglaló HTML-oldal elkészítését a nyájas olvasó/alkotó képzelőerejére bízom. A programlistában három ismeretlen sor található.