ESEMÉNYKEZELÉS FLASH MX

Cikkünkben a Flash MX eseménykezelő-modelljével ismerkedünk meg részletesebben. Nagy fejlődést mutat ez a Flash 5 lehetőségeihez képest, szépen felépített és jól áttekinthető, ugyanakkor moduláris és könnyen variálható scripteket írhatunk. Ezekkel egyszerűbben és sokrétűbben hozhatunk létre animációkat, ötletes és hasznos interaktív elemeket. Az alapvető meghatározások mellett néhány trükköt is bemutatunk, amelyek elsőre nem feltétlenül egyértelműek, de a későbbiek folyamán igen hasznosnak bizonyulhatnak.

Alapvető ismeretek

Elsőként az **esemény (event)** fogalmát kell tisztáznunk. Eseménynek nevezzük az a dolgot, ami a mozi futása közben történik. Ez lehet **a felhasználó által generált esemény**, mint pl. egy egérkattintás vagy egy billentyű lenyomása, ill. lehet **rendszeresemény**, amire jó példa az, amikor a mozi egy képkockáról a következőre lép. A mozinkban elhelyezett objektumok figyelhetik ezeket az eseményeket, és reagálhatnak is rájuk. Ezek az **eseményfigyelők (listener)**. Nem mindenféle objektum figyel minden eseményre, valamelyikhez nekünk kell hozzárendelni azt. Ilyen pl. a moziklip objektum: figyel az egéreseményekre, de a billentyűzetfigyelésre már utólag kell rábírnunk.

Az **eseménykezelő** valójában egy függvény, ami akkor hajtódik végre, amikor az adott esemény bekövetkezik. Eseménykezelő pl. az onEnterFrame vagy az onMouseMove, ez azonban nem csinál semmit addig, amíg nem definiálunk egy függvényt, amit végrehajthat (addig undefined az értéke). Itt fontos megjegyeznünk, hogy valójában az enterFrame esemény következik be, és az onEnterFrame, ill. az ahhoz kapcsolt függvény az, amivel megmondjuk, hogy mi történjen. Azt, hogy enterFrame, Flash MX-ben nem írjuk a scriptbe soha, csak azért használjuk így, hogy tudjunk beszélni róla.

Flash 5 stílusban, ha egy moziklippet mozgatni akarunk, a legkézenfekvőbb megoldásnak az bizonyul, ha kihúzunk egy példányt a színpadra, és arra ráírjuk pl. a következőt:

```
onClipEvent(enterFrame) {
_x+=Math.round(Math.random()*5);
}
```

A fenti kód az enterFrame esemény érzékelésének köszönhetően másodpercenként annyiszor lefut, amennyi az fps-nél beállított érték. Ez alapértelmezés szerint 12, de a 24-25 az ideális, mert már a folyamatosság látszatát kelti, de még nem gazdálkodik túl pazarlóan egy átlagos gép erőforrásaival (most állítsuk 25-re). Tehát az eredmény az, hogy a moziklipünk 0 és 5 közötti véletlenszerű egész egységekkel jobbra mozdul, egyfajta egyenetlen sebességű, egyirányú mozgást produkálva. Tegyük fel, hogy 15 golyóbist szeretnénk ily módon mozgatni. Így mindegyikre rá kell ezt a kódot másolni. Ez egyrészt felesleges időpocsékolás, másrészt ha egy apró kiegészítést szeretnénk a viselkedésükben (pl. az _y tulajdonságot is növelni akarjuk), mindenhol át kell írni a sorokat, ami az időt és az átláthatóságot tekintve sem optimális megoldás. Ehelyett egy más megközelítést alkalmazunk. Hozzunk létre egy új mozit, ebben egy script és egy golyok nevű réteget a fő idősíkon (_root), nevezzük el a golyobis moziklip egy példányát golyobis_mc-nek a golyok rétegen, és írjuk a script réteg egyetlen frame-jére a következőket:

```
function mozgas () {
this._x+= Math.round(Math.random()*5);
}

stop();
golyobis_mc.onEnterFrame=mozgas;
```

Az első, amire rá kell mutatnunk, a this szerepe. Ha egy változót, tulajdonságot, szimbólum példánynevet elérési út nélkül adunk meg (pl. _x vagy golyobis_mc), az az adott idősíkra fog vonatkozni, esetünkben ez a _root-ot jelentené. A this használatával viszont a hatókört a meghívó objektumra, itt éppen a golyobis_mc-re helyezzük át. Direkt meghívással (mozgas() v. _root.mozgas()) a teljes fő idősík mozdulna, de így, hogy a golyobis_mc onEnterFrame eseménykezelőjéhez rendelve hívódik meg, csak a golyobis_mc mozdul.

A második dolog, hogy attól még, hogy a stop()-pal leállítottuk a fő idősíkot, és valójában nem ugrunk a következő frame-re, (főleg, ha csak egy frame-ből áll a mozi), az onEnterFrame-hez csatolt függvény még továbbra is folyamatosan lefut a fps-nek megfelelő gyakorisággal.
A harmadik, és egyben legfontosabb említendő momentum az eseménykezelő metódus definiálása. Amikor egy névvel ellátott függvényt hozunk létre, maga a név mutat a memóriában létrehozott utasításhalmazra. Ezt a hivatkozást egy másik változónak - pl. masikMozgas=mozgas -, vagy akár a fent látott módon egy eseménykezelőnek is átadhatjuk. Vigyáznunk kell viszont arra, hogy ekkor zárójelek nélkül kell a függvénynevet használnunk, mert a zárójelpár mindenképpen meghívást jelent.

```
// az onEnterFrame a mozgas fv. visszatérési
értékét veszi fel - helytelen módszer
golyobis_mc.onEnterFrame=mozgas();

// az onEnterFrame magára a mozgas fv.-re
mutat
golyobis_mc.onEnterFrame=mozgas;
```

Tegyünk egy próbát! Az alábbi esetben a mozgas függvény mindössze egyszer hívódik meg, a visszatérési értéke a golyóbis _x tulajdonsága, amit a trace paranccsal kiíratunk az Output ablakba, mint a golyobis_mc.onEnterFrame értékét. Ennek pedig nem sok értelme van, azon felül, hogy nem is működik a mozgás, amit látni szeretnénk. Ha viszont az utolsó előtti sorból a zárójelpárt levesszük, rendesen működik az animáció, a trace pedig immár helyesen type Function-t ír ki, mivel az onEnterFrame-hez függvényt rendeltünk.

```
function mozgas () {
this._x+= Math.round(Math.random()*5);
return this._x;
}

stop();
golyobis_mc.onEnterFrame=mozgas();
trace(golyobis_mc.onEnterFrame);
```

Ezzel azonban még nem vagyunk készen, ugyanis csak egy golyóbist mozgattunk. 15 példányt a színpadra kihúzva, és azokat a golyobis0_mc, golyobis1_mc stb. sorozatként golyobis14_mc-ig elnevezve a következő egyszerű, rövid ciklussal az összes moziklipet irányíthatjuk.

```
function mozgas () {
this._x+= Math.round(Math.random()*5);
}

stop();
for (i=0; i<15; i++) {
_root["golyobis"+i+"_mc"].onEnterFrame=mozgas;
}
```

Itt azt használjuk ki, hogy különböző objektumokat, szimbólumok példányait, azok tulajdonságait, metódusait el lehet érni a tömbjelölés segítségével is, ahol a nevet sztringként alkalmazzuk. _root["golyobis"+6+"_mc"] ugyanazt jelenti, mint _root.golyobis6_mc.
Ezen kívül még létezik egy másik lehetőség is, ami adott esetben még hasznosabb lehet, mivel a for ciklusra nincs szükség, és még a sorozatos elnevezéssel sem kell bíbelődni, bár a nevek használatának nélkülözése a későbbi, ill. futásidejű változtatásokat nem könnyíti meg. Töröljünk le minden moziklip-példányt a Stage-ről, a _root-on pedig csak a mozgas függvénydeklarációt és a stop()-ot hagyjuk. Magában a golyobis Library-elemben a scriptnek hozzunk létre egy - a szokásos módon script nevű - külön réteget, ennek egyetlen frame-jére pedig a következőt írjuk:

```
stop();
onEnterFrame=_root.mozgas;
```

Ettől kezdve az összes színpadra kihúzott példány lassan áthalad a képernyőn, mindegyik a saját véletlenszerű sebességváltozásainak megfelelő módon.
Ha azt szeretnénk, hogy mondjuk az x tengelyen 300 pixelnél megálljanak a golyóbisok, lehet a mozgas függvénybe egy olyan feltételt tenni, ami ha 300-nál nagyobb lenne az _x, akkor az maradjon 300-on. Ekkor azonban az onEnterFrame még azután is fut, hogy már nincs rá szükség, mert a golyó áll. Ha a mozi futása során még sok onEnterFrame-et használunk, és egyiket sem állítjuk le, miután elvégezte a dolgát, előbb-utóbb irgalmatlanul belassul a lejátszás. A feleslegesen futó metódust érdemes ilyenkor lekapcsolni az eseménykezelőről. Tegyük a golyóbisokat 300-nál kisebb x-pozícióba, és írjuk át a mozgas függvényt a következők szerint:

```
function mozgas () {
this._x+= Math.round(Math.random()*5);
if (this._x>=300) {
    this._x=300;
    this.onEnterFrame=undefined;
}
}
```

A this.onEnterFrame=undefined ugyanazt eredményezi, mint a delete this.onEnterFrame, jóformán ízlés kérdése, hogy melyiket használjuk. Ezen utasítások nem szüntetik meg a mozgas függvényt, csupán innentől kezdve az nincs hozzárendelve az eseménykezelőhöz. Amennyiben a későbbiek során újra mozgatni akarjuk valamelyik golyóbisunkat, elég újra a golyobis1_mc.onEnterFrame=mozgas formát használni, miután az egyik moziklipünket golyobis1_mc-nek neveztük el. Máris halad tovább a moziklip, amennyiben valahogy visszatettük előtte 300-nál kisebb x-pozícióba. Erre megoldásként kínálkozik a következő példa, mely rögvest az eseménykezelők hozzárendelésének egy másik módját is

bemutatja. A _root-on lévő script végére tegyük a következőket:

```
onMouseDown=function() {
    golyobis1_mc._x=150;
    golyobis1_mc.onEnterFrame=mozgas;
};
```

Előbb visszahelyeztük a moziklipet, majd mozgásra bírtuk, és mindezek megtörténtét egy egérkattintáshoz kötöttük. Ami újszerű, az az, hogy itt közvetlenül az eseménykezelőhöz (onMouseDown) rendeltünk egy névtelen függvényt. Ezt a rövidebb módszert akkor érdemes használni, amikor tudjuk, hogy a függvényt csak egyszer akarjuk az adott kezelőhöz kapcsolni. Ha ugyanis egyszer - pl. az undefined alkalmazásával - leválaszottuk az eseménykezelőről, később már - név hiányában - nem tudunk rá sehogy sem hivatkozni.

Ezt a formát kell akkor is követnünk, amikor az eseménykezelőhöz tartozó függvénynek paramétereket kívánunk átadni. Az eddigi példánál maradva, immár paraméterekkel szeretnénk megmondani, hogy meddig menjen a golyóbis. Az onMouseDown-t és a hozzá tartozó névtelen függvényt töröljük ki a _root-ról, a mozgas függvényt pedig írjuk át az alábbiaknak megfelelően:

```
function mozgas (klip, hatar) {
klip._x+= Math.round(Math.random()*5);
if (klip._x>=hatar) {
    klip._x=hatar;
    klip.onEnterFrame=undefined;
}
```

A golyobis Library-elemben található onEnterFrame-et a következőképpen fogalmazzuk át:

```
onEnterFrame = function () {
    _root.mozgas(this, 200);
};
```

A mozgas függvényt kibővítettük két paraméterrel, a klipp az adott moziklippre fog vonatkozni, a hatar pedig azt hiszem, nem szorul magyarázatra. Az onEnterFrame-hez közvetlenül nem használhatjuk az onEnterFrame=_root.mozgas(this, 200) formát, mivel megbeszéltük, hogy a zárójelek miatt a mozgas rögtön meghívásra kerülne és csak egyszer, helytelen működést eredményezve. Így tehát egy névtelen függvénybe kell csomagolnunk a mozgas meghívását. Mivel már nem közvetlenül az eseménykezelőhöz kapcsoltuk azt, hanem direkt módon meghívtuk, ha a mozgas-ban nem lenne klip paraméter, és a this-eket meghagytuk volna, - a cikk elején bemutatottaknak megfelelően - a teljes fő idősík mozdulna el, mivel a this ez esetben arra mutatna. Így szükséges bevezetnünk egy paramétert, hogy megmondjuk, mire vonatkozzon a mozdítás. A golyobis-ban a _root.mozgas(this, 200)-ban szereplő this mindegyik golyóbis-példány saját idősíkját jelenti, így megoldottuk a feladatot.

Ugyanúgy, ahogy először hozzákapcsoltunk valamilyen függvényt egy eseménykezelőhöz, majd leválaszottuk róla, természetesen nem csak ugyanazt rendelhetjük hozzá újra, hanem akármilyen másik függvényt is. Ki is próbálhatjuk, például írjunk egy újabb függvényt a mozgas elé, ami a mozgás végeztével triplájára növeli a golyóbisunkat:

```
function noveles () {
    this._xscale=this._yscale+=10;
if (this._xscale>=300) {
        this._xscale=this._yscale=300;
this.onEnterFrame=undefined;
}
}
```

A mozgas függvényben csak a következőt kell változtatni: klip.onEnterFrame=noveles.

További trükkök, fontos apróságok

Az eseménykezelők közül érdemes még kiemelni az onMouseMove-ot, mivel segítségével könnyedén készíthetünk saját egérmutatót, és egy fontos elvi dologra világítunk rá. Egy teljesen új, egy frame-ből álló moziban hozzunk létre egy klip és egy script réteget. A klip rétegre egy tetszőleges, egeret helyettesítő moziklipet tegyünk, aminek legyen eger_mc a neve, a script rétegre pedig írjuk a következőket:

```
stop();
Mouse.hide();
eger_mc.onMouseMove=function() {
    this._x=_xmouse;
this._y=_ymouse;
};
```

A Mouse.hide()-dal eltüntetjük az eredeti egeret, és ezzel készen is volnánk, de van egy kis bökkenő. Itt ugyanis csak onnantól kezdi az egérmutató helyét követni a moziklipünk, miután megmozdítjuk egerünket. Próbáljuk ki, a Flash MX-ben Ctrl+Enter-rel teszteljük a mozit, de ne mozdítsuk meg az egeret, csak később. Addig a moziklip a színpadon van a kezdeti pozícióban. Kézenfekvőnek látszik a megoldás, hogy meg kell hívni az onMouseMove-hoz tartozó függvényt. Csakhogy az egy névtelen függvény, nem tudunk rá hivatkozni a megszokott függvényhívást alkalmazva. Beírni mégegyszer az onMouseMove-ban szereplő utasításokat, hogy ezek egyszer a mozi kezdetén mindenképp lefussanak, nem valami professzionális megoldás. Szerencsére az eseménykezelőhöz kapcsolt függvényt meg tudjuk hívni a következőképpen is: eger_mc.onMouseMove(). Írjuk is be a kódunk végére. Ez természetesen itt nem takarít meg sok helyet és munkát, de egy bonyolult, hosszú eseménykezelő metódus esetében ez a módszer nagyon hasznos lehet. Gondoljunk egy sok gombból álló menüre, mindegyik onRelease-éhez különböző függvények, utasítások tartoznak: ha valahonnan szeretnénk az egyik menüpontra történő kattintást utánozni tényleges kattintás nélkül, elég csupán valami hasonlót csinálni:

```
tesztKlip_mc.onRollOver=function() {
trace("mintha a menupont3_mc-re kattintottunk
volna");
    menupont3_mc.onRelease();
}
```

Ily módon paramétereket is átadhatunk az eseménykezelő metódusnak, ritkán van rá szükség, de jól jöhet. Hogy mikor érdemes alkalmazni, azt már az Olvasóra bízzuk:

```
proba_mc.onRelease=function(par1, par2) {
    // fuggvenytorzs
}
proba_mc.onRelease("valami", pl_egy_valtozo);
```

Egy pár mondat erejéig visszatérve az egérkövető moziklipphez, a függvénytörzs utolsó soraként érdemes beszúrni az updateAfterEvent() utasítást. Ez annyiszor frissíti a képernyőt, ahányszor csak az adott esemény bekövetkezik. onMouseMove-nál használva gyakorlatilag nagyon szép folyamatos mozgást produkálhatunk, akár 1 fps mellett is. Állítsuk ilyen alacsonyra, próbáljuk ki updateAfterEvent()-tel, aztán nélküle is - észrevehető a különbség :)

Végezetül nem szabad elmennünk a függvények deklarálásának két módja mellett:

```
function probaFuggveny() {
}

probaFuggveny=function() {
}
```

Meghíváskor ugyanúgy a probaFuggveny()-t használjuk, de az eseménykezelőkhöz rendelésnél vigyázni kell a sorrendre. Ha az első formát alkalmazzuk, mindegy, hogy a kódban az eseménykezelőhöz a függvénydeklaráció előtt vagy után rendeljük a függvényt, mert a Flash legelőször a függvényeket nézi meg. A második formánál azonban annak ellenére, hogy itt is függvényről van szó (még ha névtelenről is), egy értékadás jobb oldalán találjuk a function-t, így ezt már a szekvenciális megközelítésnek köszönhetően mindenképpen az eseménykezelőhöz kapcsolás elé kell írni, ugyanis addig nem létezik a függvény, amíg a Flash „fentről lefelé” el nem jut odáig az értelmezésben.

Összefoglalás

Áttekintettük tehát az eseménykezelés csínját-bínját, az alapokat, illetve az azokhoz kapcsolódó, nem túl kézenfekvő, de lényeges ismereteket. Mindezeket egyszerű példákon keresztül végigkövetve az Olvasó már bátran építkezhet. E módszerek ismeretében és kreatív alkalmazásával megvan a lehetősége olyan Flash-mozik készítésére, melyek a programozó és a felhasználó számára egyaránt az elégedettség élményét nyújtják.

Molnár Ákos - flessmx@yahoo.com