

DESIGNERS TOOLBOX

Marc Sauter

Flash Tutorium: Action Script Einführung

// Bei Fragen etc.: marc@psychon.net

Folgende Kenntnisse werden vermittelt:

I

**Drag-Optionen für beliebige Objekte innerhalb eines Pseudo-3D-Gitters
Effekt: Anpassung der Objektskalierung und Farbintensität für Tiefenvermittlung**

II

Minimale Animationen der Objekte sowie Grundkenntnisse für Objektbewegung

III

Soundsteuerung

I

Drag-Optionen für beliebige Objekte innerhalb eines Pseudo-3D-Gitters

Effekt: Anpassung der Objektskalierung und Farbintensität für Tiefenvermittlung

In diesem Beispiel erstellen wir einen Movieclip, der in einem 3-D-Raum umhergezogen werden kann. Was dort herumgezogen wird ist egal; zur Veranschaulichung des Prinzips verwenden wir einfache Quadrate.

Zuerst erstellen wir so etwas wie einen perspektivischen Boden, dessen oberes Maximum den Horizont ausmachen wird. Dann noch ein Objekt (in diesem Beispiel ein Quadrat) – im besten Falle auf eine neue Ebene. Dieser wird in ein Movieclip konvertiert (F8), mit Namen und nicht vergessen den Movieclip auf der Bühne ebenfalls zu benennen (=Instanzname bei „Properties“ hier: graf).

Im nächsten Schritt machen wir dieses Ding ziehbar. Mit einem Doppelclick gelangen wir in den Movieclip und generieren auf einer neuen Ebene eine Schaltfläche (F8•Button). Die Schaltfläche markiert lassen und im Actionfenster (F2 oder F9) fügen wir folgenden Code hinzu:

```
on (press){
    startDrag (this, false, _root.left, _root.top, _root.right,
    _root.bottom);
}

on (release){
    stopDrag();
}
```

Das heißt soviel wie: Solange die Maus gedrückt über der Schaltfläche bleibt, wird das Objekt (=this) mitgezogen (startDrag). Das zweite Argument in der Klammer teilt Flash mit, ob der Mauszeiger in der Mitte der Mitte des Movieclips zentriert werden soll (für dieses mal nicht, deswegen false). Die nächsten vier Argumente legen das Begrenzungsrechteck für die DragAktion fest. Wir haben hier variablen angegeben, deren Wert wir im _root-Verzeichniss angeben werden. (left, top, right, bottom).

Wenn die Maus wieder losgelassen wird (on(release)), endet die Drag-Aktion.

Kommen wir nun zu den Variablen zur Hauptzeitleiste:

Den perspektivischen Boden in einen Movieclip umwandeln (F8) und einen Namen geben (auch auf der Bühne hier: „grid“). Nun erstellen wir eine neue Ebene und nennen sie Script und öffnen das Action-Fenster für den Frame.

Folgenden Code eingeben:

```
grenzObj = grid.getBounds(_root);
left = grenzObj.xMin;
right = grenzObj.xMax;
top = grenzObj.yMin;
bottom = grenzObj.yMax;
```

getBounds() gibt ein Objekt zurück, das die vier Eckkoordinaten Koordinaten des erwähnten Movieclips beinhaltet – in diesem Fall „grid“. Der Bezug _root in Klammer bezieht sich auf den Zielkoordinatenraum (also auf welche Zeitleiste soll sich das ganze beziehen).

Um diese Koordinaten zu verwenden folgendes hinzufügen:

```
left = grenzObj.xMin;
right = grenzObj.xMax;
top = grenzObj.yMin;
bottom = grenzObj.yMax;
```

Damit sind dann die Eckvariablenwerte der `startDrag`-Aktion des Buttons verfügbar. Jetzt erst mal testen. Das nette mit den Variablen bezüglich des `grid-Movieclip` ist die Tatsache, dass der Dragbereich an den `grid-Movieclip` sich automatisch anpasst, sollte dieser skaliert werden oder sonstiges.

Jetzt muss man nur beachten, dass die Koordinaten des Quadrats auf das Zentrum bezogen ist (Rumschieberei ...)

Soweit so gut. Was jetzt noch fehlt ist die Skalierung während des Drag-Vorganges, d. h je weiter oben, desto kleiner sollte das Quadrat werden, um diesen Pseudo-3-D-Effekt zu generieren.

Auf der Ebene „Script“ folgendes hinzufügen:

```
function setScale (ypos) {
    var newScale = (100/(maxRef-zeroRef))*(ypos-zeroRef);
    return newScale;
}
```

Diese Funktion skaliert den Akteur (Quadrat) abhängig von seiner y-Position. Die Bildfläche wird in Prozente aufgeteilt, indem wir einen `maxRef` und einen `zeroRef` – Punkt bestimmen. Die Fläche dazwischen wird zur Festlegung des Prozentwertes verwendet.

Über diese Funktion fügen wir folgende Zeilen ein:

```
zeroRef = top;
maxRef = bottom;
```

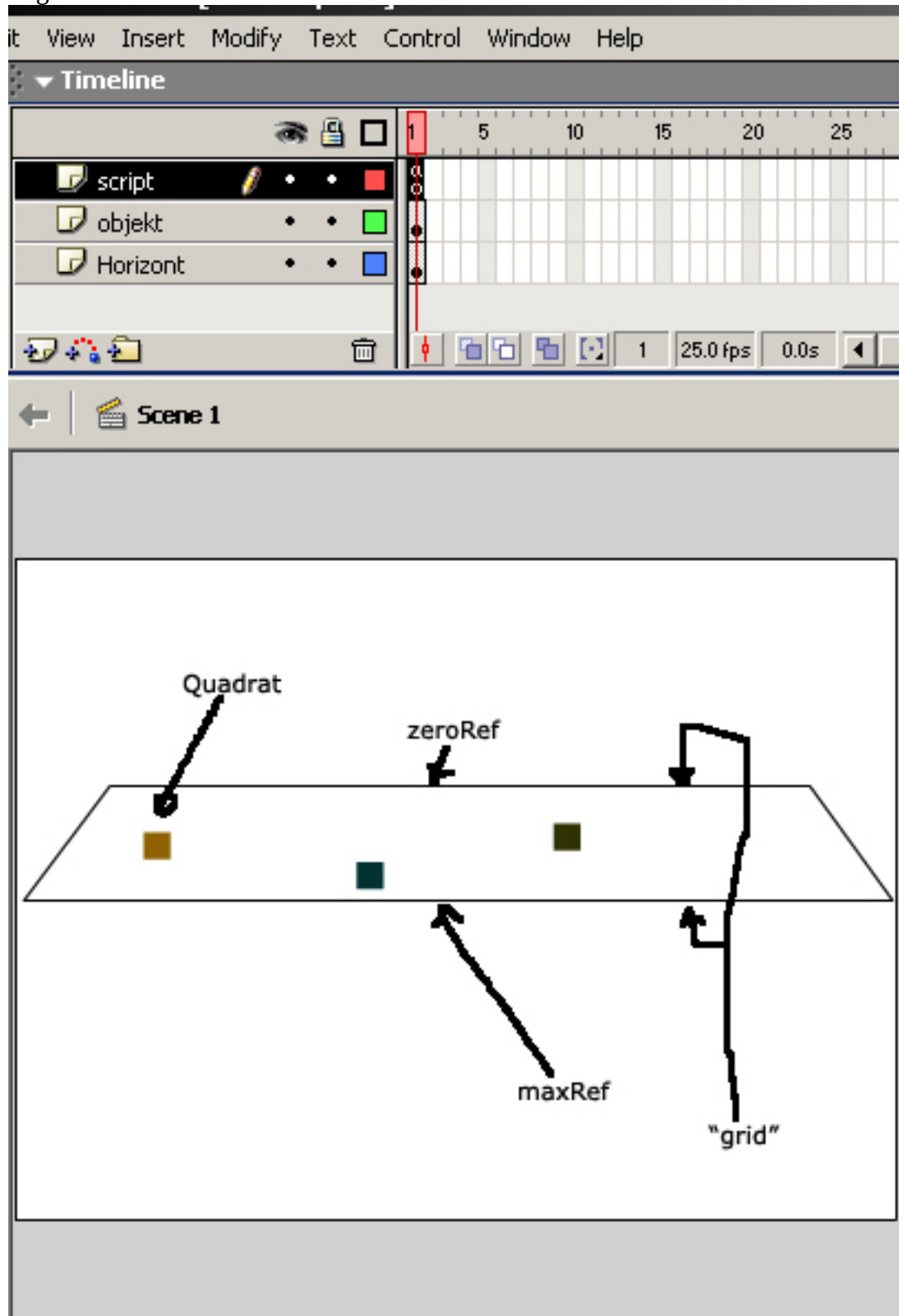
Diese Variablen wurden ja schon mit `getBounds()` erhalten. D.h. dass die Ref-Punkte ganz oben und ganz unten im des `grid-MovieClips` angesetzt werden. Wenn sich das Quadrat also hinter dem Gitter befindet, erhalten wir 0% für die Skalierung, im Vordergrund 100%. Wenn es sich hinter dem Gitter befindet muss die Skalierung ebenfalls 0% sein, andernfalls stimmt der Prozentsatz nicht, weshalb in der Funktion `zeroRef` von `ypos` abziehen, was die `ypos`-Variable auf den richtigen Wert setzt.

Damit das ganze funktioniert, muss die Funktion nur noch aufgerufen werden, wenn das Quadrat gezogen wird.

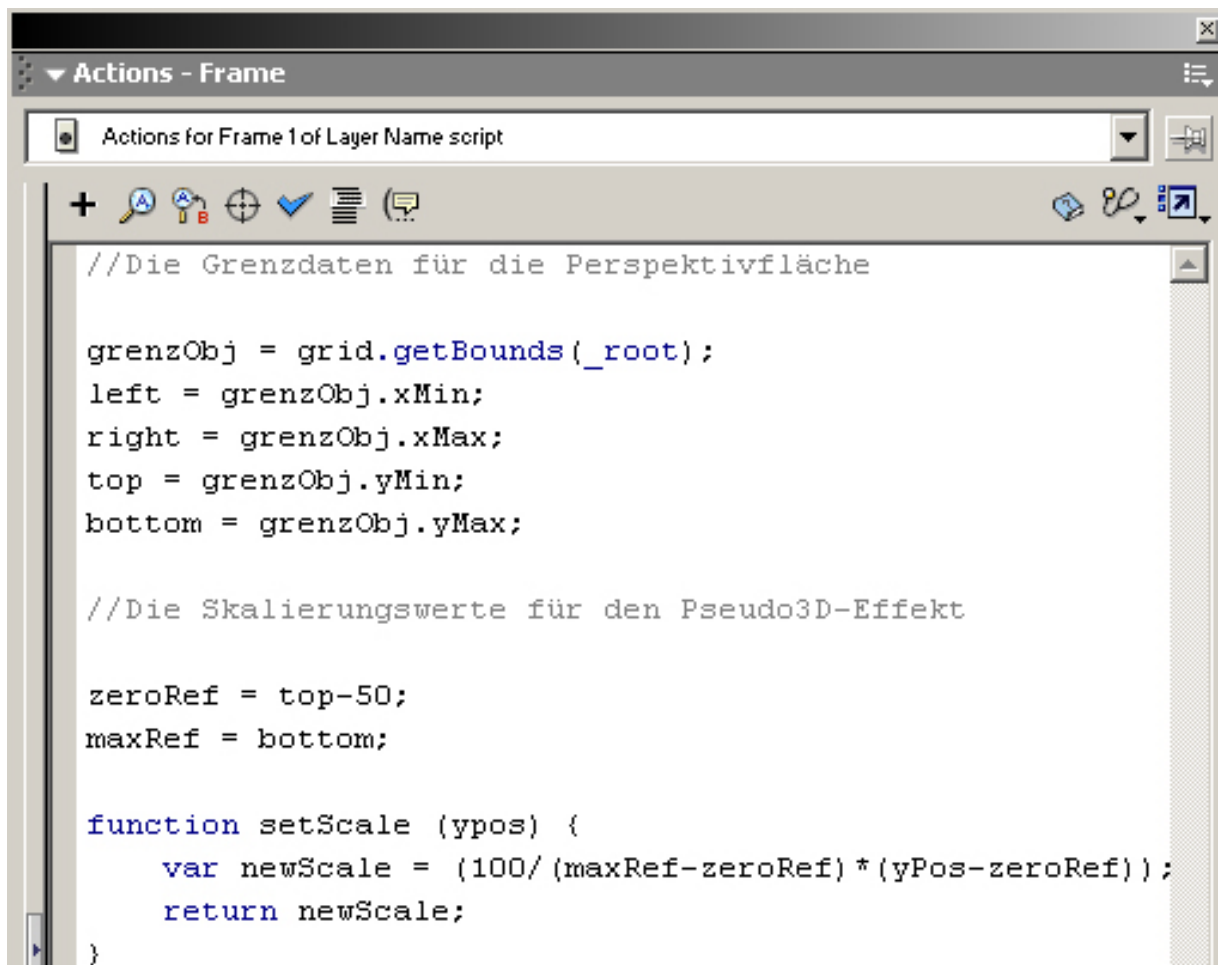
Dazu klicken wir auf das Quadrat und wechseln ins Action-Fenster um folgenden Code einzugeben:

```
onClipEvent (mouseMove) {
    this._xscale = _root.setScale(this._y);
    this._yscale = _root.setScale(this._y);
    this.alpha = _root.setScale(this._y);
    this.swapDepths(_root.setScale(this._y));
}
```

Diese Funktion wird immer dann aufgerufen, wenn sich die Maus bewegt. Der Code setzt einfach die `_xscale` und `_yscale` Eigenschaften des Movieclips auf den Wert, der nach dem Aufruf der `setScale`-Funktion zurückgegeben wurde, gleich der Alphawertänderung. Die letzte Zeile gibt den Z-Index, damit bei mehreren Quadraten (kopieren und testen!) die vorderen auch über den hinteren liegen. So, teil eins wäre geschafft. Das sieht dann alles ungefähr so aus:



Der Code für die Hauptzeitleiste:



The screenshot shows a software interface window titled "Actions - Frame". Inside, there is a text area containing ActionScript code. The code defines variables for the boundaries of a perspective view and a function to calculate a scale factor based on a y-position.

```
//Die Grenzdaten für die Perspektivfläche

grenzObj = grid.getBounds(_root);
left = grenzObj.xMin;
right = grenzObj.xMax;
top = grenzObj.yMin;
bottom = grenzObj.yMax;

//Die Skalierungswerte für den Pseudo3D-Effekt

zeroRef = top-50;
maxRef = bottom;

function setScale (ypos) {
    var newScale = (100/ (maxRef-zeroRef) * (yPos-zeroRef));
    return newScale;
}
```

II

Minimale Animationen der Objekte sowie Grundkenntnisse für Objektbewegung

Jetzt ersetzen wir diese Quadrate einfach durch andere Objekte und lassen dieses grid-Ding verschwinden und legen ein Bild drüber, damit das ganze nicht mehr so scheiße aussieht. Also, irgendein Bild her (mit Perspektive) und für die Quadrate wählte ich diese netten kleinen geflügelten >h<, die dann am Horizont verschwindend umherfliegen, sich widerwillig in den Vordergrund ziehen lassen, um dann doch auch sogleich sich wieder zu entfernen. D. h. statt des Quadrates im Movieclip nehmen wir einfach diese Buchstabenordnung und schon dragt sich diese wohlgeordnet in die Tiefe des Bildes, oder wie auch immer.

Damit diese sich am „Horizont“ hin und her bewegen ist noch eine Kleinigkeit an Code fällig. Sie sollen sich zufällig von hier nach dort bewegen. Das hier sind die jeweils momentanen x/y-Koordinaten des Objektes im Koordinatensystem. Das Dort wird per eingegrenzten Zufall gewürfelt, sobald das Objekt nah genug an seinem vorigem „Dort“ angekommen ist. Es soll sich einigermaßen smooth bewegen, d. h. zum Dort hin langsamer werdend, um dann ruckartig eine neue Richtung anzupeilen. Das wird hier folgendermaßen gelöst :

In den Quadrat-MovieClip („graf“) von I ergänzen wir das bisherige Script um folgende Zeilen:

```
onClipEvent (enterFrame){
    if (this._x < newx+10 && this._x > newx-10){
        newx = random(650)+100;
        newy = random(40)+_root.top - 20;
    }

    this._x = this._x - (this._x - newx)/60;
    this._y = this._y - (this._y - newy)/40;

    this._xscale = _root.setScale(this._y);
    this._yscale = _root.setScale(this._y);
    this._alpha = _root.setScale(this._y)-random(20)+10;
    this.swapDepths(_root.setScale(this._y));
}
```

Die unteren vier Zeilen kennen wir schon. Den `onClipEvent` haben wir in einen `enterFrame` umgewandelt, d. h. jeden Framewechsel (hier 25fps) wird dieser Code ausgeführt. Das ist notwendig, damit sich überhaupt etwas bewegt. Die erste `if`-Schleife fragt ab, ob das Objekt schon in der Nähe von Dort (`newx/newy`) ist. Wenn das der Fall ist, werden neue `newx` und `newy` „ausgewürfelt“ (`random`). Für `newx` wird ein Wertebereich von 100 bis 750 begrenzt, für `newy` liegt der Zufallswert +- 20 des Horizontwertes (`top` ist eine Variable der Hauptzeitleiste, deswegen auch `_root` davor).

Die beiden Codezeilen in der Mitte sorgen für Bewegung. Der y-Wert dieses (`this._y`) Objektes wird jeden Frame auf den neuesten Stand gebracht. D.h. der neue Wert = der jetzige Wert – (den jetzigen Wert – das gesetzte Dort (`newy`) geteilt durch einen Faktor. Je höher der Faktor, desto langsamer bewegt sich das Objekt, da der Wert vor dem Minuszeichen dadurch kleiner wird. Der errechnete Wert, der von der jetzigen Position abgezogen wird, wird immer kleiner, wodurch mit zunehmender Annäherung an das Dort (`newy`) das Objekt in immer kleineren Schritten sich bewegt, d. h., langsamer wird.

Damit sich Anfangs schon etwas tut geben wir vor diesen Zeilen einen `onClipEvent(load)` Befehl ein, der schon mal `newx` und `newy` vorsetzt:

```
onClipEvent(load){
    grad = 20;
    newx = random(650)+100;
    newy = random(40)+_root.top;
}
```

Soweit zur Grundbewegung der >h<.


Nun sollen sich diese Viecher aber noch in sich bewegen und vornehmlich hektischer, wenn man sie in den Vordergrund zieht. Dies wird mit dem Befehl `_rotation` bewerkstelligt.


```
onClipEvent(enterFrame){
    gradfaktor = (random(grad)-(grad/2));
    this._rotation = gradfaktor;
}
```



Wieder ein `enterFrame` Befehl (könnte man auch in den anderen hineinschreiben). Die Variable `grad` wird Anfangs gegeben (s. o.) und je nach Dragszustand verändert (während des Drag-Vorgangs setzt sich der Wert auf 120; einfach zu dem Buttonbefehl hinzufügen, in dem auch der `startDrag`-Befehl notiert ist.


Bei `stopDrag` dann den Wert wieder auf 20 setzen.). `gradfaktor` ist eine Variable, die den Null-Grad-Wert in die Mitte setzt. D. h. bei `grad=20`, ergeben sich Zufallswerte zwischen -10° und $+10^\circ$; bei `grad=120` (während des Dragvorganges) ändert sich der Zufallsbereich auf -60° bis $+60^\circ$. Daher zuckt dieses Ding dann so nett durch die Gegend. So weit. Testen!
Das ganze müsste dann aussehen wie die Abbildung auf der nächsten Seite :

Timeline

script • •  1 5 10 15 20 25

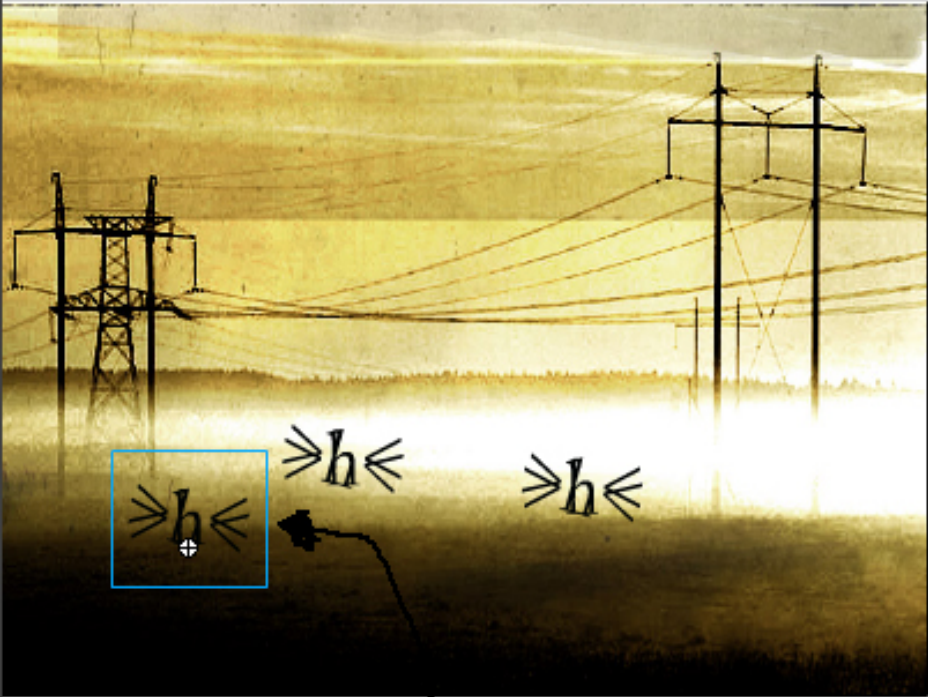
sound • • 

objekt  • • 

Horizont • • 

1 25.0 fps 0.0s

Scene 1



Die fliegenden Teilchen
(zuvor Quadrate)

Properties

Movie Clip Instance of: graf Color: Tint

graf Swap... RGB: 0 0

W: 133.9 X: 95.0

H: 117.3 Y: 389.7

III Soundsteuerung

Vorbereitet sind eine Fläche die geloopt wird, sowie drei kleine Soundschnipsel für die >h<, deren y-Position die Lautstärke bestimmen. In diesem Script über den `alpha`-Wert gekoppelt, der ja ebenfalls sich aus der `setScale` Funktion der Hauptzeitleiste errechnet. Dazu brauchen wir Sound-Objekt; und diese werden in der Hauptzeitleiste, d. h. in der Script-Ebene, erstellt. Dort steht ja schon allerlei, und danach müsste dort dann folgendes zusammengefasst stehen:

```
//Die Grenzdaten für die Perspektivfläche

grenzObj = grid.getBounds(_root);
left = grenzObj.xMin;
right = grenzObj.xMax;
top = grenzObj.yMin;
bottom = grenzObj.yMax;

//Die Skalierungswerte für den Pseudo3D-Effekt

zeroRef = top-50;
maxRef = bottom;

function setScale (ypos) {
var newScale = (100/(maxRef-zeroRef)*(ypos-zeroRef));
return newScale;
}

//Der Sound
fliege1 = new Sound (klang);
fliege1.attachSound("chor4.wav");
fliege1.start(0, 999);

fliege2 = new Sound (klang2);
fliege2.attachSound("chor5.wav");
fliege2.start(0, 999)

fliege3 = new Sound (klang3);
fliege3.attachSound("chormagni.wav");
fliege3.start(0, 999);
```

Die obersten Zeilen kennen wir ja schon. Nun zum Sound. Die Hintergrundmusik legen wir einfach auf eine extraebene und sagen ihr, sie soll sich tausendfach wiederholen (das steht hier nicht, ist nu so 'ne Bemerkung). `fliege1` und dergl. sind Variablennamen, denen ein `newSound` – Objekt zugrunde gelegt wird. In Klammern steht der Name eines leeren Movieclips, der irgendwo auf der Bühne liegt. Er ist wirklich leer, denn wir brauchen nur seinen Instanznamen, um mehrere Soundobjekten gezielt ansteuern zu können, ohne gleich alle anderen mit zu verändern. `klang2` und `klang3` sind Kopien von `klang`. Der `attachSound` – Befehl verbindet mit dem Variablennamen ein bestimmtes Soundfile aus der Library. D.h. keines dieser drei Soundfiles liegt auf der Bühne oder auf einer Ebene der Zeitleiste, sie werden über `attach` lediglich verknüpft. Damit das funktioniert müssen den jeweiligen Soundfiles in der Library (rechter Mausklick drauf) über „Linkage“ für den Actionscript Export freigegeben und benannt werden. Dieser eingegebene Name steht dann im Code in Anführungszeichen, da es sich um Buchstabenaneinanderreihungen zur Namensgebung handelt und nicht um eine Klasse oder Variable. Der `_start` – Befehl spielt das Soundfile dann ab. Der erste Wert in der Klammer gibt den Offset an, der zweite die Anzahl der Wiederholungen. Wenn das alles so dasteht, müsste auch alles funktionieren.

Dann nur noch in den jeweiligen >h< - Movies die `_setVolume` Werte bestimmen (siehe Abb. Oben). Und fertig ist die Übung. Wenn man da noch ein paar Tage weiterbastelt und hier und da kleine Raffinessen einbaut, könnte das ganz nett werden.