

A c t i o n S c r i p t # 1

T u t o r i a l

Inhaltsverzeichnis

INHALTSVERZEICHNIS	1
ABBILDUNGSVERZEICHNIS	2
TABELLENVERZEICHNIS	3
1 EINLEITUNG	4
1.1 WAS IST EIN ACTIONSCRIPT?	4
1.2 MÖGLICHE ANWENDUNGEN FÜR ACTIONSCRIPTS	4
1.3 VARIABLEN	4
1.4 CODE-KONVENTIONEN	5
1.5 KOMMENTARE	6
1.6 OPERATOREN	6
1.7 VERKNÜPFUNGS-OPERATOREN	6
1.8 EVENTS & BUTTONSKRIPTS	7
1.9 EIGENSCHAFTEN VON MOVIE-CLIPS	8
2 WORKSHOP – INTERAKTIVE SCHIFFSSTEUERUNG	10
3 ERLÄUTERUNGEN ZUR OBJEKTORIENTIERTEN PROGRAMMIERUNG	18
3.1 OBJEKTORIENTIERTE SKRIPTS	18
3.2 VERWENDUNG VON BENUTZERDEFINIERTEN OBJEKTEN	20
3.3 ERSTELLUNG VON OBJEKTEN	20
3.4 VERERBUNG	21

Abbildungsverzeichnis

Abbildung 1: Filmeigenschaften definieren	10
Abbildung 2: Erstellung des Bootsrumpfes in Freehand	11
Abbildung 3: Symboleigenschaften einstellen	11
Abbildung 4: Das Fenster Objektaktionen	12
Abbildung 5: ActionScript des MovieClips - Boot	13
Abbildung 6: Neue Ebene anlegen.....	14
Abbildung 7: ActionScript-Code – Bootssteuerung	15
Abbildung 8: Trigometrische Funktionen – Der Einheitskreis	16

Tabellenverzeichnis

Tabelle 1: Übersicht wichtiger Operatoren	7
Tabelle 2: Wichtige Movie-Clips Properties.....	9

1 Einleitung

1.1 Was ist ein ActionScript?

ActionScript ist eine sehr einfache Programmiersprache. Sie bietet die Möglichkeit Zeitleisten, Sounds, Farben, Cursor, Grafiken und Daten zu steuern.

ActionScripts basiert auf den selben Standards wie JavaScript.

1.2 Mögliche Anwendungen für ActionScripts

- Komplexe Interaktive und lineare Animationen
- Dynamische Navigation
- Interaktive Applikationssteuerung
- Spiele
- Eingabeüberprüfungen etc.

Die Programmierung in ActionScript ist relativ einfach. Zudem befindet sich im Programmierfenster ein sogenannter Debugger. Er hilft dabei, dass die generierten Szenen problemlos funktionieren und zeigt Eingabefehler schon während der Programmierung an (im Normal-, nicht im Expertenmodus). Einfachere Interaktionen wie Pull-Down-Menüs oder Buttons können mit Hilfe so genannter Smart Clips erstellt werden.

1.3 Variablen

Arten von Variablen ("TYPEN"):

- String
`student="Karl-Heinz";`
`student_lieblingsfarbe ="Rot";` ("`=`" ist ein Zuweisungsoperator)

- **Number**

`i=1; Integer` (ganze Zahl)

`x=2.34, y= 1.34;` (Fließkommazahl)

- **Boolean**

`game_over=true;` (true oder false)

⇒Flash ist eine typfreie Sprache. Man muss den Typ einer Variablen nicht vorher deklarieren. ⇒Typumwandlungen werden automatisch vorgenommen. Beispiel für eine automatische Typumwandlung:

```
a=24.34;
```

```
a="peter";
```

- **Spezieller TYP: NULL (Undefiniert)**

Alle Variablen, die noch nicht definiert sind, sind NULL. Bisher noch nicht verwendete, unbekannte Variable. Häufig, wenn man wissen will, ob es ein bestimmtes Objekt schon gibt (...)

1.4 Code-Konventionen

Benennungen von Variablen:

- Ganzzahlen, Schleifenzähler benennt man in der Regel mit i, j, k
- Variablen Namen -> ausschließlich Kleinbuchstaben
- Konstanten Namen -> ausschließlich Großbuchstaben
- Variablen-Namen sollten in keinem Fall deutsche Umlaute oder Leerzeichen enthalten
- Variablen sollten nicht mit Nummer beginnen
- keine Sonderzeichen, oder Namen aus der Flash-Syntax benutzen ⇒ diese, schlecht erkennbare Fehler-Quelle
- Namen sollten kurz & lesbar sein und den Zweck beschreiben. Man sollte möglichst in einer Sprache bleiben.

1.5 Kommentare

Kommentare werden mit Hilfe zweier Slashes eingeleitet: `// ..` und können dann im Source-Code abgelegt werden. Kommentare sind wichtig, um auch nach längerer Zeit den Quell-Code noch selbst verstehen zu können – oder damit ihn auch andere Personen schneller nachvollziehen können.

1.6 Operatoren

- Zuweisungs-Operatoren:

```
student_name = "Peter";
```

- Vergleichsoperatoren

```
student_name != "Frank";  
mein_object != null;  
student_name == "Gerlinde";  
student_anzahl >= 5;  
student_anzahl < 1;
```

⇒ Die Auswertung eines solchen Vergleichs ergibt entweder *true* oder *false* und wird fast immer mit if-Abfragen benutzt

```
guter_raum = (student_anzahl <= 10) && (anzahl_computer >= 10);
```

⇒ Dabei werden zu erst die Klammern ausgewertet und dann mit logisch "und" erkettet

1.7 Verknüpfungs-Operatoren

```
student_name = student_vorname + student_nachname;  
student_anzahl++;  
student_anzahl += 10;
```

=	Zuweisung		
+	Summe (Nummern), (!) Auch für Strings als Verkettung: name="Peter "+" Gernegroß"	-	Subtraktion
*	Multiplikation (erwartet Integer, oder Float's)	/	Subtraktion
!=	Vergleichs-OP "ungleich"	==	Identisch (!) Achtung niemals mit "=" verwechseln
>	Grösser als	<	Kleiner
>=	Grösser-Gleich	<=	Kleiner Gleich
	Logischer Operator "oder"	&&	Logischer Operator "und"
--	Decrement (Erniedrigung um 1) (Bspl.): i--; ist dasselbe wie: i = i - 1; (!) i-- ist nicht dasselbe wie --i (was es auch gibt)	++	Increment (Erhöhung um 1)
=	a=b kurz für: a=a*b;	+=	a+=b kurz für: a=a+b;
/=	a/=b kurz für: a=a/b;	-=	a-=b kurz für: a=a-b;

Tabelle 1: Übersicht wichtiger Operatoren

1.8 Events & Buttonskripts

Für Events, die abhängig von Darstellern sind (die Mouse befindet sich über einer Schaltfläche) benötigt man einen Button-Darsteller! Andere Darsteller können diese Events nicht empfangen!

<i>press</i>	The mouse button is pressed while the pointer is over the button.
<i>release</i>	The mouse button is released while the pointer is over the button.
<i>releaseOutside</i>	The mouse button is released while the pointer is outside the button.
<i>rollover</i>	The mouse pointer rolls over the button.
<i>rollout</i>	The pointer rolls outside of the button area.
<i>dragOver</i>	While the pointer is over the button, the mouse button has been pressed while rolled outside the button, and then rolled back over the button.
<i>dragOut</i>	While the pointer is over the button, the mouse button is pressed and then rolls outside the button area.
<i>keyPress ("key")</i>	The specified key is pressed.

1.9 Eigenschaften von Movie-Clips

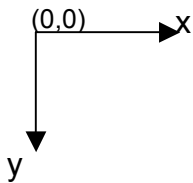
Die Instanzen von Movie-Clips besitzen Eigenschaften, die über ActionScript abgefragt und modifiziert werden können. Eigenschaften sind z.B. die Position, Rotation, Sichtbarkeit etc...

Sämtliche Properties besitzen einen Unterstrich in der Schreibweise und werden in der DOT-Schreibweise an den Movie-Clip Instanz-Namen angehängt (Eigenschaften von Objekten):

Beispiel: Setzen der X-Koordinate der Instanz "mc":

```
mc._x=300;
```

Das Koordinaten-System läuft von oben links nach unten rechts. Einheiten werden in Pixel angegeben (unskaliert).



Auf die meisten Eigenschaften kann man "lesend" als auch "schreibend" zugreifen:

Bspl.: Verschieben des Movie Clips mc um 50 Pixel nach Rechts:

```
mc._x= mc._x +50; //Kurz auch: (mc._x += 50)
```

<u>_x</u>	Bühnen-x-Koordinate	Integer/Komma-Zahlen sind möglich [Flash-Pixel]
<u>_y</u>	Bühnen-y-Koordinate	Integer [Flash-Pixel]

_rotation	Rotation	0-350 [°]
_alpha	Transparenz	0-100 [%]
_visible	Sichtbarkeit	0 oder 1 (boolean true/false)
_width	Breite	Flash-Pixel
_height	Höhe	Flash-Pixel
_xscale	X-Skalierung (horizontal)	0-100 [%]
_yscale	Y-Skalierung (vertikal)	0-100 [%]
_xmouse	X-Position der Mouse	Pixel siehe unten
_ymouse	Y-Position der Mouse	Pixel siehe unten

Tabelle 2: Wichtige Movie-Clips Properties

2 Workshop – Interaktive Schiffssteuerung

Ziel: Es soll eine einfache, interaktive Bootssteuerung (bspw. als Teil eines Spiels) in Flash entwickelt werden. Die Steuerung des Schiffes soll über die Tastatur realisiert werden.

1. Zunächst wird in Flash ein neuer Film angelegt (\Rightarrow *Datei, Neu*). Dieser wird über \Rightarrow *Modifizieren, Film* auf ein Format von bspw. 800 X 600 Pixel gebracht. Zusätzlich kann die Hintergrundfarbe eingestellt werden.

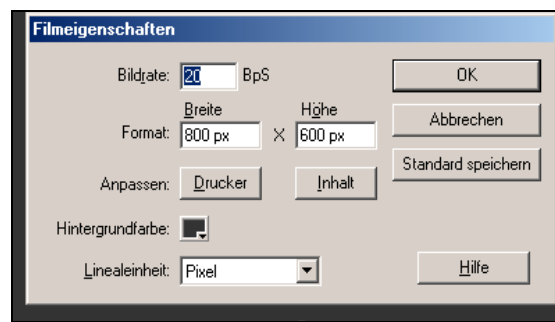


Abbildung 1: Filmeigenschaften definieren

Anschließend werden die Einstellungen mit \Rightarrow *OK* bestätigt und der Film abgespeichert (\Rightarrow *Datei, Speichern unter*).

2. Im nächsten Schritt wird ein stilisierter Bootsrumph erstellt. Sinnvoll ist bspw. eine Erstellung in Freehand. (Um spätere Probleme zu vermeiden: Bitte den Rumpf mit dem Bug nach rechts entwerfen! Vgl. Abbildung 2.) Ist der Bootsrumph designed, kann er gespeichert und anschließend in Flash importiert werden – oder er kann mittels *Copy and Paste* (\Rightarrow *Strg/Apfel + C*, *Strg/Apfel + V*) direkt in Flash eingefügt werden (natürlich kann der Rumpf auch direkt in Flash erstellt werden – hier sind die Möglichkeiten gegenüber Freehand aber wesentlich eingeschränkt).

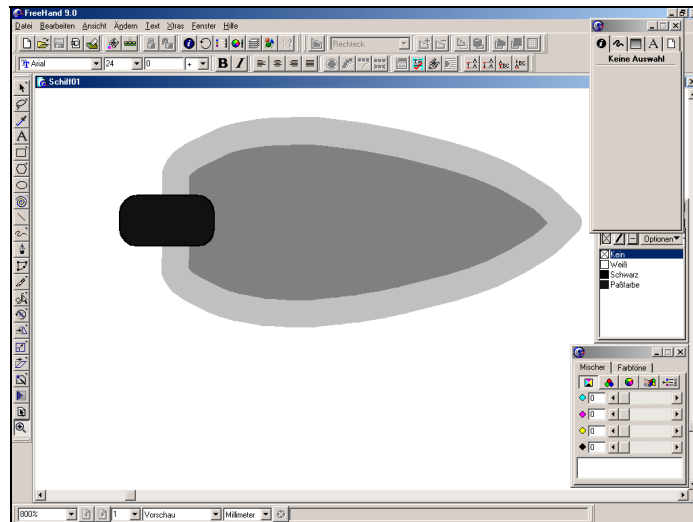


Abbildung 2: Erstellung des Bootsrumpfes in Freehand

3. Ist der Bootsrumpf in Flash eingefügt, muss dieser in ein Symbol konvertiert werden. (⇒Dazu wird der Rumpf mit gehaltener rechter Mausextaste markiert, dann: *Einfügen, In Symbol konvertieren*)

Es öffnet sich das Fenster für die Symboleigenschaften. Der Rumpf bekommt hier bspw. den Namen „Boot“ und das Verhalten „Filmsequenz“ zugewiesen.

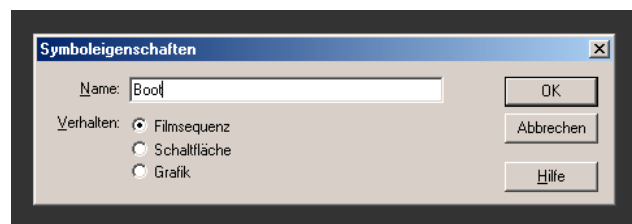


Abbildung 3: Symboleigenschaften einstellen

4. Ist der Bootsrumpf erfolgreich in ein „Filmsequenz-Symbol“ konvertiert worden, wird dieser erneut mit der Mouse aktiviert (⇒Rechtsklick). Anschließend wird das Objektaktionen-Fenster geöffnet (⇒*Fenster, Aktionen*) – hier können ActionScripts entwickelt werden. Es können zwei verschiedene Modi eingestellt werden:

Zum einen der *normale Modus* – hier kann man auf vorgefertigte „Skript-Bauteile“ zurückgreifen, um Aktionen zu erstellen, indem man auf der linken Seite des Bedienfeldes Aktionen aus der Werkzeugliste auswählt. Die Werkzeugliste enthält die Kategorien "Basisaktionen", "Aktionen", "Operatoren", "Funktionen", "Eigenschaften" und "Objekte". Die Kategorie "Basisaktionen" enthält die einfachsten Flash Aktionen und ist nur im normalen Modus verfügbar. Die ausgewählten Aktionen sind auf der rechten Seite des Bedienfeldes in der Aktionsliste aufgelistet. Sie haben die Möglichkeit, Aktionen hinzuzufügen, zu löschen oder deren Reihenfolge zu ändern. Sie können außerdem in den Parameterfeldern unten im Bedienfeld Parameter (Argumente) für Aktionen eingeben.

Zum anderen gibt es den *Expertenmodus*, hier kann man die Skripte vollständig selbst verfassen. Dieser Modus soll für den Workshop aktiviert werden (vgl. Abbildung 4).

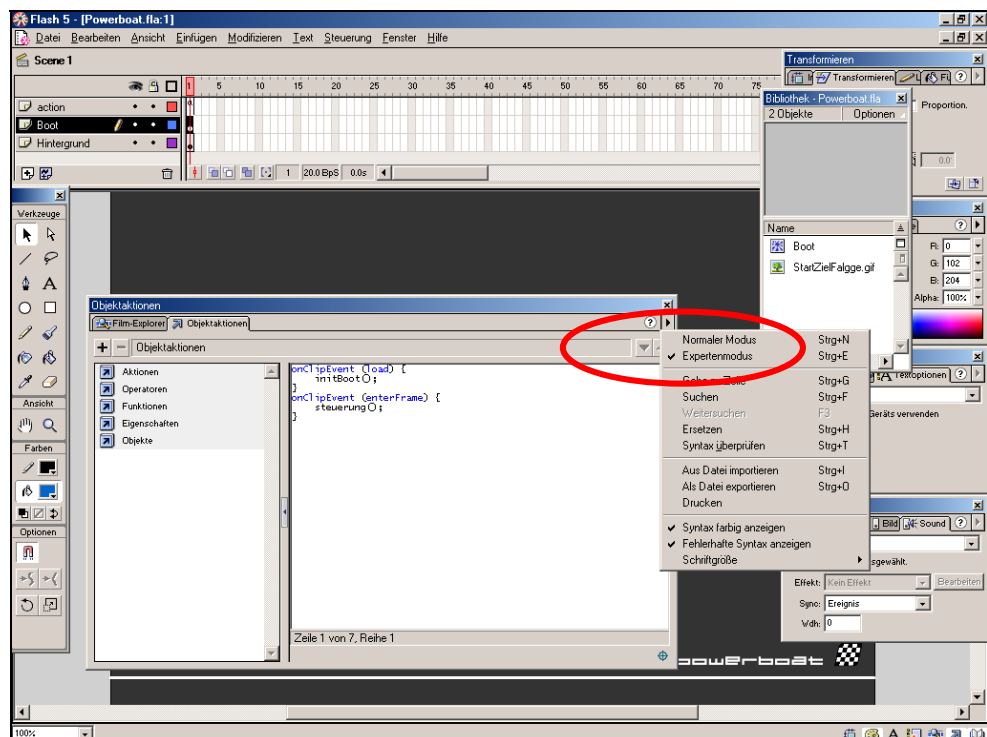


Abbildung 4: Das Fenster Objektaktionen

Tipp: Falls es Probleme beim Verfassen von ActionScripten gibt, ist die in Flash implementierte **Hilfe** bzw. das **ActionScript-Lexikon** und die **ActionScript-Referenz** sehr hilfreich!

5. Für die Steuerung des MovieClip-Objekts Boot, soll auf zwei Funktionen zugegriffen werden. Eine soll das Boot initialisieren – d.h., dass den Eigenschaften (und deren Variablen) des Objektes Boot bestimmte Anfangswerte zugewiesen werden. Diese Initialisierung soll demnach nur einmal – zu Beginn des Films aufgerufen werden. Sie wird `initBoot();` genannt.

Für die eigentliche Steuerung des Bootes, die permanent erfolgt, soll auf eine Funktion namens `steuerung();` zugegriffen werden. Es muss demnach folgender ActionScript-Code in das Objektfenster geschrieben werden (vgl. Abbildung 5):

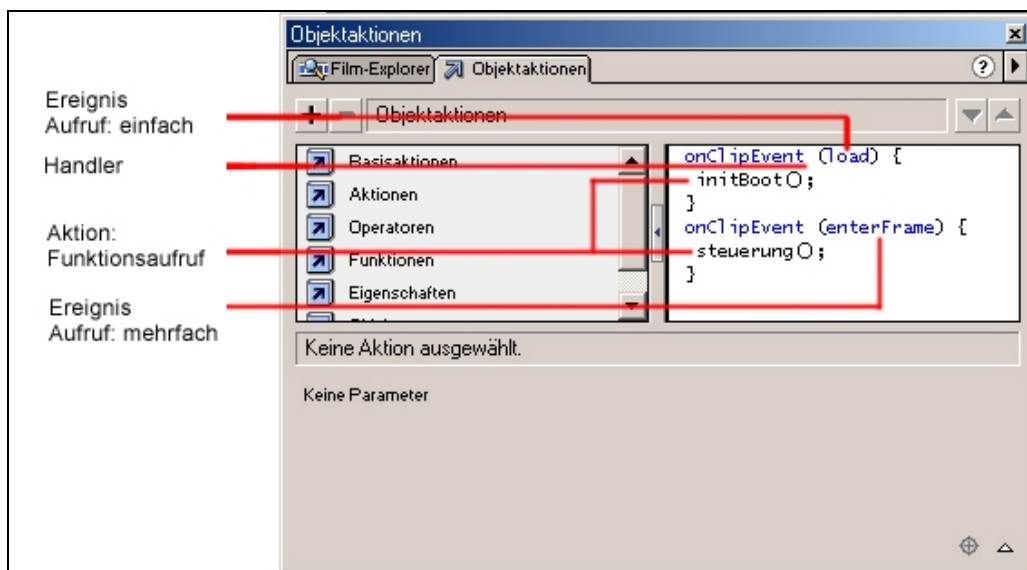


Abbildung 5: ActionScript des MovieClips - Boot

Dabei kann mit der Aktion (⇒Handler) `onClipEvent` der Filmsequenzinstanz (Instanz des Objektes Boot) auf der Bühne direkt Aktionen zugeordnet werden. Die Aktion "`onClipEvent`" enthält Ereignisse wie `load`, `Bild`

eingeben, Mouse ziehen und Daten, mit denen man komplexe Formen der Interaktivität herstellen kann.

Es sind zwei onClipEvent-Handler mit zwei unterschiedlichen Ereignissen vorhanden: load und enterFrame. Die Aktion in der Anweisung onClipEvent(load) wird nur einmal beim Laden des Filmes ausgeführt (⇒ Funktionsaufruf initBoot();)

Die Aktion in der Anweisung onClipEvent(enterFrame) wird hingegen jedes mal ausgeführt, wenn der Abspielkopf das Bild erreicht, in dem die Funktion in der Zeitleiste abgelegt ist. Selbst in einem Film mit der Länge von einem Bild wird dieses Bild vom Abspielkopf wiederholt angesteuert, wobei das Skript wiederholt abläuft. ⇒ Die Aktion des Funktionsaufrufs für die Steuerung des Schiffes (steuerung();) wird „permanent“ aufgerufen.

6. Für die ActionScript-Funktionen, mittels derer die Steuerung des Bootsrumpfes realisiert werden soll (initBoot(); und steuerung();), wird in der Zeitleiste eine neue Ebene angelegt (Name: Bspw. „action“ – vgl. Abbildung 6).



Abbildung 6: Neue Ebene anlegen

Der ActionScript-Code für die Steuerung des Bootes ist in der folgenden Abbildung dargestellt. Zur Veranschaulichung enthält er bereits die entsprechenden Erläuterungen in der Form von Kommentaren, die direkt in den Quell-Code eingefügt worden sind.

```

movieClip.prototype.initBoot = function () { // Initialisierung des Bootes - wird nur einmal aufgerufen
    winkel=-90; // Boot wird um 90 Grad nach links gedreht
    maxTempo=8; // Zuweisung der maximalen Geschwindigkeit des Bootes
    tempo=0; // aktuelles Tempo wird beim Laden des Films auf null gesetzt
}

movieClip.prototype.steuerung = function () { // Bootsteuerung über Tastatur - wird permanent durchlaufen (vgl. Aufruf)
    if (key.isDown(key.Left)) { // if-Schleife mit Bedingung Pfeil nach links gedrückt
        winkel -= 8; // Pfeil nach links -> Winkel minus 8 Grad
    } // if-Schleife mit Bedingung Pfeil nach rechts gedrückt
    if (key.isDown(key.Right)) { // Pfeil nach rechts -> Winkel plus 8 Grad
        winkel += 8;
    }

    nWinkelradiant = (winkel*Math.PI/180); // Umrechnung des Winkes in einen Winkelradiant
    // (vgl. Formel) ist für sin und cos-Funktionen
    // notwendig!

    _rotation=winkel; // Drehung des Bootes analog zum aktuellen Winkel

    posX = Math.cos(nWinkelradiant); // Ermittlung zweier Faktoren (für Bewegung in X-
    // Y-Richtung) anhand von Trigonometrischen Funktionen
    // (cos = für X- und sin = für Y-Richtung)
    posY = Math.sin(nWinkelradiant); // zur Ermittlung des Bewegungsänderungs-Verhältnisses
    // des Bootes -> Faktoren müssen anschließend mit dem
    // Tempo multipliziert werden

    if (key.isDown(key.Up)) { // if-Schleife mit Bedingung Pfeil nach oben gedrückt
        if (tempo<maxTempo) { // if-Schleife mit Bedingung: Wenn aktuelles Tempo
            // kleiner als das Maximale (=8)
            // ... dann erhöhe bei jedem Durchlauf der Schleife
            // (solange Pfeil nach oben gedrückt) das Tempo um eins
            tempo++;
        }
    }

    _x += (posX)*tempo; // MovieClip-Eigenschaft: Verschiebe Boot um ... Pixel in Richtung X
    _y += (posY)*tempo; // MovieClip-Eigenschaft: Verschiebe Boot um ... Pixel in Richtung Y
    Grenze (0,800,0,500,0.5,20,780,20,480); // Definition der Grenzen für Variablen der Funktion
    // movieClip.prototype.Grenze (siehe unten) -> Werte werden an
    // diese Funktion übergeben
}

movieClip.prototype.Grenze = function (minX,maxX,minY,maxY,faktor,nMinX,nMaxX,nMinY,nMaxY) { // Aufruf nach jeder Positionsänderung
    if (tempo>0) { // if-Abfrage: Falls tempo höher 0:
        tempo=tempo-faktor; // verringere Tempo (um 0,5) so lange, bis Tempo = 0 (=Abbruchbedingung)
    }
    if (_x >= maxX) { // if-Abfrage: Falls X-Position größer gleich 800 Pixel:
        _x = nMinX; // setze Bootsposition hinsichtlich der X-Koordinate auf 20 Pixel
    }
    if (_x <= minX) { // if-Abfrage: Falls X-Position kleiner gleich 20 Pixel:
        _x = nMaxX; // setze Bootsposition hinsichtlich der X-Koordinate auf 780 Pixel
    }
    if (_y >= maxY) { // if-Abfrage: Falls Y-Position größer gleich 500 Pixel:
        _y = nMinY; // setze Bootsposition hinsichtlich der Y-Koordinate auf 20 Pixel
    }
    if (_y <= minY) { // if-Abfrage: Falls Y-Position kleiner gleich 0 Pixel:
        _y = nMaxY; // setze Bootsposition hinsichtlich der Y-Koordinate auf 480 Pixel
    }
}

```

Abbildung 7: ActionScript-Code – Bootssteuerung

Wie in der Abbildung 7 dargestellt, gibt es neben der Initialisierungs- und der Steuerungsfunktion zusätzlich eine dritte Funktion: Die Funktion `movieclip.prototype.Grenze` definiert die Grenzen der Bootsbewegung. Die Werte für die entsprechenden Variablendefinitionen (`minX`, `maxX` etc.) werden dabei von der Steuerungsfunktion übergeben.

Für jede Positionsänderung aus der Steuerungsfunktion wird demnach in der Grenzfunktion geprüft, ob die entsprechenden Bedingungen aus den `if`-Abfrage erfüllt sind (vgl. Abbildung 7).

Exkurs in die Geometrie:

⇒ Trigometrische Funktionen:

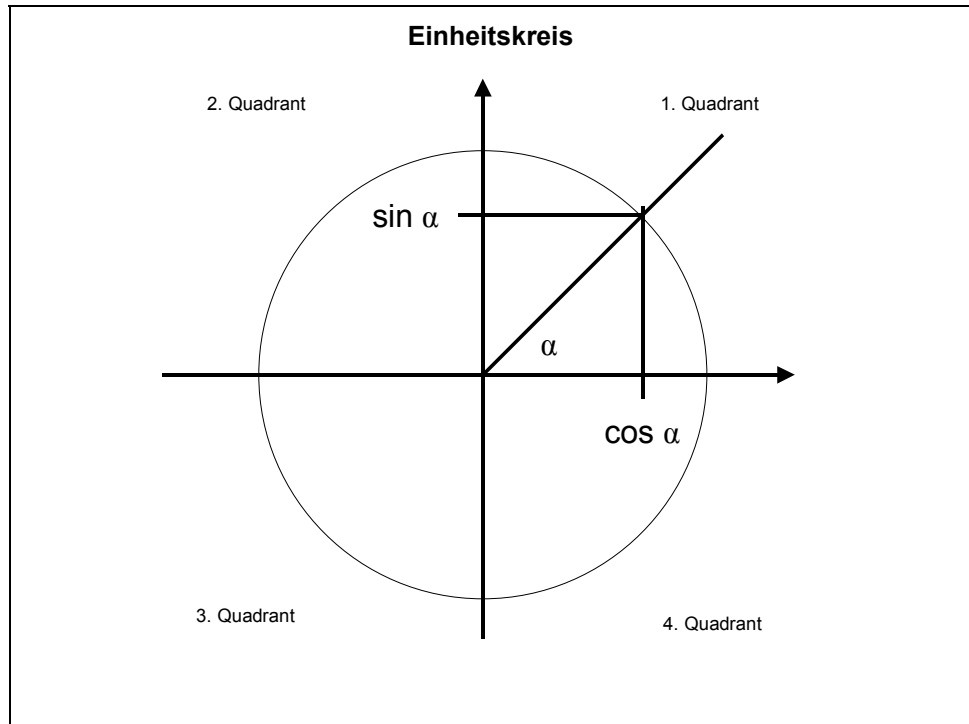


Abbildung 8: Trigometrische Funktionen – Der Einheitskreis

Um Trigometrische Funktionen wie $\sin(x)$ oder $\cos(x)$ unter ActionScript einsetzen zu können, müssen die Winkel x in Winkelradianten umgewandelt werden. Dies erfolgt anhand der Formel:

$$\text{WinkelradiantX} = \text{Winkel} * \text{PI}/180$$

Über `posX` und `posY` und das `tempo` wird die Bewegung des Bootes in X- und Y-Richtung berechnet. Dabei kommen wie in Abbildung 8 dargestellt Trigometrische Funktionen zum Einsatz.

Die Faktoren `posX` und `posY` geben dabei das Bewegungsänderungsverhältnisses des Bootes hinsichtlich der X- und Y-Richtung an. Dazu müssen die Faktoren mit dem `tempo` - also der Geschwindigkeit des Bootes multipliziert werden.

Beispiel:

$$\begin{array}{lll} \alpha = 0 & \Rightarrow \text{posX} = \cos(0) = 1 & \Rightarrow _x = 1 * \text{tempo} \\ & \Rightarrow \text{posY} = \sin(0) = 0 & \Rightarrow _y = 0 * \text{tempo} \end{array}$$

$$\begin{array}{lll} \alpha = 45 & \Rightarrow \text{posX} = \cos(45) = 0,7071 & \Rightarrow _x = 0,7071 * \text{tempo} \\ & \Rightarrow \text{posY} = \sin(45) = 0,7071 & \Rightarrow _y = 0,7071 * \text{tempo} \end{array}$$

7. Optional kann nun ein Hintergrund, weitere Gestaltungselemente, Sounds und Hindernisse etc. eingefügt werden.

Das nächste ActionScript-Tutorial (von Jochen) befasst sich u.a. mit der Reflexion und der Kollision von Objekten. Beides sind Punkte, mit denen sich die Bootssteuerung dieses Tutorials zu einem durchaus interessanten Spiel weiterentwickeln lassen.

⇒ Im folgenden Kapitel sind weitergehende Informationen zur objektorientierten Programmierung aufgeführt.

3 Erläuterungen zur objektorientierten Programmierung

3.1 Objektorientierte Skripts

In objektorientierten Skripten werden Informationen in Gruppen, den so genannten Klassen, strukturiert angeordnet. Sie können mehrere Instanzen einer Klasse, so genannte Objekte, erzeugen und sie in Ihren Skripten verwenden. Sie können die vordefinierten Klassen von ActionScript verwenden und eigene Klassen erstellen.

Beim Erstellen einer Klasse definieren Sie alle Eigenschaften (die Merkmale) und Methoden (das Verhalten) für alle von der Klasse erzeugten Objekte genau so, wie Objekte in der Realität definiert werden. Eine Person verfügt z.B. über Eigenschaften wie Geschlecht, Größe und Haarfarbe und Methoden wie Sprechen, Gehen und Werfen. In diesem Beispiel ist "Person" eine Klasse und jede Einzelperson ein Objekt oder eine Instanz dieser Klasse.

Objekte in ActionScript können Daten enthalten oder auf der Bühne grafisch als Filmsequenzen dargestellt werden. Alle Filmsequenzen sind Instanzen der vordefinierten Klasse MovieClip. Jede Filmsequenzinstanz enthält alle Eigenschaften (z.B. `_height`, `_rotation`, `_totalframes`) und alle Methoden (z.B. `gotoAndPlay`, `loadMovie`, `startDrag`) der Klasse MovieClip.

Zum Definieren einer Klasse erstellen Sie eine so genannte Konstruktorfunktion. Für vordefinierte Klassen sind die Konstrukturfunktionen bereits definiert. Wenn Sie z.B. Informationen über einen Fahrradfahrer (Biker) in Ihrem Film benötigen, können Sie eine Konstrukturfunktion `Biker` mit den Eigenschaften `time` und `distance` und der Methode `rate` erstellen, mit der Sie die Geschwindigkeit des Fahrradfahrers ermitteln können:

```
function Biker(t, d) {  
    this.time = t;  
    this.distance = d;  
}
```

```
}  
function Speed() {  
    return this.time / this.distance;  
}  
Biker.prototype.rate = Speed;
```

Anschließend können Sie Kopien, d. h. Instanzen der Klasse erstellen. Mit dem folgenden Code können Sie für das Objekt Biker die Instanzen emma und hamish erstellen.

```
emma = new Biker(30, 5);  
hamish = new Biker(40, 5)
```

Instanzen können auch miteinander kommunizieren. Für das Objekt Biker können Sie eine Methode shove erstellen, mit der ein Fahrradfahrer (Biker) einen anderen wegstößt. (Die Instanz emma kann die Methode shove aufrufen, falls hamish zu nahe kommt.) Für das Übergeben von Informationen an Methoden werden Parameter (Argumente) verwendet: Beispielsweise kann die Methode shove die Parameter who und howFar verwenden. In diesem Beispiel schiebt emma den Fahrer hamish um 10 Pixel weg:

```
emma.shove(hamish, 10);
```

In objektorientierten Skripts können Klassen die Eigenschaften und Methoden anderer Klassen übernehmen. Dies wird als Vererbung bezeichnet. Mit Hilfe von Vererbung können Sie die Eigenschaften und Methoden einer Klasse erweitern oder neu definieren. Eine Klasse, die Eigenschaften oder Methoden von einer anderen Klasse erbt, wird als Unterklasse bezeichnet. Eine Klasse, von der Eigenschaften und Methoden an eine andere Klasse übergeben werden, wird als Oberklasse bezeichnet. Eine Klasse kann sowohl Unterklasse (subclass) als auch Oberklasse (superclass) sein.

3.2 Verwendung von benutzerdefinierten Objekten

Sie können benutzerdefinierte Objekte erzeugen, um die Informationen in Ihren Skripts im Hinblick auf Speicherung und Zugriff durch die Definition von Eigenschaften und Methoden besser zu strukturieren. Nach der Erzeugung eines Master-Objektes ("Klasse") können sie Kopien (d. h. Instanzen) dieses Objektes in einem Film verwenden ("instantiieren"). Dadurch können Sie Code wiederverwenden und Speicherplatz in Dateien einsparen.

Ein Objekt ist ein komplexer Datentyp, der 0 oder mehr Eigenschaften besitzt. Jede Eigenschaft hat, ebenso wie eine Variable, einen Namen und einen Wert. Die Eigenschaften sind an das Objekt gebunden und enthalten Werte, die geändert und abgerufen werden können. Diese Werte können einen beliebigen Datentyp haben: Zeichenfolge, Zahl, Boolean, Objekt, Filmsequenz oder undefined. Die folgenden Eigenschaften haben die verschiedene Datentypen:

```
customer.name = "Jane Doe"  
customer.age = 30  
customer.member = true  
customer.account.currentRecord = 000609  
customer.mclInstanceName._visible = true
```

Eine Eigenschaft eines Objektes kann wieder ein Objekt sein. In Zeile 4 des vorigen Beispiels ist account eine Eigenschaft des Objektes customer und currentRecord eine Eigenschaft des Objektes account. Der Datentyp der Eigenschaft currentRecord ist Zahl.

3.3 Erstellung von Objekten

Zum Erstellen eines Objektes durch eine Konstruktor-Funktion wird der Operator new verwendet. Eine Konstruktor-Funktion trägt immer den Namen des Objektes, das sie erzeugt. Ein Konstruktor, der beispielsweise ein Objekt vom Typ

Account erzeugt, hieße Account. In der folgenden Anweisung wird ein neues Objekt durch die Funktion namens MyConstructorFunction erzeugt:

```
new MyConstructorFunction (argument1, argument2, ... argumentN);
```

Beim Aufrufen von MyConstructorFunction übergibt Flash das versteckte Argument this, das eine Referenz auf das von MyConstructorFunction zu erzeugende Objekt darstellt. Innerhalb eines Konstruktors ermöglicht this Bezüge auf das Objekt, das vom Konstruktor erzeugt wird. Das folgende Beispiel zeigt eine Konstruktor-Funktion, die einen Kreis erzeugt:

```
function Circle(radius) {  
    this.radius = radius;  
    this.area = Math.PI * radius * radius;  
}
```

Konstruktor-Funktionen werden normalerweise zum Einsetzen von Methoden des Objektes verwendet.

```
function Area() {  
    this.circleArea = MATH.PI * radius * radius;  
}
```

Um ein Objekt in einem Skript verwenden zu können, müssen Sie ihm einen Namen zuweisen. Sie erzeugen ein neues Objekt vom Typ Kreis mit Radius 5, in dem Sie das Objekt mit dem Operator new erzeugen und es der lokalen Variablen myCircle zuweisen:

```
var myCircle = new Circle(5);
```

Hinweis: Objekte haben denselben Gültigkeitsbereich wie die Variable, der sie zugewiesen werden. Siehe Festlegen des Gültigkeitsbereichs einer Variablen.

3.4 Vererbung

Jede Funktion besitzt eine Eigenschaft namens prototype, die automatisch bei der Definition der Funktion erzeugt wird. Beim Erzeugen eines neuen Objektes durch eine Konstruktor-Funktion werden sämtliche Eigenschaften und Metho-

den der Eigenschaft `prototype` des Konstruktors zu Eigenschaften und Methoden der Eigenschaft `__proto__` des neuen Objektes. Die Eigenschaft `prototype` beinhaltet die Standard-Eigenschaftenwerte für Objekte, die mit dieser Funktion erzeugt werden. Die Übergabe von Werten zwischen den Eigenschaften `__proto__` und `prototype` wird als Vererbung bezeichnet.

Die Vererbung erfolgt nach einer festgelegten Hierarchie. Wenn Sie eine Eigenschaft oder eine Methode eines Objektes aufrufen, prüft ActionScript, ob ein derartiges Element vorhanden ist. Wenn es nicht vorhanden ist, sucht ActionScript in der Eigenschaft `__proto__` des Objektes nach dieser Information (`object.__proto__`). Wenn die Eigenschaft, auf die zugegriffen werden soll, keine Eigenschaft des `__proto__`-Objektes des Objektes ist, sucht ActionScript in `object.__proto__.__proto__`.

Es ist üblich, Methoden an ein Objekt durch Zuweisung der Methoden an die Eigenschaft `prototype` des Objektes zu binden. Die folgenden Schritte beschreiben die Definition einer Beispielmethode:

- 1 Die Konstruktor-Funktion `Circle` wird wie folgt definiert:

```
function Circle(radius) {  
    this.radius = radius  
}
```

- 2 Dann wird die Methode `area` des Objektes `Circle` definiert. Die Methode `area` berechnet den Flächeninhalt des Kreises. Im folgenden Beispiel wird die Methode `area` durch ein Funktionsliteral definiert und die Eigenschaft `area` des `prototype`-Objektes von `Circle` gesetzt:

```
Circle.prototype.area = function () {  
    return Math.PI * this.radius * this.radius  
}
```

- 3 Dann wird eine Instanz des Objektes `Circle` erzeugt:

```
var myCircle = new Circle(4);
```

- 4 Dann wird die Methode `area` des neuen Objektes `myCircle` aufgerufen:

```
var myCircleArea = myCircle.area()
```

ActionScript sucht im Objekt `myCircle` nach der Methode `area`. Da das Objekt keine Methode namens `area` besitzt, wird sein prototype-Objekt `Circle.prototype` nach der Methode `area` durchsucht. ActionScript findet sie und ruft sie auf.

Eine Methode kann auch an ein Objekt gebunden werden, indem die Methode an jede einzelne Instanz des Objektes gebunden wird. Ein Beispiel:

```
function Circle(radius) {  
    this.radius = radius  
    this.area = function() {  
        return Math.PI * this.radius * this.radius  
    }  
}
```

Von dieser Technik ist abzuraten. Die Verwendung des prototype-Objektes ist effizienter, da nur eine Definition von `area` erforderlich ist. Diese Definition wird automatisch in alle von der Funktion `Circle` erzeugte Objekte kopiert. Die Eigenschaft `prototype` wird von Flash Player ab Version 5 unterstützt.