Thomas Wagner

dpi

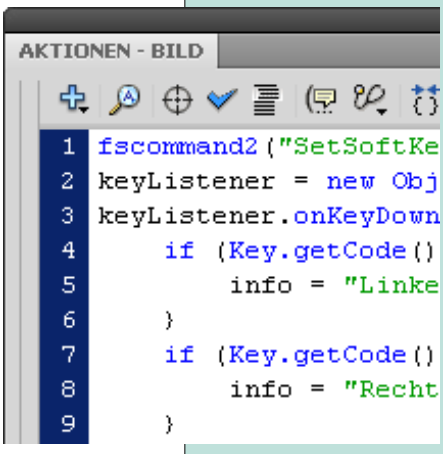
MOBILE GAMES MIT FLASH

Actionspiele entwickeln für Handy und Web mit Flash Lite

ADDISON-WESLEY

3

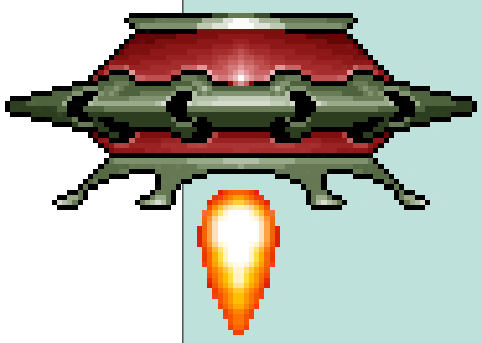
Tastensteuerung – Ein Handy ist keine Playstation



In Bezug auf Interfacedesign, und das meint auch und vor allem im haptischen, also fühl- und ertastbaren Bereich, könnte der „Idealfall“, der Controller einer Spielkonsole, und unsere Ausgangssituation, ein Mobiltelefon, nicht unterschiedlicher sein.

Auf der einen Seite das auf das Spielerlebnis optimierte, robuste Eingabegerät, das ergonomisch in der Hand liegt und nicht zuletzt (wobei das der wesentlichste Unterschied ist) einen standardisierten Satz an Tasten und Knöpfen anbietet, den die Entwickler (wie überhaupt die gesamte Hardware) mit Gewissheit voraussetzen konnten. Auf der anderen Seite ein kaum überschaubarer Pool an zum Teil völlig unterschiedlichen Tastaturkonzepten, Tastengrößen und -Anordnungen – ganz zu schweigen von den ganz und gar nicht standardisierten Handybetriebsystemen, die die Abfrage eben dieser Tasten überhaupt erst ermöglichen und vielfach auch stark limitieren.

Kurz gesagt: Konzepte für eine intuitive und gut funktionierende Bedienung sind bei Handyspielen längst nicht so nahe liegend wie es bei herkömmlicher Spielentwicklung der Fall wäre. Grund genug, diesem Thema ein eigenes Kapitel zu widmen.



3.1 Bestandsaufnahme

Zunächst sollte man bedenken, dass eine Handytastatur hauptsächlich für zwei Dinge konzipiert ist: Eine Nummer zu wählen oder eine SMS zu verfassen. Hierfür reicht der Ziffernblock an und für sich aus – nicht aber, um etwa eine Spielfigur durch ein einigermaßen intuitiv steuerbares Actionspiel zu lenken. Allerdings besitzt jedes aktuelle Handymodell auch ein mehr oder weniger einem Joypad bzw. Joystick nachempfundenes Tastenelement, mit dem der Benutzer durch die Menüs des Handybetriebssystem navigieren kann. In diesem Element findet man als Entwickler sicherlich die beste Eingabemöglichkeit für die meisten Spielkonzepte. Im Folgenden werde ich dieses Joypad (für das sich bislang noch nicht wirklich eine einheitliche Bezeichnung durchgesetzt hat), um Verwechslungen zu vermeiden, schlicht „Steuerkreuz“ nennen.

Im Zentrum dieser Richtungstasten liegt für gewöhnlich die Eingabetaste, mit der beispielsweise der jeweils markierte Menüpunkt aktiviert werden kann. Eine außerordentlich wichtige Taste, zumal sie mit Ihrer Position im Steuerkreuz sehr schnell und unmittelbar erreicht werden kann.

Außerdem liegen links und rechts von diesen Richtungstasten für gewöhnlich die so genannten „Softkeys“. Das sind kontextbasierte Tasten, die je nach Anwendung mit individuellen Funktionen belegt werden können. Was diese Funktionen jeweils genau bewirken, wird dabei meist durch entsprechende Texte verdeutlicht, die über den beiden Tasten im Display angezeigt werden (meist das Aktivieren des gerade markierten Menüpunktes auf der linken und das Abbrechen des aktuellen Vorganges auf der rechten Taste). Aus diesem Grund liegen diese Tasten auch stets direkt angrenzend unterhalb des Handydisplays, links und rechts von dem Steuerkreuz.

ABBILDUNG 3.1

Ein generisches Handy mit linkem und rechtem Softkey (1, 2), dem Steuerkreuz (3) und der Eingabetaste (4)

Gleich vorweg: Die allermeisten Steuerkonzepte lassen sich vollständig mit Steuerkreuz und Eingabetaste realisieren. Tasten des Ziffernblocks nutzen an und für sich nur sehr unkonventionelle Spielprinzipien.

3.2 Immer nur eine Taste gleichzeitig

Nachdem wir uns oben mit den Gegebenheiten der Handytastatur befasst haben, sollten wir nun der Vollständigkeit halber auch noch kurz den anderen Teil der Benutzerschnittstelle betrachten – nämlich den Benutzer selbst. Denn der ist es nicht nur gewohnt, das Mobiltelefon mit nur einer Hand zu bedienen, nein, er erwartet, dass es auch mit einem einzigen Finger (und zwar dem Daumen) zu bedienen ist.

Schon dieser Umstand alleine wäre Grund genug, dem User niemals mehr als einen Tastendruck auf einmal abzuverlangen. Allerdings gibt es noch einen weiteren recht triftigen Grund: Flash Lite kann – in keiner Version – mehr als eine Taste gleichzeitig abfragen. Das liegt sicherlich auch an der Systemarchitektur der meisten Handybetriebssysteme, ist aber nichtsdestotrotz unvermeidlich.

Dennoch ist das kein wirkliches Hindernis. Mehr als einen Finger einzusetzen wäre sowieso allenfalls durch den Einsatz der anderen Hand möglich, die dann etwa das Gerät am entgegengesetzten unteren Ende halten müsste, um mit dem anderen Daumen beispielsweise die Stern- oder Raute-Taste zu bedienen. Aber selbst bei so einer Konstellation müsste man noch ganz andere Probleme bewältigen. Denn das kontinuierliche Gedrückthalten einer Taste, während dessen eine weitere Taste gedrückt wird, würde dadurch für einen kurzen Zeitraum komplett unterbrochen. Drückt man also Taste A, dann gleichzeitig Taste B und lässt diese wieder los, wobei man Taste A immer noch gedrückt hält, so wird nun kurzzeitig gar kein Tastendruck mehr erkannt – und zwar genau für die Aktivierungsdauer der Tastenwiederholrate des Betriebssystems.

All diese Überlegungen zusammengenommen, wird schnell klar, dass man grundsätzlich mit nur einer Taste gleichzeitig, erreichbar durch ein und denselben Finger, auskommen sollte. Dabei entsteht aber auch ein weiterer positiver Aspekt, nämlich eine gewisse „erzwungene“ Vereinfachung, die man für intuitive Konzepte nutzen sollte. Generell sollte man, wie schon in den ersten Kapiteln ausgeführt, solche Limitationen (und derer gibt es bei der Entwicklung für Mobiltelefone recht viele) immer auch als Chance zur Reduktion sehen.

Realistischere Tastenabfrage mit Schaltflächen

Die meisten Beispiele in diesem Buch nutzen Schaltflächenobjekte zur Tastenabfrage. Das geschieht zwar hauptsächlich, um Kompatibilität zu Flash Lite 1.1 aufrecht zu erhalten, es hat jedoch auch noch einen weiteren ganz wesentlichen Vorteil – selbst und gerade dann, wenn man für Flash Lite 2.x oder 3.x entwickelt.

Denn nur mit Einsatz einer Schaltfläche erhält man auch beim Testen im Desktop-Flashplayer das typische Tastaturmanagement eines „realen“ Flash Lite Players. Mittels Schaltflächen kann nämlich auch immer nur eine Taste gleichzeitig erkannt werden – jeder weitere Tastendruck löst das Signal der bisherigen Taste ab, auch wenn diese noch weiterhin gedrückt gehalten wird.

Setzt man hingegen die Funktionen der Key-Klasse ein, so wird der Flashfilm beim Einsatz in einem Desktop-Player den irreführenden Anschein erwecken, dass auch mehrere Tasten gleichzeitig erkannt werden. Eine diagonale Bewegung beispielsweise, die durch die Tastenkombination „nach links“ und „nach oben“ entsteht, könnte auf einem Flash Lite Gerät nicht reproduziert werden.

Doch was bedeutet das im alltäglichen Einsatz? Ein klassisches Arcade-Ballerspiel etwa, so sollte man meinen, erfordert ja eigentlich schon per Definition die Möglichkeit, beispielsweise ein Raumschiff zu steuern und gleichzeitig Schüsse abzugeben. Das wäre ja mit der vorliegenden Problematik nicht zu realisieren. Oder doch? Tatsache ist sicher, dass kein Handybenutzer wie wild eine einzelne Schusstaste bearbeiten möchte, um das übliche Dauerfeuer in so einem Spiel zu erzeugen. Selbst ein kontinuierliches Gedrückthalten der Schusstaste wäre bestimmt keine bequeme Lösung – und letztendlich auch gar nicht notwendig. Die Lösung in einem solchen Fall liegt also auf der Hand: Man ermöglicht dem Spieler automatisiertes Dauerfeuer. Das kann dann etwa durch eine einzelne Taste an- und wieder abgeschaltet werden, es wäre aber sicher auch absolut ausreichend, wenn es permanent angeschaltet bleibt.

Entwickelt man solche Lösungen, werden die betreffenden Spiele meist auch gleichzeitig weitaus einfacher zu bedienen und damit umso attraktiver für etwas unbedarftere „Casual Gamer", ohnehin die größte Zielgruppe im Handyspielemarkt.

3.3 Flash Lite 1.1: Impulse anstatt „Dauerdrücken"

Dass man ein Spiel mit immer nur einer Taste gleichzeitig steuern können muss, ist an und für sich schon eine, sagen wir, interessante Herausforderung für den Entwickler. Nun gibt es aber noch eine weitere Besonderheit, mit der man sich auseinander setzen muss; allerdings nur, sofern man zu allen Flash Lite Versionen kompatibel bleiben möchte (und damit zu Flash Lite 1.1): Das kontinuierliche Drücken einer Taste ist hier nämlich nicht wirklich erkennbar.

Um diese Problematik besser zu verstehen, sollte man zunächst einen Blick auf die Möglichkeiten der Tastaturabfrage in Flash Lite 1.1 respektive Flash 4 werfen. Die sind recht überschaubar: Hier gab es noch keine Key-Klasse, die Methoden zum Abfragen der Tastenbefehle bieten würde. Ein Tastendruck kann hier nur mittels einer gewöhnlichen Schaltfläche ermittelt werden, auch wenn es beim Handy gar keinen Mauscursor gibt, mit dem man sie anklicken könnte. Statt auf einen „onRelease"-Event zu warten, muss man hier eben einen „keyPress" Event abfangen:

```
on (keyPress "<Right>") {
  ship._x = ship._x+1;
}
```

Dieses Beispielscript würde nun etwa dafür sorgen, dass ein Movieclip mit Instanznamen „ship" sich nach rechts bewegt, sobald man die Nach-Rechts-Taste des Steuerkreuzes drückt. Das stimmt allerdings nicht ganz: Hält man hier kontinuierlich diese Taste gedrückt, so bewegt sich der Clip zuerst nur um einen einzigen Pixel und erst nach einer kurzen Pause annähernd kontinuierlich nach rechts.

Die Erklärung: Das Script wird immer dann ausgeführt, wenn der Key-Listener den Tastendruck nach rechts meldet. Und das geschieht hier, ganz ähnlich dem Effekt, wenn man in einem Textprogramm wie Word einfach eine Buchstabentaste gedrückt hält, in Abhängigkeit von der so genannten „Zeichenwiederholrate" (die können Sie beispielsweise im Windows-Betriebssystem in der Systemsteuerung unter „Tastatur" individuell einstellen). Der Listener schlägt also erst ganz kurz und nur ein einziges mal an, dann entsteht eine Pause – und erst danach erhält man ein schnell wiederkehrendes Signal.

Auch dieser Umstand klingt nun sicherlich zunächst mal wieder nach einer existenziellen Beeinträchtigung: Wie sollte man mit dieser Technik eine fließende, direkte Steuerung ermöglichen? Die Lösung besteht in einem generellen Konzept, das ich „Impulssteuerung“ nennen möchte.

Beispieldatei auf der CD: Tastensteuerung/Tastenwiederholrate.fla

Impulssteuerung oder auch impulsbasiertes Navigieren kennen wir alle – und zwar von der ganz gewöhnlichen Menüsteuerung in jedem beliebigen Handybetriebsystem. Denn hier geht man ganz selbstverständlich davon aus, dass für jede Einflussnahme eine Taste einmalig kurz gedrückt und gleich wieder losgelassen werden muss (bzw. immer wieder erneut gedrückt werden muss). So bewegen wir beispielsweise eine Markierung durch eine Playlist mit wiederkehrendem Tastendrücken, und so wählen wir auch Zeichen für eine SMS.

Tatsächlich ist es im alltäglichen Umgang sogar eine ziemlich ungewohnte Sache, eine Handytaste permanent gedrückt zu halten. Gut, aber selbst wenn man nun seine Navigation vollständig auf einzelne, isolierte Tastenkommandos einstellt, bleibt immer noch die Frage, wie damit die Steuerung eines sich unmittelbar und fließend bewegenden Objektes in einer Spielwelt gelöst werden sollte. Dazu gibt es prinzipiell zwei sinnvolle Ansätze:

a. Trägheitsbewegung und

b. Rasterausrichtung.

Diese beiden Konzepte haben nämlich eines gemein: Die Bewegung einer Spielfigur endet grundsätzlich nicht abrupt mit dem Loslassen der betreffenden Taste, sondern wird stets noch einen Moment fortgeführt. Und genau diesen Moment braucht unsere Tastensteuerung ja auch, bevor ein etwaiges anhaltendes Drücken der jeweiligen Taste erkannt werden könnte.

Ein Beispiel für trägheitsbasierte Impulssteuerung (die in diesem Fall auch Ideengeber für das gesamte Spiel war) findet sich etwa in „KC Carhop“, wo die Heldin des Spiels auf Rollschuhen unterwegs ist und durch Tastenkommandos lediglich Schwung holt oder die Richtung wechselt. Ein anderes Beispiel für Trägheitsteuerung ist der Klassiker „Lunar Lander“, der auch in einer vereinfachten Version im nächsten Abschnitt beschrieben wird. Und die wohl extremste Form von Trägheitsbewegung bietet kein geringerer als der weltberühmte „Pac Man“ – denn der denkt gar nicht erst daran abzubremsen, wenn man die Taste loslässt. Im Gegenteil, Pac Man läuft blindlings weiter, bis ihn die nächstbeste Wand aufhält. Ausschlaggebend ist also auch hier nur der erste, beliebig kurze Tastendruck.

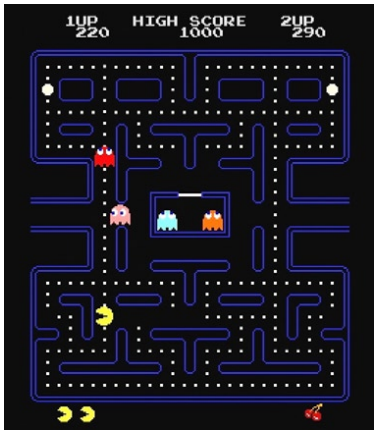


Abbildung 3.2

Stoppt erst, wenn er auf eine Wand trifft: Der gute alte „Pac Man“ benötigt keine kontinuierlichen Tastenkommandos. (© Namco 1980)

Beispiele für die zweite Methode, nämlich Rasterausrichtung in Spielen, gibt es hingegen noch weitaus mehr. Wer schon mal die Spielfigur in „Sokoban“ bewegt hat, kennt die typische Art der Fortbewegung, bei der die Spielfigur sich zwar stets fließend bewegt, aber immer nur in der Mitte eines Rasterfeldes stehen bleiben kann. Und auch hier ergibt die kurze Zeitspanne, die die Spielfigur auf ihrem Weg ins Zentrum des nächsten Rasterfeldes benötigt, die bewusste Verzögerung bis zum Erfassen eines anhaltenden Tastendrucks. Ein noch etwas älterer, ebenfalls sehr erfolgreicher Vertreter streng rasterbasierter Fortbewegung ist „Boulder Dash“, wo selbst die pseudophysikalisch umherfallenden Felsbrocken strikt von Rasterfeld zu Rasterfeld wandern. Aber der (zumindest in der Handywelt) sicherlich bekannteste Vertreter rastergebundener Bewegung ist das gute alte „Snake“. Es ist auch kein Wunder, dass ausgerechnet dieser Klassiker (der eigentlich auf ein Konzept aus dem Film „TRON“ zurückgeht) gerade auf Handybildschirmen seine größten Erfolge und seine weiteste Verbreitung feiern durfte. Denn kaum ein anderes Spielprinzip bleibt in seiner Steuerungsmechanik näher an der gewohnten, auf einzelnen Tastenkommandos basierenden Bedienung, die einem ja schließlich auch in allen anderen Handymenüs begegnet.

Tückisch: Key-Listener arbeiten unabhängig von der Framerate

Eine weitere Besonderheit „schaltflächengebundener“ Tastaturabfrage ist, dass (wie im übrigen bei allen Scripten, die in Abhängigkeit zu einem Event-Listener stehen) hier die Ausführung des Codes nicht an die Bildrate des Flashfilms gebunden ist. Im Gegenteil, die eingestellte Wiederholrate des jeweiligen Betriebssystems gibt die Geschwindigkeit vor. Daher kann sich beispielsweise ein Raumschiffobjekt, das mittels Key-Listener direkt bewegt wird, trotz einer extrem niedrigen Filmbildrate von 5 Bildern pro Sekunde auch mit gut und gerne 20 Bildern pro Sekunde ziemlich fließend davon bewegen. Das kann Fluch und Segen zugleich sein – zumal die Tastenwiederholrate auf einem PC (oder auch in Device Central) sich frappierend von der auf einem echten Handy unterscheiden kann.

Daher ist es oft sinnvoller, die auszuführenden Scripte weiterhin in Keyframes der Zeitleiste zu setzen und lediglich Schaltflächen zum Abfragen der Tasten zu benutzen. Nur so kann Synchronität mit der gewünschten Bildrate, Planbarkeit bei der korrekten Ausführung von Code und – vor allem – ein verlässliches Balancing ermöglicht werden.

Unter dem Strich ist es also durchaus nicht schwierig, mit dieser „Limitation“ ein fesselndes Spielprinzip zu realisieren. Im Gegenteil, die allermeisten erfolgreichen Spiele basieren auf einer Mechanik, die sich sehr wohl mit einer der beiden oben beschriebenen Methoden uneingeschränkt steuern lassen.

3.4 Beispiele für Impulsbasierte Steuerung

Nach diesen recht theoretischen Beschreibungen für impulsbasierte Steuerungsansätze möchte ich nun zwei konkrete Beispiele näher erläutern, die ebenfalls auf der beiliegenden CD zu finden sind.

3.4.1 Trägheit in „Lunar Lander“

Das „Lunar Lander“-Prinzip eignet sich nicht nur, um durch die typische Bewegungsträgheit etwaige Unzulänglichkeiten in der Tastaturabfrage zu kaschieren, sondern bietet auch für sich genommen eine äußerst spannende Spielmechanik. Der Spieler steuert durch das Betätigen von „Düsen“ mittels Rückstoß eine Raumkapsel, die unter Einfluss von Schwerkraft auf einer Planetenoberfläche möglichst sanft gelandet werden soll.

Dieses Raumschiff benötigt prinzipiell 4 Variablen: Geschwindigkeit in x-Dimension (horizontal), Geschwindigkeit in y-Dimension, Beschleunigung durch Düsenkraft und die Erdbeschleunigung bzw. Erdanziehungskraft.

Zum grundsätzlichen Aufbau der fla-Datei: Legen Sie in der Hauptzeitleiste einen Movieclip an, der das Raumschiff darstellen soll. Zeichnen Sie dazu einfach eine schnelle Skizze und wandeln Sie diese über F8 in einen Movieclip um – ausgefeilte Grafiken können Sie ja auch später noch einpflegen. Geben Sie dem Clip den Instanznamen „player".

Gleich vorweg: Normalerweise würde ich dringend davon abraten, wesentliche Teile einer Spiele-Engine wie hier direkt in der Hauptzeitleiste unterzubringen. Das verbietet sich aus mehreren Gründen, von denen ich einige schon im letzten Kapitel angesprochen habe. Aber hier geht es ja nun erstmal nur darum, eine grundsätzliche Struktur oder Herangehensweise möglichst transparent und verständlich zu machen.

Um die oben genannten Variablen von Anfang an zu deklarieren, bevor mit ihnen gearbeitet werden kann (was wiederum notwendig ist, um kompatibel zu Flash Lite 2.0 zu bleiben), steht in der Hauptzeitleiste der folgende Code, womit die Werte gleich zu Anfang einmalig in dem Movieclip „player" angelegt werden. Obwohl diese Datei für Flash Lite 1.1 gedacht ist, können Sie, wie im letzten Kapitel beschrieben, in eingeschränktem Rahmen die Punktnotation verwenden – das übersetzt Flash automatisch beim Exportieren. Schreiben Sie dazu den folgenden Code in den ersten (und einzigen Keyframe) der Hauptzeitleiste:

```
//
// Ausgangswerte für Geschwindigkeit des Raumschiffes:
player.xspeed = 0;
player.yspeed = 0;
//
// Wert für Beschleunigung durch Düsen:
player.accel = 1.5;
//
// Wert für Erdanziehungskraft:
player.gforce = 0.5;
```

Legen Sie noch eine weitere Ebene an, um dort getrennt vom restlichen Actionscript in einem separaten Keyframe die Flash Lite spezifische Anweisung für Vollbildmodus unterzubringen (ansonsten würde ja beim Export für Desktop-Flashplayer dieses und auch das restliche Script ungültig):

```
fscommand2("fullscreen", true);
```

Erweitern Sie die nun Zeitleiste innerhalb des „player"-Clips auf zwei Frames Länge, damit das dort untergebrachte Script durch einen Loop zwischen den beiden Frames immer wieder aufs Neue ausgeführt werden kann. Das wiederkehrende Ausführen von AS-Code geht in Flash Lite 2.0 auch mittels des onEnterFrame-Listeners, wobei dann keine solchen 2-Frame-Loops mehr benötigt werden – um aber kompatibel zu Flash Lite 1.1 zu bleiben, muss diese altmodische Methode verwendet werden, die obendrein auch noch einige weitere praktische Nebeneffekte bereithält (etwa das Anhalten von Scripten durch den stop()-Befehl und deren Wiederanwerfen durch play()).

Fügen Sie jetzt den folgenden Code im ersten Frame des Raumschiff-Movieclips("player") ein. Zunächst wird die „Erdanziehungskraft" zur vertikalen Geschwindigkeit addiert:

```
yspeed = yspeed + gforce;
```

In so einem Fall können Sie der Einfachheit halber auch eine kürzere Schreibweise wählen, siehe Kasten. Obwohl ich die ausführliche Schreibweise recht gut lesbar finde, werde ich alle weiteren Zuweisungen entsprechend abgekürzt schreiben (`yspeed += gforce;`)

Jetzt soll das Objekt bewegt werden, sprich: Die x- und y-Geschwindigkeiten werden zu den Objektkoordinaten addiert.

```
_x += xspeed;
_y += yspeed;
```

Additionszuweisung und Co.

Die Additionszuweisung += gibt es eigentlich in Flash Lite 1.1 nicht. Auch hier greift beim Export die automatische Übersetzung, die aus _x += xspeed; die Anweisung _x = _x + xspeed; macht.

An sich finde ich die alte Schreibweise (_x = _x + xspeed;) auch lesbarer und verwende sie häufig, um Studierenden in Einsteigerkursen etwas leichter verständliche Scripte zu bieten.

Hier möchte ich jedoch durchweg stets die kürzeste Schreibweise wählen, um eine bessere Übersicht bei komplexeren Scripten zu ermöglichen. Daher verwende ich im Folgenden nur noch die Kurzzuweisungen, die es im Übrigen für alle Grundrechenarten gibt:

+=	Additionszuweisung	a = a+b wird zu a += b
-=	Subtraktionszuweisung	a = a-b wird zu a -= b
*=	Multiplikationszuweisung	a = a*b wird zu a *= b
/=	Divisionszuweisung	a = a/b wird zu a /= b

Möchte man einen Wert nur um 1 erhöhen oder reduzieren, so kann man auch die folgenden Kurzformen verwenden (auch die kennt Flash Lite 1.1 eigentlich nicht, dennoch werden sie beim Export bequem in gültiges Actionscript übersetzt):

++	Inkrement	a = a+1 wird zu a++
--	Dekrement	a = a-1 wird zu a--

Außerdem soll das Objekt auf der gegenüberliegenden Seite des Bildschirms erscheinen, wenn es ihn zur anderen Seite hin verlassen hat:

```
if (_x>260) {
   _x = -20;
}
if (_x<-20) {
   _x = 260;
}
```

Nicht zwingend nötig, aber ein netter Effekt: Das stürzende Objekt kann am Boden (der hier an der Position _y=300 gedacht ist) mit halbierter Kraft abprallen.

```
if (_y>300) {
    yspeed *= -0.5;
    _y = 300;
    //
    // Bei Bodenberührung horizontale Geschwindigkeit abbremsen
    xspeed *= 0.5;
}
```

Schreiben Sie nun noch in den zweiten Keyframe

```
gotoAndPlay(1);
```

damit der Playhead immer wieder in den ersten Frame zurückkehrt und das dortige Script somit permanent ausgeführt wird.

Bisher würde das alles allerdings nichts anderes bewirken, als dass unser Raumschiff kontinuierlich und unsteuerbar in y-Richtung nach unten beschleunigt wird. Die Komponente, um die es hier ja eigentlich geht, fehlt noch völlig: Das Abfragen von Tasten zur Steuerung der Düsen unseres Schiffs. Wie bereits oben angekündigt, wird dazu eine Schaltfläche verwendet, die ebenfalls in dem Raumschiff-Movieclip liegt. Diese Schaltfläche ist zwar unsichtbar, da sie nur in ihrem „Aktiv"-Frame eine Vektorfläche enthält, sollte aber dennoch zur Sicherheit etwas weiter außerhalb des Displaybereichs liegen, damit man sie nicht zufällig beim Testen auf einem PC durch eine Berührung mit dem Mauscursor „entdecken" kann.

Unsichtbare Schaltflächen

Eine „unsichtbare" Schaltfläche (oder auch ein Button) besitzt nur in ihrem Aktiv-Frame einen Flächenzug. Die restlichen Bilder bleiben leer. Dadurch wird sie in Flash transparent cyanfarben angezeigt (rechts), in der finalen swf-Datei hingegen gar nicht mehr – und kann als generisches Multitalent in vielen Situationen wiederverwendet werden.

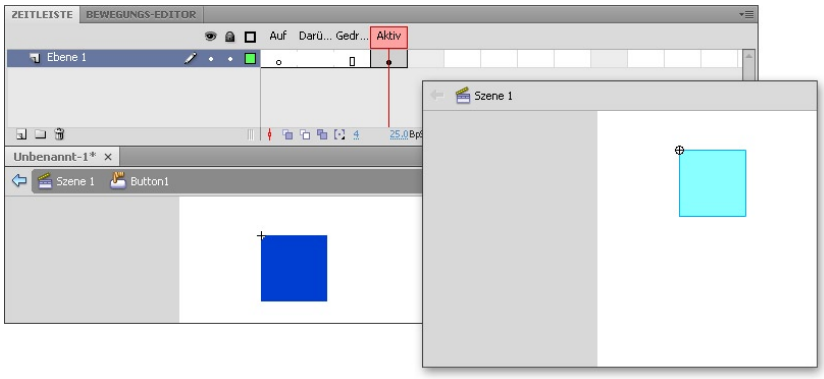


Abbildung 3.3

Eine „unsichtbare" Schaltfläche mit den internen Zustand-Keyframes (links) und ihrer Darstellung in der Entwicklungsumgebung (rechts).

Der Code, den man nun zur Bewegungserzeugung noch auf diese Schaltfläche schreiben muss, ist recht einfach:

```
on (keyPress "<Right>") {
    xspeed += accel;
}
```

Sobald das Steuerkreuz nach rechts gedrückt wird, erhöht sich die x-Geschwindigkeit um die „Düsenkraft" „accel". Das gleiche nun noch mit entsprechenden Listenern für „Left" und „Up", und schon hat man eine umfassende Steuerung für das kleine Schiff realisiert.

```
on (keyPress "<Left>") {
    xspeed -= accel;
}
on (keyPress "<Up>") {
    xspeed -= accel;
}
```

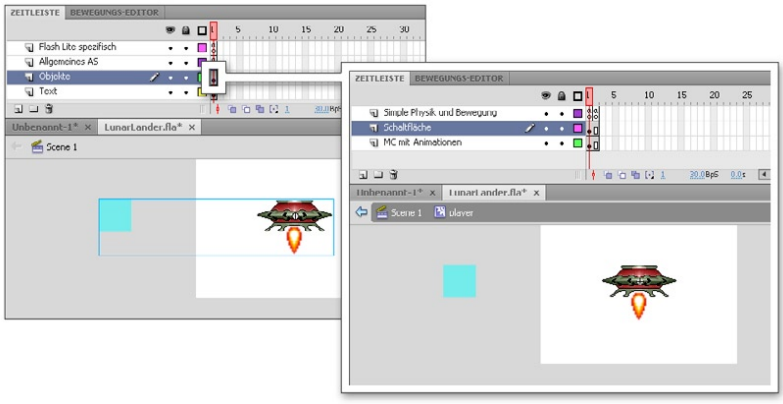


ABBILDUNG 3.4

Der „player"-Movieclip in der Hauptzeitleiste (links) und dessen interne Zeitleiste mit Actionscript-Loop und unsichtbarer Schaltfläche (rechts).

Dieses simple Beispiel zeigt, wie einfach man mit wenigen Zeilen an Code eine physikalisch glaubhafte, interaktive Bewegung erzeugen kann, die selbst mit Flash Lite 1.1 kompatiblen Mitteln durch den Benutzer intuitiv beeinflusst werden kann.

Düsenanimation

Möchten Sie Ihr Raumschiff noch ein wenig animieren, so wandeln Sie Ihre Skizze innerhalb des „player"-Clips in einen weiteren Movieclip um, geben Sie ihm den Instanznamen „animations" und legen Sie in dessen Zeitleiste drei separate Keyframes an mit den Bildmarkierungen „left", „right" und „up". Schreiben Sie in den ersten Frame dieser Zeitleiste stop();.

Bringen Sie nun in den markierten Keyframes jeweils noch eine kleine Düsen-Animation unter, die die Bewegung in die betreffende Richtung illustriert. Verwenden Sie dafür am besten einen kleinen Movieclip mit Instanznamen „booster", der eine kurz aufflackernde Düsenanimation beinhaltet, deren Zeitleiste in einem leeren Keyframe mit der Anweisung stop(); endet.

Erweitern Sie nun noch das Script auf der Steuerungs-Schaltfläche wie folgt:

```
on (keyPress "<Left>") {
    xspeed = xspeed-accel;
    gotoAndStop("animations:left");
    gotoAndPlay("animations/booster:1");
}
on (keyPress "<Right>") {
    xspeed = xspeed+accel;
    gotoAndStop("animations:right");
    gotoAndPlay("animations/booster:1");
}
on (keyPress "<Up>") {
    yspeed = yspeed-accel;
    gotoAndStop("animations:up");
    gotoAndPlay("animations/booster:1");
}
```

Und fertig ist die Comic-Flugmaschine!

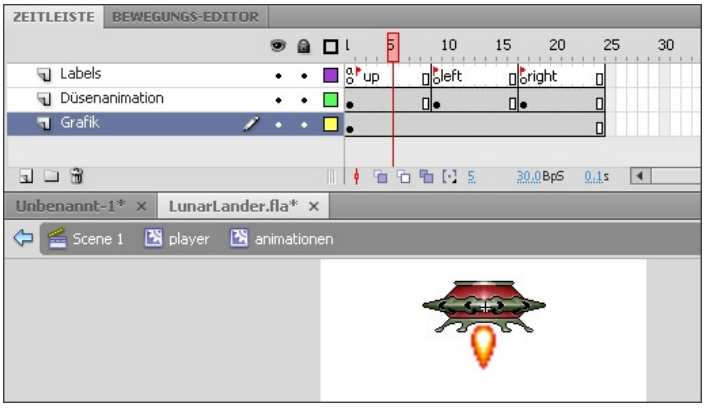


ABBILDUNG 3.5

In dem „player"-Clip liegt nun ein weiterer Movieclip „animationen", der eine Zeitleiste mit markierten Keyframes für die unterschiedlichen Platzierungen des Düseneffektes enthält.

3.4.2 Rasterbewegung in der Vogelperspektive

Das Umherbewegen einer Spielfigur auf einer ebenen Fläche, dargestellt aus der Vogelperspektive oder isometrisch in einer Isometrie (eigentlich präziser: „Axonometrie"), gehört zu den Klassikern der Computerspielinterfaces. Vom Urahn „Pac Man" über das neuere „Diablo" bis hin zu aktuellen Aufbaustrategiespielen – diesem Ansatz begegnet man sehr häufig.

Legen Sie für das folgende Beispiel erneut ein Movieclipsymbol „player" an (der Einfachheit halber wieder in der Hauptzeitleiste, auch wenn das strukturell nicht sehr sauber ist), außerdem noch

Beispieldatei auf der CD: Tastensteuerung/LunarLander.fla

ein weiteres Movieclipsymbol „helper“. Damit sind jeweils die Instanznamen der beiden Objekte gemeint – der Name in der Bibliothek ist nebensächlich, Hauptsache, Sie geben den beiden Objekten die richtigen Instanznamen.

Erstellen Sie in dem Movieclip „player“ noch einen weiteren Movieclip, der die unterschiedlichen Animationssequenzen der Figur beinhalten soll. Geben Sie ihm den Instanznamen „animations“. Legen Sie in diesem Clip eine längere Zeitleiste an, die Ihnen Raum gibt für vier Keyframes mit den Bildmarkierungen „up“, „left“, „down“, und „right“. Legen Sie in diesen Keyframes die entsprechenden Grafiken der Spielfigur an – wahlweise als Standbilder oder wiederum als Movieclips, in denen Sie animierte Sequenzen unterbringen können. Schreiben Sie hier außerdem in den ersten Frame stop(); um das Abspielen dieser Zeitleiste zu verhindern.

Erzeugen Sie in dem „player“-Movieclip nun noch eine Schaltfläche, optimalerweise wieder eine unsichtbare, die etwas weiter außerhalb von der Bühne platziert ist. Schreiben Sie auf diese Schaltfläche den folgenden Code:

```
on (keyPress "<Left>") {
   xspeed = -speed;
   gotoAndStop("animations:left");
}
on (keyPress "<Right>") {
   xspeed = speed;
   gotoAndStop("animations:right");
}
on (keyPress "<Up>") {
   yspeed = -speed;
   gotoAndStop("animations:up");
}
on (keyPress "<Down>") {
   yspeed = speed;
   gotoAndStop("animations:down");
}
```

Damit kann nun schon mal die Animation der Spielfigur gesteuert werden. Die Variable „speed“ wird gleich noch gesetzt.

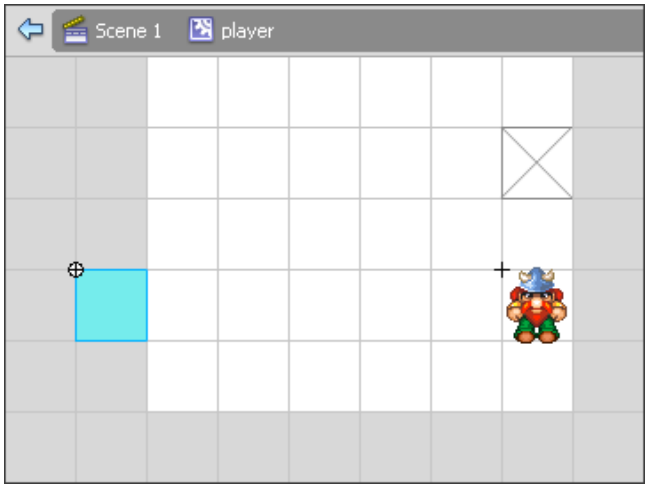


Abbildung 3.6

In dem player-Movieclip liegt eine Schaltfläche zur Tastenabfrage und ein weiterer Movieclip, der die Spielfigur-Animation beinhaltet. Abgegraut ist hier auch der „helper"-Movieclip zu sehen, der außerhalb von dem Player-Clip liegt und lediglich zum Veranschaulichen der jeweiligen Rasterposition da ist.

Erweitern Sie nun die Zeitleiste in diesem Movieclip „player" auf zwei Einzelbilder, so dass ein Flash Lite 1.1-typischer Loop für wiederkehrendes Ausführen von Actionscript zustande kommen kann – schreiben Sie in den zweiten Frame auch gleich: gotoAndPlay(1);

Zunächst müssen nun einige Variablen initialisiert werden, damit man mit ihnen Berechnungen durchführen kann – in Flash Lite 1.1 zwar ohnehin kein Problem, in Flash Lite 2.x aufwärts hingegen zwingend notwendig. Diese Anfangswerte sollen jedoch nur ein einziges Mal gesetzt werden – beim „Wachwerden" des Movieclips – danach aber bitte nicht mehr, da sie ja sonst permanent wieder auf ihren Ausgangswert zurückgesetzt würden.

Aus diesem Grund wurden die Ausgangswerte in dem Lunar-Lander-Beispiel „von außen", aus der Hauptzeitleiste, in den player-Movieclip gelegt (und damit nur ein einziges Mal gesetzt). Das bringt unseren player-Clip aber in eine gefährliche Abhängigkeit zum Vorhandensein eines fremden Scriptes, das er nicht „eingebaut" mit sich herumträgt.

Schreiben Sie also, um den Clip ein wenig autonomer zu machen, zuerst eine kleine so genannte „ini-Schleife". Die merkt sich mit Hilfe einer Variable „ini", dass das enthaltene Script schon einmal ausgeführt wurde und in allen folgenden Durchgängen gefälligst zu ignorieren ist:

```
if(!ini){
    ini=true;
    //
    // Ausgangswerte für Geschwindigkeiten:
    xspeed = 0;
    yspeed = 0;
    //
    // Die Standard-Geschwindigkeit:
    // (in diesem Beispiel immer ganzzahligen Teiler von 40 verwenden!)
    speed = 4;
```

```
        //
        // Spielfigur exakt in ein 40-Pixel-Raster setzen:
        _x = int(_x/40)*40;
        _y = int(_y/40)*40;
    }
```

Sobald dieses Script einmal ausgeführt wurde, hat `ini` den Wert „true“ (bzw. 1 in Flash Lite 1.1) und die Bedingung `!ini` (meint: „ini“ ist false, 0 oder undefined) wird nicht mehr erfüllt, das enthaltene Script also auch nicht mehr ausgeführt.

Nun geht's ans Eingemachte. Die Spielfigur soll nämlich nicht nur durch die entsprechenden Tastenkommandos loslaufen, sondern sie soll sich so lange weiterbewegen, bis sie sich in einem gedachten Grundraster im nächsten Rasterfeld befindet. Die x- und y-Position in diesem gedachten Grundraster kann man ganz einfach ermitteln, indem man die tatsächliche `_x`- und `_y`-Positionen durch die Rasterlänge dividiert und davon die Nachkommastellen abschneidet. Die x-Position 134 ergibt in einem 40-Pixel-Raster dann 134 : 40 = 3,35 – also das Feld 3 in der x-Achse (beginnend mit Feld 0). Um einfacher feststellen zu können, ob sich die Spielfigur genau auf diesem Rasterfeld befindet, multipliziert man es einfach wieder mit 40 und erhält die Rasterposition 3 x 40 = 120. Auf diese Weise kann man freie Fließkommawerte in gedachte 40-Pixel-Rasterwerte „einrasten“: 57 wird zu 40, 173,3 wird zu 160 und so weiter.

```
    //
    // Bewegung erzeugen:
    // Geschwindigkeiten zu Objektkoordinaten addieren
    _x += xspeed;
    _y += yspeed;
    //
    // Feststellen, in welchem Rasterfeld man sich befindet
    // (Kantenlänge eines Rasterfeldes sind hier 40 Pixel)
    grid_x = int(_x/40)*40;
    grid_y = int(_y/40)*40;
    //
    // Zur Verdeutlichung: Hilfsobjekt mitbewegen
    _parent.helper._x = grid_x;
    _parent.helper._y = grid_y;
    //
    // Sobald die Mitte des jeweiligen Rasterfeldes erreicht ist,
    // soll die Bewegung beendet werden
    if (_x == grid_x) {
        xspeed = 0;
    }
    if (_y == grid_y) {
        yspeed = 0;
    }
    //
    // Objekt am Bilschirmrand aufhalten
    if (_x > 200) {
        _x = 200;
    }
    if (_x < 0) {
```

```
        _x = 0;
    }
    if (_y > 280) {
        _y = 280;
    }
    if (_y < 0) {
        _y = 0;
    }
```

Native Movieclip-Eigenschaften haben ihre Tücken

Auch wenn es überschaubarer und vielleicht sogar sauberer wirkt: Es ist nicht ratsam, in einem Script immer nur direkt auf die nativen Eigenschaften eines Movieclips (wie etwa `_x`, `_y` oder `_rotation`) Einfluss zu nehmen.

In Flash können native Movieclip-Eigenschaften nicht jeden beliebigen Fließkommawert annehmen. `_x` und `_y` kennen beispielsweise nur Werte in 0,05-Schritten. Addiert man also zu `_x` = 10 noch 1,09 hinzu, so entsteht `_x` = 11,05 und nicht 11,09. Addiert oder subtrahiert man hingegen nur 0,04 oder weniger, so bleibt `_x` und `_y` gänzlich unverändert. Je nach Flashversion kann auch `_rotation` nur sehr begrenzte Werte annehmen. Diese nativen Eigenschaften eignen sich also nicht gut als Wertecontainer.

Um dieses Problem zu umgehen, sollte man stets gesonderte „Proxy-Variablen" nutzen, etwa, um die Position eines Objektes speichern. Diese Werte können dann beliebig verändert werden und sollten erst am Ende des Scriptes in eine „reale" Veränderung der nativen Movieclip-Eigenschaften umgesetzt werden.

Das Ergebnis: Die erzeugte Spielfigur bewegt sich bei jedem Tastendruck, egal, wie lange der andauert, immer mindestens um ein ganzes „Rasterfeld". Mit der voreingestellten Geschwindigkeit von speed = 4 braucht die Figur also 10 Frames, um von einem Rasterfeld ins nächste zu gelangen, wo sie erstmal wieder innehält. Falls der Spieler dann noch immer die betreffende Taste gedrückt hält, macht die Figur mit ihrer Bewegung einfach weiter – bzw. beginnt aufs Neue – ohne, dass ein störendes Abreißen des Tastensignals bemerkt werden könnte.

Und blendet man hier mal die raumgreifenden Kommentarzeilen aus, so wird deutlich, dass auch dieses anspruchsvolle Bewegungssystem noch immer mit einer recht geringen Menge an Code machbar ist. Das helper-Objekt ist dabei noch nicht mal notwendig – es dient lediglich dazu, die aktuelle Rasterposition anzuzeigen.

Beispieldatei auf der CD: Tastensteuerung/ Rasterbewegung.fla

Selten hilfreich, meistens störend, immer unschön: Das Focusrectangle

Möchte man zu allen Flash Lite Handymodellen kompatibel bleiben, sind Schaltflächen zur Erstellung einer Tastaturabfrage unumgänglich. Damit entsteht aber ein weiteres Problem, das Sie evtl. auch schon aus der Welt der „großen" Flashplayer kennen: Das sogenannte „Focusrectangle". Denn um einen Flashfilm auch für behinderte Personen oder auch auf PCs ohne Maus bedienbar zu halten, kann man genauso gut auch durch wiederholtes Drücken der ⇆-Taste nacheinander alle sichtbaren Schaltflächen anwählen (und durch die Eingabetaste auch aktivieren). Die jeweils angewählte Schaltfläche, die also gerade den Fokus besitzt, wird dabei durch ein gelbes Rechteck markiert. Eine unschöne Sache, besonders, wenn eine eigentlich unsichtbare Schaltfläche plötzlich mitten im Spielfeld sichtbar wird. Daher sollte man stets als erstes mit der Anweisung

```
_focusrect=false;
```

dafür sorgen, dass das Rechteck gar nicht erst angezeigt wird. Um aber ganz sicher zu gehen, sollte man die betreffende Schaltfläche möglichst weit außerhalb der Bühne platzieren (auch um ein Reagieren des Mauscursors beim Abspielen auf einem PC zu vermeiden).

3.5 Key-Controller: Bequeme Tastenabfrage in Flash Lite

Um eine einheitliche Struktur für das Erkennen der wichtigsten Tasten aufzubauen, sollte man sich einen gesonderten Tastatur-Movieclip anlegen, der außerhalb von der restlichen Spielmechanik in der Hauptzeitleiste des Flashfilms liegt und über verschiedene Variablen Signale liefert, ob und welche Taste gerade gedrückt wird.

Das ist sowohl für Flash Lite 2.x und 3.x mit Hilfe der Key-Klasse, als auch, mit ein paar Tricks, in Flash Lite 1.1 machbar.

Key-Controller in Flash Lite 2.x und 3.x

Sofern man sicher ausschließen kann, dass ein Projekt auch auf Flash Lite 1.1 lauffähig sein muss, kann man eine Tastaturabfrage auf Basis der Key-Klasse erstellen.

Der einzige, aber nicht unwesentliche Vorteil ist hierbei, dass die genaue Dauer eines Tastendrucks erkannt werden kann, was in Flash Lite 1.1 aufgrund der Schaltflächen-Abfrage ja leider nicht möglich ist.

Mehr als eine einzige Taste gleichzeitig kann allerdings in allen Flash Lite Versionen auch mit der Key-Klasse nicht erkannt werden.

Legen Sie einen Movieclip mit Symbolnamen „key_controls_FL2“ an und schreiben Sie in diesen Clip das folgende Script, um die wichtigsten Tasten abzufragen:

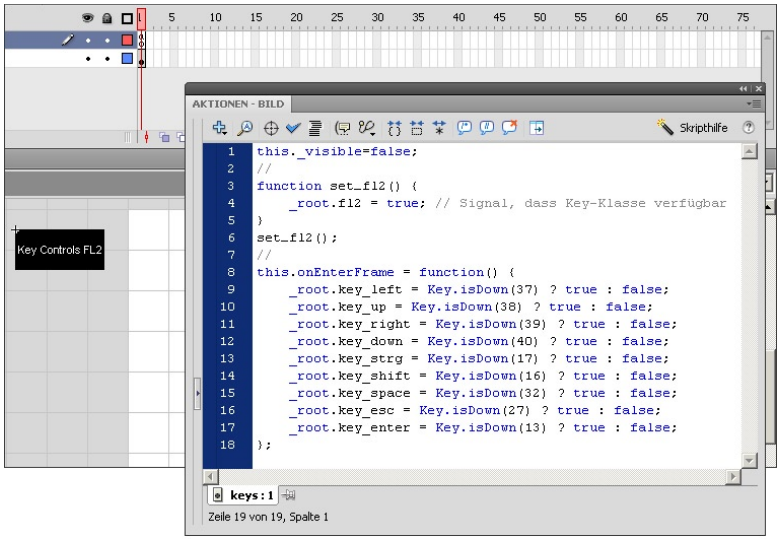


ABBILDUNG 3.7

Der Key-Controls-Movieclip enthält nur eine Platzhaltergrafik und ein statisches Textfeld, um leicht auffindbar zu sein. Dieses Objekt kann man nun bequem in jede beliebige Anwendung setzen, um mit den betreffenden Variablen arbeiten zu können. Die Key-Codes braucht man sich somit auch nicht mehr zu merken.

Die Tasten Strg, Space und Esc gibt es zwar nicht auf einem Handy, wohl aber auf einer PC-Tastatur. Und es kann nicht schaden, diese Tasten auch abzuprüfen, um bequemer am PC testen zu können und vor allem eine Anwendung auch für den Einsatz im Web vorzubereiten. Die Strg-Tasten liegen beispielsweise sehr bequem für „Aktions-Befehle“ auf der Tastatur, die Leertaste („Space“) wird gerne als schnell erreichbare Pausentaste verwendet.

Außerdem setzt dieser Clip mit der Variable _root.fl2 ein Signal, dass für Flash Lite 2 oder höher exportiert wurde und dadurch (unter anderem) mit der Key-Klasse und den Tastensignalen dieses Controller-Clips gearbeitet werden kann. Darauf werden noch einige der hier vorgestellten Routinen aufbauen.

Der Bedingung-Operator „?:“

Die Schreibweise mit dem Fragezeichen als Bedingung-Operator ersetzt eine langatmige if-Schleife, zumindest, solange es nur um eine bloße Zuweisung geht. Der folgende Doppelpunkt fungiert dabei quasi als „else“-Anweisung.

```
_root.key_left = Key.isDown(37) ? true : false;
```

Könnte man also auch so schreiben:

```
if(Key.isDown(37)){
   _root.key_left = true;
}else{
   _root.key_right = false;
}
```

Doch diese Schreibweise hätte das kurze Script deutlich aufgeblasen.

Vielleicht fragen Sie sich, warum ich hierfür nicht einfach einen Key-Listener einsetze. Ganz einfach: Ein großer Vorteil dieser Methode ist, dass die Tasten-Werte sich nur im Takt mit der Bildrate des Flashfilms ändern können. Es ist hier also nicht mehr möglich, dass ein Schaltflächen-Event „zwischen“ zwei Frames mitten in die laufenden Berechnungen grätschen könnte. Jedes Tastensignal bleibt nun für mindestens exakt einen Frame erhalten und beginnt und endet auch stets zu einem Bildwechsel. Das kann in einigen Fällen essenziell für eine komplexere Tastensteuerung sein. Andernfalls könnte es beispielsweise passieren, dass eine Figur bereits stehen geblieben ist, aber für ein weiteres Bild immer noch eine Laufanimation zeigt.

Legen Sie dieses Objekt ruhig in jede Ihrer Projektdateien. Auch wenn Sie für Flash Lite 1.1 exportieren, wird das nicht stören, da der Code in diesem Movieclip dann ganz einfach ignoriert wird.

Alle weiteren Flash Lite 2.x spezifischen Scripte in diesem Buch werden diesen Movieclip und die durch ihn erzeugten Variablen voraussetzen und auf ihnen aufbauen.

Beispieldatei auf der CD: Tastensteuerung/ key_controls_FL2.fla

Key-Controller in Flash Lite 1.1

Ein solches Objekt zum separaten Management der Tastaturabfrage lässt sich auch in Flash Lite 1.1 „simulieren“.

Legen Sie dazu ebenfalls wieder einen Movieclip an, der die Tastaturabfrage übernehmen soll – diesmal sinnfälligerweise mit dem Symbolnamen „key_controls_FL11“

Damit hier keine störende Pause zwischen dem ersten und dem anhaltenden Tastensignal entsteht, können Sie für jedes Tastensignal eine „Mindestlaufzeit“ festlegen. Erst, wenn die vorbei ist, wird

das Signal durch Loslassen der Taste abgebrochen. In dem folgenden Code wir diese „Mindest-Signaldauer“ mit der Variable „keytime“ gesteuert. Beachten Sie, dass diese Dauer auf einem realen Endgerät deutlich kürzer sein darf als auf einem Desktop-Flashplayer.

Erweitern Sie die Zeitleiste des Clips auf zwei Frames, schreiben Sie in den zweiten Keyframe wie immer gotoAndPlay(1) und in den ersten das folgende Script:

```
if (!ini) {
    ini = true;
    keytime = 10;
    u = 0;
    l = 0;
    r = 0;
    d = 0;
}
if(_root.fl2){
    stop;       // lahmlegen, falls FlashLite2-Tastenabfrage vorhanden
}else{
    _root.key_up = 0;
    _root.key_down = 0;
    _root.key_left = 0;
    _root.key_right = 0;
    _root.key_enter = 0;
    if (u>0) {
        _root.key_up = true;
        u--;
    }
   if (d>0) {
        _root.key_down = true;
        d--;
    }
    if (l>0) {
        _root.key_left = true;
        l--;
    }
    if (r>0) {
        _root.key_right = true;
        r--;
    }
}
```

Die Variablen `u`, `l`, `r` und `d` dienen als „Counter“, um die jeweiligen Signale länger aufrecht zu erhalten. Platzieren Sie nun in diesem Clip eine Schaltfläche und schreiben Sie darauf den folgenden Code:

```
on (keyPress "<Enter>") {
    if (e<1) {
        e = keytime;
    } else {
        if (e<2) {
            e = 1;
        }
    }
}
```

```
on (keyPress "<Up>") {
    d = 0;
    if (u<1) {
        u = keytime;
    } else {
        if (u<2) {
            u = 1;
        }
    }
}
on (keyPress "<Down>") {
    u = 0;
    if (d<1) {
        d = keytime;
    } else {
        if (d<2) {
            d = 1;
        }
    }
}
on (keyPress "<Left>") {
    r = 0;
    if (l<1) {
        l = keytime;
    } else {
        if (l<2) {
            l = 1;
        }
    }
}
on (keyPress "<Right>") {
    l = 0;
    if (r<1) {
        r = keytime;
    } else {
        if (r<2) {
            r = 1;
        }
    }
}
```

Nun wird jedes Tastensignal ab dem anfänglichen Tastendruck mindestens für 10 Frames aufrechterhalten – danach wird es jedoch unmittelbar abgebrochen, sobald die betreffende Taste tatsächlich nicht mehr gedrückt wird.

Beispieldatei auf der CD: Tastensteuerung/ key_controls_FL11.fla

3.6 Extrem wichtig, aber unbequem zu handhaben: Die „Softkeys“

So groß die Bedeutung der Softkeys auch sein mag, die Abfrage dieser so wesentlichen Tasten hat man bei Macromedia respektive Adobe nicht unbedingt sehr bequem – oder besser gesagt: aufwärtskompatibel – gestaltet.

Das Problem liegt dabei ausnahmsweise mal nicht auf der Flash Lite 1.1-Seite. Im Gegenteil, beim Export für Flash Lite 1.1 wird die Abfrage der Softkeys sogar zum regelrechten Kinderspiel. Wir erinnern uns: Ohne Key-Klasse muss man einfache Schaltflächen verwenden, um Tastatur-Events zu registrieren. Was bei der Steuerung eines schnellen Actionspiels durchaus eine gewisse Herausforderung darstellt, ist aber gerade für das Gestalten interaktiver Menüstrukturen wie geschaffen. Und die Softkeys werden für gewöhnlich ja ausschließlich in Menüstrukturen eingesetzt: „Select“ und „Cancel“ ist die klassische Funktionsbelegung, während man mit den Richtungstasten durch die Menüpunkte navigiert. Und in der Regel kann man auch jedes auf Schaltflächen basierende Flash Lite 1.1 Menü problemlos in einer Flash Lite 2 oder 3 Anwendung verwenden. Eigentlich nichts Besonderes – simple Aufwärtskompatibilität eben.

Das gilt jedoch nicht mehr, wenn es um die Softkeys geht. Denn: Anders als bei allen anderen Handytasten lassen sich die Softkeys bei Export für Flash Lite 2.x oder höher nicht über Schaltflächen abfragen. Hier geht das nur noch über die Key-Klasse. Und angesichts der Tatsache, dass man nicht Flash Lite 2.x-Code in einer Flash Lite 1.1 Datei unterbringen kann, ist das eine ziemlich hinderliche Sache. Das bedeutet nämlich, dass man ein Menü in Flash Lite 2 von Grund auf anders aufbauen müsste als bei der Verwendung in Flash Lite 1.1.

Doch der Reihe nach. Zuerst betrachten wir, wie man in Flash Lite 1.1 die Softkeys zugänglich macht. So trivial ist das nämlich auch wieder nicht, zumal es zu jeder Handytaste ein logisches Pendant auf der PC-Tastatur gibt – außer zu den Softkeys. Daher benötigt man zunächst den fscommand2 „SetSoftKeys“. Der sorgt dafür, dass jede künftige Softkey-Betätigung einen PageUp-Event (linke Softkey-Taste) oder einen PageDown-Event (rechte Softkey-Taste) auslöst. Diese Events können dann ganz bequem wie jede andere Taste abgefragt werden.

```
status = fscommand2("SetSoftKeys", "Beenden", "Abbrechen");
```

Und hier das Script für eine entsprechende Schaltfläche:

```
on (keyPress "<PageUp>") {
     fscommand2("quit");
}
on (keyPress "<PageDown>") {
     gotoAndStop("Hauptmenu");
}
```

Das zweite und dritte Argument dieser fscommand2-Anweisung dient dazu, die Texte, die über den Softkeys angezeigt werden, sofern der Flash Lite Player nicht im Fullscreen-Modus arbeitet, zu verändern. Da man aber ohnehin so schnell wie möglich nach dem Start einer Flash Lite Anwendung in den Fullscreen-Modus wechseln sollte, dürfte diese Funktionalität wahrscheinlich keine große Verwendung finden. Möchte man solche Texte über den Tasten darstellen, so bewerkstelligt man das besser direkt in dem Flashfilm, da man nur so völlige Gestaltungsfreiheit hat.

In Flash Lite 2 oder höher muss man hingegen die Key-Klasse bemühen, um die Softkeys abfragen zu können. Das sieht dann wie folgt aus:

```
status = fscommand2("SetSoftKeys", "soft1", "soft2");
keyListener = new Object();
keyListener.onKeyDown = function() {
      if (Key.getCode() == "soft1") {
            fscommand2("Quit");
      }
      if (Key.getCode() == "soft2") {
            gotoAndStop("Hauptmenu");
      }
};
Key.addListener(keyListener);
```

In diesem Fall ist der keyListener auch deswegen unabdingbar, da (wie schon ausgeführt) der fscommand2(„quit“) immer nur dann ausgeführt wird, wenn er in Verbindung mit einem Tastaturbefehl aufgerufen wurde.

Möchte man also einen Flashfilm dergestalt anlegen, dass man ihn sowohl für Flash Lite 1.1 als auch für Flash Lite 2.x exportieren kann, so muss man an jeder Stelle, wo die Softkeys abgefragt werden, ganz einfach immer beide Verfahren anwenden. Also überall eine entsprechende Schaltfläche anlegen, die die gewünschten Softkey-Scripte enthält, und im gleichen Frame das Script für den keyListener unterbringen. Dabei ist lediglich wichtig, dass das Script mit Flash Lite 2 Anweisungen in einem separaten Keyframe in einer gesonderten Ebene liegt – denn beim Export für Flash Lite 1.1 wird dieses Script wie gehabt als fehlerhaft angesehen und mit allen weiteren Anweisungen der betreffenden Bild-Aktionen ignoriert werden.

Auf diese Weise ergibt sich zwar für jede Abfrage die doppelte Menge an Script, man erhält jedoch die Möglichkeit, Menüstrukturen aufzubauen, die sowohl in Flash Lite 1.1- als auch in Flash Lite 2.0-Projekten verwendet werden können. Die Fehlermeldungen, die beim Export für Flash Lite 1.1 ausgegeben werden, muss man dabei ganz einfach ignorieren.

Beispieldatei auf der CD: Tastensteuerung/ Softkeys.fla