Auf dem Bildschirm erscheint eine fremde Welt: surrealistische Landschaften mit futuristischen Flugobjekten. Die Hatz geht über Schluchten, stets verfolgt und bombardiert von zahllosen Gegnern. Wer kann sich der Faszination derartiger Szenen entziehen, möchte nicht selbst so eine Fantasywelt erschaffen? Es geht.

Ein verregneter Samstagnachmittag – tote Hose. Doch zum Glück besitzen Sie ja Ihren C64. Sie legen also die Diskette mit Ihrem Lieblingsspiel ein; und nach einigen Sekunden fesselt Sie ein Spiel mit perfekter Grafik und tollem Sound.

Es ist lediglich die Kenntnis moderner Programmiertechniken nötig, um sowas selbst zu kreiern, und natürlich sollten Sie wissen, was mit Ihrem C64 machbar ist und was nicht.

Für unseren Kurs sollten Sie ein paar Kenntnisse in Assembler besitzen. Wir setzen aber kein zu hohes Niveau voraus. So genügt für noch unerfahrene Programmierer das Sonderheft 71 »Assembler« als Grundlektüre und Nachschlagewerk.

Bevor wir uns näher mit der Programmierung befassen, sehen wir uns einmal an, welche Arten von Computerspielen es gibt:

1. Adventures und Rollenspiele

... versetzt Sie in die Rolle einer anderen Person. Für diese haben Sie eine (oder mehrere) Aufgabe(n) zu lösen. Dazu wandern Sie durch Landschaften, suchen dabei in Irrgärten (Dungeons) nach versteckten Schätzen, kämpfen mit Monstern und lösen Rätsel. Eine Abwandlung sind sog. Textadventures. Bei ihnen findet ein Frage-/Antwortspiel statt.

Typische Vertreter von Adventures sind »Bard's Tale«, »Questron« oder »Detektiv 2000« in diesem Heft.

2. Geschicklichkeitsspiele

... testen und erweitern die Grenzen Ihres Reaktionsvermögens. Sie steuern ein Raumschiff mit dem Joystick durchs All oder lenken ein Fabelwesen durch finstere Gänge, wobei man Ihnen überall Hindernisse in den Weg legt, die Sie nicht mal streifen dürfen. »Mission II« in diesem Heft zeigt Ihnen deutlich was Sache ist.

3. Actionspiele

... sind meist auch zugleich Geschicklichkeitsspiele. Bei ihnen treten gehäuft Gegner auf, die Sie in einer festgesetzten Zeit vernichten müssen. Meistens lassen sie sich per Feuerknopf abschießen, und zwar in Serie. Daher nennt man diese Spieleart auch sehr ungnädig »Ballerspiele«. Die Vertreter dieser Spieleart sind primitiv bis durchaus anspruchsvoll. Beispiele für »Action« sind »Hyperthrust« oder »Flowers« in diesem Heft.

4. Jump and Run-Spiele

Bei ihnen muß der Computerspieler über Leitern oder Treppen unterschiedliche Ebenen überwinden, dabei Gegenstände durch Springen (Jump) oder Berühren aufsammeln. Das Ganze hört sich zwar einfach an, wird aber erschwert, weil bösartige Gnome die Gänge unsicher machen und jeder Kontakt mit ihnen tödlich für den Computerspieler ist. Leider sind diese Gegner nur in Ausnahmesituationen angreifbar, also hilft nichts anderes als ausreißen (and Run). Die Urversion dieser Spielephilosophie ist der berühmte »Pac Man«.

5. Strategiespiele

... läßt Sie meist durch überdimensionale Landschaften fahren oder wandern. Dabei gilt es einen oder mehrere Gegner zu vernichten. Im Gegensatz zum Adventure aber nicht durch Rätsel, sondern durch Steuerung des Kampfgeschehens. Das kann wie bei »Tron« in diesem Heft unkriegerisch, mit dem Joystick geschehen, oder wie bei »Field of Fire« in mehreren Kampfphasen. In der ersten Phase, der Bewegungsphase, bewegt der Spieler seine Truppen und der Computer die gegnerischen. Anschließend folgt die Feuerrunde, in der die zu vernichtenden Gegner ausgesucht werden. Danach werden die Ergebnisse dieser Aktionen berechnet und angezeigt.

6. Brettspiele

... simulieren herkömmlich mechanische Spiele auf dem Computer. Die Palette reicht dabei von »Monopoly« bis »Schach«.

7. Simulationen

... versuchen einen Ausschnitt der Wirklichkeit auf dem Computer darzustellen. Man unterscheidet dabei zwischen zwei grundlegenden Variationen:

1. Wirtschafts-Simulationen, wie z.B. »Omnibus« in diesem Heft. Sie entscheiden über Gedeih oder Verderb einer Firma.
2. Technische Simulationen. Bei diesen verändern Sie Konstruktionsmerkmale und beeinflussen so die Funktion des simulierten Geräts.

Wie sieht ein Spiel aus

Es gibt also Mischformen und meist lassen sich einzelne Spiele auch nicht eindeutig zuordnen. Wie auch immer – neue Ideen finden Anklang (wenn sie gut realisiert sind).Für unseren Kurs haben wir Geschicklichkeits- und Actionspiele ausgewählt, denn diese sind am eindrucksvollsten anzusehen und vereinigen die meisten Programmiermerkmale aller anderen Spielarten in sich. Überlegen wir uns, was wir für eine typische Spieleszene benötigen:

Grafik ... das Design der Spiele-Landschaft

handelnde Figuren ... auch ihr Design muß festgelegt werden

Geräusche

... bei einer außergewöhnlichen Aktion erhöht ein besonderes Geräusch die Spannung

Hintergrundmusik

... dient zur Unterhaltung und gehört aus dramaturgischen Gründen zu jedem Spiel.

Spielablauf

... ist das schwierigste am ganzen Spiel. Er übersetzt Ihre Idee in die Spielbarkeit. Die Grafik kann noch so formvollendet, die Musik noch so perfekt sein, erst der Ablauf entscheidet über die Brauchbarkeit des Spiels.

Packen wir's an:

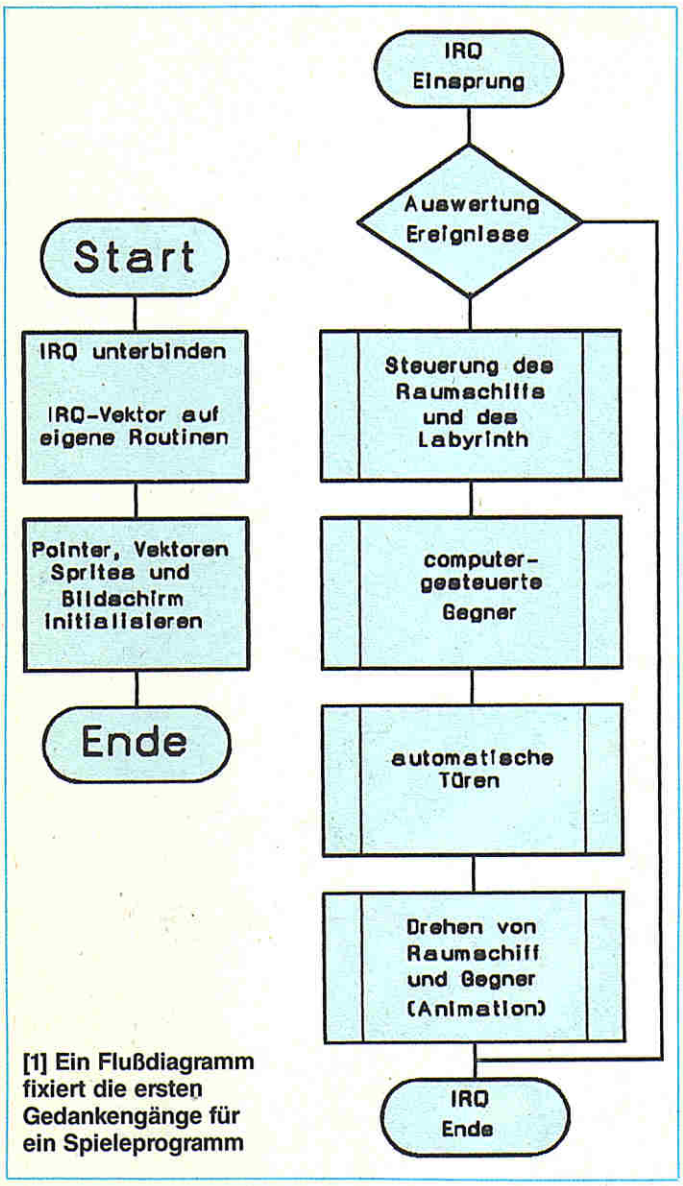
Wir werden ein Fantasieraumschiff durch ein Labyrinth steuern. Da dies zwar schön, aber zu einfach ist, lassen wir Hindernisse auftreten. Computergesteuerte Gegner und wie in einem Labyrinth üblich, noch etliche Hindernisse. Interessant sind auch Türen, die sich zyklisch öffnen und schließen und bei dieser Gelegenheit ist es vielleicht noch interessant, das Raumschiff ständig zu drehen.

Soweit zur Grundidee. Wir wollen sie in der ersten Phase schon mal festhalten – am besten in einem Ablaufplan (Abb.1, S. 16).

Danach ist es angebracht, einige Nächte drüber zu schlafen um sicherzugehen, daß alles Gewünschte mit eingeplant ist. Natürlich lassen sich kleine Verbesserungen auch noch am fertigen oder fast fertigen Spiel vornehmen, aber grund-

DIE SPIELHÖLLE NEBEN DER KÜCHE

KURS



[1] Ein Flußdiagramm fixiert die ersten Gedankengänge für ein Spieleprogramm

sätzliche Änderungungen sind nachträglich sehr schwer einzubauen.

Da der Ablaufplan nur eine grobe Hilfe ist, sollte der nächste Arbeitsschritt das Vorbereiten fertiger Bestandteile des Spiels sein. Dazu sollten Sie eine Diskette formatieren, auf die dann alle fertig erarbeiteten Teile gespeichert werden. Für unseren Kurs haben wir Ihnen alle Bestandteile schon in einem Programm zusammengefügt. Laden Sie es mit

```
LOAD"DEMO*",8
```

und starten Sie mit RUN. Das File wird anschließend entpackt und einzelne Programmteile, wie Grafik, Sprites, usw. werden an die richtigen Speicherstellen geschoben. Starten Sie aber noch nicht (SYS12288).

Damit Sie sowohl Zeichensatz als auch Sprites und Spiele-Landschaft umgestalten und in eigenen Spielen verwenden können, haben wir Ihnen einen Maschinensprache-Monitor mit auf Diskette gespeichert. Sie laden ihn mit

```
LOAD"SMON $C000",8,1
```

Danach geben Sie NEW ein. Gestartet wird der SMON mit SYS49152. Die Befehlsbeschreibung erfahren Sie aus dem Textkasten »Quickreferenz SMON« auf S. 24. Für unsere Arbeits-Diskette lassen sich die einzelnen Bestandteile des Spiels abspeichern (bis auf die Musik, sie liegt genau an der Position, wo sich jetzt der SMON befindet):

Für die Sprites geben Sie ein:

```
S"SPRITES"0800 0C00
```

Die Grafik wird mit

```
S"LANDSCHAFT"7000 8A00
```

auf Diskette gebracht. Schließlich benötigen Sie noch den geänderten Zeichensatz:

```
S"ZEICHENSATZ"2000 3000
```

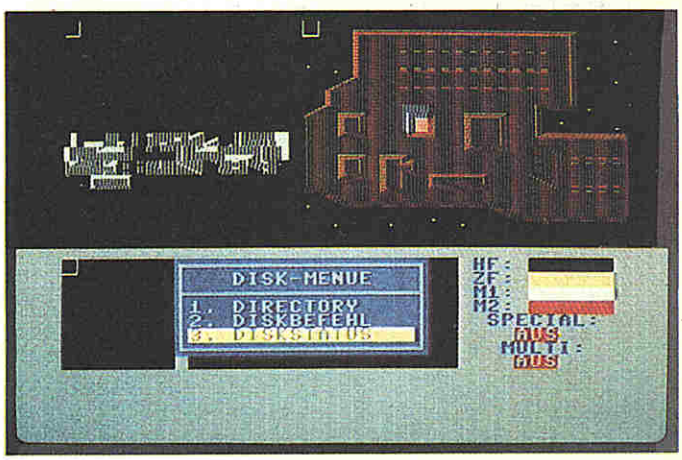
verewigt auch ihn auf der Arbeitsdiskette.

Die Grafik

Zunächst muß das Labyrinth aufgebaut werden. Das könnte man mit einer hochauflösenden Grafik realisieren, z.B. über den Umweg einer Zeichnung mit Hi-Eddi oder Koala-Painter. Diese Möglichkeit ist sicherlich nicht ideal, da eine Bitmap 8 KByte (Kilobyte, 8 x 1024 Byte) Speicher kostet. Für Spiele gibt es daher eine speicherplatzgünstigere Alternative: den normalen Zeichensatz-Modus des VIC (Videochip). Dieser benötigt nur 1 KByte für das Video-RAM.

Wenn Sie Spiele mit dem Standardzeichensatz des C64 aufbauen, bleiben sie leider nur relativ eintönig. Bedeutend mehr Möglichkeiten bieten eigene Zeichensätze. Sie kennen sicherlich die Fähigkeit des C64, umdefinierte Zeichensätze zu verwenden. Normalerweise holt sich der Video-Interface-Chip (VIC) seine Informationen aus einem Nur-Lese-Speicher mit unveränderlichem Dateninhalt (ROM). In diesem sind die im Handbuch beschriebenen Bildschirmcode-Zeichenmuster unverrückbar gespeichert. Aber der VIC ist ein sehr flexibler Baustein. Er kann seine Buchstabenmuster auch aus anderen Bereichen des Speichers holen. Man muß ihm nur sagen, wo die Daten liegen.

Ein Zeichensatz benötigt 2 KByte. Er alleine nützt uns allerdings nichts. Die Zusammenstellung der einzelnen Zeichen auf dem Bildschirm erst ergibt ein exaktes Bild des »Screens« (Bildschirms). Dafür ist das Video-RAM zuständig. Es liegt normalerweise ab Speicherstelle 1024 ($0400) im Speicher und enthält 1000 Zeichen. Jeweils 40 nebeneinander ergeben eine Bildschirmzeile und 25 Zeilen ergeben einen kompletten Bildschirm. Video-RAM und Zeichensatz benötigen



[2] Zeichensätze lassen sich komfortabel mit dem »Character-Editor« aus Sonderheft 55 gestalten

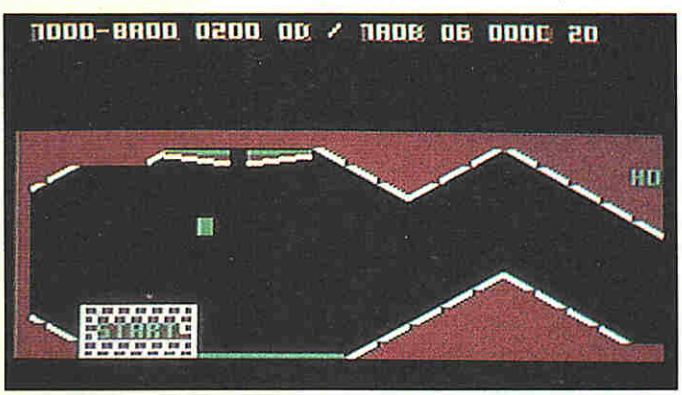
zusammen 3 KByte. Wie Sie sehen, wesentlich weniger als eine Bitmap.

Schaffen wir also zuerst die Voraussetzung für unsere Höhlenlandschaft - den sog. Spielezeichensatz. Kreiert wird er mit einem Zeichen-Editor (z.B. dem »Character-Editor« aus Sonderheft 55, Abb.2). Dieser Editor besitzt neben den Feldern zur Buchstabenumgestaltung einen Bereich, auf dem die Kombinationen der einzelnen Zeichen ausprobiert werden können. Dadurch läßt sich recht komfortabel ein neuer

Zeichensatz konstruieren und ausprobieren. Sie sollten ihn auf die Arbeitsdiskette speichern.

Wie konstruieren wir unsere Grafiklandschaft? Wir haben ja beschlossen, durch eine Höhlenlandschaft zu reisen. Dazu ist eine Bildschirmgröße alleine aber zu wenig; zu schnell würde man am Rand anstoßen. Wir benötigen also ein wesentlich längeres Labyrinth, als es der C64 auf einem Bildschirm darstellen kann. Später verschieben wir den sichtbaren Bereich kontinuierlich und erwecken so den Eindruck, als würden wir uns durch die Landschaft bewegen. Der Fachbegriff für diese Art der Hintergrundbewegung ist »scrolling«. Und noch etwas sollten wir schon bei der Konstruktion berücksichtigen:

Am oberen und unteren Bildschirmrand müssen ein paar Zeilen frei bleiben, für die Punkteanzeigen (score) usw. Für uns bedeutet dies: Wir konstruieren eine Landschaft, die mit dem eigentlichen Bildschirmformat nichts mehr zu tun hat. Achten Sie ebenfalls schon bei der Konstruktion darauf, daß bei den Zeichen, über die später unser Raumschiff fliegen soll, keine Pixel gesetzt sein dürfen. Wir überprüfen später im Programm, ob unsere Spielefigur mit dem Hintergrund zusammengestoßen ist. Daher dürfen die Gänge, durch die wir fahren wollen, auch keine Hintergrundinformationen enthalten. Im normalen Commodore-Zeichensatz ist dies das Leerzeichen mit dem Bildschirmcode »32« ($20). Um Ihnen die Konstruktion von Riesenbildschirmen zu erleichtern, befindet sich ein Editor mit auf der beiliegenden Diskette. Die Bedienungsanleitung finden Sie im Textkasten »der Grafikeditor«.



[3] Mit dem »Grafik Editor« ist die Gestaltung des Spielfeldes ein Kinderspiel. Sowohl Speicheranfang, als auch Größe lassen sich dem eigenen Spielfeld anpassen.

Die Spielfiguren

Als nächstes werden Helden und Fieslinge benötigt, also unser Raumschiff und die Gegner. Hierfür nutzen wir die Fähigkeit des C64, sog. Sprites darzustellen. Sprites sind eigene kleine Grafiken, die unabhängig vom anderen Bildschirminhalt dargestellt werden. Ihr Format ist auf 24 x 21 Punkte (einfarbig) oder 12 x 21 Punkte (mehrfarbig) festgelegt und es können (ohne Tricks) maximal acht davon zugleich erscheinen. Man bastelt sich diese Sprites mit einem Sprite-Editor (Sonderheft 33), oder klaut sie sich aus anderen Programmen und verändert sie nach eigenen Wünschen (»Sprite-Control«, Sonderheft 55).

Eines sollten wir noch berücksichtigen: Wir wollen unser Raumschiff ständig drehen. Dazu lassen wir später (wie im Film) mehrere Bilder nacheinander ablaufen (Animation). Geschieht dies in der richtigen Geschwindigkeit, wird dem Auge der Eindruck einer fließenden Bewegung vermittelt. Beim Konstruieren der Sprites müssen wir dies allerdings schon überlegen: Die Anzahl der nötigen Bilder muß festgelegt werden, und natürlich müssen die Sprites gezeichnet werden.

Zero-Page Adressen

$91/$92 - 145/146 - Bandoperationen
$9E bis $A2 - 158 bis 162 - Bandoperationen
$A7 bis $AE - 167 bis 171 - Band Ein-/Ausgabe
$FC bis $FF - 252 bis 255 - Vorsicht: findet meist bei Musikroutinen Verwendung

Der Grafikeditor

Er dient zum Erzeugen einer Grafiklandschaft für Scrollhintergründe und zeigt die Landschaft so, wie sie später im Spiel am Bildschirm sichtbar ist (Abb. 3). In unserem Kurs liegt der Scrollhintergrund im Speicher ab $7000 und hat eine Größe von 512 x 13 Zeichen. Dadurch ergibt sich eine Länge bis Speicherposition $8A00. Die Grafik ist linear abgelegt, d.h. die oberste Reihe der Grafik (512 Zeichen lang) ist ab $7000 bis $71FF abgelegt, die zweite Reihe von $7200 bis $73FF und so weiter. Für die Verwendung in eigenen Programmen wurden einige Zusatzfunktionen wie Ändern von Länge und Höhe der Grafik und des Grafikbeginns eingebaut. Beachten Sie, daß bei einer Formatänderung keine Umrechnung auf das neue Format stattfindet. Wenn Sie bei einer fertigen Grafik die Länge oder Höhe ändern, kommt daher auch das Aussehen der Landschaft durcheinander.

Zum Einüben der Funktionen gehen Sie folgendermaßen vor:
Laden Sie zuerst das komprimierte Spiel von der Seite 1 der beiliegenden Diskette

LOAD"DEMOGAME",8

und entpacken Sie es mit RUN. Danach laden Sie den Editor mit

LOAD"EDITOR",8,1

und geben »NEW« ein. Der Editor wird mit »SYS49152« gestartet.

Er beherrscht zwei Betriebsarten:

1. Editiermodus

... ist unmittelbar nach dem Starten eingestellt. Jedes eingegebene Zeichen erscheint an der Cursorposition. Der Cursor wird wie gewöhnlich mit den Cursortasten bewegt und bei Überschreitung des rechten oder linken Bildschirmrands scrollt der Inhalt entsprechend. Die erste Zeile ist eine Statuszeile und enthält von links nach rechts folgende Informationen in hexadezimaler Schreibweise:

7000-8A00 - Speicherbeginn und Speicherende der bearbeiteten Grafik.
0200 - Länge einer Grafikzeile
0D - Höhe der Grafik. Sie darf $01 bis $18 (= max. 24 Zeilen) betragen.
7000 - aktuelle Cursorposition
01 - aktuelle Cursorzeile
0001 - aktuelle Cursorspalte
62 - Zeichencode unter dem Cursor

Die letzten vier Werte verändern sich beim Bewegen des Cursors. Die Werte Speicherbeginn, Länge und Höhe können später im Befehlsmodus geändert werden. Das Speicherende wird dabei automatisch umgerechnet. Es findet keine Plausibilitätsüberprüfung statt: Das bedeutet, daß Sie eine Superlandschaft einstellen können, die alle Speichergrenzen überschreitet. Da dies natürlich nicht sinnvoll ist, sollten Sie bei Parameteränderungen auf das Speicherende achten. Ebenso können die RAM-Bereiche unter dem ROM zwar beschrieben, aber nicht sichtbar gemacht werden.

Bei Überschreitung der erlaubten Grafikhöhe führt das Programm ein RESTORE durch und muß mit SYS49152 neu gestartet werden. Achtung: Weder Grafikhöhe noch -Länge dürfen auf Null gesetzt sein.

Im Editor ist Groß- und Kleinschrift über die Tastatur erreichbar, alle anderen Zeichen über eine spezielle Option:

<CONTROL> und Zahl

... bietet die Möglichkeit über den Bildschirmcode ein Zeichen auf die Cursorposition zu setzen. Dazu drücken Sie <CONTROL> und halten diese Taste gedrückt. Bei gedrückter Taste tippen Sie über die Zifferntasten (<0> bis <9> und <A> bis <F>) zweistellig den gewünschten Bildschirmcode (hexadezimal) ein. Nach dem Loslassen von <CONTROL> erscheint das gewünschte Zeichen an der Cursorposition.

<CLR/HOME>

... bringt Sie immer an den Speicheranfang des Scrollhintergrunds

<F8>

... bietet eine Repeat-Funktion des letzten getippten Zeichens an der aktuellen Cursorposition (bei Programmstart mit $20). Da die Tastaturwiederholung eingeschaltet ist, lassen sich so auf einfachste Weise Linien oder Flächen erzeugen.

<SHIFT CLR/HOME>

... füllt die gesamte Bildschirmlandschaft mit dem letzten getippten Zeichen (bei Programmstart mit $20).

<F4>

... schaltet um zwischen Multicolor- oder Single-Farb-Modus.

<F6>

... schaltet um zwischen Commodore oder geändertem Zeichensatz. Der geänderte Zeichensatz muß ab $2000 (8192) beginnen. Eine Ladefunktion für Zeichensätze ist nicht vorhanden, Sie müssen den Zeichensatz also mit einem Maschinensprache-Monitor oder einem Basic-Lader an diese Speicherposition laden.

<F1>

... ändert die Zeichenfarbe. Jeder Tastendruck erhöht die Farbnummer (s. Handbuch). Beachten Sie: Im Multicolor-Modus dürfen für die Zeichenfarbe keine Farbnummern <8 verwendet werden, sonst schaltet das Video-Interface-Chip zurück auf Single-Farb-Modus.

<F3>

... ändert die Hintergrundfarbe (Register 53281).

<F5>

... ändert die Multicolorfarbe 1 (Register 53282).

<F7>

... ändert die Multicolorfarbe 2 (Register 53283).

<F2>

... verläßt das Programm nach Basic. Einen Neustart erreichen Sie mit SYS49152, dabei bleiben Speicherinhalt und Parameter-Einstellungen erhalten (lediglich VIC-Modus und Registerfarben sind nicht gespeichert).

<RUN/STOP>

... schalten um auf Befehlsmodus

2. Befehlsmodus

In diesem Modus lassen sich Funktionen wie Laden, Speichern, Directory und Parameteränderungen vornehmen.

<F1> - Laden

... lädt eine Grafiklandschaft nach Namenseingabe von Diskette.

<F2> - Speichern

... speichert eine Grafiklandschaft nach Namenseingabe auf Diskette. Die Parameter Speicherbeginn und Speicherende werden dabei als Anfang und Endadresse angenommen.

<F3> - Directory

... gibt das Inhaltsverzeichnis einer eingelegten Diskette am Bildschirm aus.

<F4> - Länge der Grafik

... legt die Länge (horizontal) der Grafiklandschaft in Zeichen fest. Voreingestellt sind 512 Zeichen ($0200).

<F6> - Höhe der Grafik

... legt die Höhe (vertikal) der Grafiklandschaft in Zeilen. Jede dieser Zeilen hat eine Länge, wie unter <F4> festgelegt. Voreingestellt sind »0D« Zeichen (dezimal 13). Erlaubt sind »01« bis »12« (dezimal 24) Zeilen.

<F8> - Speicherbeginn

... bestimmt den Anfang der Grafiklandschaft. Ab dieser Speicherposition wird die Grafik abgelegt. Voreingestellt ist $7000. Beachten Sie: $C000 bis $CFFF sind zwar möglich, aber nicht erlaubt, da sich hier das Editierprogramm selbst befindet. Der Editor liest keine Zeichen unter den ROMs. Falls Sie in Ihrem Programm diese Bereiche verwenden wollen, bietet es sich auch an, die Speicherlandschaft in einem anderen Speicherbereich zu kreieren, auf Diskette zu speichern und beim Spiel an die richtige Speicheradresse laden.

<RUN/STOP>

... schaltet zurück zum Editiermodus.

Kurzinfo: Editor

Programmart: Editor für Grafiklandschaften
Laden: LOAD"EDITOR",8,1
Starten: nach dem Laden NEW und SYS49152 eingeben
Benötigte Blocks: 12 Blocks
Programmautor: Herbert Großer

Die Soundroutine

Erst richtig eingesetzte Musik und Geräusche geben einem Spiel den richtigen Touch und überbrücken langweilige Passagen – alter Regietrick, siehe Autoverfolgungsjagden. Damit Sie das Rad nicht zweimal erfinden müssen, haben wir Ihnen dafür eine kurze Routine auf Diskette mitgeliefert. Geladen wird sie mit

LOAD"SOUND",8,1

Danach rückt »NEW« die Basic-Pointer wieder gerade. Initialisiert wird mit »SYS491252«. Die Routine bindet sich dabei automatisch in den Interrupt ein und Musik läuft im Hintergrund ab.

Doch für Spiele ist nicht nur Musik wichtig, sondern auch Geräusche. Beispielsweise sollte für ein schießendes Raumschiff auch ein entsprechendes Geräusch aus dem Lautsprecher tönen. Für unsere Musikroutine lassen sich bis zu vier Geräusche programmieren. Zwei sind in der Routine »Sound« schon vorbereitet. Geben Sie doch mal bei laufender Musik ein:

POKE49165,1

Ein Schuß dröhnt aus dem Lautsprecher und anschließend spielt die Musik weiter. Mit »POKE49165,2« erreichen Sie in unserem Demo eine Tonfolge (»Tirili«). Die anderen Effekte sind momentan nicht belegt.

Natürlich nützt die schönste Musik- und Soundroutine nichts, wenn man nicht auch eigene Musikstücke verwenden kann. Bei unserer Soundroutine ist das möglich. Zudem läßt sich in Maschinensprache mit Overlay-Technik (nachladen anderer Files) arbeiten, d.h. für andere Spiele-Level sind auch andere Musik- und Soundeffekte nachladbar. Als Programmierbeispiel befindet sich ein File mit auf der beiliegenden Diskette. Bevor Sie es laden, sollten Sie die Musikroutinen mit »SYS49152« ausschalten. Danach lädt

LOAD"MUSIK",8

das File. Sie starten mit RUN. Lassen Sie sich nicht durch den »OUT OF DATA ERROR ?« irritieren. Da die Musik beliebig erweitert oder geändert werden kann, wurde die Anzahl der Datazeilen nicht festgelegt. Bei der Abarbeitung gehen dem Programm also irgendwann die Data aus, und das wird natürlich mit einer Fehlermeldung quittiert. Keine Angst, das Musikstück ist trotzdem komplett im Speicher. Bevor Sie ans Komponieren gehen, sollten Sie sich das Listing einmal genauer ansehen (LIST):

Zeile 230 bis 260 enthalten die Daten von vier Sound-Datensätzen. Sie lassen sich beliebig jedem Ton zuordnen. Das Format entspricht dem einer Stimme:

1. Byte – Tastverhältnis, Low-Byte

... regelt zusammen mit dem vierten Byte das Tastverhältnis bei der Wellenform Rechteck. Bei anderen Wellenformen ist der Wert beliebig.

2. Byte – Tastverhältnis, High-Byte

... regelt zusammen mit dem dritten Byte das Tastverhältnis bei der Wellenform Rechteck. Auch dieser Wert ist bei anderen Wellenformen beliebig.

3. Byte – Wellenform

... ein gesetztes Bit hat folgende Funktion:

Bit 0 (1) = Ton Ein, muß Null sein.
Bit 1 (2) = Synchronisation
Bit 2 (4) = Ringmodulation
Bit 3 (5) = Test, muß Null sein
Bit 4 (16) = Dreieck
Bit 5 (32) = Sägezahn
Bit 5 (64) = Rechteck (s. 1.Byte und 2.Byte)
Bit 5 (128) = Rauschen

Die Wellenformen lassen untereinander mischen, lediglich »Rauschen« darf nicht mit anderen Formen zusammen erscheinen. Bit 0 (Ton Ein) wird von der Abspielroutine behandelt und darf nicht gesetzt sein, da sich der Ton sonst nicht mehr über die Sounddaten ausschalten läßt. Da Bit 3 (Testbit) einen Reset der Stimme durchführt und alle anderen Bits dabei ignoriert, darf es nicht gesetzt sein. Auch dieses Bit wird von der Routine behandelt.

4. Byte – Attack/Decay

... setzt die Anschlag-, bzw. Abschwellzeit des Tons. Attack (Anschlagzeit) benötigt dafür die unteren vier Bit (0 bis 15), Decay (Abschwellzeit) die oberen (0 bis 15 x 16).

5. Byte – Sustain/Release

... bestimmt Haltelautstärke bzw. Ausklingzeit des Tons. Sustain (Haltelautstärke) benötigt dafür die oberen vier Bits (0 bis 15 x 16), Release (Ausklingzeit) die unteren vier Bit (0 bis 15).

Achtung: Sounddaten müssen jeweils mit vier Datensätzen zu je 5Byte beginnen. Danach folgen Filterdaten (4 Byte). Erst dann beginnen Geräusche und Musik.

Zeile 270 enthält die Filterdaten:

1. Byte – Filterfrequenz, Low-Byte

... bestimmt zusammen mit dem zweiten Byte die Frequenz, wenn ein Filter mit Byte 3 eingeschaltet und sein Modus mit und Byte 4 festgelegt ist.

2. Byte – Filterfrequenz, High-Byte

... bestimmt zusammen mit dem ersten Byte die Filterfrequenz.

3. Byte – Resonanz/Filter an/aus

... legt die Resonanz der Filter (0 bis 15 x 16) fest und schaltet die einzelnen Filter ein (0 bis 15).

4. Byte – Filtermodus/Lautstärke

... setzt den Filtermodus (0 bis 15 x 16) und die Lautstärke aller drei Stimmen (0 bis 15).

Eine genauere Beschreibung der SID-Register-Funktionen finden Sie ab S. 35 im Sonderheft 53.

Ab Zeile 310 befindet sich die Daten für Geräusche uund Musik. Das erste Byte ist jeweils der ASCII-Code für die Note. Interpretiert werden:

<C> = C
<SHIFT C> = Cis
<D> = D
<SHIFT D> = Dis
<E> = E
<F> = F
<SHIFT F> = Fis
<G> = G
<SHIFT G> = Gis
<A> = A
<SHIFT A> = Ais
<H> = H

Im zweiten Byte werden Note an (1) oder aus (0), Datensatznummer (0 bis 3), Oktave (0 bis 6) und Nummer der Stimme (0 bis 3) miteinander verknüpft:

Note an/aus (Bit 7)
Datensatznummer (Bit 5 und Bit 6)
Oktave (Bit 2 bis Bit 4)
Stimmen-Nummer (Bit 0 und Bit 1)

Damit dieses zweite Byte nicht für jede Note von Hand berechnet werden muß, lassen sich die Daten nebeneinander eintragen. Beachten Sie dazu die »REM«-Zeile 290, sie zeigt die Reihenfolge der Daten in den DATA-Zeilen.

Die Töne werden innerhalb eines Interrups solange aufgerufen, bis anstelle der Note ein »W« (Wait) erscheint. Der Bytewert dahinter gibt an, wie viele Interrupt-Zyklen bis zum nächsten Ton gewartet wird (ein Zyklus entspricht 1/60 Sekunde).

Achtung: In einem »W«-Zyklus lassen sich zwar beliebig viele Noten einbauen, dieses Vorgehen ist aber nicht sinnvoll, weil jede Note eine gewisse Bearbeitungszeit benötigt. Sind zuviele Noten für einen Zyklus eingebaut, wird die zulässige Zeit für einen IRQ-Zyklus überschritten. Außerdem hören Sie bei einer Änderung einer Stimme innerhalb eines Zyklus nur den letzten (geänderten) Ton. Daher ist es sinnvoll, max. alle drei Stimmen (wie auch von Zeile 310 bis Zeile 330) mit Tönen zu belegen. Danach setzen Sie das »W« (Zeile 340).

Effektdaten müssen am Anfang der Daten stehen und mit »S« (Stop) abgeschlossen sein. Es sind max. vier Effekte erlaubt. In unserem Listing wurden zwei Effekte (Zeile 310 bis 350 und Zeile 370 bis Zeile 430) verwendet. Danach folgt die eigentliche Musik (ab Zeile 450):

Das Musikstück muß als letzten Datensatz »R« (Repeat) enthalten (Zeile 1320). Trifft die Abspielroutine auf dieses »R«, beginnt Sie das Stück wieder ab dem letzten »S«.

Die Abspielroutine erkennt beim Intialisieren automatisch, wie viele Effekte programmiert sind. Dadurch wird ein Overlay mit neuen Daten möglich. Wenn Sie aus Basic neue Daten nachladen, müssen Sie allerdings vorher mit »SYS49152« ausschalten. Danach laden Sie die neuen Sounddaten und initialisieren neu (»SYS49152«).

Bei eigenen Spielen ist es meist nicht sinnvoll, die Musik in einem eigenen Interrupt laufen zu lassen. Dazu sind Spiele durch ihre komplexe Interrupt-Programmierung zu zeitkritisch. Wir werden später noch hören warum. In so einem Fall ist es besser die Soundroutine zuerst zu initialisieren und als letzten Teil des eigenen IRQ's direkt anzuspringen. Dazu müssen Sie folgendermaßen vorgehen:

Initialisieren Sie aus der Routine, die für Ihr Spiel den IRQ verbiegt mit

```
JSR $C100 ; initialisiert die Tabelle Geräusch/Musik
JSR $C0CC ; setzt die internen Pointer richtig
```

Danach müssen Sie die Rücksprungadresse zu Ihrem Programm, oder zum alten Interrupt setzen:

```
LDA #<RUECKSPRUNG
LDX #>RUECKSPRUNG
STA $C003
STX $C004
```

wobei »RUECKSPRUNG« die von Ihnen gewünschte Adresse darstellt. Aus Ihrer Interruptschleife verwenden Sie dann die Routine mit

```
JMP $C0A5
```

Sie springt nach der Abarbeitung zur Speicherposition »RUECKSPRUNG«.

Effekte rufen Sie aus Ihrem Programm mit

```
LDA #EFFEKTNUMMER
STA $C00D
```

auf. Beim nächsten Aufruf der Routine wird der normale Sound unterbrochen, und der Effekt eingeleitet.

Zum Nachladen von Tondaten müssen Sie sicherstellen, daß die Routine während der Ladeoperation nicht angesprungen wird. Danach ist es wichtig mit

```
JSR $C100
JSR $C0CC
```

neu zu initialisieren.

Kurzinfo: Musik

Programmart: Beispielprogramm für einen Effekt- und Musikdatensatz
Laden: LOAD"MUSIK",8
Starten: RUN
Besonderheiten: Endet mit »OUT OF DATA ERROR ?«
Benötigte Blocks: 12 Blocks
Programmautor: Herbert Großer

Kurzinfo: Sound

Programmart: Abspielroutine für Effekte und Musik
Laden: LOAD"SOUND",8,1
Starten: nach dem Laden NEW und SYS49152 eingeben
Benötigte Blocks: 4 Blocks
Programmautor: Herbert Großer

Geräusche und Hintergrundmusik

Überlegen wir uns grundsätzlich, wie wir diese Musik realisieren können. Es läßt sich z.B. ein Musikstück mit einem Sound-Editor erzeugen und wir binden die erzeugte Musikroutine später in unser Programm ein. Dieser Weg wird auch von den meisten kommerziellen Programmen gegangen. Es muß dabei allerdings berücksichtigt werden, wo diese Routine, in welcher Länge plaziert ist und wie sie initialisiert und eingebunden werden muß. Daher Vorsicht vor unbekannten Sound-Editoren: Die Analyse der Soundroutinen bedeutet eine Menge Arbeit und viel Programmier-Erfahrung. Doch hier liefern wir Ihnen eine Universalroutine mit, die Sie editieren, verändern und in eigene Programme einbinden können. Beschreibung und Bedienung finden Sie im Textkasten »die Soundroutine« auf S. 22.

Programmiergrundlagen

Wir haben jetzt die meisten kreativen Vorbereitungen beendet und Zeichensatz- Grafik- und Sprites auf Diskette gespeichert. Nun ist es an der Zeit, sich zu überlegen, wie programmiert werden soll. Dafür gibt es zwei grundsätzliche Methoden:

1. Die »ich fange an und programmiere los« Methode und
2. »modulare« Programmierung.

Mit der ersten Methode kommt man zwar bei kleinen Projekten zu einem Ergebnis, es wird aber selten so aussehen, wie man es sich vorgestellt hat. Abgesehen davon ist der Zeitaufwand nicht abschätzbar und ein Fehler läßt sich fast nicht nachvollziehen, da sich einzelne Funktionsblöcke nicht abgrenzen, sondern quer durchs Programm schlängeln.

Die zweite und bessere Methode, »modulares Programmieren«, soll bei unserem Kurs zur Anwendung kommen. »Modular« bedeutet in Modulen, also mit räumlich abgegrenzten Programmteilen zu arbeiten. Sie werden später über ein Hauptmodul miteinander verbunden und haben gegenüber der »Schlauchprogrammierung« entscheidende Vorteile:

1. Es kann jede einzelne Funktion für sich getestet werden bis sie ohne Einschränkung funktioniert.
2. Nach und nach entsteht eine Bibliothek, die nach Belieben für einen anderen Programmzweck zusammengestellt werden kann - denn warum soll das Rad jedesmal neu erfunden werden.
3. Sollte sich herausstellen (z.B. bei zeitkritischen Anwendungen) daß ein Modul zu langsam ist, kann dieses geändert oder ausgetauscht werden, ohne das übrige Programm zu beeinflussen.

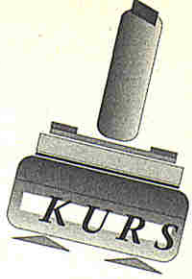
Für unser Spiel besitzen wir bereits ein Programm-Modul: Die Sound-Abspielroutine.

Überlegen wir uns anhand des Flußdiagramms (Abb. 1, Seite 16), welche Module wir weiterhin brauchen:

1. Wir steuern ein Fantasieraumschiff durch ein Labyrinth, natürlich mit dem Joystick. Also benötigen wir eine Routine, die unser Raumschiff steuert. Nun gibt es mehrere Möglichkeiten der Steuerung: Entweder wir bewegen das Sprite, oder den Hintergrund, oder beide. Das Sprite allein zu steuern, bringt nicht viel, da der Hintergrund größer als ein Bildschirm ist. Daher ist es besser, den Hintergrund zu bewegen (»scrollen«). Damit wir nach vorne und hinten ausweichen können, bewegen wir zusätzlich das Raumschiff-Sprite.
2. Dann lassen wir Gegner auftreten, sie müssen sich quasi automatisch bewegen.
3. Wir haben außerdem beschlossen, Türen einzubauen, die sich öffnen und schließen. Erreichen können wir dies durch Veränderung der Bitmuster des Zeichengenerators. Wenn man das Bitmuster eines Zeichens ändert, werden alle Zeichen des gleichen Codes am Bildschirm ebenfalls verändert.
4. Zudem haben wir beschlossen, unser Raumschiff ständig zu drehen. Dazu verwenden wir die »Animation«. Diesen optischen Trick kennen wir von Film und Fernsehen. Es werden nacheinander leicht veränderte Bilder gezeigt. Wenn dies schnell genug geschieht, entsteht der Eindruck einer fließenden Bewegung.
5. Ein wichtiges Modul fehlt uns noch:

Berührt unser Raumschiff einen Gegner oder den Hintergrund, muß dieses Ereignis festgehalten werden.

6. Schließlich und endlich soll alles miteinander zu einer Einheit verbunden, alle Module initialisiert, die einzelnen Ereignisse ausgewertet und darauf reagiert werden. Dieses sechste Modul stellt die eigentliche IRQ-Routine zur Verfügung, von der aus die anderen Module aufgerufen werden. Sie legt damit den Spielablauf fest.



Variablen

Im Gegensatz zur Programmierung in Basic verwalten wir in Assembler direkt die Werte in und aus Speicherstellen. Bei Variablen, die größer als 255 sind, verknüpfen wir mehrere davon. Zudem benötigen wir Pointer in der Zero-Page. Einige verwenden wir für indirekte Sprünge, andere ändern wir einfach und klinken uns so in schon bestehende Betriebssystem-Routinen ein.

Bei der Entwicklung der sechs Module ist einerseits darauf zu achten, daß die verwendeten Pointer und Variablen nicht von mehreren Routinen belegt und geändert werden, sehr seltsame Dinge geschähen sonst. Andrerseits müssen einige Parameterübergaben zwischen den einzelnen Programmteilen ermöglicht werden. Beispielsweise ist die Information »es fand eine Berührung Sprite/Hintergrund statt« unbedingt notwendig, sonst könnte nicht darauf reagiert werden:

1. Wir reservieren einen Bereich des Speichers speziell für alle Übergabe-Variablen und geben diesen im Assembler-Quelltext unverwechselbare Namen.

2. Variable, die nur im Modul verwendet werden, legen wir auch in diesen Speicherbereich. Dann ist es fast ausgeschlossen, daß einzelne Variablen kollidieren.

3. Wenn wir Pointer in der Zero-Page verwenden, weisen wir jeder Routine andere Speicherstellen zu. Eine andere Methode wäre es, die alten Pointerwerte zwischenzuspeichern, dann im Modul neu zu belegen und zum Schluß die vorher gespeicherten Werte wieder in den Pointer zu übertragen. Es spielt dann keine Rolle mehr, ob der Pointer auch in einer anderen Routine verwendet wird. Leider ist dieser Weg bei Spielen nicht möglich, da die meisten Routinen zeitkritisch sind und jeder unnötige Zeitzyklus vermieden werden sollte.

Im Textkasten »Zero-Page Adressen« (S. 21) sehen Sie eine Übersicht der Pointer, die das Betriebssystem nicht behindern.

Flußdiagramme

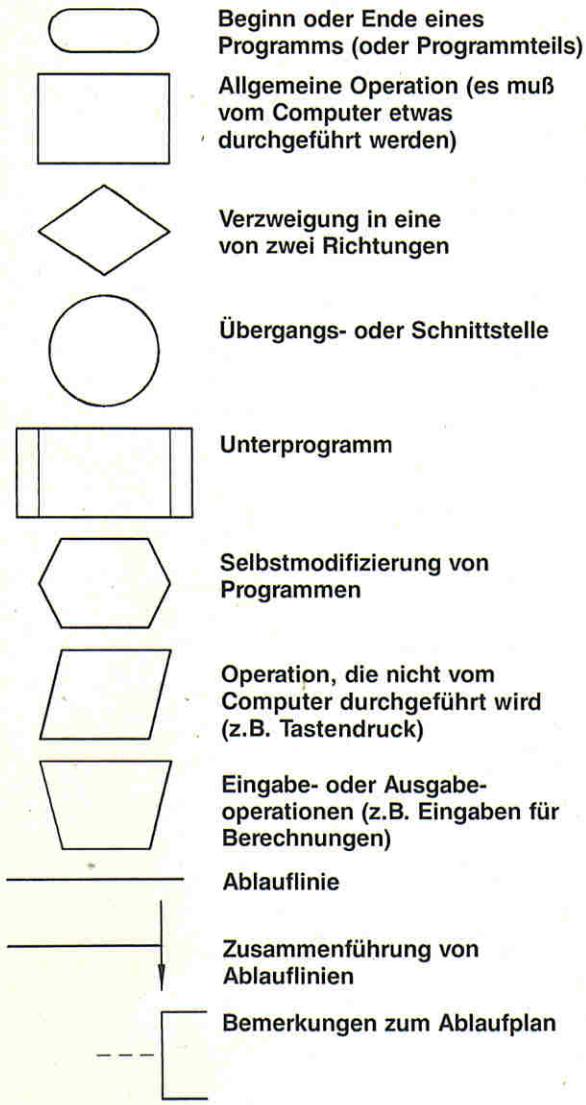
Wir haben eine wichtige Vorbereitung zur Programmierung schon gesehen – das Flußdiagramm. Für unsere Spielidee genügte es auch, einfach über ein paar Kästchen die Gedankengänge schriftlich zu fixieren. Wenn wir jetzt aber ans »Ein-

Quickreferenz SMON. Die Klammern dürfen nicht mit eingegeben werden. Die Werte in den Klammern können, aber müssen nicht eingegeben werden.

A 4000	Zeilenassembler Startadresse = $4000.
B 4000 4200	Erzeugt Basic-DATA-Zeilen im Bereich $4000 bis $41FF
C 4010 4200 4013 4000 4200	Verschieben eines Programmes mit Adreßumrechnung. Entspricht W- und V-Befehl.
D 4000 (4100)	Disassembliert den Bereich von $4000 bis $4100
F	findet Zeichenketten (F), absolute Adressen (FA), relative Sprünge (FR), Tabellen (FT), Zeropage-Adressen (FZ) und Immediate-Befehle (FI).
GO 4000	Startet Maschinenprogramm (ab $4000)
IO 1	Ein-/Ausgabegerät auf Datasette umstellen
K A000 (A100)	Im angegebenen Bereich nach ASCII-Zeichen suchen.
L"name" (4000)	Laden eines Programmes an die richtige (oder angegebene) Adresse
M 4000 (4100)	Gibt den Inhalt des angegebenen Speicherbereichs als Hex-Byte und ASCII-Zeichen aus.
O 4000 4100 12	Füllt den angegebenen Bereich mit $12.
PO 5	Setzt Drucker-Geräteadresse auf 5
R	Registerinhalte anzeigen
S"name" 4000 4500	Speichert ein Programm von $4000 bis $4FFF.
TW (4000)	Einzelschrittmodus. Mit »J« können Unterprogramme in Echtzeit ausgeführt werden.
TB 4010 (05)	Breakpoint setzen (nach dem 5. Durchlauf)
TQ 4000	Schnellschrittmodus. Springt beim Erreichen eines Breakpoints in die Registeranzeige.
TS 4000 4020	Arbeitet ein Programm ab $4000 in Echtzeit ab und springt beim Erreichen von $4020 in die Registeranzeige.
V 6000 6200 4000 4100 4200	Ändert alle absoluten Adressen $4000 bis $41FF, die sich auf den Bereich $6000 bis $6200 beziehen, auf den neuen Bereich $4000.
W 4000 4300 5000	Verschiebt den Speicherinhalt von $4000 bis $42FF nach $5000.
X	Monitor verlassen.
# 49152	Dezimal umrechnen
$ 002B	Vierstellige Hex-Zahl umrechnen.
% 01100100	Achtstellige Binärzahl umrechnen.
? 0344+5234	Addition oder Subtraktion zweier vierstelliger Hex-Zahlen.
= 4000 5000	Vergleicht den Speicherinhalt von $4000 bis $5000.
Z	Ruft den Diskettenmonitor auf (falls implementiert). Dieser verfügt über folgende Befehle
R (12 01)	Liest Track $12 Sektor $01. Fehlt die Angabe hinter »R«, wird der logisch nächste Sektor gelesen.
W (12 01)	Schreibt Track $12 Sektor $01 auf Diskette. Fehlt die Angabe hinter »W«, werden die letzten Angaben von »R« benutzt.
M	Zeigt den Pufferinhalt als Hex-Dump.
X	Rücksprung zum Monitor.
F	Weitere Diskettenbefehle initialisieren (falls implementiert). Sind die Befehle initialisiert, stehen folgende Befehle zur Verfügung.
M (07)	Memory-Dump (Floppy-RAM/ROM) ausgeben.
V 6000 0400	Verschiebt einen 256-Byte-Block von $6000 ins Floppy-RAM nach $400.
@	Normale Diskettenbefehle senden
X	Zurück in normalen Diskettenmonitor.
Ist die Erweiterung »Neues vom SMON« implementiert, stehen folgende Befehle zur Verfügung:	
Z 4000 (4100)	Gibt den Speicherinhalt von $4000 bis $40FF binär aus (ein Byte pro Zeile).
H 4000 (4100)	Gibt den Speicherbereich von $4000 bis $40FF binär aus (drei Byte pro Zeile).
N 4000 (4100)	Gibt den Speicherinhalt von $4000 bis $40FF im Bildschirmcode aus (32 Zeichen pro Zeile).
U 4000 (4100)	Wie »N« aber 40 Zeichen pro Zeile. Änderungen sind nicht möglich.
E 4000 (4100)	Füllt den Speicherbereich von $4000 bis $40FF mit $00.
Y 40	Verschiebt den SMON nach $4000.
Q 2000	Kopiert den Zeichensatz nach $2000.
J	Bringt letzten Ausgabebefehl zurück.

gemachte« gehen und das erste Modul vom Flußdiagramm bis zur Routine entwickeln, sollten wir uns ansehen, warum die einzelnen Kästchen unterschiedliche Formen haben:

Kurzinfo: Symbole nach DIN 66001 für Flußdiagramme



Ein Flußdiagramm ist eine schnelle und übersichtliche Methode bei der Entwicklung eigener Routinen, wenn Sie drei Dinge beachten:

1. Register und Pointer sollten grundsätzlich als erstes (außerhalb der Symbole) notiert werden. Ohne ihre Kenntnis stricken Sie das Flußdiagramm mehrere Male um.

2. Versuchen Sie nicht, alle Details in die Symbole einzutragen sonst leidet die Übersicht.

3. Bei zu wenig Information haben Sie keine Chance, den Ablauf nachzuvollziehen.

Beginnen wir mit Modul 1, der Joystickabfrage und Überlegen wir uns zuerst, welche Register wir dazu benötigen (Tabelle 1, S. 26 und Tabelle 3, S. 27):

Wir verwenden Joyport 2 für unsere Abfrage. Das entsprechende Register ($DC00) dafür muß zuerst vorbereitet werden, da es vom Betriebssystem zur Spaltenauswahl bei der Tastaturabfrage verwendet wird, also auf Ausgabe geschaltet ist. Wir schalten um, indem wir in Register $DC02 den Wert $E0 schreiben. Dann läßt sich in Register $DC00 das Bit-Muster des Joyport 2 auslesen. Dabei entsprechen fünf Bits jeweils einer Joystick-Funktion:

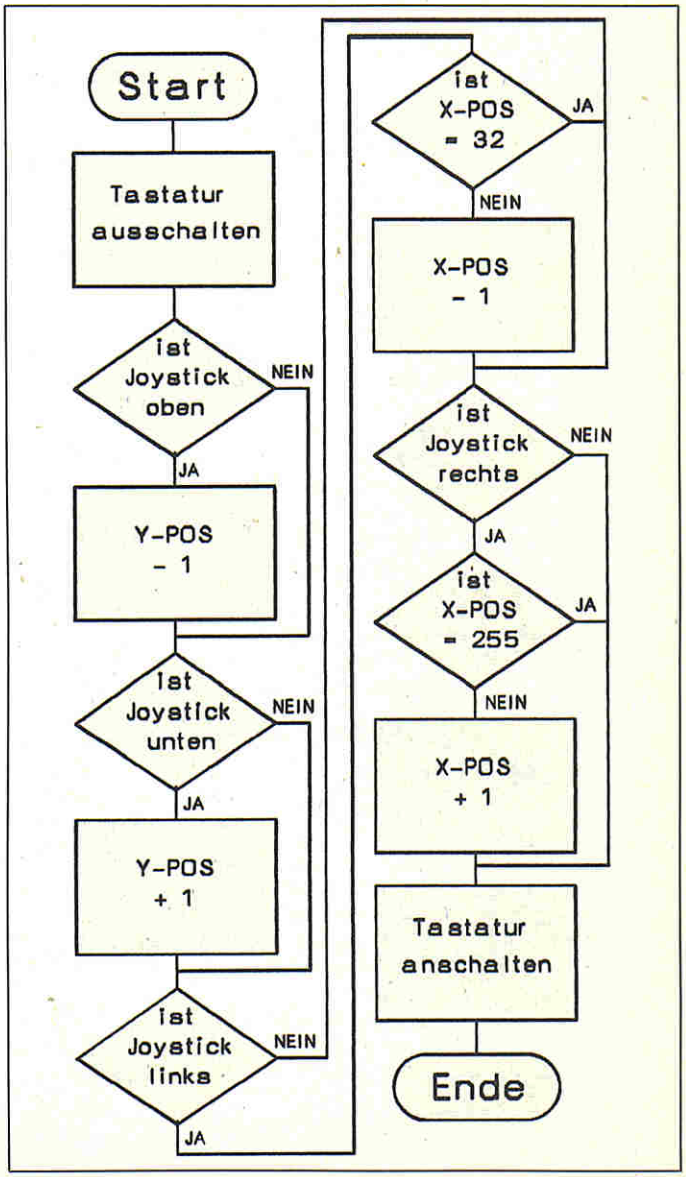
Bit 0 = oben
Bit 1 = unten
Bit 2 = links
Bit 3 = rechts
Bit 4 = Knopf (Feuer)

Achtung: Der Ausgangswert bei unbetätigtem Joystick ist $FF, nicht $00, wie man meinen könnte. Der Grund liegt im Prinzip des Joysticks. Für jede Position (oben, unten, links, rechts und Knopf) ist ein eigener Taster im Joystick eingebaut. Eine Betätigung verbindet die entsprechende Portleitung mit Masse (low). Es muß also eine Spannung an den Portleitungen anliegen (+5V). Im Computer ist grundsätzlich vereinbart, daß Spannung (+5V) als High, also gesetztes Bit interpretiert wird. Eine Joystick-Aktion löscht daher das gesetzte Bit. Wir prüfen das mit einer AND-Maske.

Legen wir noch fest, daß unser Raumschiff Sprite 0 ist. Danach notieren wir die nötigen Sprite-Register (s. Tabelle 3, S. 28) und die verwendeten Pointer:

Datenrichtungsregister = $DC02
JOYPORT = $DC00
V = $D000 (Basisadresse Video-Interface-Chip)
x-Position/Sprite 0 = V
y-Position/Sprite 0 = V+1

Aus diesen Grundlagen heraus überlegen wir uns ein Flußdiagramm:



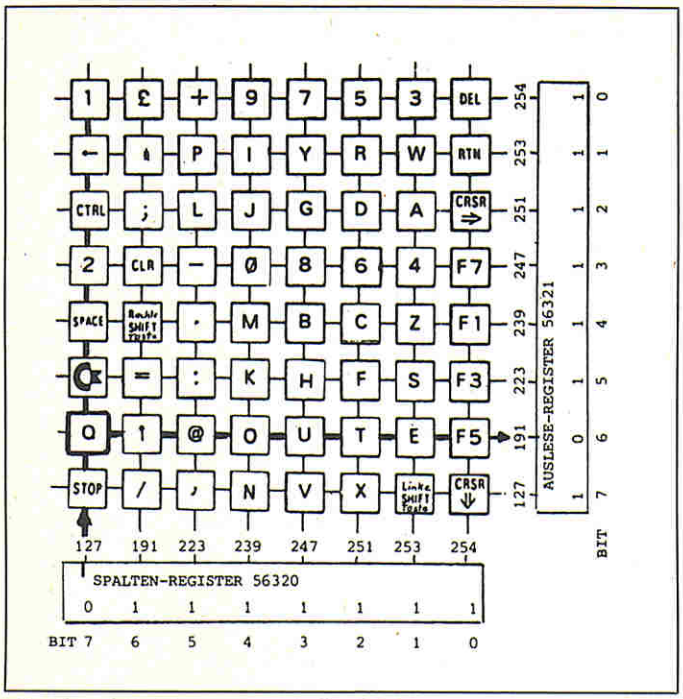


Tabelle 1. Die Tastaturmatrix im C64

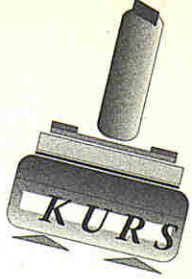
Wenn wir nun unser Maschinenprogramm für die Joystickabfrage entwickeln, müssen wir die Aktionen nur noch in Assemblerbefehle umsetzen. Sie finden diese Routine im abgedruckten Gesamt-Listing des Demo-Spiels von Zeile 1100 bis Zeile 1360.

Beachten Sie, daß ein Quelltext immer kommentiert in die eigene Bibliothek gehen sollte, obwohl die Kommentare für die Funktion des Programms nicht nötig sind. Sollten Sie später dieses Modul in ein anderes Programm einbauen, ist seine Funktion sofort wieder nachvollziehbar und kritische Partien, wie Position der Übergabe-Variablen usw., lassen sich leichter ins Gesamtkonzept einplanen.

Sollten Sie das Listing aus Zeile 1100 bis 1360 abtippen, assemblieren und testen, werden Sie feststellen, daß sich (scheinbar) rein gar nichts tut. Warum? - das Sprite ist zwar verschoben, aber weder definiert, noch auf eine Anfangsposition gesetzt und auch nicht eingeschaltet.

Wir benötigen daher zusätzlich zu diesem Modul noch ein Programm zur Initialisierung. Auch hier überlegen wir uns, welche Funktionen in der Testphase und welche später im Programm nötig sind: Zunächst muß die Routine automatisch ablaufen, daher bietet sich der Interrupt an:

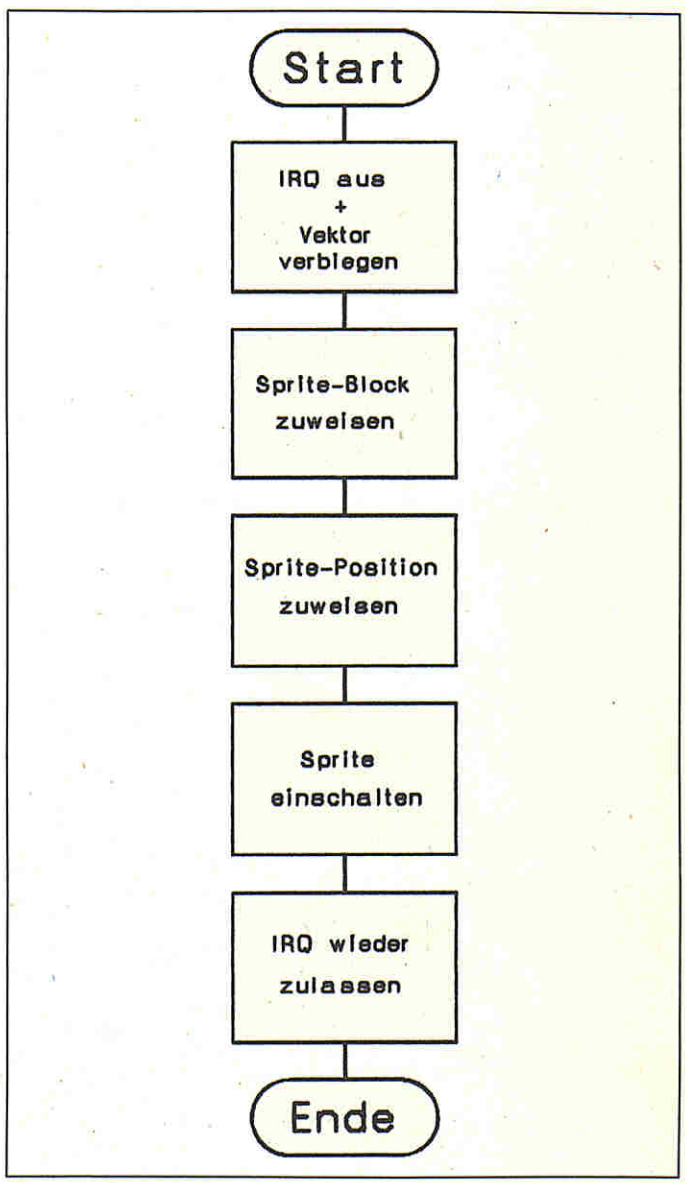
Sie errinnern sich, jede sechzigstel Sekunde unterbricht der Mikroprozessor das normale Programm und tut alles das, was im Computer scheinbar von allein abläuft. Dies geht vom Cursorblinken über die Tastaturabfrage, bis hin zur Erhöhung von TI und TI$. Dafür existiert im Betriebssystem eine extra Interrupt-Routine. Aber ein IRQ muß auch irgendwie ausgelöst werden können. Dafür ist CIA 1 (Complex-Interface-Adapter) im C64 verantwortlich. Dieser Baustein ist gleich zweimal im Computer vorhanden (Tabelle 2, S. 27). Jeder enthält zwei 16-Bit-Timer (Abwärtszähler) von denen jeder beim Zählerstand »0« einen Impuls zum Mikroprozessor sendet und ihn damit zwingt, die IRQ-Routine auszuführen. Danach wird der Baustein wieder mit dem voreingestellten Wert geladen und erneut herabgezählt. Da die Interrupt-Routine über einen indirekten Sprung (JMP ($0314)) verwendet wird, müssen wir nur den Vektor ($0314/$0315) auf unsere eigene Routine richten. Im späteren Programm ist genau dieses Verbiegen nicht erlaubt, da andere Aufgaben auch »automatisch« erledigt werden und der Vektor immer auf die zuletzt verbogene Routine zeigen würde. Alle anderen Routinen wären dadurch



ausgeschaltet. Wir werden daher später alle Module zusammenfassen und der Reihe nach aufrufen. Außerdem lernen wir noch andere Arten kennen, die den Interrupt auslösen. Zum Schluß unserer Testroutine springen wir in die Interrupt-Routine des Betriebssystems. Auch diesen Teil benötigen wir im Spiel nicht.

Bevor wir wieder ein Flußdiagramm aufzeichnen, notieren wir uns die benötigten Register und Pointer:
IRQ-Vektor - $0314/$0315
Sprite-Block - 32
Sprite-Pointer - 2040
Sprite 0 - 1 (#00000001)
Sprite Ein - VIC+21

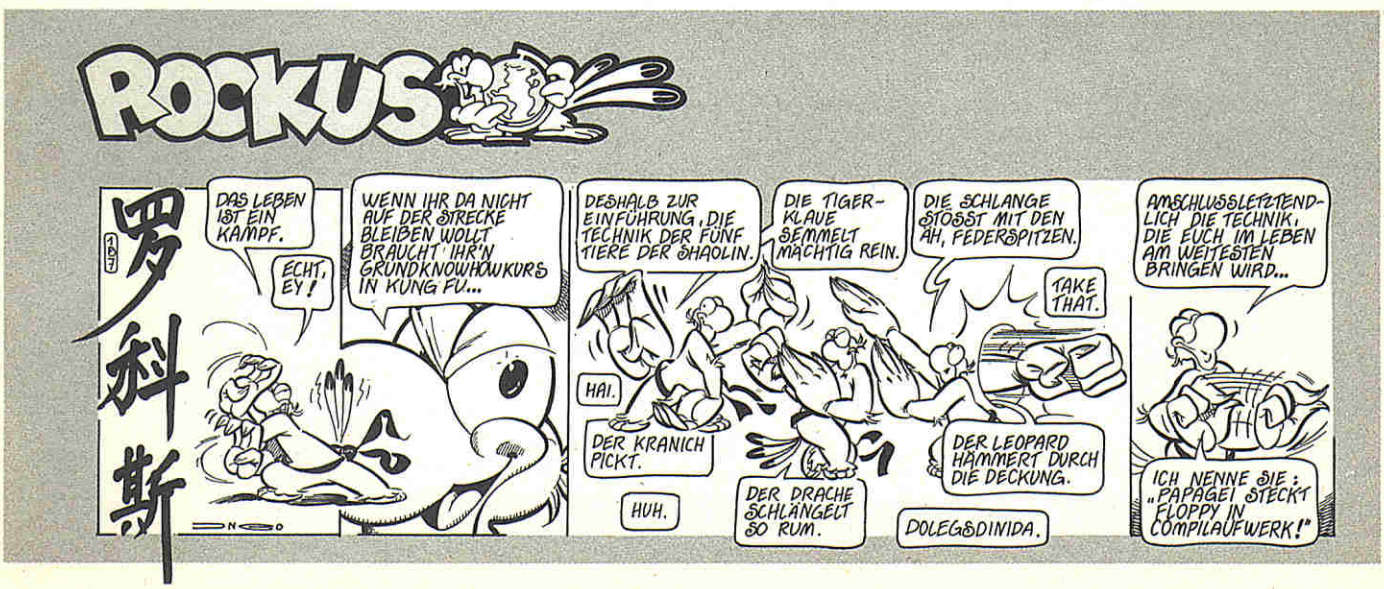
Daraus entsteht dann unser Testprogramm:



Mit ihm lassen sich alle Funktionen überprüfen. Im Demo-Spiel ist die Routine »SETSPRITE« ab Zeile 150 dafür zuständig, daß Sprite 0 auf x-Position 130, y-Position 114 und auf Normalgröße gesetzt wird. Weiterhin wird das Sprite auf Multicolor, mit den Farben Rot und Grün gesetzt. Die Daten stehen in Sprite-Block 32.

Reg.	Adresse		Funktion	Verwendung
0	56320 56576	($dc00) ($dd00)	Datenregister Port A Ein gesetztes Bit signalisiert High an der entsprechenden Port-Leitung	Tastaturabfrage IEC-Bus + RS232
1	56321 56577	($dc01) ($dd01)	Datenregister Port B Wie Register 0, jedoch für Port B	Tastaturabfrage User-Port
2	56322 56578	($dc02) ($dd02)	Datenrichtungsregister Port A Ein gesetztes Bit programmiert die zugehörige Portleitung als Ausgang	zusammen mit Register 0 zusammen mit Register 0
3	56323 56579	($dc03) ($dd03)	Datenrichtungsregister Port B Wie Register, jedoch für Port B	zusammen mit Register 1 zusammen mit Register 1
4	56324 56580	($dc04) ($dd04)	Timer A, Low-Byte Beim Lesen wird der momentane Zählerstand erhalten, beim Schreiben der Zählerstand (Low-Byte) gesetzt, von dem der 16-Bit-Zähler nach Null zählt	IRQ (alle 1/60 s) RS232
5	56325 56581	($dc05) ($dd05)	Timer A, High Byte Wie Register 4, jedoch für High-Byte, Timer A Siehe auch Register 14 (Control-Register A)	zusammen mit Register 4 zusammen mit Register 4
6	56326 56582	($dc06) ($dd06)	Timer B, Low-Byte Wie Register 4, jedoch für Timer B Siehe auch Register 15 (Control-Register B)	für Kassetten Op. RS232
7	56327 56583	($dc07) ($dd07)	Timer B, High-Byte Wie Register 5, jedoch für Timer B Siehe auch Register 15 (Control-Register B)	zusammen mit Register 6 zusammen mit Register 6
8	56328 56584	($dc08) ($dd08)	Time of Day 1/10 Sekunden Bit 0-3 enthalten die 1/10 Sekunden im BCD-Format. Ist Bit 7 in Register 15 gesetzt, so wird beim Schreiben die Alarmzeit gesetzt, ansonsten die Uhrzeit. Bit 4-7 unbenutzt.	(für RND) unbenutzt
9	56329 56585	($dc09) ($dd09)	Time of Day Sekunden Dieses Register enthält die Sekunden im BCD-Format. Schreibzugriff siehe Register 8	(für RND) unbenutzt
10	56330 56586	($dc0a) ($dd0a)	Time of Day Minuten Dieses Register enthält die Minuten im BCD-Format Schreibzugriff siehe Register 8	(für RND) unbenutzt
11	56331 56587	($dc0b) ($dd0b)	Time of Day Stunden Bit 0-3 enthalten die Stunden im BCD-Format, Bit 4 die 10er Stunden. Bit 7 ist bei AM (vormittags) 0 und bei PM (nachmittags) 1. Bit 5+6 unbenutzt Schreibzugriff siehe Register 8	(für RND) unbenutzt
12	56332 56588	($dc0c) ($dd0c)	Serial Data Register (SDR) Schieberegister, über das Daten am Pin SP herausgeschoben und hereingeholt werden. Das höchstwertige Bit erscheint zuerst.	unbenutzt unbenutzt
13	56333 56589	($dc0d) ($dd0d)	Interrupt Control Register (ICR) Bit 0: Unterlauf Timer A Bit 1: Unterlauf Timer B Bit 2: Uhrzeit und Alarmzeit sind gleich Bit 3: Schieberegister voll oder leer (je nach Betriebsart) Bit 4: 1, wenn negative Spannungsflanke an FLAG aufgetreten ist Bit 5 und Bit 6 sind immer 0 Bild 7: Es stimmt mindestens ein gesetztes Bit im INT MASK und INT DATA-Register überein Achtung: Beim Lesen wird das ICR gelöscht!	
14	56334 56590	($dc0e) ($dd0e)	Control Register A (CRA) Bit 0: 1=Timer A starten 0=Timer A stoppen Bit 1: 1=Ein Umlauf von Timer A wird an PB 6 signalisiert, auch wenn dieses Port-Bit als Eingang programmiert ist. Bit 2: 1=Bei einem Unterlauf von Timer A wird PB 6 invertiert Bit 3: 0=Continous-Mode 1=One-Shot-Mode Bit 4: Wird eine 1 eingeschrieben, so wird Timer A sofort mit dem Wert geladen, der vorher in Register 4 + 5 stand, egal ob der Timer gerade läuft oder nicht. Bit 5: 1=Timer A zählt positive Flanken an CNT 0=Timer A zählt Systemtakte Bit 6: 0=Das Schieberegister ist Eingang 1=Das Schieberegister ist Ausgang Bit 7: 1=TOD verarbeitet 50 Hz Netzfrequenz 0=TOD verarbeitet 60 Hz Netzfrequenz	
15	56335 56591	($dc0f) ($dd07)	Control Register B (CRB) Bit 0-4: entsprechen Bit 0-4 von CRA, jedoch für Timer B und PB 7 Bit 5+6 bestimmen paarweise die Triggerquelle 00=Timer B zählt Systemtakte 01=Timer B zählt positive Flanken an CNT 10=Timer B zählt Unterläufe von Timer A 11=Timer B zählt Unterläufe von Timer A nur, wenn CNT high ist Bit 7: 1=TOD Alarmzeit setzen 0=TOD Uhrzeit setzen	

Tabelle 2. Register der CIAS



Die Register des VIC
Basisadresse 53248 ($D000)

Register	Normwert	Zweck
0 53248 $D000	0, wenn Sprite nicht verwendet	x-Koordinate von Sprite #0 ist der Wert < 24, dann ist der Sprite verdeckt.
1 53249 $D001	0, wenn Sprite nicht verwendet	y-Koordinate von Sprite #0 ist der Wert < 50 > 229, dann ist der Sprite verdeckt
2 53250 $D002	siehe 0	x-Koordinate von Sprite #1 sonst siehe 0
3 53251 $D003	siehe 0	y-Koordinate von Sprite #1 sonst siehe 0
4 53252 $D004	siehe 0	x-Koordinate von Sprite #2 sonst siehe 0
5 53253 $D005	siehe 0	y-Koordinate von Sprite #2 sonst siehe 0
6 53254 $D006	siehe 0	x-Koordinate von Sprite #3 sonst siehe 0
7 53255 $D007	siehe 0	y-Koordinate von Sprite #3 sonst siehe 0
8 53256 $D008	siehe 0	x-Koordinate von Sprite #4 sonst siehe 0
9 53257 $D009	siehe 0	y-Koordinate von Sprite #4 sonst siehe 0
10 53258 $D00A	siehe 0	x-Koordinate von Sprite #5 sonst siehe 0
11 53259 $D00B	siehe 0	y-Koordinate von Sprite #5 sonst siehe 0
12 53260 $D00C	siehe 0	x-Koordinate von Sprite #6 sonst siehe 0
13 53261 $D00D	siehe 0	y-Koordinate von Sprite #6 sonst siehe 0
14 53262 $D00E	siehe 0	x-Koordinate von Sprite #7 sonst siehe 0
15 53263 $D00F	siehe 0	y-Koordinate von Sprite #7 sonst siehe 0
16 53264 $D010	siehe 0	9. Bit der x-Koordinaten für Pos. 256-319 zuständig: Bit: 7|6|5|4|3|2|1|0 Sprite: 7|6|5|4|3|2|1|0
17 53265 $D011	27	Kontrollregister für Modus des VIC Belegung: 0 1 > weiches scrollen (0-7) 2 3 25/24 Zeilen 1/0 4 VIC ein/aus 1/0 5 HiRes/Text 1/0 6 Hintergrundfarb/Text 1/0 7 9.Bit Rasterwert
18 53266 $D012	wechselt	Nummer der gerade aktuellen Rasterzeile
19 53267 $D013	wechselt	x-Koordinate des Lightpen
20 53268 $D014	wechselt	y-Koordinate des Lightpen
21 53269 $D015	0, wenn Sprite nicht verwendet	Schaltet die Sprites aus/ein aus (0) / ein (0) Bit: 7|6|5|4|3|2|1|0 Sprite: 7|6|5|4|3|2|1|0
22 53270 $D016	200	Kontrollregister für erweiterte Funktionen 0 1 > weiches scrollen (0-7) 2 3 40/38 Textspalten 1/0 4 Multicolor/Normal 1/0 5 immer 0 6 immer 1 7 immer 1

Die Register des VIC
Basisadresse 53248 ($D000)

Register	Normwert	Zweck
23 53271 $D017	0, wenn Sprite nicht verwendet	x-Vergrößerung Sprite # 0 klein (0) / groß (1) Bit: 7|6|5|4|3|2|1|0 Sprite: 7|6|5|4|3|2|1|0
24 53272 $D018	21	Speicheraufteilung des VIC 0 immer 1 1 2 > Character-RAM/ROM 3 4 5 > Bildschirm-RAM 6 7
25 53273 $D019	wechselt	IRQ Statusregister Ergebniss eines IRQ-Ereignis 0 Rasterzeile 1 Sprite-Hintergrundkollision 2 Sprite-Spritekollision 3 Lightpen 4 immer 1 5 immer 1 6 immer 1 7 Interrupt stattgefunden(1)
26 53274 $D01A	wechselt	IRQ Maske Die in Reg. 25 abzufragenden Ereignisse werden mit Bit=1 zugelassen 0 Raster-IRQ 1 Sprite-Hintergrund 2 Sprite-Sprite 3 Lightpen 4 immer 1 5 immer 1 6 immer 1 7 immer 1
27 53275 $D01B	0, wenn Sprite nicht verwendet	Priorität des Sprite 0 = vor, 1 = hinter dem Text Bit: 7|6|5|4|3|2|1|0 Sprite: 7|6|5|4|3|2|1|0
28 53276 $D01C	0, wenn Sprite nicht verwendet	Multicolor für Sprites 0 Schaltet aus, 1 schaltet ein Bit: 7|6|5|4|3|2|1|0 Sprite: 7|6|5|4|3|2|1|0
29 53277 $D01D	0, wenn Sprite nicht verwendet	y-Vergrößerung Sprite # 0 klein (0) / groß (1) Bit: 7|6|5|4|3|2|1|0 Sprite: 7|6|5|4|3|2|1|0
30 53278 $D01E	0, wenn Sprite nicht verwendet	Kollisionsregister Sprite-Sprite Bit= 0 wenn keine Kollision Bit: 7|6|5|4|3|2|1|0 Sprite: 7|6|5|4|3|2|1|0
31 53279 $D01F	0, wenn Sprite nicht verwendet	Kollisionsregister Sprite-Hintergrund Bit= 0 wenn keine Kollision Bit: 7|6|5|4|3|2|1|0 Sprite: 7|6|5|4|3|2|1|0
32 53280 $D020	254	Rahmenfarbe 0 = schwarz 8 = orange 1 = weiß 9 = braun 2 = rot 10 = hellrot 3 = cyan 11 = grau III 4 = rosa 12 = grau II 5 = grün 13 = hellgrün 6 = blau 14 = hellblau 7 = gelb 15 = grau I

Die Register des VIC Basisadresse 53248 ($D000)		
Register	Normwert	Zweck
33 53281 $D021	246	Bildschirmfarbe <Farben s. Reg. 32>
34 53282 $D022	241	Farbe 1 für Multicolor und erweiterter Hintergrundfarb-Modus <nur Text> <Farben s. Reg. 32>
35 53283 $D023	242	Farbe 2 für Multicolor und erweiterter Hintergrundfarb-Modus <nur Text> <Farben s. Reg. 32>
36 53284 $D024	243	Farbe 3 für Multicolor und erweiterter Hintergrundfarb-Modus <nur Text> <Farben s. Reg. 32>
37 53285 $D025	244	Farbe 1 für Multicolor-Sprite <Farben s. Reg. 32>
38 53286 $D026	244	Farbe 2 für Multicolor-Sprite <Farben s. Reg. 32>
39 53287 $D027	241	Farbregister Sprite #0 <Farben s. Reg. 32>
40 $D028 53288	242	Farbregister Sprite #1 <Farben s. Reg. 32>
41 $D029 53289	243	Farbregister Sprite #2 <Farben s. Reg. 32>
42 $D02A 53290	244	Farbregister Sprite #3 <Farben s. Reg. 32>
43 $D02B 53291	245	Farbregister Sprite #4 <Farben s. Reg. 32>
44 $D02C 53292	246	Farbregister Sprite #5 <Farben s. Reg. 32>
45 $D02E 53293	247	Farbregister Sprite #6 <Farben s. Reg. 32>
46 $D02F 53294	252	Farbregister Sprite #7 <Farben s. Reg. 32>

Tabelle 3. Alle Register des VIC

Mit unserer Routine läßt sich das Sprite in den von uns gesetzten Grenzen am Bildschirm bewegen. Aber wir haben noch keinen scrollenden Hintergrund. Überlegen wir uns zuerst eine Methode:

Beginnen wir bei der Initalisierung des Spiels und kopieren den ersten Teil des Landschaftsausschnitts in den Bildschirmspeicher. Danach warten wir auf den Spielebeginn (Druck auf den Feuerknopf). Anschließend kopieren wir pro Interrupt jeweils einen um ein Zeichen weiter rechts liegenden Ausschnitt aus unserer Spielelandschaft und erreichen damit ein sog. Hardscrolling. Bei der horizontalen Länge von 512 Zeichen wären wir nach 512-40=472 IRQ-Zyklen am Ende des Spielfelds angekommen. Das sind ca. acht Sekunden (472/60). Sie werden zugeben, zu schnell um im Spiel auf irgend etwas reagieren zu können. Also bauen wir zusätzlich einen Verzögerungszähler ein. Starteten wir jetzt unsere theoretische Routine, würde folgendes passieren:

1. Sowohl Sprite als auch Landschaft flackern hin und wieder.

2. Bei einer passablen Geschwindigkeit ruckelt unsere Landschaft unter dem Sprite.

Untersuchen wir das erste Phänomen genauer:

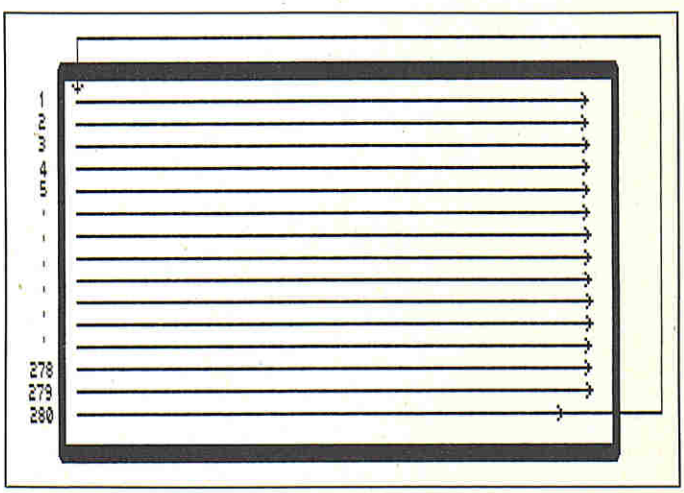
Der Grund für das Flackern ist der physikalische Aufbau eines Bildes. Er dauert 1/50stel Sekunde, findet also 50 mal pro Sekunde statt. Dabei beginnt ein Rasterstrahl (Abb. 4) links oben in der ersten Zeile am Monitor seine Arbeit. Je nachdem was er darstellen soll, leuchtet er heller oder dunkler, bzw. in einer anderen Farbe. Ist er am Ende der Zeile angekommen, beginnt seine Tätigkeit am Anfang der nächsten Zeile wieder. Da dies für 280 Zeilen innerhalb 1/50stel Sekunde passiert, wird unser Auge überlistet und für uns entsteht der Eindruck eines stehenden Bildes.

Wenn wir nun zu einem willkürlichen Zeitpunkt ein Muster am Bildschirm ändern, kann es passieren, daß ein Teil des Musters geändert wird, während der Rasterstrahl am Monitor gerade dieses Muster überholt. Bei diesem Bildzyklus sieht man also ein Teil des alten Musters und einen Teil des neuen. Da dies aber öfters geschieht, empfindet unser Auge ein deutliches Flackern. Abhilfe hätten wir, wenn es uns gelingt, die Änderung eines Bildschirminhalts mit dem physikalischen Bildaufbau zu synchronisieren. Und tatsächlich, der VIC bietet uns hier eine Möglichkeit, da er festhält, in welcher Zeile sich der Rasterstrahl gerade befindet. Zuständig dafür sind Register 53266 (V+18) und da es 280 Zeilen gibt, dient das Bit 7 von 53265 (V+17) als neuntes Bit (Abb. 5, S. 30). Wenn wir hier einen Wert vorgeben ist der VIC in der Lage, einen Interrupt auszulösen. Allerdings müssen wir dem VIC noch mitteilen, daß er einen IRQ durchführen soll. Dafür ist Bit 0 des Registers 53274 (V+26) zuständig. Dazu ein Beispiel; in ihm wird zwischen Groß/Kleinschrift und Groß/Grafik umgeschaltet:

In der Initialisierungs-Routine wird der IRQ-Vektor verbogen und danach die erste Rasterzeile für den IRQ festgelegt. Anschließend wird der IRQ durch Rasterzeilen zugelassen. Die neue IRQ-Routine löscht zuerst das Rasterregister und prüft gleichzeitig, ob der IRQ tatsächlich durch den VIC ausgelöst wurde (Negative-Flag). Die Rasterzeilen-Routine wechselt den Zeichensatz und legt die Rasterzeile neu fest, bei der ein IRQ ausgeführt wird.

Auch mit unserem zweiten Ärgernis, dem Ruckeln beim Scrollen, werden wir fertig - der VIC bietet auf einfachste Weise eine Soft-Scroll-Funktion:

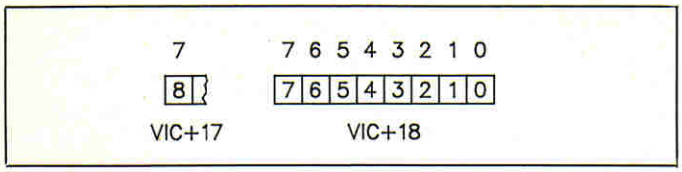
Ein Bildschirmzeichen besteht aus einer Matrix von achtmal acht Punkten. Wir sind bis jetzt davon ausgegangen, daß auch mit dem ersten Punkt des ersten Zeichens das Bild beginnt. Doch der VIC kann auch anders. Dazu bietet er ein Register. Bits 0 bis 2 des Registers 53270 (V+22) geben an, bei



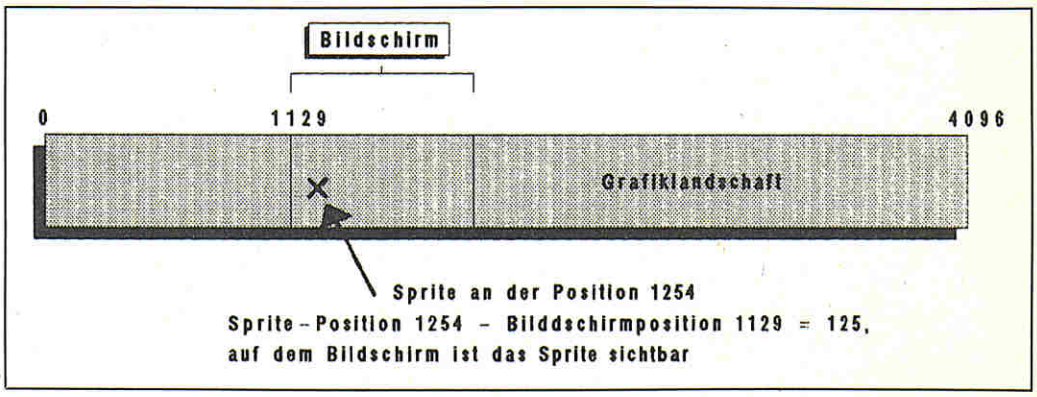
[4] Der Aufbau des Bildschirms durch den Rasterstrahl

welchem Punkt die Bildschirmdarstellung beginnt. Für normalen Beginn ist dieser Wert demzufolge »0«. Wir verschieben unseren Bildschirm nach links, daher beginnen wir mit »7«. Wir müssen also siebenmal diesen Teil des Registers herabzählen, bis es den Wert »0« hat (Zeile 2230 bis 2450 im Listing, S. 33). Gleichzeitig dazu können wir den nächsten Ausschnitt ins Bildschirm-RAM kopieren (Zeile 1370 bis 1640). Ein kontinuierliches, weiches Scrollen nach links ist der Erfolg.

Bevor Sie für ein eigenes Spiel eine Scroll-Routine entwickeln, müssen Sie allerdings beachten: Ihre Routinen dürfen nicht länger dauern als ein physikalischer Bildaufbau, da



[5] Das neunte Bit des Rasterstrahls ist Bit 7 im Register 17 des VIC. Zusammen mit Register 18 wird hier die Position eines Raster-IRQ definiert.



[6] Berechnung der Spriteposition

sich sonst das Bild selbst überholt. Es stehen Ihnen dafür während eines sichtbaren Rasterstrahldurchlaufs 200 x 63 = 12600 Taktzyklen zur Verfügung.

Automatische Gegner

Die konstruierte Grafik ist 512 Zeichen breit. Das bedeutet, sie hat 512 x 8 = 4096 Bildpunkte Ausdehnung. Über diese Breite darf sich ein computergesteuertes Hindernis (Sprite) bewegen. Es darf aber nur eingeschaltet sein, wenn es sich im momentan sichtbaren Bereich (320 Bildpunkte) der Spiele-Landschaft befindet. Daher muß berechnet werden, wann das Sprite sichtbar ist:

Wir zählen von Anfang an die aktuelle Position mit (»ZAEHLEN«, Zeile 4050 bis 4120) und reservieren dafür zwei Variablen »POSSCRLO« für das Low-Byte und »POSSCRHI« für das High-Byte. Im Unterprogramm »INITSPR« werden die Positionen der gegnerischen Sprites dann definiert. Jedem Sprite wird eine 16-Bit x-Position und 8-Bit y-Position zugeteilt; zusätzlich noch eine Speicherposition, in der die jeweilige Bewegungsrichtung gespeichert ist. Diese Daten stehen von Zeile 3730 bis 4000 im Listing. Es lassen sich durchaus mehr Gegner definieren, Sie müssen allerdings sicherstellen, daß sich nie mehr als acht Sprites gleichzeitig auf dem aktuellen Bildschirm tummeln können.

Während des Spielablaufs verfügen wir über die Position des Bildschirms und der gegnerischen Sprites. Daraus berechnen wir die jeweilige Spriteposition (Abb. 6):

Nehmen wir an, daß sich Sprite 1 an Position 1254 befindet und der Bildschirm auf Position 1129. Das Ergebnis der Rechnung Spriteposition (1254) minus Bildschirmposition (1129) ist genau die Spriteposition am Bildschirm (125). Ergibt die Rechnung einen Wert kleiner 0 oder größer 344, ist der Gegner unsichtbar. Im anderen Falle muß das Sprite an der berechneten Position eingeschaltet werden. Diese Rechnung übernimmt das Unterprogramm »SPRITE« von Zeile 2930 bis 3490:

Die Bildschirmposition wird von der Spriteposition subtrahiert. Das Low-Byte des Ergebnisses gelangt ins y-Register, das High-Byte ins x-Register. Hat das High-Byte den Wert »0«, wird das Unterprogramm »SPRITEON« ausgeführt. Ist das Ergebnis weder »0« noch »1«, wird zu »SPRITEOFF« verzweigt.

Die Bewegung der Sprites geschieht im Unterprogramm »ENEMIES« ab Zeile 4130. Als Variable dient die Speicherstelle 821. Sie wird zunächst auf »0« gesetzt. Der Inhalt der Bewegungsrichtung des ersten Gegners »ENEMY1DIR« wandert in den Akku. Ist er »0« (Bewegungsrichtung links), wird von der 16-Bit x-Position 1 subtrahiert, ansonsten 1 addiert (Bewegungsrichtung rechts). Sollte beim Subtrahieren die x-Position = 160 sein, wird die Richtung durch Erhöhen von »ENEMY1DIR« auf »1« nach rechts geändert.

Im Unterprogramm »ENEMIES2« (Zeile 4590 bis 4860) geschieht ähnliches für die y-Position der Sprites. Der wichtigste Unterschied ist, daß wir die Werte 96 bis 152 als Bewegungsgrenzen festlegen.

Türen öffnen und schließen sich

Wir haben beschlossen, dafür Zeichenmuster im Zeichensatz zu ändern. In unserem Demo-Listing sind dies die Zeichen 68 bis 85. Wir wollen jeweils die Zeichen 68 bis 76 und die Zeichen 77 bis 85 durchscrollen lassen. Das heißt: 68 wird 76, 69 wird 68, 70 wird 69 bis zum Zeichen 76, es bekommt das Muster von 75. Das gleiche machen wir mit den Zeichen 77 bis 85. Dazu speichern wir zuerst die Zeichenmuster von 76 und 85 (je acht Byte). Dann scrollen wir alle Zeichenmuster ab 76 bis 85 um acht Byte (ein Zeichen). Zum Schluß übertragen wir die Muster der gespeicherten Zeichen in 68 und 77. Damit dies nicht zu schnell geschieht, bauen wir noch einen Verzögerungszähler ein. Im Listing reservieren wir zum Zwischenspeichern 16 Byte (von Zeile 8000 bis Zeile 8150) und für den Verzögerungszähler ein Byte (Zeile 8151). Ab Zeile 8152 beginnt das eigentliche Programm.

Wenn Sie ein eigenes Playfield entwickeln, sollten Sie berücksichtigen, daß die Zeichen 68 bis 85 später im Spiel ständig getauscht werden.

Animation von Raumschiff und Gegnern

Zur Animation des Raumschiffs haben wir acht Spriteformen in Spriteblock 32 ($0800) bis 39 zur Verfügung. Wir haben mit einem Kreis begonnen, der über immer kleiner werdende Elypsen bis zum Strich in Block 39 wird. Eine (scheinbare) Drehung erreichen wir durch zwei Phasen. Bei jedem Aufruf der Routine wird von Spriteblock 32 über 33 bis 39 umgeschaltet, dann von 39 über 38 bis 32. Beim Wechsel der Drehrichtung ändern wir die Spritefarbe, durch Invertieren von Bit 2 im Farbregister von Sprite »0«. Da sich unser Sprite zu schnell drehen würde, wenn wir bei jedem Rasterinterrupt aufrufen, verwenden wir auch hier eine Zählvariable.

Zeile 320 bis 590 sind für diese Aufgabe zuständig.

In dem Demo-Game sind allen Gegnern dieselben Spritemuster zugeteilt. Das vereinfacht die Konstruktion. Die Konstruktion eines sich drehenden Rings ist von Blocknummer 40 bis 47 abgelegt. Animiert wird analog der Routine für das Raumschiff. Im Listing sind dafür die Zeilen 6280 bis 6620 zuständig.

Kollision führt zum Spielende

Wir wollen das Spiel beenden, wenn unser Raumschiff mit dem Hintergrund oder einem Gegner zusammengestoßen ist. Dies ist wieder durch einen Interrupt des VIC möglich. Dieser Baustein ist in der Lage, durch eine Berührung zweier Sprites und/oder eines Sprite mit dem Hintergrund, einen IRQ auszulösen. Dazu ist Register 53274 (V+26) zuständig. Bit 1 erlaubt einen IRQ für Sprite/Hintergrund, Bit 2 den für Sprite/Sprite. In unserem Demo-Spiel machen wir von diesem Interrupt keinen Gebrauch. Für uns ist es uninteressant, wenn die Gegner mit dem Hintergrund kollidieren, daher müssen wir auf jeden Fall das Sprite/Hintergrund-Kollisionsregister (53279 = V+31) abfragen, wer mit wem zusammengestoßen ist. Jedes gesetzte Bit in diesem Register entspricht einer Sprite/Hintergrund-Berührung des entsprechenden Sprites. Beim zweiten Kollisions-Register (53278 = Sprite/Sprite) prüfen wir lediglich, ob überhaupt ein Bit gesetzt ist, da unsere Gegner nicht miteinander kollidieren können. Wenn wir dies ganz normal im Raster-Interrupt machen, sind wir im ungünstigsten Fall um 1/25 Sekunde später dran, als mit einem eigenen IRQ. Das stört uns nicht, wichtig ist, daß überhaupt reagiert wird. Im Demo-Programm erledigen diese Abfrage die Zeilen 2790 bis 2816 für uns. Die Maske in Zeile 2791 überprüft, ob das Raumschiff (Sprite 0 = Bit 0 = %00000001 = 1) bei der Kollision beteiligt war. Wenn ja, wird eine Spielvariable erhöht, und beim nächsten Rasterinterrupt das Spiel abgebrochen.

Initialisierung und Aufruf der Routinen

Das Wichtigste für unser Spiel fehlt uns noch: Die Initialisierung unserer Routinen und ihre Verbindung. Diesen Programmteil kann man nicht als eigenständiges Modul betrachten, da er als einziger für jede neue Spielvariante auch neu geschrieben werden muß. Überlegen wir uns, was wir alles initialisieren:

1. Blocknummer für Sprite 0 zuweisen, das Sprite positionieren und einschalten (schaltet die anderen aus).
2. Softscroll-Zeiger = »7«.
3. Spielstandsvariable = »0«
4. Sonstige Pointer und Variablen auf Ursprungswerte.
5. Bildschirm löschen, mit dem ersten Ausschnitt der Spiele-Landschaft füllen.
6. Bildschirm- und Rahmenfarben setzen.
7. Multicolor einschalten
8. Kollisionsregister löschen
9. Musikroutine vorbereiten
10. IRQ-Vektoren auf eigene Routine.

Im Programm geschieht dies durch die Zeilen 120 bis 140 (ab S. 32), wobei jeweils Unterroutinen aufgerufen werden.

Beispielsweise wird »INITIAL«, von Zeile 600 bis 860 nach jedem Spielende aufgerufen, wogegen IRQ (ab Zeile 880 nur zu Spielbeginn benötigt wird.

Der Spielablauf selbst gestaltet sich, dank der modularen Programmierung, noch einfacher:

Es ist die Routine »IRQNEU« von Zeile 2460 bis 2920 und ruft je nach Zustand der Spielevariablen entweder »WARTEN« (0), »BOING« (2) oder »OVER« (3) auf:

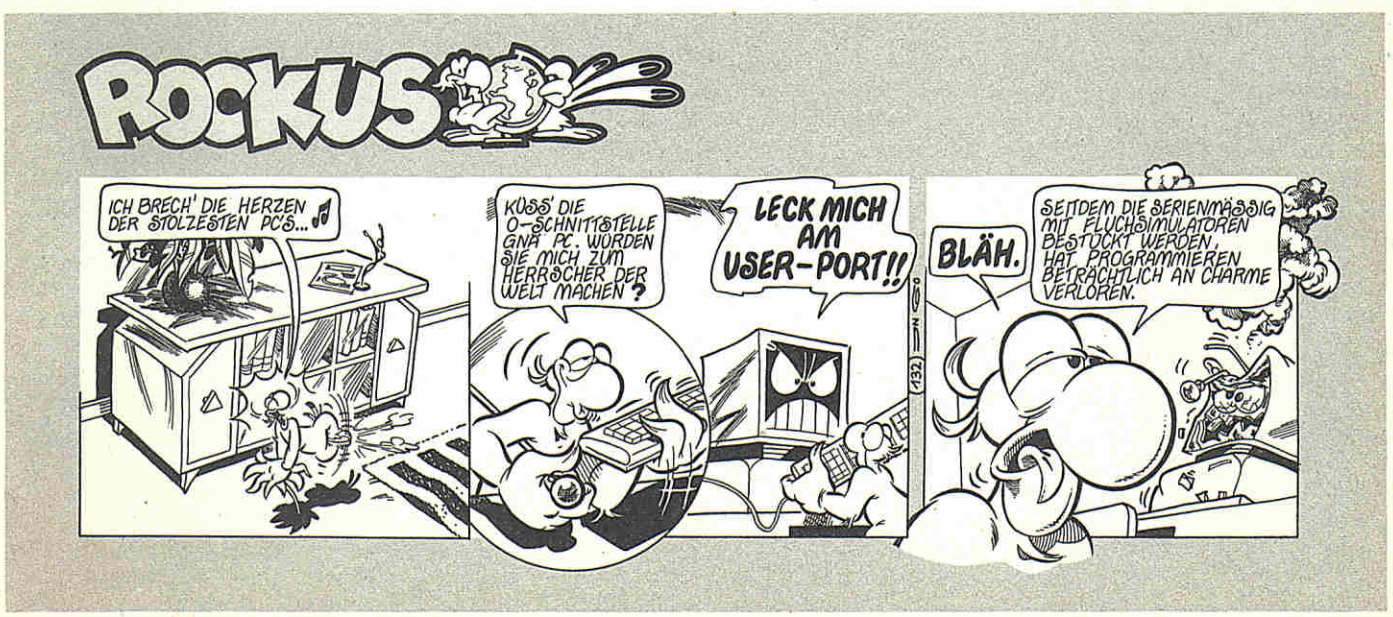
»WARTEN« ist die erste Routine, gibt den Text »Bitte Knopf drücken« (Text 1) aus und setzt nach »Feuer« die Spielvariable (Game, Zeile 1840) auf »1«.

»BOING« tritt nach einem verpatzten Spiel in Kraft, gibt sinngemäß den Text »daneben« aus und löscht ihn nach einer Weile wieder. Danach wird »INITIAL« aufgerufen, die Spielevariable auf »0« gesetzt und beim nächsten IRQ zwangsläufig »WARTEN« aufgerufen.

»OVER« gibt »gewonnen« aus, wartet eine Weile, löscht danach den Text wieder und ruft »INITIAL« auf. Danach steht die Spielevariable wieder auf »0« und beim nächsten IRQ geht's wieder zu »WARTEN«.

SPIEL Ist die Spielevariable jedoch »1«, ist das Spiel in vollem Gange und die eigentlichen Spieleroutinen »SCROLL«, »ANIMATION« und »JOYAB« finden Verwendung.

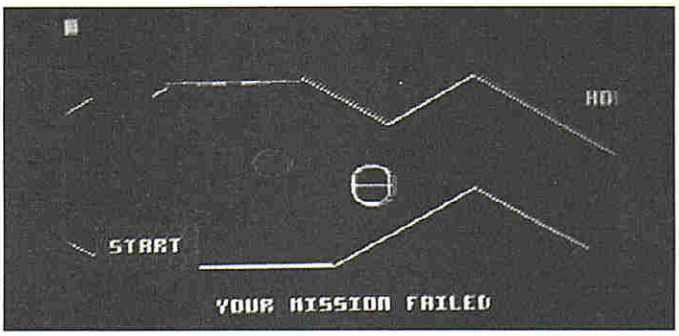
Das Demo-Spiel dient nur zur Anregung eigener, vielleicht besserer Routinen. Es lassen sich z.B. in der Funktion »WARTEN« Highscores anzeigen, oder in »OVER« der Highscore in eine Liste eintragen. »OVER« könnte auch den IRQ ausschalten, eine Kennvariable belegen und in Zeile 140 würde dann nicht RTS stehen, sondern wenn diese Variable gesetzt ist, würde der nächste Level nachgeladen, der IRQ wieder auf die eigene Routine verbogen und der neue Level neu initialisiert. Durch die modulare Programmierung steht einer Erweiterung des Spiels nichts mehr im Wege. Wenn Sie jetzt unser



Kurzinfo: DEMO

Programmart: DEMO-File zum Spielekurs
Spielziel: Manövrieren
Sie ohne Karambolage durchs Labyrinth
Laden: LOAD"DEMOGAME",8
Starten: nach dem Laden RUN eingeben, danach SYS 12288
Steuerung: Joystick in Port 2
Besonderheiten: gesamtes Spiel läuft im IRQ
Benötigte Blocks: 27 (gepackt)
Programmautoren: Andreas von Lepel/N. Heusler

[7] Das komplette Spiel läuft im IRQ ab. Daher blinkt auch der Cursor am Bildschirm.



kleines Demo ausprobieren wollen, sollten Sie den Computer kurz ausschalten und es danach nochmal laden:
`LOAD"DEMOGAME",8`

Jetzt entpacken Sie mit RUN und starten mit SYS12288 (Abb. 7). Warum trotz des Spielablaufs ein Cursor am Bildschirm blinkt, ist Ihnen sicherlich auch klar. (gr)

Das kommentierte Listing zum Kurs

```
110 V =53248
120 JSR INITIAL ; ADRESSEN INIT
125 JSR INITSPR
130 JSR IRQ     ; IRQ UMLENKEN
140 RTS
145 ;
150 SETSPRITE
151 LDA #130    ; SPRITE 0 AUF
160 STA V       ; POSITION (130/114)
170 LDA #114
180 STA V+1
190 LDA #0
200 STA V+16
210 STA V+29
220 STA V+23
225 LDA #2      ; ROT
230 STA V+39    ; SETZEN
240 LDA #5      ; GRUEN
250 STA V+37    ; LISTCOLOR1
260 LDA #32     ; SPRITEPOINTER
270 STA 2040
280 LDA #1      ; LISTCOLOR AN
290 STA V+28
300 STA V+21    ; SPRITE AN
310 RTS
315 ;
320 ANIMATION:LDA ZEIT
330 CMP #5
340 BEQ DOANIMA
350 INC ZEIT
360 RTS
370 DOANIMA:LDA #0
380 STA ZEIT
390 LDA DREHEN
400 BNE ZU
405 ;
410 AUF:LDA 2040
420 CMP #32
430 BNE AUF2
440 INC DREHEN
450 JMP ZU2
460 AUF2:DEC 2040
470 RTS
475 ;
480 ZU:LDA 2040
490 CMP #39
500 BNE ZU2
510 DEC DREHEN
520 LDA V+39
530 EOR #4
540 STA V+39
550 JMP AUF2
560 ZU2:INC 2040
570 RTS
580 DREHEN:BRK
590 ZEIT:BRK
595 ;
600 INITIAL:LDA #39 ; SPR.0 AUS BL 39
610 STA 2040    ;
620 LDA #0      ; NULL
650 STA 65      ; LB GRAFIKPOINT = 0
660 STA GAME    ; SPIELSTAND     = 0
670 LDA #7      ; SOFTSCROLLZEIGER
680 STA 2       ; AUF SIEBEN
690 LDA #200    ; VERZ. NACH SPIELENDE
700 STA DELAY   ; 200/25 SEK.
710 LDA #$70    ; HB GRAFIKPOINT =$70
720 STA 66      ; ($7000 = GRAFIK)
730 LDA #<1144  ; BEGINN BILDSCHIRM
740 STA 67      ;
750 LDA #>1144  ; 67/68 = 1144
760 STA 68
770 LDA #5      ;
772 STA 53281   ; FUER ALTE 64'ER
780 STA 646     ; UND CURSORFARBE
790 JSR $E544   ; SCNSLR1
791 ;
795 JSR FARBINI
796 ;
800 LDA 53270   ; 38 ZEICHEN PRO ZEILE
810 AND #247    ; %11110111
820 STA 53270
830 JSR RAMSCREEN ; GRAFIK -> SCHIRM
840 JSR SETSPRITE ; SPRITE 0 AN
842 LDA #0      ; SCHWARZ
844 STA 53280   ; -> RAHMEN
846 STA 53281   ; -> HINTERGRUND
850 LDA 53279   ; LOESCHEN KOLL SPR/HGR
855 LDA 53278   ;          KOLL SPR/SPR
860 RTS
861 ;
870 DELAY:BRK   ; VERZOEGERUNGSZAEHLER
880 IRQ:SEI     ; IRQ AUS
881 ;
885 ; ******************************
890 ; ** AN DIESE STELLE KOMMT DIE **
900 ; **     INITIALISIERUNG      **
910 ; **  FUER DIE MUSIK IM IRQ    **
920 ; ******************************
955 ;
960 LDA #<IRQNEU  ; IRQ LB AUF EIGENE
970 STA $314      ; ROUTINE
980 LDA #>IRQNEU  ; IRQ HB AUF EIGENE
990 STA $315      ; ROUTINE
1000 LDA #74    ; RASTERZEILE 74
1010 STA V+$12  ; INS RASTERREG.
1020 LDA V+$11  ; ZEILE < 255
1030 AND #$7F   ; %01111111 (S.TABELLE)
1040 STA V+$11  ; DAHER 9.BIT AUS
1050 LDA #$81   ; RASTERZEILENIRQ VIC
1060 STA V+$1A  ; ZULASSEN
1070 CLI        ; IRQ WIEDER AN
1080 RTS
1090 GAME:BRK   ; SPIELSTANDVARIABLE
1095 ;
1100 JOYAB:LDA #224 ; TASTATUR
1110 STA 56322  ; UMSCHALTEN
1120 LDA 56320  ; JOYPORT 2
1130 AND #1     ;= BIT 1 GESETZT (UP)
1140 BNE JOY2   ; JA, DANN WEITER
```

```
1150 DEC V+1     ; SPRITE Y-POS -1
1160 JOY2:LDA 56320
1170 AND #2      ;= BIT 2 GESETZT (DOWN)
1180 BNE JOY3    ; JA, DANN WEITER
1190 INC V+1     ; SPRITE Y-POS +1
1200 JOY3:LDA 56320
1210 AND #4      ;= BIT 3 GESETZT (LEFT)
1220 BNE JOY4    ; JA, WEITER
1230 LDA V       ; WENN X-POS = 32
1240 CMP #32     ; IGNORIEREN
1250 BEQ JOY4
1260 DEC V       ; SONST X-POS -1
1270 JOY4:LDA 56320
1280 AND #8      ;= BIT 4 GESETZT (RIGHT)
1290 BNE JOY5    ; JA, WEITER
1300 LDA V       ; WENN X-POS = 255
1310 CMP #255    ;
1320 BEQ JOY5    ; IGNORIEREN
1330 INC V       ; SONST X-POS +1
1340 JOY5:LDA #255 ;
1350 STA 56322   ; TASTATUR WIEDER EIN
1360 RTS
1365 ;
1370 RAMSCREEN:LDX #0     ; 13 ZEILEN
1380 RAMSCRL1:LDY #0      ; 40 ZEICHEN
1390 RAMSCRL2:LDA (65),Y  ; AUS
1400 STA (67),Y           ; GRAFIKBEREICH
1410 INY                  ; ZUM
1420 CPY #40              ; BILDSCHIRM
1430 BNE RAMSCRL2         ; KOPIEREN
1440 INC 66
1450 INC 66      ; GRAFIKVEKTOR IST
1460 LDA 67
1470 CLC         ; IN DER ZERONEW
1480 ADC #40
1490 STA 67      ; 65/66
1500 LDA 68      ; BILDSCHIRMVEKTOR IST
1510 ADC #0
1520 STA 68      ; 67/68
1530 INX
1540 CPX #13       ; SCHON 13 ZEILEN
1550 BNE RAMSCRL1  ; NEIN
1560 LDA 66        ; GRAFIKVEKTOR
1570 SEC
1580 SBC #26
1590 STA 66        ; UND
1600 LDA #<1144    ; BILDSCHIRMVEKTOR
1610 STA 67
1620 LDA #>1144
1630 STA 68        ; RESTAURIEREN
1640 RTS
1645 ;
1650 WARTEN:LDY #0 ; TEXT1 AUSGEBEN
1660 WARTEN1:LDA TEXT1,Y
1670 STA 1754,Y
1680 INY           ; AB BSCH POS 1754
1690 CPY #20       ; 20 ZEICHEN
1700 BNE WARTEN1
1710 LDA #224    ; TASTATUR UMSCHALTEN
1720 STA 56322   ;
1730 LDA 56320   ; = JOYKNOPF GEDR.
1740 CMP #111    ; %01101111
1750 BNE WARTENEND ; NEIN, ENDE
1760 LDA #255    ; TASTATUR
1770 STA 56322   ; EIN
1780 LDY #0      ; TEXT1 LOESCHEN
1790 LDA #32     ; = SPACES DRUEBER
1800 WARTEN2:STA 1754,Y
1810 INY
1820 CPY #20
1830 BNE WARTEN2
1840 INC GAME    ; SPIELSTAND AUF 1
1850 WARTENEND:RTS
1860 TEXT1 .8,9,20,32,20,18,9
1862 .7,7,5,18,32,20
1865 .15,32,19,20,1,18,20
1866 ;
1870 BOING:LDY #0 ; TEXT2 AUSGEBEN
1880 BO1:LDA TEXT2,Y
1890 STA 1754,Y
1900 INY
1910 CPY #20     ; 20 ZEICHEN
1920 BNE BO1
1930 DEC DELAY   ; WARTEN BIS
1940 LDA DELAY   ; VERZOEGERUNG
1950 BNE BOEND   ; BEI NULL
1960 LDY #0
1970 LDA #32     ; TEXT2 LOESCHEN
1980 BO2 STA 1754,Y
1990 INY
2000 CPY #20
2010 BNE BO2
2020 JSR INITIAL ; ALLES NEU INITIEREN
2025 JSR INITSPR
2030 BOEND:RTS
2040 TEXT2 .25,15,21,18,32,13,9
2041 .19,19,9,15,14,32
2042 .6,1,9,12,5,4,32
2043 ;
2050 OVER:LDY #0  ; TEXT3 AUSGEBEN
2060 OVERL1:LDA TEXT3,Y
2070 STA 1754,Y
2080 INY
2090 CPY #20       ; 20 ZEICHEN
2100 BNE OVERL1
2110 DEC DELAY     ; WARTEN BIS
2120 LDA DELAY     ; VERZOEGERUNG
2130 BNE OVEREND   ; NULL IST
2140 LDY #0
2150 LDA #32       ; TEXT LOESCHEN
2160 OVERL2:STA 1754,Y
2170 INY
2180 CPY #20
2190 BNE OVERL2
2200 JSR INITIAL   ; ALLES INITIALISIEREN (NEUSTART)
2205 JSR INITSPR
2210 OVEREND:RTS
2220 TEXT3 .25,15,21,32,13,1,4,5
2221 .32,9,20,44,15,12
2222 .4,32,2,15,25,44
2223 ;
2230 SCROLL:LDA 2  ; 7 PUNKTE LINKS
2240 BEQ BLKSCROLL ; HARDSCROLLING
2250 DEC 2         ; EINEN PUNKT NACH LINKS
2260 LDA 65        ; ABFRAGEN
2270 CMP #$D8      ; OB DIE GRAFIK AM ENDE
2280 BNE BLKEND
2290 LDA 66        ; ANGELANGT IST
2300 CMP #$71      ; (BEI $71D8)
2310 BNE BLKEND ; NEIN, WEITER
2320 INC GAME      ; GAME = GAME +2
2330 INC GAME
2340 BLKEND:RTS
2350 BLKSCROLL:LDA #7; VARIABLE SOFTSCR
2360 STA 2         ; AUF SIEBEN
2370 LDA 65        ; GRAFIKPOINTER +1
2380 CLC
2390 ADC #1
2400 STA 65
2410 LDA 66        ; HIGHBYTE
2420 ADC #0
2430 STA 66
2440 JSR RAMSCREEN ; TRANSFERROUTINE
2450 RTS
2455 ;
2460 IRQNEU:LDA V+$19 ; IRQ-REGISTER
2470 STA V+$19        ; LOESCHEN
2480 BMI RASTER ; WAR RASTER-IRQ!
2490 LDA $DC0D  ; SONST NORMALER IRQ
2492 ;
2493 ; **********************
2494 ; **   HIER AUFRUF    **
2495 ; ** AUFRUF DER MUSIK **
2496 ; **********************
2497 ;
2500 CLI         ; IRQ ZULASSEN
2510 JMP $EA31   ; ZUR SYSTEMROUTINE
2515 ;
2520 RASTER:LDA V+$12 ; RASTERIRQ
2530 CMP #178         ; BEI ZEILE 178
2540 BCS ZWEITER      ; JA, WEITER
2550 LDA 53270   ; AB ZEILE 74
2560 AND #248    ; UM DEN WERT IN
2570 ORA 2       ; SPEICHERZELLE 2
2580 STA 53270   ; NACH LINKS SCROLLEN
2590 LDA #5      ; RAHMENFARBE
2600 STA 53280   ; GRUEN
2601 LDA #12
2602 STA 53282
2605 JSR ENEMIES
2606 JSR ENEANI
2609 JSR CHARANI
2610 LDA #178    ; NEXT  IRQ ZEILE 178
2620 STA $D012
2630 JMP $FEBC   ; IRQ ENDE
2635 ;
2640 ZWEITER:LDA 53270 ; BILDSCHIRM AB
2650 AND #248          ; ZEILE 178
```


```
2660 ORA #7             ; NORMAL SCHALTEN
2670 STA 53270
2680 LDA #0      ; SCHWARZ
2690 STA 53280   ; RAHMENFARBE
2700 LDA GAME    ; SPIELSTAND
2710 BNE GAME1   ; NICHT NULL = WEITER
2720 JSR WARTEN  ; (SPIELBEGINN)
2730 JMP EXIT
2740 GAME1:CMP #1
2750 BNE GAME2   ; NICHT 1 = WEITER
2760 JSR SCROLL     ; SCROLLEN (SPIEL)
2770 JSR ANIMATION  ; SPRITE
2780 JSR JOYAB      ; JOYST. ABFRAGEN
2785 JSR ZAEHLEN
2790 LDA 53279   ; SPR/HGR KOLLISION
2791 AND #1
2800 BEQ NOBOING1; KEIN ZUSAMMENSTOSS
2810 INC GAME    ; SONST VERMERKEN
2811 JMP EXIT
2812 NOBOING1
2813 LDA 53278
2814 AND #1
2815 BEQ NOBOING
2816 INC GAME
2820 NOBOING:JMP EXIT
2825 ;
2830 GAME2:CMP #2
2840 BNE GAME3   ; NICHT 2 = WEITER
2850 JSR BOING   ; (KOLLISION)
2860 JMP EXIT
2865 ;
2870 GAME3 CMP #3
2880 BNE EXIT    ; NICHT 3, DANN ENDE
2890 JSR OVER    ; (SPIELENDE)
2900 EXIT:LDA #74 ; NAECHSTER IRQ
2910 STA V+$12    ; AB ZEILE 74
2920 JMP $FEBC   ; IRQ BEENDEN
2925 ;
2930 SPRITE:SEC   ;POSITIONSBERECHNUNG
2940 LDA POSENELO ; UND EIN/AUS
2950 SBC POSSCRLO
2960 TAY
2970 LDA POSENEHI
2980 SBC POSSCRHI
2990 TAX
3000 BEQ SPRITEON
3010 CMP #1
3020 BNE SPRITESAVE
3030 TYA
3040 CMP #89
3050 BCS SPRITESAVE
3060 LDA V+16
3070 ORA 820
3080 STA V+16
3090 JSR NUMMER
3100 LDA V+21
3110 ORA 820
3120 STA V+21
3130 LDA #1
3140 STA 821
3150 RTS
3155 ;
3160 SPRITESAVE
3161 LDX 820
3170 TXA
3180 EOR #255
3190 STA 820
3200 LDA V+16
3210 AND 820
3220 STX 820
3230 STA V+16
3240 LDX 820
3250 TXA
3260 EOR #255
3270 STA 820
3280 LDA V+21
3290 AND 820
3300 STX 820
3310 STA V+21
3320 LDA #0
3330 STA 821
3340 RTS
3345 ;
3350 SPRITEON:LDX 820
3360 TXA
3370 EOR #255
3380 STA 820
3390 LDA V+16
3400 AND 820
3410 STX 820
3420 STA V+16
3430 JSR NUMMER
3440 LDA V+21
3450 ORA 820
3460 STA V+21
3470 LDA #1
3480 STA 821
3490 RTS
3495 ;
3500 NUMMER:LDA 820
3510 CMP #2
3520 BNE NUMMER0
3530 STY V+2
3540 NUMMER0:CMP #4
3550 BNE NUMMER1
3560 STY V+4
3570 NUMMER1:CMP #8
3580 BNE NUMMER2
3590 STY V+6
3600 NUMMER2:CMP #16
3610 BNE NUMMER3
3620 STY V+8
3630 NUMMER3:CMP #32
3640 BNE NUMMER4
3650 STY V+10
3660 NUMMER4:CMP #64
3670 BNE NUMMER5
3680 STY V+12
3690 NUMMER5:CMP #128
3700 BNE NUMMER6
3710 STY V+14
3720 NUMMER6:RTS
3725 ;
3730 ENEMY1XLO:BRK ;DATEN FUER SPR-
3740 ENEMY1XHI:BRK ;POSITION
3750 ENEMY1Y  :BRK
3760 ENEMY1DIR:BRK
3770 ENEMY2XLO:BRK
3780 ENEMY2XHI:BRK
3790 ENEMY2Y  :BRK
3800 ENEMY2DIR:BRK
3810 ENEMY3XLO:BRK
3820 ENEMY3XHI:BRK
3830 ENEMY3Y  :BRK
3840 ENEMY3DIR:BRK
3850 ENEMY4XLO:BRK
3860 ENEMY4XHI:BRK
3870 ENEMY4Y  :BRK
3880 ENEMY4DIR:BRK
3890 ENEMY5XLO:BRK
3900 ENEMY5XHI:BRK
3910 ENEMY5Y  :BRK
3920 ENEMY5DIR:BRK
3930 ENEMY6XLO:BRK
3940 ENEMY6XHI:BRK
3950 ENEMY6Y  :BRK
3960 ENEMY6DIR:BRK
3970 ENEMY7XLO:BRK
3980 ENEMY7XHI:BRK
3990 ENEMY7Y  :BRK
4000 ENEMY7DIR:BRK
4010 POSSCRLO :BRK
4020 POSSCRHI :BRK
4030 POSENELO :BRK
4040 POSENEHI :BRK
4045 ;
4050 ZAEHLEN:LDA POSSCRLO
4060 CLC
4070 ADC #1
4080 STA POSSCRLO
4090 LDA POSSCRHI
4100 ADC #0
4110 STA POSSCRHI
4120 RTS
4130 ENEMIES:LDA #0
4140 STA 821
4150 LDA ENEMY1DIR
4160 BEQ LINKS1
4170 LDA ENEMY1XLO
4180 CLC
4190 ADC #1
4200 STA ENEMY1XLO
4210 LDA ENEMY1XHI
4220 ADC #0
4230 STA ENEMY1XHI
4240 LDA ENEMY1XLO
4250 CMP #70
4260 BNE VIEW1
```

```
4270 LDA ENEMY1XHI
4280 CMP #1
4290 BNE VIEW1
4300 DEC ENEMY1DIR
4310 JMP VIEW1
4320 LINKS1:LDA ENEMY1XLO
4330 SEC
4340 SBC #1
4350 STA ENEMY1XLO
4360 LDA ENEMY1XHI
4370 SBC #0
4380 STA ENEMY1XHI
4390 LDA ENEMY1XLO
4400 CMP #160
4410 BNE VIEW1
4420 LDA ENEMY1XHI
4430 CMP #0
4440 BNE VIEW1
4450 INC ENEMY1DIR
4460 VIEW1:LDA ENEMY1XLO
4470 STA POSENELO
4480 LDA ENEMY1XHI
4490 STA POSENEHI
4500 LDA #2
4510 STA 820
4520 JSR SPRITE
4530 LDA 821
4540 BEQ ENEMIES2
4550 LDA ENEMY1Y
4560 STA V+3
4570 LDA #1
4580 STA V+40
4590 ENEMIES2:LDA #0
4600 STA 821
4610 LDA ENEMY2DIR
4620 BEQ OBEN1
4630 INC ENEMY2Y
4640 LDA ENEMY2Y
4650 CMP #152
4660 BNE VIEW2
4670 DEC ENEMY2DIR
4680 JMP VIEW2
4690 OBEN1:DEC ENEMY2Y
4700 LDA ENEMY2Y
4710 CMP #96
4720 BNE VIEW2
4730 INC ENEMY2DIR
4740 VIEW2:LDA ENEMY2XLO
4750 STA POSENELO
4760 LDA ENEMY2XHI
4770 STA POSENEHI
4780 LDA #4
4790 STA 820
4800 JSR SPRITE
4810 LDA 821
4820 BEQ ENEMIES3
4830 LDA ENEMY2Y
4840 STA V+5
4850 LDA #1
4860 STA V+41
4870 ENEMIES3:LDA #0
4880 STA 821
4890 LDA ENEMY3DIR
4900 BEQ OBEN2
4910 INC ENEMY3Y
4920 LDA ENEMY3Y
4930 CMP #152
4940 BNE VIEW3
4950 DEC ENEMY3DIR
4960 JMP VIEW3
4970 OBEN2:DEC ENEMY3Y
4980 LDA ENEMY3Y
4990 CMP #96
5000 BNE VIEW3
5010 INC ENEMY3DIR
5020 VIEW3:LDA ENEMY3XLO
5030 STA POSENELO
5040 LDA ENEMY3XHI
5050 STA POSENEHI
5060 LDA #8
5070 STA 820
5080 JSR SPRITE
5090 LDA 821
5100 BEQ ENEMIES4
5110 LDA ENEMY3Y
5120 STA V+7
5130 LDA #1
5140 STA V+42
5150 ENEMIES4:LDA #0
5160 STA 821
5170 LDA ENEMY4DIR
5180 BEQ OBEN3
5190 INC ENEMY4Y
5200 LDA ENEMY4Y
5210 CMP #152
5220 BNE VIEW4
5230 DEC ENEMY4DIR
5240 JMP VIEW4
5250 OBEN3:DEC ENEMY4Y
5260 LDA ENEMY4Y
5270 CMP #96
5280 BNE VIEW4
5290 INC ENEMY4DIR
5300 VIEW4:LDA ENEMY4XLO
5310 STA POSENELO
5320 LDA ENEMY4XHI
5330 STA POSENEHI
5340 LDA #16
5350 STA 820
5360 JSR SPRITE
5370 LDA 821
5380 BEQ ENEMIES5
5390 LDA ENEMY4Y
5400 STA V+9
5410 LDA #1
5420 STA V+43
5430 ENEMIES5:LDA #0
5440 STA 821
5450 LDA ENEMY5DIR
5460 BEQ OBEN4
5470 INC ENEMY5Y
5480 LDA ENEMY5Y
5490 CMP #152
5500 BNE VIEW5
5510 DEC ENEMY5DIR
5520 JMP VIEW5
5530 OBEN4:DEC ENEMY5Y
5540 LDA ENEMY5Y
5550 CMP #96
5560 BNE VIEW5
5570 INC ENEMY5DIR
5580 VIEW5:LDA ENEMY5XLO
5590 STA POSENELO
5600 LDA ENEMY5XHI
5610 STA POSENEHI
5620 LDA #32
5630 STA 820
5640 JSR SPRITE
5650 LDA 821
5660 BEQ ENEMIES6
5670 LDA ENEMY5Y
5680 STA V+11
5690 LDA #1
5700 STA V+44
5710 ENEMIES6:LDA #0
5720 STA 821
5730 LDA ENEMY6DIR
5740 BEQ OBEN5
5750 INC ENEMY6Y
5760 LDA ENEMY6Y
5770 CMP #152
5780 BNE VIEW6
5790 DEC ENEMY6DIR
5800 JMP VIEW6
5810 OBEN5:DEC ENEMY6Y
5820 LDA ENEMY6Y
5830 CMP #96
5840 BNE VIEW6
5850 INC ENEMY6DIR
5860 VIEW6:LDA ENEMY6XLO
5870 STA POSENELO
5880 LDA ENEMY6XHI
5890 STA POSENEHI
5900 LDA #64
5910 STA 820
5920 JSR SPRITE
5930 LDA 821
5940 BEQ ENEMIES7
5950 LDA ENEMY6Y
5960 STA V+13
5970 LDA #1
5980 STA V+45
5990 ENEMIES7:LDA #0
6000 STA 821
6010 LDA ENEMY7DIR
6020 BEQ OBEN6
6030 INC ENEMY7Y
6040 LDA ENEMY7Y
6050 CMP #152
6060 BNE VIEW7
```


```
6070 DEC ENEMY7DIR
6080 JMP VIEW7
6090 OBEN6:DEC ENEMY7Y
6100 LDA ENEMY7Y
6110 CMP #96
6120 BNE VIEW7
6130 INC ENEMY7DIR
6140 VIEW7:LDA ENEMY7XLO
6150 STA POSENELO
6160 LDA ENEMY7XHI
6170 STA POSENEHI
6180 LDA #128
6190 STA 820
6200 JSR SPRITE
6210 LDA 821
6220 BEQ ENEMIES8
6230 LDA ENEMY7Y
6240 STA V+15
6250 LDA #1
6260 STA V+46
6270 ENEMIES8:RTS
6275 ;
6280 ENEANI:LDA ZEIT2 ;VERZOEGERUNG
6290 CMP #2           ;TESTEN (=2)
6300 BEQ DOANIMA2
6310 INC ZEIT2
6320 RTS
6330 DOANIMA2:LDA #0
6340 STA ZEIT2
6350 LDA DREHEN2      ;SPRITE ANIMATION
6360 BNE ZU3
6370 AUF3:LDA 2041
6380 CMP #40
6390 BNE AUF4
6400 INC DREHEN2
6410 JMP ZU4
6420 AUF4:DEC 2041    ;BLOCKPOINTER
6430 DEC 2042         ;ERNIEDRIGEN
6440 DEC 2043
6450 DEC 2044
6460 DEC 2045
6470 DEC 2046
6480 DEC 2047
6490 RTS
6495 ;
6500 ZU3:LDA 2041     ;2. PHASE
6510 CMP #47
6520 BNE ZU4
6530 DEC DREHEN2
6540 JMP AUF4
6550 ZU4:INC 2041     ;BLOCKPOINTER
6560 INC 2042         ;ERHOEHEN
6570 INC 2043
6580 INC 2044
6590 INC 2045
6600 INC 2046
6610 INC 2047
6620 RTS
6630 DREHEN2:BRK
6640 ZEIT2:  BRK
6645 ;
6650 INITSPR:LDA #0   ;BSCHPOSITION
6660 STA POSSCRLO     ;AUF NULL
6670 STA POSSCRHI
6680 LDA #43          ;POINTER FUER
6690 STA 2041         ;ENEMIES
6700 STA 2042         ;FESTLEGEN
6710 STA 2043
6720 STA 2044
6730 STA 2045
6740 STA 2046
6750 STA 2047
6760 LDA #16         ;POSITIONEN
6770 STA ENEMY1XLO   ;ENEMIES
6780 LDA #1
6790 STA ENEMY1XHI
6800 LDA #126
6810 STA ENEMY1Y
6820 LDA #12
6830 STA ENEMY2XLO
6840 LDA #3
6850 STA ENEMY2XHI
6860 LDA #100
6870 STA ENEMY2Y
6880 LDA #52
6890 STA ENEMY3XLO
6900 LDA #3
6910 STA ENEMY3XHI
6920 LDA #121
6930 STA ENEMY3Y
6940 LDA #92
6950 STA ENEMY4XLO
6960 LDA #3
6970 STA ENEMY4XHI
6980 LDA #142
6990 STA ENEMY4Y
7000 LDA #132
7010 STA ENEMY5XLO
7020 LDA #3
7030 STA ENEMY5XHI
7040 LDA #100
7050 STA ENEMY5Y
7060 LDA #172
7070 STA ENEMY6XLO
7080 LDA #3
7090 STA ENEMY6XHI
7100 LDA #121
7110 STA ENEMY6Y
7120 LDA #212
7130 STA ENEMY7XLO
7140 LDA #3
7150 STA ENEMY7XHI
7160 LDA #142
7170 STA ENEMY7Y
7180 RTS
7185 ;
8000 CHSAVE76:BRK    ;FREIHALTER FUER
8010 BRK             ;ZEICHENSATZ-
8020 BRK             ;ANIMATION
8030 BRK             ;16 BYTE = 2ZEICHEN
8040 BRK
8050 BRK
8060 BRK
8070 BRK
8080 CHSAVE85:BRK
8090 BRK
8100 BRK
8110 BRK
8120 BRK
8130 BRK
8140 BRK
8150 BRK
8151 CHARDELAY:BRK
8152 CHARANI:LDY #0  ;ZEICHENSATZ-
8161 INC CHARDELAY   ;ANIMATION
8162 LDA CHARDELAY
8163 CMP #5
8164 BEQ CHARGOON1
8165 RTS
8166 CHARGOON1:STY CHARDELAY
8167 LDA #0
8170 CHLOOP1:LDA 8800,Y
8180 STA CHSAVE76,Y
8190 LDA 8872,Y
8200 STA CHSAVE85,Y
8210 INY
8220 CPY #8
8230 BNE CHLOOP1
8240 LDY #135
8250 LDX #143
8260 CHLOOP2:LDA 8736,Y
8270 STA 8736,X
8280 DEX
8290 DEY
8300 CPY #255
8310 BNE CHLOOP2
8320 LDY #0
8330 CHLOOP3:LDA CHSAVE76,Y
8340 STA 8736,Y
8350 LDA CHSAVE85,Y
8360 STA 8808,Y
8372 INY
8380 CPY #9
8390 BNE CHLOOP3
8400 RTS
8500 FARBINI:LDA #12
8510 STA 53282
8520 LDA #11
8530 STA 53283
8540 LDA 53270
8550 ORA #16
8560 STA 53270
8565 LDA #24
8566 STA 53272
8570 LDY #0
8580 LDA #9
8590 FARBLOOP:STA 55296,Y
8600 STA 55546,Y
8610 STA 55796,Y
8620 STA 56041,Y
8630 INY
8640 BNE FARBLOOP
8650 RTS
8652 ENDE
```