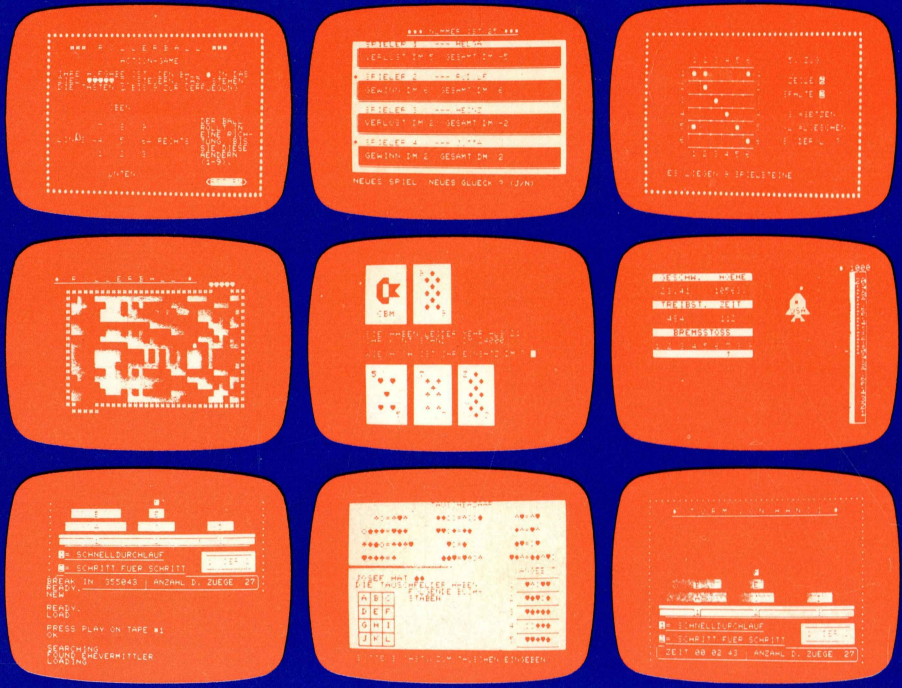
Andreas Dripke

23014

VC-20

Spiele-Buch 1

Programmieren mit unterhaltsamen Spielen für VC-20



COMPUTER LIFE™

Andreas Dripke

VC-20

Spiele—Buch I

Programmieren mit unterhaltsamen Spielen

Für die tatkräftige Hilfe beim Erstellen der Programme dieses Buches danke ich besonders Herrn Ralph Babel. Mein Dank gilt ebenfalls dem Verlag INTERFACE AGE für die Herausgabe des Werkes.

Wiesbaden, im Juni 1982

Der Autor

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Dripke Andreas:
VC-20 Spiele—Buch I: Programmieren mit unterhaltsamen Spielen.
Andreas Dripke.—München
Interface Age Verlagsgesellschaft
(Computerlife)
1. (1983)

ISBN 3-88623-014-7

ISBN 3-88623-014-7
1. Auflage 1983

Alle Rechte, auch die der Übersetzung, vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlages reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Der Verlag übernimmt keine Gewähr dafür, daß die beschriebenen Schaltungen, Baugruppen, Verfahren, Programme usw. funktionsfähig und frei von Schutzrechten Dritter sind.

Printed in the United States of America
Copyright 1983 by Interface Age Verlagsgesellschaft
München
Satz: Klaus Oberhofer und Partner, 6600 Reutte, Austria
Druck: Apollo Printing, Bellflower, Calif., USA

Inhaltsverzeichnis

	Seite
Einleitung	7
I. Teil	9
Kapitel 1: ROULETTE	11
Kapitel 2: ROLLERBALL	23
Kapitel 3: DREI AUS ACHTZEHN	33
Kapitel 4: PERNIX	43
Kapitel 5: ROBOTER	55
Kapitel 6: NUCLEAR	65
Kapitel 7: FLAK	77
Kapitel 8: STARIS	89
Kapitel 9: LIFE	103
II. Teil	113
Kapitel 10: WELTRAUMSCHLACHT	115
Kapitel 11: RECTUS	129
Kapitel 12: MOONRAKER	141
Kapitel 13: BREAKOUT	153
Kapitel 14: AUF LOS GEHT'S LOS	161
Kapitel 15: GIMMY-NIMM	173
Kapitel 16: BLACKJACK	185
Kapitel 17: TURM VON HANOI	195
Kapitel 18: EHEVERMITTLER	207
Anhang A: Wie lese ich ein Computerlisting	217
Anhang B: Commodore-Basic	221
Bonbon: MONSTER	241

EINLEITUNG

Dieses Buch enthält 18 Spielprogramme, die Ihnen Unterhaltung und Freude bereiten sollen. Es sind alles Programme, die eine Spitzenleistung in Programmiertechnik darstellen und die vom Computer gegebenen Möglichkeiten – besonders hinsichtlich Graphik, Farbe und Sound – voll ausnutzen. Alle Spiele wurden mit größter Sorgfalt erstellt und ausführlich getestet.
Der Sinn dieses Buches ist aber nicht nur, Ihnen eine Reihe faszinierender Spiele in die Hand zu geben, sondern Sie werden anhand der Spielprogramme nach und nach eine Fülle von Dingen über Ihren Computer erfahren. Die Grundidee lautet "spielend lernen", und das im wahrsten Sinne des Wortes.
Nach der Lektüre dieses Buches haben Sie also nicht nur Ihre Spielesammlung beträchtlich erweitert, sondern Sie haben gelernt, Spielprogramme selbst zu programmieren. Das dabei erworbene Wissen steht Ihnen natürlich auch bei anderen als bei Spielprogrammen zur Verfügung.
Alle Programme laufen auf dem Commodore "VolksComputer" VC 20. Die Spiele sind in zwei Teile gegliedert, 13 von ihnen sind für den VC 20 in der Grundversion, also mit 5 KByte freiem Speicherplatz geeignet, die restlichen 5 Spiele benötigen 8 KByte, also mindestens eine 3 KByte Speichererweiterung. Wenn Sie mehr Speicherkapazität zur Verfügung haben, können Sie die Programme nach Ihren Wünschen ausbauen.
In jedem Kapitel werden wir eine Reihe von Tricks besprechen. Sie sollten sich das Gesagte immer anhand des genannten Programms verdeutlichen und auch ein bißchen selbst damit experimentieren. Schließlich wollen wir ja keine graue Theorie betreiben, sondern Praxis. Genug geredet, fangen wir gleich mit dem ersten Spiel an!

I. TEIL

Die Propgramme dieses ersten Teils funktionieren auf einem VC 20 in der Grundkonfiguration mit 5 KByte. Irgenwelche Zusätze sind nicht erforderlich, im Einzelfall – Kassettenrekorder oder Floppydisk, Joystick – aber sinnvoll. Die Funktion eines Programmes wird durch das Fehlen solcher Zusätze aber in keiner Weise beeinflußt. Zusätzliche Speicherkapazität erlaubt Ihnen, die Spiele nach Ihren Wünschen weiter auszubauen.

1. ROULETTE

Beginnen wir unsere Spielesammlung mit einem Glücksspiel. Falls Sie schon neugierig darauf sind und sofort mit dem Eintippen des Programmes beginnnen möchten, so empfehle ich Ihnen, zunächst Anhang A "Wie lese ich ein Computerlisting" zu Rate zu ziehen. Sie finden in diesem Anhang insbesondere die Cursor- und Farbsteuerzeichen, die in praktisch jedem Listing vorkommen, erklärt. Sehen Sie sich diese Zeichen einmal in den Zeilen 1000 bis 1030 in ROULETTE an. Sie erkennen die Steuerzeichen sofort daran, daß sie die einzigen revers gedruckten Zeichen sind. Soviel vorab für diejenigen, die es nicht erwarten können. Wir wollen aber jetzt einmal gemeinsam sehen, was man an der Roulettesimulation alles lernen kann.
Die REM-Anweisung am Anfang des Programmes sind Ihnen wahrscheinlich nicht neu. REM ("remark" engl. = Bemerkung) dient dazu, Kommentare in ein Programm einzustreuen. Alles, was hinter REM steht, wird vom Computer beim Programmlauf gar nicht beachtet und dient ausschließlich dazu, einem menschlichen Leser das Verständnis des Programmes zu erleichtern. Wollen Sie der Übersichtlichkeit halber lediglich eine Leerzeile in Ihr Programm einbauen, dann zeigt Ihnen Zeile 9 in ROULETTE eine elegante Möglichkeit hierzu. Anstelle eines REM-Befehls genügt ein Doppelpunkt direkt am Zeilenanfang. Aber im Unterschied zu REM darf hinter diesem Doppelpunkt kein weiterer Kommentar folgen. Der Doppelpunkt dient in diesem Zusammenhang also wirklich nur dazu, eine Leerzeile zu erzeugen.
Die erste BASIC-Zeile, die der Computer auszuführen vermag, ist Zeile 100. Sie mag Ihnen auf den ersten Blick verwirrend erscheinen, sie ist aber äußerst wichtig und wird uns in fast jedem Programm dieses Buches begegnen. Der Bildschirm des VC 20 besteht aus 23 Zeilen mit jeweils 22 Zeichen, mithin also 506 Zeichen insgesamt. Nun müssen ja die Zeichen, die auf dem Bildschirm zu sehen sind, auch irgendwo im Computer selbst abgespeichert sein – schließlich kann die Maschine nicht auf dem Bildschirm nachsehen, was dort steht. Diese interne Speicherung des Bildschirminhaltes geschieht in zwei Speicherbereichen im Computer. Der eine Bereich hält fest, welches Zeichen wo auf dem Schirm steht und der andere Bereich, welche Farbe dieses Zeichen hat. An welchen Adressen diese beiden Bereiche liegen, hängt beim VC 20 von der Gesamtspeicherkapazität ab, also davon, wie-

viele bzw. welche Speichererweiterungsmodule Sie angeschlossen haben. Und genau das prüft Zeile 100. In Abhängigkeit davon, welche Kapazität gerade verfügbar ist, wird den beiden Variablen VB (Variable Bildschirm) und VC (Variable Color) die Anfangsadresse des Bildschirm- bzw. Farbbereiches zugeordnet. Sie können das sehr leicht überprüfen, indem Sie das Programm starten, es mit STOP unterbrechen und sich dann mit

```
PRINT VB,VC
```

die beiden Adresswerte ausgeben lassen. Probieren Sie, nachdem VB und VC nun die richtigen Werte haben, doch einmal folgende Befehlsfolge:

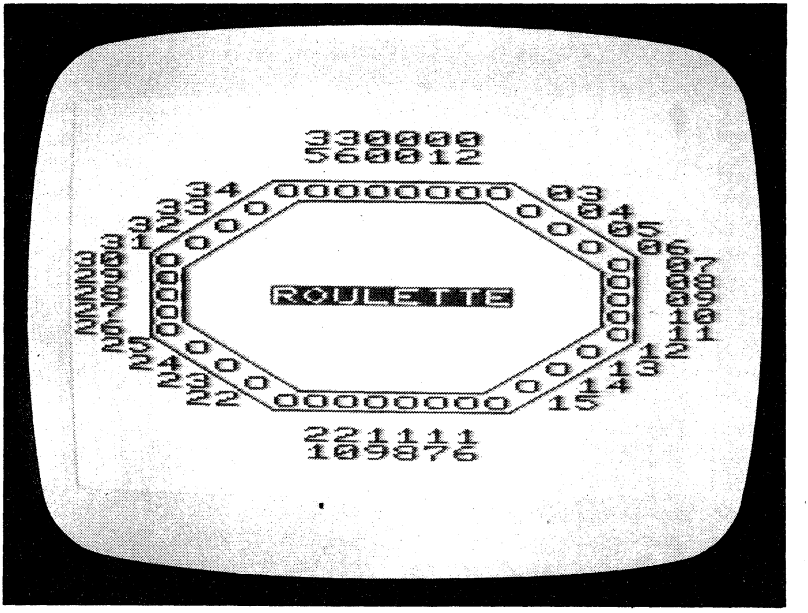
```
FORI=0TO505:POKEVB+I,42:POKEVC+I,6:NEXTI
```

Verblüffend, nicht wahr? Die Erklärung dafür ist einfach: 'POKEVB+I,42' setzt das Sternchen dadurch, daß sich I mit jedem Schleifendurchlauf um eins erhöht, jeweils eine Stelle weiter (42 ist der Bildschirmcode für das Sternchen) und 'POKEVC+I,6' legt die Farbe des jeweiligen Sternchens mit "blau" fest. Ersetzen Sie versuchsweise ruhig die 42 durch eine andere Zahl zwischen 0 und 255 und die 6 durch eine Zahl zwischen 0 und 7 (dies sind die acht möglichen Farben). Wir wollen unsere Befehlsfolge noch etwas abändern:

```
FORI=0TO505:POKEVB+I,42:POKEVB+I-1,32:POKEVC+I,6:NEXTI
```

Hierbei spielt es – genau wie beim letzten Beispiel – keine Rolle, daß sich die Anweisungsfolge über mehrere Bildschirmzeilen erstreckt, solange nicht insgesamt mehr als vier Bildschirmzeilen beansprucht werden. Bis zu vier Bildschirmzeilen vermag der VC 20 nämlich als logisch zusammengehörig zu erkennen. Doch probieren Sie nun aus, wie sich die Änderung ausgewirkt hat. Diesmal entsteht der Eindruck, daß das Sternchen sich über den Bildschirm bewegt. Dieser Effekt kommt dadurch zustande, daß 'POKEVB+I-1,32' das jeweils vorangehende Zeichen wieder löscht. 32 ist nämlich der Bildschirmcode für ein Leerzeichen (auch "space" oder "blank';).
Diese Möglichkeit, den Bildschirm nicht über PRINT, sondern mittels POKE direkt anzusprechen, bietet sich für eine ganze Reihe von Anwendungen an. In diesem Kapitel möchte ich Ihnen zeigen, wie man auf diese Art und Weise sehr einfach eine Umrahmung um den Bildschirm programmieren kann.

Betrachten Sie dazu die Routine (Routine meint hier Funktionseinheit) von Zeile 4000 bis 4020. In diesen drei Zeilen wird der Rahmen gezeichnet, der gleich bei Programmstart mit RUN erscheint. Jedes Programm – aber besonders ein Spielprogramm – sieht schöner aus, wenn der Bildschirm eine Umrahmung bekommt. Wie sieht das nun im einzelnen aus?



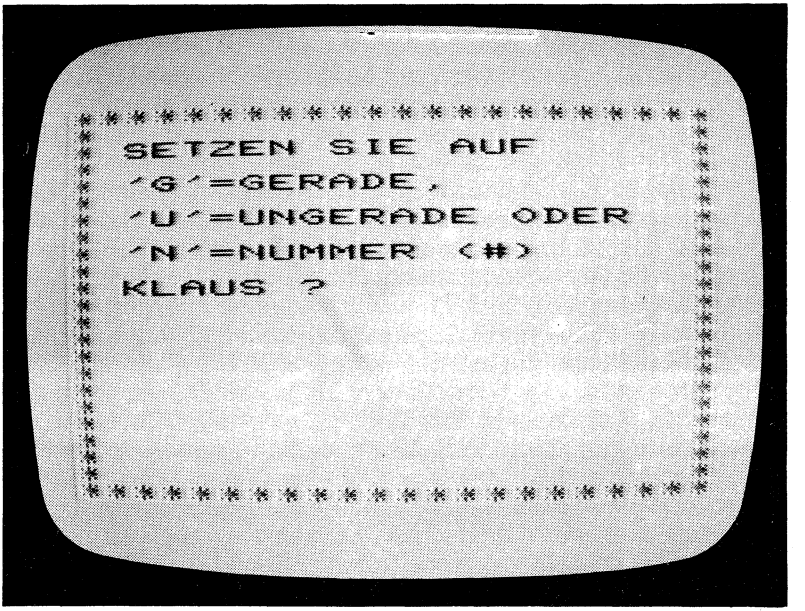
In Zeile 4000 wird zunächst durch den PRINT-Befehl der Bildschirm gelöscht. Sie erinnern sich, reverse Zeichen im Listing sind Steuerzeichen – dieses hier dient eben zum Bildschirmlöschen. Dann erfolgt, immer noch in Zeile 4000, die Ausgabe der oberen und unteren Bildschirmabgrenzung. Da die Bildschirmbreite 22 Zeichen beträgt, wird die Variable A von 0 bis 21 in einer FOR-NEXT-Schleife hochgezählt. Die Variablen VB und VC bezeichnen in uns schon bekannter Weise die Bildschirmposition links oben, mit immer größer werdendem A wird somit Zeichen für Zeichen nach rechts gerückt. Die Zahl 484 erscheint Ihnen vielleicht zunächst etwas unverständlich, sie ergibt sich aber einfach aus 506-22 für die untere Bildschirmabgrenzung.
Die Routine ist universell konzipiert, der Bildschirmcode kann mit der Variablen X, die Farbe mit der Variablen Y frei gewählt werden. Wie schon bei unserer Spielerei mit dem Sternchen,

so gilt auch hier, der Bildschirmcode für das Zeichen kann zwischen 0 und 255, der Farbcode zwischen 0 und 7 liegen. Andere Werte führen zu Fehlermeldungen bzw. Unsinn.
Der linke und rechte Teil der Bildschirmumrahmung wird in den Zeilen 4010 und 4020 gekennzeichnet. Diesmal wird A von 22 (erste Position der zweiten Zeile) bis 462 (erste Position der vorletzten Zeile) gezählt. Die Schrittweite mit 'STEP22' ergibt sich direkt aus der Schirmbreite von 22 Zeichen pro Zeile. Dies bedeutet nämlich, daß für jede neue Zeile um 22 Positionen weitergezählt werden muß.
'POKEVB+A,X' und 'POKEVC+A,Y' sind verantwortlich für den linken Teil des Rahmens, 'POKE+VB+21+A,X' und 'POKEVC+21+A,Y' für den rechten Teil. Und damit ist der Rahmen geschlossen.
Abhängig von den Werten in X und Y kann der Rahmen, der mit dieser Routine ausgegeben wird, ganz verschieden aussehen. Im Programm wird dies auch kräftig ausgenutzt. Da die Routine von verschiedenen anderen Stellen im Programm aus aufgerufen wird (mittels GOSUB), ist sie als Unterprogramm ausgelegt und schließt somit mit dem Befehl RETURN ab. Wir werden uns mit den Möglichkeiten der Unterprogrammtechnik bei anderen Programmen noch ausführlich befassen.
Eine andere wichtige Routine, die Sie auch in Ihren eigenen Programmen mit Gewinn einsetzen können, ist die Cursorsimulation. Sie liegt in den Zeilen 3000 bis 3020.
Der einfachste Befehl zur Eingabe ist 'INPUT"Text";H$'. Wenn man aber über das, was eingegeben wird, eine direkte Kontrolle haben will, dann muß 'GETA$' benutzt werden. Denn bei INPUT ist zunächst einmal jede beliebige Eingabe erlaubt und erst nach Drücken der RETURN-Taste kenn das Programm überhaupt feststellen, was eingetippt wurde. Bei GET ist die sofortige Prüfung möglich. Um z. B. einen Namen einzugeben, könnten wir folgende Routine verwenden:

```
10 GETA$:IFA$=""THEN10
20IFA$<"A"ORA$>"Z"THEN10
30 N$=N$+A$
```

Diese Routine verhindert, daß im Namensstring N$ ein anderes Zeichen steht als Buchstaben. Die Selektion erfolgt in Zeile 20.
Dem Vorteil der Kontrolle steht bei der Eingabe mittels GET aber auch ein Nachteil gegenüber: es gibt bei GET im Unterschied zu INPUT keinen blinkenden Cursor. Gerade der Cursor lenkt aber die Aufmerksamkeit eben auf die Eingabe, die zu tätigen ist. Möchte man hier das Optimum erreichen, dann

simuliert man bei GET ganz einfach einen Cursor. Und genau das tut die Routine bei 3000.
In Zeile 3000 wird ein eventuell gesetzter Cursor gelöscht, es folgt eine kurze Verzögerungsschleife (FOR-NEXT) und dann GET. Der Rest von Zeile 3000 (hinter IF) und Zeile 3010 dienen ausschließlich der Selektion, ähnlich wie in dem oben angegebenen Beispiel Zeile 20. Wichtig ist nun Zeile 3020. Hier wird das Cursorzeichen – das Karomuster – auf den Bildschirm gebracht. Direkt danach folgt im dem PRINT-Befehl die Steueranweisung 'Cursor nach links'. Ziehen Sie ggf. noch einmal Anhang A zu Rate, um das Steuerzeichen zu identifizieren. In den Zeilen 3000 und 3020 wird also abwechselnd das Cursorzeichen gelöscht bzw. gedruckt. Die Cursorposition wird durch das in beiden Fällen im PRINT-Befehl folgende 'Cursor nach links' und das am Ende der PRINTs stehende Semikolon nicht verändert. Somit ensteht der perfekte Eindruck des blinkenden Cursors. Als Cursorsymbol können Sie in Zeile 3020 übrigens jedes beliebige andere Zeichen einsetzen, probieren Sie es aus.
Wir wollen jetzt etwas näher auf die Selektion in 3000 und 3010 eingehen. In beiden Zeilen wird mittels IF...THEN nach bestimmten Bedingungen abgefragt und falls diese Bedingungen erfüllt sind, erfolgt der Aussprung aus der gesamten Routine durch RETURN. Zeile 3000 prüft, ob die Eingabevariable G$ gleich CHR$(13) oder CHR$(20) ist. Um das zu verstehen, müssen Sie wissen, daß 13 der Code für die RETURN-Taste und 20 der Code für die Löschtaste DEL ist. Falls nämlich eine dieser beiden Tasten gedrückt wurde, so will der Benutzer entweder die Eingabe beenden (RETURN) oder das vorangegangene Zeichen löschen (DEL). In beiden Fällen muß die Eingaberoutine verlassen werden, damit das Programm gesondert auf diese Kontrolltasten reagieren kann. Zeile 3010 überprüft, ob ein sonstiges gültiges Zeichen eingetippt wurde. Gültig sind hierbei alle Zeichen zwischen "A" und "Z", also alle Buchstaben, der Bindestrich "–" und alle Ziffern zwischen 1 und 5. Wird eine dieser Tasten gedrückt, so folgt der Rücksprung in den jeweilig aufgerufenen Programmteil mittels RETURN. Dort erfolgt dann meist noch eine weitere Selektion. So springt z. B. Zeile 1080 die GET-Routine mit 'GOSUB3000' an, testet, ob die Eingabe zwischen 1 und 5 liegt und falls dem so ist, so wird die Zahl durch 'PRINTG$' ausgedruckt. Auf die Konstruktion 'ON–(..)GOTO' werden wir noch in kommenden Programmen zu sprechen kommen. In Ihren eigenen Programmen können Sie zusammen mit GET natürlich noch weitere Selektionen einbauen, so daß z. B. bei einer Ja/Nein-Frage nur "J" und "N" angenommen wird.



Wenden wir uns nun einer völlig anderen Sache zu. Wie Sie sicher wissen, besitzt der VC 20 zwei Bildschirmmodi, nämlich einmal den Graphik- und zum anderen den Groß-/Kleinschreibmodus. Wir können zwischen diesen beiden Modi hin- und herschalten durch gleichzeitiges Betätigen von SHIFT und der COMMODORE-Taste. Oftmals ist es aber wichtig, auch vom Programm aus durch einen Befehl den gewünschten Modus einzuschalten. Dies geschieht in ROULETTE z. B. in Zeile 1000.
Durch

PRINTCHR$(14)

wird auf Groß-/Kleinschreibung geschaltet. Probieren Sie es aus, indem Sie per Hand wie oben beschrieben auf Graphik schalten und dann den genannten Befehl eingeben. Um wieder zum Graphikmodus zurückzukommen, steht

PRINTCHR$(142)

zur Verfügung. Testen Sie auch dies aus.
Wenn nun ein Programm mit einem der beiden PRINT-Befehle einen bestimmten Modus wählt, dann ist es natürlich nicht wünschenswert, daß der Benutzer – vielleicht versehentlich –

in den jeweils anderen Modus durch Drücken von SHIFT und COMMODORE schaltet. Wie läßt sich dem vorbeugen? Nun, sehen Sie sich einmal den zweiten Befehl in Zeile 1000 an.

```
POKE657,128
```

blockiert das Umschalten von der Tastatur aus. Der Modus kann jetzt nur noch mit den genannten PRINT-Anweisungen geändert werden, also bei laufendem Programm nur noch durch das Programm. Um diese Tastaturblockade wieder aufzuheben, geben Sie einfach

```
POKE657,0
```

ein. Danach ist die Umschaltung wieder problemlos mit SHIFT und COMMODRE möglich. Eigentlich recht einfach, nicht wahr? Man muß es eben nur wissen.
Übrigens, es ist wichtig, daß Sie in Ihren Programmen immmer gleich am Anfang den gewünschten Bildschirmmodus einschalten. Sonst erscheint der in Groß-/Kleinschreibung gedachte Text u. U. als eine wirre Folge von Graphikzeichen oder statt einer ordentlichen Kugel rollt bei ROULETTE ein kleines "Q". Vergessen Sie daher nie den entsprechenden PRINT-Befehl am Anfang Ihrer Programme und blockieren Sie am besten auch gleich die Tastaturumschaltung wie gezeigt!
Eine andere, gerade bei Glücksspielen wichtige Sache wollen wir auch noch kennenlernen. Schauen Sie sich dazu in ROULETTE Zeile 1400 an. Mit Hilfe der RND-Funktion lassen sich Zufallszahlen erzeugen. Eigentlich sollte ich besser Pseudozufallszahlen sagen, denn so zufällig sind die Zahlen gar nicht – schließlich werden Sie intern im Computer nach genau vorgegebenen Regeln erzeugt. Immerhin sind die Zahlenfolgen zufällig genug, um den meisten daraufprüfenden statistischen Methoden zu genügen.
Allgemein erzeugt RND Zufallszahlen zwischen 0 (einschließlich) und 1 (ausschließlich). Sie sehen das sehr leicht, wenn Sie einmal folgenden Einzeiler ausprobieren:

```
10 X=RND(1):PRINTX:GOTO10
```

Das Argument zu RND, also in diesem Beispiel die 1 in Klammern, hat keinen Einfluß, es kann sich um eine beliebige positive Zahl handeln.
Wie sich Zufallszahlen in anderen Bereichen als nur zwischen 0 und 1 erzeugen lassen, zeigt Ihnen Zeile 1400 in ROULETTE. Hier werden Zahlen zwischen 0 und 2 bzw.

zwischen 0 und 36 (nicht 37) generiert. Die 37 wird deswegen nie erreicht, weil RND(1) niemals genau 1 werden kann. Ganz allgemein gilt:

```
X=INT(A*RND(1)+B)
```

erzeugt eine positive ganze Zahl zwischen B (untere Grenze) und A+B–1 (obere Grenze). Läßt man B weg, dann liegt der Bereich zwischen 0 und A–1.
Die Funktion INT dient dazu, den Nachkommateil der entstehenden Zahl abzuschneiden. Ohne sie entstehen Dezimalzahlen.
Nun ließe sich noch viel zu dem Roulette-Programm sagen.
Aber Sie haben ja ein Spielebuch gekauft und da wollen Sie nicht nur lernen, sondern auch spielen. Wenn Sie es noch nicht getan haben, dann sollten Sie jetzt in Ruhe das Programm eintippen und dann als Belohnung sich und Ihre Familie oder Freunde in Ihre private Spielbank einladen.

Zusammenfassung:
Am Ende dieses ersten Kapitels sollten Sie außer einem faszinierenden Spiel auch einiges gelernt haben. Hier sind noch einmal die Lerninhalte in Kürze:
- Doppelpunkt zum Einstreuen von Leerzeilen in ein Programm
- Berechnung der Anfangsadresse des Bildschirm- und Farbspeicherbereiches
- Programmierung eines Rahmens (experimentieren Sie hier selbst ein bißchen!)
- Cursorsimulation mit beliebigem Zeichen bei GET
- Umschaltung Graphikzeichen/Kleinschreibung mit PRINT und Blockade der entsprechenen Tastaturumstellung
- Erzeugung von Zufallszahlen

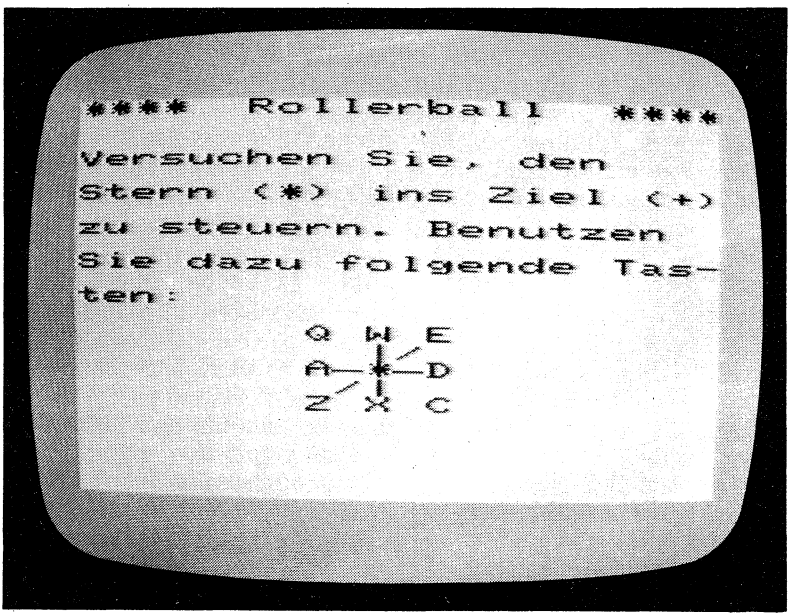
```
1 REM ---------------
2 REM
3 REM -- ROULETTE --
4 REM
5 REM ---------------
6 REM
7 REM COPYRIGHT 1982
8 REM ANDREAS DRIPKE
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 PRINTCHR$(14):POKE657,128:POKE36879,123:X=42:Y=2:GOSUB4000:A$="■■■■■■■■■■■■"
1010 FORA=0TO29:PRINTA$:B$="■■■■■■■■■■■■
```

```
1180 PRINTMID$("S",1-(RIGHT$(N$(A),1)="S"),1)" EINSATZ
1190 PRINT"■■                    ":PRINT"■■? ";:INPUT#1,A$
1200 PRINT:V=VAL(A$):ON-(V>=1ANDV<=25000)GOTO1230:IFV>25000GOTO1220
1210 PRINT"■■MINIMAL DM 1!       ■":FORB=1TO2E3:NEXT:GOTO1190
1220 PRINT"■■HAUSLIMIT DM 25000!■":FORB=1TO2E3:NEXT:GOTO1190
1230 E(A)=V:NEXT:CLOSE1:X=42:Y=4:GOSUB4000:PRINT"■■SETZEN SIE AUF
1240 PRINT"■■'G'=GERADE,":PRINT"■■'U'=UNGERADE ODER":PRINT"■■'N'=NUMMER (#)
1250 FORA=0TOG:PRINT
1260 PRINT"■■"N$(A)"?           ■■■■■■■■■";
1270 GOSUB3000:IFG$="G"THENPRINT"GERADE":V=-2:GOTO1320
1280 IFG$="U"THENPRINT"UNGERADE":V=-1:GOTO1320
1290 IFG$<>"N"GOTO1270
1300 PRINT"#: ";:OPEN1,0:INPUT#1,A$:PRINT:CLOSE1:V=VAL(A$)
1310 IFV<1ORV>36THENPRINT"■";:GOTO1260
1320 S%(A)=V:NEXT:PRINT"■■■■_____":PRINT"■■■RIEN NE VA PLUS
1330 FORA=1TO3E3:NEXT:PRINT"■■"SPC(52)"330000"SPC(16)"560012"SPC(15)"_______
1340 PRINTSPC(4)"34/oooooooo\03"SPC(7)"33/o/_____\o\04":PRINT"  32/o/"SPC(8);
1350 PRINT"\o\05":PRINT" 31/o/"SPC(10)"\o\06":PRINT"30 |o/"SPC(12)"\o| 0729 |o| ";
1360 PRINT"  ■■_____■■   |o| 0828 |o|   ■■ROULETTE■■   |o| 0927 |o| "SPC(12);
1370 PRINT" |o| 1026 |o\"SPC(12)"/o| 11  25\o\"SPC(10)"/o/12":PRINT"  24\o\"SPC(8);
1380 PRINT"/o/13":PRINT"   23\o\_____/o/14":PRINT"    22\oooooooo/15
1390 PRINTSPC(7)"_____":PRINTSPC(8)"221111":POKE36878,8:PRINTSPC(8)"109876
1400 W=0:U=3*RND(1):V=INT(37*RND(1)):FORA=0TOU:FORB=0TO36:P=81:GOSUB2000
1410 FORC=0TOW:NEXT:P=87:GOSUB2000:NEXT:NEXT:FORB=0TOV:P=81:GOSUB2000
1420 FORC=0TOW:NEXT:P=87:GOSUB2000:NEXT:B=B-1:P=81:GOSUB2000:POKE36878,0
1430 FORA=1TO2E3:NEXT:W=2*(V/2-INT(V/2)):FORA=0TOG:IFS%(A)<>-2GOTO1450
1440 ON-(W=0)GOTO1480:GOTO1490
1450 IFS%(A)<>-1GOTO1470
```

```
1460 ONWGOTO1480:GOTO1490
1470 ON-(V<>S%(A))GOTO1490:V(A)=E(A)*37:GOTO1500
1480 V(A)=E(A)*2:GOTO1500
1490 V(A)=-E(A):GOTO1500
1500 G(A)=G(A)+V(A):NEXT
1510 PRINT"■■*** NUMMER"V"***■":FORA=0TOG:PRINT"_____";
1520 PRINT"■SPIELER"A+1"■: "N$(A);:FORB=1TO11-LEN(N$(A)):PRINT" ";:NEXT
1530 PRINT"■";:IFV(A)<0THENPRINT"VERLUST   :";-V(A):GOTO1550
1540 PRINT"GEWINN    :"V(A)
1550 PRINT"GESAMT    :"G(A):NEXT:PRINTSPC(38)"RETUN■■R";
1560 GETA$:ON-(A$<>CHR$(13))GOTO1560:PRINT"■NEUES SPIEL,
1570 PRINT"■NEUES GLUECK? (J/N)
1580 GETA$:ON-(A$="J")GOTO1160:IFA$<>"N"GOTO1580
1590 PRINT"■■DIE SPIELBANK":PRINT"■SCHLIESST FUER HEUTE.":POKE657,0:END
2000 POKEVB+117+X%(B),P:W=W+.4:POKE36876,254-W:RETURN
3000 PRINT" ■";:FORB=0TO99:NEXT:GETG$:IFG$=CHR$(13)ORG$=CHR$(20)THENRETURN
3010 IFG$>="A"ANDG$<="Z"ORG$="-"ORG$>"0"ANDG$<"6"THENRETURN
3020 PRINT"■■";:FORB=0TO99:NEXT:GOTO3000
4000 PRINT"■":FORA=0TO21:POKEVB+A,X:POKEVC+A,Y:POKEVB+484+A,X:POKEVC+484+A,Y
4010 NEXT:FORA=22TO462STEP22:POKEVB+A,X:POKEVC+A,Y:POKEVB+21+A,X
4020 POKEVC+21+A,Y:NEXT:RETURN
```

2. ROLLERBALL

Hier handelt es sich um ein Graphik-Spiel, bei dem es auf Schnelligkeit und Geschicklichkeit ankommt. Während bei ROULETTE keine Erklärung zum Spiel nötig war, ist dies bei ROLLERBALL unbedingt wichtig. Bedenken Sie bei Ihren eigenen Spielprogrammen immer, daß diese für jemanden anderen nur von Nutzen sind, wenn er weiß, wie sie zu handhaben sind. Wenn möglich, sollten Sie alles, was zum Verständnis des Programmes wichtig ist, in das Programm selbst hineinschreiben. Ist dies aus Speicherplatzgründen nicht möglich, so bietet es sich an, zwei getrennte Programme zu schreiben, von denen das eine nur die Aufgabe hat, die Funktion des anderen zu erklären. Auch dies wird uns im Verlaufe des Buches noch begegnen.
Doch wenden wir uns jetzt ROLLERBALL selbst zu!



Fangen wir auch diesmal wieder mit der ersten Zeile – abgesehen von dem REM-Zeilen – an. In Zeile 100 prüft das Programm, ob es korrekt von Kassette oder Diskette eingeladen wurde. Ist dies nicht der Fall, so wird "?LADE-FEHLER" ausgedruckt und der Programmlauf abgebrochen.

Wie läßt sich nun feststellen, ob ein Programm fehlerfrei eingeladen wurde oder nicht? Da ist einmal Speicherzelle 158. Unter dieser Adresse läßt sich nach dem Laden mit PEEK die Anzahl der aufgetretenen Ladefehler feststellen. Ist also der Inhalt dieser Speicherstelle größer als Null, so ist beim Einlesen des Programmes mindestens ein Fehler übertragen worden, ROLLERBALL druckt einen ?LADEFEHLER aus.
Eine andere Möglichkeit, einen eventuellen Fehler beim Einladen eines Programmes festzustellen, ist die Abfrage der Statusvariablen ST. Die Variable ST hat im Commodore BASIC eine ganz besondere Bedeutung, Sie sollten diese Variable also nie in Ihren eigenen Programmen verwenden, es sei denn, eben für den gedachten Statuszweck. Status bedeutet hierbei einfach, daß die Variable ST bestimmte genau definierte Werte annimmt, und zwar in Abhängigkeit davon, wie sich einzelne Peripheriegeräte verhalten – also z. B. der Rekorder, von dem wir ein Programm einlesen. Wir wollen uns hier nicht im einzelnen mit ST beschäftigen, sondern uns nur merken, daß diese Variable nach dem Einladen eines Programmes grundsätzlich Null zu sein hat, anderenfalls ist irgendein Lesefehler aufgetreten. Außerdem sollten Sie in Erinnerung behalten, daß der Variablenname ST für den genannten besonderen Zweck reserviert ist. Wenn Sie nämlich ST in Ihre Programme einbauen, ohne dies zu wissen, so kann es zu Fehlern kommen, an denen Sie als Programmierer schier verzweifeln – und das nur, weil Ihnen diese einzige, aber wichtige Information fehlt.
Unsere Zeile 100 benutzt übrigens beide genannten Möglichkeiten, um einen möglichen Ladefehler zu entdecken. Sicher ist sicher.
ROLLERBALL ist ein Graphikspiel. Und allen Graphikspielen ist gemeinsam, daß sich etwas auf dem Bildschirm bewegt. Im letzten Kapitel haben wir bereits einmal ein Sternchen über den Bildschirm sausen sehen und wir wollen jetzt lernen, jede beliebige Bewegung zu programmieren.
Wir wissen schon, daß jedem der 506 Punkte des Bildschirms eine bestimmte Adresse zugeordnet ist. Dabei ist die niedrigste Adresse der Punkt oben links und die höchste Adresse der Punkt unten rechts. Die exakten Werte dieser Adressen hängen von der Ausbaustufe des Speicherplatzes ab. In der Grundversion des VC 20 mit 5 KByte freiem Speicherplatz ist die niedrigste Adresse 7680 und die höchste Adresse 8185. Um nun z. B. unser Sternchen auf die 12. Position der 6. Zeile des Bildschirms zu setzen, tippen wir: 'POKE 7680+5*22+12,81'. Wir fangen also links

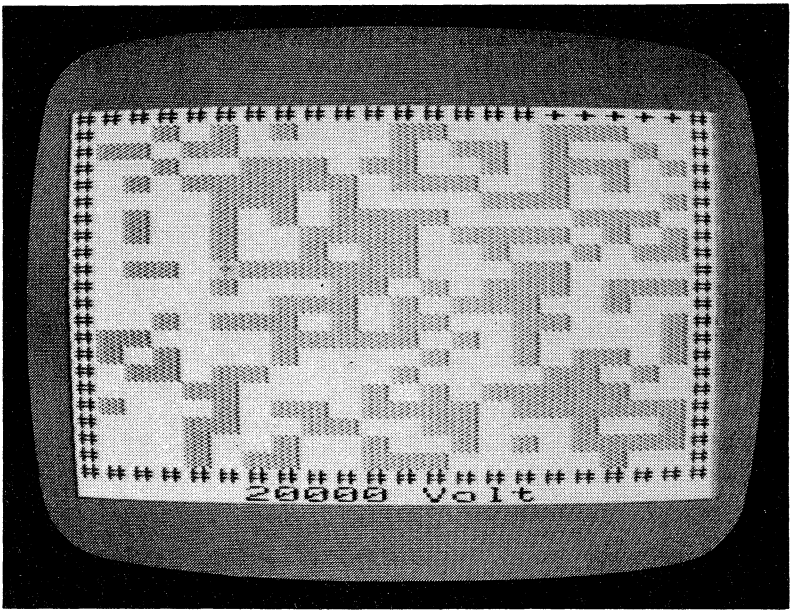
oben an zu zählen, addieren 5 Zeilen à 22 Zeichen und dann noch 12 Zeichen von links. Beachten Sie, daß die Adressen fortlaufend sind, d. h., wenn unser Sternchen am rechten Rand angekommen ist, dann wird es, wenn es eins weiter gesetzt wird, wieder am linken Rand einer Zeile tiefer erscheinen.
Nun ist aber für den Computer nicht nur wichtig, wie das Zeichen aussehen und an welcher Bildschirmposition es stehen soll, sondern auch seine Farbe. Wird nämlich keine bestimmte Farbe gewählt, so bekommt das Zeichen automatisch dieselbe Farbe, wie sie der Rest des Bildschirmes hat – mit der Folge, daß das Zeichen gar nicht sichtbar ist. Oder vermögen Sie einen schwarzen Stern auf einem schwarzen Untergrund zu erkennen? Auch die 506 Farbadressen laufen von links oben nach rechts unten. Die genauen Werte hängen auch hier vom Gesamtspeicherplatz ab. In der VC 20 Grundversion mit 5 KByte liegen die Adressen von 38400 (links oben) bis 38905 (rechts unten).
Damit Sie diesen ganzen Komplex besser verstehen lernen, sollten Sie jetzt einmal das folgende kurze Programm eintippen:

```
10 INPUT"POSITION";P
20 INPUT"FARBE";F
30 PRINT"Bildschirm löschen"
40 POKE 7680+P,42
50 POKE 38400+P,F
60 GOTO 10
```

Bevor Sie dieses Programm ausprobieren, sollten Sie evtl. Speichereinheiten von Ihrem VC 20 abnehmen, da das Programm die Adressen für die Grundversion benutzt. Geben Sie als Position eine Zahl zwischen 0 und 505 und als Farbe einen Wert zwischen 0 und 7 ein. Die Farbcodierung ist recht einfach: Die Zusammengehörigkeit von bestimmten Farben zu den Ziffern 1 bis 9 auf der Tastatur des VC 20 verschiebt sich genau um eins nach unten, so daß die Reihe eben bei POKE von 0 bis 8 statt von 1 bis 9 geht.
Nachdem wir nun die Bildschirmpunkte und die Farben mittels POKE ansprechen können, müssen wir uns noch über den Code der Zeichen unterhalten. In unserem Beispiel ist dieser Code 42, was – wie wir mittlerweile wissen – dem Sternchen entspricht. Will man einen bestimmten Punkt löschen, dann ist 32 der richtige Code. Weitere Zeichen sind: 1...26 = Alphabet A...Z, 48...57 = Ziffern 0 bis 9, 160 = volles Feld.
Dieser Code entspricht nicht dem normalen 7-Bit ASCII-Code

und kann daher auch nicht mit ASC(..) abgefragt werden. Es ist vielmehr ein 6-Bit ASCII-Code. Sie können ihn durch Erweiterung unserer Routine erforschen:

```
10 INPUT"POSITION";P
15 INPUT"ZEICHENCODE";C
20 INPUT"FARBE";F
30 PRINT"Bildschirm löschen"
40 POKE 7680+P,C
50 POKE 38400+P,F
60 GOTO 10
```



Auch dieses Programm ist für die VC 20 Grundkonfiguration gedacht. In Zeile 40 wird der entsprechende Bildschirmpunkt mit dem Zeichen C belegt und die Farbe in Zeile 50 gewählt. Probieren Sie für C Werte zwischen 0 und 255 aus.
Wir waren bisher immer davon ausgegangen, daß der VC 20 in der Grundversion mit 5 KByte vorliegt. Nun wollen wir klarstellen, welche Bildschirm- und Farbadressen bei wieviel Speicherplatz gültig sind. Eigentlich sind die Regeln ganz einfach. Solange die Speicherkapazität insgesamt nicht mehr als 8 KByte beträgt (mit anderen Worten: solange nicht mehr als eine 3 KByte Erweiterung eingesteckt wird), liegen die Adressen wie folgt vor:

Anfangsadresse des Bildschirmbereiches: 7680
Anfangsadresse des Farbbereiches: 38400
Sobald die Gesamtkapazität die 8 KByte überschreitet, verschieben sich diese Adressen auf folgende Werte:
Anfangsadresse des Bildschirmbereichs: 4096
Anfangsadresse des Farbbereichs: 37888
Damit nun ein Programm universell auf allen VC 20 Konfigurationen einsetzbar ist, muß es diese Adressen selbstständig berechnen. Wie Sie bereits aus dem letzten Kapitel wissen, dient dazu in ROLLERBALL Zeile 110. Den Variablen VB und VC wird die entsprechende Anfangsadresse zugewiesen.
So, jetzt haben wir eigentlich alles besprochen, was zur Simulation von Bewegung auf dem Bildschirm Ihres Computers nötig ist. Sie sollten nun an jedem beliebigen Punkt jedes beliebige Zeichen mit der von Ihnen gewünschten Farbe setzen und auch wieder löschen können. Um Ihnen die Arbeit zu erleichtern, habe ich Ihnen ein Bildschirmentwurfsblatt zusammengestellt. Daraus sind sämtliche Adressen leicht ersichtlich. Sie finden es im Anschluß an den Text.
Nachstehend habe ich Ihnen noch ein paar Einzeiler aufgelistet, um die Möglichkeiten zu zeigen (Bedenken Sie dabei immer, daß eine Zeile 22 Zeichen breit ist). Wenn Ihr VC 20 insgesamt mehr als 8 KByte Speicher hat, so ersetzen Sie die Adressen entsprechend.

```
FORI=0TO505:POKE7680+I,81:POKE38400+I,2:NEXTI
FORI=0TO22:POKE7680+I*22,81:POKE38400+I*22,6:NEXTI
FORI=0TO505STEP22:POKE7680+I,81:POKE38400+I,6:NEXTI
FORI=0TO505STEP23:POKE7680+I,81:POKE38400+I,6:NEXTI
FORI=505TO0STEP-21:POKE7680+I,81:POKE38400+I,4:NEXTI
```

ACHTUNG: Nehmen Sie als Adressen für POKE nur Zahlen, die im Bildschirm- oder Farbbereich liegen! Wenn Sie wahllos andere Zahlen nehmen, kann es passieren, daß Ihr Computer funktionsunfähig wird und nur durch Aus- und Einschalten wieder in Ordnung gebracht werden kann. Dabei geht aber das Programm verloren! In vielen Fällen läßt sich der Computer ohne Ausschalten noch "zurückholen", indem Sie RUN/STOP und RESTORE gleichzeitig betätigen. Sie sollten diese Möglichkeit auf jeden Fall ausprobieren, bevor Sie zum Netzschalter greifen. Und zu Ihrer Beruhigung: Was immer Sie auch mit Ihrem VC 20 anfangen, Sie können ihn durch eine wie auch immer geartete falsche Eingabe nicht kaputtmachen. Das Schlimmste, das Ihnen passieren kann, ist also ein totaler Programmverlust. Wenn Sie das Programm nicht nochmals auf

Kassette oder Diskette gespeichert haben, mag dies allerdings traurig genug sein.
Jetzt sollten Sie sich ROLLERBALL einmal ausführlich daraufhin betrachten, wie dort die Bewegungen zustande kommen. Sie werden feststellen, es ist alles genau so, wie wir es besprochen haben. Beachten Sie, daß die unterschiedlichen Richtungen, in die der Ball rollen kann, in Zeile 2020 unter DATA abgelegt sind (+21, +22, +23, –1, 0, +1, –23, –22, –21). In Zeile 2050 werden die Werte für die Richtungen in die Variablen A(1) bis A(9) mittels READ eingelesen. Von 3030 bis 3100 wird R in Abhängigkeit davon, welche Richtung in G$ gewählt wurde eine Zahl von 1 bis 9 zugewiesen. Schließlich wird in Zeile 3110 zu der Positionsvariablen P mit A(R) der für die gewählte Richtung jeweils richtige Wert hinzuaddiert, die Position des Balles also um einen Schritt in diese Richtung geändert. 'POKEP,32' dient dazu, den Ball auf dem vorhergehenden Feld zu löschen, wodurch der Eindruck der Bewegung entsteht. Eigentlich recht einfach, nicht wahr!
Wir haben nicht nur die Möglichkeit mit POKE Zeichen auf dem Bildschirm zu setzen und zu löschen, sondern wir können auch mit PEEK abfragen, welches Zeichen und mit welcher Farbe an einer bestimmten Stelle des Bildschirms ist. Schreiben Sie doch einmal ganz links oben ein "A" hin und tippen Sie dann ein paar Zeilen tiefer '?PEEK(7680)' bzw. '?PEEK(4096)' bei mehr als 8 KByte Gesamtkapazität. Das Ergebnis ist 1 als 6-Bit ASCII-Code für das "A". Sicher haben Sie das Prinzip schon durchschaut, es lautet:

Code = PEEK(Adresse)

Um die Farbe eines Zeichens an einer bestimmten Bildschirmstelle festzustellen, gehen wir ganz ähnlich vor. Nur wählen wir als Adressen natürlich nicht den Bildschirm-, sondern eben den Farbbereich. Für unser "A" also '?PEEK(38400)' bzw. bei über 8 KByte Kapazität '?PEEK(37888)'.
In ROLLERBALL wird PEEK laufend benutzt, um zu überprüfen, ob der Ball in ein Hindernis oder den Zaun hineingerät. Sehen Sie, wie in Zeile 3120 auf den Code 102 (Hindernis) und in Zeile 3150 auf 35 (Zaun) abgefragt wird.
Damit wollen wir unsere Bewegunsstudien abschließen. Machen Sie sich mit ROLLERBALL so lange vertraut, bis Sie alles verstanden haben. Sehen Sie sich auch einmal das ROULETTE aus Kapitel 1 an, dort wird ja auch der Lauf der Kugel simuliert.
Zum Schluß wollen wir uns noch anschauen, wie das bunte Blinken und Flackern am Anfang des Spieles – wenn die Er-

klärungen erscheinen – zustandekommt. Dazu muß man wissen, daß im Computer eine einzige ganz bestimmte Speicherzelle für die Farbe sowohl des Bildschirmhintergrundes als auch des Randes verantwortlich ist. Die Adresse dieser Speicherzelle ist 36879.
Um das auszuprobieren, geben Sie einfach zum Beispiel 'POKE36879,0' ein. Wenn Ihr Fernseher richtig eingestellt ist, erhalten Sie nun einen schwarzen Rand, einen weißen Hintergrund und blaue Schrift. Um die Schrift besser lesen zu können, wählen Sie mit CTRL und 2 gleichzeitig die Cursorfarbe weiß und bewegen dann den Cursor über die Schrift. Eigenartig, nicht wahr?
Allgemein gilt:

POKE36879, Farbkombination

Dabei können Sie als Farbkombination jeden Wert zwischen 0 und 255 benutzen.
In den Zeilen 5000 bis 5010 in ROLLERBALL passiert nun nichts anderes, als daß der Wert von 36879 mit jedem Durchlauf um eins erhöht wird und somit jedesmal eine andere Farbkombination gewählt wird. Der Zusatz 'AND255OR8' dient ausschließlich dazu, zu verhindern, daß der Wert 255 überschritten wird, 'OR8' dazu, reverse (inverse) Schrift zu verhindern. Die Routine bei 5000 wird solange durchlaufen, bis irgendeine Taste gedrückt wird. Erfolgt dies, so wird der RETURN-Befehl in Zeile 5010 eingeleitet.
Natürlich gebe es noch viele andere Sachen zu besprechen. ROLLERBALL arbeitet zum Beispiel mit Tonerzeugung und selbstverständlich ließe sich darüber vieles sagen. Wir wollen dies aber erst in einem der nächsten Kapitel tun, dann haben wir dafür nämlich wieder ein neues interessantes Spiel.

Zusammenfassung:
Wenn Sie dieses Kapitel aufmerksam durchgelesen und nachvollzogen haben, dann können Sie jetzt:
- Ladefehlerabfragen beim Einlesen von Programmen vornehmen
- Graphikspiele mit POKE und PEEK programmieren (Übung macht den Meister)
- Bildschirmrand- und Hintergrundfarbe steuern

```
1 REM + ROLLERBALL +
2 REM
3 REM COPYRIGHT 1982
4 REM
5 REM        BY
6 REM
7 REM ANDREAS DRIPKE
8 REM
9 :
100 IFPEEK(158)OR191ANDSTTHENPRINT"■?LADEFEHLER";:END
110 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 POKE657,128:T=36874:L=T+4:PRINTCHR$(14)"■■■■■  ROLLERBALL  ■■■■■
1010 PRINT"VERSUCHEN SIE, DEN
1020 PRINT"■STERN (*) INS ZIEL (+)■ZU STEUERN. BENUTZEN■
1030 PRINT"SIE DAZU FOLGENDE TAS-■TEN:■■"SPC(4)"Q W E":PRINTSPC(10)"|/
1040 PRINTSPC(8)"A-*-D":PRINTSPC(9)"/|":PRINTSPC(8)"Z X C":GOSUB5000
1050 PRINT"■DABEI STEHEN IHNEN":PRINT"■HINDERNISSE IM WEG,
1060 PRINT"■DIE SIE NICHT BERUEH-■■REN DUERFEN. WENN ES■■■IHNEN JEDOCH NICHT
1070 PRINT"■MOEGLICH IST, ZWISCHEN■ZWEI HINDERNISSEN HIN-■DURCHZUKOMMEN (SIE
1080 PRINT"■KOENNEN NICHT ANHAL-■■■TEN), SO DRUECKEN SIE■■'S', DAS HINDERNIS
1090 PRINT"■LOEST SICH AUF.";:GOSUB5000:PRINT"■WAEHLEN SIE NUN DEN
1100 PRINT"■SCHWIERIGKEITSGRAD:■■":PRINT"1 = FEIGLING":PRINT"2 = KINDERLEICHT
1110 PRINT"3 = FUER IHRE OMA":PRINT"4 = PROBIEREN SIE'S":PRINT"5 = SCHWIERIG
1120 PRINT"6 = RISKANT":PRINT"7 = NERVENZERREISSEND":PRINT"8 = HOELLISCH
1130 PRINT"9 = VERGESSEN SIE'S":PRINT"■■IHRE WAHL? ";:POKE198,0:POKE204,0
1140 POKE36879,PEEK(36879)+1AND255OR8:GETG$:S=VAL(G$):IFS=0GOTO1140
1150 POKE207,0:POKE204,1
2000 PRINT"■";:POKET+5,252:FORI=VCTOVC+505:POKEI,2:NEXT:FORI=VBTOVB+21
```

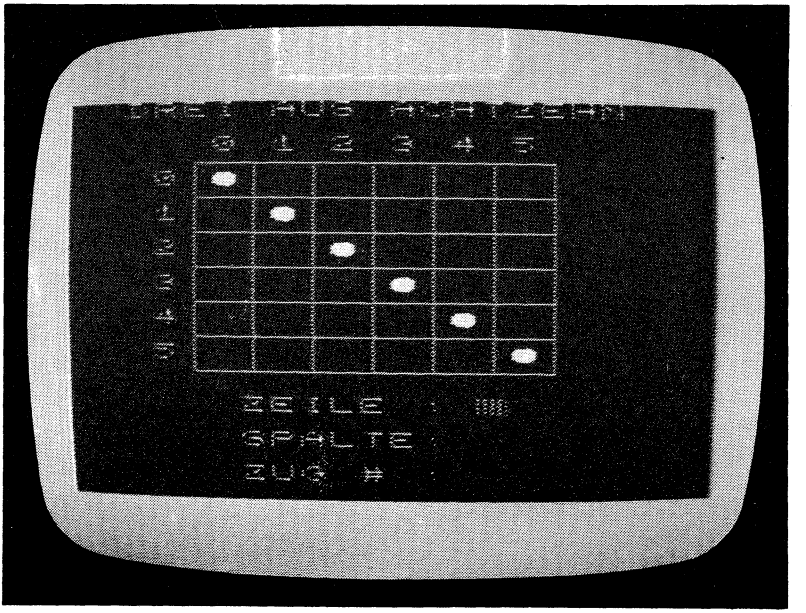
```
2010 POKEI,35:POKEI+462,35:NEXT:FORI=VBTOVB+440STEP22:POKEI,35:POKEI+21,35
2020 NEXT:PRINTSPC(16)"+++++":DATA21,22,23,-1,0,1,-23,-22,-21:FORI=1TO200
2030 X=RND(1)+VB+462*RND(1):ON-(PEEK(X)<>32)GOTO2030:POKEX,102:NEXT
2040 P=VB+442:PRINT"■■■
```

```
4010 PRINT"■SIE HABEN DAS ZIEL IN■":PRINTMID$(TI$,3,2)" MINUTEN UND ";
4020 PRINTRIGHT$(TI$,2)" SE-■■KUNDEN ERREICHT.":PRINT"■DABEI BEKAMEN SIE■
4030 IFE=0THENPRINT"K";
4040 IFE>1THENA$="S":GOTO4060
4050 PRINT"EINEN";:GOTO4070
4060 PRINTMID$(STR$(E),2);
4070 PRINT" ELEKTROSCHOCK"A$:PRINT"■AB.■
4080 IFE>70RTI>3600THENPRINT"DAS WAR JA AEUSSERST":PRINT"■LAHM.":GOTO4120
4090 IFE<S/4-1THENPRINT"DAS WAR AUSGEZEICHNET.":GOTO4120
4100 IFE>S/2THENPRINT"GAR NICHT SO SCHLECHT.":GOTO4120
4110 PRINT"MACHEN SIE WEITER SO.
4120 PRINT"■■NOCH EINMAL? ";
4130 I=I+1AND1:FORJ=0TO99:NEXT:PRINTMID$(" \/",I+1,1)"■■";:GETG$:IFG$="J"THENRUN
4140 IFG$<>"N"GOTO4130
4150 PRINT"■ICH SPIELE AUCH NICHT":PRINT"■MEHR MIT IHNEN.":POKE657,0:END
5000 POKE36879,PEEK(36879)+1AND255OR8:FORI=0TO499:NEXT:GETG$:IFG$=""GOTO5000
5010 RETURN
```

3. "DREI AUS ACHTZEHN"

Nach einem Glücks- und einem Graphikspiel folgt nun ein Denkspiel. In Ihrem Interesse sollten Sie Ihre Konzentration aber nicht nur auf das Spiel selbst, sondern auch auf die nachfolgenden Erklärungen lenken.
Sehen Sie sich in "DREI AUS ACHTZEHN" einmal die Zeilen mit PRINT-Anweisungen an, also zum Beispiel die Zeilen 1000 bis 1060. Fällt Ihnen etwas auf? Alle PRINT-Befehle, denen nichts mehr nachfolgt, enden nicht wie üblich mit einem abschließenden Anführungszeichen, sondern – mit gar nichts. Statt 'PRINT"HALLO"' – wie es korrekt sein sollte – steht nur 'PRINT"HALLO', also nur ein Anführungszeichen. Nun, daran, daß "DREI AUS ACHTZEHN" korrekt abläuft, können Sie sehen, daß es auch so geht. Durch das Weglassen des Anführungszeichens sparen wir sogar jeweils ein Byte. Aber aufgepaßt: Das abschließende Anführungszeichen darf bei PRINT nur dann weggelassen werden, wenn kein weiterer Befehl nachfolgt. Anderenfalls versteht der Computer nämlich auch diese weiteren Befehle schlichtweg als Text und druckt sie aus, statt sie auszuführen. Wegen dieser Gefahr ist die Methode nicht unbedingt empfehlenswert – immerhin, wenn der Speicherplatz einmal wirklich knapp wird, können Sie auf diese Art und Weise ein paar Byte sparen. Und im Verlauf eines Programmes summiert sich da durchaus eine ganze Menge Speicherplatz zusammen.
Da wir gerade beim Einsparen von Speicherkapazität sind, sehen Sie sich einmal an, wie im Programm das Spielfeld, auf dem die Steine gesetzt werden, gezeichnet wird. Dafür verantwortlich sind die Zeilen 2010 und 2020. Uns soll nicht der Ober- und Unterteil des Spielfeldes interessieren, sondern der Mittelteil. Er wird nämlich nicht etwa Zeile für Zeile in PRINT-Befehle gesetzt, sondern einfach in einer Schleife ('FORI=0TO4') ausgegeben. Der Variablen A$ wird einmal die entsprechende Graphik zugewiesen und diese Variable wird dann in der Schleife fünfmal verwendet. Dadurch ist natürlich insgesamt nur ein Fünftel des normalerweise üblichen Speicherplatzes für diese Graphik verbraucht.
Wenn Sie in Ihren eigenen Programmen Graphiken auf dem Bildschirm erstellen, sollten Sie immer überprüfen, welche Teile des Bildes identisch sind. Diese Teile können dann nämlich einer einzigen Stringvariablen zugewiesen und an den entsprechenden Stellen mit PRINT ausgedruckt werden. Wenn Sie dies beim Programmieren konsequent befolgen,

sparen Sie eine ganze Menge Speicherplatz. Und bekanntlich hat man davon – egal wieviel man hat – immer zu wenig.
Die Würze eines Programmes liegt oft in seinen POKE- und PEEK-Befehlen. Wie wir gesehen haben, dienen POKE und PEEK sowohl der Simulation von Bewegungen auf dem Bildschirm als auch der Farbsteuerung von Bildschirmhintergrund und -rand. Ein anderer großer Komplex für POKE liegt in der Tonerzeugung.
Wir werden darauf noch zu sprechen kommen. Mit PEEK und besonders POKE läßt sich aber noch eine Vielzahl von kleinen Raffinessen in unsere Programme einbauen, die weit über die Grundfunktionen Graphik, Farbe und Sound hinausgehen. Einen dieser unzähligen "POKE-Tricks" wollen wir jetzt kennenlernen.
Wie Sie vielleicht wissen, merkt sich Ihr Computer bis zu zehn Tastendrücke. Sie drücken zum Beispiel während ein Programm läuft die Buchstaben "ABC" und wenn das Programm dann an einen INPUT stößt, hat es sich "ABC" gemerkt und wertet das als Eingabe. Unangenehm ist das bei Eingaben mit GET, wenn man vorher aus Versehen irgendeine Taste gedrückt hat. Der Computer nimmt dann nämlich die versehentlich vorher gedrückte Taste an und reagiert entsprechend. Der Fehler ist meist nicht mehr korrigierbar und man muß neu starten. Dem wollen wir jetzt abhelfen.



In einer Speicherzelle merkt sich der Computer, wieviele Tastendrücke gültig sind. Die Adresse ist 198 für den VC 20. Indem man nun kurz vor einer GET-Eingabe dem Computer mit

```
POKE198,0
```

mitteilt, daß null, also gar kein vorher gedrücktes Zeichen gültig ist, "vergißt" er sozusagen alles, was vorher auf der Tastatur vor sich gegangen ist. Egal, welche Taste Sie betätigt haben, für den Computer ensteht der Eindruck, daß nichts dergleichen passiert sei. Schauen Sie sich Zeile 7000 an. Hier steht der entsprechende POKE-Befehl kurz vor der GET-Abfrage, ob die RETURN-Taste gedrückt wurde. In Zeile 4010 steht der POKE direkt vor der GET-Eingabe J/N in Zeile 4020.
Zeile 4020 zeigt übrigens noch etwas anderes Interessantes, nämlich daß das Kommando RUN nicht nur im Direktmodus zum Starten eines Programmes benutzt werden kann, sondern auch programmierbar ist. Wird ein neues Spiel gewünscht, so startet sich einfach das Programm selbst mit RUN neu. Das ist in vielen Fällen einfacher als der Ausstieg aus komplizierten FOR-NEXT-Schleifen oder Unterprogrammebenen. Der Nachteil dabei ist allerdings, daß auch die Erklärung nochmals erscheint, eben all das, was auch beim normalen Programmstart mit RUN passiert. Übrigens sind auch andere Kommandos, die üblicherweise nur von Hand eingetippt werden, programmierbar, so zum Beispiel LIST. Nach LIST wird allerdings das Programm nicht mehr weiter abgearbeitet, es kommt also zu einem Stopp.
Aber da wir eben gerade bei den "POKE-Tricks" waren, wollen wir gleich noch zwei anschließen. Schließlich möchten Sie doch Ihren Computer besser kennenlernen, nicht wahr?
Sie wissen bereits, daß sich der Computer in Speicherzelle 198 bis 10 Tastendrücke merkt. Nun haben wir zwar gelernt, wie man diese Tastendrücke unterbinden kann, aber nicht, was zu tun ist, damit er mehr als diese 10 Tastendrücke speichert. Wenn Sie schnell auf der Schreibmaschine sind, mag es durchaus sein, daß der Computer in einem Programm mit Ihrem Eingabetempo nicht Schritt halten kann und Sie deswegen zu langsamerem Tippen gezwungen sind. Um so etwas zu umgehen, steht uns Speicherzelle 649 zur Verfügung. Sehen Sie einmal nach, welche Zahl normalerweise dort abgespeichert ist. Dies geht am einfachsten mit

```
?PEEK(649)
```

Es ist eine 10, richtig? Diese 10 steht für die Anzahl zu merkender Tastendrücke. Wir können diesen Wert leicht mit POKE verändern. Allgemein gilt:

POKE 649, Anzahl Tastendrücke

Der maximale Wert ist 255. In diesem Fall merkt sich der Computer also bis 255 Tastendrücke im voraus. Im allgemeinen wird dies allerdings nicht sinnvoll sein – die ursprüngliche Zahl 10 reicht in den meisten Fällen aus. Es ging ja hier auch nur um die Möglichkeit, auch auf solche Eigenheiten des Computers Einfluß zu nehmen.
Eine andere interessante Möglichkeit, den Computer mit POKE zu verändern, wollen wir auch noch kennenlernen. Geben Sie einmal

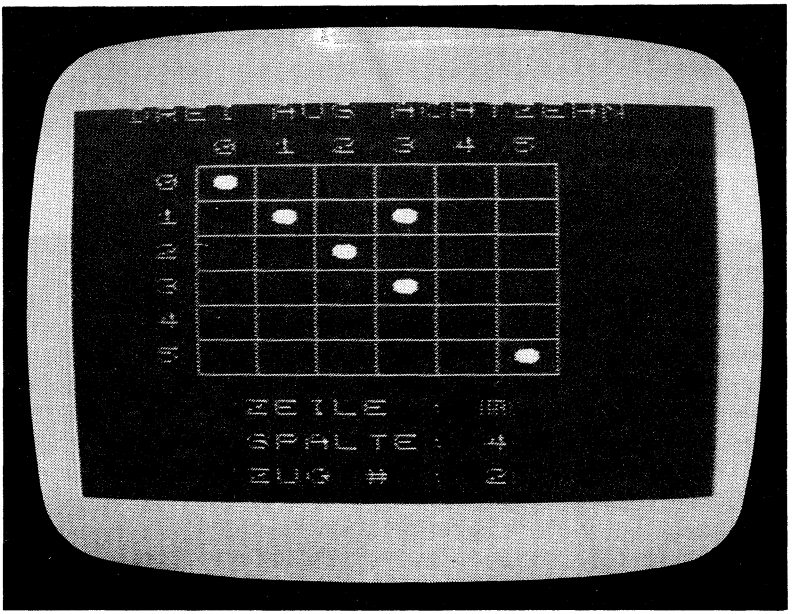
POKE 650,128

ein. Noch passiert nichts. Drücken Sie nun die Taste – sagen wir – X und halten Sie ein klein wenig länger niedergedrückt. Wenn Sie alles richtig gemacht haben, dann wiederholt sich jetzt laufend das X. Der Fachmann sagt auch, die Taste X ist mit einer Repeatfunktion belegt. Durch diesen POKE-Befehl werden auch alle anderen Tasten mit Repeat versehen, probieren Sie es aus. Und wenn Sie dessen überdrüssig geworden sind, schalten Sie einfach zurück mit

POKE 650,0

Wir haben bereits in Kapitel 1 über eine Cursorsimulation bei GET gesprochen. "DREI AUS ACHTZEHN" enthält nun eine besonders interessante Routine, die noch dazu mit Sound versehen ist. Es ist die Eingaberoutine für die Zeilen- und Spaltenangabe. Sie liegt in den Zeilen von 5000 bis 5040. Zeile 5010 dient ausschließlich der Tonerzeugung. In Zeile 5020 wird eines der beiden sich jeweils abwechselnden Cursorzeichen gedruckt. Damit durch den PRINT nicht die Position verändert wird, ist ein 'Cursor nach links' und ein Semikolon angefügt. Dadurch bleibt die Cursorposition beim Blinken unverändert. Daran, daß sich zwei Zeichen laufend abwechseln, ist der MID$-Befehl schuld. Die Variable J wird durch den Ausdruck 'J=1–J' jedesmal zwischen 0 und 1 "umgeschaltet". Das bedeutet, daß in dem Befehl MID$ einmal das erste und ein andermal das zweite Zeichen herausgefiltert und dann gedruckt wird. Und so entsteht das alternierende Blinken beider Cursorzeichen. Die Tastatur wird laufend mit

GET abgefragt, wobei nur Ziffern zwischen 0 und 5 angenommen und in Zeile 5040 ausgedruckt werden. Der Wert, also eine Zahl zwischen 0 und 5, wird der Variablen I zugewiesen. Drücken von "Pfeil nach links" bewirkt einen Sprung zu Zeile 4040.



An den Zeilen 5020 und 5030 wollen wir gleich noch etwas anderes lernen. Vielleicht haben Sie die Konstruktion 'IF..GOTO' mit Verwunderung gesehen. Schließlich wird im allgemeinen 'IF..THEN' verwendet. Der Unterschied ist wie folgt: Nach 'IF..THEN' kann eine beliebige Befehlsfolge kommen, die eben nur dann ausgeführt wird, wenn die Bedingung zwischen IF und THEN erfüllt ist, also zum Beispiel:

```
IF A=1 THEN PRINT "A ist eins"
```

Nach THEN darf auch direkt eine Zeilennummer folgen, zu der verzweigt werden soll. Dazu ein Beispiel:

```
IF A=1 THEN 1000
```

Wenn die Bedingung erfüllt ist, d. h., wenn A gleich 1 ist, dann springt das Programm an dieser Stelle zu Zeile 1000.
Und in diesem Fall – wenn nach THEN direkt eine Sprung-

zeilennummer folgt – dann dürfen wir statt 'IF..THEN' auch 'IF..GOTO' schreiben, in unserem Beispiel also

```
IF A=0 GOTO 1000
```

Für diejenigen unter der verehrten Leserschaft, die einen gewissen Hang zur Korrektheit verspüren, steht übrigens auch die Möglickeit 'IF..THEN GOTO' zur Verfügung. Die beiden anderen Versionen sind im Grunde genommen nämlich nur Abkürzungen dieser korrekten Schreibweise.
Beachten Sie bitte, daß zwar 'IF..GOTO' erlaubt ist, nicht aber 'IF..GOSUB'. In diesem Fall muß man unbedingt zur Form 'IF..THEN GOSUB' greifen.
Wir wollen uns jetzt einer ganz anderen Sache zuwenden, und zwar der Tonerzeugung mit dem VC 20. Gerade ein Spielprogramm ist um ein Vielfaches schöner, wenn es "sound" hat. Ihr Computer ist schon dafür vorbereitet, Töne über den Lautsprecher am angeschlossenen Fernseher zu erzeugen. Wir wollen nun sehen, wie das funktioniert.
Insgesamt stehen uns für die Soundprogrammierung fünf Speicherzellen zur Verfügung. Da ist zunächst einmal Adresse 36878. In dieser Speicherstelle wird die Lautstärke festgelgt.

```
POKE 36878, Lautstärke
```

Der Wert für die Lautstärke kann zwischen 1 und 15 liegen, eine 0 in dieser Speicherzelle bedeutet Stille. In "DREI AUS ACHTZEHN" wird in Zeile 4060 die Lautstärke mit 2 festgelegt. An anderen Stellen im Programm werden andere Lautstärken definiert. Das Explosionsgeräusch am Ende des Spieles entsteht unter anderem dadurch, daß in Zeile 4080 die Lautstärke in der FOR-NEXT-Schleife von 15 auf 0 langsam absinkt.
Nun macht die Lautstärke allein ja noch keinen Ton aus. Deswegen stehen uns zur eigentlichen Tonerzeugung drei andere Speicherzellen zur Verfügung. Nämlich 36874, 36875, 36876.
Allgemein können wir schreiben:

```
POKE 36874, T1
POKE 36875, T2
POKE 36876, T3
```

Diese drei Speicherstellen sind unabhängig voneinander, d. h., wir können mit jeder von ihnen einen anderen Ton kreieren. Mit anderen Worten, es lassen sich bis zu dreistimmige Akkorde mit dem VC 20 spielen. Das ist doch schon eine

ganze Menge, nicht wahr? Je größer eine der T-Variablen ist, desto höher klingt der erzeugte Ton. Der höchste Ton wird mit 255 erzielt, noch höher geht es nicht. Werte unter 128 ergeben gar keinen Ton, 0 schaltet den Ton völlig ab.
Sehen Sie sich "DREI AUS ACHTZEHN" daraufhin an, wie hier die einzelnen Töne erzeugt werden. Die Möglichkeiten sind äußerst vielfältig, denken Sie nur an die Kombination mit FOR-NEXT-Schleifen und ähnliches. Ein Beispiel für einen solchen Effekt finden Sie in Zeile 5010. Geben Sie einmal 'RUN5000' ein, dann hören Sie, was ich meine.
Außer den genannten drei Adressen zur Musikprogrammierung bietet der VC 20 noch einen Sound-Generator mit der Adresse 36877. Dieser steht für die verschiedensten Geräuscheffekte zur Verfügung und wird genau wie die anderen Speicherzellen mit

```
POKE 36877, X
```

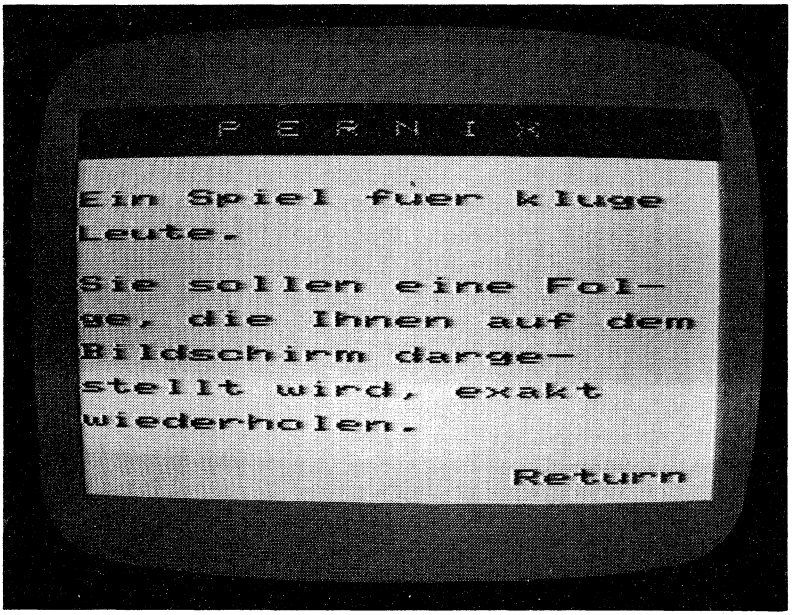
angesprochen. Dabei darf X Werte zwischen 1 und 255 annehmen, 0 schaltet den Generator aus. Die Möglichkeiten, die sich damit eröffnen, sind wirklich äußerst vielfältig. Probieren Sie es aus. Zeile 4070 in Verbindung mit Zeile 4080 zeigt Ihnen nur eine mögliche Verwendung.
Übrigens, wie immer bei POKE, sollten Sie auch bei der Tonprogrammierung äußerst vorsichtig sein, daß Sie nicht versehentlich eine falsche Adresse eingeben. Dies kann u. U. den "Scheintod" des Computers bedeuten, so daß er nur noch durch Aus- und Einschalten erneut zum "Leben" erweckt werden kann. Also: "Finger weg von falschen POKEs".
So, nun will ich Sie mit "DREI AUS ACHTZEHN" alleinlassen und wünsche Ihnen viel Spaß beim Spielen!

Zusammenfassung:
Diesmal haben Sie ein Spiel zum Training Ihrer "grauen Zellen" und (hoffentlich) folgendes gelernt:
- Speicherplatzersparnis durch Weglassen abschließender Anführungszeichen
- Graphikprogrammierung mit Stringvariablen
- Nullen und Verändern des Tastaturpuffers (Anzahl Tastendrücke)
- Ein- und Ausschalten der Repeatfunktion für alle Tasten
- Cursorsimulation mit zwei Zeichen bei GET
- Programmierung von IF..THEN/IF..GOTO-Strukturen
- Erzeugung von Toneffekten

```
1 REM   '3 AUS 18'
2 REM ---------------
3 REM
4 REM (C) 02/1982 BY
5 REM
6 REM ANDREAS DRIPKE
7 REM
8 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 POKE657,128:POKE36879,110:PRINTCHR$(14)"■■■DREI AUS ACHTZEHN■
1010 PRINT"■DIES■IST■EIN■GEDULDS-■■SPIEL.■VERSUCHEN■SIE,
1020 PRINT"■DIE■18■SPIELSTEINE■SO■■AUF■DEM■QUADRAT■ZU
1030 PRINT"■VERTEILEN,■DASS■IN■JE-■DER■ZEILE■UND■SPALTE
1040 PRINT"■GENAU■DREI■STEINE■LIE-■GEN.":GOSUB7000:PRINT"■GEBEN■SIE■ZUERST■DIE
1050 PRINT"■ZEILE■UND■DANN■DIE":PRINT"■SPALTE■AN.■IST■DAS
1060 PRINT"■FELD■BEREITS■BESETZT,":PRINT"■SO■WIRD■ES■GELOESCHT
1070 PRINT"■(UND■UMGEKEHRT).":PRINT"■'←'■=■SPIELABBRUCH.":GOSUB7000
2000 POKE36879,10:PRINTCHR$(142)"■  DREI AUS ACHTZEHN■":PRINTSPC(4)0"1 2 3 4 5
2010 PRINTSPC(4)"┌─┬─┬─┬─┬─┐":A$="│ │ │ │ │ │":FORI=0TO4:PRINTSPC(2)I"■"A$
2020 PRINT"  ├─┼─┼─┼─┼─┤":NEXT:PRINTSPC(3)"5"A$:PRINTSPC(4)"└─┴─┴─┴─┴─┘"
2030 FORZ=0TO5:S=Z:GOSUB6000:NEXT:X=6:PRINTSPC(6)"■ZEILE :
2040 PRINTSPC(6)"■SPALTE:":PRINTSPC(6)"■ZUG # :
3000 PRINT"■■■■■■■■■■■■■■■■■■■■■":GOSUB5000:Z=I:GOSUB5000:S=I:GOSUB6000:N=N+1
3010 PRINT:PRINTSPC(13)N:IFX-18GOTO3000
3020 FORI=0TO5:Z=0:S=0:FORJ=0TO5:Z=Z+A(I,J):S=S+A(J,I):NEXT
3030 IFS<>3ORZ<>3THENI=5:NEXT:GOTO3000
3040 NEXT
4000 POKE36879,43:PRINTCHR$(14)"■■SIE HABEN ES MIT"N:PRINT"■ZUEGEN GESCHAFFT.
```

```
4010 PRINT"■■WOLLEN SIE ES NICHT":PRINT"■NOCHMAL VERSUCHEN?":POKE198,0
4020 GETG$:IFG$="J"THENRUN
4030 IFG$<>"N"GOTO4020
4040 POKE36879,74:PRINTCHR$(14)"■■■■■WENN IHNEN DENKSPIELE
4050 PRINT"■NICHT LIEGEN, SOLLTEN":PRINT"■SIE ES EINMAL MIT EI-
4060 PRINT"■NEM ACTIONSPIEL VERSU-":PRINT"CHEN.":POKE36878,2
4070 FORI=192TO128STEP-.2:POKE36876,I:NEXT:POKE36876,0:POKE36877,190
4080 FORI=15TO0STEP-.03:POKE36878,I:NEXT:POKE36877,0:POKE657,0:END
5000 PRINT:PRINTSPC(14);
5010 POKE36878,7:FORI=192TO254:POKE36876,I:NEXT:POKE36876,0:POKE36878,0
5020 J=1-J:PRINTMID$("■?",J+1,1)"■";:GETG$:IFG$="←"GOTO4040
5030 IFG$<"0"ORG$>"5"GOTO5010
5040 PRINTG$:I=VAL(G$):RETURN
6000 L=Z*44+S*2+93:POKEL+VC,3:F=A(Z,S):IFF=1GOTO6020
6010 A(Z,S)=0:X=X-1:POKEL+VB,32:RETURN
6020 IFX=18THENA(Z,S)=1:X=X+1:POKEL+VB,81
6030 RETURN
7000 POKE198,0:POKE36878,7:PRINTSPC(8)"■■RETURN
7010 POKE36879,(PEEK(36879)+1)AND255:POKE36876,192+63*RND(1)
7020 FORI=1TO1000:NEXTI
7030 GETG$:ON-(G$<>CHR$(13))GOTO7010:POKE36876,0:POKE36878,0:RETURN
```

4. PERNIX



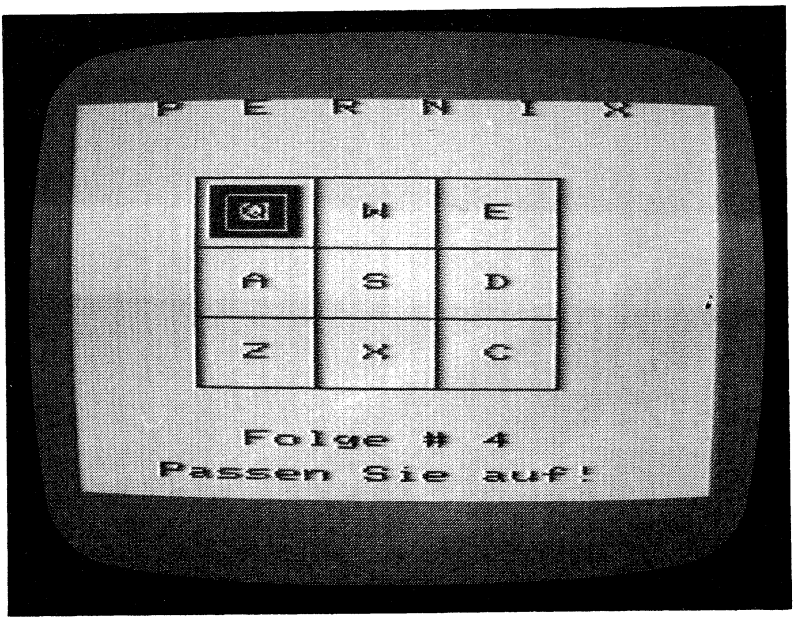
Mit PERNIX stelle ich Ihnen ein geradezu wahnwitziges Spiel vor, das den Spieler – einmal in seinen Bann gezogen – so schnell nicht wieder losläßt. Die Spielidee ist im Grunde genommen recht einfach, es geht darum, eine sich ständig verlängernde Folge von audio-visueller Art zu wiederholen – natürlich fehlerfrei. Daß das aber gar nicht so einfach ist, werden Sie spätestens nach dem ersten Spieldurchgang selber merken.
Uns soll jetzt aber mehr interessieren, was an dem Programm alles zu lernen ist.
Wenn Sie PERNIX schon eingetippt und gespielt haben, dann wissen Sie, daß Sie vor Spielanfang verschiedene Parameter, wie Länge der Gesamtfolge und anderes wählen können. Diese Eingaben sind mit INPUT programmiert, allerdings erscheint nicht der normale Cursor, sondern an seiner Stelle ein blinkendes Karomuster. Die vier Stellen im Programm haben die Zeilennummern 1160, 1190, 1210 und 1220.
Sehen Sie sich die vier INPUT-Befehle an. Das gemeinsame Schema sieht wie folgt aus:

INPUT "TEXT crcrZclcl";G$

cr= Cursor nach rechts
cl= Cursor nach links

Das "Z" steht dabei für das Zeichen. Anstelle des Z schreiben Sie also das Graphikzeichen hin, das Sie bei der INPUT-Eingabe gerne blinken sehen möchten. Welches Zeichen man wählt, hängt nur vom persönlichen Geschmack ab. Ersetzen Sie versuchshalber das Karomuster in PERNIX ruhig einmal durch ein anderes Zeichen. Je mehr Sie selber experimentieren und programmieren, desto mehr werden Sie nämlich auch mit diesem Buch lernen.
In PERNIX ist jede dieser INPUT-Eingaben noch mit einer nachfolgenden Prüfung versehen, die testet, ob gültige Werte eingetippt wurden. Beachten Sie, wie in Zeile 1190 mit IF abgefragt wird, ob auch wirklich 1 oder 2 eingegeben wurde und wenn dies nicht der Fall sein sollte, zu Zeile 1190 zurückverzweigt wird. Ähnlich testen die Zeilen 1160 und 1220, ob die Eingabe zwischen 1 und 9 liegt.
Haben Sie schon einmal probiert, mit PRINT ein Zeichen in die rechte untere Bildschirmecke zu setzen, also in die letzte Position der letzten Zeile. Nein? Nun das wundert mich nicht, dies ist nämlich gar nicht so einfach, wie es vielleicht auf den ersten Blick erscheinen mag. Denn – nach einem PRINT in die letzte Position einer Zeile rutscht der Cursor automatisch eine Zeile tiefer, was bei der untersten Zeile ein Hochschieben ("scrolling") des gesamten Bildschirmes zur Folge hat.
Wenn Sie einen Moment über dieses Problem nachdenken, wird Ihnen sicher eine Lösung einfallen. Man kann nämlich einfach den PRINT an der allerletzten Zeilenposition weglassen und bringt dort stattdessen mit POKE das gewünschte Zeichen auf den Bildschirm. Dies sollte Ihnen nach den Erläuterungen der vorangegangenen Kapitel keine Probleme mehr bereiten.
Auch in PERNIX läuft der Text "Sie sind dran, nur Mut" bis in die letzte Position der untersten Zeile, allerdings ohne POKE. Wie ist das gemacht, werden Sie fragen? Nun, dann sehen Sie sich einmal den letzten PRINT-Befehl in Zeile 4050 an. Wie Sie sehen, schließt der String mit "Mt" und "Cursor nach links" ab. Durch das "Cursor nach links" steht der Cursor zu diesem Zeitpunkt auf dem "t". Das sich anschließende 'CHR$(148)' bewirkt nun die Funktion Insert (INST). 148 ist der ASCII-Code der INST-Taste. Durch dieses INST wird – genau wie beim Drücken der Taste INST – das "t" um eine Position nach rechts verschoben, mithin also von der vor-

letzten auf die letzte Stelle der Zeile. Dadurch, daß dies mit INST geschieht, fällt der Cursor nicht in die nächste Zeile, sondern bleibt an seiner Position – zwischen "M" und "t" stehen. In diese leere Stelle wird noch das "u" eingefügt und schon steht das Wörtchen "Mut" auf dem Bildschirm, natürlich bis hin zur letzten Spalte der untersten Zeile. Dieses Beispiel zeigt sehr deutlich, mit welchen Tricks sich ein erfahrener Programmierer oftmals zu helfen weiß, wo der noch nicht so versierte manchmal schier verzweifelt. Aber beim Programmieren gilt eben in ganz besonderem Maße "Übung macht den Meister".



In den Zeilen 4200 und 4210 begegnet uns noch einmal derselbe Trick. In diesen Zeilen ergibt sich aber noch zusätzlich das Problem, das letzte Zeichen der untersten Zeile – also das "T" von "Mut" – zu löschen. Einfaches Überschreiben mit Leerzeichen ist nicht möglich, da dies ein Fallen des Cursors in die nächste Zeile und somit das Hochrollen des Bildschirmes verursachen würde. Deswegen bedienen sich die beiden genannten Zeilen zusätzlich zu 'CHR$(148)' für INST noch der Funktion 'CHR$(20)', was dem Drücken der DEL-Taste entspricht. Durch DEL wird der Rest der Zeile um eins nach links gezogen, mithin also das Zeichen an der letzten Position der Bildschirmzeile auf die vorletzte Position ge-

bracht. Und jetzt steht dem Überschreiben und anschließendem Schieben mit INST an die letzte Stelle nichts mehr im Wege. Raffiniert, nicht wahr?
Wenn Sie sich die PRINT-Befehle, die mehrere Strings oder Variablen enthalten, in PERNIX ansehen, dann stellen Sie fest, daß in ihnen jegliche Semikolons fehlen. Ein Beispiel dafür ist Zeile 4010. Zwischen der Variablen D$ und dem nachfolgenden String ist kein Semikolon, wie man es üblicherweise erwarten würde.
Dazu sollten Sie sich folgendes merken: Wenn der Cursor nach einem PRINT-Befehl stehen bleiben soll (also keinen Zeilenvorschub auf die nächste Zeile), dann muß man grundsätzlich ein Semikolon setzen. Ein Beispiel dazu:

```
10 PRINT "Testwert";
20 PRINT X
```

Das Semikolon in Zeile 10 bewirkt, daß im Ausdruck der Wert von X direkt hinter das Wort "Testwert" gesetzt wird.
Das Semikolon kann entfallen, wenn in einem einzigen PRINT-Ausdruck mehrere Texte in Anführungszeichen und/oder Variablen stehen. Wenn wir die zwei PRINT-Anweisungen unseres letzten Beispiels in eine Zeile zusammenfassen, können wir also schreiben:

```
10 PRINT "Testwert" X
```

Das Semikolon zwischen "Testwert" und X entfällt. Natürlich muß die Trennung eindeutig sein. Nehmen wir an, Sie wollen drei numerische Variablen A, B und C nach einem Text ausdrucken lassen. Dann dürfen Sie nicht etwa

```
PRINT "Testwerte"ABC
```

schreiben, denn dann faßt Ihr Computer ABC als eine Variable auf. In dem Fall ist die ausführliche Form auch die richtige:

```
PRINT "Testwerte"A;B;C
```

Nach den Funktionen TAB(..) und SPC(..) können wir das Semikolon ebenfalls weglassen. Ein gutes Beispiel für eine völlige Mischung ganz ohne Semikolon zeigt Ihnen Zeile 4050. Hier stehen eine Stringvariable, ein String, SPC und ein numerischer Ausdruck in einer einzigen PRINT-Anweisung. Jedes Semikolon, das da ausgelassen wird, spart natürlich ein Byte. Wenn man dies in einem Programm konsequent durch-

hält, kann man auf diese Art und Weise durchaus eine ganze Menge Speicherplatz einsparen. Vielleicht denken Sie einmal an diese Methode, wenn es in Ihrem Computer eng wird.
Wir wissen bereits, wie wir dem Computer Töne und Geräusche entlocken können. Die Lautstärke wird eingestellt mit 'POKE36878,X', wobei X zwischen 0 (kein Ton) und 15 liegen darf. Zur Tonerzeugung selbst stehen die Speicherzellen 36874, 36875 und 36876 zur Verfügung, für Geräusche Zelle 36877. Wenn wir einmal von den Geräuscheffekten absehen, ist es für die Programmierung von Musik natürlich wichtig, welche Werte welchen Frequenzen und damit welchen Noten entsprechen.
Für diejenigen unter Ihnen, die nicht ganz so musikalisch veranlagt sind, um dies selbst festzustellen, habe ich das einmal übersichtlich in einer Tabelle zusammengefaßt (Musikentwurfsblatt). Damit wird es Ihnen keine Schwierigkeiten mehr bereiten, irgendein Musikstück zu programmieren, und zwar mit bis zu drei Stimmen.
In PERNIX sind die Werte für die Töne, die den einzelnen Feldern entsprechen, unter DATA abgelegt. Sehen Sie sich Zeile 3000 daraufhin an. Mit Hilfe des Musikentwurfblattes sollte es Ihnen keine Probleme mehr bereiten, hier andere wohlklingende Töne zu definieren. Probieren Sie es!
Fällt Ihnen eigentlich bei der FOR-NEXT-Schleife in Zeile 4220 etwas auf? Ich meine die Tatsache, daß diese Schleife von 1 bis 2E3 zählt. Nun, 2E3 ist die sogenannte Exponentialschreibweise und genau dasselbe wie 2000 (also eine 2 mit drei Nullen). Im allgemeinen wird die Exponentialdarstellung für mathematische Zwecke gebraucht. Sie ermöglicht es nämlich, Zahlen darzustellen, die sich in normaler Schreibweise wegen der begrenzten Anzahl von 9 signifikanten Stellen nicht mehr schreiben ließen, weil sie zu groß oder zu klein sind. So bedeutet z. B. 5E20 eine 5 mit 20 Nullen, was mit 9 Stellen beim besten Willen nicht mehr darstellbar wäre.
Wie Sie an PERNIX sehen, kann man die mathematische Schreibweise aber auch verwenden, einfach um ein paar Bytes zu sparen. 2E3 verbraucht nämlich nur 3 Byte, wohingegen die Normalschreibweise 2000 4 Bytes belegt. Zugegeben, es ist nur ein einziges Byte, aber immerhin. Wenn Sie alle Tricks zusammennehmen, die wir schon besprochen haben, um ein paar Bytes zu sparen, dann mag bei einem längeren Programm schon ein gutes halbes Kilo zusammenkommen.
Wir wollen uns noch mit den Zeilen 2040 und 2050 in PERNIX beschäftigen. Es geht in diesen beiden Zeilen darum, alle vorher definierten Variablen zu löschen (CLR in Zeile 2040), ohne

daß dadurch auch die wichtigen Werte, die noch benötigt werden, verloren gehen.
Dazu werden vor dem CLR-Befehl die Variablen, deren Werte erhalten bleiben sollen – nämlich L, F, M, und S – in die Speicherzellen 828 bis 831 "gePOKEt". Diese Speicherzellen werden aber durch CLR nicht gelöscht. Folglich können in Zeile 2050 die Werte mit PEEK wieder hervorgeholt und den entsprechenden Variablen zugewiesen werden. Beachten Sie, daß die Befehle PEEK und POKE nur für positive ganze Zahlen zwischen 0 und 255 zuläsig sind. Hätte eine unserer vier Variablen einen abweichenden Wert, käme es zu einem ILLEGAL QUANTITY ERROR.
Die Adressen 828 bis 831 sind nicht zufällig gewählt. Im Bereich von 828 bis 1019 befindet sich der Eingabepuffer für den Kassettenrekorder. Dieser Bereich wird nur benötigt, wenn ein Programm von Kassette eingelesen wird. Anderenfalls steht er zur Verfügung.
Achtung: Benutzen Sie für POKE keine beliebigen Adressen, denn dies kann zur Funktionsuntüchtigkeit des Computers führen! Das Gerät geht dabei zwar nicht kaputt, wohl aber das Programm verloren.
Stringvariablen können auf die gezeigte Art und Weise ebenfalls über CLR hinaus erhalten werden, allerdings nicht ganz so einfach. Vielmehr muß der String in einzelne Zeichen zerlegt werden, die dann mit POKE abgelegt werden. Eine Routine dafür könnte zum Beispiel so aussehen:

```
FORX=1TOLEN(S$):POKE829,ASC (MID$(S$,X)):NEXTX
```

Diese Zeile speichert die Stringvariable S$ ab Speicherzelle 829 ab. Damit der String auch wieder korrekt eingelesen werden kann, muß noch die Länge abgelegt werden. Wir nehmen dazu Speicherstelle 828:

```
POKE828,LEN(S$)
```

Um den String nach CLR wieder herzustellen, dient folgende Programmzeile:

```
FORX=1TOPEEK(828):S$+CHR$(PEEK(829+X)):NEXTX
```

Im Einzelfall bleibt abzuwägen, ob der Speicherplatzvorteil durch CLR den erhöhten Aufwand zum Abspeichern eines Strings auf die gezeigte Weise rechtfertigt. Die Löschung von unwichtigen Variablen lohnt sich auf jeden Fall immer dann, wenn nur noch sehr wenig Speicherplatz frei bleibt. Ein Pro-

gramm läuft nämlich wesentlich langsamer, wenn die noch freie Speicherkapazität ca. 300 Byte unterschreitet.
Zum Abschluß dieses Kapitels wollen wir uns noch anschauen, wie die erzeugte Folge eigentlich auf dem Bildschirm dargestellt wird, also das Springen des Rechteckes von Feld zu Feld. Starten Sie dazu das Programm und lassen Sie es laufen, bis die erste Folge kommt. Unterbrechen Sie nun mit STOP, löschen den Bildschirm und geben '?F$' ein. Das Feld wird gezeichnet. Gehen Sie mit dem Cursor über das "F" von '?F$' und ersetzen Sie da "F" durch ein "L". Drücken Sie RETURN. Was passiert diesmal?
Definiert werden die beiden Strings F$ und L$ in Zeile 3010. Beachten Sie, daß sowohl F$ als auch L$ die mittlere Position, an der beim Spiel der entsprechende Buchstabe steht, mit "Cursor nach rechts" übergehen, mithin also unverändert lassen. Dadurch bleibt der Buchstabe stehen.
Die Routine, die nun F$ auf den Bildschirm bringt und anschließend mit L$ wieder löscht, liegt in den Zeilen 6000 bis 6040. Beachten Sie, wie in 6000 und 6040 in einer recht komplizierten Formel die jeweils richtige Bildschirmposition berechnet wird. Die Zeilen 4070 bis 4148 zeigen, wie der Variablen X ein Wert von 0 bis 8 zugeordnet wird, in Abhängigkeit davon, welche Taste Sie drücken. Die zum Verständnis der Formel noch wichtige Variable D$ wird in Zeile 3010 definiert. Gehen Sie die Formel einmal Schritt für Schritt durch, dann werden Sie auch verstehen, wie sie funktioniert. Die Zeilen 6010 und 6030 dienen ausschließlich der Tonerzeugung, der zum jeweiligen Feld zugehörige Ton ist in A(X) gespeichert. Die Geschwindigkeit, mit der die Folge insgesamt abläuft, wird in Zeile 6020 gesteuert. S kann dabei Werte zwischen 1 und 9 annehmen.
Aufgerufen wird die vollständige Folge in Zeile 4040 mittels 'GOSUB6000' innerhalb einer FOR-NEXT-Schleife. Mit MID$ wird aus der Variablen A$, die die Folge in Ziffern enthält (probieren Sie es mit '?A$' aus), ein Zeichen nach dem anderen herausgeholt. Der Ausdruck erfolgt dann wie gezeigt in der Routine bei 6000. Die Kreation einer neuen Folge bzw. das Anhängen eines neuen Zeichen an A$ geschieht in den Zeilen 5000 und 5010. Mit RND wird eine Zahl zwischen 0 und 8 ausgewählt.
Nun wünsche ich Ihnen viel Spaß mit PERNIX! Wenn Sie es öfter spielen, dann werden Sie sehen, wie leicht man sein audiovisuelles Gedächtnis schulen kann. Ich muß allerdings gestehen, daß es mir nie gelungen ist, die maximale Länge der Folge – also 50 – zu erreichen. Aber vielleicht schaffen Sie es ja.

Zusammenfassung:
Der Lehrstoff dieses Kapitels ist:
- INPUT mit beliebigem Blinkzeichen
- PRINT an die letzte Bildschirmposition
- Gebrauch des Semikolons im PRINT-Befehl
- Exponentialschreibweise
- Rettung von Variablen über CLR hinaus
- Musikprogrammierung nach Noten

```
1 REM >>> PERNIX <<<
2 REM
3 REM
4 REM  (C) 1982 BY
5 REM
6 REM ANDREAS DRIPKE
7 REM
8 REM
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 POKE657,128:POKE36879,138:PRINTCHR$(14)"■■":FORI=0TO65:PRINT"PERNIX ";
1010 NEXT:FORI=0TO21:POKEVC+I,0:POKEVB+I,102:POKEVC+I+484,0:POKEVB+I+484,102
1020 NEXT:FORI=22TO462STEP22:POKEVC+I,0:POKEVB+I,102:POKEVC+I+21,0
1030 POKEVB+I+21,102:NEXT:FORI=1TO3E3:NEXT
1040 PRINT"■■                                P E R N I X   ";
1050 PRINT"                                  ■■■EIN SPIEL FUER KLUGE":PRINT"■LEUTE.■■
1060 PRINT"SIE SOLLEN EINE FOL-":PRINT"■GE, DIE IHNEN AUF DEM
1070 PRINT"■BILDSCHIRM DARGE-":PRINT"■STELLT WIRD, EXAKT":PRINT"■WIEDERHOLEN.
1080 GOSUB7000:PRINT"■DABEI WIRD DIE FOLGE":PRINT"■IMMER LAENGER UND DA-
1090 PRINT"■MIT DAS SPIEL IMMER":PRINT"■SCHWIERIGER.":GOSUB7000
1100 PRINT"■ZU JEDEM FELD ERTOENT":PRINT"■EIN ANDERER TON.
1110 PRINT"■■DIESES SPIEL EIGNET":PRINT"■SICH ALSO HERVORRA-
1120 PRINT"■GEND DAZU, IHR AUDIO-":PRINT"■VISUELLES GEDAECHTNIS
1130 PRINT"■ZU SCHULEN.":GOSUB7000:PRINT"■SIE HABEN DIE MOEG-■
1140 PRINT"LICHKEIT, PERNIX NACH":PRINT"■IHREN WUENSCHEN ZU":PRINT"■GESTALTEN.
1150 PRINT"■■UM WIEVIEL SOLL DIE":PRINT"■FOLGE JEDESMAL LAEN-■■
1160 INPUT"■GER WERDEN (1-9)■■■■";G$:L=VAL(G$):IFL<1ORL>9GOTO1160
1170 GOSUB7000:PRINT"■SOLL 1. DIE FOLGE PRO":PRINT"■RUNDE UM"L"VERLAEN-
```

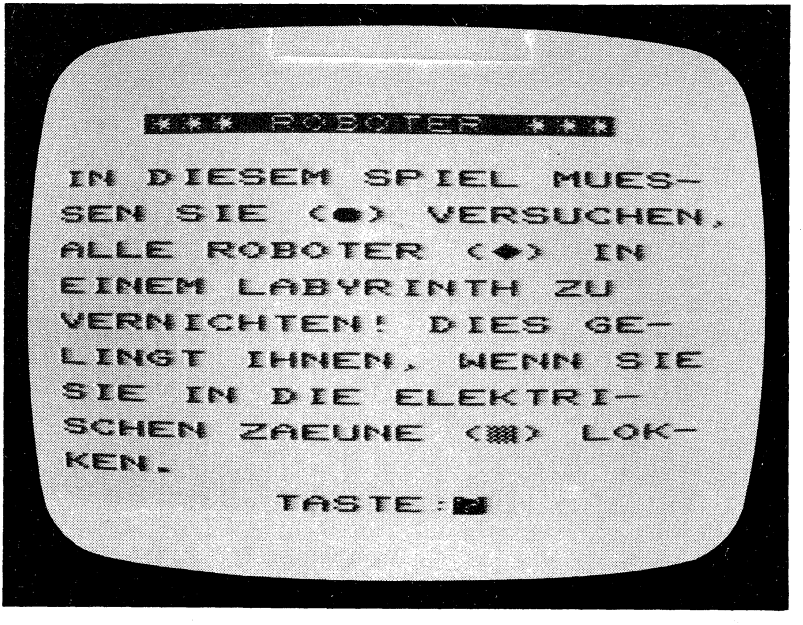
```
1180 PRINT"■GERT ODER 2. JEDESMAL":PRINT"■NEU ENTWORFEN WERDEN■■
1190 INPUT"□(1 ODER 2)■■■■";G$:F=VAL(G$):IFF<>1ANDF<>2GOTO1190
1200 PRINT"■MAXIMALE FOLGENLAENGE■■
1210 INPUT"□(HOECHSTENS 50)■■■■";G$:M=VAL(G$):ON-(M<50RM>50)GOTO1210:PRINT"■
1220 INPUT"□DIE SCHNELL (1-9)■■■■";G$:S=VAL(G$):IFS<1ORS>9GOTO1220
2000 PRINT"□■■■P  E  R  N  I  X■":A$="■■■■|   |   |   |":B$="■■■■|---+---+---+---|
2010 PRINTSPC(4)"■■┌---┬---┬---┐":PRINTA$:PRINTSPC(4)"| ● | o | - |":PRINTA$
2020 PRINTB$:PRINTA$:PRINTSPC(4)"| ♠ | ♥ | - |":PRINTA$:PRINTB$:PRINTA$
2030 PRINTSPC(4)"| ♦ | ♣ | - |":PRINTA$:PRINTSPC(4)"└---┴---┴---┘
2040 PRINT"■■"SPC(6)"_OLGE #":POKE828,L:POKE829,F:POKE830,M:POKE831,S:CLR
2050 L=PEEK(828):F=PEEK(829):M=PEEK(830):S=PEEK(831)
3000 FORI=0TO8:READA(I):NEXT:DATA135,147,159,163,175,183,191,195,201
3010 F$="■┌-┐■■■■|■|■■■■└-┘":L$="    ■■■■ ■ ■■■■    ":D$="■■■■■■■■■■■■■■■■
4000 A$=""
4010 PRINTD$"■■■■■■■■■■   PASSEN SIE AUF!     ";
4020 IFF=2THENX=LEN(A$):A$="":FORI=1TOX+L:GOSUB5000:NEXT:GOTO4040
4030 FORI=1TOL:GOSUB5000:NEXT
4040 FORI=1TOLEN(A$):X=VAL(MID$(A$,I,1)):GOSUB6000:NEXT:I=0:POKE198,0
4050 PRINTD$"■■■■■■■■"SPC(13)LEN(A$):PRINT"■SIE SIND DRAN, NUR MT■■"CHR$(148)"U";
4060 I=I+1
4070 GETG$:G=9:IFG$="Q"THENG=0
4080 IFG$="W"THENG=1
4090 IFG$="E"THENG=2
4100 IFG$="A"THENG=3
4110 IFG$="S"THENG=4
4120 IFG$="D"THENG=5
4130 IFG$="Z"THENG=6
4140 IFG$="X"THENG=7
```

```
4150 IFG$="C"THENG=8
4160 IFG=9GOTO4070
4170 IFG<>VAL(MID$(A$,I,1))GOTO4210  .
4180 X=G:GOSUB6000:IFI<LEN(A$)GOTO4060
4190 IFLEN(A$)<MTHENFORI=1TO2E3:NEXT:GOTO4010
4200 PRINTD$"■■■■■■■■■SEHR GUT - NOCH EINMA"CHR$(20)"L■"CHR$(148)"A";:GOTO4220
4210 PRINTD$"■■■■■■■■■   F  A  L  S  C  H  "CHR$(20)" ";
4220 FORI=1TO2E3:NEXT:GOTO4000
5000 X=INT(RND(1)*9):B$=MID$(STR$(X),2):IFB$=RIGHT$(A$,1)GOTO5000
5010 A$=A$+B$:RETURN
6000 PRINTLEFT$(D$,6+INT(X/3)*4)SPC(5+4*(X-INT(X/3)*3))F$
6010 POKE36878,10:POKE36876,A(X)
6020 FORJ=1TO(9-S)*100:NEXT
6030 POKE36878,0:POKE36876,0
6040 PRINTLEFT$(D$,6+INT(X/3)*4)SPC(5+4*(X-INT(X/3)*3))L$:RETURN
7000 PRINTSPC(15)"■■ETURN":POKE198,0
7010 GETG$:IFG$<>CHR$(13)GOTO7010
7020 RETURN
```

5. ROBOTER

Bei diesem spannenden Echtzeitspiel geht es darum, Ihre Haut vor dem Zugriff der Roboter zu retten. Sie erreichen dies, indem Sie die Roboter durch geschicktes Taktieren in die überall herumstehenden Elektrozäune locken, ohne selbst diesen mit Hochspannung geladenen Drahtverhauen zu nahe zu kommen - andernfalls bekommen Sie die knisternde Spannung dieses Spiels am eigenen Leib zu spüren. Also Vorsicht!
In der vorausgehenden packenden Beschreibung des ROBOTER-Spiels fällt der Ausdruck "Echtzeitspiel". Dies ist der Begriff für eine von zwei grundsätzlich verschiedenen Arten von Programmen. Unter einem Echtzeitprogramm (engl. "real time program") versteht man ein Programm, bei dem die Zeit ein ganz wesentlicher Faktor ist. Wie dies zu verstehen ist, wird am einfachsten klar an einem Gegenbeispiel. Nehmen wir zum Beispiel einen Schachcomputer. Ein Schachprogramm hat unendlich viel Zeit, sich einen Zug zu überlegen (zumindest theoretisch, praktisch sicherlich nicht), da das Spiel eben solange ruht, bis dieser nächste Zug erfolgt. Mit anderen Worten, die Spielsituation verändert sich nicht, während der Schachcomputer - oder auch Sie - über den nächsten Zug nachsinnt. Dies ist also gerade keine Echtzeitverarbeitung, da das Fortschreiten des Spiels in keiner Weise durch die Zeit beeinflußt wird, sondern ausschließlich durch die einzelnen Züge der Spieler.
Ein Beispiel für Echtzeitverarbeitung dagegen ist unser Programm ROBOTER. Einmal gestartet, läuft das Spiel ab, unabhängig davon, ob Sie überhaupt mitspielen oder nicht. Mit voranschreitender Zeit läuft das Spiel, wobei Sie allerdings die Möglichkeit haben, es zu beeinflussen (andernfalls wäre es sicherlich recht langweilig). Echtzeitspiele bringen den Spieler immer in eine enorme Streßsituation, da er unter ständigem Zeitdruck steht. Dem steht oft eine erhöhte Spannung verglichen mit ruhigen Spielen gegenüber. Nun, Sie finden in diesem Buch beide Arten von Spielen - schließlich gibt erst die Abwechslung die richtige Würze. Doch wenden wir uns jetzt den programmtechnischen Aspekten von ROBOTER zu.
Wenn Sie sich die Zeile 1200 ansehen, dann mag es Ihnen vielleicht merkwürdig erscheinen, daß in einem Programm wie ROBOTER die Möglichkeit genutzt wird, mit DEF FN mathematische Funktionen zu definieren. Schließlich hat ROBOTER auf den ersten Blick nicht viel mit Mathematik zu tun. Nun, die beiden in dieser Zeile definierten Funktionen X und Y be-

stimmten die X/Y-Koordinaten der Roboter. Warum soll man dafür nicht einmal Funktionen festlegen, die dann an den gewünschten Stellen im Programm aufgerufen werden (Zeilen 2130 und 4030). Viele Programmierer denken bei einem Spielprogramm gar nicht an die mathematischen Möglichkeiten des Computers und kommen deswegen auch gar nicht auf die Idee, diese einzusetzen, wo es sinnvoll ist. Sie sollten immer wenn es um numerische Berechnungen geht, überprüfen, ob sich DEF FN mit Erfolg einsetzen läßt. Sie werden feststellen, das ist öfter der Fall, als man zunächst geneigt ist anzunehmen.
Übrigens zeigt uns Zeile 1200 gleich noch eine Besonderheit im Zusammenhang mit DEF FN. Die Funktionsdefinition Y beinhaltet unter anderem die bereits vorher definierte Funktion X. Hier sind also Funktionen in der Festlegung geschachtelt.
Wichtig zu wissen: Ein Funktionsaufruf mit FN darf nur dann innerhalb einer DEFFN-Anweisung vorkommen, wenn die Funktion bereits definiert ist. Insbesondere ist es nicht möglich, daß sich eine Funktion in ihrer Definition selbst aufruft. Das ist eigentlich auch ganz logisch, nicht wahr?
Es gibt einen Schönheitsfehler, der bei Programmen für die Commodore Computer immer wieder auftaucht. Einen revers gedruckten Text (RVS ON) sollten Sie, wenn irgend möglich,



überstreichen. Was ich damit meine, sehen Sie in den Zeilen 1000/1010 in ROBOTER. Über den reversen Text "*** ROBOTER ***" wird ein dünner Balken nicht-revers gedruckt. Dadurch wirkt der Text nicht - wie sonst üblich - wie oben abgeschnitten.
Sie wissen, daß zwei POKE-Befehle nötig sind, um ein Zeichen auf den Bildschirm zu "POKEn", einer für das Zeichen und seine Position, der andere für die Farbe. Bei Graphikspielen oder überhaupt bei Programmen, die schnell ablaufen sollen, stört es oftmals, daß für jedes Zeichen zwei POKEs notwendig sind. Auch bei ROBOTER stand dieses Problem an und dieses Spiel ist tatsächlich so programmiert, daß für jede Bewegung auf dem Bildschirm nur noch ein POKE - nämlich der für das Zeichen selbst - erforderlich ist. Wie das funktioniert?
Nun, dann machen Sie doch einmal folgendes Experiment (Wenn Ihr VC 20 mehr als 8 KByte Gesamtkapazität hat, verwenden Sie bitte andere Adressen gemäß Kapitel 2.).Stellen Sie durch gleichzeitiges Drücken von RUN/STOP und RESTORE die Grundfarben her und löschen Sie den Bildschirm. Gehen Sie nun mit dem Cursor ein paar Zeilen tiefer und tippen Sie

```
POKE 38400,5
```

ein. noch ist nichts zu sehen. Geben Sie jetzt

```
POKE 7680,42
```

ein. Dieser letzte – eine – POKE hat an einer vorher frei erscheinenden Stelle ein grünes Sternchen erzeugt. Dies war nur möglich, weil die Farbe schon vorab, quasi unsichtbar, festgelegt wurde.
Sehen Sie sich nun einmal Zeile 2050 in ROBOTER an. Die Variable VC ist wie üblich die niedrigste Adresse des Farbbereiches, bei einem VC 20 mit bis zu 8 KByte also 38400. In Zeile 2050 wird der gesamte Farbbereich mit seinen 506 Positionen mit durch RND zufällig erzeugten Farbwerten belegt. Da aber in diesem Moment keine Zeichen in den Bildschirmbereich gebracht werden, bleibt die gesamte Farbmanipulation des Bildschirms zunächst unsichtbar. Wenn nun aber im Verlaufe des Spiels Zeichen "gePOKEt" werden, dann treten die Farben automatisch zum Vorschein. Nach der Farbvorbelegung reicht es also, wenn pro Zeichen nur noch ein einziger POKE, nämlich in den Bildschirmbereich, abgesetzt wird. Der zweite POKE, in den Farbbereich, kann entfallen.

Und auf diese Art und Weise erhöht sich natürlich insgesamt die Geschwindigkeit, mit der das Spiel abläuft.
Oftmals ist es gar nicht nötig, den kompletten Bildschirm mit Farbwerten vorzubelegen, sondern es genügt, einen bestimmten, kleineren Bereich vorab mit einer unsichtbaren Farbe zu versehen. Und selbstverständlich wird man in den meisten Fällen die Farben nicht mit RND erzeugen, sondern exakt definieren. Die Möglichkeiten, die sich durch die gezeigte Methode für uns eröffnen, sind recht vielfältig. Experimentieren Sie ruhig selbst ein bißchen in dieser Richtung.
Wenn Sie sich die DATA-Anweisung in Zeile 1190 anschauen, dann werden Sie entdecken, daß zwischen der –1 und der 1 zwei Kommata stehen. Normalerweise werden die einzelnen Elemente einer DATA-Liste nur durch jeweils ein einziges Komma getrennt. Warum also hier zwei? Nun, die zwei Kommata ergeben einen zusätzlichen Wert 0. Machen wir uns das an einem Beispiel klar. Statt

```
DATA –1,,1
```

könnten wir ausführlicher auch

```
DATA –1,0,1
```

schreiben. Die zwei Kommata wirken also so, als wenn an dieser Stelle ein zusätzliches DATA-Element mit dem Wert 0 steht.
Gleiches gilt übrigens auch für ein Komma am Ende einer DATA-Zeile. Ein Beispiel dazu: Die Zeile

```
DATA 1,2,3,
```

ist gleichbedeutend mit

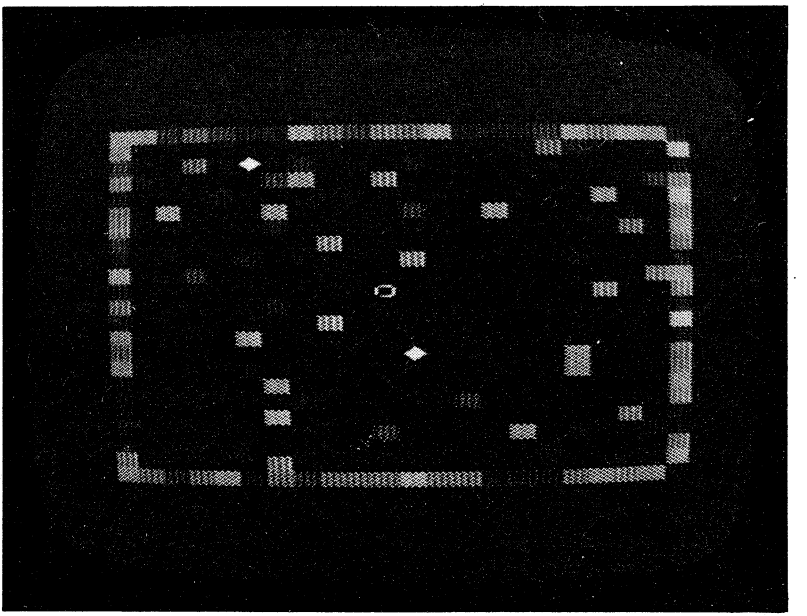
```
DATA 1,2,3,0
```

Das zusätzliche Komma hinter der 3 wirkt also auch hier, als ob ein weiteres Element mit dem Wert 0 angefügt wird. Die logische Begründung wird Ihnen sicher klar, wenn Sie sich bewußt machen, daß ein Komma bei DATA ausschließlich zur Trennung von verschiedenen Elementen dient, und nicht etwa zum Beispiel zum Abschluß eines Elementes.
Wenn in Ihrem eigenen Programm einmal eine DATA-Liste nicht korrekt, sondern irgenwie versetzt eingelesen wird, dann sollten Sie nachsehen, ob Ihnen nicht vielleicht ein Komma zuviel irgendwo hineingeraten ist. Denn durch einen solchen

kleinen Fehler verschiebt sich – wie gezeigt –, die gesamte Liste, eben dadurch, daß eine Element eingefügt wird.
Wir wollen uns jetzt einen wichtigen Aspekt der Benutzung von Variablen klarmachen. Sehen Sie sich dazu Zeile 100 an. Wenn man die fertige Programmzeile ansieht, fällt einem zunächst gar nichts auf. Aber, mal ehrlich, hätten Sie nicht wahrscheinlich auch

```
100 S=36876:L=36878:C=36879 ...
```

geschrieben. In ROBOTER dagegen werden die Variablen L und C aus S berechnet, was natürlich Speicherplatz spart, im Beispiel genau vier Byte. Auch hier gilt – wie so oft, wenn es um Speicherplatzersparnis geht – der Einzelfall bringt nicht allzuviel, aber die konsequente Beachtung solcher Programmiertechniken dafür umso mehr. Achten Sie also in Ihren eigenen Programmen darauf, ob sich nicht eine Variable aus einer anderen, schon definierten, leichter errechnen läßt als die ganze Zahl auszuschreiben.



Ein weit verbreiteter Fehler ist, daß die Variablen überhaupt zu wenig eingesetzt werden. Wenn in Ihrem Programm eine bestimmte Zahl oft vorkommt, ersetzen Sie die Zahl durch eine Variable. Dies bringt Ihnen nicht nur zusätzliche Speicher-

kapazität, sondern macht Ihre Programme auch schneller. Der Computer ist nämlich in der Lage wesentlich rascher eine Variable als eine Zahl zu interpretieren. Das liegt daran, daß die Variable schon im richtigen, computerinternen Format abgespeichert ist, während eine Zahl innerhalb eines Programmes erst in dieses Format umgewandelt werden muß. Ein Beispiel dafür, wo sich so etwas "bezahlt" macht, sind die beiden Bildschirmvariablen VB und VC. Sie geben die linke obere Bildschirmposition an, so daß alle anderen Positionen relativ zu ihnen errechnet werden können.
Da wir gerade beim Thema Variable sind, wollen wir auch gleich noch etwas über die Länge von Variablennamen sagen. Sie wissen sicher, daß im Commodore BASIC die ersten zwei Buchstaben eines Variablennamens signifikant sind.
Zwischen den Variablen ZAHL und ZAHL2 besteht für den Computer also kein Unterschied, vielmehr sieht er sie beide als dieselbe Variable ZA an. Übrigens ein Umstand, auf den Sie beim Programmieren achten sollten, sonst kommt es schnell zu Schwierigkeiten. Was ich Ihnen aber eigentlich zeigen wollte, sehen Sie in Zeile 2040, ich meine die Zuweisung 'O=VB'. Nach dieser Zuweisung verwendet das Programm fortan die Variable O anstelle von VB. Und das spart natürlich jedesmal genau ein Byte und erhöht wiederum die Geschwindigkeit. Bei Variablen, die sehr oft in einem Programm benutzt werden, kann es also durchaus lohnend sein, einen Variablennamen zu wählen, der aus nur einem einzigen Buchstaben besteht. Natürlich geht damit auch die Aussagekraft des Namens weitgehend verloren; die Bedeutung von VB als "Variable Bildschirm" ist durchaus sinnfällig, was bei O sicherlich nicht der Fall ist. Aber schließlich gehen die meisten Tricks zum Einsparen von Speicherplatz zu Lasten der Lesbarkeit des Programmes, weswegen Sie sie auch nur bei wirklich knapper Kapazität anwenden sollten. Wenn Sie Ihren Computer genügend ausgebaut haben, nutzen Sie den zur Verfügung stehenden Speicherraum und verzichten Sie auf die Kniffe, die zwar Platz sparen, es aber gleichzeitig schwer machen, ein Programm zu lesen. Schließlich haben Sie sich ja dafür die zusätzliche Speicherkapazität zugelegt, nicht wahr?
Wir wollen uns nun noch den Programmteil anschauen, der dafür sorgt, daß bei einer Berührung mit den "elektrischen Zäunen" der ganze Bildschirm zu beben scheint. Es ist Programmzeile 3060. Die Variable C hat dabei den Wert 36879, das ist die Speicherzelle, wo die Farbkombination von Bildschirmrand und -hintergrund festgelegt ist. Wie Sie sehen, werden in Zeile 3060 dreimal (FORI=0TO2) direkt hintereinander sämtliche möglichen Kombinationen von 0 bis 255 durch-

gespielt. Den damit erzielten Effekt kennen Sie, wenn Sie schon einmal ROBOTER gespielt haben und dabei leider auch in einen "Zaun" geraten sind.

Zusammenfassung:
Wenn es Ihnen gelungen ist, den Robotern zu entkommen und Sie somit Zeit gefunden haben, sich auch den Lehrinhalten des vorliegenden Kapitels zu widmen, dann sollten Sie jetzt folgendes beherrschen:

- Erkennen von echtzeitverarbeiteten Programmen
- Benutzung von DEF FN und FN
- Überstreichen von revers gedruckten Texten
- "unsichtbare"Farbvorgabe für den Bildschirmbereich
- Bedeutung des Kommas in DATA-Anweisungen
- Verwendung von numerischen Variablen

```
1 REM - - - - - - - - -
2 REM  R O B O T E R
3 REM - - - - - - - - -
4 REM
5 REM (C) 1982 BY
6 REM
7 REM ANDREAS DRIPKE
8 REM
9 :
100 S=36876:L=S+2:C=S+3:VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 POKE657,128:POKEC,59:DIMA(15),B(9):PRINTCHR$(142)"
1010 PRINT"    *** ROBOTER ***":PRINT"IN DIESEM SPIEL MUES-
1020 PRINT"SEN SIE (●) VERSUCHEN,":PRINT"ALLE ROBOTER (◆) IN
1030 PRINT"EINEM LABYRINTH ZU":PRINT"VERNICHTEN! DIES GE-
1040 PRINT"LINGT IHNEN, WENN SIE":PRINT"SIE IN DIE ELEKTRI-
1050 PRINT"SCHEN ZAEUNE (▒) LOK-":PRINT"KEN.":GOSUB5000
1060 POKEC,42:PRINT"DOCH VORSICHT! SIE":PRINT"KOENNEN SICH NICHT
1070 PRINT"EINFACH HINTER EINIGEN":PRINT"ZAEUNEN VERBARRIKADIE-
1080 PRINT"REN. DIESE LOESEN SICH":PRINT"NAEMLICH NACH KONTAKT
1090 PRINT"MIT EINEM ROBOTER AUF.":GOSUB5000
1100 POKEC,76:PRINT"ACHTUNG! NICHT NUR DIE":PRINT"ROBOTER WERDEN IN DEN
1110 PRINT"ZAEUNEN ZERSTOERT. SIE":PRINT"UEBERLEBEN EINE BE-
1120 PRINT"RUEHRUNG LEIDER AUCH":PRINT"NICHT.":GOSUB5000
1130 POKEC,127:PRINT"SIE STEUERN MIT DEN":PRINT"FOLGENDEN BUCHSTABEN:
1140 PRINTSPC(5)"Q    W    E":PRINTSPC(7)"┌  ↑  ┐":PRINTSPC(8)"\ | /
1150 PRINTSPC(5)"A <- S -> D":PRINTSPC(8)"/ | \":PRINTSPC(7)"└  V  ┘
1160 PRINTSPC(5)"Z    X    C":PRINT"SIE BEHALTEN DIE GE-
1170 PRINT"WAEHLTE RICHTUNG BEI,":PRINT"BIS SIE DIE TASTE 'S'
```

```
1180 PRINT"(STOP) DRUECKEN.":GOSUB5000:FORI=1TO9:READB(I):NEXT
1190 DATA21,22,23,-1,,1,-23,-22,-21
1200 DEFFNX(I)=INT((I-O)/P):DEFFNY(I)=INT(((I-O)/P-FNX(I))*P+.5)
2000 POKEC,25:B$="":PRINT"■HALT --- WELCHEN":PRINT"■SCHWIERIGKEITSGRAD
2010 PRINT"■WAEHLEN SIE? (1-3): ";:B$="
2020 GOSUB5010:I=VAL(G$):ON-(I<1ORI>3)GOTO2020:R=INT(I*5-3*RND(1))
2030 PRINTG$:PRINT"■■■"R"ROBOTER MACHEN JAD■■G":PRINT"■AUF SIE. LASSEN SIE
2040 PRINT"■SICH NICHT ERWISCHEN!":FORI=1TO3E3:NEXT:O=VB:P=22:POKEC,8:PRINT"■■
2050 FORI=VCTOVC+505:POKEI,1+7*RND(1):NEXT
2060 FORI=OTOO+21:POKEI+484,102:POKEI,102:NEXT:FORI=O+PTOO+484STEPP:POKEI,102
2070 POKEI+21,102:NEXT:FORI=0TO49:POKEO+484*RND(1),102:NEXT
2080 X=INT(O+512*RND(1)):ON-(PEEK(X)<>32)GOTO2080:POKEX,81:POKEL,5:FORI=1TOR
2090 A(I)=INT(O+512*RND(1)):ON-(PEEK(A(I))<>32)GOTO2090:POKEA(I),90
2100 POKES,192+I:NEXT:POKES,0:POKEL,0:GOSUB4000
2110 Z=1
2120 A=Z
2130 G=A(A):A(A)=G+SGN(X1-FNX(G))*P+SGN(Y1-FNY(G)):POKEG,32:E=PEEK(A(A))
2140 POKEL,15:POKES,224:FORI=0TO19:NEXT:POKES,0:POKEL,0
2150 IFE=90THENA(A)=G
2160 IFE=81THEN3010
2170 IFE=102GOTO2210
2180 POKEL,8:FORI=0TO29:POKES,160:POKEA(A),90:POKES,192:POKEA(A),102:NEXT
2190 POKES,0:POKEL,0:POKEA(A),32:A(A)=A(R):R=R-1:IFR=0GOTO3040
2200 IFR<=ATHENA=R:GOTO2130
2210 POKEA(A),90:ON-(Z=INT(R/2))GOSUB4000:Z=Z+1:ON(Z<=R)+2GOTO2120,2110
3000 A=0:A(A)=X
3010 FORI=0TO9:POKEA(A),90:FORJ=0TO99:NEXT:POKEA(A),87:FORJ=0TO99:NEXT:NEXT
3020 FORI=1TO1E3:NEXT:PRINT"■■SIE SIND LEIDER GEFAN-":PRINT"GEN WORDEN!
```

```
3030 GOTO3090
3040 FORI=1TO2E3:NEXT:PRINT"■ALLE ROBOTER SIND VER-
3050 PRINT"NICHTET --- ICH FORDE-":PRINT"RE REVANCHE!":GOTO3090
3060 FORI=0TO2:FORJ=0TO255:POKEC,J:NEXT:NEXT:POKEC,27
3070 PRINT"■DAS WAREN 50000 VOLT":PRINT"■ZUVIEL. WARUM LAUFEN
3080 PRINT"■SIE AUCH AUSGERECHNET":PRINT"■IN EINEN ZAUN?
3090 POKEC,27:PRINT"■■NOCH EINMAL? (J/N)■":POKE198,0
3100 F=F+1AND1:PRINT"■"SPC(14)"   ■■■"SPC(F*2)"↑":FORI=0TO299:NEXT
3110 GETG$:ON-(G$="J")GOTO2000:IFG$<>"N"GOTO3100
3120 PRINT"■ANGST, WAS?":POKE657,0:END
4000 GETG$:FORI=1TO9:IFG$=MID$("ZXCASDQWE",I,1)THENB$=STR$(I)
4010 NEXT
4020 POKEX,32:X=X+B(VAL(B$)):ON-(PEEK(X)=102)GOTO3060:ON-(PEEK(X)=90)GOTO3000
4030 POKEX,81:X1=FNX(X):Y1=FNY(X):RETURN
5000 PRINTSPC(7)"■TASTE:";
5010 POKE198,0:POKEL,4
5020 F=F+1AND1:POKES,192+15*F:PRINTCHR$(18+F*128)"?■";:FORI=0TO49:GETG$
5030 IFG$THENPRINT" ■■";:POKES,0:POKEL,0:RETURN
5040 NEXT:GOTO5020
```

6. NUCLEAR



Bei NUCLEAR handelt es sich um ein Spiel für zwei Personen, bei dem es darum geht, schnell die sich ändernde Spielsituation zu überblicken und ebenso rasch zu entscheiden – bis hin zur nuklearen Kettenreaktion. Die genauen Spielregeln finden Sie im Programm selbst erklärt.
Um NUCLEAR noch auf der VC 20 Grundversion mit 5 KByte lauffähig zu machen, mußten wir leider auf Tonerzeugung verzichten. Der Speicher ist bereits ohne diese Option bis zum äußersten gefüllt. Wenn Sie aber mehr Speicherkapazität zur Verfügung haben, dann sollten Sie dem Programm unbedingt noch die folgenden Zeilen hinzufügen. Dadurch wird es mit Sound ausgestattet.

```
6010 PRINT"■■ ":FORA=0TO255:POKEC,A:NEXT:POKEC,58
6015 POKE36877,180:FORI=15TO0STEP-.05:POKE36878,I:NEXT:POKE36877,0
6017 RETURN
```

Wenn Sie in Ihren Programmen den Cursor ein paar Zeilen nach unten bewegen wollen, wie programmieren Sie das

üblicherweise? Wahrscheinlich nehmen Sie entweder ein paar reine PRINT-Befehle der Art

```
PRINT:PRINT:PRINT
```

oder Sie verwenden Die Steuerfunktion "Cursor nach unten" innerhalb einer PRINT-Anweisung. Das mag etwa so aussehen:

```
PRINT "cucucucu"      cu = Cursor nach unten
```

NUCLEAR zeigt Ihnen eine Alternative zu diesen beiden Methoden. Sehen Sie sich Zeile 1020 an. Sie finden dort den Befehl 'SPC(201)'. Normalerweise werden TAB und SPC höchstens mit der maximalen Bildschirmbreite als Argument benutzt, d. h., beim VC 20 mit 21. Wird ein größerer Wert eingesetzt, so rutscht der Cursor damit an die entsprechende Position der nächsten Zeile. Und da TAB und SPC Argumente bis hin zu 255 erlauben, bietet es sich geradezu an, auf diese Weise ein paar Leerzeilen einzubauen und gleichzeitig den nachfolgenden Text auf die Bildschirmseite zu setzen. Wir haben quasi zwei Richtungen in einem einzigen Befehl. 'SPC(51)', ebenfalls in Zeile 1020, wirkt ebenso über mehrere Zeilen.
Wir wollen uns nun mit einer recht ausgeklügelten und sehr interessanten Möglichkeit der ON..GOTO-Anweisung beschäftigen. Ein grundlegendes Beispiel für das, was wir jetzt besprechen wollen, finden Sie in Programmzeile 1040. Ähnliches kam bereits hin und wieder in den vorangegangenen Programmen vor, vermutlich haben Sie sich manchmal darüber gewundert. Rufen wir uns aber zunächst einmal ins Gedächtnis zurück, wie ON..GOTO funktioniert. Nehmen wir dazu ein Beispiel:

```
ON X GOTO 100,200,300,400,500
```

Wenn der Computer an diesen Befehl stößt, verzweigt er in Abhängigkeit davon, welchen Wert X hat, zu einer der in der Zeilennummernliste aufgeführten Zeilen. Für X gleich 1 erfolgt also ein Sprung zu Zeile 100, für X gleich 2 nach Zeile 200, für X gleich 3 nach Zeile 300 usw. Für X gleich 0 oder X größer als 5 verzweigt das Programm überhaupt nicht. Und genau dieser Umstand ist für unseren Trick von wesentlicher Bedeutung. Sehen Sie sich diese Programmzeile an:

```
100 ON X GOTO 1000,2000 : PRINT " Hallo"
```

Besitzt X die Werte 1 oder 2, erfolgt ein Sprung nach Zeile 1000 bezw. 2000. Andernfalls – X gleich 0 oder größer 2 – wird der nachfolgenden PRINT-Befehl ausgeführt und "Hallo" ausgedruckt. Etwas allgemeiner ausgedrückt bedeutet dies: Mit ON..GOTO kann nach bestimmten Bedingungen abgefragt werden. Sind die Bedingungen erfüllt, so wird dementsprechend verzweigt, andernfalls die Kommandos direkt nach ON..GOTO in derselben Programmzeile ausgeführt.
Machen Sie sich den Unterschied zu IF..THEN klar. Wenn bei IF..THEN die Bedingung erfüllt ist, so werden alle (!) Befehle nach THEN ausgeführt, auch wenn es mehrere sind, die durch Doppelpunkte voneinander getrennt sind.
Nun ist es oftmals erforderlich, eine Programmstruktur der Form

WENN...DANN...ANDERNFALLS...

Dies ist innerhalb einer Programmzeile mit IF..THEN nicht möglich.

```
100 IF X=1 THEN 1000 : PRINT " Hallo"
```

führt niemals zum Ausdruck von "Hallo". Entweder ist die Bedingung erfüllt – also X gleich 1 – dann erfolgt vorher ein Sprung zu Zeile 1000 oder die Bedingung ist nicht erfüllt – X ungleich 1 – und es wird keiner der Befehle nach THEN ausgeführt, also auch nicht der PRINT.
Mit ON..GOTO ist es nun möglich, dies mit der Struktur "Wenn..Dann..Andernfalls" zu programmieren. Wie das funktioniert? Ganz einfach:

```
100 ON X GOTO 1000 : PRINT "Hallo"
```

Was passiert? Wenn X gleich 1 ist – Bedingung erfüllt – verzweigt der Programmlauf zu Zeile 1000. Andernfalls – nun andernfalls wird der Befehl 'PRINT"Hallo" ' abgearbeitet.
Wenn Ihnen an diesem letzten Abschnitt noch etwas unklar geblieben sein sollte, so lesen Sie ihn am besten gleich noch ein zweites Mal. Dies war nämlich erst der erste Schritt zum Verständnis von Zeile 1040. Stellen Sie sicher, daß Sie diesen voll verstanden haben.
Der zweite Schritt erfordert, daß wir uns mit vergleichenden Ausdrücken befassen. Damit Sie sehen, was darunter zu verstehen ist, will ich Ihnen gleich ein paar Beispiele präsentieren.

X = 1
A < > 5
G$ = "N"

"Vergleichender Ausdruck" meint also ganz einfach, daß zwei Dinge miteinander verglichen werden, zum Beispiel auf Gleichheit oder Ungleichheit, aber auch auf größer, kleiner usw. Wenn der Computer nun solche Vergleiche durchführt, dann kommt er entweder zu dem Ergebnis, daß der Vergleich stimmt – daß also X gleich 1 oder A größer als 5 oder G$ gleich "N" ist – oder er kommt zu dem Ergebnis, daß der Vergleich falsch ist. Das Ergebnis ist entweder –1, wenn der Ausdruck wahr ist, oder aber 0, wenn der Ausdruck falsch ist.
Sie sehen dies am einfachsten wieder an einem Beispiel. Tippen Sie einmal

```
? 3=3
```

ein. Was erhalten Sie? Ist der Vergleich wahr oder falsch? Geben Sie nun

```
? 2=3
```

ein. Was ist diesmal das Ergebnis?
Um uns in unseren Betrachtungen der Programmzeile 1040 zu nähern, geben Sie jetzt

```
G$="N"
```

und anschließend

```
? G$="N"
```

ein. Beachten Sie den Unterschied zwischen der Wertzuweisung im ersten Fall und dem Vergleich im zweiten Fall.
Jetzt, nachdem Sie ON..GOTO so weit kennengelernt haben und auch wissen, daß Vergleiche der gezeigten Art mit 0 bzw. –1 als falsch bzw. wahr verarbeitet werden, können wir uns endlich der anfangs erwähnten Programmzeile 1040 in NUCLEAR zuwenden. ' GETG$' dürfte Ihnen klar sein. Es geht darum, die in Zeile 1030 gestellte Frage nach den Spielregeln mit "J" oder "N" zu beantworten. Gehen wir davon, Sie müssen einen Moment darüber nachdenken und drücken deshalb zunächst überhaupt keine Taste. Also ist G$ gleich " ", sprich leer. Die Bedingung ' (G$="N")' ist somit nicht erfüllt, das Ergebnis dieses Vergleiches ist 0 (Vergleich falsch).

Das Minuszeichen direkt vor der Klammer ändert daran nichts, denn schließlich ist –0 dasselbe wie 0. Die ON..GOTO-Anweisung hat jetzt also die Form 'ON 0 GOTO 3000'. Wie Sie wissen, erfolgt bei 0 kein Sprung, hier also keine Verzweigung zu Zeile 3000. Vielmehr werden die Befehle direkt hinter dieser ON..GOTO-Anweisung ausgeführt. In Zeile 1040 folgt eine IF..THEN-Abfrage. Da G$ leer und somit ungleich "J" ist, erfolgt an dieser Stelle der Rücksprung zu Zeile 1040. Das Programm befindet sich in einer Schleife, die solange nicht verlassen wird, bis eine Taste gedrückt wird.
Nehmen wir nun an, Sie betätigen "J", da Sie die Spielregeln noch nicht kennen und sie folglich zu sehen wünschen. Die Bedingung '(G$="N")' ist nach wie vor falsch (G$ ist eben nicht gleich "N", sondern gleich "J"), das Ergebnis des Vergleiches ist wiederum 0. Der Sprung zu Zeile 3000 wird nicht ausgeführt. Die IF..THEN-Abfrage führt aber diesmal nicht zum Rücksprung nach Zeile 1040, da G$ diesmal gerade nicht ungleich "J" ist, sondern im Gegenteil gleich "J" ist. Folglich kommt es jetzt zum Ausdruck der Spielregeln ab Zeile 2000.
Was geschieht aber nun, wenn Sie die Regeln schon kennen und deswegen auf die entsprechende Frage in Zeile 1030 mit "N" antworten. G$ wird dann gleich "N". Das aber bedeutet, daß der Vergleich '(G$="N")' diesmal wahr ist und somit als Ergebnis –1 liefert. Durch das Minuszeichen direkt vor der Klammer wird daraus 1 und somit entsteht praktisch die Anweisung 'ON 1 GOTO 3000'. In diesem Fall überspringt das Programm also die Spielregeln bei 2000 und fährt direkt bei Zeile 3000 fort.
Tja, damit wäre die zunächst so eigenartig aussehende Programmzeile 1040 erklärt. Am besten lesen Sie sich die Erläuterungen dazu in einiger Zeit nochmals durch, damit Sie mit dieser Programmiertechnik vertraut werden. Sehen Sie sich auch die vorangegangenen Programme auf die Benutzung von 'ON –(..) GOTO' an. In NUCLEAR selbst finden Sie weitere Beispiele in den Programmzeilen 4060, 4180 und 7010.
Sind Sie eigentlich mit den beiden im Commodore BASIC verfügbaren Zeitvariablen TI und TI$ voll vertraut? Wenn ja, dann können Sie diesen Abschnitt getrost überschlagen. Wir wollen uns jetzt nämlich die Zeile 4040 mit der Anweisung 'TI$="000000" zum Anlaß nehmen, über TI und TI$ zu reden.
Mit der Zuweisung zu TI$ in Zeile 4040 wird die Zeit für den Computer auf Null zurückgesetzt. TI$ ist eine Spezialvariable, die die Zeit im Format HHMMSS (Stunden, Minuten, Sekunden, jeweils zwei Stellen) enthält. Sie sehen das sehr leicht an folgendem Programm:

```
100 PRINT"CLR"      CLR = Bildschirm löschen
110 PRINT"HOME"     HOME = Cursor auf HOME-Position
120 PRINT TI$
130 GOTO 110
```

Das kleine Programm blendet die laufende Zeit in der linken Bildschirmecke ein. Gestellt werden kann TI$ mit

```
TI$="HHMMSS"
```

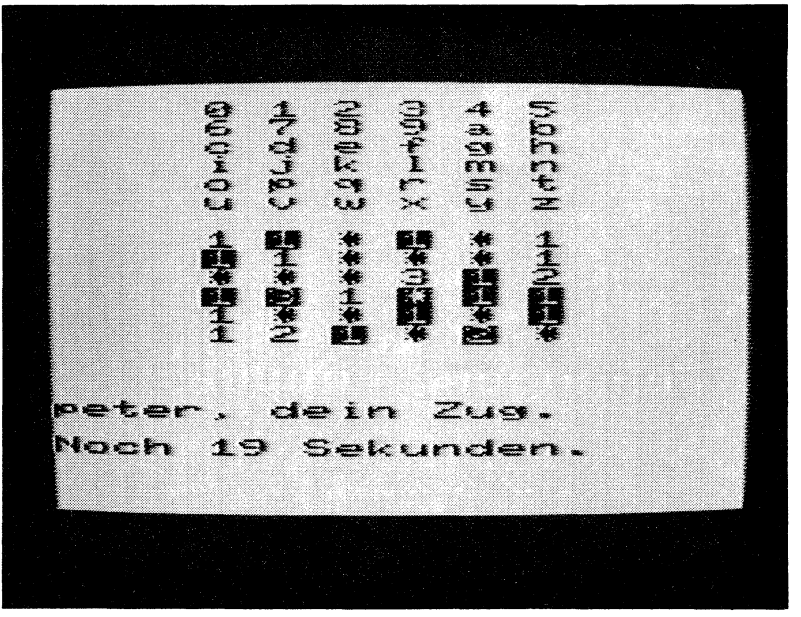
Nehmen wir an, die Uhrzeit betrage 21 Uhr, 5 Minuten und 20 Sekunden, dann geben wir

```
TI$="210520"
```

ein. Der String muß exakt sechs Stellen lang sein, anderenfalls kommt es zu einem ILLEGAL QUANTITY ERROR.
Die zweite Zeitvariable ist TI. TI enthält die Zeit in 1/60 Sekunden. Der Variablen TI einen Wert zuweisen ist nicht möglich.

```
TI=0
```

zum Beispiel führt zu einem SYNTAX ERROR. TI kann also nicht direkt ein Wert zugewiesen werden, wohl aber indirekt über TI$. Das wird Ihnen klar, wenn Sie um das Verhältnis zwischen TI und TI$ wissen. Die eigentliche Uhr liegt in TI$. TI nimmt nun diesen Wert in TI$ als Grundlage und berechnet daraus die Zeit in 1/60 Sekunde. TI hängt also von TI$ ab. Das bedeutet, wenn TI$ eine bestimmte Uhrzeit zugewiesen wird, dann erhält TI die entsprechende Zeit ebenfalls – allerdings im 1/60-Format – zugeordnet.
Jetzt wird Ihnen sicher auch verständlich, warum in NUCLEAR in Zeile 4040 TI$ gleich "000000" gesetzt wird, obwohl TI$ im weiteren Verlauf des Programmes gar nicht mehr benutzt wird. Damit wird nämlich erreicht, daß auch TI gleich 0 wird. Und TI wiederum wird in Zeile 4050 verwendet, um die für den Spielzug noch verbleibende Zeit in Sekunden auszugeben.
Beachten Sie in Ihren eigenen Programmen übrigens unbedingt, daß die Variablen TI und TI$ die gezeigte Sonderfunktion besitzen. Die Variablen können also nicht wie normale andere Variablen behandelt werden. Eine andere Spezialvariable, die ebenfalls normalerweise nicht eingesetzt werden darf, haben wir schon kennengelernt, es ist ST.
Im letzten Kapitel hatte ich Sie darauf aufmerksam gemacht, daß auch ein so mathematischer Befehl wie DEF FN durchaus in einem Spielprogramm sinnvoll sein kann. NUCLEAR liefert



einen weiteren Beweis dafür, daß die mathematischen Funktionen des Computers auch oft dort mit Erfolg eingesetzt werden können, wo es vordergründig nicht um Mathematik geht. Sehen Sie sich die Zeilen 4100 bis 4120 an. Die Kreisfunktionen Sinus (SIN) und Cosinus (COS) werden benutzt, um die um ein "explodierendes" Feld herumliegenden Felder um jeweils eins zu erhöhen. Die FOR-NEXT-Schleife in Zeile 4100 sorgt dafür, daß die Funktionen im Abstand π/2 von 0 bis 2*π aufgerufen werden. Bekannterweise nehmen Sinus und Cosinus an den dadurch angesprochenen Argumenten die Werte 0, 1 und –1 an, und zwar mit einer Verschiebung von π/2. Damit werden in NUCLEAR nacheinander die umliegenden Felder angesprochen. Die Addition von 0.1 in Zeile 4110 dient ausschließlich dazu, etwaige Rundungsfehler auszugleichen.
So, damit haben wir für diesmal genug besprochen. Schließlich sollen die Erklärungen nicht die Spielprogramme verdrängen, sondern nur eine Bereicherung bringen. Laden Sie also NUCLEAR und einen Mitspieler ein und lassen Sie sich von NUCLEAR faszinieren.

Zusammenfassung:
Ich hoffe, daß die Menge dessen, was wir in diesem Kapitel Neues besprochen haben, noch nicht die für Sie "kritische Masse" erreicht hat. Im einzelnen waren dies:

- TAB und SPC zur Erzeugung von Leerzeilen
- Verwendung von ON..GOTO zur Programmierung von Strukturen der Art "Wenn..Dann..Andernfalls"
- Auswertung vergleichender Ausdrücke durch den Computer
- Benutzung der Zeitvariablen TI und TI$

```
1 REM ** NUCLEAR **
2 REM
3 REM  (C) JAN 1982
4 REM
5 REM ANDREAS DRIPKE
6 :
100 C=36879:T=198:POKE657,128:PRINTCHR$(14)
1000 DIMI(5,5),J(5,5):I(0,0)=2:I(0,5)=2:I(5,0)=2:I(5,5)=2:FORI=1TO4:FORJ=1TO4
1010 I(I,J)=4:NEXT:NEXT:FORI=0TO5STEP5:FORJ=1TO4:I(I,J)=3:I(J,I)=3:NEXT:NEXT
1020 POKEC,61:PRINT"■■"SPC(201)"*** NUCLEAR ***"SPC(51)"(C) BY A.DRIPKE
1030 FORI=1TO3E3:NEXT:POKEC,43:PRINT"■■■■■SPIELREGELN (J/N)?":POKET,0
1040 GETG$:ON-(G$="N")GOTO3000:IFG$<>"J"GOTO1040
2000 POKEC,108:PRINT"■■"SPC(7)"________":PRINTSPC(7)"■NUCLEAR■
2010 PRINT"■IN DIESEM SPIEL SET-":PRINT"■ZEN ZWEI SPIELER AB-
2020 PRINT"■WECHSELND SPIELSTEINE":PRINT"■AUF EIN 6*6 FELDER
2030 PRINT"■GROSSES BRETT.":GOSUB7000:POKEC,156:PRINT"■■WENN EIN FELD SEINE■
2040 PRINTSPC(4)"________________":PRINTSPC(4)"■KRITISCHE MASSE■
2050 PRINT"■ERREICHT HAT, EXPLO-":PRINT"■DIERT ES UND SCHICKT
2060 PRINT"■PARTIKEL AUF DIE":PRINT"■NACHBARFELDER (NICHT":PRINT"■DIAGONAL).
2070 GOSUB7000:POKEC,9:PRINT"■■DIE 'KRITISCHE MASSE'":PRINT"■EINES FELDES WIRD
2080 PRINT"■DURCH DIE ANZAHL SEI-":PRINT"■NER NACHBARN BESTIMMT■■":FORI=0TO5
2090 PRINT:PRINTSPC(4):FORJ=0TO5:PRINTI(I,J)"■";:NEXT:NEXT:PRINT:GOSUB7000
2100 POKEC,58:PRINT"■■MAN KANN AUF LEERE":PRINT"■ODER EIGENE FELDER
2110 PRINT"■SETZEN. DIE SPIEL-":PRINT"■STEINE UNTERSCHEIDEN
2120 PRINT"■SICH DADURCH, DASS":PRINT"■EIN SPIELER POSITIVE
2130 PRINT"■STEINE UND DER ANDERE":PRINT"_________":PRINT"■NEGATIVE■ HAT.
2140 GOSUB7000:POKEC,123:PRINT"■■WENN EIN FELD EXPLO-
2150 PRINT"■DIERT, WIRD JEDES":PRINT"■FELD EROBERT, AUF DEM
```

```
2160 PRINT"■DIE PARTIKEL LANDEN.":GOSUB7000:POKEC,77
2170 PRINT"□■DENN DAS BRETT SICH":PRINT"■ZU FUELLEN BEGINNT,
2180 PRINT"■KANN EINE EXPLOSION":PRINT"■ANDERE NACH SICH ZIE-
2190 PRINT"■HEN - KETTENREAKTION.":GOSUB7000:POKEC,44
2200 PRINT"□■DAS SPIELFELD IST UN-":PRINT"■TEN. DAS OBERE FELD
2210 PRINT"■GIBT AN, WELCHE TASTE":PRINT"■FUER WELCHES FELD GE-
2220 PRINT"■DRUECKT WERDEN MUSS.":PRINT"■EIN SPIELER GEWINNT
2230 PRINT"■DANN, WENN SEIN GEG-":PRINT"■NER KEINE STEINE MEHR":PRINT"■HAT.
2240 GOSUB7000
3000 OPEN1,0:POKEC,58:PRINT"□■NAME SPIELER #1":PRINT"■-> ";:INPUT#1,N2$:PRINT
3010 N2$="■"+N2$+"■":PRINT"■■NAME SPIELER #2":PRINT"■-> ";:INPUT#1,N1$:PRINT
3020 CLOSE1:PRINT"□";:FORI=0TO5:PRINT:PRINTSPC(5):FORJ=0TO5
3030 PRINTCHR$(48+I*6+J-7*(I*6+J>9))" ";:NEXT:NEXT:PRINT:FORI=0TO5:PRINT
3040 PRINTSPC(5):FORJ=0TO5:PRINT"* ";:NEXT:NEXT:PRINT
4000 PRINT:P$=N1$:Z=1:IFRND(1)<.5THENZ=-1:P$=N2$
4010 PRINTP$" BEGINNT.■■■":GOTO4040
4020 POKET,0:PRINT:Z=-Z:P$=N1$:GOSUB6000:IFZ=1THENP$=N2$
4030 PRINT"□□□□□":FORI=0TO3:PRINT"                    ":NEXT
4040 TI$="000000":PRINT"□□"P$", DEIN ZUG.■■
4050 A=INT(21-TI/60):PRINT"□NOCH"A"■■ SEKUNDEN. ":IFA=0GOTO4020
4060 GETG$:ON-(G$="")GOTO4050:J=ASC(G$)-48+7*(G$>"@"):IFJ<0ORJ>35GOTO4050
4070 I=INT(J/6):J=J-I*6:ON-(J(I,J)/Z<0)GOTO4050:N=0:J(I,J)=J(I,J)+Z
4080 B=0:FORI=0TO5:FORJ=0TO5:IFABS(J(I,J))<ABS(I(I,J))GOTO5000
4090 E=J(I,J):J(I,J)=8*Z:GOSUB6000:J(I,J)=Z*(ABS(E)-ABS(I(I,J)))
4100 FORE=π/2TO2*πSTEPπ/2
4110 V=-(INT(COS(E)+.1))+I:W=INT(SIN(E)+.1)+J:IFV<0ORV>5ORW<0ORW>5GOTO4130
4120 J(V,W)=Z*(ABS(J(V,W))+1)
4130 NEXT:GOSUB6000:B=1:N=N+1:FORV=0TO5:FORW=0TO5:IFJ(V,W)=0GOTO4160
```

```
4140 IFSGN(J(V,W))=ZGOTO4160
4150 V=5:W=5:NEXT:NEXT:GOTO5000
4160 NEXT:NEXT:PRINTP$" GEWINNT!  ":GOSUB7000:PRINT"NOCH EINMAL?
4170 GETG$:IFG$="N"THENPRINT"TSCHUESS!":POKE657,0:END
4180 ON-(G$<>"J")GOTO4170:FORI=0TO5:FORJ=0TO5:J(I,J)=0:NEXT:NEXT:GOTO3020
5000 NEXT:NEXT:ONBGOTO4080:ONN+1GOTO4020:G$="":IFN>1THENG$="EN
5010 PRINTN"  EXPLOSION"G$".      ":FORI=1TO2E3:NEXT:GOTO4020
6000 PRINT"          ":FORV=0TO5:PRINTSPC(5):FORW=0TO5:IFABS(J(V,W))-8GOTO6020
6010 PRINT"  ":FORA=0TO255:POKEC,A:NEXT:POKEC,58:RETURN
6020 IFJ(V,W)=0THENPRINT"* ";:GOTO6040
6030 O=J(V,W):PRINTCHR$((O<1)*-18)MID$(STR$(ABS(O)),2)" ";
6040 NEXT:PRINT:NEXT:PRINT"     ":RETURN
7000 PRINTSPC(52)"_______":PRINTSPC(8)"RETURN":POKET,0
7010 GETG$:ON-(G$<>CHR$(13))GOTO7010:RETURN
```

7. FLAK

Flak ist ein Actionspiel, bei dem es gilt, feindliche Raketen vom Himmel zu holen. Damit Flak noch auf einem VC 20 ohne Speichererweiterung läuft, habe ich das Programm in zwei Teile zerlegt, der Spielanleitung einerseits und dem eigentlichen Spielprogramm andererseits. Wenn Sie die beiden Programmteile in dieser Reihenfolge hintereinander auf eine Kassette aufzeichnen, dann lädt die Anleitung das Spielprogramm automatisch nach. Wie das funktioniert, werden wir uns gleich ansehen. Zuvor aber noch ein Tip für diejenigen, die keinen Kassettenrekorder zum Abspeichern ihrer Programme benutzen, sondern eine Floppydisk-Station. Ändern Sie in diesem Fall die folgenden Zeilen bzw. fügen Sie sie hinzu:

```
1165 PRINT"■■■    "CHR$(34)"FLAK"CHR$(34)",8■■■
```

Durch diese Änderungen wird der zweite Programmteil von Diskette nachgeladen, vorausgesetzt, daß er unter dem Namen "FLAK" abgespeichert ist.
Wir wollen uns aber jetzt anschauen, wie das automatische Nachladen von Kassette funktioniert. Verantwortlich dafür ist Zeile 1170 (in der Originalversion ohne die obigen Änderungen). Vielleicht sind Sie jetzt erstaunt, daß das Nachladen aus nur zwei POKE-Befehlen und END besteht. Was steckt nun dahinter?
Sie wissen bereits aus Kapitel 3, daß sich der Computer Tastendrücke im voraus merken kann. Wir haben gelernt, daß in Speicherzelle 198 steht, wieviele Tastendrücke jeweils gültig sind. Mit 'POKE198,0' haben wir den Computer so programmiert, daß er vor GET alle vorangegangenen Tastendrücke wieder "vergißt". In Flak finden Sie in Zeile 1170 den Befehl

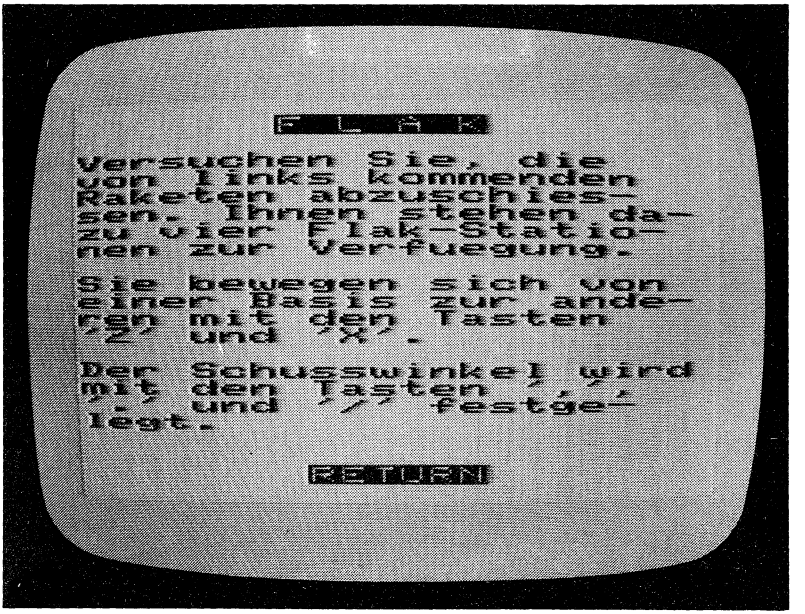
```
POKE 198,1
```

wodurch also ein Tastendruck als gültig erklärt wird. Wir sind zwar in Kapitel 3 nicht näher darauf eingegangen - aber natürlich reicht es dem Computer nicht, zu wissen, wieviele Tasten betätigt wurden, sondern er muß auch irgendwo abspeichern, welche Tasten das waren. Dies legt er Wert für Wert – jeder Taste ist eine bestimmte Zahl zugeordnet – ab Speicherzelle 631 ab. Wie weit die Liste reicht, hängt davon ab, wieviele

Tastendrücke in Speicherzelle 198 als gültig erklärt sind.In unserer Zeile 1170 ist dies genau ein Tastendruck (POKE198,1), der direkt vorher vom Programm selbst in die Liste ab 631 eingeschrieben wird (POKE631,131). Nun ist es eigentlich nur noch wichtig zu wissen, welcher Taste die Zahl 131 in dem Befehl

POKE 631,131

entspricht. Wir hatten ja gesagt, daß jeder Taste eine bestimmte Zahl zugeordnet ist. Nun, 131 entspricht der Tastenkombination SHIFT und RUN/STOP, mithin also der RUN-Funktion. Wie Sie sicher wissen, lädt die RUN-Taste das nächste auf Kassette befindliche Programm und startet dieses automatisch (Machen Sie sich diesen Unterschied zwischen der RUN-Taste und dem Eintippen des Wortes "RUN" klar).
Zeile 1170 hat dieselbe Wirkung, als wenn Sie RUN von Hand drücken. Zunächst wird der Tastenwert in 631 gebracht und anschließend für gültig erklärt. END bricht den Programmablauf ab und es kommt zur Ausführung der RUN-Tastenfunktion.



Die Änderungen für das automatische Nachladen von Floppydisk-Station funktionieren übrigens recht ähnlich. Da der VC

20 keine RUN-Tastenfunktion kennt, die auf die Floppydisk wirkt, werden die entsprechenden Texte 'LOAD"FLAK",8' und 'RUN' auf den Bildschirm geschrieben (Zeile 1170). Dann wird der Tastaturpuffer zweimal mit dem Wert 13 gefüllt und diese zwei Zahlen für gültig erklärt (Zeile 1180). 13 ist der Code für die RETURN-Taste. Es ist nun nur noch eine Frage der Geschicklichkeit im Umgang mit der Cursorsteuerung (Zeile 1170), so daß der Cursor nach dem END direkt auf der Zeile 'LOAD"FLAK",8' steht. Das (vom Computer simulierte) Drücken der RETURN-Taste lädt dann den zweiten Programmteil, wonach der Cursor auf dem Wort 'RUN' steht. Nun wird das zweite RETURN ausgeführt und somit das Programm gestartet.

Beachten Sie, daß im Unterschied zum Nachladen von Rekorder nicht der eigentlich LOAD-Befehl in den Tastaturpuffer "gePOKEt" wird, sondern dieser vielmehr mit PRINT auf den Bildschirm gebracht wird und nur das Drücken der RETURN-Taste per POKE simuliert wird. Wollte man den gesamten Text im Tastaturpuffer unterbringen, dann müßte man diesen erweitern (Sie erinnern sich, das ist mit 'POKE649,X' möglich) und eine ganze Reihe von POKEs hinzufügen, es wäre auf jeden Fall wesentlich umständlicher.

Wenn Sie FLAK schon eingetippt und gespielt haben, dann wissen Sie, daß es mit einer Laufschrift beginnt. (Wir sind immer noch im Anleitungsteil). Sehen Sie sich einmal die Routine dazu – sie liegt in den Zeilen 1000 bis 1040 – genauer an. Der Text ist in Zeile 1040 unter DATA abgelegt. Sie dürfen die Laufschriftroutine gerne vollständig übernehmen und in Ihre eigenen Programme einbauen. Damit Sie dann aber auch die Funktion verstehen, wollen wir die Routine hier erst einmal besprechen.

In den Zeilen 1000 und 1010 wird die Röhre gezeichnet, in der sich die Laufschrift dann später bewegen soll. Die Anweisung 'FORI=0TO2' in Zeile 1010 legt fest, daß insgesamt drei Texte über den Bildschirm laufen sollen. Danach liest das Programm in Zeile 1020 mit 'READA$' den Text ein. A$ wird an den schon in Zeile 1010 definierten String I$, der aus lauter Leerzeichen besteht, angehängt und ein weiteres Leerzeichen nachgestellt. Aus diesem gesamten String wird nun mit MID$ ein Bereich von 20 Zeichen herausgefiltert, und zwar mit jedem Durchlauf ('FORJ=1TOLEN(A$)') um jeweils eins nach rechts verschoben. Dadurch entsteht der perfekte Eindruck, daß der Text über den Bildschirm wandert. Die POKE-Befehle in den Zeilen 1020 und 1030 dienen ausschließlich der Tonerzeugung und haben mit der eigentlichen Laufschrift nichts zu tun.

So, ich hoffe, daß Ihnen jetzt die Routine zur Erzeugung der Laufschrift klar geworden ist. Wie gesagt, Sie dürfen sie in Ihren eigenen Programmen gerne benutzen. Überlegen Sie sich doch auch einmal, welche Änderungen notwendig sind, damit der Text von links nach rechts durch die Röhre läuft.
Damit wollen wir das Anleitungsprogramm verlassen und uns dem eigentlichen Spielprogramm zuwenden. Lassen Sie nach dem Einladen der Anleitung die PLAY-Taste am Rekorder niedergedrückt, damit der zweite Programmteil eingelesen werden kann.
Wenn Sie FLAK schon gespielt haben, dann wissen Sie, daß vor jedem neuen Durchgang die Meldung erscheint, wieviele Raketen "in unterschiedlichen Höhen von Westen" zu erwarten sind. Der Text wird schnell abwechseln revers (RVS ON) und normal gedruckt, wodurch der blinkende Eindruck entsteht. Wenn Sie sich das Programmlisting ansehen, dann finden Sie dort aber den Text nur ein einziges Mal, und zwar in Normalschrift. Wie also wird das Umschalten zwischen normal- und RVS-Schrift programmiert? Die dafür zuständige Routine liegt bei 1060 bis 1090. Die Variable U enthält die Anzahl der zu erwartenden Raketen. In einer Schleife ('FORI=0TO15') kommt es sechzehnmal zum Blinken. Dabei wird in den Zeilen 1070 bis 1080 dem eigentlichen Text die Variable A$ vorangestellt. A$ enthält jeden Durchlauf abwechselnd entweder das Steuerzeichen RVS ON oder aber RVS OFF.
Und dieser ständige Wechsel von A$ ist in Zeile 1060 programiert. Durch MID$ wird A$ jeweils eines der beiden Steuerzeichen RVS ON und RVS OFF zugewiesen. Der entscheidende Befehl dabei ist

```
F=NOTF
```

Die NOT-Funktion wird im allgemeinen recht selten eingesetzt, hier zeigt sich aber, daß sie hin und wieder doch ganz brauchbar ist. NOT dient der Verneinung von Ausdrücken gleich welcher Art. So kann man zum Beispiel statt

```
IF X=5 THEN...
```

auch

```
IF NOT(X<>5) THEN...
```

schreiben. In diesem Beispiel macht NOT aus "gleich" das Gegenteil "ungleich". Allgemein gilt:

NOTX ist dasselbe wie – (X+1)

Statt 'F=NOTF' könnten wir also auch

F=–(F+1)

schreiben. Spielen wir nun einmal die Möglichkeiten in FLAK durch. Zunächst wird F gleich 0 gesetzt (Zeile 1060). Das erste Mal, wenn das Programm an 'F=NOTF' stößt, ist F also gleich 0. Setzen wir diese Null in die Ersatzformel für NOT ein, so erhalten wir

F=–(0+1)

mithin also 'F=–1'. Im MID$-Befehl wird 2 dazuaddiert, so daß die MID$-Funktion die Form

A$=MID$(" ",1,1) Die Steuerzeichen sind weggelassen

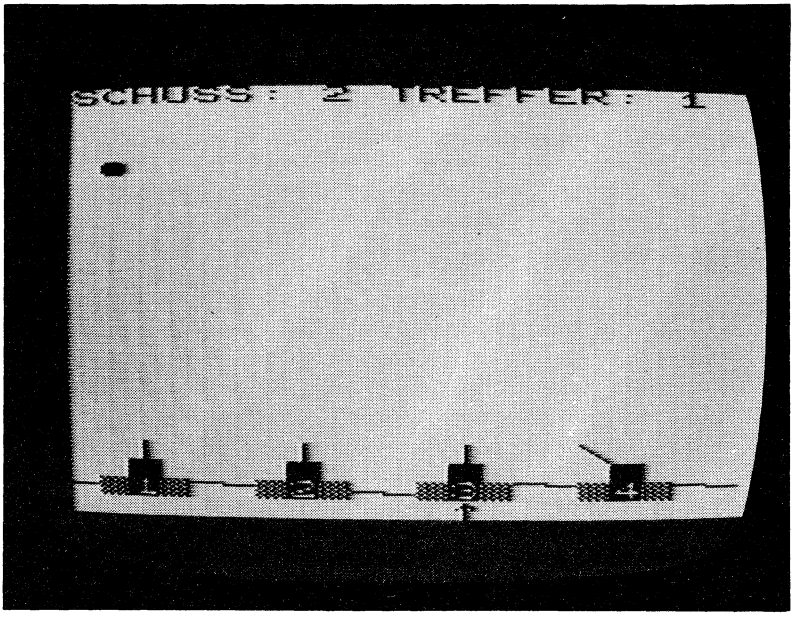
annimmt. Diesmal wird also das erste Steuerzeichen, nämlich RVS ON, an A$ zugewiesen. Wenn jetzt A$ den nachfolgenden Texten jeweils vorangestellt wird, so werden diese natürlich revers dargestellt, bis hin zu NEXT in Zeile 1080.
Beim zweiten Durchlauf der Schleife kommt das Programm wiederum an 'F=NOTF'. Diesmal ist F gleich –1, die Ersatzformel für NOT liefert mithin

F=–(–1+1)

also 0. Die MID$-Funktion sieht dann so aus:

A$=MID$(" ",2,1) Die Steuerzeichen sind weggelassen

Es wird das zweite Zeichen, also RVS OFF, genommen. In dieser Schleife wird folglich der gesamte Text in Normaldarstellung ausgegeben. Tja, und auf diese Art und Weise entsteht das Blinken. Die GOSUB-Anweisung in Zeile 1080 dient lediglich der Tonerzeugung und hat mit den von uns betrachteten Tatsachen nichts zu tun. Zeile 1090 hat die Funktion, den Text vollständig zu löschen, damit das Spiel beginnen kann.
Während des Spiels werden in der oberen Bildschirmzeile die Schüsse und die Treffer angezeigt. Sowohl für die Raketen als auch für die Abwehrraketen existiert diese obere Zeile prak-



tisch nicht. Wie können wir so eine Einblendung einer Zeile in das laufende Bildschirmgeschehen programmieren? Wir wissen bereits, daß in Zeile 100 der Variablen VB die kleinste Bildschirmadresse (Position links oben) zugewiesen wird. Um nun im Programm zu verhindern, daß irgendeines der "Flugobjekte" oben aus dem Bildschirm "herausfliegt", muß ständig geprüft werden, ob die Adressen der beiden Flugzeichen (Vollkreis und leerer Kreis) kleiner als VB ist. Ist dies der Fall, dann wird dieses "Flugobjekt" gestrichen. In FLAK wird nun statt VB der Wert VB+22 verwendet. Nehmen Sie jetzt einmal das Bildschirm-Entwurfsblatt aus Kapitel 2 zur Hand. Sie sehen, wie durch den Wert VB+22 als niedrigst mögliche Adresse die oberste Bildschirmzeile für die "Flugobjekte" unzugänglich wird. Eigentlich ganz einfach, nicht wahr?
Wir wollen uns das Gesagte an FLAK verdeutlichen. Da die Rakete (Vollkreis, Bildschirmcode 81) ja nur horizontal fliegt, gilt für sie nur die Bedingung, daß sie nicht in der obersten Bildschirmzeile ihren Flug beginnen darf. Wie das programmiert ist, können wir in der Programmzeile 2000 sehen. Die Variable X, die die Position der Rakete angibt, kann niemals kleiner als 22+VB werden.

X=22+(Zufallszahl zwischen 0 und 373)+VB

Falls Sie sich über die Erzeugung von Zufallszahlen nicht mehr so ganz im klaren sein sollten, schlagen Sie noch einmal in Kapitel 1 nach.
Dafür, daß die Abwehrraketen (leerer Kreis, Bildschirmcode 87) nicht in Adressen kleiner als VB+11 geraten, ist in Zeile 2110 gesorgt.

```
IFP(I)<VB+22THENP(I)=0
```

eliminiert jede Abwehrrakete, bevor sie in die obere Bildschirmzeile eindringen kann. P(I) beinhaltet die Positionsangabe der einzelnen Abwehrraketen. Ersetzen Sie die 22 in Programmzeile 2110 doch einmal durch den Wert 44 (VB+44). Nach RUN können Sie dann beobachten, daß nun die oberen zwei Bildschirmzeilen unberührt bleiben.
Wir haben nicht nur die Möglichkeit, die oberen Bildschirmzeilen einzublenden, es geht genau so gut auch mit den unteren. Allerdings kann uns hier FLAK nicht mehr als Beispiel dienen, da hierbei dieser Fall sowieso niemals auftreten kann. Nehmen wir deshalb einmal an, in einem anderen Programm stünde die Adresse, wo sich unser bewegliches Graphikzeichen befindet, in der Variablen P (für Position). Eine Programmzeile mit einer Bewegung nach unten und der Einblendung der untersten Bildschirmzeile könnte dann so aussehen:

```
100 POKE, 32:P=P+22:IFP<VB+483THENPOKEP,81
```

Der erste POKE mit 32 dient zum Löschen der "vorherigen Position, danach wird diese um 22 erhöht, was einer Bewegung um exakt eins nach unten entspricht. An die neu errechnete Position gesetzt wird das Graphikzeichen (Code 81) aber nur solange die unterste Bildschirmzeile noch nicht erreicht ist. Andernfalls wäre P größer/gleich VB+505–22, also gleich VB+483.
Und eine beliebige Zeile in der Mitte des Bildschirmes? Auch sie können wir einblenden. Für die 12. Zeile zum Beispiel sieht das folgendermaßen aus:

```
100POKEP,32:P=P+22:IFP<VB+242ORP>VB+264THENPOKEP,81
```

Für eine Bewegung nach oben ersetzt man den Ausdruck 'P=P+22' einfach durch 'P=P–22'. Warum? Die Position links oben hat die niedrigste Bildschirmadresse (VB), die Position rechts unten die höchste (VB+505).
Wenn Sie das bisher Gesagte verstanden haben, dann sollte

es Ihnen jetzt nicht mehr schwerfallen, auch mehrere Zeilen oben, unten oder in der Mitte in ein bewegliches Bildschirmgeschehen einzublenden. Ich gebe Ihnen hier noch ein Beispiel für die unteren 5 Zeilen:

```
100 POKEP,32:P=P+22:IFP<VB+(22*18)THENPOKEP,81
```

Alles klar? Gut, dann will ich Sie nun mit FLAK allein lassen und wünsche Ihnen viel Spaß beim Spielen.

Zusammenfassung:
Inhalt dieses Kapitels waren folgende Themen:
- Nachladen eines Programmes von Kassette oder Diskette (Tastaturpuffermethode)
- Laufschriftroutine
- NOT-Funktion
- Einblendung von Zeilen in das laufende Bildschirmgeschehen

```
1 REM ---FLAK/ANL---
2 REM
3 REM -(C) 1982 BY -
4 REM
5 REM ANDREAS DRIPKE
6 REM
7 REM   WIESBADEN
8 REM
9 :
100 IFPEEK(158)OR191ANDSTTHENPRINT"■?LOAD ERROR";:END
1000 POKE657,128:PRINTCHR$(14)"■■■■■■■■■":FORI=0TO1:PRINT"■■";:FORJ=0TO19
1010 PRINT"─";:NEXT:PRINT:NEXT:PRINT"■";:FORI=0TO19:I$=I$+" ":NEXT:FORI=0TO2
1020 READA$:A$=I$+A$+" ":FORJ=1TOLEN(A$):PRINT"■■"MID$(A$,J,20):POKE36878,15
1030 POKE36876,230:FORK=0TO99:NEXT:POKE36876,0:POKE36878,0:NEXT:NEXT
1040 DATA"FLAK","EIN NEUES SPIEL VON","ANDREAS DRIPKE
1050 PRINT"■"SPC(7)"_______":PRINTSPC(7)"■F L A K":PRINT"■VERSUCHEN SIE, DIE
1060 PRINT"VON LINKS KOMMENDEN":PRINT"RAKETEN ABZUSCHIES-
1070 PRINT"SEN. IHNEN STEHEN DA-":PRINT"ZU VIER FLAK-STATIO-
1080 PRINT"NEN ZUR VERFUEGUNG.":PRINT"■SIE BEWEGEN SICH VON
1090 PRINT"EINER BASIS ZUR ANDE-":PRINT"REN MIT DEN TASTEN":PRINT"'Z' UND 'X'.
1100 PRINT"■DER SCHUSSWINKEL WIRD":PRINT"MIT DEN TASTEN ',',
1110 PRINT"'.' UND '/' FESTGE-":PRINT"LEGT.":GOSUB2010
1120 PRINT"■VON EINER BASIS KANN":PRINT"IMMER NUR EIN GE-
1130 PRINT"SCHOSS ABGESCHOSSEN":PRINT"WERDEN, INSGESAMT AL-
1140 PRINT"SO VIER GESCHOSSE ZUR":PRINT"GLEICHEN ZEIT.
1150 PRINT"■SCHUSS MIT 'SPACE'.":PRINT"■STOP MIT '←'.
1160 GOSUB2010:PRINT"■     ICH LADE NUN":PRINTSPC(6)"■■F  L  A  K":POKE657,0
1170 POKE631,131:POKE198,1:END
```

```
2010 PRINTSPC(8)"■_____":PRINTSPC(8)"■RETURN":POKE198,0
2020 GETG$:IFG$<>CHR$(13)GOTO2020
2030 RETURN



1 REM ---- FLAK ----
2 REM
3 REM -(C) 1982 BY -
4 REM
5 REM ANDREAS DRIPKE
6 REM
7 REM   WIESBADEN
8 REM
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
110 IFPEEK(158)OR191ANDSTTHENPRINT"■?LOAD ERROR";:END
1000 S=36876:N=S+1:L=S+2:C=S+3:DIMP(3),R(3),Z(8),D(8):FORI=0TO8:READD(I),Z(I)
1010 NEXT:DATA21,78,22,93,23,77,-1,64,,87,1,64,-23,77,-22,93,-21,78:FORI=0TO3
1020 FORI=0TO3:R(I)=22:P(I)=0:NEXT:U=INT(10+10*RND(1)):R=U:P=0:H=0:G=0
1030 POKE657,128:PRINTCHR$(142)"■■■
```

```
2010 POKES,224:POKEL,7:POKEX,81:GETG$:IFG$=""GOTO2100
2020 IFG$="←"GOTO3000
2030 IFG$="Z"THENP=P-1:IFP<0THENP=0
2040 IFG$="X"THENP=P+1:IFP>3THENP=3
2050 GOSUB2220:IFG$=","THENR(P)=23
2060 IFG$="."THENR(P)=22
2070 IFG$="/"THENR(P)=21
2080 GOSUB2200:IFG$<>" "ORP(P)<>0GOTO2100
2090 G=G+1:P(P)=VB+440-R(P)-R(P)+5*P+2
2100 POKES,0:PRINT"■SCHUSS:"G"■ TREFFER:"H:FORI=0TO3:IFP(I)=0GOTO2130
2110 POKES,192:POKEP(I),32:P(I)=P(I)-R(I):IFP(I)<VB+22THENP(I)=0:GOTO2130
2120 POKEP(I),87:IFP(I)=XTHENGOSUB2160
2130 POKES,0:NEXT:POKEL,0:POKEX,32:X=X+1:IFX=YGOTO2000
2140 GOTO2010
2150 POKEL,15:FORJ=254TO128STEP-6:POKES,J:NEXT:POKES,0:POKEL,0:RETURN
2160 H=H+1:FORJ=0TO8:Z=X+D(J):IFZ>=VBTHENPOKEZ,Z(J)
2170 NEXT:POKES,0:POKEN,185:FORJ=15TO0STEP-.03:POKEL,J:NEXT:POKEN,0:POKEL,7
2180 FORJ=0TO8:Z=X+D(J):IFZ>=VBTHENPOKEZ,32
2190 NEXT:Y=X+1:P(I)=0:RETURN
2200 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■"SPC(P*5+1)MID$("\   |   /",(23-R(P))*3+1,3)
2210 RETURN
2220 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■":PRINT"                              ■"
2230 PRINTSPC(P*5+2)"↑■":RETURN
3000 POKEL,0:POKES,0:PRINTCHR$(14)"■SIE HABEN VON"R"RA-"
3010 PRINT"■KETEN"H"ABGESCHOSSEN":PRINT"■DAZU BENOETIGTEN SIE
3020 PRINT"■"G"GESCHOSSE.■":ONG+1GOTO3180:Q=INT(H/R*H/G*10)+1
3030 ONQGOSUB3080,3100,3120,3130,3150:IFQ>5THENGOSUB3160
3040 PRINT"■_____":PRINT"■ NOCHMAL? J/N ":PRINT"─────────────
```

```
3050 F=F+1AND1:PRINT"■"SPC(10)"    ■■■"SPC(F*2)"↑":FORI=0TO299:NEXT
3060 GETQ$:IFQ$="J"GOTO1020
3070 ON-(Q$<>"N")GOTO3050:PRINT"■FLAK - ENDE":POKE657,0:END
3080 PRINT"DAS IST EIN AUSSERST":PRINT"■SCHLECHTES ERGEBNIS.
3090 PRINT"■SIE SIND ENTLASSEN!":RETURN
3100 PRINT"DAS WAR JA NICHT GE-":PRINT"■RADE DAS GELBE VOM
3110 PRINT"■EI - SIE MUESSEN NOCH":PRINT"■VIEL UEBEN.":RETURN
3120 PRINT"ICH HABE SCHON":PRINT"■SCHLECHTERE GESEHEN.":RETURN
3130 PRINT"AUS IHNEN KOENNTE MAL":PRINT"■ETWAS WERDEN. HALTEN
3140 PRINT"■SIE SICH RAN.":RETURN
3150 PRINT"DAS WAR JA GAR NICHT":PRINT"■UEBEL. WEITER SO!":RETURN
3160 PRINT"ERSTKLASSIG. SOLCHE":PRINT"■LEUTE KANN MAN BRAU-":PRINT"■CHEN.
3170 RETURN
3180 PRINT"SIE HABEN NOCH NICHT":PRINT"■EINMAL GESCHOSSEN!!!
3190 PRINT"■PROBIEREN SIE'S DOCH":PRINT"■WENIGSTENS EINMAL.":GOTO3040
```

8. STARIS

Nach einem Actionspiel folgt nun ein Spiel, bei dem es ganz auf Geschicklichkeit und Geduld ankommt. STARIS ist ein Einpersonenspiel für verregnete Sommertage und lange Winterabende. Wir wollen uns jetzt aber von den spielerischen Elementen abwenden und uns den programmtechnischen interessanten Aspekten widmen.
Genau wie FLAK besteht auch STARIS aus zwei Teilen. Nehmen wir uns zunächst den ersten Programmteil – die Spielanleitung – vor. Das automatische Nachladen des eigentlichen Spiels ist in Zeile 1040 programmiert. Wie es funktioniert, haben wir im letzten Kapitel besprochen. Im Listing ist die Programmzeile für das Nachladen von Kassette ausgelegt, für Diskette muß folgende Änderung vorgenommen werden:

```
1035 PRINT"■■■    "CHR$(34)"STARIS"CHR$(34)",8■■■
```

Voraussetzung für das Funktionieren ist, daß der zweite Teil auch unter dem Namen "STARIS" auf Diskette abgespeichert wurde.
Um die Spielanleitung etwas attraktiver zu machen, haben wir ihr ein dynamisches Titelbild vorangestellt. Wenn Sie sich STARIS ANLEITUNG einmal ansehen, dann werden Sie feststellen, daß das Programm im Wesentlichen aus zwei Teilen besteht. Sie werden in den Zeilen 1010 und 1020 nacheinander mittels GOSUB aufgerufen. Der erste Teil "Graphik" ist für das sich ständig ändernde Titelbild verantwortlich. Wenn Sie sich die Zeilen 2000 bis 2160 anschauen, bekommen Sie einen Eindruck von der Komplexität, mit der die bewegliche Graphik programmiert ist. Der zweite Teil von 3000 bis 3170 enthält den Text, der STARIS erklärt.
Zu diesen zwei Hauptteilen hinzu kommt eine Routine, die auf das Drücken der RETURN-Taste wartet. Sie liegt bei 4000 bis 4030. Wenn Sie das Titelbild genügend gewürdigt haben, können Sie nämlich durch Drücken von RETURN die Spielanleitung erscheinen lassen.
Das Anleitungsprogramm zu STARIS zeigt Ihnen übrigens auch, daß in BASIC auch die Zeilennummer 0 zugelassen ist. Da sie normalerweise nicht benutzt wird, ist dies den wenigsten Programmierern geläufig. Die höchste Zeilennummer, die der Computer noch verarbeiten kann, ist übrigens 63999. Im allgemeinen sollte man allerdings solche Extrema vermeiden.
Wenden wir uns nun dem zweiten Teil, also dem eigentlichen

Spielprogramm STARIS zu. Die Struktur des Programmes ist in den Zeilen 1000 bis 1080 erkennbar. Die einzelnen Programmteile werden mit GOSUB aufgerufen. Wir wollen aber mit unserer Betrachtung schon vorher einsetzen, nämlich bei Zeile 110. Wie Sie wissen, erkennt das Programm an dieser Stelle eventuelle Ladefehler. Wenn Sie sich Zeile 110 einmal genauer ansehen, mag sie Ihnen aber vielleicht doch einiges Kopfzerbrechen bereiten.

```
110 IFPEEK (158)OR191ANDSTTHEN . . . . . .
```

Ist das korrektes BASIC? Normalerweise finden wir bei einer IF . . THEN-Zeile zwischen dem IF und dem THEN eine Bedingung. Diese Bedingung beinhaltet üblicherweise Gleichheits- und/oder Kleiner/Größer-Zeichen. Beispiele dazu:

```
10 IF X=1 THEN X=X+1
20 IF A<10 OR A>20 THEN B=A
30 IF Z<>X THEN GOTO 10
```

In unserer IF . . THEN-Zeile 110 in STARIS fehlt ein Vergleichszeichen völlig. Das ist im Commodore BASIC erlaubt und gleichbedeutend mit einer Bedingung "ungleich NULL". Am besten machen wir uns dies an einem einfacheren Beispiel als Zeile 110 klar.

```
IF X THEN PRINT "***"
```

Diese Programmzeile meint exakt dasselbe wie

```
IF X<>0 THEN PRINT "***"
```

Immer, wenn die Bedingung lautet, daß eine Variable (in unserem Beispiel X) nicht Null sein darf, damit der Befehl nach THEN ausgeführt wird, läßt sich die Operation '<>0' auch weglassen. Ein solcher Fall ist in Zeile 110 gegeben.
Bitte ziehen Sie aus dem Gesagten nicht den Trugschluß, daß die Bedingung "gleich NULL" mit Hilfe der NOT-Funktion ähnlich formuliert werden könnte. Es ist falsch zu schreiben

```
FALSCH!     IF NOTX THEN . . .
```

wenn dadurch

```
IF X=0 THEN . . .
```

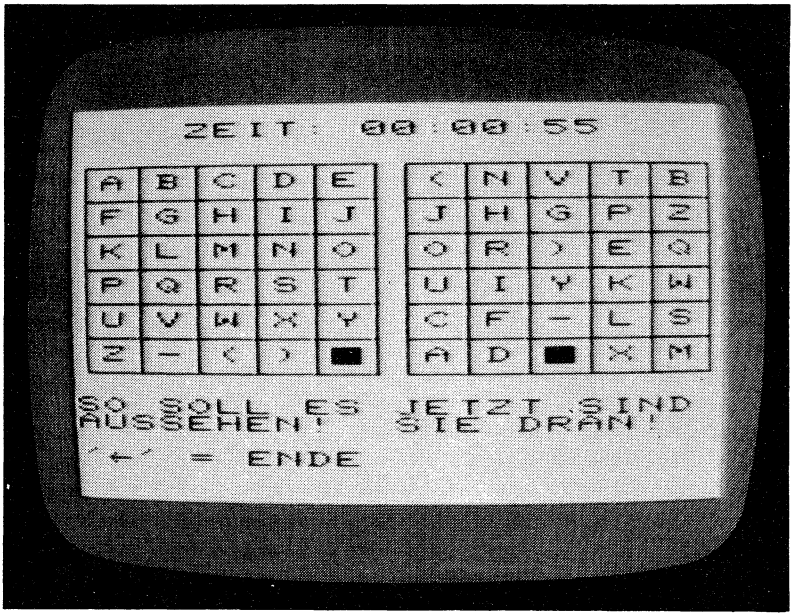
ausgedrückt werden soll. Bitte erinnern Sie sich an die numerische Entsprechung für NOT:

NOTX meint –(X+1)

Mithin ist die Abfrage 'IF NOTX THEN . . . ' also gleichbedeutend mit

IFX<>–1 THEN . . .

Es mag Ihnen merkwürdig erscheinen, daß der Befehl NOT ausgerechnet den Wert –1 in dieser Art abfragt. Der Grund dafür liegt in der bitweisen Verarbeitung von Vergleichen durch den Computer. Was das genau ist und wie es funktioniert, werden wir uns gleich beim nächsten Problem noch ausführlicher anschauen.



Sicher wissen Sie – und Programmzeile 110 zeigt es Ihnen auch nochmals –, daß wir eine IF . . THEN-Anweisung von mehreren Bedingungen gleichzeitig abhängig machen können. Wir benutzen dazu die Verknüpfungsbefehle AND (UND) und OR (ODER). Allgemein gilt:

IF (Bedingung 1) AND (Bedingung 2) AND . . . THEN . . .

für AND und analog für OR:

```
IF (Bedingung 1) OR (Bedingung 2) OR ... THEN ...
```

Die Programmzeilen 6010, 6020 und 6030 in STARIS geben Ihnen einen Eindruck von den unterschiedlichen Verknüpfungsmöglichkeiten.
Die Klammern und Leerzeichen können – wie auch in diesen Zeilen zu sehen ist – bei Eindeutigkeit entfallen. Im Zweifelsfall sollten Sie sie allerdings setzen. So verursacht die Zeile

```
IFX=TANDA=BTHEN ...
```

einen SYNTAX ERROR, weil der Computer aus 'X=T' und AND die BASIC-Funktion TAN (Tangens) herausliest und danach das Argument für diese Funktion (in Klammern) vermißt. In so einem Fall muß es richtig heißen:

```
IF(X=T)ANDA=BTHEN ...
```

Beachten Sie, daß die Zeile

```
IF A<>0 AND B<>0 THEN ...
```

nicht durch den Ausdruck

```
IF A AND B THEN ...
```

ersetzt werden kann. Auch bei Verwendung von OR muß jede einzelne Bedingung ein Gleichheits- und/oder Kleiner/Größer-Zeichen enthalten. Sehen Sie sich die genannten Zeilen in STARIS daraufhin noch einmal an.
Wenn Sie AND und OR gemischt verwenden, müssen Sie beachten, daß AND enger bindet als OR. Ähnlich wie beim täglichen Rechnen gilt "Punkt- vor Strichrechnung", so gilt hier übertragen "AND vor OR". Nehmen wir ein Beispiel zu Hilfe.

```
IF A=1 AND B=2 OR C=3 THEN ...
```

wirkt in derselben Weise, als wenn Klammern wie folgt gesetzt wären:

```
IF (A=1 AND B=2) OR C=3 THEN ...
```

Machen Sie sich klar, daß dies eine völlig andere Bedeutung ist als

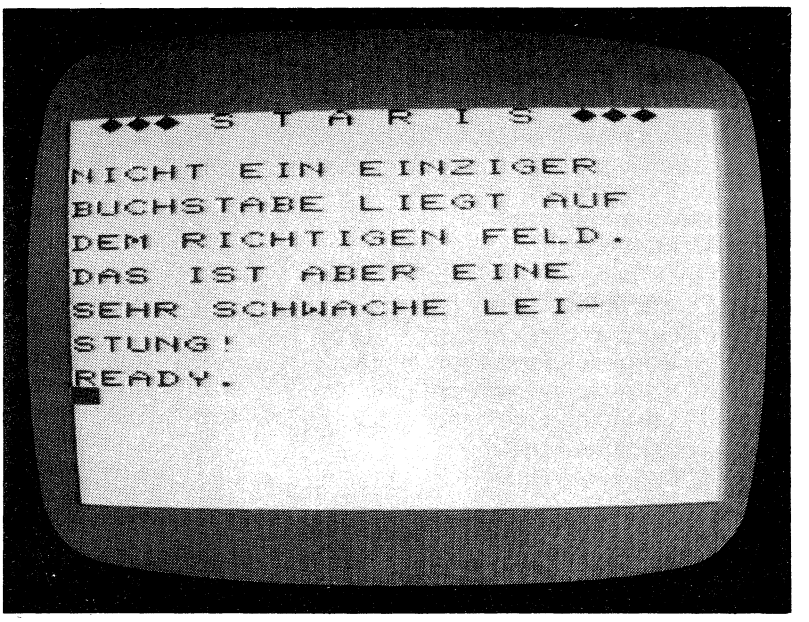
```
IF A=1 AND (B=2 OR C=3) THEN ...
```

Da die Strenge der Bindung – mathematisch ausgedrückt, die Priorität – bei Ausdrücken grundsätzlich sehr wichtig ist, habe ich Ihnen hier einmal eine Prioritätenliste erstellt, auch für die anderen Operationssymbole:

↑	Potenzieren	
Vorzeichenwechsel		
*	Multiplikation	gleichberechtigt
/	Division	
+	Addition	gleichberechtigt
–	Subtraktion	
	Vergleichsoperator "kleiner"	
	Vergleichsoperator "gleich"	gleichberechtigt
	Vergleichsoperator "größer"	
AND	logisches UND	
NOT	logisches NICHT	
OR	logisches ODER	

Sollten Sie also einmal diesbezüglich Schwierigkeiten haben, schlagen Sie einfach in dieser Liste nach oder – noch einfacher – setzen Sie Klammern, um die Eindeutigkeit herzustellen.
Wir wollen uns aber jetzt nochmals AND, OR und NOT widmen. Die Besonderheiten dieser Funktionen lassen sich nämlich durch die bitweise Verarbeitung durch den Computer erklären. Wie Sie vielleicht wissen, arbeitet jeder Computer intern mit sogenannten Bits. Ein Bit ist die Entscheidung zwischen 'Ja' und 'Nein' oder zwischen 'ein' und 'aus' oder ganz einfach zwischen '0' und '1'. Diese Null- und Eins-Darstellungsform wird in diesem Zusammenhang im allgemeinen verwendet. Der Commodore VC 20 verarbeitet intern jeweils acht Bits gleichzeitig. In seinem Innern steckt nämlich ein 8-Bit-Mikroprozessor. Ein solches "Paket" von acht Bits nennt man ein Byte (sprich "Beit"). Ein Beispiel für ein Byte ist also '00101110'.
Auch wenn der Computer die Programmiersprache BASIC versteht und auch sonst noch eine ganze Menge kann, so versteht er eben doch tief in seinem Innersten nur Bits und Bytes. Zahlen speichert er also genauso, und zwar braucht er zwei Byte, um eine ganze Zahl darin abzulegen. Diese zwei Bytes, die aus insgesamt 16 Bits bestehen, bearbeitet der Computer

nun mit den Befehlen AND, OR und NOT. Für die Fachleute sei gesagt, daß das die gleichlautenden Operationen der Boole'-schen Algebra sind.



Wir wollen uns die Verknüpfungsregeln der drei Operationen im einzelnen ansehen:

AND:	0 AND 0 = 0	OR:	0 OR 0 = 0
	0 AND 1 = 0		0 OR 1 = 1
	1 AND 0 = 0		1 OR 0 = 1
	1 AND 1 = 1		1 OR 1 = 1
NOT:	NOT 0 = –1		NOT –1 = 0

Eine Zahl (gleich 16 Bits) wird nun bitweise von rechts nach links mit einer anderen Zahl durch AND oder OR verknüpft. Ein Beispiel soll Ihnen das klarer machen:

1. Zahl 1000 (dezimal)	00000011 11101000	OR
2. Zahl 2097 (dezimal)	00001000 00110001	
	00001011 11111001	

Verknüpfen Sie die gleichen Zahlen mit AND, dann erhalten wir:

1. Zahl 1000 (dezimal)	00000011 11101000	AND
2. Zahl 2097 (dezimal)	00001000 00110001	
Ergebnis 32 (dezimal)	00000000 00100000	

Sie sehen, daß sich die Ergebnisse von OR und AND nicht so einfach berechnen lassen.
Machen Sie sich bitte klar: AND, OR und NOT können zum Verknüpfen von Bedingungen bei IF . . THEN-Anweisungen eingesetzt werden. Sie können aber außerdem die aufgezeigten Boole'schen Funktionen erfüllen. Diese Boole'sche Operation ist zum Beispiel in einem Ausdruck wie diesem hier gemeint:

X = A AND B

Die Werte A und B werden auf die gezeigte Weise miteinander verknüpft und das Ergebnis schließlich der Variablen X zugewiesen.
Wenn Sie diesen letzten Abschnitt nicht ganz verstanden haben, dann nehmen Sie das nicht allzu tragisch. Sie haben zumindest einmal einen Einblick in diese Dinge bekommen und wissen, worum es geht. Merken sollten Sie sich nur die praktischen Programmierregeln. Denn nur die und sonst nichts sind beim Programmieren wichtig. Schließlich wollen Sie ja kein Diplom-Informatiker werden, oder?
Wir wollen uns jetzt noch einer ganz anderen Sache zuwenden. Sie wissen, daß in der Variablen TI$ die Uhrzeit im Format 'HHMMSS' läuft. Wenn wir in einem Programm die Uhrzeit digital einblenden lassen wollen, dann sieht es meist besser aus, wenn Stunden, Minuten und Sekunden durch Doppelpunkte voneinander getrennt sind (wie bei "echten" Digitaluhren). Eine solche Trennung läßt sich recht einfach programmieren. Wie, das sehen Sie in STARIS in Zeile 4010. Beachten Sie, daß TI$ trotz seiner Sonderfunktion wie eine normale Stringvariable behandelt werden kann. Lediglich die Länge des Strings ist mit sechs fest vorgegeben.
Nun hoffe ich, daß Ihnen beim Eingeben des Programmes nicht allzu viele Tippfehler unterlaufen. Als Belohnung erhalten sie dafür ein Spiel, das garantiert jede Langeweile vertreibt. Ich wünsche Ihnen viel Freude beim Puzzlen.

Zusammenfassung:
Kapitel 8 verhalf Ihnen zu einem Spiel für einsame Stunden und (hoffentlich) zu Wissen über diese Themen:
- "Titelbild" zur Aufnahme in eigene Programme
- Formulierung von Bedingungen in IF . . THEN-Anweisungen
- Verknüpfung von Bedingungen und Zahlen mittels AND, OR und NOT
- Prioritäten beim Auswerten von Ausdrücken
- Aufbereitung von TI$ zum Format 'HH:MM:SS'

```
0 REM ***************
1 REM *             *
2 REM * - STARIS - *
3 REM *             *
4 REM * ANLEITUNG  *
5 REM *             *
6 REM * A. DRIPKE  *
7 REM *             *
8 REM ***************
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
110 IFPEEK(158)OR191ANDSTTHENPRINT"■?LADEFEHLER";:END
1000 REM ------------
1010 GOSUB2000:REM GRAPHIK
1020 GOSUB3000:REM ANLEITUNG
1030 REM ------------
1040 POKE198,1:POKE631,131:END
2000 POKE657,128:POKE36879,250:PRINTRND(-TI)"■■S   T   A   R   I   S":DIMR(6)
2010 FORI=VCTOVC+505:POKEI,2:NEXT:FORI=VB+21TOVB+505STEP22:POKEI,160:NEXT:A=10
2020 B=12:FORI=0TO6:R(I)=64-128*(RND(1)>.5)+64*RND(1):NEXT
2030 FORI=1TO9:X=A+I:Y=B:GOSUB2070:X=A-I:GOSUB2070:X=A:Y=B+I:GOSUB2070:Y=B-I
2040 GOSUB2070:X=A+I:Y=B+I:GOSUB2070:X=A-I:Y=B-I:GOSUB2070:Y=B+I:GOSUB2070
2050 X=A+I:Y=B-I:GOSUB2070:NEXT:FORI=1TO2E3:NEXT:GETG$:IFG$THENRETURN
2060 GOTO2020
2070 POKEVB+22*Y+X,R(0):IFI=1THENRETURN
2080 W=INT(I/2):FORJ=1TOW
2090 IFX=ATHENQ=Y:P=X+J:GOSUB2150:P=X-J:GOSUB2150:NEXT:RETURN
2100 IFY=BTHENP=X:Q=Y+J:GOSUB2150:Q=Y-J:GOSUB2150:NEXT:RETURN
```

```
2110 Q=Y:IFX<ATHENP=X+J:GOSUB2150:GOTO2130
2120 P=X-J:GOSUB2150
2130 P=X:IFY<BTHENQ=Y+J:GOSUB2150:NEXT:RETURN
2140 Q=Y-J:GOSUB2150:NEXT:RETURN
2150 POKE36878,I:POKE36875,R(J)OR128:POKEVB+22*Q+P,R(J)
2160 POKE36875,0:POKE36878,0:RETURN
3000 A$="──────────────────────────":PRINT"■■■"A$"** S  T  A  R  I  S **"A$
3010 PRINT"■AUF EINEM 5*6 SPIEL-":PRINT"■BRETT IST DAS ALPHA-
3020 PRINT"■BET WAHLLOS ANGEORD-":PRINT"■NET, EIN FELD BLEIBT":PRINT"■FREI.
3030 GOSUB4000:PRINT"■SIE SOLLEN VERSUCHEN,":PRINT"■DAS ALPHABET ZU ORD-
3040 PRINT"■NEN, INDEM SIE EINEN":PRINT"■BUCHSTABEN AUF DAS
3050 PRINT"■FREIE FELD SCHIEBEN;":PRINT"■DANN IST NATUERLICH
3060 PRINT"■DAS FELD LEER, AUF":PRINT"■DEM VORHER DER BUCH-":PRINT"■STABE LAG.
3070 GOSUB4000:PRINT"■ES SIND VERSCHIEBUN-":PRINT"■GEN IN HORIZONTALER,
3080 PRINT"■VERTIKALER UND WAHL-":PRINT"■WEISE AUCH IN DIAGO-
3090 PRINT"■NALER RICHTUNG ER-":PRINT"■LAUBT.":GOSUB4000:PRINT"■■DIE ZEICHEN";
3100 PRINT" WERDEN":PRINT"■MIT FOLGENDEN TASTEN":PRINT"■VERSCHOBEN
3110 PRINTSPC(4)"■■\  |  /":PRINTSPC(5)"Q W E":PRINTSPC(7)"┌──────── LEERES
3120 PRINTSPC(4)"─A S D─     FELD":PRINTSPC(5)"■Z X C":PRINTSPC(4)"/  |  \
3130 GOSUB4000:PRINT"■■ERLAUBTE RICHTUNGEN:":PRINT"■■1. HORIZONTAL UND
3140 PRINT"■■■■VERTIKAL UND":PRINT"■■■■DIAGONAL":PRINTSPC(8)"■■ODER■
3150 PRINT"■2. NUR HORIZONTAL":PRINT"■■■■UND VERTIKAL
3160 PRINT"■■1 IST LEICHTER ALS 2!":GOSUB4000:PRINT"■  ICH LADE JETZT DAS
3170 PRINTSPC(7)"■PROGRAMM":PRINTSPC(8)"■■STARIS":POKE657,0:RETURN
4000 PRINT"■■"SPC(8);:POKE198,0:IFPEEK(36869)AND2THENPRINT"RETURN":GOTO4020
4010 PRINT"RETURN
4020 GETG$:IFG$<>CHR$(13)GOTO4020
4030 RETURN
```

```
1 REM ***************
2 REM *** STARIS ***
3 REM ***************
4 REM
5 REM  (C) 1982 BY
6 REM
7 REM ANDREAS DRIPKE
8 REM
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
110 IFPEEK(158)OR191ANDSTTHENPRINT"■?LADEFEHLER";:END
1000 REM ------------
1010 GOSUB2000:REM VORBEREITUNG
1020 GOSUB3000:REM SPIELBRETT
1030 GOSUB4000:REM UHRZEIT
1040 GOSUB5000:REM GET
1050 IFX=3GOTO7000:REM ENDE?
1060 GOSUB6000:REM ZUG AUSFUEHREN
1070 GOTO1030
1080 REM ------------
2000 A=126:POKE36879,A:POKE657,128:PRINT"■■■S  T  A  R  I  S
2010 PRINT"■■1.■HORIZONTAL,":PRINT"■■■■VERTIKAL■UND":PRINT"■■■■DIAGONAL
2020 PRINTSPC(8)"■■ODER■":PRINT"■2.■NUR■HORIZONTAL":PRINT"■■■■UND■VERTIKAL
2030 PRINT"■■DAS■WAEHLEN■IE?":POKE198,0
2040 A=AAND-33ORNOTAAND32:POKE36879,A:T=TI
2050 ON-(T+30>TI)GOTO2050:GETG$:G=VAL(G$):ONGGOTO2060,2070:GOTO2040
2060 PRINT"■      FEIGLING!       ":GOTO2080
2070 PRINT"■      ANGEBER!        "
```

```
2080 POKE36879,126:FORI=1TO1E3:NEXT:DIMA$(29),B$(29):FORI=0TO29:READA$(I)
2090 X=30*RND(1):IFB$(X)<>""GOTO2090
2100 B$(X)=A$(I):NEXT:RETURN
2110 DATAA,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,-,(,),"■■"
3000 PRINT"■■";:P=0:GOSUB3100:GOSUB3050:PRINT"■■SO SOLL ES":PRINT"AUSSEHEN!
3010 FORI=1TO2E3:NEXT:P=11:GOSUB3100:GOSUB3050:PRINTSPC(11)"■■JETZT SIND
3020 PRINTSPC(11)"SIE DRAN!":PRINT"■'←' = ENDE":TI$="000000":FORI=0TO29
3030 IFB$(I)="■■"THENL=I
3040 NEXT:RETURN
3050 PRINT"■■■■";:IFPGOTO3080
3060 FORI=0TO5:PRINT:FORJ=0TO4:POKE36878,J+9:POKE36875,I*5+J+128
3070 PRINT"■"A$(I*5+J);:NEXT:PRINT:NEXT:POKE36875,0:POKE36878,0:RETURN
3080 FORI=0TO5:PRINT:PRINTSPC(11);:FORJ=0TO4:POKE36878,J+9:POKE36875,I*5+J+128
3090 PRINT"■"B$(I*5+J);:NEXT:PRINT:NEXT:POKE36875,0:POKE36878,0:RETURN
3100 PRINT"■■■■"SPC(P)"┌┬┬┬┬┬┐■■"CHR$(148)"─":FORI=0TO4:GOSUB3120:GOSUB3130
3110 NEXT:GOSUB3120:PRINTSPC(P)"└┴┴┴┴┴┘■■"CHR$(148)"─":RETURN
3120 PRINTSPC(P)"│ │ │ │ │■■"CHR$(148):RETURN
3130 PRINTSPC(P)"├┼┼┼┼┼┤■■"CHR$(148)"─":RETURN
4000 IFT$=TI$THENRETURN
4010 T$=TI$:PRINT"■■■■■■ZEIT: "LEFT$(T$,2)":"MID$(T$,3,2)":"RIGHT$(T$,2)
4020 POKE36878,5:POKE36876,200:FORI=0TO99:NEXT:POKE36876,0:POKE36878,0:RETURN
5000 X=2:GETG$:IFG$="←"THENX=3
5010 IFG$="W"THENX=-5
5020 IFG$="X"THENX=5
5030 IFG$="D"THENX=1
5040 IFG$="A"THENX=-1
5050 IFG=2THENRETURN
5060 IFG$="Q"THENX=-6
```

```
5070 IFG$="E"THENX=-4
5080 IFG$="Z"THENX=4
5090 IFG$="C"THENX=6
5100 RETURN
6000 IFX=2THENRETURN
6010 M=L+X:IFM<0ORM>29THENRETURN
6020 N=INT((L/5-INT(L/5))*10+.5):IFN=0AND(X=-1ORX=-6ORX=4)THENRETURN
6030 IFN=8AND(X=1ORX=-4ORX=6)THENRETURN
6040 A$=B$(L):B$(L)=B$(M):B$(M)=A$:L=M:GOTO3050
7000 POKE657,0:R=0:FORI=0TO29:IFA$(I)<>B$(I)THENR=I:I=30
7010 NEXT:S=0:FORI=0TO29:IFA$(I)=B$(I)THENS=S+1
7020 NEXT:PRINT"■ ◆◆◆ S T A R I S ◆◆◆":ONS+1GOTO7060:IFS<20THENC$=" NUR
7030 PRINT"■■SIE HABEN ES"C$:C$="FELDER":D$="DEN":IFS=1THENC$="FELD":D$="DEM
7040 PRINT"■GESCHAFFT,"S;C$:PRINT"■MIT "D$" RICHTIGEN
7050 PRINT"■BUCHSTABEN ZU BESET-":PRINT"■ZEN.":GOTO7090
7060 PRINT"■■NICHT EIN EINZIGER":PRINT"■BUCHSTABE LIEGT AUF
7070 PRINT"■DEM RICHTIGEN FELD.":PRINT"■DAS IST ABER EINE
7080 PRINT"■SEHR SCHWACHE LEI-":PRINT"■STUNG!":END
7090 IFR=0THENEND
7100 PRINT"■VON VORNE GEZAEHLT":PRINT"■(A, B, C ...)":IFR=1GOTO7120
7110 PRINT"■LIEGEN DAVON"R"RICH-":PRINT"■TIG HINTEREINANDER.":END
7120 PRINT"■LIEGT NUR DAS 'A'":PRINT"■RICHTIG.":END
```

9. LIFE

"Life" ist englisch und heißt übersetzt "Leben", gemeint ist das "Spiel des Lebens". Erfunden wurde Life von John Horton Conway. Es postulierte vier Regeln, nach denen sich eine Gruppe von Lebewesen von Generation zu Generation entwickeln sollte. Die genauen LIFE-Spielregeln finden Sie im Programmlisting selbst. Life ist ein Spiel, das für Computer geradezu ideal ist. Seit seinem Erscheinen ist es deswegen auf den verschiedensten Computern, vom programmierbaren Taschenrechner bis zur Großrechenanlage, programmiert worden. So wollte ich Ihnen Life auch für Ihren Computer nicht vorenthalten.



Als ich anfing, LIFE zu programmieren, stellte sich mir sofort ein Problem: die Zeit. Das Programm muß aus einer Generation von Lebewesen unter Berücksichtigung der Regeln die nachfolgende Generation innerhalb einer angemessenen Zeit entwickeln und auf dem Bildschirm darstellen. Dabei soll möglichst der gesamte Bildschirm genutzt werden, wie wir wissen, sind das 506 Bildschirmpunkte. Für diese Aufgabe benötigt BASIC ungefähr eine halbe Minute.

Weil aber eine Geschwindigkeit von zwei Generationen pro Minute viel zu langsam ist, um noch effektvoll zu sein, habe ich einen Teil von LIFE gar nicht in BASIC programmiert, sondern in einer anderen Computersprache, genannt 6502-ASSEMBLER. Moment mal, werden Sie jetzt sagen, der VC 20 versteht doch nur eine Sprache, und das ist BASIC. Was hat es also mit der 6502-ASSEMBLER-Sprache auf sich?
Wir haben bereits im letzten Kapitel gesehen, daß der Computer tief in seinem Innersten nur mit Bits operiert. Wir wissen, daß er dabei jeweils acht Bits zu einem Byte zusammenfaßt. BASIC-Befehle wandelt der Computer nun in Bytes um, damit er sie überhaupt verarbeiten kann. Wenn ein BASIC-Programm abläuft, dann übersetzt der Computer das Programm Schritt für Schritt in Bytes und führt es dann aus. Mit anderen Worten: BASIC wird auf eine niedere Stufe von Bytes zurückgeführt. Den Teil im Computer, der diese Aufgabe verrichtet, nennen wir "Interpreter". Er interpretiert quasi das BASIC-Programm.
Wenn der Computer BASIC nacheinander in Bytes umwandeln muß, um damit überhaupt etwas anfangen zu können, dann muß es doch eine Möglichkeit geben, ihn gleich mit Bytes zu "füttern". Es gibt sie! Wir nennen ein solches in einzelnen Bytes vorliegendes Programm Maschinencode.
Ein Byte besteht zwar aus Nullen und Einsen (Beispiel '00101110'), wir haben aber die Möglichkeit, es auch als ganz normale Zahl zu schreiben. Die Umrechnung in Nullen und Einsen lassen wir den Computer selbst vornehmen. Maschinencode ist also im allgemeinen nichts weiter als eine Zahlenfolge. In LIFE ist der Maschinencode in den BASIC-Zeilen 3000 bis 3080 zwischengespeichert. Statt Maschinencode können wir auch Maschinenprogramm sagen.
Die Zwischenspeicherung in DATA-Anweisungen ist die einfachste Art, ein BASIC- und ein Maschinenprogramm miteinander zu verbinden. Um mit dem Maschinencode nun etwas anfangen zu können, muß er in freie Speicherzellen des Computers gebracht werden.
Dies geschieht in LIFE in Zeile 2000. Der Maschinencode wird mit 'READ A' eingelesen und mit 'POKE I,A' in die entsprechenden Speicherzellen transportiert. Sicher erinnern Sie sich, daß der Speicherbereich von 829 bis 1019 nur im Zusammenhang mit einem angeschlossenen Kassettenrekorder benutzt wird. Dieser Bereich steht uns somit für das LIFE-Maschinenprogramm voll zur Verfügung, und wir nutzen ihn von 828 bis 989 aus (Zeile 2000). Nachdem unser Maschinencode in Speicherzellen steht, können wir ihn erst richtig

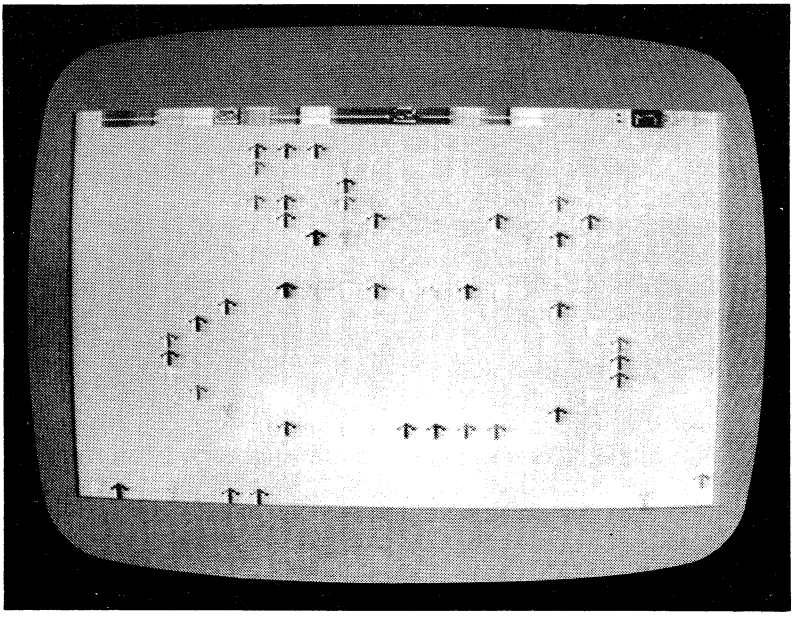
benutzen. Wir können nun das Maschinenprogramm starten. Was für BASIC der Befehl RUN bedeutet, das ist für Maschinencode 'SYS(Adresse)'. Zu SYS müssen wir also immer die Adresse mit angeben, wo unser Maschinenprogramm anfängt. Die Klammern kann man übrigens auch weglassen, statt 'SYS(828)' dürfen wir folglich auch 'SYS828' schreiben.
Zwischen RUN und SYS bestehen aber einige sehr wesentliche Unterschiede. Nach einem Start mittels SYS ist der BASIC-Interpreter abgeschaltet, weil er nicht mehr gebraucht wird (Es muß ja kein BASIC mehr in Bytes und damit in Maschinencode umgewandelt werden!). Gleichzeitig tritt die STOP-Taste außer Funktion. Wenn der Computer ein Maschinenprogramm abarbeitet, ist er also nicht mehr mit STOP zu unterbrechen.
Manchmal hilft noch STOP und RESTORE gleichzeitig, aber auch diese Methode führt nicht immer zum gewünschten Erfolg, dem Abbrechen des Maschinenprogrammes. Bei einem Fehler im Maschinencode hilft oft nur noch Aus- und Wiedereinschalten, um den Computer wieder in Ordnung zu bringen. Wählen Sie deshalb für SYS niemals fremde Adressen, sonst laufen Sie Gefahr, daß Ihnen das gerade geladene Programm verloren geht.
Aus diesem Grund sollten Sie LIFE gleich nach dem Eintippen erst einmal auf Kassette oder Diskette abspeichern. Sonst kann es passieren, daß Sie nach einem Tippfehler in den Maschinencode-DATAs das ganze Programm gleich noch einmal eingeben dürfen, weil es nach RUN abgestürzt ist. Und das wäre doch ärgerlich.
Wir wissen nun, wie ein Maschinenprogramm aussieht, wie wir es am besten mit einem BASIC-Programm verbinden (DATA), wo und wie wir es unterbringen können (Adressen 828 bis 1919) und wie wir es starten (SYS). Nur eines wissen wir noch nicht, nämlich wie wir in Maschinencode programmieren können. Hier kommen wir nun auf den 6502-ASSEMBLER zurück.
Direkt in Maschinencode zu programmieren, ist praktisch unmöglich. Schließlich besteht Maschinencode nur aus Zahlen und wie soll man damit programmieren. Deshalb hat man die Programmiersprache 6502-ASSEMBLER erfunden. Der eigentliche Programmiervorgang findet in dieser Sprache statt, und das fertige Programm wird dann in Maschinencode übersetzt. Das Übersetzen kann man von Hand tun oder aber von einem sogenannten Assemblerprogramm vornehmen lassen. Den Vorgang des Übersetzens nennen wir auch Assemblieren.
6502-ASSEMBLER ist eine Programmiersprache, die stark

den heutigen weit verbreiteten Taschenrechnersprachen gleicht. Der Namenszusatz "6502" rührt daher, daß diese Sprache nur für den Mikroprozessor 6502, der im VC 20 eingebaut ist, geeignet ist. Für andere Mikroprozessoren gibt es ähnliche Sprachen. Leider können wir uns in diesem Buch nicht ausführlicher mit 6502-ASSEMBLER befassen. Im INTERFACE AGE Verlag (COMPUTER LIFE) gibt es ein ASSEMBLER-Entwicklungs- und Lehrsystem, das Ihnen sicherlich nützlich sein wird, wenn Sie 6502-ASSEMBLER auf Ihrem VC 20 lernen möchten. Wenn ich Ihnen dieses System so empfehle, dann deswegen, weil es von mir entwickelt und das Lehrbuch von mir verfaßt wurde.
Damit Sie wenigstens einen Eindruck von ASSEMBLER erhalten, stelle ich Ihnen ein ganz kurzes Programm in dieser Sprache vor. Es handelt sich um eine Verzögerungsschleife, die der BASIC-Befehlsfolge 'FORX=1TO9:NEXTX' entspricht.

```
        LDX #9
LOOP    DEX
        BNE LOOP
        RTS
```

Eine genaue Anleitung zum Programmieren in 6502-ASSEMBLER wäre mindestens so dick wie das hier vor Ihnen liegende Buch. Wir müssen also beim besten Willen an dieser Stelle darauf verzichten.
Wir wollen uns jetzt noch einmal die Zusammenhänge zwischen BASIC, ASSEMBLER und Maschinencode klarmachen. Ganz so einfach ist die Sache schließlich nicht. Also, wenn der Computer ein BASIC-Programm ausführt, dann wandelt er dieses nacheinander mit Hilfe des Interpreters in Bytes um, die er dann verarbeiten kann. Diese Folge von Bytes nennen wir Maschinencode oder Maschinenprogramm. Weil die Interpretation des BASIC-Programmes nicht unerheblich viel Zeit beansprucht, ist es manchmal besser, den Maschinencode direkt einzugeben. Da aber das Programmieren in Maschinencode praktisch unmöglich ist, benutzt man meist 6502-ASSEMBLER. Mit Hilfe eines Assemblerprogrammes kann man sich dann das in 6502-ASSEMBLER-Sprache geschriebene Programm automatisch in Maschinencode übersetzen lassen. Bei dieser Übersetzung (Assemblierung) wird mit einem Mal das gesamte 6502-ASSEMBLER-Programm in Maschinencode umgewandelt, während der BASIC-Interpreter Befehl für Befehl arbeitet.
Das Arbeiten in 6502-ASSEMBLER ist deswegen so vorteilhaft, weil der Computer den daraus erzeugten Maschinencode

um ein Vielfaches schneller verarbeiten kann als BASIC. Und genau dies ist für LIFE wichtig. Die beiden SYS-Einsprünge liegen in den Zeilen 2010 und 2030.
Jetzt haben Sie zwar noch nicht gelernt, in Maschinencode bzw. 6502-ASSEMBLER zu programmieren, aber ich hoffe, daß Sie doch einen Einblick in diese Dinge gewinnen konnten. Wie wirkungsvoll Maschinencode ist, können Sie an LIFE selbst sehen. Mit jedem Durchgang wird LIFE etwas schneller, bis hin zur durch die Arbeitsgeschwindigkeit des Mikroprozessors gesetzte Grenze. Es liegt bei Ihnen, ob Sie sich näher mit ASSEMBLER befassen möchten oder es bei dem Streifzug bleiben lassen.



In LIFE finden Sie einen POKE-Befehl, den Sie mit dem bisher in diesem Buch erarbeiteten Wissen noch nicht erklären können, ich meine

POKE 56,X

Sehen Sie sich die betreffenden Zeilen in LIFE daraufhin an, es sind 2010 und 2030.
Wie Sie sicher wissen, können wir die freie Speicherkapazität des Computers jederzeit mit

```
? FRE(0)
```

abrufen. Anstelle der Null kann auch jede andere Zahl stehen, sie ist ohne Einfluß auf das Ergebnis. Probieren Sie es aus! Wir wollen nun ein Experiment mit Speicherzelle 56 machen. Geben sie einmal

```
POKE 56,17
```

ein. Noch passiert nichts. Wenn wir aber jetzt nochmals den freien Speicher mit '?FRE(0)' überprüfen, dann scheint der VC 20 ganz schön geschrumpft zu sein. Wir haben mit dem POKE-Befehl also dem Computer "vorgegaukelt", daß er nur noch sehr wenig Kapazität zur Verfügung hätte. Dies ist möglich, da in Speicherzelle 56 in 256er Einheiten die obere Grenze des Gesamtspeichers festgelegt ist. Diese Grenze wird in Zeile 2010 um 2*256, also 512 Bytes, herabgesetzt. Damit steht BASIC dieser Bereich nicht mehr zur Verfügung. LIFE benötigt nämlich soviel Platz zur Zwischenspeicherung des Bildschirminhaltes, der ja 506 Positionen hat.
Die obere Speichergrenze läßt sich aber noch genauer als in 256er Einheiten festlegen, und zwar unter Benutzung von Speicherzelle 55. Die obere Kapazitätsgrenze ist nämlich exakt festgelegt durch

```
? PEEK(55)+256*PEEK(56)
```

Probieren Sie es aus! Wenn Sie nun den Inhalt von Speicherplatz 55 um z.B. 10 verändern, so verändert sich entsprechend ebenfalls die Speicherkapazität, die bei FRE angezeigt wird. Beachten Sie, daß sich auf diese Art und Weise die für BASIC zur Verfügung stehende Speicherkapazität zwar verringern läßt – um z.B. die restlichen Speicherzellen für ein Maschinenprogramm zu verwenden –, aber erhöhen läßt sich die Gesamtkapazität natürlich leider nicht. Es wäre schön, wenn nur ein POKE-Befehl nötig wäre, um mehr Speicherplatz zur Verfügung zu haben. Hier hilft aber nur ein Erweiterungsmodul.
Für diejenigen unter Ihnen, die schon etwas tiefer in die Materie eingedrungen sind, hier noch ein Tip. Wenn der VC 20 auf eine Speicherkapazität über 8 KByte ausgebaut ist, so wird der Adressbereich von 1024 bis 4095 nicht von BASIC belegt und steht somit ohne Einschränkung zum Beispiel für Maschinenprogramme zur Verfügung. Verglichen mit dem Kassettenpuffer mit 192 Zeichen sind dies natürlich beträchtlich mehr Bytes; allerdings – wie gesagt – nur bei einer Gesamtkapazität von mehr als 8 KByte. Ein längeres Maschinenprogramm ließe

sich in diesem Fall auch teilen. Ein Teil stünde im Bereich von 828 bis 1019 (Kassettenpuffer), der andere von 1024 bis 4095. Das sind immerhin bis zu 3264 Bytes Maschinencode. Darin läßt sich durchaus ein ASSEMBLER-Programm mittlerer Komplexität unterbringen.
Nun will ich Sie nicht länger vom Spielen abhalten. Ich wünsche Ihnen mit LIFE gute Unterhaltung. Probieren Sie doch einmal die verschiedensten Anfangskonfigurationen aus und testen Sie, welche eingehen, welche sich immer weiter entwickeln und welche stabil werden.

Zusammenfassung:
Außer LIFE, einem der bekanntesten Computerspiele überhaupt, war der Inhalt des 9. Kapitels:
- Einführung in Maschinencode, 6502-ASSEMBLER und die entsprechenden Zusammenhänge
- Speicherplatzanzeiger in den Zellen 55/56
- Ausnutzung des Bereiches 1024 bis 4095 für Maschinenprogramme

```
1 REM ↑↑↑↑↑↑↑↑
2 REM ↑       ↑
3 REM ↑ LIFE  ↑
4 REM ↑       ↑
5 REM ↑↑↑↑↑↑↑↑
6 REM
7 REM (C) 1982 BY
8 REM ANDREAS DRIPKE
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 POKE657,128:GOSUB4000:PRINT"■■■***  L I F E  ***":PRINTSPC(7)"————————
1010 PRINT"■■'LIFE' SIMULIERT":PRINT"■■DAS FASZINIERENDE■
1020 PRINT"■■SPIEL DES LEBENS.":PRINT"■■DIE LEBEWESEN SEHEN":PRINT"■■SO AUS:↑
1030 PRINT"■■DIESE LEBEWESEN":PRINT"■■ENTWICKELN SICH
1040 PRINT"■■NACH BESTIMMTEN":PRINT"■■REGELN:":GOSUB5000:GOSUB4000
1050 PRINT"■■SIE HABEN DEN":PRINT"■■GLEICHEN WERDEGANG":PRINT"■■WIE ALLES ";
1060 PRINT"LEBENDE:":PRINT"■■SIE WERDEN GEBOREN,":PRINT"■■LEBEN UND STERBEN.
1070 GOSUB5000:GOSUB4000:PRINT"■■DIE LEBEWESEN ↑":PRINT"■■ENTWICKELN SICH
1080 PRINT"■■NACH FOLGENDEN":PRINT"■■REGELN:":PRINT"■■1) WENN EIN FELD
1090 PRINTTAB(5)"VON DREI ↑":PRINTTAB(5)"UMGEBEN IST,
1100 PRINTTAB(5)"WIRD EIN NEUES":PRINTTAB(5)"↑ GEBOREN.
1110 PRINT"■■2) EIN ↑, DAS VON":PRINTTAB(5)"VIER ↑ DIREKT
1120 PRINTTAB(5)"UMGEBEN IST,":PRINTTAB(5)"STIRBT (UEBER-
1130 PRINTTAB(5)"BEVOELKERUNG).":GOSUB5000:GOSUB4000
1140 PRINT"■■3) EIN ↑, DAS MIT":PRINTTAB(5)"WENIGER ALS
1150 PRINTTAB(5)"ZWEI ANDEREN ↑":PRINTTAB(5)"KONTAKT HAT,
1160 PRINTTAB(5)"STIRBT AN":PRINTTAB(5)"VEREINSAMUNG.":PRINT"■■4) IN ALLEN
1170 PRINTTAB(5)"ANDEREN FAELLEN":PRINTTAB(5)"LEBT DAS ↑
```

```
1180 PRINTTAB(5)"LUSTIG VOR SICH":PRINTTAB(5)"HIN.":GOSUB5000:GOSUB4000
1190 PRINT"■■GEBEN SIE NUN EIN,":PRINT"■■WIE LANGE EINE
1200 PRINT"■■GENERATION VON ↑":PRINT"■■UEBERDAUERN SOLL.":PRINT"■■ZEITFAKTOR
1210 PRINT"■■(SEC., Z.B. 1):":POKE198,0
1220 PRINTTAB(18)"□"MID$("▒ ",D+2,1):D=NOTD:I=0
1230 I=I+1:GETY$:IFY$>"0"ANDY$<":"THENZF=VAL(Y$):PRINTTAB(17)"□"ZF:GOTO1260
1240 IFI<20GOTO1230
1250 GOTO1220
1260 PRINT"■■WENN SIE GLEICH":PRINT"■■'SPACE' DRUECKEN,
1270 PRINT"■■WIRD DER BILD-":PRINT"■■SCHIRM GELOESCHT.":PRINT"■■GEBEN SIE DANN
1280 PRINT"■■EINE ANFANGS-":PRINT"■■KONFIGURATION":PRINT"■■DER ↑ EIN.
1290 GOSUB5000:GOSUB4000:PRINT"■■BENUTZEN SIE DAZU
1300 PRINT"■■DIE CURSORSTEUER-":PRINT"■■TASTEN SOWIE '↑'.
1310 PRINT"■■BEENDEN SIE DIE":PRINT"■■EINGABE MIT":PRINT"■■'RETURN'.
1320 PRINT"■■ABBRUCH MIT '←'.":PRINT"■■WAEHLEN SIE DOCH
1330 PRINT"■■FUER DEN ANFANG":PRINT"■■DIESES MUSTER.
1340 PRINT"■■(BILDSCHIRMMITTE)":PRINT,"■↑↑↑":PRINT,"↑ ↑":PRINT,"↑ ↑":GOSUB5000
2000 PRINT"□";:FORI=828TO989:READA:POKEI,A:NEXT:POKE841,VB/256:POKE908,VB/256
2010 POKE56,PEEK(56)-2:FORI=VCTOVC+505:POKEI,8*RND(1):NEXT:SYS939
2020 A=PEEK(55)+256*PEEK(56):FORI=ATOA+22:POKEI,32:NEXT
2030 SYS828:GETA$:T=TI:IFA$="←"THENPOKE56,PEEK(56)+2:POKE657,0:CLR:END
2040 POKE36878,10:I=I+1AND3:POKE36876,217-I*2:ZF=ZF*.99
2050 IFTI<T+ZF*60GOTO2050
2060 POKE36876,0:POKE36878,0:GOTO2030
3000 DATA120,169,2,141,251,3,169,,133,57,133,59,169,,133,58,165,56,133,60,169,
3010 DATA162,8,188,162,3,24,113,57,202,208,247,201,250,240,10,201,252,208,9
3020 DATA160,23,177,57,208,7,169,30,44,169,32,160,23,145,59,230,57,230,59,208
3030 DATA214,230,58,230,60,206,251,3,208,205,162,2,160,,132,57,132,59,169,30
```

```
3040 DATA133,58,165,56,133,60,177,59,145,57,200,208,249,230,58,230,60,202,208
3050 DATA242,88,96,,1,2,22,24,44,45,46,32,202,3,32,228,255,162,11,221,210,3
3060 DATA240,5,202,208,248,240,241,72,32,202,3,104,32,210,255,201,13,208,226
3070 DATA96,164,211,177,209,73,128,145,209,96,19,141,17,145,29,157,20,148,32
3080 DATA94,13
4000 POKE36879,172:PRINT"■"CHR$(14):FORA=0TO21:POKEVB+A,158:POKEVC+A,4
4010 POKEVB+484+A,158:POKEVC+484+A,4:NEXT:FORA=22TO462STEP22:POKEVB+A,158
4020 POKEVC+A,4:POKEVB+21+A,158:POKEVC+21+A,4:NEXT:RETURN
5000 POKE198,0:PRINTTAB(15)"■_____"SPC(17)"■SPACE
5010 GETA$:IFA$<>" "GOTO5010
5020 RETURN
```

II. Teil

Die nun folgenden Spielprogramme benötigen mindestens 8 KByte Gesamtspeicherkapazität, also einen Speicherzusatz von mindestens 3 KByte. Falls Sie damit noch nicht ausgestattet sind, dann ist dies jetzt sicherlich der richtige Zeitpunkt, zu einer solchen Erweiterung zu greifen – nicht nur, um dieses Buch erfolgreich weiterverfolgen zu können, sondern weil die Grundkapazität des VC 20 von 5 KByte auf Dauer sowieso nicht ausreichend ist. Schließlich möchten Sie bestimmt nicht immer nur Miniprogramme laufen lassen, oder?
Wir werden uns jetzt auf jeden Fall größere und somit auch in ihrem Aufbau komplexere Programme ansehen. Auch dabei werden wir wieder eine ganze Reihe von Dingen über unseren Computer lernen, und natürlich unsere Spielesammlung mit weiteren interessanten Programmen ausstatten.

10. WELTRAUMSCHLACHT

Zerstören Sie feindliche Schlachtschiffe und Weltraumbasen, aber passen Sie auf, daß Ihr eigenes Raumschiff nicht abgeschossen wird! WELTRAUMSCHLACHT ist ein Graphikspiel, bei dem es auf Konzentration und schnelle Reaktion ankommt. Auch WELTRAUMSCHLACHT besteht aus zwei Einzelprogrammen – dem Anleitungsteil und dem eigentlichen Spielteil. Wie üblich lädt die Anleitung den zweiten Teil automatisch nach – vorausgesetzt, daß beide in der richtigen Reihenfolge hintereinander auf Kassette stehen. Soll das Nachladen von Diskette erfolgen, ändern Sie das Anleitungsprogramm einfach wie folgt:

```
2360 PRINT"■■DISKETTE BEFINDET.":LOAD"WELTRAUMSCHLACHT",8
```

Da das Nachladen von Kassette bei WELTRAUMSCHLACHT anders programmiert ist, als wir es bisher kennengelernt haben, wollen wir es uns genauer ansehen. Im Anleitungsprogramm von WELTRAUMSCHLACHT finden wir einen ganz normalen LOAD-Befehl, wie Sie ihn üblicherweise von Hand eingeben (Zeile 2360 des Anleitungsprogrammes). So geht es also auch. Dennoch gilt es hier, ein paar Besonderheiten zu beachten.
LOAD im Programm beinhaltet ein folgendes RUN. Anders gesagt, ein Programm, das von einem anderen mit LOAD eingeladen wird, beginnt nach dem Einladen sofort zu starten. Praktisch, nicht wahr? Auf diese Weise sparen wir uns den Trick mit dem über Tastaturpuffer simulierten Drücken von SHIFT und RUN/STOP. Im Unterschied zum LOAD von Hand (genauer gesagt, im Direktmodus) erscheinen beim programmierten LOAD keine Statusmeldungen auf dem Bildschirm wie FOUND"NAME" oder LOADING oder ähnliches. Nur wenn die PLAY-Taste am Kassettenrekorder nicht gedrückt ist, dann meldet sich der Computer in jedem Fall mit PRESS PLAY ON TAPE und quittierendem OK. Schließlich kann er andernfalls gar kein Programm einladen.
Ein weiterer wesentlicher Unterschied zwischen LOAD im Direktmodus und im Programm besteht in der Variablenbehandlung. Wenn wir LOAD eintippen und danach mittels RUN starten (oder gleich SHIFT und RUN/STOP auf der Tastatur drücken), dann werden dabei sämtliche Variablen Null gesetzt (numerische Variablen) bzw. gelöscht (Stringvariablen). Anders ist es beim programmierten LOAD. Hier gilt: Wenn das

einzuladende Programm kleiner ist als das ladende Programm, dann bleiben alle Variablen erhalten. Wenn also in Programm P1 der Befehl 'LOAD"P2" 'steht, dann darf P2 nicht größer sein als P1, falls Variablen von P1 nach P2 gerettet werden sollen. Ist P2 doch größer, so werden auch bei LOAD im Programm alle Variablen gelöscht, außerdem ist in diesem Fall in P2 noch eine zusätzliche Befehlsfolge nötig. Sie sehen dies im WELTRAUMSCHLACHT-Hauptprogramm in den Zeilen 100 und 110. Zeile 100 prüft, ob das Programm mit RUN gestartet wurde – in diesem Fall werden ja sowieso alle Variablen gelöscht und Zeile 110 wird übersprungen. Wurde das Programm dagegen durch programmiertes LOAD geladen, so können noch Variablenreste vom Vorprogramm im Speicher stehen. Da das zweite Programm – der Hauptteil – aber größer als das Anleitungsprogramm ist, können diese Variablen nicht genutzt werden. Sie werden deswegen in Zeile 110 gelöscht. Die Speicherzellen 45 und 46 enthalten die Adresse, wo der BASIC-Bereich oben aufhört und der Bereich zum Abspeichern der Variablen beginnt. Dies ist so ähnlich wie – Sie haben es im letzten Kapitel gelesen – die Speicherzellen 55 und 56 einen Verweis auf das Ende der Gesamtspeicherkapazität enthalten.

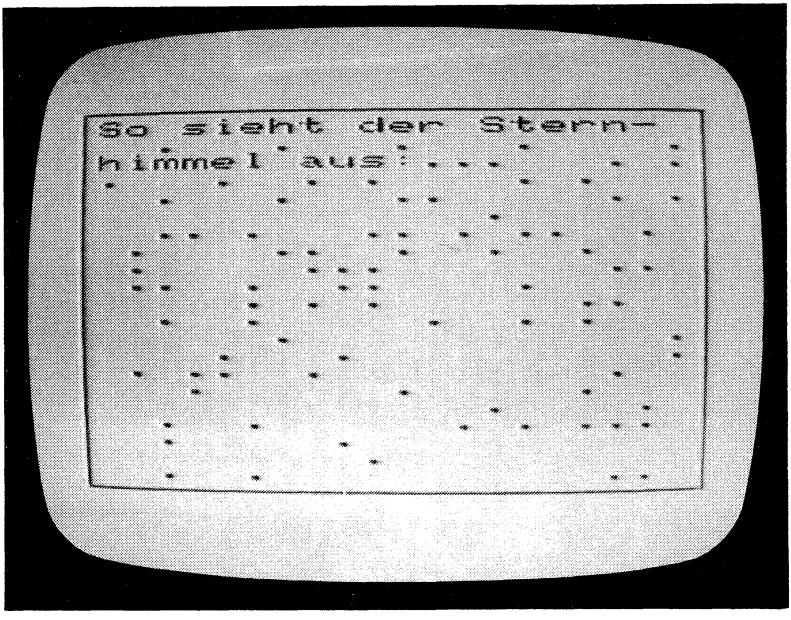
```
Ende BASIC/Anfang Variablen = PEEK(45)+256*PEEK(46)
```

Die Speicherplätze 47 und 48 bestimmen die Endadresse des Bereiches der Variablen.

```
Ende Variablen = PEEK(47)+256*PEEK(48)
```

Diese Angaben sind aber eigentlich gar nicht so wichtig. Entscheidend ist nur, daß Sie in Ihren eigenen Programmen, bei denen sie LOAD programmieren, den zweiten Programmteil mit zwei Zeilen 100 und 110 gemäß unserem Beispiel ausstatten. Um es noch einmal klar zu machen: Dies gilt nur, wenn das zweite Programm größer als das erste ist, so daß man in den meisten Fällen versuchen wird, dies zu vermeiden.
Gelingt dies – d.h. kann man es vermeiden –, so ist es relativ einfach, mehrere Programme zu einem durchlaufenden Gesamtprogramm zu verbinden. Die einzelnen Programme rufen das jeweils nächste einfach mit LOAD oder LOAD"NAME" auf. Bedenken Sie dabei, je größer das erste Einzelprogramm ist, desto größer dürfen auch die folgenden sein, ohne daß Variablen verloren gehen. Notfalls können Sie da auch mit ansonsten unnötigen REM-Zeilen nachhelfen.
Obwohl das bisher über programmiertes LOAD Gesagte für

den Rekorder gedacht war, gilt es genauso auch für Floppydisk. Allerdings muß hier statt LOAD natürlich 'LOAD"NAME",8' geschrieben werden. Der folgende Abschnitt dagegen ist ausschließlich für den Kassettenrekorder gedacht, wenn Sie eine Floppystation benutzen, können Sie ihn getrost überschlagen.



Damit das automatische Nachladen von Kassette funktionieren kann, ist es erforderlich, daß Sie die PLAY-Taste nach dem Einladen des Anleitungsprogrammes gedrückt lassen. Das WELTRAUMSCHLACHT-Anleitungsprogramm prüft dies und gibt entsprechend entweder die Meldung "Bitte lassen Sie die PLAY-Taste gedrückt" oder aber "Bitte drücken Sie die Play-Taste". Wie erkennt das Programm, ob die Taste gedrückt ist oder nicht? Nun, die Abfrage liegt in Zeile 1000 mit

```
PEEK(37151)AND64
```

Ist das Ergebnis dieses Ausdrucks 0, so ist die PLAY-Taste niedergedrückt. Andernfalls ist die Taste nicht gedrückt. Wenn Sie sich hinsichtlich der Auswertung solcher Ausdrücke nicht mehr ganz sicher sind, so schlagen Sie ruhig in Kapitel 8 nach. 'AND64' ist erforderlich, da nur Bit 6 der Speicherzelle 37151 für die PLAY-Taste entscheidend ist (2 hoch 6 ist 64).

Wenn hier immer von der PLAY-Taste die Rede ist, so ist das eigentlich nicht völlig korrekt. Statt PLAY kann nämlich auch REW oder F.FWD betätigt werden, ohne daß das Programm dies merken würde. Eine Unterscheidung zwischen diesen einzelnen Tasten ist nicht möglich.
Kommen wir nun zu der Anweisung

```
WAIT 37151,64,64
```

in Programmzeile 1020. An dieser Stelle wartet das Programm so lange, bis PLAY oder eine der anderen aufgeführten Tasten betätigt wird. WAIT ist ein äußerst selten verwendeter Befehl, der normalerweise nur im Zusammenhang mit Peripheriegeräten – bei uns der Rekorder – gebraucht wird. WAIT kann nicht durch Drücken der STOP-Taste am Computer unterbrochen werden, es ist vielmehr ein Rücksetzen ("reset") mit STOP und RESTORE gleichzeitig nötig.
Nun würde die Kassette nach Drücken von PLAY durchlaufen, was aber in unserem Fall nicht erwünscht ist. Schließlich soll der Spieler ja Gelegenheit bekommen, zunächst in aller Ruhe die Spielanleitung zu lesen und erst anschließend das Hauptprogramm einladen. Aus diesem Grunde muß das Programm nach Drücken von PLAY den Rekorder sofort abstellen. Dies geschieht in Zeile 1030 mittels POKE.

```
POKE 192,1
```

übergibt die Steuerung an den Computer, so daß dieser jetzt den Rekorder kontrollieren kann. Dann wird der Rekordermotor mit

```
POKE 37148 ,254
```

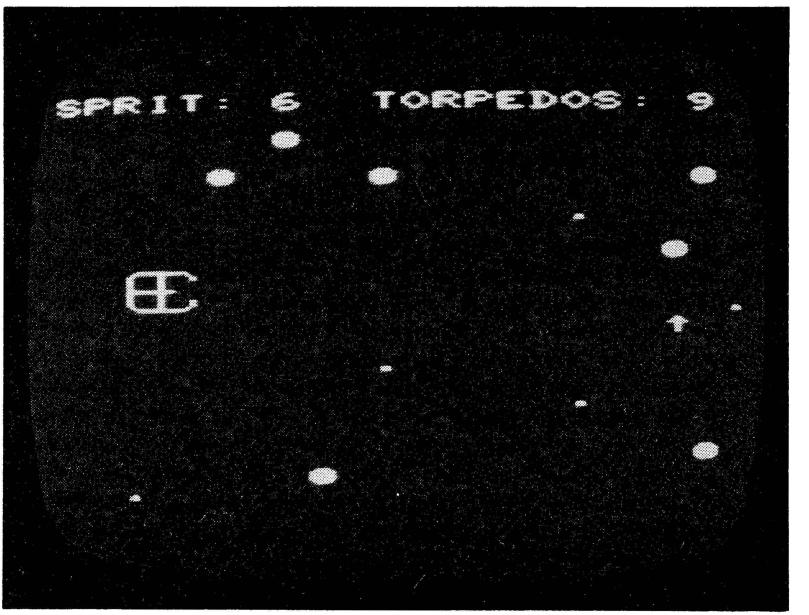
angehalten. Damit ist der Kassettenrekorder vorbereitet, um beim LOAD in Zeile 2360 sofort loslaufen zu können.
Einen Gag haben sowohl das Anleitungsprogramm als auch WELTRAUMSCHLACHT selbst. In den Zeilen 1 bis 9, die mit Kommentaren versehen sind, fehlt jegliche REM-Anweisung. Der Befehl 'GOTO100' in Zeile 0 bewirkt, daß die Programmzeilen 1 bis 9 niemals abgearbeitet werden – was ja auch gar nicht möglich wäre bzw. sofort zu einem SYNTAX ERROR führte.
Daß dies überhaupt möglich ist, liegt daran, daß der VC 20 bei der Eingabe einer Programmzeile diese ohne weitere Prüfung in den Speicher übernimmt. Solange diese Zeile nicht abgearbeitet werden soll, mag in ihr drinstehen, was will – der VC 20

jedenfalls kümmert sich nicht darum. Andere Computer prüfen neue Programmzeilen sofort bei der Eingabe, ob sie korrekt sind. Dies hat den Vorteil, daß man eine fehlerhafte Programmzeile schnell erkennt und korrigieren kann. Dafür sind bei einer solchen Konzeption aber eine ganze Reihe von doch recht interessanten Programmiertricks nicht möglich, wie zum Beispiel das Einstreuen von Kommentaren ohne REM. Immerhin spart das Weglassen von REM jedesmal genau ein Byte.
Moment mal, werden Sie jetzt sagen, wieso genau ein Byte und nicht drei. Schließlich besteht das Wort REM aus drei Buchstaben. Nun, die Erklärung dafür ist recht einfach. REM ist ein sogenanntes Schlüsselwort (engl. "keyword"), genau wie PRINT, LIST und alle anderen BASIC-Kommandos, die der Computer versteht. Wenn ein solches Schlüsselwort auftritt, erkennt der Interpreter dies und speichert es nicht Buchstabe für Buchstabe ab, sondern als eine einzige Zahl. Man nennt diese Zahl den Interpretercode des dazugehörigen Schlüsselwortes. Oft wird statt Interpretercode auch "token" gesagt. Durch diese Methode der internen Abspeicherung wird sehr viel Speicherkapazität eingespart. Stellen Sie sich nur vor, wieviele PRINT-Anweisungen oft in einem Programm vorkommen und was geschähe, wenn diese jeweils fünf statt nur ein Byte belegen würden.
Ein Rahmen um den Bildschirm ist uns schon begegnet. Im Anleitungsprogramm zu WELTRAUMSCHLACHT finden wir aber eine ganz besondere Variante vor. Diesmal wird der Rahmen symmetrisch gezeichnet. Wenn Sie WELTRAUMSCHLACHT kennnen, wissen Sie, was ich meine. Sehen Sie sich die Routine in den Zeilen 3000 bis 3030 an.
In Zeile 3000 wird auf Kleinschrift umgeschaltet, der Bildschirm gelöscht, die Cursorfarbe blau gewählt und schließlich jede einzelne Bildschirmposition in der FOR-NEXT-Schleife mit blau vorbelegt. Zeile 3010 bringt in die vier Bildschirmekken die passenden Zeichen. Dabei entspricht VB der linken oberen Ecke (wie üblich in unseren Programmen), VB+21 der rechten oberen, VB+484 der linken unteren und VB+505 der rechten unteren. In den Zeilen 3020 und 3030 wird die Umrahmung komplett gemacht. Hier kommt es auf eine geschickte Wahl der Adressrechnung bei den einzelnen POKE-Befehlen an, um während des Zeichnens jederzeit die Symmetrie zu gewährleisten.
Wenn die Umrahmung einmal fertig ist, besteht natürlich kein Unterschied mehr zwischen der uns schon bekannten Methode und der neuen symmetrischen. Aber während der Zeit des Zeichnens macht sich die Symmetrie doch positiv bemerkbar. Schließlich möchte ich, daß man Ihren Programmen doch ein

klein wenig anmerkt, daß Sie "bei mir durch die Schulung gegangen sind".
Sehen Sie sich im WELTRAUMSCHLACHT-Hauptprogramm – jetzt kommen wir also endlich zum zweiten Teil – einmal den INPUT-Befehl an, bei dem der Spieler seinen Namen eingeben soll. Verantwortlich dafür sind die Programmzeilen 1000 bis 1020. Sicher fällt Ihnen auf, daß statt des normalen INPUT die Anweisung INPUT# (genannt File-INPUT, sprich "Feil-INPUT") genommen wird. Die zu diesem File-INPUT zugehörenden Befehle herausgeschrieben ergeben folgende Struktur:

```
1000 OPEN 1,0
1010 INPUT#1,N$
1020 CLOSE 1
```

In Zeile 1000 wird die Tastatur (Primäradresse 0)als Peripheriegerät mit der logischen Filenummer 1 angesprochen.



Dieses File-Konzept entspricht der Eingabe von externen Geräten wie zum Beispiel auch vom Rekorder (Primäradresse 1) oder von Floppydisk (Primäradresse 8), um Daten von diesen Geräten einzulesen. Mit OPEN wird also quasi ein Eingabekanal eröffnet, der im weiteren Verlauf mit der logischen File-

nummer 1 angesprochen werden kann. Es würde zu weit führen, an dieser Stelle das relativ komplexe Datenaustauschkonzept mit externen Geräten abzuhandeln. Außerdem ist dies für unseren Zweck auch gar nicht nötig.
In Zeile 1010 wird nun statt des normalen INPUT der File-INPUT verwendet. Wozu dieser Umstand überhaupt gut ist? Wenn Sie bei einem normalen INPUT ein Komma eingeben, so trennt der Computer die Eingabe in zwei Teile – einen vor und den anderen nach dem Komma – und liest nur den ersten Teil ein, während der zweite mit EXTRA IGNORED abgelehnt wird. Diese Fehlermeldung sieht natürlich nicht sehr elegant aus. Bei INPUT# wird zwar auch nur der erste Teil vor dem Komma angenommen und der Rest ignoriert, es kommt aber nicht zur Ausgabe irgendeiner Meldung. Der Bildschirmaufbau kann sich also nicht aufgrund einer falschen Eingabe völlig verzerren. Dies ist dann besonders wichtig, wenn sehr viele Informationen gleichzeitig auf dem Schirm stehen und eine unverhoffte Fehlermeldung unter Umständen einen Teil mittels "scrolling" (Bildschirmrollen) am oberen Rand hinausschiebt. INPUT# hilft Ihnen, solche kleinen Pannen von vornherein zu vermeiden.
Wichtig ist der CLOSE-Befehl in Zeile 1020, um das mit OPEN eröffnete File wieder zu schließen. Andernfalls kommt es beim nächsten Versuch, dasselbe oder ein anderes File mit der gleichen Filenummer 1 zu öffnen, zur Fehlermeldung FILE OPEN ERROR.
Beachten Sie, daß zu INPUT# im Unterschied zum normalen INPUT kein Text direkt mitgegeben werden darf. Der Text "Bitte geben Sie Ihren Namen ein" ist deswegen in WELTRAUMSCHLACHT Zeile 1010 dem INPUT#-Befehl vorangestellt. Er ist durch ein Semikolon abgeschlossen, damit die Eingabe direkt hinter dem Text stattfinden kann.
Nun will ich Sie aber nicht länger vom Abschießen der feindlichen Raumbasen abhalten. Und nehmen Sie es nicht zu tragisch, wenn Ihr Raumschiff in einem der unzähligen Sterne verglüht. Das Schöne an Computerspielen ist ja, daß man immer wieder von vorne anfangen kann, wenn etwas schiefgegangen ist.

Zusammenfassung:
Kapitel 10 hat Ihnen außer einem spannenden Spiel folgendes geboten:

- programmiertes LOAD (mit und ohne Variablenübernahme)
- Abfrage der PLAY-Taste des Rekorders mit PEEK und WAIT
- Motorsteuerung des Rekorders mit POKE
- Programmierung eines sich symmetrisch zeichnenden Rahmens
- Gebrauch von INPUT# für Eingaben von der Tastatur

```
0 GOTO100
1 :
2 WELTRAUMSCHLACHT
3 :
4 ANLEITUNG
5 :
6 (C) 1982 BY
7 :
8 ANDREAS DRIPKE
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 POKE657,128:POKE36879,25:PRINT"■"CHR$(14):IFPEEK(37151)AND64GOTO1020
1010 PRINT"BITTE LASSEN SIE DIE":PRINT"PLAY-TASTE GEDRUECKT.":GOTO1040
1020 PRINT"BITTE DRUECKEN SIE":PRINT"DIE PLAY-TASTE.":WAIT37151,64,64
1030 PRINT"OK":POKE192,1:POKE37148,254
1040 FORI=1TO3E3:NEXT:PRINT"■++ WELTRAUMSCHLACHT ++
1050 PRINT"++     ANLEITUNG      ++":FORI=1TO3E3:NEXT
2000 GOSUB3000:FORI=1TO99
2010 X=VB+506*RND(1):IFPEEK(X)-32GOTO2010
2020 POKEX,46:NEXT
2030 PRINT"SO SIEHT DER STERN-":PRINT"HIMMEL AUS:":FORI=1TO3E3:NEXT
2040 PRINT"WENN SIE MIT IHREM":PRINT"RAUMSCHIFF IN EINEN
2050 PRINT"DER STERNE HINEINGE-":PRINT"RATEN, VERGLUEHT ES
2060 PRINT"(PARDON, SIE AUCH)":PRINT"AUGENBLICKLICH.
2070 PRINT"■■SEIEN SIE VORSICHTIG":GOSUB4000:GOSUB3000
2080 PRINT"SO SIEHT IHR RAUM-":PRINT"SCHIFF AUS, WENN ES
2090 PRINT"NACH RECHTS FLIEGT:":PRINT"'>'. NACH LINKS
2100 PRINT"FLIEGEND NIMMT ES":PRINT"DIESE FORM AN: '<'.":GOSUB4000
```

```
2110 GOSUB3000:PRINT"FUER DIE ANDEREN":PRINT"■■RICHTUNGEN:■
2120 PRINTTAB(6)"Q   W   E":PRINTTAB(8)"■┌ ↑ ┐":PRINTTAB(6)"■A < * > S
2130 PRINTTAB(8)"■└ V ┘":PRINTTAB(6)"■Z   X   C":PRINT"■■DER BUCHSTABE GIBT
2140 PRINT"■■AN, WELCHE TASTE GE-":PRINT"■■DRUECKT WERDEN MUSS.":GOSUB4000
2150 GOSUB3000:PRINT"UM EIN TORPEDO ABZU-":PRINT"■■SCHIESSEN, MUESSEN
2160 PRINT"■■SIE AUF 'SPACE'":PRINT"■■DRUECKEN.":GOSUB4000:GOSUB3000
2170 PRINT"DER SINN DES SPIELS?":PRINT"■■NUN, DER BESTEHT DA-
2180 PRINT"■■RIN, DIE FEINDLICHEN":PRINT"■■RAUMBASEN ABZU-":PRINT"■■SCHIESSEN.
2190 GOSUB4000:GOSUB3000:PRINTCHR$(142)"DIE FEINDLICHEN BA-
2200 PRINT"■■SEN SEHEN SO AUS: ●":PRINT"■■IHRE BASIS HAT FOL-
2210 PRINT"■■GENDE FORM:  ┌┬┐":PRINTTAB(13)"├┼":PRINTTAB(13)"└┴┘":GOSUB4000
2220 GOSUB3000:PRINT"SIE DIENT DAZU, AUF-":PRINT"■■ZUTANKEN, ODER NEUE
2230 PRINT"■■TORPEDOS AUFZUNEH-":PRINT"■■MEN. SIE KOENNEN AUF
2240 PRINT"■■JEDER SEITE LANDEN.":PRINT"■■IHR RAUMSCHIFF STAR-
2250 PRINT"■■TET VON DER SEITE":PRINT"■■AUS, AN DER DIE BA-
2260 PRINT"■■SIS 'OFFEN' IST.":GOSUB4000:GOSUB3000:PRINT"VERGESSEN SIE NICHT,
2270 PRINT"■■DIE BASIS ANZUSTEU-":PRINT"■■ERN, FALLS IHNEN DER
2280 PRINT"■■TREIBSTOFF AUSGEHT.":PRINT"■■UND PASSEN SIE AUF,
2290 PRINT"■■DASS SIE NICHT IN":PRINT"■■EINEM DER STERNE
2300 PRINT"■■VERGLUEHEN. ES WAERE":PRINT"■■SCHADE!":GOSUB4000:GOSUB3000
2310 PRINT"NUN■■WUENSCHE■ICH■IHNEN":PRINT"■■VIEL■SPASS■BEIM■SPIELEN
2320 PRINT"■■IHR■ANDREAS■DRIPKE■■■■■":GOSUB4000:GOSUB3000:PRINT"DAS PROGRAMM■
2330 PRINT"■■WELTRAUMSCHLACHT■
2340 PRINT"■■WIRD JETZT GELADEN,":PRINT"■■FALLS ES SICH IM AN-
2350 PRINT"■■SCHLUSS AN DIESES":PRINT"■■PROGRAMM AUF DER
2360 PRINT"■■CASSETTE BEFINDET.":LOAD
3000 PRINTCHR$(14)"■■■";:FORI=VCTOVC+505:POKEI,6:NEXT
3010 POKEVB,112:POKEVB+21,110:POKEVB+484,109:POKEVB+505,125:FORI=1TO11
```

```
3020 POKEVB+I,64:POKEVB+21-I,64:POKEVB+484+I,64:POKEVB+505-I,64:POKEVB+22*I,93
3030 POKEVB+21+22*I,93:POKEVB+484-I*22,93:POKEVB+505-I*22,93:NEXT:RETURN
4000 POKE198,0:PRINTTAB(60)"SPACE";
4010 GETA$:IFA$<>" "GOTO4010
4020 RETURN



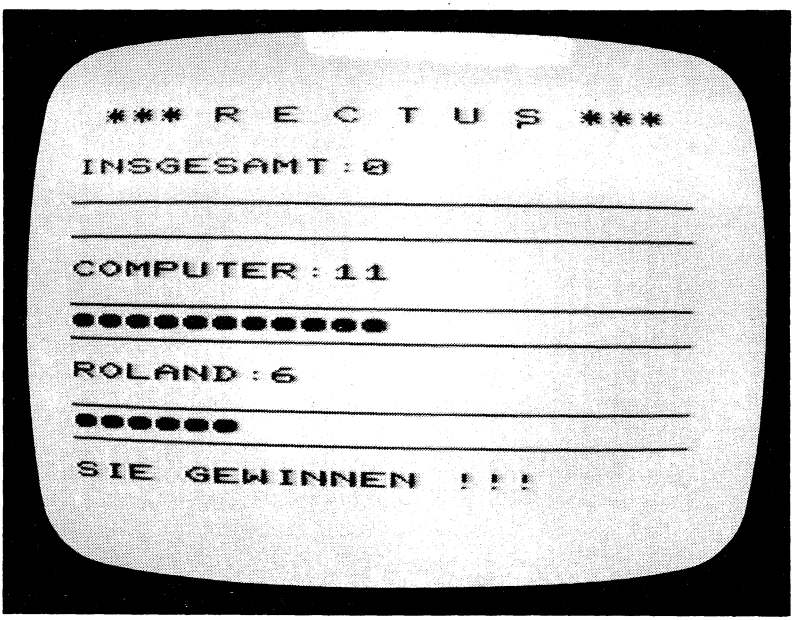
0 GOTO100
1 :
2 WELTRAUMSCHLACHT
3 :
4 (C) 1982 BY
5 :
6 ANDREAS DRIPKE
7 :
8 :
9 :
100 IFPEEK(45)=PEEK(47)ANDPEEK(46)=PEEK(48)GOTO120
110 POKE45,PEEK(174):POKE46,PEEK(175):CLR
120 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 POKE657,128:POKE36879,8:PRINTCHR$(142)"■■  WELTRAUMSCHLACHT■■■":OPEN1,0
1010 POKE198,0:PRINT"BITTE GEBEN SIE IHREN■■NAMEN EIN: ";:INPUT#1,N$:PRINT"■■■
1020 CLOSE1:PRINTN$", WELCHEN":PRINT"■SCHWIERIGKEITSGRAD","■WAEHLEN SIE? ";
1030 POKE204,0
1040 GETG$:S=VAL(G$):ONS+1GOTO1040:POKE204,1:POKE207,0:DIMA(3),B(3),D(9),R(9)
1050 FORI=1TO9:READR(I),D(I):NEXT:FORI=0TO3:READA,B(I):A(I)=A+VB+200:NEXT
1060 DATA109,21,22,22,125,23,60,-1,,,62,1,112,-23,30,-22,110,-21
```

```
1070 DATA1,114,22,107,45,113,24,115
2000 PRINT"■SPRIT:     TORPEDOS:":FORI=VCTOVC+505:POKEI,6:NEXT:B=VB+200
2010 G=INT(4+4*RND(1)):FORI=0TOS*4:POKEVB+484*RND(1)+22,46:NEXT:POKEB,85
2020 POKEB+1,114:POKEB+2,73:POKEB+22,107:POKEB+23,91:POKEB+24,115:POKEB+44,74
2030 POKEB+45,113:POKEB+46,75:R=4:FORI=1TOG
2040 X=484*RND(1)+VB+22:ON-(PEEK(X)<>32)GOTO2040:POKEX,81:NEXT:TI$="000000
2050 R=10-R:F=10:T=9:X=VB+223+D(R):POKE36878,2:FORI=0TO3:POKEA(I),B(I):NEXT
2060 FORI=57TO48STEP-1:POKEX,I:POKE36875,I+197:FORJ=0TO499:NEXT:NEXT:Y=X
2070 ONG+1GOTO3090:ONF+1GOTO3070:FORI=0TO49:NEXT:POKEX,32:X=Y:POKEX,R(R)
2080 F=F-.05:PRINT"■"SPC(6);:IFF<3THENPRINT"■";
2090 PRINTINT(F)"■■■"SPC(10)T:POKEA(P),B(P):P=P+1AND3:POKEA(P),32:POKE36878,2
2100 POKE36875,254-P:GETG$:IFG$=" "GOTO4000
2110 IFG$="Z"THENR=1
2120 IFG$="X"THENR=2
2130 IFG$="C"THENR=3
2140 IFG$="A"THENR=4
2150 IFG$="D"THENR=6
2160 IFG$="Q"THENR=7
2170 IFG$="W"THENR=8
2180 IFG$="E"THENR=9
2190 Y=X+D(R):POKEX,R(R):Y=Y-484*(Y<VB+22)+484*(Y>VB+505):PE=PEEK(Y)
2200 IFPE=32GOTO2070
2210 IFPE=46ORPE=81GOTO3000
2220 IFPE=85ORPE=73ORPE=74ORPE=75THENR=10-R:Y=X:GOTO2070
2230 IFPE=91THENPOKEX,32:GOTO2050
2240 GOTO2070
3000 Z=Y:GOSUB5000:PRINT"■"N$", SIE HABEN","■VERLOREN.
3010 S=S+(S>1)
```

```
3020 PRINT"■SIE BENOETIGTEN DAZU■":PRINTINT(TI/60)"SEKUNDEN.
3030 PRINT"■■NOCH EINE WEITERE","■RUNDE?":POKE198,0
3040 GETG$:IFG$="J"GOTO2000
3050 IFG$<>"N"GOTO3040
3060 PRINT"■ICH HOFFE, ES HAT","■SPASS GEMACHT.":POKE657,0:END
3070 FORI=230TO150STEP-.2:POKE36875,I:NEXT:PRINT"■DER TREIBSTOFF IST
3080 POKE36875,0:PRINT"■IHNEN AUSGEGANGEN.":GOTO3010
3090 PRINT"■ICH GRATULIERE IHNEN,■":PRINTN$", SIE HABEN
3100 PRINT"■ALLE FEINDLICHEN":PRINT"■BASEN VERNICHTET.":S=S+1:GOTO3020
4000 ONT+1GOTO2070:POKE36875,0:T=T-1:FORI=1TO6:J=Y+D(R)*I
4010 IFPEEK(J)=81THENI=6:NEXT:G=G-1:Z=J:GOSUB5000:GOTO2070
4020 IFPEEK(J)-32THENI=6:NEXT:GOTO2070
4030 POKEJ,R(R):FORK=0TO99:NEXT:POKEJ,32:NEXT:GOTO2070
5000 POKE36875,0:V=58:GOSUB5020:POKE36877,180:FORI=15TO0STEP-.05:POKE36878,I
5010 NEXT:POKE36877,0:V=32
5020 POKEZ,58:FORI=1TO2:FORJ=1TO9:K=Z+D(J)*I:IFK<VBORK>VB+505GOTO5040
5030 PE=PEEK(K):IFPE=32ORPE=58THENPOKEK,V
5040 NEXT:NEXT:RETURN
```

11. RECTUS

Diesmal serviere ich Ihnen ein Spiel, bei dem der Gegner Ihr eigener Computer ist. RECTUS erklärt sich selbst, die Spielregeln sind einfach.
Das Besondere an RECTUS ist, daß der Computer während des Spiels laufend dazulernt und somit von Runde zu Runde immer besser wird. Er bekommt also nicht irgendeine Gewinnstrategie mit auf den Weg, sondern fängt mit den grundlegenden Spielregeln an und muß sich erst langsam zum Meister emporarbeiten. Wir haben es also mit einer sogenannten Lernmaschine zu tun. Faszinierend, nicht wahr?
Das Prinzip des Maschinenlernens, das hier angewandt wird, ist sehr einfach. Wir wollen es jetzt besprechen, und Sie sollten deswegen RECTUS erst einmal eintippen und ein paarmal spielen. Andernfalls können Sie mit der Erklärung nämlich nicht viel anfangen.



Ich gehe davon aus, daß Sie nun schon einige Spiele hinter sich haben. Dann ist Ihnen vielleicht aufgefallen, daß der Computer am Anfang, also wenn er noch ziemlich "dumm" ist, immer genau 4 Steine wegnimmt, sobald er an der Reihe ist. Das

ist sicher nicht die beste Spielstrategie, aber immerhin eine mögliche. Wenn der Computer nun verliert, dann merkt er sich das. Kommt er zum zweiten Mal in eine bestimmte Situation, dann nimmt er diesmal 3 Steine weg. Gewinnt er auch damit nicht, wählt er beim nächsten Mal 2 Steine.
Das vollzieht er immer so weiter, bis er einmal gewinnt. Wenn dem so ist, dann vergißt er alle seine bisherigen Versuche und merkt sich nun, mit welchem Zug er gewonnen hat. Damit ist ein Lernschritt beendet.
Nun speichert der Computer aber nicht nur eine einzige Spielsituation, sondern alle. Und in welcher er sich auch befindet, stets geht er in der gleichen geschilderten Art und Weise vor. Irgendwann ist er dann soweit, daß er jede Situation perfekt beherrscht. Allerdings kann er nur so gut spielen, wie es ihm beigebracht wird. Das bedeutet also, umso besser Sie RECTUS beherrschen, umso besser spielt es auch Ihr Gegenüber, der Computer.
Wir wollen uns nun dem Programm zuwenden. Schließlich soll ja nicht nur der Computer lernen, sondern auch wir wollen ein paar neue Dinge erfahren.
Wichtig für RECTUS ist, daß zu Spielbeginn eine ungerade Anzahl von Steinen erzeugt wird. Bei einer geraden Anzahl am Anfang bekämen auch beide Spielpartner am Ende eine gerade Zahl und es ließe sich kein Gewinner feststellen. Dafür, daß auch immer eine ungerade Anzahl erzeugt wird, sorgt Zeile 1180.

```
1180 V=9+2*INT(RND(1)*7)
```

Zunächst wird mit 'INT(RND(1)*7)' eine beliebige ganze Zahl zwischen 0 und 6 erzeugt. Die Nachkommastellen werden durch INT abgeschnitten. Das Resultat hiervon wird mit 2 multipliziert. Durch diese Mulitplikation mit 2 entsteht auf jeden Fall eine gerade Zahl, und zwar zwischen 0 und 12. Mathematisch ausgedrückt:

```
X=2*K
```

führt immer dazu, daß X gerade wird, und zwar unabhängig davon, welchen Wert K besitzt. Probieren Sie es aus!
Zu der so erzeugten geraden Zahl wird nun noch 9 hinzuaddiert, und es entsteht eine ungerade Zahl zwischen 9 und 21. Sie sehen, wenn man die Formel so auseinandernimmt, ist sie gar nicht so schwierig, wie sie vielleicht auf den ersten Blick aussieht.

Wir haben auch die Möglichkeit, innerhalb eines Programmes nachzuprüfen, ob eine bestimmte Zahl gerade oder ungerade ist. Nehmen wir an, unsere Zahl steht in der Variablen X, dann liefert folgende Programmzeile für F den Wert 1, wenn X gerade ist. Im anderen Fall nimmt F den Wert 0 an.

```
F=0 : IF (X/2)=INT(X/2) THEN F=1
```

Eine ähnliche Abfrage finden Sie in RECTUS zum Beispiel in Programmzeile 1290.
Das war ein bißchen Mathematik aus der Schulzeit. Für manche Programme ist sie eben doch unerläßlich. Doch nun genug davon, wenden wir uns anderen Dingen zu.
Die Routine, in der Sie eingeben können, wieviele Spielsteine Sie nehmen möchten, liegt in den Zeilen 1240 bis 1260. Der GET-Befehl und die 'ON-(..)GOTO'-Anweisung dürften für Sie – als aufmerksamer Leser des vorliegenden Buches – ohne weiteres verständlich sein, ebenfalls der Befehl 'POKE198,0'. Sie finden in den genannten zwei Zeilen aber noch zwei weitere POKE-Befehle, nämlich 'POKE204,0' (Zeile 1240) und 'POKE204,1' (Zeile 1260). Was hat es damit auf sich?
Wir haben in anderen Kapiteln schon besprochen, wie es möglich ist, auch bei GET-Eingabe einen blinkenden Cursor auf dem Bildschirm darzustellen. Mit den beiden genannten POKE-Befehlen geht dies nun ganz besonders einfach. Mit

```
POKE 204,0
```

wird nämlich einfach der Cursor abgeschaltet, d.h. unsichtbar, nach

```
POKE 204,1
```

erscheint der Cursor wieder. Auf diese Art und Weise erspart man sich also das Programmieren einer Cursorsimulation mit PRINT, was ja doch mit mehr Aufwand verbunden ist. Hier sieht man einmal mehr, daß man sich durch das richtige "know-how" oft viel Arbeit ersparen kann.
Sicher ist Ihnen die MID$-Funktion bekannt. Sie ist eine der am häufigst benutzten Stringfunktionen überhaupt. Allgemein hat sie das Format

```
X$ = MID$(String,Position,Länge)
```

Als String kann hierbei entweder ein Text in Anführungszeichen oder aber eine Stringvariable dienen. Die numerischen Parameter, also Position und Länge, können ebenfalls durch Variablen ersetzt werden. Zur kurzen Auffrischung des Gedächtnisses: MID$ nimmt einen Teilstring aus dem Gesamtstring ab der angegebenen Position mit der entsprechenden Länge heraus. Soweit ist alles klar. Sehen Sie sich aber jetzt einmal die Programmzeile 3030 in RECTUS an. Der dort vorkommende MID$-Befehl hat die Form

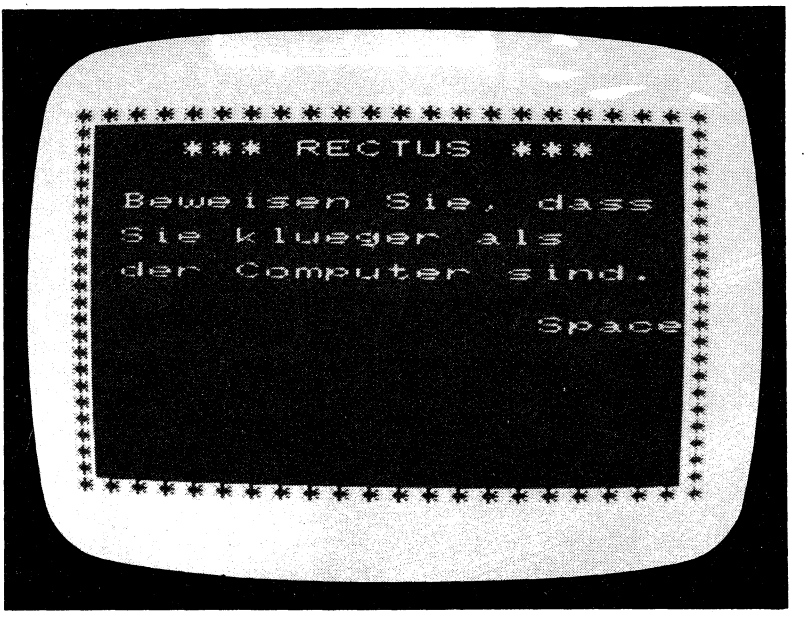
MID$(String,Position)

Es spielt hierbei keine Rolle, daß der String erst durch STR$ zustandekommt. Nein, was ich meine ist, daß der dritte Parameter, die Längenangabe, fehlt. Ich erlebe es immer wieder, daß selbst ansonsten recht erfahrene Programmierer um diese Variante der MID$-Funktion nicht wissen. MID$ muß mindestens zwei Argumente – String und Position – haben, das dritte Argument – die Länge – ist optional, es kann auch weggelassen werden. Sind bei MID$ nur zwei Argumente angegeben, so wird der Ursprungsstring ab der gewählten Position in seiner vollen Länge genommen. Ein Beispiel macht Ihnen das sofort klar.

X$ = MID$("INTERFACE AGE",6)

weist der Variablen X$ den Inhalt "FACE AGE" zu, also die größtmögliche Länge. Zu Ihrer Information: INTERFACE AGE (COMPUTER LIFE) ist der Verlag, in dem das vorliegende und auch alle anderen Bände der VC/CBM-Buchreihe erschienen sind.
Wenn ein Text auf dem Bildschirm ausgegeben werden soll, so nimmt man üblicherweise für jede neue Textzeile auch einen neuen PRINT-Befehl. RECTUS zeigt, daß es auch anders geht. Nehmen wir als Beispiel die Programmzeile 1010 heraus. Die Steuerzeichen "Cursor nach unten" dienen dazu, jeweils eine Leerzeile zwischen den Textzeilen auf dem Bildschirm zu lassen, die Zeichen "Cursor nach rechts" an jedem Textanfang sind notwendig, damit der vorher auf den Bildschirm gebrachte Rahmen nicht überschrieben wird. Und obwohl Zeile 1010 nur einen einzigen PRINT-Befehl beinhaltet, werden damit drei verschiedene Texte ausgedruckt. "Beweisen Sie, dass" ist die erste Zeile. Da der Satzteil genau bis zum rechten Bildschirmrand reicht, rutscht der Cursor automatisch eine Zeile tiefer. Das "Cursor nach unten" schafft eine weitere Leerzeile, und die zwei "Cursor nach rechts" bewirken die

korrekte Positionierung des folgenden Textes "Sie klueger als". Da dieser Satzteil ein wenig kürzer ist, steht der Cursor diesmal nicht automatisch am Anfang der nächstfolgenden Zeile. Deswegen ist an dieser Stelle im Programm ein Komma eingefügt. Dieses Komma bei PRINT bewirkt einen Sprung an die nächste vortabulierte Position – entweder der linke Bildschirmrand oder die Bildschirmmitte, je nachdem, welches von beiden als erstes erreicht wird. Da in unserem Fall der Text schon über die Mitte des Bildschirmes hinausreicht, ist die nächste Position also der linke Rand der nächsten Bildschirmzeile. Nun ist nur noch die schon bekannte Positionierung mit einmal "Cursor nach unten" und zweimal "Cursor nach rechts" nötig, und schon kann das Satzende "der Computer sind" ausgedruckt werden. Ein abschließendes Anführungszeichen ist nicht notwendig, da in dieser Programmzeile kein weiterer Befehl mehr folgt – es ist deswegen in RECTUS in allen diesen Fällen weggelassen.



Sehen Sie sich auch die anderen PRINT-Zeilen an, sie sind sehr ähnlich programmiert. Auf diese Weise können wir also auf mehrere PRINT-Anweisungen pro Programmzeile verzichten.
Was haben alle NEXT-Anweisungen in RECTUS gemeinsam? Sie finden NEXT in den Programmzeilen 1150, 1190, 1200,

1260, 1310, 1320, 1350, 1360, 2020, 3050, 5000 und 5010. Richtig, allen diesen NEXT-Anweisungen ist nicht die Schleifenvariable der zugehörigen FOR-Anweisung beigegeben. Eine korrekte FOR-NEXT-Schleife sollte folgende Form haben:

```
FOR X = A TO B STEP C
NEXT X
```

Wird STEP weggelassen, so wird C gleich 1 angenommen. Wie diese allgemeine Form zeigt, wird zu NEXT die Variable angegeben, die die Werte von A bis B durchlaufen soll, die sogenannte Schleifenvariable, in unserem Beispiel X. Wird diese Variable weggelassen – wie in RECTUS immer geschehen – so schließt der Computer mit NEXT die Schleife, die zuletzt durch ein FOR eröffnet wurde. Nehmen wir als Beispiel:

```
FOR X = A TO B
FOR Y = K TO L
NEXT
```

Da dem NEXT-Befehl keine Schleifenvariable beigegeben ist, bezieht der Computer dieses NEXT auf die zuletzt mit FOR geöffnete Schleife, was in diesem Fall also 'NEXT Y' gleichkommt. Soll die FOR-NEXT-Schleife mit X angesprochen werden, so muß X explizit angegeben sein.

```
FOR X = A TO B
FOR Y = K TO L
NEXT X
```

In diesem Fall wird durch 'NEXTX' nicht nur die Schleife X geschlossen (d.h., X um eins erhöht und solange X nicht größer als B ist, zum Befehl direkt nach 'FORX=. . 'zurückverzweigt), sondern auch die Schleife Y. Dies kann zum Beispiel bei folgender Programmstruktur sinnvoll sein:

```
FOR X = A TO B
FOR Y = K TO L
IF ... THEN GOTO
NEXT Y
NEXT X
```

Dies soll nur die Struktur aufzeigen, selbstverständlich sind in einem echten Programm weitere Befehle und Zeilennummern vorhanden. Ist nun die Bedingung bei IF . . THEN erfüllt, so verzweigt das Programm zu 'NEXT X', wodurch die

X-Schleife wiederholt wird. Die Y-Schleife wird dadurch automatisch ebenfalls abgeschlossen, so daß man sich darum im Grunde nicht mehr zu kümmern braucht. Durch die Verwendung der Schleifenvariablen bei NEXT ist Eindeutigkeit geschaffen. An diesem einfachen Beispiel zeigt sich auch schon die Gefahr, die darin steckt, bei NEXT keine Schleifenvariable anzugeben. Wenn wir diese nämlich in unserem Beispiel weglassen, dann sieht es so aus:

```
FOR X = A TO B
FOR Y = K TO L
IF ... THEN GOTO
NEXT
NEXT
```

Ist nun die Bedingung zwischen IF und THEN erfüllt, so springt der Programmablauf auf den zweiten NEXT-Befehl, der eigentlich als 'NEXTX' gedacht ist. Da aber kein X angegeben ist, bezieht der Computer dieses NEXT auf die zuletzt mit FOR eröffnete Schleife – und das ist die Y-Schleife. Damit kommt es zu einem völligen Chaos bei der Verschachtelung der zwei FOR-NEXT-Schleifen. In einem komplexen Programm mit einer Vielzahl von Verschachtelungen wird die Gefahr für einen solchen Fehler natürlich noch um ein Vielfaches größer, insbesondere, wenn – wie in unserem Beispiel – aus Schleifen heraus mittels GOTO oder GOSUB gesprungen wird.
Wenn in RECTUS bei NEXT trotzdem alle Schleifenvariablen weggelassen sind, dann deswegen, weil dies natürlich Speicherplatz spart. Außerdem ist NEXT ohne Zusatz ein wenig schneller als zum Beispiel 'NEXTX'. Gerade bei einer FOR-NEXT-Schleife, die ja oftmals mehrere hundert oder gar mehrere tausend Mal durchlaufen wird, macht sich ein solcher Geschwindigkeitsfaktor durchaus bemerkbar. Da in RECTUS die Schachtelung von FOR-NEXT-Schleifen sowieso nicht sehr ausgeprägt ist, habe ich also hier bloßes NEXT gewählt.
Bei größeren und komplexeren Programmen sollten Sie dies aber nicht tun. Ich habe Ihnen an einem Beispiel gezeigt, wie leicht Fehler beim Ablauf eines Programmes auftreten, wenn die Schleifenvariablen bei NEXT weggelassen werden. Eine gut durchdachte Schleifenprogrammierung mit vollständigem NEXT vermeidet solche Fehler von vornherein.
Wenn Sie RECTUS spielen, dann bedenken Sie, daß der Computer erst nach und nach lernt. Stempeln Sie das Spiel nicht als langweilig ab, nur weil Sie am Anfang vielleicht zehnmal hintereinander gewinnen. Geben Sie dem Programm etwas Zeit, und es wird Ihnen ein (fast) unschlagbarer Gegner sein.

Zusammenfassung:
Mit RECTUS haben sie endlich die Möglichkeit zu beweisen, daß Sie Ihrem Computer wirklich überlegen sind. Wenn Sie zusätzlich noch folgendes gelernt haben, freue ich mich:

- Strategie einer Lernmaschine
- Erzeugung gerader und ungerader Zufallszahlen in einem Bereich
- Überprüfung einer Variablen auf gerade oder ungerade
- Ein- und Ausschalten des Cursors mit POKE
- MID$ mit zwei Argumenten
- PRINT-Befehl über mehrere Bildschirmzeilen
- NEXT mit und ohne Schleifenvariable

```
1 REM >>>>>>>>>>
2 REM <          <
3 REM > RECTUS <
4 REM <          <
5 REM >>>>>>>>>>
6 REM
7 REM (C) 1982 BY
8 REM ANDREAS DRIPKE
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 POKE657,128:P=170:Q=1:R=9:GOSUB5000:PRINT"■■■■■■■*** RECTUS ***■
1010 PRINT"■■■BEWEISEN SIE, DASS■■■■SIE KLUEGER ALS","■■■DER COMPUTER SIND.
1020 GOSUB4000:GOSUB5000:PRINT"■■■AM ANFANG LIEGT","■■■EINE UNGERADE AN-
1030 PRINT"■■■ZAHL AN SPIELSTEI-■■■■NEN DA, WOVON SIE■■■■UND DER COMPUTER
1040 PRINT"■■■NUN ABWECHSELND 1,■■■■2, 3 ODER 4 STEINE■■■■WEGNEHMEN.
1050 GOSUB4000:GOSUB5000:PRINT"■■■GEWONNEN HAT DER-■■■■JENIGE, DER AM EN-
1060 PRINT"■■■DE EINE GERADE AN-■■■■ZAHL AN SPIELSTEI-■■■■NEN HAT.
1070 PRINT"■■■DER COMPUTER LERNT■■■■MIT JEDEM SPIEL","■■■EIN BISSCHEN DAZU.
1080 GOSUB4000:GOSUB5000:PRINT"■■■WAEHREND ER AM AN-■■■■FANG NOCH RECHT
1090 PRINT"■■■UNGESCHICKT","■■■SPIELT, ERHOEHT","■■■SICH SEINE SPIEL-
1100 PRINT"■■■STAERKE ALSO LAU-■■■■FEND.":GOSUB4000:GOSUB5000
1110 PRINT"■■■NACH CA. 30 SPIE-■■■■LEN BEHERRSCHT ER■■■■DAS SPIEL PERFEKT.
1120 PRINT"■■■ABBRUCH ERFOLGT","■■■DURCH EINGABE VON■■■■0 SPIELSTEINEN.
1130 GOSUB4000:PRINT"■IHR SPIELPARTNER IST":PRINT"■DER VC-20.
1140 PRINT"■■UND WIE HEISSEN SIE ?■■: ";:OPEN1,0:INPUT#1,S$:PRINT:CLOSE1
1150 DIMF%(1,5):FORI=0TO5:F%(0,I)=4:F%(1,I)=4:NEXT:POKE36879,25
1160 X=0:Y=0:PRINTCHR$(142)"■■■*** R E C T U S ***":A$="■————————————————————
1170 PRINT"■■INSGESAMT:":PRINTA$:PRINTA$:PRINT"■COMPUTER:":PRINTA$:PRINTA$
```

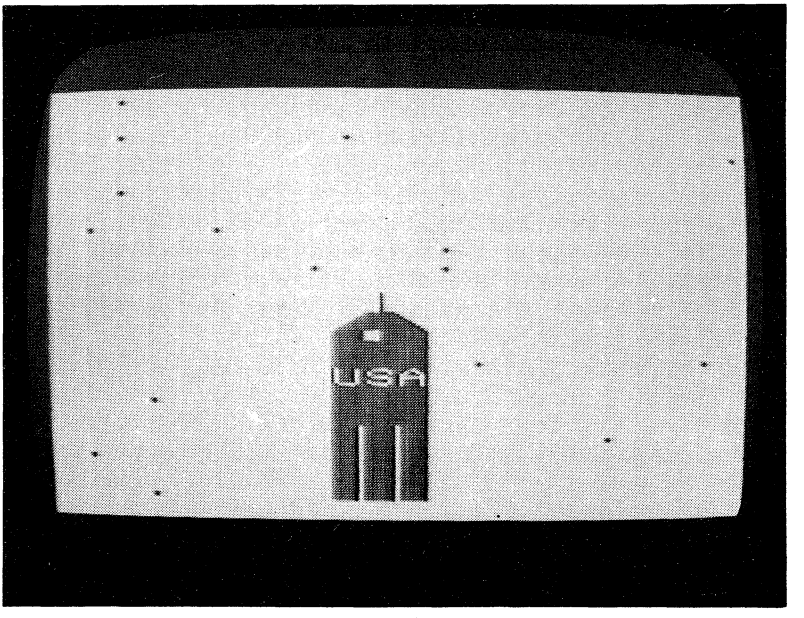
```
1180 PRINT"■"S$":":PRINTA$:PRINTA$:V=9+2*INT(RND(1)*7):GOSUB3000
1190 PRINT"■■ES SIND"V"STEINE.":FORI=1TO3E3:NEXT:PRINT"□COMPUTER BEGINNT. "
1200 FORI=1TO3E3:NEXT
1210 GOSUB3000:A1=A:B1=B:A=(X/2-INT(X/2))*2:B=INT((V/6-INT(V/6))*6+.5)
1220 S=F%(A,B):IFS>=VGOTO1280
1230 IFS<=0THENV=0:Y=Y+S:GOTO1300
1240 V=V-S:GOSUB2000:PRINT"■■SIE NEHMEN (1-4) ?  ■■";:POKE204,0:POKE198,0
1250 GETA$:ON-(A$="0")GOTO1390:S=VAL(A$):IFS<1ORS>4ORS>VGOTO1250
1260 POKE204,1:PRINTA$:FORI=1TO2E3:NEXT:IFS=VTHENV=0:X=X+S:GOTO1290
1270 V=V-S:X=X+S:GOTO1210
1280 F%(A,B)=V:S=V:V=0:GOSUB2000
1290 IF(Y/2)=INT(Y/2)GOTO1340
1300 GOSUB3000:PRINT"■■SIE GEWINNEN !!!      ":H=H+1:POKE36878,10
1310 FORI=235TO254STEP.05:POKE36876,I:NEXT:POKE36876,0:POKE36878,0
1320 FORI=1TO2E3:NEXT:IFF%(A,B)=1GOTO1370
1330 F%(A,B)=F%(A,B)-1:GOTO1160
1340 GOSUB3000:PRINT"■■RETSCH - ICH GEWINNE.":C=C+1:POKE36878,10
1350 FORI=254TO235STEP-.05:POKE36876,I:NEXT:POKE36876,0:POKE36878,0
1360 FORI=1TO2E3:NEXT:GOTO1160
1370 IFF%(A1,B1)=1GOTO1160
1380 F%(A1,B1)=F%(A1,B1)-1:GOTO1160
1390 POKE204,1:PRINTCHR$(14)"□SIE HABEN"H"MAL,":PRINT"■ICH HABE"C"MAL VON■
1400 POKE657,0:PRINTC+H"SPIELEN GEWONNEN.■■":IFC<>HGOTO1430
1410 PRINT"ICH REICHE IHNEN DIE":PRINT"■HAND (REIN SYMBO-
1420 PRINT"■LISCH) - WIR SIND":PRINT"■GLEICH GUT.":END
1430 IFH>CTHENPRINT"SEHR GUT - ICH GRATU-■■LIERE IHNEN.":END
1440 PRINT"SIE HABEN SICH IHRE","■BILDUNG WOHL IN EINER
1450 PRINT"■BAUMSCHULE ANGEEIG-","■NET?!":END
```

```
2000 PRINT"■■ICH NEHME";:IFS=1THENPRINT" EINEN STEIN":GOTO2020
2010 PRINTS"■ STEINE  "
2020 Y=Y+S:FORI=1TO2E3:NEXT
3000 PRINT"■"SPC(76);:Z=V:GOSUB3030
3010 PRINTSPC(53);:Z=Y:GOSUB3030
3020 PRINTSPC(45+LEN(S$));:Z=X
3030 PRINT"   ■■■"MID$(STR$(Z),2):PRINT"■■                                    ■
3040 ONZ+1GOTO3060:POKE36878,10:FORI=1TQZ:PRINT"●";:POKE36876,128+I
3050 FORJ=0TO29:NEXT:NEXT:POKE36876,0:POKE36878,0
3060 PRINT:RETURN
4000 POKE198,0:PRINTSPC(60)"SPACE
4010 GETA$:ON-(A$<>" ")GOTO4010:RETURN
5000 PRINT"■"CHR$(14):POKE36879,R:FORA=0TO21:GOSUB5020:NEXT
5010 FORA=22TO462STEP22:GOSUB5020:NEXT:RETURN
5020 POKEVB+A,P:POKEVC+A,Q:POKEVB+505-A,P:POKEVC+505-A,Q:RETURN
```

12. MOONRAKER

Trotz des englischen Namens ist auch dieses Spielprogramm in deutsch gehalten. Es geht darum, die von einem Raumschiff abgekoppelte Landefähre sicher auf der Mondoberfläche zu landen. Das ist zwar nicht ganz einfach, aber – ich habe es oft genug ausprobiert – doch möglich.
Während die zur Steuerung der Landefähre wichtigen Werte auf der linken Seite des Bildschirms digital zu sehen sind, erscheint rechter Hand eine Analogskala für die Höhe. Obwohl man heutzutage gerne Digitalanzeigen für alles Mögliche benutzt, denken Sie nur an die Digitaluhren, so ist die Analoganzeige doch geeigneter, wenn es nicht nur darum geht, einen einzelnen Wert exakt ablesen zu können, sondern auch die Tendenz zu sehen. Aus diesem Grund hat also auch MOONRAKER eine analoge Höhenskala.
Wir wollen uns jetzt ansehen, wie diese Analogskala programmiert ist. Da wird zunächst einmal die Skala selbst ausgedruckt, die Höhenskala von 20000 bis 1000 m liegt in den Zeilen 4300 und 4310. Da für Höhen unter 2000 m eine genauere Auflösung nötig ist, gibt es eine extra Skala für den Bereich von 2000 bis 0 m. Auf den Bildschirm gebracht wird dies in den Zeilen 4400 und 4410. Der Maximalwert beider Skalen wird jeweils der Variablen MA zugewiesen (Zeilen 4310 und 4410). Auf der ersten Skala bewegt sich der Zeiger in 1000er Schritten, auf der zweiten in 100er Schritten. Die Ablesegenauigkeit ist somit um den Faktor 10 besser. Diese Schrittweiten werden in den Zeilen 4310 bzw. 4410 den Variablen DV zugeordnet. Sie erinnern sich, '1E3' ist die sogenannte Exponentialschreibweise und bedeutet dasselbe wie 1000, also eine Eins mit drei Nullen (Zeile 4310).
Auf dem Bildschirm sind nicht die vollen Werte ausgeschrieben, also zum Beispiel 20000, sondern um den Faktor tausend geringere Zahlen. Dies ist einfach aus Platzgründen so programmiert. Wenn also auf dem Bildschirm der Zeiger bei 18 steht, so bedeutet dies, daß die Landefläche sich 18 Mal 1000, also 18000 m, über der Mondoberfläche befindet. Stünde die 18000 direkt in der Anzeige, so würde die ganze Analogskala unförmig breit sein. Wenn Sie also selbst einmal eine Analoganzeige programmieren, überlegen Sie, ob sich nicht auch ein Multiplikationsfaktor finden läßt, der die Skala wesentlich vereinfacht.
Wie die beiden beschriebenen Routinen wirken, sehen Sie am einfachsten, wenn Sie einmal 'GOSUB4300' und 'GO-

SUB4400' eingeben. Die Benutzung von GOSUB anstelle von GOTO ist sinnvoller, da beide Teile als Unterprogramme ausgelegt sind und somit mit RETURN abschließen. Bei GOTO käme es deshalb zu einem RETURN WITHOUT GOSUB ERROR.



Nachdem wir nun die Skala besprochen haben, fehlt uns eigentlich noch der wichtigste Teil einer Analoganzeige, nämlich der Zeiger. Wie können wir einen in Form einer Variablen ja digital (d.h. als Zahl) vorliegenden Wert in eine analoge Bewegung umsetzen? Nun, sehen Sie sich einmal die Zeilen 5000 bis 5040 an. Allerdings wollen wir Zeile 5000 für einen Moment ausklammern, wir werden später noch darauf kommen. In den Programmzeilen 5010 und 5020 wird anhand der Variablen DV (Schrittweite der Skala) und MA (Maximalwert der Skala) bestimmt, welche der beiden die Skala zeichnenden Routinen (bei 4300 oder bei 4400) aufgerufen wird. Wir haben diese Zeilen bereits besprochen. In Zeile 5030 wird der gesamte Bereich, in dem sich der Zeiger bewegen kann, gelöscht. Sie erinnern sich, 32 ist der Bildschirmcode für das Leerzeichen. VB ist wie üblich die niedrigste Bildschirmadresse, also die Position links oben. Die Schrittweite von 22 ergibt sich aus der Bildschirmbreite von 22 Zeichen, und die Rechnungen 'VB+43' und 'VB+485' resultieren daraus, daß der Zei-

ger für die Höhenmessung am rechten Bildschirmrand läuft. Außerdem wird in Zeile 5030 die aktuelle Höhe der Mondlandefähre ausgerechnet und der Variablen X1 zugewiesen. Sollte X1 dabei größer als das Maximum der Skala werden, so wird X1 gleich MA gesetzt. Dies ist bei Spielbeginn der Fall. Die Fähre beginnt nämlich in einer Höhe von 23000 m, wohingegen die Analogskala erst bei 20000 m Höhe anfängt. Die richtige Positionierung des Zeigers (Pfeil nach links) erfolgt in Zeile 5040. Zuerst wird die Farbe (VC) mit schwarz festgelegt und sodann der Pfeil (Bildschirmcode 31) auf den Bildschirm "gePOKEt". Machen Sie sich die dabei verwendete Formel klar:

```
POKE VB+43+(20-X1)*22,31*
```

Die 43 dient dazu, den Pfeil in die entsprechende Spalte auf der rechten Seite des Bildschirms zu bringen. Wird sie weggelassen, so würde sich der Zeiger am linken Bildschirm bewegen. Die Zahl 20 ergibt sich daraus, daß die Skala in 20 unterschiedlichen Positionen untereinander aufgeteilt ist und die 22 wiederum aus der Bildschirmbreite von 22 Zeichen pro Zeile. Je geringer nun X1 als Indikator für die Höhe der Landefähre wird, desto größer wird der Ausdruck (20–X1), der – mit 22 multipliziert – auf eine immer tiefere Bildschirmzeile verweist. Je kleiner also X1 wird, desto tiefer wird der Zeiger gesetzt. Für die Farbsteuerung gilt entsprechendes.
Sie sehen, eine Analoganzeige zu programmieren ist gar nicht so schwer. Wir wollen uns deswegen auch gleich noch die Horizontalanzeige anschauen, welche die Stärke der Bremsstöße von 1 bis 9 anzeigt. Auch hier wird keine Ziffer dargestellt, sondern ein Zeiger bewegt sich an einer Skala entlang.
Die dafür verantwortliche Zeile hat die Nummer 5000, wir haben sie vorhin ausgeklammert, da sie erst hierher gehört. Zunächst wird durch PRINT der gesamte Bereich, in dem der Zeiger (Pfeil nach oben) laufen kann, gelöscht. Das "HOME" und "Cursor nach unten" in der PRINT-Anweisung dienen dazu, die entsprechende Bildschirmposition anzufahren. Der Pfeil selbst wird mittels POKE auf den Bildschirm gebracht. Zuerst wird die Farbe, dann das Zeichen festgelegt. Der Bremsstoß ist in der Varialben S von 0 bis 9 gespeichert. Je größer S wird, desto weiter nach rechts wird der Pfeil dargestellt. Die Zahl 243 ist lediglich ein Korrekturfaktor, um die Zeigerbewegung an der richtigen Stelle auf dem Bildschirm zu haben. Sie kann sehr einfach durch Probieren oder mit Hilfe des Bildschirmentwurfblattes aus Kapitel 2 bestimmt werden.

Damit wollen wir uns den Analoganzeigen ab- und den Digitalanzeigen zuwenden.
Geschwindigkeit, Höhe, Treibstoff und Zeit werden mittels Zahlen dargestellt. Das Hauptproblem hierbei ist, diese Zahlen an der richtigen Stelle abzuschneiden, damit nicht durch zu viele Nachkommastellen der gesamte Bildschirmaufbau gestört wird. Dies ist am einfachsten durch Stringmanipulation zu erreichen. Zeile 4200 zeigt Ihnen, wie ich das meine. Die Zahl, die dargestellt werden soll, ist in der Variablen C abgespeichert, die Anzahl der Stellen, die zur Anzeige gebracht werden sollen, in der Variablen D. C$ hat nach dem Aufruf von Zeile 4200 die Zahl von C im Format mit der Länge D. Machen wir uns nun Zeile 4200 in Ruhe klar, schließlich sieht sie auf den ersten Blick ziemlich verwirrend aus. Da wird C mittels der Funktion STR$ zunächst in einen String umgewandelt. Nehmen wir an, C habe den Wert 5.7 (angelsächsischer Dezimalpunkt anstelle des kontinentaleuropäischen Dezimalkommas), so entsteht mit STR$ der String " 5.7". Das Leerzeichen am Anfang steht als Ersatz für ein positives Vorzeichen. Wäre unsere Zahl –5.7, so entstünde der String "–5.7". Da wir es bei MOONRAKER nur mit positiven Zahlen zu tun haben, wird das Leerzeichen am Anfang des Strings abgeschnitten. Dafür verantwortlich ist die 2 in dem MID$-Befehl. Sie bedeutet, daß der String, der durch 'STR$(C)' entstanden ist, erst ab dem zweiten Zeichen herauszunehmen ist. Dadurch fällt das erste Zeichen – nämlich das bewußte Leerzeichen – weg. Mit D wird die Länge des Strings angegeben. Nachkommastellen, die darüber hinausreichen, werden einfach abgeschnitten. Ist eine Zahl kleiner als die durch D festgelegte Länge, so wird sie mit Leerzeichen am Anfang aufgefüllt. Aus der Zahl 5.7 mit D gleich 5 wird also der String " 5.7". Damit dies unter allen Umständen gewährleistet ist, befindet sich in Zeile 4200 die RIGHT$-Funktion. Wie Sie sehen, hat sie als Stringargument einen String, der aus lauter Leerzeichen besteht, das numerische Argument zu RIGHT$ ist das zweite D. Auf diese Weise wird der Wert in C rechtsbündig in C$ übernommen und gleichzeitig von links mit Leerzeichen aufgefüllt. Dieser Trick wird ziemlich oft angewandt, um ein festes Format mit genau definierter Länge zu erhalten. Am einfachsten probieren Sie die Arbeitsweise von Zeile 4200 einmal aus, indem Sie den Variablen C und D Werte zuweisen und dann 'GOSUB4200' eingeben. Mit '?C$' können Sie sich das Ergebnis ansehen.
Die Aufrufe von Programmzeile 4200 liegen in den Zeilen 3080, 3100, 3120 und 3130. Beachten Sie, wie den Variablen C und D jeweils vorher die aktuellen Werte zugewiesen werden.

C$ wird durch PRINT-Befehle an den gewünschten Stellen auf den Bildschirm gebracht.
Verwechseln Sie dieses Formatieren einer Zahl nicht mit dem mathematischen Runden. Durch RIGHT$ und MID$ wird lediglich eine feste Länge des formatierten Strings (C$) bestimmt, unabhängig von der Zahl selbst. Für das mathematische Runden gilt dagegen folgende Formel:

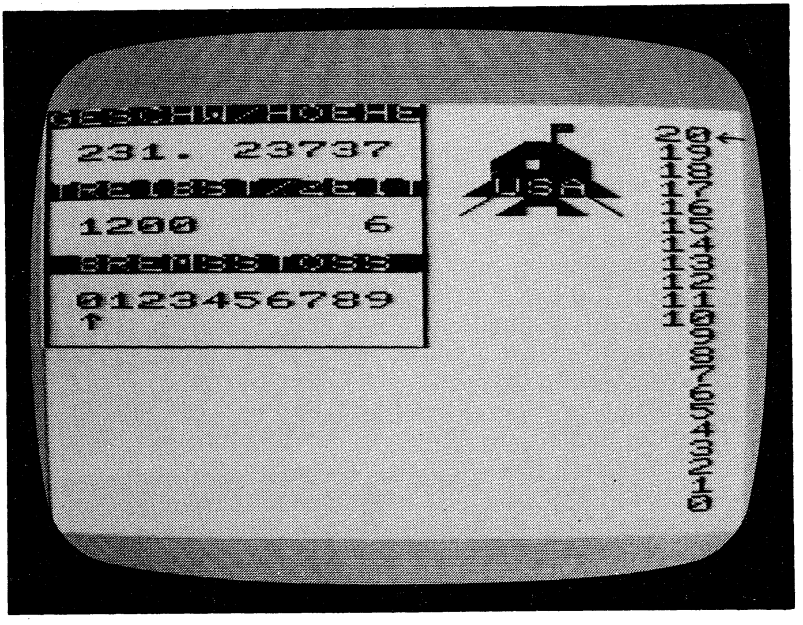
Z=INT((X*10↑Y)+0.5)/(10↑Y)

Dabei bedeuten: X = Zahl, die gerundet werden soll
Y = gew. Anzahl der Nachkommastellen
Z = Ergebnis

Um also zum Beispiel eine beliebige Zahl X auf 2 Nachkommastellen zu runden, müssen wir unsere Formel wie folgt ändern:

Z = INT((X*100)+0.5)/100

Probieren Sie es aus, indem Sie die Formel programmieren und dann für X verschiedene Werte einsetzen. Durch einige Überlegungen kann man sich ebenfalls leicht von der Richtigkeit überzeugen. Zunächst wird X mit 100 multipliziert. Dadurch rückt die zweite Stelle nach dem Komma an die letzte



Stelle vor dem Komma. Nun erfolgt die eigentliche Rundung durch Addition von 0.5. Entweder kommt es zu einem Übertrag (nämlich wenn vorher bereits 0.5 oder mehr an der ersten Nachkommastelle vorhanden war) oder nicht. Mit INT wird nun der Rest, der hinter der ursprünglichen zweiten Nachkommastelle folgte, abgeschnitten. Ein durch die Addition von 0.5 eventuell hervorgerufener Übertrag steht ja vor dem Dezimalpunkt und wird damit folglich nicht abgeschnitten. Danach wird die Zahl wieder durch 100 geteilt, damit sie wieder in ihre ursprüngliche Größenordnung kommt. Ja, und damit ist die Rundung beendet.
Eine solchermaßen mathematisch korrekt gerundete Zahl kann natürlich immer noch mit STR$ in einen String umgewandelt werden und dann mittels der Stringfunktionen auf ein bestimmtes Format gebracht werden. In MOONRAKER habe ich auf die Rundung verzichtet, da hier ja die völlig exakte Angabe der Höhe oder des Treibstoffvorrates auf zum Beispiel zwei Nachkommastellen sinnlos ist. Bei nicht-spielerischen Anwendungen ist das natürlich etwas anderes.
So, ich glaube, damit haben wir genug über Digital- und Analoganzeigen und alles, was damit zusammenhängt, geredet. Schauen wir uns noch einmal einen ganz anderen Teil von MOONRAKER an.
Am Spielanfang steigt eine Rakete von unten aus dem Bildschirm auf, überquert diesen und verschwindet sodann oben wieder. Diese Bewegung ist so einfach programmiert, daß ich Ihnen den dahintersteckenden Trick nicht vorenthalten möchte.
Wie Sie wissen, rollt der Bildschirm bei einem PRINT in die letzte Zeile automatisch eins nach oben. Man nennt diesen Vorgang "scrolling". Alles, was auf dem Schirm steht, rutscht also ebenfalls eins nach oben. Nun, genau so bewegt sich auch unsere Rakete.
Die einzelnen Teile des Raumschiffes sind in den Programmzeilen 4000 bis 4020 unter DATA abgelegt. Die vielen einzelnen Kommata in Zeile 4020 wirken als leere Strings, sobald sie mit READ eingelesen werden. Dieses Einlesen geschieht in Zeile 1010. Die einzelnen Teile der Rakete werden in einer FOR-NEXT-Schleife ('FORI=0TO38') nacheinander mittels READ in A$ eingelesen und anschließend mit PRINT in die unterste Bildschirmzeile ausgedruckt, wodurch der schon fertige Teil jeweils eins weiter hoch rutscht ("scrolling"). Auch wenn die Rakete vollständig zu sehen ist, werden weiterhin leere Strings eingelesen (die Kommata) und ausgedruckt. Dadurch bewegt sich das nun vollständige Raumschiff nach oben weiter, und zwar solange, bis

es am oberen Bildschirmrand herausgeschoben ist.
Diese einfache Programmierung einer Bewegung ist nur aufgrund des automatischen "scrolling" möglich. Selbst größere Objekte wie unsere Rakete können so leicht über den Bildschirm transportiert werden. Die Bewegungsrichtung ist allerdings festgelegt: von unten nach oben, denn das "scrolling" funktioniert eben auch nur in diese Richtung.
Damit sich das Raumschiff möglichst naturgetreu im Weltall bewegt, muß der Sternhimmel im Hintergrund unbeweglich sein. Durch den PRINT in die letzte Bildschirmzeile würden aber auch die Sterne nach oben verschoben. Genau dies wird in den Zeilen 1030 und 1040 verhindert. Hat sich ein Stern um eins nach oben verschoben (–22), so wird er an dieser Stelle gelöscht (Zeile 1030) und eine Position tiefer neu gesetzt (Zeile 1040). Die Positionen der einzelnen Sterne sind in S(J) abgelegt. Dies wird in Zeile 1000 für alle 20 Sterne so definiert. 42 ist der Bildschirmcode für einen Stern, 32 der für ein Leerzeichen. Durch diese ständige Korrektur der Sternpositionen bewegt sich das Raumschiff, während das Sternenmuster unverändert bleibt. So entsteht der perfekte Eindruck einer im Weltraum fliegenden Rakete.
Zum Abschluß wollen wir uns noch eine kleine Kuriosität anschauen. Listen sie sich dazu Zeile 3090 auf. Dort finden Sie bei einer Berechnung tatsächlich zwei Grundoperationszeichen direkt nebeneinander, nämlich '*' und '–'. Sie wissen, daß so etwas üblicherweise nicht erlaubt ist. So führt zum Beispiel die Zeile

```
X = 12+*2
```

zu einem SYNTAX ERROR. Klar, der Computer weiß nicht, soll er nun addieren oder multiplizieren. Doch schreiben wir jetzt die beiden Operationszeichen einmal umgekehrt:

```
X = 12*+2
```

Sehen Sie sich das Ergebnis mit '?X' an! Was ist passiert, wieso kann der Computer diese zweite – scheinbar ebenfalls falsche – Rechnung verarbeiten? Nun, die Antwort ist verblüffend einfach. Die Maschine versteht das Pluszeichen in diesem Zusammenhang als Vorzeichen der Zahl 2. Im ersten Fall ist dies natürlich nicht möglich, da das '+' nicht direkt vor der 2 steht. Ein Vorzeichen '*' aber gibt es nicht.

Ähnlich wird das Minuszeichen in Programmzeile 3090 in MOONRAKER als Vorzeichen zu 18 interpretiert. Eindeutig müßte dies eigentlich so geschrieben werden: '*(–18)'. Aber ohne Klammern geht es eben auch.
Damit wollen wir den Lernstoff für diesmal beiseite schieben. Besteigen Sie Ihr Raumschiff, brausen Sie los und – nehmen Sie sich eine mißglückte Landung nicht allzu sehr zu Herzen. Schließlich ist noch kein Astronaut vom Himmel gefallen! (Oder doch?)

Zusammenfassung:
Die Lerninhalte von Kapitel 12 sind:
- Programmierung von vertikaler und horizontaler Analoganzeige
- Stellenbegrenzung bei Digitalanzeigen
- Rundung von Zahlen auf beliebige Nachkommastellen
- Programmierung einer Bewegung auf dem Bildschirm mit ''scrolling''
- Unterscheidung von Vor- und Operationszeichen bei '+' und '–'.

```
1 REM ..............
2 REM .            .
3 REM . MOONRAKER .
4 REM ..............
5 REM
6 REM COPYRIGHT '82
7 REM
8 REM ANDREAS DRIPKE
9 :
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
1000 PRINTCHR$(142)"■■■■":POKE36879,232:DIMS(19):FORI=0TO19:S(I)=506*RND(1)
1010 PRINT:NEXT:POKE657,128:FORI=0TO38:READA$:PRINTSPC(8)A$:POKE36878,10
1020 POKE36876,245:FORJ=0TO49:NEXT:POKE36876,0:POKE36878,0:FORJ=0TO19
1030 IFPEEK(VB+S(J)-22)=46THENPOKEVB+S(J)-22,32
1040 IFPEEK(VB+S(J))=32THENPOKEVB+S(J),46:POKEVC+S(J),6
1050 NEXT:NEXT:PRINT"■■ M O O N R A K E R <■":PRINT"■SIE SIND MIT DER RA-
1060 PRINT"■KETE IN EINE UMLAUF-":PRINT"■BAHN DES MONDES VOR-
1070 PRINT"■GESTOSSEN. DIE MOND-":PRINT"■LANDEFAEHRE HAT ABGE-
1080 PRINT"■KOPPELT UND ES IST":PRINT"■JETZT IHRE AUFGABE,
1090 PRINT"■COMMANDER, DIESE SI-":PRINT"■CHER AUF DEM MOND ZU
1100 PRINT"■LANDEN!":GOSUB4100:PRINT"■■SIE BEDIENEN DIE
1110 PRINT"■RUECKSTRAHLDUESEN MIT":PRINT"■DEN TASTEN VON '0'
1120 PRINT"■BIS '9'. DER RUECK-":PRINT"■STRAHL ZEIGT GEGEN
1130 PRINT"■DIE MONDOBERFLAECHE":PRINT"■UND BREMST SOMIT DIE
1140 PRINT"■FAEHRE. BEDIENEN SIE":PRINT"■IHN SO, DASS SIE
1150 PRINT"■WEICH LANDEN (GE-":PRINT"■SCHWINDIGKEIT < 15).":GOSUB4100
1160 PRINT"■■■WENN SIE ABER ZUVIEL":PRINT"■BREMSEN, GEHT IHNEN
1170 PRINT"■DER TREIBSTOFF VOR-":PRINT"■ZEITIG AUS, UND SIE
```

```
1180 PRINT"■PRALLEN IM FREIEN":PRINT"■FALL AUF DIE MOND-
1190 PRINT"■OBERFLAECHE.":GOSUB4100
2000 T=1200:X0=25E3:V=-200:F=1:SG=1:S=0:KS=0:POKE36879,27:A$="|              |
2010 PRINT"■■_____":PRINT"■■GESCHW/HOEHE":PRINTA$:PRINTA$:PRINTA$
2020 PRINT"■TREIBST/ZEIT":PRINTA$:PRINTA$:PRINTA$:PRINT"■ BREMSSTOSS ":PRINTA$
2030 PRINT"| 0123456789 |":PRINTA$:PRINT"|_____|":GOSUB4300:PRINT"■";
2040 GOSUB6000
3000 IF(TI-TT>30)THENTT=TI:GOTO3020
3010 GOTO3000
3020 GETA$:IFA$="0"ORVAL(A$)>0THENS=VAL(A$)
3030 KS=KS+1:T=T-S:A=(1-S*.2)*5.4:X=V+A*.5:X0=X0+X:IFX0>0GOTO3050
3040 X=X-X0:F=0:X0=0
3050 Y=V*V-X*A*2:IFY>0GOTO3070
3060 Y=-Y:SG=-1
3070 V=SG*SQR(Y)*SGN(X):SG=1
3080 GOSUB5000:PRINT"■■■    ■■■■";:C=V:D=4:GOSUB4200:IFV>0THENPRINT"____";
3090 PRINT:PRINTCHR$((V>0)*-18)"■"C$"■ ";:IFX0=0THENPRINT" NULL":GOTO3110
3100 C=X0:D=5:GOSUB4200:PRINTC$
3110 PRINT"■■■■";:IFT<=0THENPRINT"LEER";:S=0:GOTO3130
3120 C=T:D=4:GOSUB4200:PRINTC$;
3130 C=KS:D=3:GOSUB4200:PRINT"    "C$:IFF=0GOTO7000
3140 IFT<=0GOTO3030
3150 GOTO3000
4000 DATA"   |  "," ■■ ■","  ■ ■  "," ■    "," ■USA"," ■    "," ■    "," ■    "," ■||
4010 DATA" ■||  "," ■||  "," ■||  "," ■/  \","  ■    "," ■    "," ■    "," ■┴┴
4020 DATA"/ ■ ■ \",,,,,,,,,,,,,,,,,,,,,,
4100 POKE198,0:WAIT198,1:GETG$:RETURN
4200 C$=RIGHT$("      "+MID$(STR$(C),2,D),D):RETURN
```

```
4300 PRINT"■":FORI=20TO0STEP-1:PRINTSPC(19)RIGHT$(STR$(I),2):NEXT
4310 DV=1E3:MA=20:RETURN
4400 PRINT"■■"SPC(19)" 2":GOSUB4410:PRINTSPC(19)" 1":DV=100:MA=20
4410 FORI=0TO8:PRINTSPC(19)" .":NEXT:RETURN
5000 PRINT"■■■
```

13. BREAKOUT

Bei BREAKOUT gilt es, durch das geschickte Steuern eines Balles Mauersteine aus einer Wand herauszuschlagen. Die genauen Spielregeln brauche ich Ihnen an dieser Stelle nicht zu erklären, sie stehen im Programm selbst. Hingegen wollen wir wiederum einzelne Aspekte der Programmierung kennenlernen.
Am Interessantesten an BREAKOUT ist wohl die Steuerung des Schlägers mit den beiden Cursorsteuerungstasten CRSR. Wenn Sie so etwas programmieren wollten, würden Sie vermutlich mit GET arbeiten. Nun hat GET aber grundsätzlich den Nachteil, daß das längere Niederdrücken einer Taste auf diese Weise nicht abgefragt werden kann. Machen Sie sich das an folgendem kurzen Programmabschnitt klar.

```
10 GET A$
20 PRINT A$
30 GOTO 10
```

Wenn Sie mit RUN starten, läuft der Bildschirm durch. Drücken Sie jetzt eine Taste, so wird das entsprechende Zeichen dargestellt. Aber auch wenn Sie die Taste längere Zeit niedergedrückt halten, wird das Zeichen nur ein einziges Mal ausgedruckt. Soll dies immer wieder geschehen, so müssen Sie fortlaufend die Taste erneut betätigen, d.h. loslassen und jedesmal von neuem drücken. Für BREAKOUT hieße das, daß Sie, um den Schläger um zum Beispiel fünf Positionen nach rechts zu bewegen, auch fünfmal hintereinander die entsprechende Taste betätigen müßten – was natürlich sehr unbequem wäre. Wenn Sie jetzt sagen: "Aber nein, bei BREAKOUT werden doch die Cursorsteuertasten CRSR benutzt und diese sind doch mit automatischer Wiederholfunktion versehen", dann haben Sie damit völlig recht. Wird eine der beiden Cursortasten CRSR für längere Zeit betätigt, so wirkt sich dies im Direktmodus so aus, daß der Cursor freilaufend in die entsprechende Richtung geht. Nun, diese Eigenschaft der CRSR-Tasten geht auch während des Programmlaufs nicht verloren. Es wäre folglich durchaus möglich, den Schläger in BREAKOUT mittels GET zu bewegen, da eben in diesem Fall die automatische Wiederholfunktion der Cursorsteuerungstasten zur Geltung kommt. Dies spräche für einen fortgeschrittenen Programmierstil. In Wirklichkeit ist die Schlägerbewegung aber in BREAKOUT völlig anders programmiert – noch raffinierter.

Um das zu verstehen, ist es wieder einmal nötig, sich das Innenleben des Computers etwas genauer anzuschauen. Geben Sie dazu das folgende Programm ein.

```
10 A=PEEK(203)
20 PRINT A
30 GOTO 10
```

Starten Sie mit RUN und drücken Sie nun ein paar beliebige Tasten. Sie werden feststellen, daß jeder Taste ein anderer Wert in Speicherzelle 203 zugewiesen ist. Und zwar steht dieser Wert solange in der Adresse 203, wie wir die entsprechende Taste wirklich gedrückt halten – im Gegensatz zu GET. Bei

```
PEEK(203)
```

handelt es sich also um eine Echtzeitabfrage. Es zählt nur, welche Taste exakt zu dem Augenblick gedrückt ist, wenn der PEEK-Befehl abgearbeitet wird. Ist gar keine Taste gedrückt zu diesem Zeitpunkt, so enthält Speicherzelle 203 den Wert 64. Im Unterschied zu GET haben die Tasten SHIFT, COMMODORE und CTRL keinen Einfluß auf den Wert in Speicherstelle 203. Es ist also gleichgültig, ob wir zum Beispiel "K" oder SHIFT und "K" betätigen. 'PEEK(203)' gibt in beiden Fällen denselben Wert 44 aus. Man nennt Speicherzelle 203 übrigens auch Tastaturmatrix-Adresse, da hierbei die Tastatur in Form einer Matrix abgefragt wird. Dies ist im Grunde nur eine technische Einzelheit, wir wollen deshalb nicht näher darauf eingehen.
Vielmehr soll uns interessieren, wie die Tastaturmatrix-Abfrage in BREAKOUT genutzt wird. Sehen Sie sich dazu die Programmzeilen 1010 und 1020 an. In Zeile 1010 wird der aktuelle Wert aus Speicherzelle 203 der Variablen J zugewiesen. Hat J den Wert 31, so ist die Taste "Cursor nach links" betätigt. In diesem Fall wird die Position des Schlägers (in der Variablen S) um eins nach links verschoben ('S=S–1'). Ist J dagegen 23, so entspricht dies der Taste "Cursor nach rechts", und die Schlägerposition wird um eins nach rechts gesetzt (Zeile 1020: 'S=S+1'). Da der Schläger länger als eine einzige Stelle ist, sind mehr als ein POKE-Befehl nötig.
Die Abfragen 'IFS>VB+485' (Zeile 1010) und 'IFS<VB+502' (Zeile 1020) verhindern, daß der Schläger links oder rechts aus dem Bildschirm hinauslaufen kann. Dies entspricht im Grunde der Eingrenzung von Bildschirmbewegungen, wie wir sie schon bei FLAK in Kapitel 7 kennengelernt haben.

Wenden wir uns anderen Dingen zu. Bestimmt ist Ihnen schon die FOR-Anweisung in Zeile 1010 aufgefallen, und vielleicht haben Sie sich auch schon darüber gewundert. Der einzelne Dezimalpunkt steht für die Zahl Null. Immer da, wo eine einzelne Null hingehört, darf stattdessen auch ein Dezimalpunkt ohne etwas davor oder dahinter stehen. Sicher werden Sie jetzt nach dem Sinn fragen. Daß es diesmal nicht darum geht, Bytes zu sparen, ist klar. Schließlich belegt sowohl das Nullzeichen "0" ein Byte als auch das Dezimalzeichen Punkt ".". Trotzdem bringt der Punkt einen Vorteil, und zwar eine Erhöhung der Arbeitsgeschwindigkeit. Der Computer vermag den einzelnen Punkt schneller zu erkennen und als Null zu interpretieren, als er Zahlen, also auch die Null, verarbeiten kann. Natürlich ist der Geschwindigkeitsunterschied sehr gering und macht sich im allgemeinen gar nicht bemerkbar. Innerhalb von Schleifen, die sehr oft wiederholt werden, mag sich das aber zu mehreren Sekunden aufaddieren. Und das führt dann zum Beispiel bei bewegten Graphiken oder Musikprogrammierung zu einem merklichen Unterschied. Sie finden den Dezimalpunkt als Ersatzzeichen für die Null in BREAKOUT auch in den Zeilen 1060, 1080, 3000, 3010, 3020, 4000, 4010, 5040 und 5060. Sehen Sie dort nach.
In den Programmzeilen 3000 und 3010 finden Sie die Anweisung

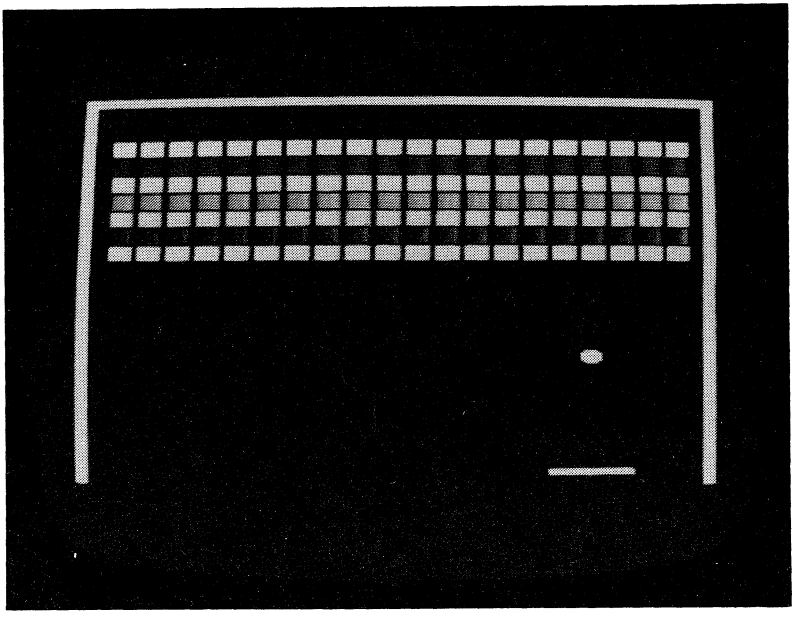
```
I=LOG(π)
```

Wenn Sie den Programmlauf verfolgen, werden Sie feststellen, daß der so errechnete Wert für die Variable I an keiner anderen Stelle im Programm mehr benötigt wird. Was hat es also mit der scheinbar sinnlosen Anweisung auf sich? Die Antwort mag Sie verblüffen: Der Befehl 'I=LOG(π)' dient ausschließlich dazu, die Programmausführung zu verlangsamen. Immerhin benötigt der Computer für die Abarbeitung der relativ komplizierten Berechnung ein paar Bruchteile einer Sekunde. Und bei Bildschirmbewegungen macht sich diese Zeit bemerkbar. Natürlich kann man zur Verzögerung auch eine leere Schleife der Art

```
FORI=1TO1:NEXTI
```

machen. Aber erstens ist das umständlicher und zweitens verbraucht es mehr Speicherplatz. In so einem Fall bietet sich also eine Verzögerungsfunktion wie 'I=LOG(π)' geradezu an. Sehen Sie sich die Funktion in BREAKOUT in den Zeilen 3000 und 3010 einmal genauer an. In beiden Programmzeilen wird

zunächst ein Ton eingeschaltet (224 bzw. 192), dieser Ton ist für kurze Zeit zu hören (eben durch unsere Verzögerungsfunktion), und schließlich wird der Ton wieder abgestellt (“.” als Null). Nehmen Sie doch einmal die Verzögerungsfunktion in beiden Zeilen heraus, und lassen Sie dann das Programm ablaufen. Merken Sie den Unterschied?



Wir wollen uns noch eine besondere Möglichkeit des INPUT-Befehls ansehen. Sie findet sich in BREAKOUT bei der Wahl der gewünschten Lautstärke in den Programmzeilen 5050 und 5060. Haben Sie schon einmal probiert, bei INPUT nichts einzugeben, also bloß die RETURN-Taste zu drücken? In so einem Fall fährt der Computer in der Programmbearbeitung fort. Die Variable, in die bei INPUT etwas eingegeben werden sollte, behält ihren ursprünglichen Inhalt oder Wert. Nehmen wir als Beispiel:

```
10 X=5
20 INPUT X
30 PRINT X
```

Starten Sie das Programm und drücken Sie bei INPUT in Zeile 20 RETURN, ohne etwas einzugeben. Dann wird in Zeile 30 der Wert 5 für X ausgedruckt. Und diesen Umstand macht sich

BREAKOUT in den genannten Programmzeilen 5050 und 5060 zunutze. Die Eingabevariable I$ wird in Zeile 5050 mit "15" vorbelegt. Wenn nun in Zeile 5060 bei 'INPUTI$' nichts eingetippt wird außer RETURN, so entspricht dies also der Maximallautstärke von 15.
Man nennt eine solche Eingabeform ganz allgemein Ersatzwert- oder auch Standardwerteingabe. Das heißt: Es kann eine beliebige Eingabe getätigt werden, die das Programm akzeptiert. Wird aber keine bestimmte Eingabe vorgenommen, so nimmt das Programm selbständig eine vordefinierte Ersatz- oder Standardeingabe. Als Standardwert wählt man die Eingabe, die sowieso am häufigsten gewünscht wird. Sie kann dann besonders einfach vorgenommen werden. In Ihren eigenen Programmen sollten Sie diese Möglichkeit des INPUT-Befehls ruhig einmal benutzen. Ihre Programme können dadurch nur besser oder genauer gesagt komfortabler werden.
In Programmzeile 5040 von BREAKOUT finden Sie einen BASIC-Befehl, der eigentlich recht selten verwendet wird. Ich meine die WAIT-Anweisung. Gedacht ist WAIT für Sonderfälle beim Arbeiten mit Peripheriegeräten, es läßt sich aber auch für andere Zwecke "mißbrauchen". Sehen wir uns die relevante Befehlsfolge an.

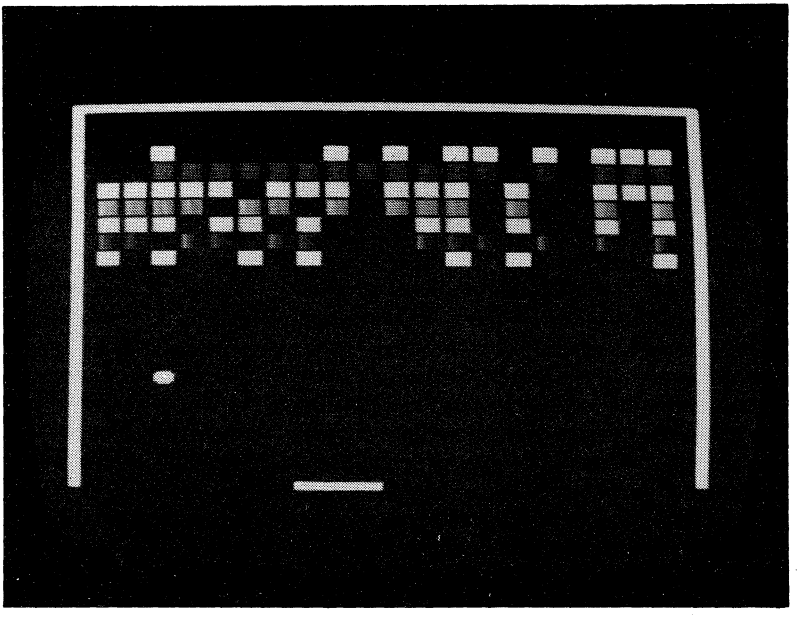
```
5040 POKE198,0:WAIT198,1:GETG$
```

Die Adresse 198 ist uns ja schon als Anzeichen dafür, wieviele Tastendrücke gültig sind, bekannt. Wenn Sie sich hierüber nicht mehr ganz im Klaren sind, schlagen Sie ruhig noch einmal in Kapitel 3 nach. Mit 'POKE198,0' wird festgelegt, daß die Anzahl der Tastendrücke Null ist, mit anderen Worten, daß keine Taste gedrückt wurde. 'WAIT198,1' hingegen wartet darauf, daß Sie irgendeine Taste betätigen. Welche Taste es war, kann mit einem anschließenden GET festgestellt werden, in unserem Fall 'GETG$'. Im allgemeinen wird man dies in der Form

```
5040 GETG$ : IF G$="" " THEN 5040
```

programmieren. In dieser herkömmlichen Programmierung wird für das Warten auf einen Tastendruck auf jeden Fall eine neue Programmzeile benötigt. Bei der Verwendung von WAIT dagegen ist dies nicht erforderlich. Eine Programmzeile, die auf das Drücken der RETURN-Taste wartet, könnte zum Beispiel wie folgt aussehen:

```
1000 POKE198,0:WAIT198,1:GETG$:IFG$<>CHR$(13)THEN1000
```



Ganz schön kurz, nicht wahr?
So, damit will ich Sie mit meinen Erklärungen nicht weiter vom Spielen abhalten. Es bleibt mir nur noch, Ihnen viel Spaß mit BREAKOUT zu wünschen.

Zusammenfassung:
In diesem Kapitel haben wir uns mit folgenden nützlichen Dingen beschäftigt:
- Echtzeitabfrage der Tastaturmatrix
- Benutzung des Dezimalpunktes anstelle der Null
- Verzögerungsfunktionen
- Vorgabe einer Standardeingabe bei INPUT
- WAIT-Anweisung zum Warten auf einen Tastendruck

```
1 REM"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
2 REM"▒▒▒▒BREAKOUT▒▒▒▒
3 REM"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
4 REM
5 REM (C)OPYRIGHT 82
6 REM
7 REM ANDREAS DRIPKE
8 REM 6200 WIESBADEN
9 REM
100 IFPEEK(158)OR191ANDSTTHENPRINT"▒?LOAD ERROR";:END
110 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
120 T=36874:H=T+2:N=T+3:L=T+4:C=T+5
1000 S=VB+495:B=VB+250:R=21:GOSUB5000:GOSUB4000
1010 FORI=.TO1:J=PEEK(203):IFJ=31THENIFS>VB+485THENS=S-1:POKES+3,32:POKES,98
1020 IFJ=23THENIFS<VB+502THENS=S+1:POKES-1,32:POKES+2,98
1030 NEXT:GOSUB2000:ON-(B<VB+484)GOTO1010:G=140:FORI=VBTOVB+242
1040 G=G+(PEEK(I)=250):NEXT:POKEC,110:PRINTCHR$(14)"▒▒=▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```

```
2060 GOSUB3000:IFRND(1)>.9THENR=-22:RETURN
2070 IFR=23THENR=-21:RETURN
2080 IFR=21THENR=-23:RETURN
2090 IFR=-22THENIFRND(1)>.5THENIFPEEK(B+1)=32THENPOKEB,32:POKEB+1,81:B=B+1
2100 IFR=22THENR=-21-2*(INT(2*RND(1)))
2110 IFR=-21THENR=23:RETURN
2120 IFR=-23THENR=21:RETURN
2130 IFR=-22THENR=21+INT(3*RND(1))
2140 RETURN
3000 POKEH,224:I=LOG(π):POKEH,.:RETURN
3010 POKET,192:I=LOG(π):POKET,.:RETURN
3020 FORI=255TO200STEP-1:POKEN,I:NEXT:POKEN,.:RETURN
4000 POKEC,8:PRINTCHR$(142)"■■■■■";:FORI=1TO7:POKE646,I:FORJ=.TO21:PRINT"■";
4010 NEXT:NEXT:FORI=.TO21:POKEVB+I,98:POKEVC+I,1:NEXT:FORI=22TO484STEP22
4020 POKEVB+I,97:POKEVC+I,1:POKEVB+I+21,225:POKEVC+I+21,1:NEXT:RETURN
5000 POKEC,75:PRINTCHR$(14)"■■ ____________________":PRINT"   ■B R E A K O U T■
5010 PRINT"■■VERSUCHEN SIE, MOEG-":PRINT"■LICHST VIELE MAUER-
5020 PRINT"■STEINE ABZUSCHIESSEN.":PRINT"■STEUERN SIE DEN
5030 PRINT"■SCHLAEGER MIT DEN":PRINT"■CURSORSTEUERTASTEN.■
5040 PRINT"■RECHTS: CRSR →":PRINT"■LINKS:  CRSR ↑":POKE198,.:WAIT198,1:GETG$
5050 PRINT"■WELCHE LAUTSTAERKE?":PRINT"■VON 0 BIS 15:":I$="15
5060 PRINT"■"SPC(14):INPUTI$:I=VAL(I$):ON-(I<.ORI>15)GOTO5060:POKEL,I:RETURN
```

14. "AUF LOS GEHT'S LOS"

Bei diesem Spiel geht es darum, bekannte deutsche Sprichworte zu raten. In das Programm wurden so viele Sprüche hineingeschrieben, wie in einem Computer mit 8 KByte Speicherplatz passen (also 3 KByte Zusatz zur Grundkonfiguration des VC 20). Sollten Sie mehr Speicherkapazität zur Verfügung haben, dann können Sie natürlich auch mehr Sprichworte unterbringen. Schreiben Sie in diesem Fall den Text in DATA-Zeilen ab der Zeilennummer 9210. Ändern Sie auch den Wert für die Anzahl der Sprichworte in Zeile 3020. Verantwortlich ist die Variable AZ. Für zum Beispiel 100 Sprichworte würden Sie diese Anweisung also in 'AZ=100' ändern. Andernfalls stehen die von Ihnen dazugeschriebenen Sprichworte zwar im Programm, bleiben aber trotzdem wirkungslos, da das Programm nicht "weiß", daß sie vorhanden sind.
Wir wollen uns ausführlich mit der Datenspeicherung in DATA-Anweisungen befassen. Diese Form der Datenspeicherung stellt einen Kompromiß dar zwischen der Speicherung auf einem externen Medium (Kassette, Diskette) und der Festlegung in Variablen. Auf extern gespeicherte Daten kann man von einem Programm aus nur sehr langsam zugreifen (speziell die Kassette hat eine extrem hohe Zugriffszeit), auf in Variablen gespeicherte Daten dagegen sehr schnell, wenn man den Variablennamen kennt.
Machen wir uns diesen Unterschied an einem Beispiel deutlich. Wir wollen zwei Daten ausdrucken. Sind sie in Variablen enthalten und wissen wir die Variablennamen – nehmen wir einmal an, es seien A und B – dann geben wir einfach

```
PRINT A,B
```

ein und schon erscheinen unsere Daten auf dem Bildschirm. Sind sie dagegen auf Kassette oder Diskette gespeichert, so müssen wir zunächst einen Datenkanal eröffnen mit zum Beispiel

```
OPEN1,1,0,"FILENAME"
```

oder

```
OPEN1,8,5,"FILENAME"
```

Der Ausdruck "File", der einem in diesem Zusammenhang oft

begegnet, ist übrigens nur ein anderes Wort für "Datei", also eben für Daten, die auf einem externen Speichergerät ausgelagert sind.
Nachdem nun der Datenkanal mit OPEN geöffnet wurde, müssen jetzt die Daten eingelesen werden. Dies geschieht mittels INPUT#.

```
INPUT#1,A : INPUT#1,B
```

Erst jetzt enthalten die Variablen A und B die gewünschten Daten (Werte), und diese können mit

```
PRINT A,B
```

sichtbar gemacht werden.
Die DATA-Anweisung stellt nun einen Kompromiß zwischen der ersten extrem schnellen und der zweiten recht langsamen Form der Abspeicherung von Daten dar. DATA-Anweisungen stehen zwar wie Variablen schon fest im Programm, aber um mit ihnen etwas anfangen zu können, muß man sie erst noch mittels READ einlesen.
Wir stellen uns DATA-Zeilen am besten als eine Liste vor, deren Elemente man nur nacheinander mit READ lesen kann. Der Nachteil gegenüber Variablen ist offensichtlich: um zum Beispiel auf das 100. Element zugreifen zu können, muß das Programm erst die 99 davorliegenden Elemente lesen. Und das kostet natürlich Zeit. Wir nennen diese Zugriffsart auf Daten ganz allgemein "sequentiell". Bei einer sequentiellen Datenstruktur können die einzelnen Datenelemente nur eines nach dem anderen erreicht werden. Dies ist die allereinfachste Möglichkeit der Anordnung von Daten.
Aber kommen wir zurück zu unseren DATA-Anweisungen. Wie sieht die Sache mit dem sequentiellen Zugriff in der Praxis aus? Nehmen wir einmal an, wir wollen innerhalb eines Programmes wirklich das 100. Datenelement einer DATA-Liste in der Variablen D$ zwischenspeichern (zum Beispiel unser hundertstes Sprichwort). Die Programmzeile dazu lautet folgendermaßen:

```
RESTORE:FORX=1TO100:READD$:NEXTX
```

RESTORE gewährleistet, daß wir beim Einlesen mit READ auch wirklich mit dem ersten Datenelement der DATA-Liste anfangen. Man sagt auch, der DATA-Zeiger (oder "DATA-Pointer" auf neudeutsch) wird auf das erste Datenelement gesetzt. Dann werden die ersten 100 Daten in der FOR-NEXT-



Schleife eingelesen, und zwar in die Variable D$. Weil nun aber immer in dieselbe Variable eingelesen wird, überschreiben sich die Daten gegenseitig, und wenn das Programm die Schleife verläßt, dann steht in D$ das zuletzt gelesene Datenelement, im Beispiel eben das hundertste.
In "AUF LOS GEHT'S LOS" ist die Programmzeile 4010 dafür verantwortlich, welches unter den gespeicherten Sprichworten zufälligerweise ausgesucht wird. Mit RESTORE wird – wie uns bekannt – der DATA-Zeiger auf das erste Datenelement gesetzt und danach in der FOR-NEXT-Schleife, deren Endwert mit der Zufallszahlenfunktion RND bestimmt wird, ein Sprichwort mit READ in die Variable W$ eingelesen. Es ist also alles genau so, wie wir es besprochen haben.
Es ist schon viel von Datenelementen die Rede gewesen. Wir wollen uns jetzt klar darüber werden, was das eigentlich genau ist. Sicher haben Sie schon gemerkt, daß in unserem Programm jedes Sprichwort ein Datenelement darstellt. Die Gesamtzahl aller Sprichworte bildet eine DATA-Liste (auch DATA-Block genannt). Der Platz einer DATA-Anweisung innerhalb der DATA-Liste wird durch die Zeilennummer, unter der die DATA-Anweisung abgelegt ist, bestimmt. Je höher die Zeilennummer ist, desto weiter hinten in der Liste stehen die in der Zeilen gespeicherten Datenelemente. Das ist eigentlich auch ganz logisch, nicht wahr? Wenn in einer DATA-Anwei-

sung mehrere Datenelemente stehen, dann müssen diese durch Kommata voneinander getrennt werden. Und wenn ein Datenelement selbst ein Komma enthält? Nun, dann müssen Sie ein solches Datenelement in Anführungszeichen setzen. Ein Beispiel dazu:

```
100 DATA "TRAUTES HEIM, GLUECK ALLEIN"
```

Obwohl in dieser DATA-Anweisung ein Komma steht, beinhaltet sie dennoch nur ein einziges Datenelement, da das Komma in Anführungszeichen steht.
Zeichen, die nur über eine der beiden Umschaltmöglichkeiten mit SHIFT oder mit der COMMODORE-Taste erzeugt werden können, wollen ebenfalls in Anführungszeichen geschrieben werden. Dies ergibt sich üblicherweise bei Groß- und Kleinschreibung sowie bei der Verwendung von graphischen Zeichen. Werden diese nicht in Anführungszeichen gesetzt, so löscht der Computer sie selbständig wieder aus. Sie sehen das sehr schnell, wenn sie einmal auf den Groß-/Kleinschreibmodus umschalten und dann

```
1 data Hallo
```

eingeben. Das "H" muß mit SHIFT eingetippt werden. Geben Sie nun 'LIST 1' ein. Was passiert?
Zahlen in DATA werden grundsätzlich nicht in Anführungszeichen gesetzt, auch dann nicht, wenn sie einen Dezimalpunkt enthalten. Allerdings dürfen Sie nicht den Fehler machen, statt des Punktes als (angelsächsisches) Dezimalzeichen ein Komma – was ja unserer Norm entspräche – zu verwenden. Während diese Zeile

```
100 DATA 1.2
```

nur ein Datenelement mit dem Wert 1.2 enthält, sind hier

```
100 DATA 1,2
```

die zwei Datenelemente 1 und 2 vorhanden.
Beachten Sie in diesem Zusammenhang auch, daß das Komma nur zur Trennung der einzelnen Datenelemente dient. Das bedeutet insbesondere, daß eine DATA-Anweisung nicht mit einem Komma abgeschlossen werden darf.

Zum Abschluß unserer Betrachtungen über DATA, READ und RESTORE wollen wir noch einen weit verbreiteten Irrtum aufklären. Entgegen anderslautenden Meinungen ist es völlig egal, wo innerhalb eines Programmes die DATA-Anweisungen stehen. Entscheidend ist lediglich die Reihenfolge. Es ist also nicht notwendig, daß das Programm zunächst über die DATA-Zeilen laufen muß, damit die darin stehenden DATA-Elemente später mittels READ eingelesen werden können. Den besten Beweis dafür bietet "AUF LOS GEHT'S LOS" selbst, dort stehen die DATAs nämlich ganz am Schluß. Ebenfalls falsch ist die Meinung, daß in einer Programmzeile, die DATA-Anweisungen enthält, keine anderen BASIC-Befehle stehen dürfen. Zur Abtrennung der DATA-Anweisung ist lediglich, wie zu jedem anderen Befehl auch, ein Doppelpunkt nötig. Ein Beispiel für eine solche Programmzeile:

```
100 PRINT:DATA1,2,3,4,5,6,7,8,9:PRINT"HALLO"
```



Soll der Doppelpunkt Inhalt eines Datenelemetes sein, so muß er zur Unterscheidung in Anführungszeichen eingeschlossen werden.
So, damit haben wir alles Wesentliche bezüglich der DATA-Anweisung besprochen. Schauen Sie sich "AUF LOS GEHT'S LOS" noch einmal auf das Gesagte hin an. Und auch hier gilt:

Probieren schadet nicht!
Wir wollen uns jetzt noch einer anderen Sache zuwenden, über die sie ebenfalls unbedingt Bescheid wissen sollten – die Dimensionierung von Variablenfeldern. Sehen Sie sich dazu Zeile 3020 im Programm an. Sie finden dort den Befehl

```
3020 DIM N$(G),F(G)
```

Die Variable G enthält dabei die Anzahl der Mitspieler. Hier werden zwei eindimensionale Variablenfelder N$ und F festgelegt. Beachten Sie, daß zum Beispiel der Befehl

```
DIM N$(6)
```

nicht nur eine einzige Variable festsetzt, sondern sieben: N$(0), N$(1), N$(2), N$(3), N$(4), N$(5) und N$(6). Daher rührt die Bezeichnung "Variablenfeld" (engl. "array"). Da alle Variablen eines solchen Feldes einen Index in Klammern dazugesetzt bekommen, spricht man auch von "indizierten Variablen". Das oben definierte Feld N$ hat eine einzige Dimension, es ist eindimensional. Wir stellen uns dies am einfachsten mit folgendem Schema vor:

N$	(0)	(1)	(2)	(3)	(4)	(5)	(6)

Es gibt auch zweidimensionale Variablenfelder. Ein Beispiel dafür ist

```
DIM N$(2,4)
```

Damit stehen zur Verfügung: N$(0,0), N$(0,1)...N$(0,4) und N$(1,0)...N$(1,4) und N$(2,0)...N$(2,4). Wenn wir uns dieses zweidimensionale Feld in einem Schema klarmachen, dann sieht dies wie folgt aus:

N$	(0)	(1)	(2)	(3)	(4)
(0)					
(1)					
(2)					

Da man bei einem zweidimensionalen Variablenfeld leicht von Zeilen und Spalten sprechen kann, nennt man es auch Matrix. Dies entspricht der mathematischen Definition einer Matrix.
Wir können auch mehrdimensionale Variablenfelder festlegen, so zum Beispiel das folgende fünfdimensionale Feld:

```
DIM X(2,3,2,2,10)
```

Dadurch werden 3*4*3*3*11 gleich 1188 einzelne indizierte Variablen erzeugt, wozu immerhin über 8000 Byte Speicherkapazität erforderlich sind. Während man ein dreidimensionales Feld noch räumlich darstellen kann, ist dies bei einem fünfdimensionalen leider nicht mehr gegeben. Da nützt uns selbst die Zeit – in der Akrobatik der modernen Physik gerne als vierte Dimension bezeichnet – nichts. Oder können sie sich einen vier- oder fünfdimensionalen Raum vorstellen?
Wenn in einem Programm eine indizierte Variable benutzt wird, nehmen wir als Beispiel einmal A(4), und vorher kein entsprechender DIM-Befehl zur Abarbeitung kam, dann reserviert der Computer automatisch 11 Einzelvariablen pro Dimension. Dies entspricht in unserem angenommenen Fall 'DIM A(10)', da durch A(4) nur eine Dimension vorgegeben wird. Bei A(4,4) entstünde ein Feld 'DIM A(10,10)', also zweidimensional. Sollten Sie aber nur einen Index von zum Beispiel 4 benötigen, dann hat der Computer 6 indizierte Variablen mehr reserviert als Sie brauchen. Das wiederum kostet Speicherplatz, wie die obige Rechnung gezeigt hat, sogar eine ganze Menge.
Wir wollen uns deshalb grundsätzlich angewöhnen, indizierte Variablen in einem Programm mit DIM zu dimensionieren, bevor wir sie das erste Mal benutzen. Unbedingt notwendig ist das sowieso, wenn wir weiter als 10 in einer Dimension gehen wollen (andernfalls erscheint ein BAD SUBSCRIPT ERROR). Wenn wir aber unter 10 bleiben, dann spart uns die Dimensionierung mit DIM Speicherkapazität.
Beachten Sie übrigens, daß ein einmal dimensioniertes Variablenfeld nicht mehr neu dimensioniert werden darf. So führt zum Beispiel dieses kurze Programm

```
10 A(3)=25
20 DIM A(5)
```

zu einem REDIM'D ARRAY ERROR. In Zeile 10 wird durch die Benutzung von A(3) automatisch ein Feld 'DIM A(10)' festgelegt, das dann in Zeile 20 nicht mehr nachdimensioniert werden darf. Das Löschen eines einmal festgelegten Feldes ist

nicht möglich, es sei denn durch CLR, wodurch alle Variablen gelöscht werden oder NEW, wodurch zusätzlich noch das Programm gelöscht wird.
So, nun haben Sie aber wirklich genug gelernt. Sie sollten sich jetzt die Zeit nehmen und in Ruhe das Programm eintippen, und dann – rätseln Sie los!

Zusammenfassung:
Dieses Kapitel sollte Ihnen außer einem weiteren Spiel in Ihrer Sammlung noch folgendes beigebracht haben:
- Datenspeicherung in DATA-Anweisung und Einlesen mit READ
- Dimensionierung von Variablenfeldern mittels DIM

```
1 REM     AUF LOS
2 REM    GEHTS LOS
3 REM
4 REM (C) FEBRUAR 82
5 REM
6 REM ANDREAS DRIPKE
7 :
100 S=36876:L=S+2:C=S+3:IFPEEK(158)OR191ANDSTTHENPRINT"■?LOAD ERROR";:END
1000 GOSUB2000:REM ANLEITUNG
1010 GOSUB3000:REM SPIELVORBEREITUNG
1020 GOSUB4000:REM SPIEL
1030 GOSUB5000:REM NEUES SPIEL? - ENDE
1040 IFG$="J"GOTO1020
1050 POKE657,0:END
2000 POKEC,25:PRINT"■■■■  ■ ■ ■  ":PRINT"■ ■ ■ ■ ■ ":PRINT"■  ■ ■ ■  "
2010 PRINT"■ ■ ■ ■ ■ ":PRINT"■ ■ ■  ■ ■■":PRINTSPC(4)"■ ■■■  ■  "
2020 PRINTSPC(4)"■ ■■■ ■ ■ ":PRINTSPC(4)"■ ■■■ ■ ■  ":PRINT"■■■■■ ■■■ ■ ■■■ "
2030 PRINTSPC(4)"■  ■  ■  ■■":PRINT"■  ■  ■ ■ ■  ■ ■  "
2040 PRINT"■ ■■■ ■■■ ■ ■■ ■■ ■ ":PRINT"■ ■■■  ■■  ■■ ■■■■  "
2050 PRINT"■ ■ ■ ■■■ ■ ■■ ■■■■■■ ":PRINT"■  ■  ■ ■ ■■ ■■■■  ■
2060 PRINTSPC(8)"■ ■■■  ■  ":PRINTSPC(8)"■ ■■■ ■ ■ "
2070 PRINTSPC(8)"■ ■■■ ■ ■  ":PRINTSPC(8)"■ ■■■ ■ ■■■ "
2080 PRINTSPC(8)"■  ■  ■  ■■":FORI=1TO4E3:NEXT:POKEC,58:POKE657,128
2090 PRINTCHR$(14)"■■■■■■■  AUF LOS GEHT'S LOS■":PRINT"■KENNEN SIE DIE SPIEL-
2100 PRINT"■REGELN (J/N)?";POKE198,0
2110 GETG$:ON-(G$="N")GOTO2120:ON-(G$<>"J")GOTO2110:RETURN
2120 PRINT"■VERSUCHEN SIE, EIN":PRINT"■BEKANNTES SPRICHWORT
2130 PRINT"■ZU ERRATEN. SIE GEBEN":PRINT"■BUCHSTABEN EIN, VON
```

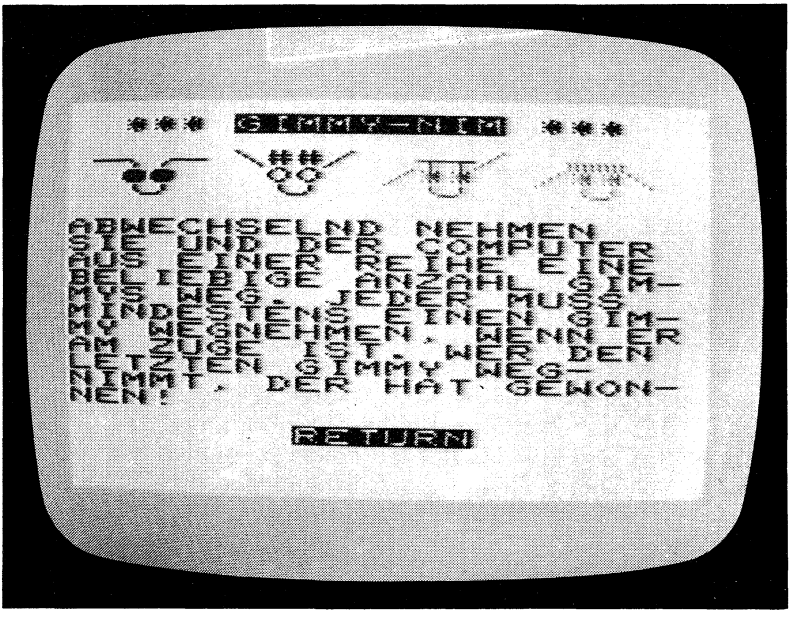
```
2140 PRINT"■DENEN SIE GLAUBEN,":PRINT"■DASS SIE IM SPRICH-
2150 PRINT"■WORT VORKOMMEN.":GOSUB8000:PRINT"■IST DIES DER FALL, SO
2160 PRINT"■WIRD DER BUCHSTABE IN":PRINT"■DIE LUECKE EINGETRA-
2170 PRINT"■GEN, ANDERNFALLS ER-":PRINT"■HALTEN SIE EINEN FEH-
2180 PRINT"■LERPUNKT.":PRINT"■SIEGER WIRD DERJENIGE":PRINT"■MIT DEN WENIGSTEN
2190 PRINT"■FEHLERN.":GOTO8000
3000 PRINT"■■WIEVIELE SPIELER":PRINT"■SPIELEN MIT (1-5) ";
3010 GOSUB7500:G=VAL(G$):IFG<1ORG>5GOTO3010
3020 PRINTG$:AZ=39:DIMN$(G),F(G):PRINT"■■BITTE GEBEN SIE NUN
3030 PRINT"■DIE NAMEN DER SPIELER":PRINT"■EIN.■":FORI=1TOG:PRINT
3040 PRINTMID$(STR$(I),2)". ";:GOSUB7000:N$(I)=I$:NEXT:PRINT"■■■■VIELEN DANK.
3050 PRINT"■■GLEICH GEHT'S LOS!"
4000 FORI=1TOG:FORJ=1TO3E3:NEXT:N$=N$(I):PRINTCHR$(142)"■■■■■■■■"N$" IST DRAN.
4010 FORJ=1TO3E3:NEXT:S$="":RESTORE:FORJJ=0TO(1+AZ)*RND(1):READW$:NEXTJJ
4020 FORJ=1TOLEN(W$):IFMID$(W$,J,1)=" "THENS$=S$+" ":GOTO4040
4030 S$=S$+".
4040 NEXT:A=11-LEN(N$)/2:PRINT"■"SPC(A)LEFT$("__________",LEN(N$))
4050 PRINTSPC(A)"■"N$:PRINT"■■SPRICHWORT GESUCHT!
4060 GOSUB6000:PRINT"■■WELCHER BUCHSTABE MAG":PRINT"■VORKOMMEN ";
4070 F=1:GOSUB7500:IFG$<"A"ORG$>"Z"GOTO4070
4080 FORJ=1TOLEN(W$):IFMID$(W$,J,1)=G$THENS$=LEFT$(S$,J-1)+G$+MID$(S$,J+1):F=0
4090 NEXT:F(I)=F(I)+F:PRINT:PRINT"■■FEHLERPUNKTE:"F(I):PRINT,"■■■■■";:POKEL,7
4100 IFFTHENPRINT"FALSCH":FORJ=254TO192STEP-.2:GOTO4120
4110 PRINT"RICHTIG":FORJ=192TO254STEP.5
4120 POKES,J:NEXT:POKES,0:POKEL,0:PRINTSPC(10)"■        ":IFS$<>W$GOTO4060
4130 GOSUB6000:PRINT,"■■■■__________":PRINT,"■■■GESCHAFFT":FORJ=1TO2E3:NEXT
4140 NEXT:RETURN
5000 FORI=1TO3E3:NEXT:PRINT"■UEBERSICHT:■":FORI=1TOG:PRINT
```

```
5010 PRINTN$(I)TAB(15)F(I):NEXT:PRINT"■■NOCH EINE RUNDE?":POKE198,0
5020 GETG$:IFG$="J"THENRETURN
5030 IFG$<>"N"GOTO5020
5040 PRINT"□"SPC(6)"SIEGER BEI":PRINT"■■■AUF LOS GEHT'S LOS
5050 PRINTSPC(8)"■■WURDE:■■":N=1E38:FORI=1TOG:IFF(I)<NTHENN=I
5060 NEXT:A=11-LEN(N$(N))/2:PRINTSPC(A)LEFT$("__________",LEN(N$))
5070 PRINTSPC(A)"■"N$(N):PRINT"■■■MIT"F(N)"FEHLERPUNKTEN.■":RETURN
6000 PRINT"■■■■■■":N=1
6010 PRINT:Z=23
6020 Z=Z-1:A$=MID$(S$,Z+N-1,1):IFA$<>""ANDA$<>" "GOTO6020
6030 PRINTMID$(S$,N,Z-1):IFA$=" "THENN=N+Z:GOTO6010
6040 RETURN
7000 I$=""
7010 GOSUB7500:IFG$=CHR$(13)ANDI$<>""THENPRINT" ":RETURN
7020 IFG$=CHR$(20)ANDI$<>""THENI$=LEFT$(I$,LEN(I$)-1):PRINTG$;:GOTO7010
7030 ON-(LEN(I$)=10)GOTO7010:IFG$>"@"ANDG$<"["THENI$=I$+G$:PRINTG$;
7040 GOTO7010
7500 P=POS(0):POKE198,0:POKEL,3
7510 PRINT:B=B+1AND1:PRINT"□"SPC(P)MID$("■?■?",B*2+1,2);:POKES,7*B+128
7520 FORJ=0TO49:GETG$:IFG$THENPRINT"■■";:POKES,0:POKEL,0:RETURN
7530 NEXT:GOTO7510
8000 POKE198,0:PRINT"■■■
8010 B=B+1AND1:PRINT"□"SPC(8)CHR$(18+B*128)"RETURN":POKEL,15:POKES,192:T=TI
8020 ON-(T+10>TI)GOTO8020:POKES,0:POKEL,0
8030 ON-(T+30>TI)GOTO8030:GETG$:ON-(G$<>CHR$(13))GOTO8010:RETURN
9000 DATAWER DEN SCHADEN HAT BRAUCHT FUER DEN SPOTT NICHT ZU SORGEN
9010 DATADEM DUEMMSTEN BAUERN WACHSEN DIE GROESSTEN KARTOFFELN
9020 DATAWER ANDERN EINE GRUBE GRAEBT FAELLT SELBST HINEIN,PAPIER IST GEDULDIG
```

```
9030 DATADEM GESCHENKTEN GAUL SCHAUT MAN NICHT INS MAUL,IRREN IST MENSCHLICH
9040 DATADER KLUEGERE GIBT NACH DER ESEL FAELLT IN DEN BACH,DER SCHEIN TRUEGT
9050 DATAMESSER GABEL SCHERE UND LICHT SIND FUER KLEINE KINDER NICHT
9060 DATAKLEINE GESCHENKE ERHALTEN DIE FREUNDSCHAFT,HOCHMUT KOMMT VOR DEM FALL
9070 DATAWENN ZWEI SICH STREITEN FREUT SICH DER DRITTE,NOT MACHT ERFINDERISCH
9080 DATAES IST NOCH KEIN MEISTER VOM HIMMEL GEFALLEN,UEBUNG MACHT DEN MEISTER
9090 DATAEIN BLINDES HUHN FINDET AUCH EIN KORN,FRISCH GEWAGT IST HALB GEWONNEN
9100 DATAERST DIE ARBEIT DANN DAS VERGNUEGEN,WO EIN WILLE IST IST AUCH EIN WEG
9110 DATAES IST NICHT ALLES GOLD WAS GLAENZT,WER ZULETZT LACHT LACHT AM BESTEN
9120 DATAIN DER NOT FRISST DER TEUFEL FLIEGEN,EIN UNGLUECK KOMMT SELTEN ALLEIN
9130 DATAWO KEIN KLAEGER IST IST AUCH KEIN RICHTER,WIE GEWONNEN SO ZERRONNEN
9140 DATAEINE HAND WAESCHT DIE ANDERE,WER NICHT HOEREN WILL MUSS FUEHLEN
9150 DATASCHUSTER BLEIB BEI DEINEN LEISTEN,VIELE KOECHE VERDERBEN DEN BREI
9160 DATAIN DER KUERZE LIEGT DIE WUERZE,WAS SICH LIEBT DAS NECKT SICH
9170 DATAMORGENSTUND HAT GOLD IM MUND,WER ZUERST KOMMT MAHLT ZUERST
9180 DATAGOLD UND SILBER LIEB ICH SEHR,LUEGEN HABEN KURZE BEINE
9190 DATAUNDANK IST DER WELT LOHN,WIE DU MIR SO ICH DIR,UNKRAUT VERDIRBT NICHT
9200 DATAOHNE FLEISS KEIN PREIS,UNVERHOFFT KOMMT OFT
```

15. GIMMY-NIMM

In diesem Kapitel möchte ich Ihnen das bekannte Nimm-Spiel in einer Variante mit lustigen Gimmys vorstellen. Nimm gehört zur Gruppe der "Wegnehmspiele", und es wird von zwei Personen (Sie und – pardon – der Computer) gespielt.
Ausgangsposition ist eine beliebige Anzahl von Reihen, die mit einer beliebigen Anzahl von Spielsteinen belegt sind. Bei GIMMY-NIMM sind es 5 Reihen mit bis zu 4 Steinen. Wenn ein Spieler am Zuge ist, kann er so viele Spielsteine wegnehmen, wie er will, vorausgesetzt, sie liegen alle in einer Reihe. Wer den letzten Spielstein wegnimmt, hat gewonnen.



Kein Zweifel, damit der Computer das Nimm-Spiel erfolgreich kann, muß er in irgendeiner Form "denken". In RECTUS haben wir ja schon ein in dieser Beziehung ähnliches Spiel kennengelernt, und wir wollen uns jetzt einmal anschauen, was hinter dieser künstlichen Intelligenz bei GIMMY-NIMM steckt. Sie sollten diesmal das Programm schon jetzt, am Anfang des Kapitels eintippen. Dann bekommen Sie erstens schon einen guten Eindruck von der Intelligenz Ihres Computers, und zwei-

tens sind die nachfolgenden Erklärungen dann leichter für Sie zu verstehen.
Für Nimm gibt es eine vollendete Strategie. GIMMY-NIMM ist genau nach dieser Strategie programmiert. Das bewirkt, daß das Programm von vornherein optimal spielt – im Unterschied zu RECTUS, wo der Computer erst mit jedem Zug dazugelernt hat. Wie funktioniert nun GIMMY-NIMM. Wir stellen uns dazu einmal folgende Spielsituation vor:

1. Reihe G G G G G G G
2. Reihe G G G G G
3. Reihe G G G
4. Reihe G

G = Gimmy

Um die Strategie verstehen zu können, müssen wir einen kurzen Ausflug in die Zahlensystemtheorie, einen Zweig der Mathematik, machen. Aber keine Angst, es wird nicht zu kompliziert. Sie wissen, daß wir normalerweise im Zehnersystem rechnen. Das bedeutet nichts anderes, als daß wir uns zehn verschiedener Zeichen bedienen, um eine Zahl darzustellen. Gemeint sind natürlich die zehn Ziffern von 0 bis 9. Die Zahl 537 zum Beispiel kann man im Zehnersystem auch so schreiben:

$5*10^2+3*10^1+7*10^0$

Wir können jede Zahl im Zehnersystem in solche Zehnerpotenzen umschreiben. Nun gibt es aber noch unendlich viele andere Zahlensysteme. Von weitreichender Bedeutung in der gesamten Computerei ist das Hexadezimalsystem (16er System). Da wir nur zehn Ziffern überhaupt zur Verfügung haben, nimmt man für die restlichen sechs erforderlichen Zeichen die Buchstaben A bis F. Uns soll aber hier besonders das Zweiersystem interessieren. Genauso, wie im Zehnersystem nur die Ziffern von 0 bis 9 existieren, so gibt es im Zweiersystem nur die Ziffern 0 und 1. Deswegen nennt man das Zweiersystem auch Binärsystem (binär meint "zweier Zustände fähig"). Wenn Sie das daher an den Begriff "Bit" erinnert, dann liegen Sie völlig richtig. Das, was von der elektronischen Seite her eine Glühlampe mit den zwei möglichen Zuständen "ein" und "aus" ist, das ist von der mathematischen Seite gesehen eine Stelle des Zweiersystems. Die Zahl 1001 im Binärsystem bedeutet also

$1*2^3+0*2^2+0*2^1+1*2^0$

gleich 9 im Zehnersystem.
Ich habe Ihnen die Zahlen von 0 bis 9 einmal in beiden Systemen gegenübergestellt:

Zehnersystem	Binärsystem
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Was hat das nun mit dem Nimm-Spiel zu tun? Wir skizzieren uns dazu noch einmal die Spielsituation und schreiben daneben die Anzahl der Spielsteine (Gimmys) pro Zeile binär auf. Das sieht dann so aus:

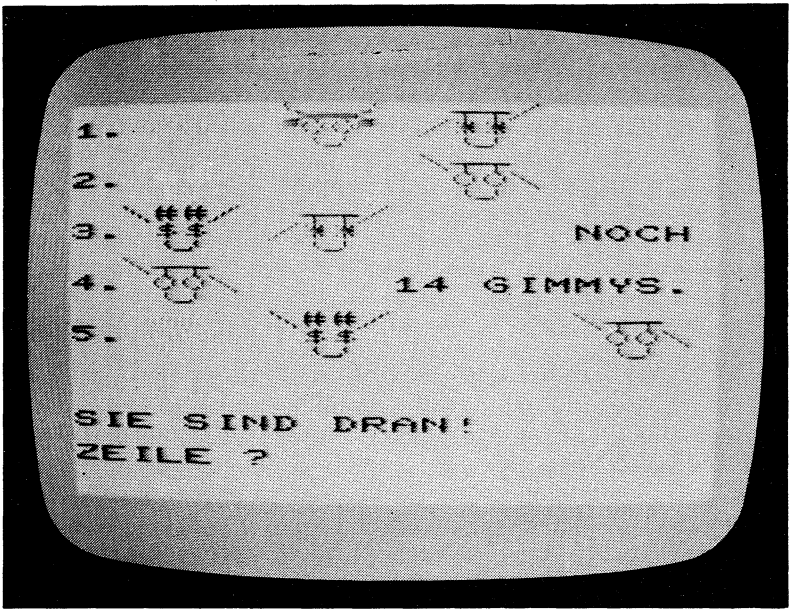
1. Reihe	G G G G G G G	= 7	= 0111
2. Reihe	G G G G G	= 5	= 0101
3. Reihe	G G G	= 3	= 0011
4. Reihe	G	= 1	= 0001

Jetzt addieren wir noch die Binärzahlen spaltenweise:

```
0 1 1 1
0 1 0 1
0 0 1 1
0 0 0 1
---------
0 2 2 4
```

Für Nimm gilt nun folgende Strategie. Wenn die Summe der Binärzahlen in jeder Spalte eine gerade Zahl oder Null ist, wie in diesem Fall, so wird die Situation als sicher bezeichnet. Das bedeutet, daß der Spieler, der jetzt am Zuge ist, egal wieviele Spielsteine er wegnimmt, immer mindestens eine ungerade Spalte hinterläßt. Der zweite Spieler kann beim nächsten Zug dies wieder in eine sichere Anordnung mit geraden Spalten umwandeln. Und wenn der zweite Spieler so spielt, daß er nach jedem Zug immer eine sichere Situation zurückläßt, so wird er gewinnen. Und genau das tut Ihr Computer, wenn Sie mit ihm GIMMY-NIMM spielen. Er simuliert also Intelligenz, in-

dem er diese eigentlich doch recht simple Strategie konsequent spielt. Aber wer das "Geheimnis" nicht kennt, der hat tatsächlich eine denkende Maschine vor sich.



Wir wollen uns jetzt noch genauer ansehen, wie die genannte Gewinnstrategie auf dem Computer programmiert ist. Die "Denkroutine" liegt in den Zeilennummern von 5000 bis 5400. Das Programm druckt den Text "ICH DENKE" aus (Zeile 5000) und legt fest, daß das "Denken" in der ersten Gimmy-Zeile beginnt (Programmzeile 5010: Y=1). Ist die erste Gimmy-Zeile leer (FL(Y)=0), so wird zu Zeile 5050 gesprungen und dort die zweite Gimmy-Zeile untersucht (Y=2). Ist die erste Gimmy-Zeile nicht leer, so nimmt das Programm dort einen Gimmy weg (Zeile 5020:FL(Y)=FL(Y)–1), rechnet dann die Binärzahlen pro Zeile aus und addiert diese spaltenweise, wie es der Gewinnstrategie entspricht (GOSUB 5290). Wenn die Summe in allen drei Spalten gerade ist, sind die Variablen C, D und E gleich 1, so hat das Programm die Lösung gefunden. Andernfalls nimmt es einen weiteren Gimmy weg (Zeile 5030), sofern in dieser Gimmy-Zeile noch einer vorhanden ist (FL(1)>0) und probiert von neuem (GOTO 5020). Wenn die erste Gimmy-Zeile keinen Erfolg brachte, d.h. wenn nie alle drei Binärspalten zugleich gerade oder Null wurden, dann fügt das Programm die entnommenen Gimmys wieder hinzu (Zeile

5040: FL(1)=FL(1)+X) und versucht sein Glück in der zweiten Zeile (Programmzeilen 5050 bis 5080). In einer der vier Zeilen klappt es bestimmt.
Die Abfrage, ob die Summe der drei Binärspalten gerade oder ungerade ist, erfolgt in den Zeilen 5290 bis 5360. Erinnern Sie sich noch, wie die Funktion INT dazu benutzt werden kann, um festzustellen, ob eine Zahl gerade oder ungerade ist? Da vor dieser Abfrage die Binärsummen gebildet werden müssen, wird dies in Zeile 5300 vorgenommen. Dabei stehen die Gimmy-Zeilen in FL(1 . . 5) und die Binärsummen in FS(1 . . 3).
In den Programmzeilen 3200 bis 3310 erfolgt die Umwandlung der im Zehnersystem gegebenen Zahlen (in A) in Binärzahlen (A$). Der hin und wieder in der gesamten "Denkroutine" vorkommende Befehl 'GOSUB 6000' dient lediglich der Darstellung eines Gimmys auf dem Bildschirm. Dadurch erscheint für Sie als Spieler die Zeit, die das Programm benötigt, um seinen Zug zu finden, nicht zu lange.
Sie haben nun einmal an einem Beispiel gesehen, wie man einem Computer tatsächlich ein intelligentes Verhalten anprogrammieren kann. Beim Nimm-Spiel ist das deswegen so einfach, weil auch die Gewinnstrategie noch relativ einfach ist. Ganz anders sieht das natürlich bei komplexen Spielen wie zum Beispiel Schach aus. Hier läßt sich überhaupt keine eindeutige Gewinnstrategie formulieren. Dies macht es auch so schwierig, ein Programm dafür zu erstellen. Die heutigen Schachcomputer funktionieren deswegen auch nach einem völlig anderen Prinzip als GIMMY-NIMM (natürlich abgesehen davon, daß die Regeln für Schach selbstverständlich anders als für Nimm sind).
An GIMMY-NIMM können wir noch etwas ganz anderes lernen. Wenn sie sich die REM-Zeilen von 0 bis 9 anschauen, dann stellen Sie fest, daß sie durchweg ein Anführungszeichen hinter REM stehen haben. Dies ist notwendig, weil hier nicht nur die Standardzeichen (Buchstaben, Zahlen, eben alle Zeichen, die ohne SHIFT oder COMMODORE erreichbar sind) stehen, sondern graphische Zeichen. Warum hier Anführungszeichen stehen müssen, machen Sie sich am einfachsten klar, indem Sie einmal eines – sagen wir in Zeile 0 – wegnehmen. Listen Sie die Zeile ohne Anführungszeichen auf – Sie werden staunen, was passiert.
Und hier gleich noch eine Feinheit, die den gehobenen Programmierstil ausmacht: Was würden Sie zu dem Text "Noch 0 Gimmys da." sagen oder gar zu "Noch 1 Gimmys da."? Oft wird in einem Text eine Anzahl von Dingen angezeigt oder ausgezählt, ohne die Sonderfälle 0 und 1 zu berücksichtigen. Überlegen Sie bei Ihren eigenen Programmen in solchen Fäl-

len also immer, ob der Text auch in den Sonderfällen noch korrekt ist. Wie dieses Problem bei GIMMY-NIMM gelöst wurde, sehen Sie in den Programmzeilen 2020 bis 2070. Die Anzahl der noch vorhandenen Gimmys ist in der Variablen G abgelegt.
Da Ihnen die Gewinnstrategie von Nimm bekannt ist, können Sie sich leicht Ihre Chancen ausrechnen, gegen das Programm zu gewinnen (Sie sollten das unbedingt einmal tun!). In der Grundversion ist GIMMY-NIMM also ein Spiel, das sich für Sie zum Selberspielen eigentlich nicht eignet. Vielmehr sollten Sie GIMMY-NIMM den Leuten vorführen, die behaupten, das Gerede von Elektronengehirnen sei alles Quatsch und ein Computer eine bessere Rechenmaschine.
Um GIMMY-NIMM auch für Sie selbst spielenswert zu machen, haben Sie die Möglichkeit, die Anfangskonfiguration der Gimmys zu verändern. Diese ist nämlich in DATA in Zeile 2090 abgelegt. Das Format ist wie folgt: Jeweils 4 Stellen geben an, ob ein Gimmy an dieser Position ist (1) oder nicht (0), die fünfte Stelle beinhaltet die Summe der vorangegangenen vier. Die vier Zeilen sind durchlaufend hintereinandergehängt. Sie verstehen das Prinzip am einfachsten, indem Sie Zeile 2090 einmal wie folgt ändern:

```
2090 DATA 1,1,1,4,1,1,1,,3,1,1,,,2,1,,,,1,1,1,1,,3
```

Dies verändert die Spielsituation so, daß jetzt in der vierten Zeile nur noch 3 statt wie bisher 4 Gimmys erscheinen. Sie erinnern sich, daß eine leere Position zwischen zwei Kommata bei DATA als 0 interpretiert wird, wenn sie in eine numerische Variable mit READ eingelesen wird. Beginnen Sie in dieser neuen Situation einmal mit dem Zug, in Zeile 1 einen Gimmy wegzunehmen. Mit "Sie spielen wirklich gut!" wird das Programm Ihren Erfolg quittieren. Wenn Sie nun die dargelegte Gewinnstrategie konsequent verfolgen – am besten mit Papier und Bleistift – dann haben Sie durchaus eine Chance, wenigstens einmal gegen GIMMY-NIMM zu gewinnen. Probieren Sie ruhig auch andere Anfangssituationen aus. Zeile 2090 läßt Ihnen in dieser Hinsicht alle Möglichkeiten offen.

Zusammenfassung:
Das fünfzehnte Kapitel brachte Ihnen das altbekannte Nimm-Spiel in einer modernen Version, bei der Ihre Chancen zu gewinnen, recht gering sind – es sei denn, Sie modifizieren das Programm wie gezeigt. Ich hoffe, daß Sie außerdem noch einiges Neues gelernt haben:
- Programmierung maschineller Intelligenz am Beispiel NIMM
- Einbau von Graphikzeichen in REM-Anweisungen
- Beachtung der Sonderfälle 0 und 1 bei Textaufzählungen.

```
0 REM"▒▒▒
```

```
2000 G=0:PRINT"□":FORI=1TO5:PRINTMID$(STR$(I),2)".□";:T=2
2010 IFFL(I)THENG=G+FL(I):FORJ=1TOFL(I):GOSUB6000:FORK=0TO99:NEXT:T=T+5:NEXT
2020 PRINT"■■■":NEXT:POKETS,0:PRINT"■■■■■■■■■■"SPC(13):IFG=0THENPRINT"   KEINE
2030 IFG=1THENPRINT"NUR NOCH
2040 IFG>1THENPRINTSPC(4)"NOCH
2050 PRINTSPC(53):IFG=0THENPRINT"GIMMYS MEHR!":RETURN
2060 IFG=1THENPRINT"  EIN GIMMY!":RETURN
2070 PRINTSPC(2-INT(G/10))G"GIMMYS.":RETURN
2080 POKETL,10:FORI=1TO5:FORJ=1TO4:READF(I,J):NEXT:READFL(I):NEXT:RETURN
2090 DATA1,1,1,1,4,1,1,1,,3,1,1,,,2,1,,,,1,1,1,1,1,4
3000 PRINTQ$"MEIN ZUG: "Y","X:RETURN
3100 PRINTQ$"IHR ZUG : "Y","X:RETURN
3200 B=8:A$="":FORK=1TO3:B=INT(B/2+.2):A=A-B:IFA<0THENA=A+B:A$=A$+"0":GOTO3220
3210 A$=A$+"1"
3220 NEXT:RETURN
3300 IFX>FL(Y)THENX=FL(Y)
3310 FORI=FL(Y)TOFL(Y)-X+1STEP-1:F(Y,I)=0:NEXT:FL(Y)=FL(Y)-X:RETURN
4000 PRINTQ$:PRINT"■SIE SIND DRAN!■":POKEP,0:PRINT"ZEILE
4010 I=0:F=NOTF:PRINTSPC(6)"□"MID$("?▒",F+2,1)
4020 GETG$:Y=VAL(G$):ON-(Y>0ANDY<6)GOTO4030:I=I+1:ONI/10+1GOTO4020,4010
4030 ONFL(Y)+1GOTO4020:PRINTSPC(6)"□□_":PRINTSPC(6)"■"G$:PRINT"■ANZAHL□□
4040 I=0:F=NOTF:PRINT:PRINTSPC(7)MID$("?▒",F+2,1)"□";
4050 GETG$:X=VAL(G$):ON-(X>0ANDX=<FL(Y))GOTO4060:I=I+1:ONI/10+1GOTO4050,4040
4060 PRINT:PRINTSPC(7)"□_":PRINTSPC(7)"■"G$;:FORI=1TO2E3:NEXT:RETURN
5000 T=15:PRINTQ$:PRINT"■_____":PRINT"■ICH DENKE !■
5010 PRINTQ$"■":GOSUB6000:Y=1:X=0:IFFL(Y)=0GOTO5050
5020 FL(1)=FL(1)-1:GOSUB5290:X=X+1:IFC=1ANDD=1ANDE=1GOTO5370
5030 IFFL(1)>0GOTO5020
```

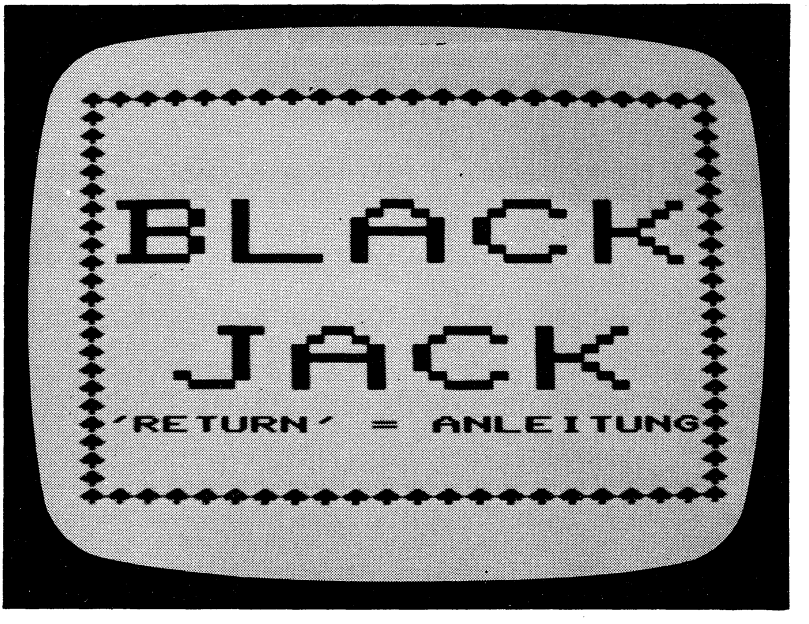
```
5040 FL(1)=FL(1)+X:PRINTQ$"■":GOSUB6000
5050 Y=2:X=0:IFFL(Y)=0GOTO5090
5060 FL(2)=FL(2)-1:GOSUB5290:X=X+1:IFC=1ANDD=1ANDE=1GOTO5370
5070 IFFL(2)>0GOTO5060
5080 FL(2)=FL(2)+X:PRINTQ$"■":GOSUB6000
5090 Y=3:X=0:IFFL(Y)=0GOTO5130
5100 FL(3)=FL(3)-1:GOSUB5290:X=X+1:IFC=1ANDD=1ANDE=1GOTO5370
5110 IFFL(3)>0GOTO5100
5120 FL(3)=FL(3)+X:PRINTQ$"■":GOSUB6000
5130 Y=4:X=0:IFFL(Y)=0GOTO5180
5140 FL(4)=FL(4)-1:GOSUB5290:X=X+1:IFC=1ANDD=1ANDE=1GOTO5370
5150 IFFL(4)>0GOTO5140
5160 FL(4)=FL(4)+X
5170 PRINTQ$"■":GOSUB6000
5180 Y=5:X=0:IFFL(Y)=0GOTO5230
5190 FL(5)=FL(5)-1:GOSUB5290:X=X+1:IFC=1ANDD=1ANDE=1GOTO5370
5200 IFFL(5)>0GOTO5190
5210 FL(5)=FL(5)+X
5220 PRINTQ$"■":GOSUB6000
5230 PRINTQ$:PRINT"■■■■ SIE SPIELEN WIRKLICH  GUT!!!";
5240 FORI=1TO2500:NEXT:Y=0
5250 Y=Y+1:GOSUB5270:IFX=1THENRETURN
5260 GOTO5250
5270 IFFL(Y)>0THENFL(Y)=FL(Y)-1:X=1
5280 RETURN
5290 FORI=1TO3:FS(I)=0:NEXT:FORJ=1TO3:FORI=1TO5:PRINTQ$"■":T=15
5300 GOSUB6000:PRINT:A=FL(I):GOSUB3200:FS(J)=FS(J)+VAL(MID$(A$,J,1)):NEXT
5310 NEXT:C=0:D=0:E=0
```

```
5320 PRINTQ$"■":GOSUB6000
5330 IF(FS(1)/2)-INT(FS(1)/2)=0THENC=1
5340 IF(FS(2)/2)-INT(FS(2)/2)=0THEND=1
5350 IF(FS(3)/2)-INT(FS(3)/2)=0THENE=1
5360 RETURN
5370 FL(Y)=FL(Y)+X
5380 PRINTQ$SPC(15)"*  *":PRINTSPC(14)"* ** *
5390 PRINTSPC(13)"* \||/ *":PRINTSPC(14)"*—OO—*":PRINTSPC(16)"◡"
5400 FORI=128TO254STEP3:POKETS,I:FORJ=0TO49:NEXT:NEXT:POKETS,0:RETURN
6000 U=16*RND(1)+1:IFU=HGOTO6000
6010 H=U:ONUGOSUB6040,6050,6060,6070,6080,6090,6100,6110
6020 IFU>8THENONU-8GOSUB6120,6130,6140,6150,6160,6170,6180,6190
6030 POKETS,255-U:PRINT"■■■■";:RETURN
6040 PRINTTAB(T)"■\##/":PRINTTAB(T)" OO ":PRINTTAB(T)"  ◡":RETURN
6050 PRINTTAB(T)"■\▒▒▒ ":PRINTTAB(T)" OO\":PRINTTAB(T)"  ◡":RETURN
6060 PRINTTAB(T)"■—, —":PRINTTAB(T)" ●● ":PRINTTAB(T)"  ◡":RETURN
6070 PRINTTAB(T)"■^ ^":PRINTTAB(T)" ●● ":PRINTTAB(T)"  ⊔":RETURN
6080 PRINTTAB(T)"■\▒▒▒/":PRINTTAB(T)" OO ":PRINTTAB(T)"  ◡":RETURN
6090 PRINTTAB(T)"■ ___ ":PRINTTAB(T)"/OO\":PRINTTAB(T)"  ◡":RETURN
6100 PRINTTAB(T)"■\##/":PRINTTAB(T)" $$ ":PRINTTAB(T)"  ◡":RETURN
6110 PRINTTAB(T)"■ ▒▒▒/":PRINTTAB(T)"/OO ":PRINTTAB(T)"  ◡":RETURN
6120 PRINTTAB(T)"■ ▒▒▒ ":PRINTTAB(T)"/**\":PRINTTAB(T)"  ◡":RETURN
6130 PRINTTAB(T)"■\___/":PRINTTAB(T)" "OO" ":PRINTTAB(T)"  ◡":RETURN
6140 PRINTTAB(T)"■ WW ":PRINTTAB(T)"/OO\":PRINTTAB(T)"  ◡":RETURN
6150 PRINTTAB(T)"■\WW/":PRINTTAB(T)" ** ":PRINTTAB(T)"  ◡":RETURN
6160 PRINTTAB(T)"■\┬┬ ":PRINTTAB(T)" OO\":PRINTTAB(T)"  ◡":RETURN
6170 PRINTTAB(T)"■ ┬┬/":PRINTTAB(T)"/** ":PRINTTAB(T)"  ◡":RETURN
6180 PRINTTAB(T)"■ ┬┬ ":PRINTTAB(T)"—OO—":PRINTTAB(T)"  ◡":RETURN
```

```
6190 PRINTTAB(T)"■ ,,,,, ":PRINTTAB(T)"—oo—":PRINTTAB(T)"  ◡":RETURN
7000 IFGTHENRETURN
7010 GOSUB9000:PRINT"■_____"
7020 PRINT"■LEIDER HABEN SIE VER-":PRINT"_____":PRINT"■LOREN!■ LASSEN SIE
7030 PRINT"■SICH DESWEGEN ABER":PRINT"■KEINE GRAUEN HAARE
7040 PRINT"■WACHSEN, SONDERN VER-":PRINT"■SUCHEN SIE ES NOCH":PRINT"■EINMAL!
7050 FORI=254TO128STEP-2:POKES,I:FORJ=0TO49:NEXT:NEXT:POKES,0:GOSUB9000:RUN
7060 IFGTHENRETURN
7070 GOSUB9000:PRINT"■_____"
7080 PRINT"■GRATULIERE, SIE HABEN":PRINT"_____"
7090 PRINT"■GEWONNEN! ABER GEBEN":PRINT"■SIE MIR DOCH NOCH EINE
7100 PRINT"■CHANCE, AUCH MAL ZU":PRINT"■GEWINNEN!
7110 FORI=128TO254STEP2:POKES,I:FORJ=0TO49:NEXT:NEXT:POKES,0:GOSUB9000:RUN
8000 POKE36879,61:PRINTCHR$(142)"■■"SPC(6)"_____"
8010 PRINT"  *** ■GIMMY-NIM■ ***■":FORT=1TO16STEP5:GOSUB6000:NEXT
8020 PRINT"■■■■ABWECHSELND NEHMEN":PRINT"SIE UND DER COMPUTER
8030 PRINT"AUS EINER REIHE EINE":PRINT"BELIEBIGE ANZAHL GIM-
8040 PRINT"MYS WEG. JEDER MUSS":PRINT"MINDESTENS EINEN GIM-
8050 PRINT"MY WEGNEHMEN, WENN ER":PRINT"AM ZUGE IST. WER DEN
8060 PRINT"LETZTEN GIMMY WEG-":PRINT"NIMMT, DER HAT GEWON-":PRINT"NEN!
9000 POKEP,0:PRINTSPC(8)"_____":PRINTSPC(8)"■RETURN
9010 GETG$:ON-(G$<>CHR$(13))GOTO9010:RETURN
```

16.BLACKJACK

Mit BLACKJACK haben Sie nach ROULETTE eine weitere Möglichkeit, der Spielleidenschaft zu frönen. Auch hier können Sie viel Geld gewinnen – oder verlieren. Erwarten Sie aber bitte nicht, daß ich für die Schulden des Computers aufkomme, nur weil das Programm von mir geschrieben wurde. Diesbezügliche Anfragen an Commodore werden wahrscheinlich ebenfalls zur Erfolgslosigkeit verurteilt sein.
Was Sie aber auf jeden Fall gewinnen, ist Erfahrung im Programmieren. Dafür wollen wir nämlich jetzt gemeinsam sorgen. Sehen wir uns als erstes die Routine an, welche die Spielkarten mischt und austeilt. Sie ist recht universell und kann auch für andere Kartenspiele eingesetzt werden.
Zu Spielbeginn wird zunächst ein String mit den vier Farben festgelegt. Dies geschieht in Zeile 8000 (FK$). Außerdem wird jeder Farbe ein String mit den Kartenwerten (W$) und den Kartensymbolen (S$) beigegeben (ebenfalls Zeile 8000). Die Routine, die W$(..) und S$(..) zusammensetzt, liegt in den Programmzeilen 9000 und 9010. Wie Sie sehen, stehen die Kartensymbole (z.B. "D" für "Dame") an den gleichen Positionen innerhalb des Strings wie die entsprechenden Werte.



Ausgewählt werden die Karten per Zufallsfunktion in den Zeilen 6000 bis 6060. In Zeile 6000 wird als erstes eine der vier Farben ausgesucht und in F$ gespeichert. Bedenken Sie, daß FK$ die Farbsymbole nacheinander enthält. Durch die MID$-Funktion wird eines davon ausgewählt, in Abhängigkeit davon, welche Zufallszahl zwischen 1 und 4 der Variablen F durch RND zugewiesen wurde. Nachdem nun die Farbe der Karte feststeht, wird in Zeile 6010 ein ebenfalls durch Zufall bestimmter Wert aus W$(F) geholt und in der Variablen WK festgehalten. Als nächstes muß das dem Wert entsprechende Kartensymbol gefunden werden.Wenn der Wert aus W$ von ganz links stammt (S=1), dann geschieht das Symbolsuchen in der Zeile 6050, in allen anderen Fällen in Zeile 6030. Das Symbol steht danach in der Variablen SK$. Nachdem also nun eine Karte "gezogen" ist, muß sie auch vom Stapel genommen werden. Sonst tauchen auf einmal zwei Pik-Könige oder gar drei Herz-Damen auf, was ja nun doch allen Regeln beim Kartenausteilen widerspräche (von der Möglichkeit des Falschspielens wollen wir einmal absehen). Die ausgeteilten Symbole und Werte müssen also aus dem Gesamstring entfernt werden, damit sie nicht noch einmal drankommen. Für die Werte passiert das in Zeile 6030 (Für S gleich 1 in Zeile 6060), für die Kartensymbole in Zeile 6040 (für S gleich 1 in Zeile 6060), für die Kartensymbole in Zeile 6040 (für S gleich 1 in Zeile 6060). Und damit ist eine Spielkarte vom Kartenstapel "gezogen". So einfach ist das.
Und das Mischen? Nun, das geschieht ganz einfach, indem alle Strings für die Kartenwerte und -symbole (W$ und S$) wieder neu festgelegt werden. Wie wir gesehen haben, geschieht dies in der Zeile 8000.
Dafür, daß die auf die gezeigte Weise ausgewählte Spielkarte auch auf dem Bildschirm dargestellt wird, sorgen die Programmzeilen 5000 bis 5200. Es dürfte Ihnen nicht schwerfallen, hier selbst herauszufinden, wie es zur Darstellung der einzelnen Kartensymbole kommt.
Die in BLACKJACK benutzte Methode zur Kartenverteilung läßt sich auch für alle anderen Kartenspiele verwenden, wo ein 52er Blatt gebraucht wird. Die bei BLACKJACK verwendeten amerikanischen Kartenwerte (zum Beispiel zählt der König zehn statt wie in Deutschland üblich vier) können Sie leicht in Zeile 9000 innerhalb der Variablen W$(..) ersetzen. Tauschen Sie dazu einfach die 10er gegen die entsprechenden deutschen Werte aus. Die Reihenfolge der Karten können Sie in Zeile 9010 an S$(..) ablesen.
Wenn Sie sich die Darstellung der einzelnen Karten in den Zeilen 5030 bis 5200 ansehen, dann werden Sie feststellen,

daß sich dort laufend Cursorsteuerungszeichen in Strings und die Variable F$ in PRINT-Anweisugen abwechseln. F$ enthält das Symbol für die entsprechende Kartenfarbe (Kreuz, Pik, Herz oder Karo). Um Speicherplatz zu sparen, sind in diesen PRINT-Befehlen alle Semikolons weggelassen. Statt

```
PRINT F$;"  ";F$;" "F$;"  ";F$
```

wird einfach

```
PRINT F$"  "F$" "F$"  "F$
```

geschrieben. Das ist durchaus korrekt oder jedenfalls erlaubt der Computer die abgekürzte Schreibweise. Machen Sie sich aber klar, daß dies ausschließlich bei PRINT-Befehlen geht, nicht aber bei Zuweisungen. Sowohl dieser Ausdruck

```
FALSCH

A$ = F$;"  ";F$;"  ";F$;"  ";F$;"  "
```

als auch dieser

```
FALSCH

A$=F$"  "F$"  "F$"  "F$"  "
```

sind beide falsch. Bei einer Zuweisung an eine Variable muß der Stringausdruck mittels des Stringverkettungsoperators '+' korrekt gebildet werden. Unser Beispiel sieht dann so aus:

```
RICHTIG

A$ = F$+"  "+F$+"  "+F$+"  "+F$+"  "
```

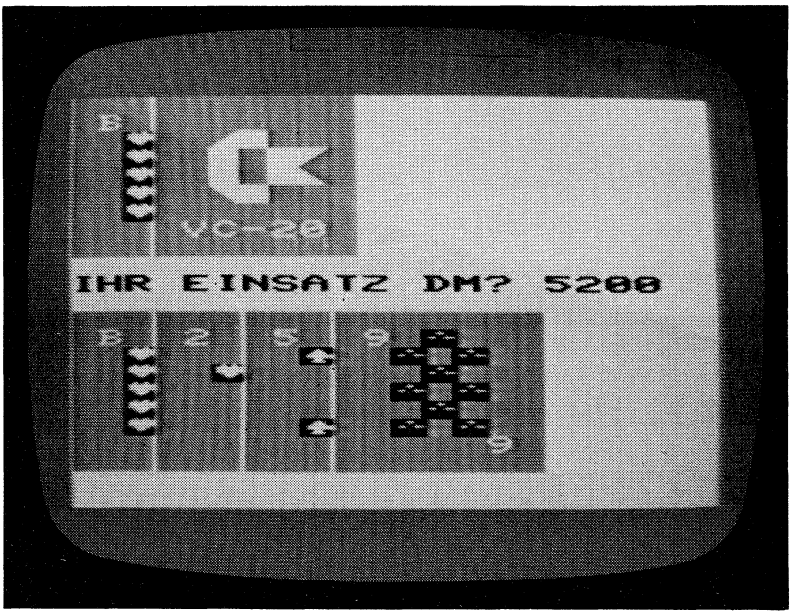
Umgekehrt hingegen ist es sehr wohl möglich, auch in PRINT-Anweisungen mit '+' zu arbeiten. Unser Beispiel:

```
RICHTIG

PRINT F$+"  "+F$+"  "+F$+"  "+F$+"  "
```

Viele Programmierer beachten diesen grundsätzlichen Unterschied zwischen PRINT und einer Zuweisung an eine Stringvariable nicht bzw. kennen ihn gar nicht und suchen dann oft verzweifelt den Fehler in einem Programm. Aber Sie wissen ja

jetzt um diesen Punkt und können ihn in ihren zukünftigen Programmen beachten.



In manchen Fällen ist es erforderlich, eine Zahl, die einer Variablen zugewiesen wurde, daraufhin zu prüfen, ob sie Nachkommastellen enthält. In BLACKJACK zum Beispiel sind als Einsätze nur ganze Mark und keine Pfennigbeträge zugelassen. Wie so eine Abfrage am einfachsten programmiert werden kann, zeigt Ihnen Programmzeile 7030. Ist

E–INT(E)

gleich Null, dann hat der E zugewiesene Wert keine Nachkommastellen. Andernfalls ist das Ergebnis dieses Ausdrucks der Dezimalteil der Zahl, also bei '3.45' zum Beispiel '.45'. In Zeile 7030 können Sie sehen, wie dieser Umstand mit IF . .THEN leicht geprüft werden kann.
Sie wissen bereits, daß wir mit 'PRINT CHR$(142)' den Computer in den Graphikmodus schalten, und mit 'PRINT CHR$(14)' in den Groß/Kleinschreibmodus. Daß diese CHR$-Funktionen in den PRINT-Anweisungen auch mit Text gemischt werden können, zeigt Ihnen Programmzeile 8020. Der CHR$-Funktion folgt sofort ein String. Auf diese Weise kann nicht nur die Umschaltung zwischen den zwei Tastatur-Modi

vorgenommen werden, sondern CHR$ kann noch viele weitere Steuerfunktionen bewirken. Probieren Sie doch einmal die folgende Programmzeile aus:

```
10 PRINT CHR$(147)"Hier ist der VC-20"
```

Geben Sie RUN ein. Was passiert? Wie Sie sehen, ist CHR$(147) gleichbedeutend mit der Funktion 'Bildschirm löschen'. Natürlich könnten wir in diesem Fall auch das bekannte Herzchen direkt in die Anführungszeichen setzen. Die Verwendung von CHR$ ist aber im Grunde die korrektere Lösung. Weitere CHR$-Steuercodes sind:

29	Cursor nach rechts
157	Cursor nach links
17	Cursor nach unten
145	Cursor nach oben
18	RVS ON
146	RVS OFF
5	WHT (weiß)
29	GRN (grün)
31	BLU (blau)
144	BLK (schwarz)
156	PUR (purpurn)
158	YEL (gelb)
159	CYN (Cyan)

Aus Gründen der Speicherkapazität wird man aber im Normalfall die entsprechende Taste im Anführungszeichenmodus drücken als CHR$ zu benutzen. Sie sollten aber auf jeden Fall um diese Zusammenhänge wissen. Übrigens können Sie den CHR$-Wert eines beliebigen Zeichens mittels ASC abfragen. Geben Sie doch einmal

```
?ASC("A")
```

ein. Das Ergebnis ist der ASCII-Code des Buchstabens "A". Mit

```
?CHR$(65)
```

wird das "A" wieder dargestellt. ASC und CHR$ sind die Umkehrfunktionen zueinander.
Damit will ich Sie dem Spielgenuß von BLACKJACK überlassen. Und nehmen Sie es Ihrem Computer nicht krumm, wenn er Sie verlieren läßt; er kann ja nichts dafür!

Zusammenfassung:
Damit haben Sie Ihre Spielesammlung um ein interessantes Kartenspiel erweitert. Ihr Wissen sollten Sie um folgendes erweitert haben:

- Routine zur Kartenverteilung und Ausspielen (52er Blatt)
- Unterscheidung zwischen der Verkettung mit Strings mit ';' und mit '+' bei PRINT und Zuweisungen
- Prüfen einer Zahl auf Nachkommastellen
- Benutzung von CHR$ zur Erzeugung von Steuercodes

```
1 REM BLACKJACK/17&4
2 REM
3 REM (C) 04/1982 BY
4 REM
5 REM ANDREAS DRIPKE
6 REM AUG-BEB-STR 38
7 REM 6200 WIESBADEN
8 :
100 IFPEEK(158)OR191ANDSTTHENPRINT"■?LADEFEHLER:"PEEK(158);:END
110 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888:A=36879
1000 REM----------
1010 GOSUB8000:REM ANFANG/ERKLAERUNG
1020 GOSUB7000:REM EINSATZ
1030 GOSUB4000:REM STARTKONFIGURATION
1040 GOSUB2000:REM NOCH EINE KARTE?
1050 IFKGOTO1040:REM NOCH EINE KARTE!
1060 GOSUB2060:REM COMPUTERKARTEN
1070 GOSUB3000:REM ANALYSE
1080 GOTO1030
1090 REM----------
2000 IFSP>21THENK=0:RETURN
2010 PRINTLEFT$(A$,10):K=1:PRINTX$:PRINT"■□NOCH EINE KARTE?
2020 PRINTSPC(17)"□"MID$("JN",D+2,1):D=NOTD:I=0
2030 GETG$:IFG$="J"THENPRINTSPC(17)"□JA":GOSUB9210:GOTO9300
2040 IFG$="N"ANDSP>16THENPRINTSPC(17)"□NEIN":GOSUB9210:K=0:RETURN
2050 I=I+1:ON-(I<12)GOTO2030:GOTO2020
2060 IFSP>21THENRETURN
2070 P$="GUT, ICH BIN DRAN.":GOSUB9200:Z=1:T=3:PRINT"■":GOSUB6000:C=C+WK
```

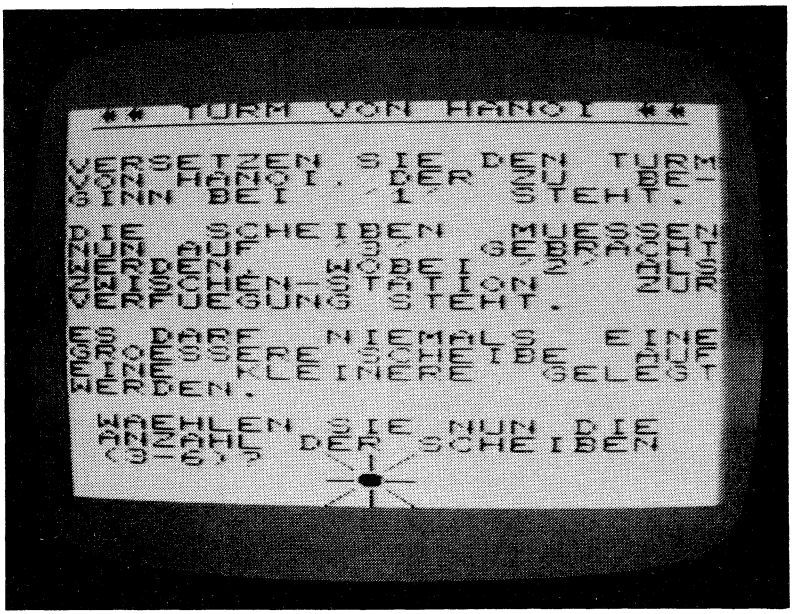
```
2080 GOSUB5000:T=6:IFC>=SPTHENRETURN
2090 PRINT"■":GOSUB6000:C=C+WK:GOSUB5000:T=T+3:IFC<SPANDT<40GOTO2090
2100 RETURN
3000 P=0:IFSP=21ANDC=21THENP$="ZWEIMAL BLACKJACK!":GOTO3070
3010 IFSP>21THENP$="LEIDER MEHR ALS 21!":P=-1:GOTO3070
3020 IFC<22GOTO3040
3030 P$="ICH HABE ZUVIEL.":GOSUB9200:P$="SIE HABEN GEWONNEN!":P=1:GOTO3070
3040 IFSP=CTHENP$="UNENTSCHIEDEN!":P=0
3050 IFSP<CTHENP$="SIE HABEN VERLOREN!":P=-1
3060 IFSP>CTHENP$="SIE HABEN GEWONNEN!":P=1
3070 GOSUB9200:GV=E*P+GV:P$="KONTOSTAND: DM "+STR$(GV):GOSUB9200:GOTO7000
4000 PRINT"■";:GOSUB6000:C=WK:Z=1:T=0:GOSUB5000:FORI=0TO8
4010 PRINTSPC(3)"■■■
```

```
5130 PRINT"■"F$"■"F$"■"F$:RETURN
5140 PRINT"■"F$"■"F$"■■■■"F$"■"F$:RETURN
5150 PRINTF$"■"F$"■■■■"F$"■■■■"F$"■"F$:RETURN
5160 PRINTF$"■"F$"■■■■"F$"■"F$"■■■■"F$"■"F$:RETURN
5170 PRINTF$"■"F$"■■■"F$"■■■"F$"■"F$"■■■■"F$"■"F$:RETURN
5180 PRINTF$"■"F$"■■■"F$"■■■"F$"■"F$"■■■"F$"■■■"F$"■"F$:RETURN
5190 PRINTF$"□"F$"■"F$"■■■"F$"■■■"F$"■"F$"■■■"F$"■■■"F$"■"F$:RETURN
5200 PRINTF$"□"F$"■"F$"■■■"F$"■■■"F$"■"F$"■■■■■"F$"■"F$"■"F$"■■■■■"F$:RETURN
6000 F=INT(4*RND(1)+1):F$=MID$(FK$,F*2-1,2):IFLEN(W$(F))<3GOTO6070
6010 W=INT(LEN(W$(F))*RND(1)/2)*2+1:WK$=MID$(W$(F),W,2):WK=VAL(WK$)
6020 S=W:IFS=1GOTO6050
6030 SK$=RIGHT$(MID$(S$(F),S,2),1):W$(F)=LEFT$(W$(F),S-1)+MID$(W$(F),S+2)
6040 S$(F)=LEFT$(S$(F),S-1)+MID$(S$(F),S+2):RETURN
6050 SK$=RIGHT$(LEFT$(S$(F),2),1)
6060 W$(F)=RIGHT$(W$(F),LEN(W$(F))-2):S$(F)=RIGHT$(S$(F),LEN(S$(F))-2):RETURN
6070 PRINTLEFT$(A$,10)"■_____":PRINT"■MOMENT, ICH MISCHE!□□
6080 GOSUB9000:FORI=1TO2E3:NEXT:PRINTX$:PRINTX$:GOTO6000
7000 PRINTLEFT$(A$,10):INPUT"IHR EINSATZ DM";E$:GOSUB9210:E=VAL(E$)
7010 IFE>25E3THENP$="MAXIMUM DM 25000!":GOSUB9200:GOTO7000
7020 IFE<1THENP$="MINIMUM DM 1!":GOSUB9200:GOTO7000
7030 IFE-INT(E)THENP$="NUR GANZE MAERKER!":GOSUB9200:GOTO7000
7040 RETURN
8000 POKE650,128:DIMW$(4),S$(4):FK$="■↑■↑■↓■":GOSUB9000:A$="■":FORI=0TO24
8010 A$=A$+"■":NEXT:X$="                                    ":POKEA,25
8020 PRINTCHR$(142)"■■■■■■■■ "■■■■ ■■■■    ■■■■ ■■■■ ■■■■
```

```
8060 PRINT"     ■■■  *  *  *■■■  *  *":FORI=1TO1E3:NEXT:FORI=VCTOVC+505
8070 POKEI,0:NEXT:PRINTLEFT$(A$,19)" 'RETURN' = ANLEITUNG"
8080 Z=128*INT(2*RND(1))OR64+63*RND(1):FORI=VBTOVB+21:POKEI,Z:POKEI+484,Z:NEXT
8090 FORI=VBTOVB+484STEP22:POKEI,Z:POKEI+21,Z:NEXT:GETG$
8100 IFG$<>CHR$(13)GOTO8080
8110 POKEA,125:PRINT"■ ┌──────────────────────┐":PRINT" |BLACKJACK - 17 & 4|
8120 PRINT" └──────────────────────┘":PRINT"■BLACKJACK IST EIN
8130 PRINT"KARTENSPIEL GEGEN DEN":PRINT"COMPUTER. JEDER ZIEHT
8140 PRINT"KARTEN VOM STAPEL UND":PRINT"GEWINNER IST, WER DIE
8150 PRINT"HOECHSTE SUMME HAT.":PRINT"DIESE SUMME DARF ABER
8160 PRINT"NICHT GROESSER ALS 21":PRINT"UND NICHT KLEINER ALS
8170 PRINT"17 SEIN (17 + 4 =21).":GOSUB9100:POKEA,41:PRINT"■■WER DIE 21 UEBER-
8180 PRINT"ZIEHT, HAT VERLOREN,":PRINT"EBENSO, WER MIT 5
8190 PRINT"KARTEN KEINE 17 PUNK-":PRINT"TE ERREICHT HAT. AM
8200 PRINT"BESTEN IST EIN BLACK-":PRINT"JACK (21)! DIE KARTEN
8210 PRINT"ZAEHLEN WIE FOLGT:":PRINT"■2 BIS 9 = WERT (2-9)
8220 PRINT"ZEHN, BUBE, DAME,":PRINT"KOENIG = 10, ASS = 11":GOSUB9100
8230 POKEA,59:PRINT"■■■
```

17. TURM VON HANOI

Nachdem wir im letzten Kapitel ein Glücksspiel kennengelernt haben, folgt diesmal wieder etwas zum Nachdenken. Sie haben die Aufgabe, den TURM VON HANOI zu versetzen – was gar nicht so einfach ist. Deswegen können Sie auch, wenn Sie nicht mehr weiter wissen, sich die Lösung vom Computer vorführen lassen. Allerdings sollten Sie zu dieser Möglichkeit nur greifen, wenn Sie wirklich keine eigene Lösung finden.



Wir wollen nun sehen, was wir in TURM VON HANOI Neues lernen können. Zuvor noch ein Tip: "SAVEn" Sie TURM VON HANOI auf einer Kassette oder Diskette ab, bevor Sie es das erste Mal spielen! Das Programm enthält nämlich einen "Selbstvernichtungsbefehl", der unter bestimmten Umständen anspricht. Und es wäre doch schade, wenn Sie wegen einer Unbedachtheit das gesamte Programm ein zweites Mal eintippen müßten.
Mit diesem Selbstvernichtungsbefehl wollen wir uns als erstes beschäftigen. Eine Möglichkeit, daß sich ein Programm selbst löscht, kennen Sie schon. Der Befehl NEW darf nicht nur von Hand eingegeben werden, sondern kann auch innerhalb eines

Programmes stehen. Geben Sie einmal folgende Zeile ein:

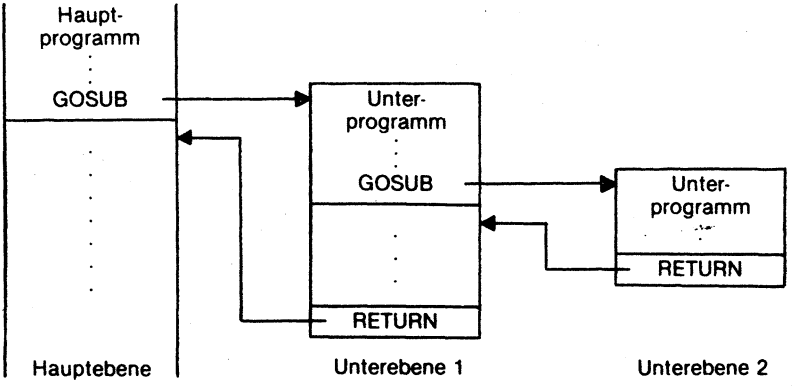
```
10 PRINT"HIER IST DAS PROGRAMM." : NEW
```

Wenn Sie jetzt RUN tippen, erscheint der Text "HIER IST DAS PROGRAMM." auf dem Bildschirm und danach 'READY.'. Geben Sie nun LIST ein und Sie sehen, daß das Programm verschwunden ist. NEW hat also gewirkt.
In TURM VON HANOI ist die Selbstzerstörung allerdings noch etwas raffinierter programmiert. Probieren Sie doch einmal diesen Befehl aus (Vorsicht! Das Programm geht verloren!):

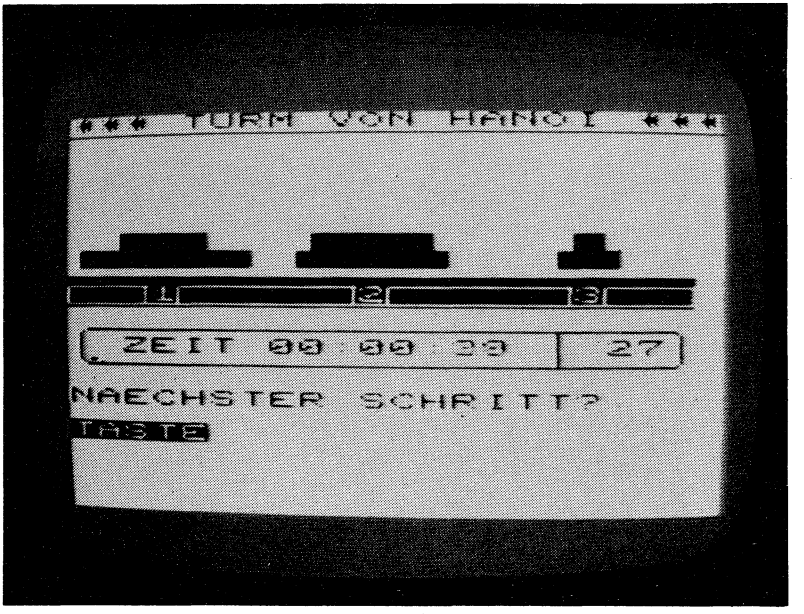
```
SYS 64802
```

Verblüffend, nicht wahr? Der Computer meldet sich gerade so, als ob er eben eingeschaltet worden wäre. Genau dieser Trick findet sich auch in TURM VON HANOI (Zeile 9150). In den Zeilen 9090 bis 9140 wird vorgetäuscht, daß das Programm beendet sei. Dieser Eindruck wird durch das 'READY.' (Zeile 9100) und die nachfolgende Cursorsimulation (Zeilen 9110 bis 9140) hervorgerufen. Wie der Cursor zum Blinken gebracht wird, sollten Sie ohne Schwierigkeiten verstehen, wir haben über solche Blinkroutinen schon ausführlich gesprochen. Wenn nun der Spieler irgendeine Taste drückt, um zum Beispiel das Programm aufzulisten oder abzuspeichern, dann ..... ja, was mag dann wohl passieren?
Diese Routine hat nichts mit dem eigentlichen Programm zu tun, sie ist nur als kleiner Scherz gedacht. Wir wollen uns aber jetzt dem Programm in seiner Gesamtheit zuwenden.
TURM VON HANOI bietet nämlich noch genug Platz für eine Vielzahl von REM-Zeilen, und wir wollen uns dies zunutze machen und diesmal nicht nur einzelne Routinen besprechen, sondern einmal einen Gesamtüberblick gewinnen.
Sehen Sie sich im Programm die Zeilen 1000 bis 1120 an. Sie enthalten das Grundgerüst des Programmes, die sogenannte "Grobstruktur". Aus der Grobstruktur ist sofort der Ablauf ersichtlich. Ein strukturiertes Programm zeichnet sich dadurch aus, daß die einzelnen Funktionseinheiten ganz klar voneinander getrennt sind und der logische Zusammenhang gut erkennbar ist. In TURM VON HANOI können wir das ganze Programm auf den 11 Zeilen von 1010 bis 1110 erkennen. Und der Vorteil? Nun – strukturiert Programmieren ist einfacher, einfacher beim Erstellen des Programmes, einfacher bei Modifikationen.
Seit Anbeginn der Computertechnologie bemühen sich die Ersteller von Software um Methoden, die Programmierung der

Computer einfacher und somit rationeller und effektiver zu gestalten. Im Verlaufe dieser Entwicklung sind eine Vielzahl von verschiedenen Programmiersprachen entstanden. Eine der besten Methoden, um die Softwareerstellung sinnvoller und effektiver zu machen, ist – nach heutigem Erkenntnisstand – die strukturierte Programmiersprache. Machen wir uns klar, warum das so ist.
Wenn die Idee für ein Programm da ist und die vorbereitenden Arbeiten wie das Erstellen von Flußdiagrammen oder Struktogrammen abgeschlossen sind, dann beginnt man mit der Grobstruktur, d.h. man überlegt sich, aus welchen einzelnen Funktionsteilen das Programm bestehen soll und wie diese zusammenpassen müssen. Ohne Details zu kennen, kann man somit den logischen Zusammenhang schon programmieren. Erst jetzt werden nach und nach die einzelnen Funktionseinheiten geschrieben und eingepaßt. Wenn eine solche Funktionseinheit sich aus mehreren Untereinheiten zusammensetzt, dann kann man hier wieder genauso vorgehen. Wir nennen diese Methodik "Top-Down", d.h. man fängt auf der höchsten Stufe an und fügt nach und nach immer mehr Details hinzu – solange, bis das Programm fertig ist.
In Ihren eigenen Programmen sollten Sie sich bemühen, so strukturiert wie nur möglich zu arbeiten. Praktisch heißt das: Führen Sie einzelne Funktionseinheiten als Unterprogramm aus, und rufen Sie diese dann mittels GOSUB auf. Sie werden sehen, daß sich auf diese Art und Weise auch komplexe Programme in relativ kurzer Zeit anfertigen lassen.
Wir wollen uns einmal ein solches Unterprogrammschema verdeutlichen:



Wie das in der Praxis aussieht, zeigt TURM VON HANOI. Nehmen Sie sich Zeit, und verfolgen Sie dort die Struktur des Programmes. Na, sehen Sie, was ich meine?
Wichtig ist natürlich in diesem Zusammenhang auch die Dokumentation des Programmes. In TURM VON HANOI finden Sie REM-Anweisungen an allen entscheidenden Programmstellen. Dies und die konsequente Strukturierung ermöglichen einen leichten Überblick über das gesamte Programm. Natürlich kann man oft aus Mangel an Speicherplatz nicht immer so viele REMs aufnehmen, Sie sollten aber auch nicht an ihnen sparen, wenn es gar nicht nötig ist. Je ausführlicher Sie ein Programm dokumentieren, desto einfacher ist es für Sie, das Programm auch nach einem längeren Zeitraum noch zu verstehen und zum Beispiel Änderungen durchzuführen.



Wenden wir uns jetzt einmal den nicht ganz so ernsten Dingen des Programmierens zu. Oft ist es ganz schön, wenn ein Programm ein nettes Bildchen auf den Bildschirm zaubert. In TURM VON HANOI haben wir zwar kein Bild, aber der Programmname erscheint am Anfang in übergroßen Buchstaben. Das Problem ist dasselbe, nämlich wie müssen die einzelnen PRINT-Befehle aussehen, so daß sich zum Schluß das Gesamtbild (oder der Gesamtschriftzug) ergibt? Die Antwort darauf wollen wir jetzt suchen. Dazu nehmen wir einmal an, wir

wollten ein kleines Bild in die Mitte des Bildschirmes zeichnen. Nun, dann tun wir es auch. Mit dem Cursor malen wir unser Bild gerade so, wie wir es möchten, fix und fertig. Nur die RETURN-Taste dürfen wir nicht drücken, weil es sonst zu Fehlermeldungen kommen könnte. Wir können aber SHIFT und RETURN betätigen, was bis auf die Fehlermeldungen den gleichen Effekt wie bloßes RETURN hat, der Cursor springt also zum Anfang der nächsten Bildschirmzeile.
Wenn das Bild fertig ist, dann müssen wir es mit Zeilennummern versehen, damit es als Programm übernommen werden kann. Wir setzen also an den Anfang jeder Bildschirmzeile, die einen Teil unseres Bildes enthält, eine Zeilennummer. Von oben nach unten sollten die Nummern größer werden.
Der Zeilennummer muß ein Fragezeichen (als Abkürzung für PRINT) und diesem ein Anführungszeichen (") folgen. Dann drücken wir RETURN, um die Zeile in das Programm zu übernehmen.
Wenn Sie es korrekt machen wollen, dann führen Sie den Cursor noch einmal über alle Zeilen hinweg und schließen jeden PRINT-Befehl mit einem Anführungszeichen ab. Dies ist aber nicht unbedingt notwendig.
Kommen im Bild revers (RVS ON) gedruckte Zeichen vor, dann müssen wir diesen jetzt noch eine Sonderbehandlung zuteil werden lassen. Dabei gehen wir Zeile für Zeile vor. Wir setzen den Cursor auf das erste reverse Zeichen der jeweiligen Zeile. Sodann drücken wir einmal INST (mit SHIFT) und dann CTRL und RVS ON gleichzeitig. Der Erfolg ist ein reverses "R". Nun überschreiben wir die reversen Zeichen nichtrevers. Aus einem RVS-"A" wird also ein Normal-"A". Dies tun wir solange, bis das letzte Reverszeichen umgeändert ist. Der Cursor steht dann auf dem ersten folgenden von Anfang an nicht reversen Zeichen. Jetzt drücken wir nochmals INST und dann CTRL und RVS OFF. Es erscheint das dazugehörige Steuerzeichen.
Für jede Folge von Reverszeichen der Zeile müssen wir diese Prozedur wiederholen. Wenn wir fertig sind, drücken wir RETURN und nehmen uns die nächste Zeile vor (sofern sie RVS-Zeichen enthält). Das Programm "AUF LOS GEHT'S LOS" aus Kapitel 14 bietet Ihnen ein gutes Beispiel für ein Titelbild, das ausschließlich aus reversen Zeichen besteht.
Zum Schluß beginnt also jedes einzelne und jede Folge von Reverszeichen mit RVS ON und endet mit RVS OFF. Die Reverszeichen selbst sind nicht revers, sondern in normaler Darstellung.
Warum ist diese umständliche Umänderung überhaupt notwendig? Es ist nun einmal so, daß in einer PRINT-Anweisung

Zeichen, die im Ausdruck revers dargestellt werden sollen, im Listing selbst nicht revers erscheinen dürfen, sondern von RVS ON und RVS OFF begrenzt sein müssen. Der Computer versteht es eben nicht besser.
Nachdem wir nun unser Bild soweit übernommen haben (durch die RVS-Prozedur sieht es etwas verzerrt aus, das macht aber nichts), können wir den Bildschirm löschen und uns das Ergebnis mit RUN ansehen. Wenn wir alles richtig gemacht haben, sieht das Resultat so prächtig aus wie unser Anfangsentwurf. Falls nicht, listen wir uns die entsprechenden Zeilennummern auf und führen Verbesserungen und Korrekturen durch.
Die Methode, erst frei auf dem Bildschirm zu zeichnen und das Bild erst danach in PRINT-Anweisungen unterzubringen, ist auf jeden Fall besser als von Anfang an mit PRINT zu operieren. Hat das Bild nach der Übernahme in das Programm noch nicht die richtige Position, so läßt sich das leicht mit TAB(. .) oder SPC(. .) zwischen dem PRINT-Befehl und dem ersten Anführungszeichen beheben. Die übergroße Schrift in TURM VON HANOI steht in den Zeilennummern 8010 bis 8060. Auch hier wird die Schrift mit TAB und SPC auf Mitte gesetzt.
Ich denke, daß Sie jetzt Bilder selbständig mit dem Computer zeichnen können. Hier gilt allerdings in ganz besonderem Maße: Übung macht den Meister. Entwerfen sie einmal ein paar stilvolle Gemälde, und Sie werden die Sache perfekt beherrschen.
Wir wollen uns jetzt mit etwas beschäftigen, das genau wie Bilder der Schönheit eines Programmes dient. Wenn Sie die Erklärung in PRINT-Anweisungen von TURM VON HANOI schon auf dem Bildschirm gelesen haben, dann ist Ihnen bestimmt aufgefallen, daß der Text wie gedruckt angeordnet ist. Er schließt links und rechts immer an der gleichen Stelle ab, dafür sind die Abstände zwischen den Worten im Text zum Teil unterschiedlich. Dadurch wird die Erklärung leichter lesbar und sieht einfach eleganter aus, als wenn – wie normalerweise üblich – der Text nur am linken Rand glatt anliegt und am rechten je nach Wortlänge variiert. Wenn ein Text rechts- und linksbündig anliegt, dann nennt man das übrigens "Blocksatz".
Beachten Sie beim Eintippen von TURM VON HANOI unbedingt die unterschiedlichen Abstände zwischen den Worten der Spielerklärung.
In TURM VON HANOI begegnet uns bei der Aufforderung "Waehlen Sie nun die Anzahl der Scheiben (3–6)" zum ersten Mal der Fall, daß nicht nur ein einziges Zeichen blinkt, sondern gleich neun verschiedene auf einmal. Die Routine dazu liegt in

den Zeilen 8170 bis 8210. In Zeile 8170 werden die blinkenden Strings S$(0) und S$(1) festgelegt. Sie sehen, wie die vielen blinkenden Zeichen zustandekommen.
Das eigentliche Blinken geschieht in Zeile 8180 dadurch, daß die Variable D jeweils zwischen 0 und -1 wechselt ('D=NOTD'). Einmal wird somit S$(0) und ein anderesmal S$(1) angesprochen. Die nachfolgende Abfrage der Tastatur mit GET ist für uns nichts Neues mehr, wir haben schon darüber gesprochen.
Sie sehen, die Sache ist gar nicht so kompliziert, wie es vielleicht auf den ersten Blick erscheinen mag. Der einzige Trick dabei ist im Grunde, daß alle Zeichen in zwei Strings zusammengefaßt werden, und zwar einschließlich Cursorsteuerzeichen.
So, nun versuchen Sie einmal, das Rätsel des Turmes von Hanoi zu lösen. Es gibt eine ganz bestimmte Gewinnstrategie, die Sie herausfinden sollen. Probieren geht natürlich auch. Wenn Sie gar nicht zum Ziel kommen, lassen Sie es sich vom Computer vorführen. Sie erreichen das, indem Sie auf die Fragen "VON" und "NACH" mit "←" antworten. Das Programm zeigt Ihnen dann die Gewinnstrategie – vorausgesetzt, Sie haben die DATAs bei 10080..... richtig eingetippt.

Zusammenfassung:
Kapitel 17 präsentiert Ihnen ein Programm zum Training der Gehirnwindungen und folgenden Lehrstoff:
- "Selbstvernichtungsbefehle" NEW und SYS 64802
- Strukturierte Programmierung
- Übernahme von Bildern in Programme
- Blocksatz bei PRINT
- Blinken mehrerer Zeichen bei GET-Eingaben (großer Cursor)

```
1 REM TURM VON HANOI
2 REM """""""""""""""""
3 REM
4 REM  (C) 1982 BY
5 REM
6 REM ANDREAS DRIPKE
7 REM
8 REM
9 :
100 IFPEEK(158)OR191ANDSTTHENPRINT"■?LOAD ERROR";:END
110 TP=198:DIMM(3,6),P$(6),H(3),S$(1),D$(1):DO$="■":FORI=1TO22:DO$=DO$+"■
120 U$=U$+"─":RI$=RI$+"■":NEXT:AZ=0:X$="                                    ":POKE657,128
1000 REM------------
1010 GOSUB8000:REM ANLEITUNG
1020 GOSUB3000:REM SCHEIBEN DEFINIEREN
1030 GOSUB2000:REM FELD ZEICHNEN
1040 GOSUB6000:REM ANFANGSSTELLUNG
1050 GOSUB4000:REM EINGABE
1060 IFFTHENGOSUB10000:GOTO1100
1070 GOSUB5000:REM FELD AENDERN
1080 ONFGOSUB5050,5070:IFFGOTO1050
1090 GOSUB7000:REM FELD ZEICHNEN
1100 GOSUB9000:REM SPIELENDE?
1110 GOTO1050
1120 REM------------
2000 REM SPIELFELD
2010 PRINT"■*** TURM VON HANOI **■"CHR$(148)"*":PRINTLEFT$(U$,22)LEFT$(DO$,9)
2020 FORI=1TO21:PRINT"▄";:NEXT:PRINT
```

```
2030 PRINT"▞ 1▛ 2▛ 3▛ ":PRINTLEFT$(U$,21):PRINT"▗";:FORI=1TO19
2040 PRINT"─";:NEXT:PRINT"▖":PRINT"▚";:FORI=1TO19:PRINT"─";:NEXT:PRINT"▗▖"
2050 PRINT"| ZEIT 00:00:00 |▖▙▚▙▖    |":RETURN
3000 REM SCHEIBEN DEFINIEREN
3010 P$(1)="▞      ▀":P$(2)=" ▞     ▀":P$(3)=" ▞    ▀"
3020 P$(4)="  ▞   ▀":P$(5)="  ▞ ▀":P$(6)="   ▞ ▀":RETURN
4000 REM ZUGEINGABE
4010 F=0:PRINTLEFT$(DO$,15):PRINTX$:PRINTX$
4020 PRINT"▖IHR ZUG: VON _":T=13:GOSUB4130:IFG$=CHR$(20)THEN4010
4030 V=VAL(G$):PRINTLEFT$(DO$,16):IFF=2THEN4070
4040 PRINTTAB(15)"NACH _":T=20:GOSUB4130:N=VAL(G$):IFV=NORF=1THEN4000
4050 IFF=2THEN4070
4060 RETURN
4070 PRINTLEFT$(DO$,15):PRINTX$:PRINTX$
4080 PRINT"▖▞1▙▖▙= SCHNELL":PRINTSPC(3)LEFT$(U$,7)
4090 PRINT"▞2▙▖▙= EINZELSCHRITT":PRINTSPC(3)LEFT$(U$,13)
4100 POKETP,0:WAITTP,1:GETG$:IFG$="1"THENTS=1:RETURN
4110 IFG$="2"THENTS=2:RETURN
4120 GOTO4100
4130 PRINTLEFT$(DO$,16):D=NOTD:PRINTTAB(T)MID$("_¯",D+2,1):I=5
4140 GETG$:IFG$GOTO4160
4150 I=I-1:GOSUB7040:ONIGOTO4130:GOTO4140
4160 IFG$<>"1"ANDG$<>"2"ANDG$<>"3"ANDG$<>"←"ANDG$<>CHR$(20)THEN4130
4170 IFG$=CHR$(20)THENF=1:RETURN
4180 IFG$="←"THENF=2:RETURN
4190 PRINTLEFT$(DO$,16):PRINTTAB(T)"▞"G$"▙▖▙":RETURN
5000 REM MATRIX AENDERN
5010 F=0:IFH(V)=0THENF=1:RETURN
```

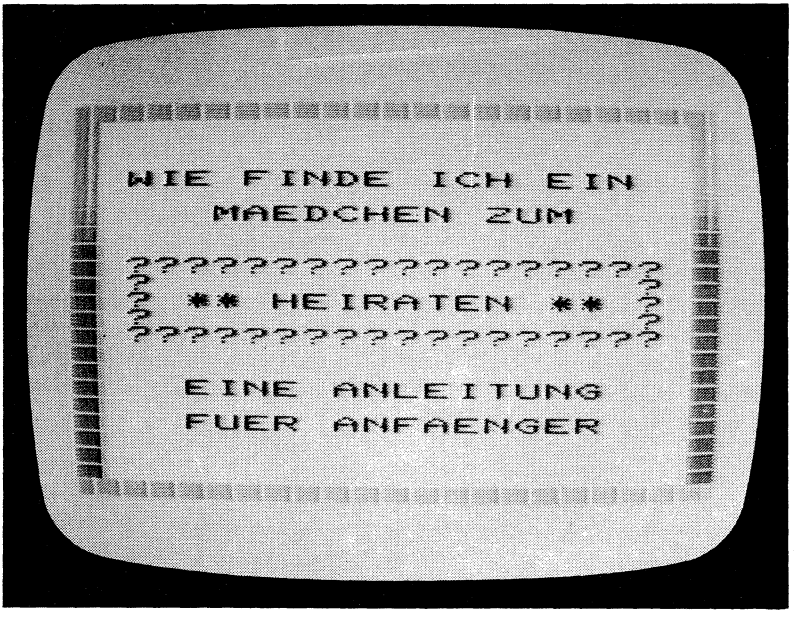
```
5020 IFM(V,H(V))<M(N,H(N))ANDM(N,H(N))>0THENF=2:RETURN
5030 IFH(3)=PTHENF=3:RETURN
5040 H(N)=H(N)+1:M(N,H(N))=M(V,H(V)):H(V)=H(V)-1:AZ=AZ+1:RETURN
5050 PRINTLEFT$(DO$,15):PRINTX$:PRINTX$:POKETP,0
5060 PRINT"■KEINE SCHEIBE AUF"V"■■."":FORI=1TO2E3:NEXT:RETURN
5070 PRINTLEFT$(DO$,15):PRINTX$:PRINTX$:POKETP,0
5080 PRINT"■"V"GROSSER ALS"N"■■!"":FORI=1TO2E3:NEXT:RETURN
6000 REM ANFANGSSTELLUNG
6010 FORI=1TOP:M(1,I)=I:NEXT:AZ=0:H(1)=P:H(2)=0:H(3)=0:PRINTLEFT$(DO$,8)
6020 FORI=1TOH(1):PRINTP$(I)"■■":NEXT:TI$="000000":C=0:RETURN
7000 REM VERAENDERUNGEN ZEICHNEN
7010 PRINTLEFT$(DO$,8-H(V)):PRINTTAB(V*7-7)"        "LEFT$(DO$,9-H(N))
7020 PRINTTAB(N*7-7)P$(M(N,H(N)))
7030 PRINTLEFT$(DO$,14)SPC(17)RIGHT$(" "+STR$(AZ),3):RETURN
7040 REM UHRZEIT PRINTEN
7050 PRINTLEFT$(DO$,14)SPC(7);
7060 IFT$=TI$THENRETURN
7070 T$=TI$:GOSUB11000
7080 PRINTLEFT$(T$,2)":"MID$(T$,3,2)":"RIGHT$(T$,2)"■":RETURN
8000 REM ANLEITUNG
8010 PRINT"■"CHR$(142)SPC(3)"...":PRINTSPC(4)".":PRINTSPC(4)".  . . .. ...
8020 PRINTSPC(4)".  . . .  ...":PRINTSPC(4)".  ... .  . .■■
8030 PRINTSPC(6)". . .. ..":PRINTTAB(7)".  .. ..■■
8040 PRINT".  .  ..  .  .  ..  ."
8050 PRINT".  . .  . .. . .  . .":PRINT".... .  . .  . .  . ."
8060 PRINT".  . .... . .. .  . .":PRINT".  . .  . .  . .  .■■■■
8070 FORI=0TO49:D=NOTD:PRINTTAB(T)"■"MID$("■■",D+2,1)"■   TURM VON HANOI   "
8080 POKE36878,15:POKE36876,192:POKE36876,254:POKE36878,0
```

```
8090 NEXT:PRINT"■ ** TURM VON HANOI **":PRINT"■"LEFT$(U$,20)
8100 PRINT"■VERSETZEN SIE DEN TUM■■R":PRINT"VON HANOI, DER ZU  BE-";
8110 PRINT"GINN BEI  '1'  STEHT.":PRINT"■DIE  SCHEIBEN  MUESSEN";
8120 PRINT"NUN AUF  '3'  GEBRACHT";:PRINT"WERDEN,  WOBEI '2' ALS";
8130 PRINT"ZWISCHEN-STATION   ZUR";:PRINT"VERFUEGUNG STEHT.
8140 PRINT"■ES DARF  NIEMALS  EINE";:PRINT"GROESSERE SCHEIBE  AUF";
8150 PRINT"EINE  KLEINERE  GELEGT";:PRINT"WERDEN.":PRINT"■ WAEHLEN SIE NUN DIE
8160 PRINT" ANZAHL DER SCHEIBEN":PRINT" (3-6)?■"
8170 S$(0)="\|/■■■■■■●■■■■■/|\■■■":S$(1)="    ■■■■■ o ■■■■■    ■■■
8180 PRINTTAB(9)S$(-D):D=NOTD:I=0:GOSUB11000
8190 GETG$:P=VAL(G$):IFP>2ANDP<7THEN8220
8200 I=I+1:IFI<15THEN8190
8210 GOTO8180
8220 PRINTTAB(10)"■■"G$"■■■":FORI=1TO1E3:NEXT:D$(0)="↑  ":D$(1)="  ↑
8230 PRINT"■■■ WENN SIE  SICH  BEIM":PRINT" SPIELEN VOELLIG VER-
8240 PRINT" ZETTELN  UND   NICHT":PRINT" MEHR WEITER  WISSEN,
8250 PRINT" DANN  DRUECKEN   SIE":PRINT" DIE TASTE '←'."
8260 PRINT"■ DAS PROGRAMM  FUEHRT":PRINT" IHNEN DANN DIE  LOE-
8270 PRINT" SUNG FUER 6 SCHEIBEN":PRINT" SELBSTAENDIG  VOR.
8280 PRINT"■■ ALLES KLAR? (J/N)■":POKETP,0
8290 PRINTTAB(14)"■"D$(-D):D=NOTD:I=0:GOSUB11000
8300 GETG$:IFG$="N"THENRUN
8310 IFG$="J"THENRETURN
8320 I=I+1:IFI<12THEN8300
8330 GOTO8290
9000 REM SPIELENDE?
9010 IF(H(1)>0)OR(H(2)>0)THENRETURN
9020 ONGOTO9030:PRINTLEFT$(DO$,15):PRINTX$:PRINTX$:PRINT"■GRATULIERE!
```

```
9030 PRINT"                    ":PRINT"  NOCHMAL? J/N  ":PRINT"                    "
9040 PRINTTAB(11)"  "D$(-D):D=NOTD:I=0:GOSUB11000
9050 GETG$:IFG$="J"THENRUN
9060 IFG$="N"THEN9090
9070 I=I+1:IFI<12THEN9050
9080 GOTO9040
9090 REM 'READY.'
9100 PRINT"  TURM VON HANOI - ENDE":POKETP,0:PRINT"READY."
9110 PRINT"  "MID$("  ",D+2,1)"  ":D=NOTD:I=0
9120 GETG$:IFG$<>""THEN9150
9130 I=I+1:IFI<16THEN9120
9140 GOTO9110
9150 SYS64802
10000 REM VORFUEHRUNG
10010 P=6:PRINT"  ":GOSUB2000:GOSUB6000:PRINTLEFT$(DO$,16):RESTORE:TI$="000000
10020 IFTS=1THENPRINT"TURM VON HANOI -":PRINT"SCHNELLDURCHLAUF
10030 IFTS=2THENPRINT"NAECHSTER SCHRITT?":PRINT"TASTE
10040 C=1:FORK=1TO63:READV,N:GOSUB5000:GOSUB7000:GOSUB7040
10050 IFTS=2THENPOKETP,0:WAITTP,1
10060 NEXT:PRINTLEFT$(DO$,17):PRINTX$:PRINTX$:PRINT"   SEHEN SIE, SO GEHT'S.
10070 RETURN
10080 DATA1,2,1,3,2,3,1,2,3,1,3,2,1,2,1,3,2,3,2,1,3,1,2,3,1,2,1,3,2,3,1,2,3,1
10090 DATA3,2,1,2,3,1,2,3,2,1,3,1,3,2,1,2,1,3,2,3,1,2,3,1,3,2,1,2,1,3,2,3,2,1
10100 DATA3,1,2,3,1,2,1,3,2,3,2,1,3,1,3,2,1,2,3,1,2,3,2,1,3,1,2,3,1,2,1,3,2,3
10110 DATA1,2,3,1,3,2,1,2,1,3,2,3,2,1,3,1,2,3,1,2,1,3,2,3
11000 POKE36878,15:POKE36876,128-D*8:FORJ=0TO9:NEXT:POKE36876,0:POKE36878,0
11010 RETURN
```

18. EHEVERMITTLER

Schon der Titel zeigt, daß das letzte Programm dieses Buches etwas ganz Besonderes ist. Keine Angst, Sie sollen nicht den Computer heiraten, aber er will Ihnen zu einem passenden Partner verhelfen.
EHEVERMITTLER ist ein Spielprogramm! Bitte fassen Sie es auch so auf! Sie brauchen also (ja sollen nicht einmal) die Fragen keinesfalls wahrheitsgetreu beantworten. Nehmen Sie auch die Antworten nicht allzu ernst.
Sollten Sie weiblichen Geschlechts sein, so möchte ich mich an dieser Stelle dafür entschuldigen, daß dieses Programm in der vorliegenden Fassung für Sie ungeeignet sein dürfte. Ich bin mir aber sicher, daß es für Sie ein leichtes sein wird, die einzelnen Texte entsprechend anzupassen.
Es gibt verschiedene Arten von Spielprogrammen, wie zum Beispiel Graphik- und Ratespiele. EHEVERMITTLER gehört nun in die Klasse der Textspiele.



Meist ist es sehr schwierig, ein vielfältiges Textspiel in nur 8 KByte unterzubringen. Jeder Buchstabe belegt nämlich ein Byte, und das summiert sich somit schnell. Hinzu kommen die

vielen Verzweigungen, denn für jede Antwort, die der Spieler gibt, muß das Programm entscheiden, was als nächstes geschehen soll. Sind nicht genügend Variationsmöglichkeiten gegeben, so wird das Spiel langweilig.
Eine Möglichkeit, ein Textspiel aufzubauen, besteht aus lauter IF-THEN-Abfragen und PRINT-Anweisungen. Das ist aber sehr aufwendig und daher für eine Speicherkapazität von nur 8 KByte ungeeignet.
EHEVERMITTLER ist aus diesem Grund anders programmiert. Die einzelnen Texte sind in DATA-Zeilen abgelegt und werden nach Bedarf eingelesen (Zeile 2010). Entscheidend aber ist, daß auch die IF-THEN-Abfragen durch DATA-Elemente ersetzt werden. Auf jeden unter DATA stehenden Text folgen drei Zahlen, die in Zeile 2010 mittels 'READ A,B,C' in Variablen gelesen werden. Dabei gibt A an, welchen Text das Programm als nächstes holen soll, falls der Spieler auf die soeben gestellte Frage mit "ja" antwortet und falls es sich gar nicht um eine Frage, sondern um einen Kommentar handelt. B enthält die Nummer des nächsten Textes für eine verneinende Antwort und C schließlich die Information, ob es sich um ein Frage oder einen Kommentar handelt (0 = Frage, 1 = Kommentar). Wie Sie sehen, sind alle wichtigen Informationen in DATA-Anweisungen enthalten, das Programm hat nur noch die Aufgabe, diese Informationen richtig zu verarbeiten. Nur so ist es möglich, mit relativ wenig Speicherkapazität ein doch recht umfangreiches Textspiel zu schreiben. Wenn Sie einmal selbst ein Textspiel programmieren möchten, dann sollten Sie sich dieser Methode erinnern.
Wir wollen uns noch einen ganz anderen Trick anschauen. Wenn Sie das "Titelblatt" zu EHEVERMITTLER eine Weile betrachten, dann werden Sie feststellen, daß die Umrahmung, die sich ständig neu zeichnet, ausschließlich aus Graphikzeichen besteht, und nicht zum Beispiel aus Buchstaben, Zahlen oder Sonderzeichen. Letzteres sähe ja auch wirklich nicht besonders elegant aus. Für diesen Umstand verantwortlich ist der erste Befehl in Zeile 1050.

```
X=64+64*RND(1)+128*(INT(2*RND(1)))
```

Die Formel sieht auf den ersten Blick kompliziert aus, wir wollen sie aber jetzt gemeinsam zerlegen. Nehmen wir zunächst nur den ersten Teil:

```
X=64+64*RND(1)
```

Hierdurch werden der Variablen X zufallsabhängig Werte zwi-

schen 64 und 127 zugewiesen. In diesem Bereich des Bildschirmcodes liegen ausschließlich Graphikzeichen. Die Buchstaben, Zahlen und Sonderzeichen befinden sich hingegen im Bereich von 0 bis 63, sie werden auf diese Weise also von vornherein ausgeschlossen.
Um den zweiten Teil der Formel zu verstehen, müssen Sie folgendes wissen. Zwischen den normalen Zeichen und deren reversen Darstellungen besteht ein Zusammenhang im Bildschirmcode. Wir erhalten den Code des RVS-Zeichens, indem wir zu dem Normalcode noch 128 hinzuaddieren. Das Ergebnis darf nicht größer als 255 sein. Sehen Sie sich

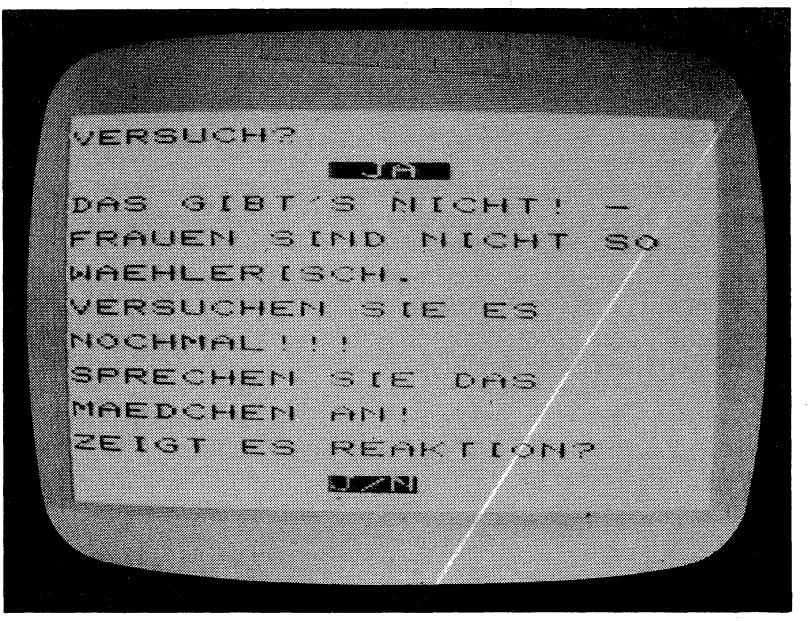
```
X= (64 . . 127) + 128*(INT(2*RND(1)))
```

daraufhin an. Mit

```
INT(2*RND(1))
```

werden Zufallszahlen von 0 bis 1 erzeugt. Durch INT werden Nachkommastellen abgeschnitten, so daß entweder genau 0 oder genau 1 das Ergebnis sein kann. Dieses wird nun der Formel entsprechend mit 128 multipliziert. Dabei kann das Ergebnis entweder genau 0 oder genau 128 sein. Einleuchtend, nicht wahr? Dieses Ergebnis wird nun zu dem Bildschirmcode – der zwischen 64 und 127 liegt – dazuaddiert. Bei 0 bleibt das Zeichen also in Normaldarstellung, bei 128 dagegen wird es revers. Auf diese Art und Weise erscheinen im Rahmen alle Graphikzeichen sowohl normal als auch revers. Und das alles mit einer einzigen Formel.
Im letzten Kapitel haben wir bereits einen SYS-Befehl kennengelernt, mit dem der Computer zurückgesetzt werden kann in einen Zustand, als ob er gerade eingeschaltet worden wäre (man nennt dies auch "reset"). Auch in EHEVERMITTLER finden wir diesen Befehl in Zeile 3020. Wir wollen noch ein paar weitere SYS-Adressen kennenlernen, die von Interesse sind:

Bildschirm löschen	58719
LOAD	65493
SAVE	65496
VERIFY	65499
CLR	50784
Löschen des Stacks	50810



Die relativ geringe Bildschirmbreite von 23 Zeichen pro Zeile bringt es mit sich, daß manche der unter DATA abgelegten Texte nicht in eine einzige Zeile passen. Nun sähe es unschön aus, wenn diese Sätze einfach in die nächste Bildschirmzeile überlappten, weil dies in den allermeisten Fällen die Trennung direkt innerhalb eines Wortes bedeuten würde. In EHEVERMITTLER wird das vermieden, indem jeder Satz auf seine Länge überprüft wird und gegebenenfalls korrekt auf zwei oder sogar drei Bildschirmzeilen aufgeteilt wird. Die Routine, die dies besorgt, befindet sich in den Zeilen 2020 bis 2040. In I wird die maximale Bildschirmbreite mit 23 festgelegt (Zeile 2020), der Text, der insgesamt ausgegeben werden soll, befindet sich in der Variablen A$. B$ ist lediglich eine Hilfsvariable. Wie die kurze Routine im einzelnen funktioniert, wollen wir hier nicht besprechen – weil, ja weil Sie sie nach der Lektüre des vorliegenden Buches alleine verstehen sollten. Verstehen Sie dies ruhig als Aufforderung!
Ja, damit sind wir (fast) am Ende des Buches angelangt. Es folgen noch ein paar Anhänge, die Sie auch einmal durchlesen sollten, falls Sie es noch nicht getan haben.
Ich hoffe, daß Ihnen die Programme gefallen und Freude bereitet haben. Zusätzlich zu der beträchtlichen Erweiterung Ihrer Programmbibliothek haben wir auch eine Menge Dinge miteinander besprochen. Wenn Sie all die Tips und Tricks in Ihren

eigenen Programmen mit berücksichtigen, dann hat dieses Buch seinen Zweck erfüllt.
Vielleicht begegnen wir uns ja noch einmal in dem einen oder anderen Band der INTERFACE AGE (COMPUTER LIFE) "CBM/VC-20 Buchreihe", ich würde mich auf jeden Fall darüber freuen. Sie sollten sich auch einmal die am Ende dieses Buches abgedruckte Werbung für die Standard-Basicerweiterung EXBASIC LEVEL II ansehen. Na, wäre das nicht etwas für Sie?
Und nun zum Schluß wie immer die

Zusammenfassung:
Das letzte Kapitel brachte ein Textspiel und Ihnen (wie ich hoffe) folgendes Wissen:
- Methode zur Programmierung von Textprogrammen
- Zusammenhang zwischen Normal- und Reverszeichen im Bildschirmcode
- interessante SYS-Adressen
- automatische Texttrennung für zu lange Sätze

```
1 REM"***************
2 REM
3 REM EHEVERMITTLER
4 REM
5 REM COPYRIGHT 1982
6 REM
7 REM ANDREAS DRIPKE
8 ;
100 VB=PEEK(648)*256:VC=4*(PEEK(36866)AND128)+37888
110 IFPEEK(158)OR191ANDSTTHENPRINT"■?LADEFEHLER";:END
1000 POKE657,128:POKE36879,123:PRINTCHR$(142)"■■■■■  WIE FINDE ICH EIN
1010 PRINTSPC(5)"■MAEDCHEN ZUM■■":PRINT"  ??????????????????
1020 PRINT"  ?"SPC(16)"?":PRINT"  ? ** HEIRATEN ** ?":PRINT"  ?"SPC(16)"?
1030 PRINT"  ??????????????????":PRINTSPC(4)"■■EINE ANLEITUNG
1040 PRINTSPC(4)"■FUER ANFAENGER
1050 X=64+64*RND(1)+128*(INT(2*RND(1))):Y=8*RND(1):FORI=0TO21:POKEVB+I,X
1060 POKEVC+I,Y:POKEVB+I+484,X:POKEVC+I+484,Y:NEXT:FORI=22TO462STEP22
1070 POKEVB+I,X:POKEVC+I,Y:POKEVB+I+21,X:POKEVC+I+21,Y:NEXT
1080 GETG$:IFG$=""GOTO1050
2000 POKE36879,27:PRINT"■■";:X=25
2010 N=1:RESTORE:FORI=1TOX+72:READA$:NEXT:READA,B,C:IFC=0THENA$=A$+"?
2020 PRINT:I=23
2030 I=I-1:B$=MID$(A$,I+(N-1),1):IFB$<>""ANDB$<>" "GOTO2030
2040 PRINTMID$(A$,N,I-1):IFB$=" "THENN=N+I:GOTO2020
2050 IFA=888GOTO3030
2060 IFA=999GOTO3000
2070 IFC=1THENX=A:GOTO2010
2080 PRINT:PRINT
```

```
2090 PRINTSPC(9)"■■ /N":FORI=0TO299:NEXT:PRINTSPC(9)"■■J/ ":FORI=0TO299:NEXT
2100 GETG$:IFG$="J"THENPRINTSPC(9)"■■ JA ":X=A:GOTO2010
2110 IFG$="N"THENPRINTSPC(9)"■■NEIN":X=B:GOTO2010
2120 GOTO2090
3000 FORI=1TO3E3:NEXT:PRINT"■DRUECKEN SIE JETZT":PRINT"■'STOP', ";
3010 FORI=0TO999:NEXT:PRINT"SONST IST":PRINT"■GLEICH ALLES ZU":PRINT"■SPAET!
3020 POKE657,0:FORI=1TO3E3:NEXT:SYS64802
3030 FORI=1TO3E3:NEXT:PRINT"■■■■■■■  HERZLICHEN GLUECK-
3040 PRINT"■ WUNSCH ZUR HOCHZEIT!":RESTORE:POKE36878,8:T=36876:FORI=0TO35
3050 READA,B:POKET,A:FORJ=1TOB*150:NEXT:POKET,0:NEXT:POKE36878,0:POKE657,0:END
3060 DATA135,4,163,3,163,1,163,8,135,4,175,3,159,1,163,8,135,4,163,3,183,1,195
3070 DATA4,183,3,163,1,163,4,159,3,163,1,175,8,135,4,163,3,163,1,163,8,135,4
3080 DATA175,3,159,1,163,8,135,4,163,3,183,1,195,4,183,3,163,1,147,4,175,3,159
3090 DATA1,163,8
4000 DATASCHEREN SIE SICH ZUM TEUFEL!,999,,1
4010 DATAWANDERN SIE AUS IN LAENDER MIT VIELWEIBEREI!!!,999,,1
4020 DATAWARTEN SIE BIS SIE ETWAS AELTER SIND!,999,,1
4030 DATABEGINNEN SIE NOCHEIMAL VON VORNE!,999,,1
4040 DATAWARUM LESEN SIE DANN DIESE ANLEITUNG ?!?!,1,,1
4050 DATAWOLLEN SIE NOCHEINMAL HEIRATEN?,5,17,0
4060 DATASIND SIE VERHEIRATET,21,49,,SEHR SCHUECHTERN?,9,57,
4070 DATASIE HABEN DIE WAHL ZWISCHEN STRASSENBAHN UND UNKONSERVATIV.,57,,1
4080 DATAETWAS MUESSEN SIE SCHON INVESTIEREN!,33,,1
4090 DATA"DIESER WEG IST MUEHSAM, ABER VERSUCHEN SIE ES!",65,,1
4100 DATAGEHEN SIE ZUR HEIRATSVERMITTLUNG!,41,,1,SIND SIE AELTER ALS 40,45,53
4110 DATA,SIND SIE SCHUECHTERN,29,73,,SAMMELN SIE MUTRESERVEN!!!,73,,1
4120 DATABESUCHEN SIE EINE TANZSTUNDE,81,,1
4130 DATANUTZEN SIE DIE FREIE AUSWAHL DER DAMENKARTEI.,89,,1
```

```
4140 DATA** SIE GLUECKSPILZ **,113,,1,SIND SIE KONSERVATIV,77,69,
4150 DATAFAHREN SIE STRASSENBAHN,97,81,,KOENNEN SIE TANZEN,101,85,
4160 DATAWOLLEN SIE TANZEN LERNEN,61,37,,SCHON WAS GEFUNDEN,109,65,
4170 DATAWECHSELN SIE DIE GEGEND!,97,,1,WOHNEN SIE IN REGENARMER GEGEND,93
4180 DATA117,,SUCHEN SIE EIN TANZLOKAL!,121,,1,DIE ANTWORT WAR ZU ERWARTEN.
4190 DATA141,,1,GEHEN SIE ZUM VEREINBARTEN TREFF!,129,,1
4200 DATASUCHEN SIE SICH IRGENDWO EIN MAEDCHEN AUS!,161,,1
4210 DATA BIETEN SIE BEI REGEN EINEM MAEDCHEN IHREN SCHIRM AN.,161,,1
4220 DATAFINDEN SIE EINS,137,125,,SIND SIE ETWA STUDENT,105,141,
4230 DATAERKENNUNGSZEICHEN NICHT VERGESSEN!!!,145,,1
4240 DATASUCHEN SIE EIN MAEDCHEN AN DER BAR!,161,,1,TANZEN SIE GUT,153,133,
4250 DATALASSEN SIE SICH AUF TANZFESTE MITNEHMEN Z.B. AUF HOCHZEITEN.,137,,1
4260 DATAKOMMT SIE ?????,149,65,,GRAUST'S IHNEN BEIM ERSTEN ANBLICK,65,161,
4270 DATAFORDERN SIE EIN MAEDCHEN AUF!,161,,1
4280 DATASIE SOLLTEN SICH NICHT GAR ZU DAEMLICH ANSTELLEN!,45,,1
4290 DATASPRECHEN SIE DAS MAEDCHEN AN!,189,,1,VERSUCHEN SIE ES NOCHMAL!!!,161
4300 DATA,1,DAS GIBT'S NICHT! - FRAUEN SIND NICHT SO WAEHLERISCH.,165,,1
4310 DATAHIER LOHNT ES SICH NICHT!,133,,1,GEBEN SIE'S AUF!,157,,1
4320 DATAIST DAS SCHON DER 4. VERSUCH,177,185,0
4330 DATAPROBIEREN SIE ES NOCH EINMAL!,161,,1,ZEIGT ES REAKTION,193,181,
4340 DATAANGENEHME REAKTION,213,197,,GAB ES OHRFEIGEN,217,201,
4350 DATASCHNIPPISCHE ABFUHR,221,241,
4360 DATA"ESSEN SIE MEHR JOGHURT, DAS GIBT KRAFT!",209,,1
4370 DATAACHTEN SIE IN ZUKUNFT AUF MAUERBLUEMCHEN!!!,45,,1
4380 DATAVORSICHT!! DAS KANN EINE FALLE SEIN!,241,,1,DAS IST FAST NORMAL.,241
4390 DATA,1,IST DAS SCHON DER 4. VERSUCH,169,217,
4400 DATASCHALTEN SIE DIE KONKURRENZ PHYSISCH AUS!,249,,1,SIND SIE STARK,225
4410 DATA253,,DAS MACHT DEN FALL NUR INTERESSANTER.,229,,1
```

```
4420 DATASCHALTEN SICH NEBENBUHLER EIN,233,257,,BALZEN SIE AUS LEIBESKRAEFTEN
4430 DATA237,,1,BLEIBEN SIE AM BALL MIT WEITEREN VERSUCHEN!!!,221,,1
4440 DATAGELINGT ES,265,205,,SCHALTEN SIE DIE KONKURRENZ PSYCHISCH AUS!,249,
4450 DATA1,BITTEN SIE UM EIN WIEDERSEHEN!,261,,1,LEHNT SIE AB,245,277,
4460 DATASIE HABEN SCHON FAST GEWONNEN!,257,,1,SIE MUESSEN ABER!!!
4470 DATA285,,1,ARRANGIEREN SIE HAEUFIG TREFFS!,289,,1
4480 DATAEIGENTLICH KANN JETZT NICHTS MEHR PASSIEREN.,273,,1
4490 DATAKOENNEN SIE SICH FUER EINE ENTSCHEIDEN,321,269,
4500 DATADURCH PARALLELEN EINSATZ ENTSCHEIDUNG VORBEREITEN!,281,,1
4510 DATAKENNEN SIE NOCH ANDERE MAEDCHEN,285,293,
4520 DATAMAN SOLLTE ABER MEHRERE EISEN IM FEUER HABEN!,297,,1
4530 DATAHABEN SIE DIE RICHTIGE GEFUNDEN,321,301,
4540 DATADANN HABEN SIE IRGENDWO EIN FEHLER GEMACHT!!!,45,,1
4550 DATAGEHEN SIE ZUM FRISEUR!,333,,1,DRACHENFUTTER NICHT VERGESSEN!,333,,1
4560 DATAANSTANDSBESUCH MACHEN.,309,,1
4570 DATAKENNEN SIE SCHON DIE ELTERN,333,313,0,SIND SIE KONSERVATIV,317,325,
4580 DATASCHLAGEN SIE IHR EINE GEMEINSAME WOHNUNG VOR!,345,,1
4590 DATATRAGEN SIE LANGE HAARE,305,353,
4600 DATASIND DIE ELTERN UMGAENGLICH,337,329,
4610 DATASIE HABEN WICHTIGE VERBUENDETE GEWONNEN!,341,,1
4620 DATALOBEN SIE DIE VORZUEGE EINER EHE!,385,,1
4630 DATAEMPOERTE ABLEHNUNG,349,365,,DAS IST NICHTS FUER SIE!,45,,1
4640 DATADANN VERSUCHEN SIE ES OHNE ELTERLICHEN SEGEN!,357,,1
4650 DATAABER ERWARTEN SIE ZUSAETZLICHE SCHWIERIGKEITEN!,341,,1
4660 DATAVERSUCHEN SIE ES WEITER!!!,341,,1
4670 DATAZIEHEN SIE MIT IHR ZUSAMMEN!,389,,1
4680 DATASCHNELL AUSSTEIGEN UND VON VORNE ANFANGEN!!!,49,,1
4690 DATADAS WAR EINE FALLE FUER LIEBHABER!,369,,1
```

```
4700 DATA+++++ SIE PECHVOGEL +++++,373,,1
4710 DATAIST DAS MAEDCHEN AELTER ALS 40,377,361,
4720 DATAFINDET DIE LOBREDE BEACHTUNG,405,381,
4730 DATAHEIRATEN WOLLEN DANN ALLE FRAUEN IRGENDWANN.,409,,1
4740 DATASIE HABEN ALS KONSERVATIVER GUT DURCHGEHALTEN.,888,,1
4750 DATASCHNELL ZUGREIFEN!!!,393,,1,LEHNT SIE GELANGWEILT ODER VERAERGERT AB
4760 DATA381,397,,MACHEN SIE IHR EINEN HEIRATSANTRAG!,401,,1
4770 DATA"SIE HATTEN LEICHTES SPIEL, DAS PASST ZU IHNEN!!!",888,,1
```

ANHANG A
Wie lese ich ein Computer-listing

Alle 18 Spiele dieses Buches liegen als Listings vor. Um sie in den Computer hineinzubringen, haben Sie nur eine Möglichkeit: Eintippen! Dann kann zwar schon ziemlich mühselig werden, aber dafür haben Sie nach zwei Stunden Tipparbeit auch eine ganz andere Beziehung zu dem Spiel als nach zwei Minuten von Kassette oder zwei Sekunden von Diskette laden.
Immerhin, wenn Sie auf die Tipperei verzichten wollen, können Sie alle 18 Spielprogramme auch fertig auf Kassette beim Verlag beziehen. In diesem Fall dürfte der Rest dieses Anhangs für Sie kaum noch von Interesse sein.
Wir wollen uns nämlich jetzt darüber unterhalten, wie so ein Computerlisting überhaupt zu lesen ist. Sicher, meist ist die Sache ganz klar, da steht 'PRINT''ROULETTE'' ' und Sie tippen ein 'PRINT''ROULETTE'' '. Kein Problem, nicht wahr?
Aber die Listings enthalten auch eine Reihe von Steuerzeichen, die durch Reversdruck hervorgehoben sind. Diese Steuerzeichen sind im einzelnen:

- Bildschirm löschen (SHIFT + CLR)
- Cursor Home (HOME)
- Cursor nach rechts (CRSR)
- Cursor nach links (SHIFT + CRSR)
- Cursor nach unten (CRSR)
- Cursor nach oben (SHIFT + CRSR)
- Reversmodus ein (CTRL + RVS ON)
- Reversmodus aus (CTRL + RVS OFF)
- Cursorfarbe schwarz (CTRL + BLK)
- Cursorfarbe weiß (CTRL + WHT)
- Cursorfarbe rot (CTRL + RED)

 Cursorfarbe zyan (CTRL + CYN)

 Cursorfarbe purpurn (CTRL + PUR)

 Cursorfarbe grün CTRL + GRN)

 Cursorfarbe blau (CTRL + BLU)

 Cursorfarbe gelb (CTRL + YEL)

Wenn nun im Listing ein solches Steuerzeichen auftritt, dann drücken Sie einfach die entsprechende Taste, gegebenen falls zusammen mit SHIFT oder CTRL. Versuchen Sie nicht, das ebenfalls so aussehende Graphikzeichen hinzusetzen.
An einem Beispiel wollen wir uns das klarmachen. Nehmen wir einmal an, in einem Listing finden Sie folgende Zeile:

```
8450 PRINT"■■■ICH SPIELE AUCH NICHT MEHR MIT IHNEN!"
```

Sie geben jetzt zuerst die Zeilennummer 8450 ein, danach ein Leerzeichen (SPACE-Taste), dann PRINT (oder als Abkürzung einfach ein Fragezeichen). Nun kommen die entscheidenden Schritte:
Drücken Sie das Anführungszeichen und dann SHIFT und CLR gleichzeitig. Mit 'CLR' ist die Bildschirmlöschtaste gemeint. Wenn Sie alles richtig gemacht haben, erscheint nun ein reverses Herzchen (Graphikmodus) bzw. ein großes reverses "S" (Groß-/Kleinschreibmodus).
Ist das Herzchen oder "S" zu sehen und somit alles in Ordnung, dann tippen Sie jetzt zweimal " Cursor nach unten". Das ist die Taste 'CRSR '. Diesmal darf SHIFT nicht gedrückt werden. Neben dem Herz bzw. "S" müßten jetzt zwei reverse "Q" stehen. Ja? Fein, dann haben Sie alles richtig gemacht.
Der Rest ist einfach, Sie geben die Zeile so weiter ein, wie es im Listing steht ("ICH SPIELE..."). Klar!
Beachten Sie, daß die reversen Steuerzeichen nichts mit den genauso aussehenden Graphikzeichen zu tun haben. Die Computerentwickler bei Commodore hatten lediglich keine anderen Zeichen mehr übrig.
Wir haben schon gesehen, daß man PRINT auch durch '?' ersetzen kann. Die Eingabe wird dadurch erleichtert. Für das Programm selbst hat dies keinerlei Auswirkungen, da die Fragezeichen alle vom Computer automatisch in PRINT umgewandelt werden (nur im Programm, nicht bei direkter Eingabe). Um das zu überprüfen, geben Sie eine BASIC-Zeile mit Fragezeichen anstelle von PRINT ein und listen Sie die Zeile danach

mit LIST auf. Wo bei der Eingabe '?' stand, steht nun PRINT. Ebenso druckt der Computer beim Listen zwischen der Zeilennummer und dem ersten BASIC-Befehl immer ein Leerzeichen. Beim Eingeben kann man diesen Zwischenraum also weglassen, der Computer fügt ihn automatisch ein.
Wir können nicht nur PRINT abkürzen, sondern auch alle anderen BASIC-Befehle. In den Listings sind selbstverständlich keine Abkürzungen verwandt, aber bei der Eingabe können Sie diese nutzen. Untenstehende Abkürzungen sind unabhängig von der Betriebsart Graphikzeichen oder Kleinschreibung. Lediglich auf dem Bildschirm erscheint verschieden.

BASIC-Wort	Abkürz.	BASIC-Wort	Abkürz.
ABS	Ab	NOT	No
AND	An	OPEN	Op
ASC	As	PEEK	Pe
ATN	At	POKE	Po
CHR$	Ch	PRINT	?
CLOSE	Clo	PRINT#	Pr
CLR	Cl	READ	Re
CMD	CM	RESTORE	REs
CONT	Co	RETURN	REt
DATA	Da	RIGHT$	Ri
Def	De	RND	Rn
DIM	Di	RUN	Ru
END	En	SAVE	Sa
EXP	Ex	SGN	Sg
FOR	Fo	SIN	Si
FRE	Fr	SPC(	Sp
GET	Ge	SQR	Sq
GOSUB	GOs	STEP	STe
GOTO	Go	STOP	St
INPUT	In	STR$	STr
LEFT$	LEf	SYS	Sy
LET	Le	TAB(	Ta
LIST	Li	THEN	Th
LOAD	Lo	USR	Us
MID$	Mi	VAL	Va
NEXT	Ne	VERIFY	Ve

Zu den Abkürzungen: Die Buchstaben, die in obiger Liste groß geschrieben sind, müssen ohne SHIFT eingegeben werden, ganz gleich, wie sie dann auf dem Bildschirm erscheinen. Die Kleinbuchstaben dagegen werden mit SHIFT eingetippt, ebenfalls unabhängig davon, was dadurch auf dem Bildschirm dargestellt wird.
Einen Rat möchte ich Ihnen noch geben, den Sie zu ihrem Vorteil beherzigen sollten: Wenn Sie ein Programm vom Listing in den Computer fertig eingetippt haben, sichern Sie dieses unbedingt erst mit SAVE auf einer Kassette oder Diskette, bevor Sie es mit RUN starten! Ein Fehler beim Eingeben des Programmes kann leicht dazu führen, daß das Programm nach RUN "abstürzt" und dadurch verlorengeht. Oder TURM VON HANOI und EHEVERMITTLER haben eine "Selbstvernichtungsautomatik" eingebaut, die unter bestimmten Umständen anspricht.
Deswegen: ERST SICHERN DANN STARTEN! Denn ein Programm zweimal hintereinander einzugeben ist eine der langweiligsten Sachen der Welt.

ANHANG B – Commodore-BASIC

Dieser Anhang bietet Ihnen eine vollständige Auflistung aller BASIC-Kommandos, die Ihnen Commodore-BASIC zur Verfügung stellt. Die Computer der Serien CBM 4001 und CBM 8001 weisen einige Befehle auf, die im Zusammenhang mit Floppyoperationen stehen und die die Computer CBM 2001, CBM 3001 und VC 20 nicht zur Verfügung stellen. Darauf ist bei den jeweiligen Befehlen hingewiesen. Steht kein entsprechender Vermerk, so funktioniert der Befehl auf allen CBM-Computern und auf dem VC 20.
Die Liste ist sowohl zum Durchlesen wie auch zum Nachschlagen gleichermaßen gut geeignet.

ABS bildet den Absolutwert einer Zahl.

```
ABS(8) = 8
ABS(-8) = 8
ABS(-8.2) = 8.2
```

AND 1. dient zur Verbindung mehrerer Bedingungen in IF THEN-Anweisungen mit "UND"

```
120 IF X=1 AND Y=1 THEN 1000
```

2. erlaubt die Boole'sche Operation AND durchzuführen.

```
? 25  AND 12
```

AND wandelt seine Argumente in 16 Bit Binärzahlen um und führt die Boole'sche Operation bitweise durch. Der zulässige Zahlenbereich liegt zwischen –32767 bis +32767. Die Boole'sche Operation AND ist folgendermassen definiert:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

3. ist geeignet für Bitmanipulationen einzelner Speicherzellen des Computers.

APPEND ist nur bei Computern der Serien CBM 4001 und 8001 verfügbar.
APPEND ermöglicht es, eine serielle Datei auf Diskette zu erweitern. APPEND wirkt wie DOPEN auf ein sequentielle Datei, kann aber nur auf eine schon bestehende Datei angewandt werden. Es öffnet diese und setzt den Schreibzeiger auf das Dateiende. Die nächste Schreiboperation auf diese Datei schreibt also ab dem Dateiende weiter. Damit besteht die Möglichkeit, zusätzliche Daten in eine schon geschlossene serielle Datei (SEQ, PRG, USR) zu schreiben.
Beispiel:

```
APPEND #2,"DATEI",D1
```

An die Datei mit dem Namen 'DATEI' mit der logischen Adresse 2 auf Laufwerk 1 sollen weitere Daten angehängt werden.

ASC bringt zu einem Zeichen den ASCII-Code (ASCII = American Standard Code for Information Interchange).

```
ASC("X") = 88
```

Wenn der String mehrere Zeichen lang ist, dann nimmt ASC nur das erste Zeichen.

```
ASC("XYZ") = 88
```

Statt des Strings kann auch eine Stringvariable benutzt werden.

```
ASC(A$)
```

mit

```
A$="XYZ"
```

liefert auch den Wert 88.
Die Umkehrfunktion zu ASC lautet CHR$.

ATN ist die Arkustangensfunktion. S. TAN/ATN

BACKUP ist ein Befehl der CBM 4001 und CBM 8001 Computer. BACKUP dupliziert eine Diskette.

```
BACKUP D0 TO D1
```

dupliziert die Diskette in Laufwerk 0 auf die in Laufwerk 1. Es wird Block für Block übertragen.

CATALOG steht ausschließlich bei den Computern CBM 4001 und CBM 8001 zur Verfügung.
CATALOG listet das Inhaltsverzeichnis von Disketten auf.

```
CATALOG D1
```

listet das Verzeichnis von Laufwerk 1.

```
CATALOG
```

listet beide Laufwerke, sofern Disketten darin stecken.

CHR$ ist die Umkehrfunktion zu ASC. Zu einem Zahlenwert zwischen 0 und 255 wird das im ASCII-Code dazugehörige Zeichen dargestellt.

```
CHR$(65) = "A"
```

Dementsprechend ist

```
ASC("A") = 65
```

CLOSE schließt eine offene Datei. S. OPEN/CLOSE

CLR löscht sämtliche Variablen, Variablenfelder, FOR-NEXT-Zusammengehörigkeiten und Unterprogrammbeziehungen.

CMD dient der Ein- und Ausgabe von Peripheriegeräten, wie z. B. Floppy-Disk oder Drucker. Durch CMD werden sämtliche BASIC-Befehle auf dieses Peripheriegerät bezogen. Vor CMD muß ein entsprechendes File mit DOPEN/OPEN eröffnet worden sein.

```
OPEN 4,4 : CMD 4 : LIST
PRINT #4
CLOSE 4
```

Dieses Beispiel listet ein im Computer befindliches Programm auf einem Drucker aus.

COLLECT ist ein Kommando der Serien CBM 4001 und CBM 8001. COLLECT hat auf eine Diskette folgende Wirkung: Alle eventuell offenen Dateien werden geschlossen, die Diskette wird von mit einem Stern im Inhaltsverzeichnis gekennzeichneten Dateien gesäubert.

CONCAT funktioniert nur auf den Serien CBM 4001 und CBM 8001. Mit dem Befehl CONCAT können mehrere Dateien des gleichen Typs auf Diskette zu einer einzigen Datei zusammengefaßt werden. Beide Dateinamen dürfen nicht länger als 20 Zeichen sein.
Die Zieldatei, also die Datei, an die eine andere angehängt wird, wird dadurch verändert, sie existiert nach dem Befehl CONCAT nicht mehr in ihrer ursprünglichen Form. Die angehängte Datei dagegen bleibt weiterhin zusätzlich in ihrer alten Form auf Diskette erhalten.

```
CONCAT "DAT1",D0 TO "DAT2",D1
```

verbindet 'DAT1' mit 'DAT2'. Dabei steht 'DAT1' auf Laufwerk 0 und 'DAT2' auf Laufwerk 1.

CONT steht für "continue" (= weitermachen, fortfahren). Wird ein Programm durch END oder STOP im Programm selbst oder durch Drücken der STOP-Taste angehalten, dann kann die Programmausführung mit CONT wieder fortgesetzt werden. Dazwischen dürfen Variablen verändert oder ausgedruckt werden, es darf aber keine Programmzeile geändert werden.
Andernfalls kommt es zur Fehlermeldung ?CAN'T CONTINUE ERROR. Auch wenn ein Programm durch Drücken der RETURN-Taste ohne Eingabe bei einem INPUT-Befehl verlassen wird, kann es mit CONT fortgesetzt werden.

COPY ist ein Kommando der Serien CBM 4001 und CBM 8001. Mit COPY lassen sich Files aller Art von einer Diskette auf eine andere kopieren.

```
COPY "EINGABE",D0 TO "EINGABE 1",D1
```

kopiert 'EINGABE' von Laufwerk 0 nach Laufwerk 1 und ändert gleichzeitig den Namen in 'EINGABE 1'.

Wird kein Filename angegeben, so wird die ganze Diskette kopiert.

```
COPY D0 TO D1
```

kopiert die Diskette in Laufwerk 0 auf Laufwerk 1.

COS bezeichnet die Cosinus-Funktion. Der Winkel ist in Bogenmaß anzugeben. Altgradangaben werden folgendermaßen umgerechnet:

$$X(\text{Bogenmaß}) = \frac{\pi}{180} \alpha \text{ (Altgrad)}$$

Die Arkuscosinus-Funktion ist in Commodore-BASIC nicht implementiert. Sie kann folgendermaßen ausgedrückt werden:

$$\text{ARCCOS}(X) = \text{ATN}(\text{SQR}(1-X*X)/X) \text{ mit } 0 < X \leqslant 1$$

DATA/READ/RESTORE sind Anweisungen zur Datenspeicherung innerhalb von Programmen. Die Daten werden mit Kommata getrennt unter DATA abgelegt und können mit READ in Variablen eingelesen werden.
Unter DATA dürfen Zahlen und Zeichen stehen. Wenn ein String geshiftete Zeichen, Doppelpunkte oder Kommata oder führende Leerzeichen enthält, muß er in Anführungszeichen stehen.

```
250 DATA "10:5=2","A,B,C","  HALLO "
```

RESTORE dient dem mehrmaligen Einlesen mit READ. Nach einem RESTORE-BEfehl ist das nächste Datenelement, das eingelesen wird, das erste Elemente der DATA-Liste.

DCLOSE steht auf den Computern CBM 4001 und CBM 8001 zur Verfügung.
DCLOSE schließt eine offene Disketten-Datei.
S. DOPEN/DCLOSE

DEFFN/FN erlauben eigene Funktionen zu definieren.

```
1 DEF FNAC(X)=ATN(SQR(1-X*X)/X)
```

definiert die Arkuscosinus-Funktion.

Diese kann nun mit z. B.

```
? FNAC(.2)
```

aufgerufen werden.
Die Funktionsdefinition muß in eine Programmzeile passen. Sie darf außerdem nur ein Argument (im Beispiel X) haben. Stringfunktionen können mit DEFFN nicht definiert werden.

DIM dimensioniert Variablenfelder.

```
10 DIM X(10,10,2)
```

Durch DIM wird für die Datenmenge eines Feldes genügend freier Raum im Computerspeicher geschaffen.
Werden in einem Programm dimensionierte Variablen angesprochen, ohne vorherige explizite Dimensionierung, dann wird ein der Form entsprechendes Feld mit Indizes von 0 bis 10 implizit festgelegt.

DIRECTORY ist ein Kommando der Computer CBM 4001 und CBM 8001. DIRECTORY bewirkt exakt dasselbe wie CATALOG. S. CATALOG

DLOAD steht nur bei den Serien CBM 4001 und CBM 8001 zur Verfügung.
DLOAD dient dazu, Programme von Diskette zu laden.

```
DLOAD "NAME",D0
```

lädt das Programm 'NAME' von Laufwerk 0.

```
DLOAD "NAME",D1
```

lädt das Programm 'NAME' von Laufwerk 1.
Wird kein Laufwerk spezifiziert, so nimmt der Computer Laufwerk 0 als Ersatzwert an. Mit

```
DLOAD "*"
```

wird das erste Programm von Diskette in Laufwerk 0 geladen.
Wird das gesuchte Programm nicht gefunden, kommt es zur Meldung ?FILE NOT FOUND ERROR.

DOPEN/DCLOSE sind die analogen Befehle zu OPEN/CLOSE zur Behandlung von Dateien auf Disketten. Mit DOPEN wird eine Datei geöffnet, mit DCLOSE wieder geschlossen. In eine offene Datei können Daten hineingeschrieben werden. Zum Schreiben dienen die Befehle PRINT#, zum Lesen INPUT# oder GET#.
DOPEN und DCLOSE gibt es nur bei den Serien CBM 4001 und CBM 8001.

```
DOPEN #1,"NAME"
```

öffnet eine Datei mit dem Namen 'NAME' und der logischen Adresse 1.

```
DCLOSE #1
```

schließt diese Datei wieder.

DSAVE ist ein Befehl der Serien CBM 4001 und CBM 8001. Während DLOAD ein Programm von Diskette lädt, speichert DSAVE das im Computerspeicher befindliche Programm auf Diskette ab.

```
DSAVE "NAME",D0
```

schreibt das Programm 'NAME' auf Diskette in Laufwerk 0.

```
DSAVE "NAME",D1
```

schreibt auf Laufwerk 1.

END beendet die Programmausführung und druckt

```
READY.
■
```

Mit CONT ist eine Fortsetzung möglich, falls END nicht der letzte Befehl des Programms ist.

EXP/LOG sind die Exponentialfunktion bzw. Logarithmusfunktion zur Basis e (e = 2,71828...). Das Argument für EXP darf nicht wesentlich größer als 88 sein, das für LOG muß größer als 0 sein.

FN ruft eine vorher mit DEFFN definierte Funktion auf.
S. DEFFN/FN

FOR-NEXT/STEP ermöglicht eine einfache Programmierung von Schleifen. Der Schleifenbeginn wird mit FOR angegeben, gefolgt vom Anfangswert der Schleife (A), Endwert (E) und Schrittweite (S):

```
FOR I=A TO E STEP S
```

Dabei wird I die Schleifenvariable genannt. Werden STEP und die Schrittweite weggelassen, so wird als Ersatzwert 1 genommen.
Das Ende einer FOR-NEXT-Schleife ist mit NEXT gekennzeichnet.

```
NEXT I
```

schließt die vorher mit FOR geöffnete Schleife. Bei NEXT muß nicht unbedingt die Schleifenvariable mit angegeben sein, es empfiehlt sich aber, die Variable mit anzugeben.

FRE gibt den noch freien Speicherplatz aus.

```
? FRE(1)
```

Das Argument hat keinerlei Bedeutung, es muß aber vorhanden sein.

GET ist für die Eingabe e i n e s Zeichens von der Tastatur gedacht. Der Computer wartet im Gegensatz zu INPUT nicht auf den Tastendruck.

```
GET X$
GET X
```

Wird keine Taste während der Ausführung von GET betätigt, dann ist X$ gleich leer (" ") bzw. X gleich 0.
Soll auf einen Tastendruck gewartet werden, so läßt sich das folgendermaßen programmieren:

```
10 GET X$ : IF X$="" THEN 10
20 PRINT X$
```

GET# liest ein einzelnes Zeichen von einem Peripheriegerät ein. Zuvor muß ein entsprechendes FILE mit OPEN oder DOPEN geöffnet werden.

GOSUB/RETURN erlauben Unterprogrammtechnik. Ein Unterprogramm ("subroutine") wird vom Hauptprogramm mit GOSUB aufgerufen. Stößt das Programm danach auf einen RETURN-Befehl, dann springt es zurück in das Hauptprogramm direkt hinter die GOSUB-Anweisung, die es aufrief. Die Schachtelung von Unterprogrammen ist in Commodore-BASIC erlaubt, d.h. ein Unterprogramm darf ein anderes mit GOSUB aufrufen.
Ist die Unterprogrammschachtelung zu tief, so erscheint die Fehlermeldung ?OUT OF MEMORY ERROR. Dabei ist aber nur die Verschachtelungstiefe entscheidend, nicht die Anzahl der GOSUB-Befehle im Programm.

GOTO stellt eine unbedingte Sprunganweisung zu einer Zeilennummer dar.

```
250 GOTO 520
```

Statt 'GOTO' darf auch 'GO TO' geschrieben werden.

IF..GOTO macht den Sprung zu einer Zeile von einer oder mehreren Bedingungen abhängig. Diese Bedingungen stehen zwischen IF und GOTO, die Sprungzeilennummer folgt auf GOTO.

```
130 IF A>10 GOTO 1000
```

IF..THEN macht alle nach THEN folgenden Programmanweisungen von der Bedingung zwischen IF und THEN abhängig.

```
130 IF X=5  THEN PRINT "FUENF"
```

Auf THEN darf auch direkt eine Zeilennummer folgen. IF..THEN wirkt in diesem Fall genauso wie IF..GOTO.

```
130 IF A>10 THEN 1000
```

Ist die zwischen IF und THEN stehende Bedingung erfüllt, dann werden sämtliche auf THEN folgenden Anweisungen ausgeführt, auch die durch Doppelpunkt getrennten. Bei Nichterfüllung erfolgt ein Sprung auf die nächstfolgende Zeilennummer.

INPUT dient der Eingabe von Zahlen oder Zeichen über die Tastatur. Der Eingabeabschluß erfolgt mit der RETURN-Taste.

```
10 INPUT A,B,C
```

Dieser Befehl erwartet drei durch Komma getrennte Zahlen als Eingabe. Zur Aufforderung wird ein Fragezeichen auf den Bildschirm gedruckt.
Mit INPUT läßt sich auch ein Aufforderungstext verbinden:

```
10 INPUT "DREI ZAHLEN";A,B,C
```

Stringeingaben mittels INPUT müssen ggf. in Anführungszeichen gesetzt werden, und zwar immer dann, wenn auch in einer DATA-Anweisung Anführungszeichen stehen müßten (s. DATA/READ/RESTORE).
Werden mehr Daten (durch Kommata getrennt) eingegeben, als der INPUT-Befehl anfordert, erscheint der Hinweis EXTRA IGNORED, werden dagegen zu wenig Daten eingegeben, besteht die Antwort aus '??' und die restlichen Daten können eingetippt werden. INPUT ist ein BASIC-Wort, das nur programmiert und nicht direkt eingegeben werden kann.

INPUT# ist ein Befehl für Eingabe von Peripheriegeräten. Vorher muß ein entsprechendes File mit OPEN oder DOPEN eröffnet werden.

INT liefert zu einem Argument die nächstkleinere Zahl. Für positive Zahlen wird somit der Nachkommateil ohne Rundung abgeschnitten, für negative Zahlen wird zusätzlich noch eine 1 subtrahiert.

```
INT(5.2) = 5
INT(5.8) = 5
INT(-5.2) = -6
INT(0) = 0
```

HEADER kennen nur die Computer der Serien CBM 4001 und CBM 8001.
HEADER dient dazu, eine Diskette zu löschen oder neu zu formatieren. Nach einem HEADER-Befehl sind alle Daten der Diskette unwiederbringlich gelöscht. Jede Diskette, die zum ersten Mal benutzt werden soll, muß vorher formatiert werden.

```
HEADER "NAME",IXY,D0
```

formatiert eine Diskette in Laufwerk 0 mit dem Namen 'NAME' und dem Identitätszeichen XY. Als Identität kann jede Zeichenkombination aus zwei Zeichen gewählt werden.
Ist keine Identität angegeben, so wird die Diskette nur gelöscht und nicht formatiert. Dies geht nur, wenn sie vorher schon einmal formatiert wurde.

```
HEADER "NAME",D0
```

formatiert nicht, aber löschlle Daten auf der Diskette und ändert den Diskettennamen in 'NAME'.
Nach der Eingabe von HEADER erfolgt die Sicherheitsabfrage

```
ARE YOU SURE ?
```

Erst wenn jetzt 'Y' oder 'YES' eingegeben wird, kommt der HEADER-Befehl zur Ausführung.

LEFT$ erlaubt aus einem String eine bestimmte Anzahl von Zeichen von links gezählt herauszunehmen.

```
LEFT$("ABCDEF",3)
```

liefert den neuen String "ABC" (3 Zeichen von links). Statt eines Strings darf auch eine Stringvariable stehen. Enthält ein String weniger Zeichen als mit LEFTS herausgenommen werden sollen, so wird der gesamte String übernommen.

LEN ergibt die Länge eines Strings.

```
LEN("ABCD") = 4
```

Der String kann auch durch eine Stringvariable ersetzt werden.

```
LEN(A$) = 5
```

mit

```
A$="12345"
```

LET weist einer Variablen einen Wert zu.

```
LET X=33
```

LET kann im Commodore-BASIC weggelassen werden, es genügt also zu schreiben

```
X=33
```

statt

```
LET X=33
```

LIST listet Programmzeilen auf. Mit LIST wird das gesamte Programm auf dem Bildschirm aufgelistet, 'LIST A–B' listet von Zeile A bis Zeile B, 'LIST A–' listet ab A und 'LIST –B' listet bis B. Der Befehl LIST ist auch programmierbar. In diesem Fall listet sich das Programm selbst und hält dann an.

LOAD lädt ein Programm von Kassette.

```
LOAD "NAME"
```

lädt das Programm 'NAME'. Bei den CBM-Computern kann noch ein zweiter Kassettenrekorder angeschlossen werden. Dieser wird mit

```
LOAD "NAME",2
```

angesprochen.
LOAD ist programmierbar, so daß also ein Programm ein anderes nachladen kann. Das erste Programm geht dabei vollständig verloren. Solange das zweite aber nicht größer als das erste ist, bleiben die Variablen erhalten und können somit weitergegeben werden.

LOG ruft die Logarithmusfunktion auf. S. EXP/LOG

MID$ erlaubt, aus einem String einen bestimmten Teil aus der Mitte herauszunehmen.

```
MID$("ABCDEF",2,3) = "BCD"
```

Es werden ab dem 2. Zeichen insgesamt 3 Zeichen herausgegriffen.

```
MID$("ABCDEFGH",8,1) = "H"
```

Ein Zeichen ab dem 8. Zeichen oder einfach das 8. Zeichen wird mit MID$ bezeichnet.
MID$ kann auch von nur einem Zahlenargument begleitet sein. In diesem Fall entfällt die Angabe, wieviele Zeichen genommen werden sollen.

```
MID$("ABCDEF",3) = "CDEF"
```

Es wird also der gesamte String ab dem 3. Zeichen herausgegriffen.
Werden im MID$-Befehl mehr Zeichen spezifiziert als vorhanden sind, wird der gesamte String ab der Anfangsposition übernommen.

```
MID$("ABCDEFG",3,10) = "CDEFG"
```

Anstelle eines Strings kann auch die Stringvariable stehen.

NEW löscht das im Computerspeicher stehenden Programm und alle Variablen.

NEXT kennzeichnet das Ende einer FOR..NEXT-Schleife. S. FOR-NEXT/STEP

NOT 1. dient der Verneinung von Bedingungen in IF..THEN-Anweisungen.

```
50 IF NOT(X=0) THEN PRINT "X <> 0"
```

2. erlaubt die Boole'sche Operation NOT durchzuführen.

```
NOT(20) = -21
NOT(-1) = 0
```

NOT wandelt sein Argument in eine 16 Bit Binärzahl um und negiert dann bitweise. Dezimal läßt sich die Operation folgendermaßen ausdrücken: NOT(X) = – (X+ 1).

ON..GOSUB stellt einen bedingten Unterprogrammsprungbefehl dar.

```
780 ON X GOSUB 100,200,300,400,500,600
```

Wenn X den Wert 1 hat, wird Unterprogramm 100 angesprungen, für X gleich 2 Unterprogramm 200 usw.
Falls X größer als 6 ist, wird gar kein GOSUB-Sprung ausgeführt, sondern der nächstfolgende Befehl abgearbeitet.
Anstelle der Variablen X kann ein beliebiger numerischer Ausdruck stehen, also z. B.

```
780 ON 2+X-100 GOSUB 100,200,300,400
```

ON. .GOTO funktioniert im Prinzip genau wie ON. .GOSUB, nur werden keine Unterprogramme angesprungen, so daß zur Verzweigungstelle nicht mit RETURN zurückgekehrt werden kann. Sonst gilt alles zu ON. .GOSUB Gesagte.

OPEN/CLOSE sind Befehle zur Behandlung von Files auf Peripheriegeräten. Mit OPEN wird ein File eröffnet

```
OPEN 1,1,0
```

Die erste 1 ist die logische Filenummer. Diese kann zwischen 1 und 255 liegen und willkürlich gewählt werden. Alle Ein-und Ausgabebefehle sowie CLOSE beziehen sich auf die logische Filenummer, auch logische Adresse genannt.
Die zweite 1 bezeichnet ein bestimmtes Gerät, im Beispiel den Kassettenrekorder 1 (2 = Rekorder 2, 8 = Floppystation). Die zweite 1 wird Gerätenummer oder Primäradresse genannt. Die 0 im OPEN-Befehl eröffnet ein Eingabefile. Andere Zahlen an dieser Stelle haben je nach Gerät unterschiedliche Bedeutung. Diese dritte Nummer im OPEN-Kommando heißt Sekundäradresse.
Eröffnete Files müssen mit CLOSE wieder geschlossen werden.

```
CLOSE 1
```

Damit wird das File mit der logischen Adresse 1 geschlossen.

OR 1. dient der Verbindung mehrerer Bedingungen in IF. .THEN-Anweisungen mit "ODER".

```
310 IF X=1 OR X=2 THEN 500
```

2. erlaubt die Boole'sche Operation OR durchzuführen.

```
? 12 OR 32
```

OR wandelt seine Argumente in 16 Bit Binärzahlen um und führt die Boole'sche Operation bitweise durch. Der zulässige Zahlenbereich geht von –32767 bis +32767. Die Operation OR ist folgendermaßen definiert:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

3. ist geeignet für Bitmanipulationen einzelner Speicherzellen des Computers.

PEEK/POKE sind Befehle zur Abfrage und Manipulation einzelner Speicherzellen des Computers. Mit PEEK(S) kann man den Inhalt der Speicherzelle S abfragen, mit POKE S,W den Wert W in die Speicherzelle S auf den Bildschirm bringen. W darf Werte zwischen 0 und 255 annehmen. Bei wahllosen Speicherzellen S kann es zur Funktionsuntüchtigkeit des Computers kommen.
Sowohl bei den CBM-Computern als auch beim VC 20-Computer wird die Tonerzeugung mit POKE programmiert. Beim VC 20 läßt sich mit POKE außerdem noch die Farbe programmieren.

POKE erlaubt die Veränderung des Inhalts einer Speicherzelle. S. PEEK/POKE

POS ist ein recht selten benutzter Befehl. POS gibt die Stelle an, wo sich der Cursor innerhalb einer Zeile befindet.

```
120 PRINT TAB(15); : A=POS(1) : PRINT A
```

liefert das Ergebnis 15, d.h. der Cursor stand während der Ausführung der POS-Anweisung an der 15. Stelle einer Bildschirmzeile.
Das Argument zu POS ist unerheblich, es muß aber vorhanden sein.

PRINT ist ein Druckbefehl für den Bildschirm. Es lassen sich Zahlen, Strings und Variablen ausdrucken.

```
110 PRINT "ERGEBNIS =";X;A$;12
```

Folgt dem zu druckenden Ausdruck ein Semikolon, so bleibt der Cursor in derselben Zeile direkt nach der Ausgabe stehen. Folgt ein Komma, so wird der Cursor auf die nächstfolgende Zehnerposition gesetzt.
Fehlen Komma oder Semikolon, dann fällt der Cursor nach dem PRINT automatisch an den Anfang der nächsten Bildschirmzeile.

PRINT# ist ein Ausgabebefehl auf ein Peripheriegerät. Das kann ein Drucker, eine Floppystation, ein Kassettenrekorder oder ein anderes Gerät sein.

```
OPEN 1,4
PRINT #1,"HALLO"
```

druckt "HALLO" auf einen evtl. angeschlossenen Drucker aus. Das File muß mit CLOSE wieder geschlossen werden.

```
CLOSE 1
```

READ liest ein unter DATA abgespeichertes Datenelement in eine Variable ein. S. DATA/READ/RESTORE.

RECORD steht ausschließlich auf den Computern CBM 4001 und CBM 8001 zur Verfügung.
RECORD dient zum Stellen des Schreib-/Lesezeigers bei REL-Dateien auf Diskette.

REM erlaubt, Kommentare in ein Programm einzustreuen. Alle auf REM folgenden Texte, Zeichen, Anweisungen werden vom Computer während der Programmausführung ignoriert und haben nur den Sinn, von einem Menschen gelesen zu werden.
REM verbraucht Speicherplatz und verlangsamt die Programmausführung ein klein wenig.

RENAME ist ein Kommando der Serien CBM 4001 und CBM 8001. RENAME dient zum Umbenennen einer Datei auf Diskette.

```
RENAME "NAME 1",D1 TO "NAME 2"
```

RESTORE stellt den DATA-Zeiger auf das erste DATA-Element im Programm zurück. S. DATA/READ/RESTORE

RETURN bewirkt einen Sprung aus einem Unterprogramm in das Hauptprogramm bzw. in die nächsthöhere Unterprogrammebene. S. GOSUB/RETURN

RIGHT$ erlaubt, aus einem String eine bestimmte Anzahl von Zeichen von rechts gezählt herauszunehmen.

```
RIGHT$("ABCDEF",3) = "DEF"
```

Der String kann auch durch eine Stringvariable ersetzt werden. Enthält der String bzw. die Variable weniger Zeichen, als mit RIGHT$ herausgenommen werden sollen, so wird der gesamte String übernommen.

```
RIGHT$("XYZ",5) = "XYZ"
```

RND ist eine Funktion zum Erzeugen von Zufallszahlen. Sie liefert Werte zwischen 0 (einschließlich) und 1 (ausschließlich). Für negative Argumente einschließlich null entspricht jedem Argument eine bestimmte "Zufallszahl". Für positive Argumente ist die Zahl wirklich zufälliger Natur.

```
RND(1) = ?
```

Um Zufallszahlen größer als eins zu erhalten, bedient man sich folgender Funktion (Zahlen zwischen B und A+B–1):

```
X=INT(A*RND(1)+B)
```

RUN startet das im Speicher stehende Programm. Gleichzeitg werden alle Variablen gelöscht.

SAVE speichert ein Programm auf Kassette. Ein Name, der aus höchstens 16 Zeichen bestehen darf, wird dem Programm so mitgegeben:

```
SAVE "NAME"
```

Um Rekorder 2 bei den CBM-Computern anzusprechen:

```
SAVE "NAME",2
```

SCRATCH ist ein Befehl der Serien CBM 4001 und CBM 8001. SCRATCH löscht eine auf Diskette stehende Datei oder ein Programm

```
SCRATCH "NAME",D1
```

löscht das Programm oder die Datei mit dem Namen 'NAME' auf Laufwerk 1. Nach der Eingabe von SCRATCH erscheint die Sicherheitsabfrage

```
ARE YOU SURE ?
```

Erst wenn jetzt 'Y' oder 'YES' eingegeben wird, kommt SCRATCH zur Ausführung.

SGN ist die Signumfunktion. SGN(X) liefert für

$X > 0$ den Wert +1
$X = 0$ den Wert 0
$X < 0$ den Wert –1

SIN bezeichnet die Sinus-Funktion. Der Winkel ist in Bogenmaß anzugeben. Umrechnung von Altgrad s. COS.
Die Arkussinus-Funktion ist nicht implementiert. Sie kann auf folgende Weise ausgedrückt werden:

$$\text{ARCSIN}(X) = \text{ATN}(\text{SQR}(-1X*X)) \text{ mit } 0 \leqslant X \leqslant 1$$

SPC setzt den Cursor um die im Argument angegebenen Stellen weiter nach rechts. SPC bezieht sich im Gegensatz zu TAB nicht auf den linken Bildschirmrand, sondern auf die momentane Cursorposition innerhalb der Zeile.

```
50 PRINT TAB(10);"A";SPC(3);"B"
```

SQR bezeichnet die Quadratwurzel-Funktion. Das Argument darf nicht negativ sein.

```
SQR(4) = 2
```

STEP dient der Spezifizierung der Schrittweite bei FOR-NEXT-Schleifen. S. FOR-NEXT/STEP

STOP beendet die Programmausführung mit der Meldung

```
BREAK IN ZEILENNUMMER
READY.
■
```

Mit CONT ist eine Fortsetzung möglich, falls STOP nicht der letzte Befehl des Programms ist.

STR$ wandelt einen Zahlenwert in einen String um.

```
STR$(13.8) = " 13.8"
```

Das Leerzeichen vor der 1 steht für ein unterdrücktes Pluszeichen. Minus wird dagegen übernommen.

```
STR$(-13.8) = "-13.8"
```

STR$ ist die Umkehrfunktion zu VAL.

SYS erlaubt, von einem BASIC-Programm aus Maschinencode-Programme aufzurufen. Das Argument von SYS ist die Startadresse des Maschinenprogramms.

TAB setzt den Tabulator auf die im Argument angegebene Stelle innerhalb der Bildschirmzeile.

```
860 PRINT TAB(16);"HALLO"
```

druckt "HALLO" an die 16. Stelle.
Im Gegensatz zu SPC bezieht sich TAB nicht auf die momentane Cursorposition, sondern immer auf den linken Bildschirmrand.

TAN/ATN bezeichnet die Tangens- bzw. Arkustangens-Funktion. Alle Berechnungen geschehen im Bogenmaß. Umrechnungen s. COS.
Mit Hilfe von ATN können auch die Umkehrfunktionen zu Sinus und Cosinus ausgedrückt werden (s. SIN und COS).

THEN ist der Anfang des zweiten Teils einer IF. .THEN-Anweisung. S. IF. .THEN

USR dient ähnlich SYS zum Aufruf von Maschinencode-Programmen. Die Sprungadresse zu USR wird in den Adressen 1 (LSB) und 2 (MSB) festgelegt.

VAL wandelt einen String in einen Zahlenwert um.

```
VAL("12.5") = 12.5
```

Enthält der String Buchstaben oder Sonderzeichen gleich am Anfang, so liefert VAL das Ergebnis 0.

```
VAL("ABC") = 0
```

Beginnt der String mit umwandlungsfähigen Ziffern und folgen Buchstaben oder Sonderzeichen, so funktioniert VAL so weit wie möglich:

```
VAL("52X83") = 52
```

VAL ist die Umkehrfunktion zu STR$.

VERIFY vergleicht ein im Arbeitsspeicher des Computers befindliches Programm mit einem anderen auf Kassette gespeicherten. Bei Gleichheit erfolgt die Meldung OK, sonst ein ?VERIFY ERROR.

```
VERIFY "NAME"
```

VERIFY funktioniert auch bei auf Disketten gespeicherten Programmen, wenn es den Zusatz ",8" bekommt:

```
VERIFY "NAME",8
```

VERIFY ist programmierbar.

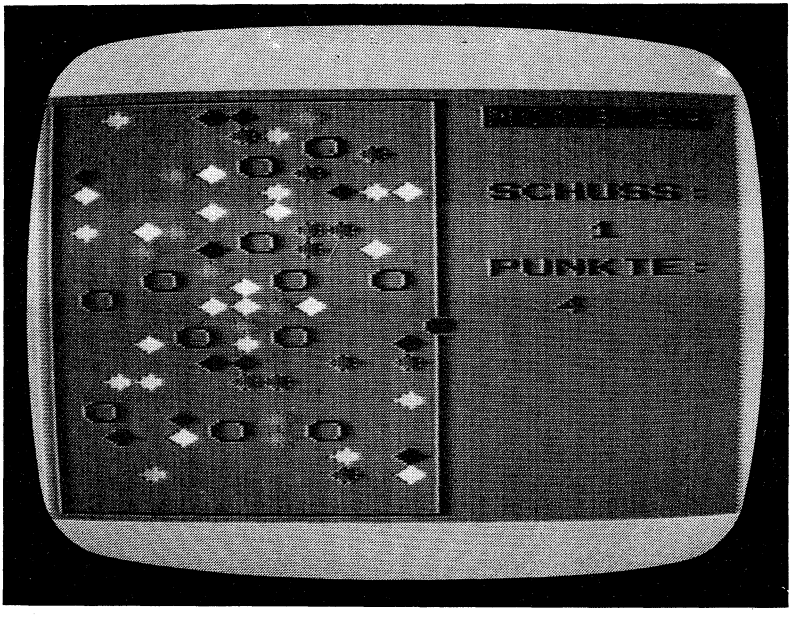
WAIT I,J,K liest den Zustand der Adresse I, exklusiv ORd (XOR) ihn mit K und AND'T das Ergebnis so lange mit J, bis das Resultat von null verschieden ist. Wenn WAIT nur zwei Parameter hat (WAIT I,J), dann wird K gleich null angenommen.
WAIT ist im Zusammenhang mit besonderen Peripheriegeräten von Interesse.

BONBON: MONSTER

Dieses Spiel gehört eigentlich gar nicht mehr in dieses Buch hinein. Schließlich besitzen Sie ja schon 18 neue Spiele. Verstehen Sie daher MONSTER bitte als Zusatz, eben als Bonbon. Um was es geht? Nun tippen Sie MONSTER ein und probieren Sie selbst.
Wenn Sie mit einer Floppystation statt mit einem Rekorder arbeiten, dann ist folgende Änderung notwendig, damit das Anleitungsprogramm das Hauptprogramm automatisch nachlädt:

```
2100 PRINTSPC(7)"PROGRAMM■":PRINTSPC(5)"■■M O N S T E R■"
2110 LOAD"MONSTER",8
```

Nun wünsche ich Ihnen viel Spaß beim Entdecken von MONSTER. Passen Sie aber auf, daß das Monster Sie nicht zwischen die Zähne bekommt!



```
1 REM  ------------
2 REM  --MONSTER--
3 REM  -ANLEITUNG-
4 REM  ------------
5 REM
6 REM  (C) 1982 BY
7 REM ANDREAS DRIPKE
8 REM
9 :
1000 POKE657,128:POKE36879,143:PRINT"■■■■"CHR$(14):IFPEEK(37151)AND64GOTO1020
1010 PRINT"BITTE LASSEN SIE DIE":PRINT"■PLAY-TASTE GEDRUECKT.":GOTO2000
1020 PRINT" BITTE DRUECKEN SIE":PRINT"■■■ DIE PLAY-TASTE.":WAIT37151,64,64
2000 POKE192,1:POKE37148,254:PRINTTAB(10)"■OK":FORI=1TO3E3:NEXT:GOSUB4000
2010 FORI=0TO255:POKE36879,IOR8:NEXT:POKE36879,143:FORI=1TO3E3:NEXT
2020 PRINTCHR$(14)"■■■■■    M O N S T E R■":PRINT"■ IHRE AUFGABE BESTEHT■
2030 PRINT" DARIN, MIR, DEM MON-■■■STER, SO VIELE KEKSE■■■ WIE MOEGLICH ZU BE-■
2040 PRINT"SCHAFFEN.":GOSUB3000:PRINT"■■■■■■■■ DAS SPIEL GEHT UEBER■
2050 PRINT" FUENF RUNDEN,  WOBEI","■ SIE JE EINEN  SCHUSS","■ ABFEUERN KOENNEN.
2060 GOSUB3000:PRINT"■■ DIE TASTENFUNKTIONEN■":PRINT"■ ZIELEN:
2070 PRINT"■ ',' ',' '/'":PRINT"■ LINKS  MITTE  RECHTS■
2080 PRINT"■ BEWEGEN:■":PRINT"■ 'HOME' SPACE  'DEL'
2090 PRINT"■ LINKS  FEUERN RECHTS":GOSUB3000:PRINT"■■■■■  ICH LADE JETZT DAS■
2100 PRINTSPC(7)"PROGRAMM■":PRINTSPC(5)"■■M O N S T E R■":LOAD
3000 PRINTSPC(60)"TASTE":POKE198,0:WAIT198,1:GETG$:RETURN
4000 PRINTCHR$(142)"■■"SPC(117)"■■       ■■":PRINTSPC(6)"■■ ●    ● ■■
4010 PRINTSPC(5)"■■    ..    ■■":PRINTSPC(4)"■■ ■■■■■■■■
```

```
4040 PRINTSPC(4)"■  ■ ◆ ◆◆   ◆ ◆■  ■":PRINTSPC(4)"■  ■◆ ◆◆ ◆ ◆◆ ■  ■
4050 PRINTSPC(4)"■ ▼▼▼▼▼▼ ■▼":PRINTSPC(5)"■           ■▼
4060 PRINTSPC(6)"■          ■▼":FORI=1TO2E3:NEXT:P$="■■■
```

```
1030 PRINTCHR$(142)"■┌──────────┐ ________":FORI=0TO20:PRINT"|"SPC(11)"|":NEXT
1040 PRINT"└──────────┘■"SPC(36)"■MONSTER■"SPC(81)"SCHUSS:"SPC(81)"PUNKTE:
1050 POKE650,128:POKE657,128:Z=0:FORI=0TO11
1060 P=FNR(17)+VB:Q=P-VB+VC
1070 IFPEEK(P)<>32ORPEEK(P+1)<>32ORPEEK(P+22)<>32ORPEEK(P+23)<>32GOTO1060
1080 POKEP,85:POKEP+1,73:POKEP+22,74:POKEP+23,75:POKEQ,0:POKEQ+1,0:POKEQ+22,0
1090 POKEQ+23,0:NEXT:FORI=0TO59
1100 P=FNR(20)+VB:ON-(PEEK(P)<>32)GOTO1100:POKEP,90:POKEP-VB+VC,8*RND(1):NEXT
1110 FORA=1TO5:N=0:PRINT"■"SPC(170)A:S=93:R=8:P=VB+468:POKEP,S:POKEP-VB+VC,0
1120 Q=0:GETA$:IFA$=""GOTO1120
1130 IFA$=" "GOTO2000
1140 IFA$=","THENR=7:S=77
1150 IFA$="."THENR=8:S=93
1160 IFA$="/"THENR=9:S=78
1170 IFA$="■"THENQ=-1
1180 IFA$=CHR$(20)THENQ=1
1190 IFP+Q=VB+462ORP+Q=VB+474GOTO1120
1200 POKEP,32:P=P+Q:POKEP,S:POKEP-VB+VC,0:GOTO1120
2000 O=32
2010 N=N+1:POKEP,O:P=P+R%(R):O=PEEK(P):POKEP,81:POKEP-VB+VC,0
2020 FORC=0TO89:NEXT:IFN=50THENPOKEP,O:NEXT:GOTO3000
2030 IFO=32THEN2010
2040 IFO=90THENZ=Z+1:PRINT"■■"SPC(235)Z:O=32:GOTO2010
2050 POKE36878,13:POKE36876,128+64*RND(1):FORC=0TO9:NEXT
2060 POKE36876,0:POKE36878,0
2070 IFO=109ORO=110ORO=112ORO=125THENR=10-R:GOTO2010
2080 IFO<>64GOTO2110
2090 IFR=10RR=9THENR=8+6*(R=9):GOTO2010
```

```
2100 R=R+5+10*(R>5):GOTO2010
2110 IFO<>93GOTO2140
2120 IFR=3ORR=7THENR=6+2*(R=3):GOTO2010
2130 GOTO2100
2140 IFO<>85GOTO2180
2150 IFR=1THENR=4:GOTO2010
2160 IFR=2THENR=7:GOTO2010
2170 R=R+1+2*(R>7):GOTO2010
2180 IFO<>73GOTO2230
2190 IFR=7THENR=8:GOTO2010
2200 IFR=2THENR=9:GOTO2010
2210 IFR=3THENR=6:GOTO2010
2220 R=R+5:GOTO2010
2230 IFO<>74GOTO2280
2240 IFR=3THENR=2:GOTO2010
2250 IFR=8THENR=1:GOTO2010
2260 IFR=7THENR=4:GOTO2010
2270 R=R-5:GOTO2010
2280 IFO<>75THENSTOP
2290 IFR=9THENR=6:GOTO2010
2300 IFR=8THENR=3:GOTO2010
2310 IFR=1THENR=2:GOTO2010
2320 R=R-1:GOTO2010
3000 POKE36879,138:PRINTCHR$(14)"■GESAMTPUNKTZAHL"Z"■■":IFZ>40GOTO3040
3010 PRINT"AUF UNFAEHIGE LEUTE","■WIE SIE HABE ICH HEUTE■BESONDEREN APPETIT!
3020 PRINT"■HOFFENTLICH SIND SIE■■NICHT ZU ZAEH!!!":FORA=1TO8E3:NEXT
3030 GOSUB4000:POKE650,0:POKE657,0:END
3040 IFZ>45GOTO3080
```

```
3050 PRINT"ES IST IHNEN DOCH","▯HOFFENTLICH KLAR, DASS▯DAS NICHT MAL MEINEN
3060 PRINT"▯HOHLEN ZAHN FUELLT.","▯IHR GLUECK, DASS ICH ▯▯HEUTE MEINEN VEGETA-
3070 PRINT"▯RISCHEN TAG HABE!!!":GOTO3120
3080 IFZ>50GOTO3100
3090 PRINT"JETZT BIN ICH ZWAR","▯NOCH NICHT SATT, ABER ▯IMMERHIN ...":GOTO3120
3100 PRINT"DAS WAR WIRKLICH EIN ▯▯OPULENTES MAHL, DAS","▯SIE MIR DA BESORGT
3110 PRINT"▯HABEN, ABER BILDEN SIE▯SICH DARAUF BLOSS","▯NICHTS EIN!!!
3120 PRINT"▯▯NOCHMAL?"
3130 GETA$:ON(A$="J")-2*(A$<>"N")GOTO1030,3130:POKE650,0:POKE657,0:END
4000 PRINTCHR$(142)"▯"SPC(117)"▗▄▄▄
```

Der neue Basicstandard
für Commodore Computer ×

× CBM • VC »VolksComputer«

Hilfsfunktionen:
FIND Suchen nach Text, Variablen, mit Bereichsangabe. • **AUTO** automatische Zeilennummerngenerierung. • **DEL** Zeilennummernbereich löschen. • **RENUM** Programmzeilen Umnumerierung. • **TRACE (OFF)** Einzelschritt- oder langsamer Programmablauf mit Einblendung der jeweils abgearbeiteten Zeile (vollständige Zeile). • **ON/OFF** Ein-/Ausschalten der Repeatfunktion für die Tasten. Außerdem Fehlermodus: Tritt ein Programmfehler auf, so wird die Zeile automatisch (!) gelistet und der Cursor an der fehlerhaften Stelle positioniert. • **DUMP** listet alle normalen Variablen mit Inhalten. • **MATRIX** listet alle Feldvariablen (Arrays) mit Inhalten. • **LETTER (OFF)** Umschaltung Grafik-/Großkleinschreibmodus. • **FAST (OFF)** schnelle Bildschirmausgabé bei PRINT und LIST. • "." Symbol für „zuletzt bearbeitete Zeilennummer" (z.B.: 100 GETA$:IFA$ = "THEN." • **MEM** gibt Speicherplatzbelegung aus: MAIN MEMORY, PROGRAM, VARIABLES, ARRAYS, STRINGS, REK, FREE. • **HIMEM** Speicherplatzabgrenzung für Maschinenprogramme. • **GO** Einsprung in Monitor TIM bzw. EXMON. • **SPACE (OFF)** formatiertes Listen. • **STOP ON/OFF** Die Stoptaste bleibt auch beim Ablauf von Maschinenprogrammen aktiv. • **HELP** listet alle EXBASIC LEVEL II Befehle. • **HELP•** listet die Befehle des normalen Basic. • **BASIC** Ausschalten von EXBASIC LEVEL II. • **MERGE** Zusammenfügen von Basicprogrammen (auch ineinander) von Kassette. • **MERGE•** wie MERGE, aber von Floppy.

Grafikbefehle:
PRINT AT Druck an spezifizierte Bildschirmstelle. • **HPLOT** horizontales Plotten. • **VPLOT** vertikales Plotten. • **SET** Grafikpunkt setzen. • **RESET** Grafikpunkt löschen. • **POINT** Abfrage, ob Grafikpunkt gesetzt oder nicht.

Mathematische Funktionen:
MAX sucht das Maximum aus einer Variablenliste. • **MIN** gibt das Minimum einer Variablenliste aus. • **FRAC(X)** Nachkommateil einer Zahl. • **ROUND(X)** rundet die Zahl X auf eine ganze Zahl. • **ROUND(X,Y)** rundet die Zahl X auf Y Nachkommastellen. • **ODD(X)** Abfrage, ob X gerade oder ungerade ist. • **RND(X)** erzeugt Zufallszahlen ohne Nachkommateil zwischen 1 und X. • **HEX$(X)** wandelt die Zahl X in den entsprechenden hexadezimalen String um. • **DEC(string)** wandelt umgekehrt Hexstring in die Dezimalzahl.

LEVEL II Basic:
ELSE ermöglicht strukturierte IF..THEN..ELSE-Anweisungen, THEN kann bei Eindeutigkeit entfallen. • **RESTORE** mit Zeilennummernangabe für gezieltes DATA-Zugriff. • **ON..RESTORE** gesteuertes RESTORE, ähnlich ON..GOTO. • **PRINT USING** Ausdruckformatierung für Bildschirm oder Drucker (z.B. für Komma unter Komma Druck von Zahlen). • **REK** Erweiterung der Unterprogrammebenen für rekursives Programmieren. • **DISPOSE NEXT** Stackrückstellung für FOR-NEXT-Schleifen („Notausgang" aus einer Schleife ohne NEXT). • **DISPOSE RETURN** Wechsel auf eine höhere Unterprogrammebene ohne RETURN-Sprung • **DISPOSE CLR** Ausspruna aus allen Schleifen und Unterprogrammen, unabhängig von Schachtelung und Tiefe. • **INPUTLINE** ermöglicht die Eingabe aller Zeichen, also auch Komma, Anführungszeichen, Steuerungszeichen, Doppelpunkt ohne Einschränkungen, bei leerer Eingabe wird das Programm nicht abgebrochen, sondern die Eingabevariable behält ihren vorherigen Inhalt. • **INPUTFORM** Eingabe von Standardzeichen (keine Cursorsteuerung oder RVS, aber Komma, Anführungszeichen etc.), sonst wie INPUTLINE, aber es kann eine bestimmte Eingabelänge spezifiziert werden und anderes. • **DEF USR=** definiert USR-Vektor. • **DEF CALL=** definiert CALL-Vektor. • **CALL** Aufruf von Maschinenprogrammen mit Parameterübergabe. • **DOKE** Doppelbyte-POKE bis 65535. • **DEEK** Doppelbyte-PEEK. • **VARPTR** gibt die RAM-Adresse an, wo eine bestimmte Variable abgespeichert ist. • **SPACE** Druck auf definiertes Bildschirmfeld. • **STRING$** Beispiel: A$=STRING$ (5,"-") ist gleichbedeutend mit A$="-----". • **INSTR** untersucht ob ein String in einem anderen enthalten ist. • **EVAL** ähnlich VAL, aber Bsp.: VAL ("12•4") ist 12, während EVAL ("12•4") den Wert 48 ergibt. EVAL verarbeitet alle Funktionen. • **EXEC** führt Basic-Befehle im String aus. • **SWAP** vertauscht zwei Variablen ohne Zwihilfenahme einer dritten. • **SEC** Programmpause für X Sekunden. • **BEEP** Tonerzeugung an CB2. Tohnhöhe und -dauer können mit angegeben werden. • **ON ERROR GOTO** ermöglicht die selbständige Fortsetzung eines Programmes nach Auftreten eines Programmfehlers. Zur Fehlerbehandlung stehen die Variablen EC (error code) und EL (error line) zur Verfügung. • **RESUME** Abschluß einer Fehlerbehandlungsroutine, die mit ON ERROR GOTO angesprungen wird, mit verschiedenen Möglichkeiten: RESUME RESUME NEXT, RESUME Zeilennummer. • **""** Ersatzsymbol für den Befehl REM. • **HARDCOPY** kopiert den Bildschirm auf einen Drucker.

FAST TAPE (CBM 2/3/4, VC 20):
Für die genannten Serien stehen neben den normalen Kassettenrekorderbefehlen neue zur Verfügung, die mit 5facher Geschwindigkeit arbeiten. Programmnamen dürfen bis zu 30 Zeichen (normal 16) lang sein. **MOD** erzeugt ein Programm, das von jedem anderen Computer auch ohne EXBASIC LEVEL II fünfmal so schnell eingeladen werden kann und egal wie es eingelesen wird, automatisch startet.

DOS SUPPORT (CBM 2/3/4, VC 20):
Das DOS SUPPORT bietet komfortable Unterstützung beim Arbeiten mit einer Floppystation durch die Verwendung von Kurzbefehlen. Mit EXEC sind alle DOS SUPPORT Funktionen auch programmierbar.

SCREEN SUPPORT (CBM 8):
Bildschirmsonderbefehle: **SCREEN** definiert Bildschirmfenster. • **DELLINE** löscht eine oder mehrere Bildschirmzeilen. • **INSTLINE** schafft Platz für eine oder mehrere Bildschirmzeilen. • **ENDLINE** löscht ab Cursorposition bis Zeilenende • **BEGINLINE** löscht vom Zeilenanfang bis zur Cursorposition. • **SCREEN UP/DOWN** Bildschirmrollen nach oben und unten • **SCREEN** • macht den 80-Zeichen-Bildschirm des CBM 8000 kompatibel zum 40-Zeichen-Schirm der anderen Serien.

EXMON (CBM 8):
Für die Serie 8000 enthält EXBASIC LEVEL II einen Maschinensprachemonitor, der mit GO aufgerufen wird und folgende Befehle bietet: **A** — Assemblieren • **C** — Berechnung eines relativen Offsets • **D** — Disassemblieren auf dem Bildschirm • **G** — Ausführung eines Assemblerprogrammes • **L** — Laden eines Programmes (Kassette oder Diskette) • **M** — Hexdump • **P** — Disassemblieren auf einem Drucker • **S** — Abspeichern eines Assemblerprogrammes

COLOR SUPPORT (VC 20)
mit speziellen Kommandos zur Farbgestaltung.

SOUND SUPPORT (VC 20):
Sonderbefehle für die Ton- und Soundprogrammierung.

KEY DEFINER (VC 20)
erlaubt Tastendefinitionen.

Der Preis? Er ist – gemessen an den vielfältigen Möglichkeiten, die **EXBASIC LEVEL II** bietet – so klein, daß wir ihn hier fast vergessen hätten: DM 392,– inkl. MwSt. (unverbindliche Verkaufsempfehlung).



Unternehmensberatung Andreas Dripke präsentiert

EXBASIC LEVEL II™

SOFTMODULE – EXBASIC LEVEL II ist erweiterbar mit SOFTMODULEN. Ein SOFTMODUL ist ein Assemblerprogramm, das in den Computer eingeladen wird und den Befehlsvorrat von **EXBASIC LEVEL II** erweitert. Das bedeutet für Sie, wenn Sie sich einmal **EXBASIC LEVEL II** zugelegt haben, so haben Sie Zugriff zu einer Auswahl an SOFTMODULEN; die Sie nach Belieben hinzufügen können. Mit anderen Worten: Sie allein bestimmen, wie weit Sie **EXBASIC LEVEL II** ausbauen wollen, ganz nach Ihren Wünschen. Zwei Standard SOFTMODULE werden zu **EXBASIC LEVEL II** mitgeliefert; sie enthalten die Befehle SORT/CLEAR u. GOTO X/GOSUB X. In Kürze werden weitere SOFTMODULE lieferbar sein: mehrfachgenaue Arithmetik, erweiterte Graphik, Codieren und Decodieren von Programmen, Matrizenrechnungen, Search & Replace und vieles andere mehr. Schaffen Sie sich schon jetzt die Basis für die Zukunft – mit **EXBASIC LEVEL II.**

EXBASIC LEVEL II ist der neue Basic-Standard für Commodore Computer. „Extended Level II -Basic" stellt für Ihren Computer ein stark erweitertes Level II-Basic – wie es der TRS-80 kennt – zur Verfügung. Mit **EXBASIC LEVEL II** haben Sie über 75 neue, äußerst leistungsfähige Funktionen für Commodore Computer, die das Programmieren wesentlich erleichtern und um ein Vielfaches komfortabler gestalten. Die Implementierung von **EXBASIC LEVEL II** erfolgt in freie Sockel des Computers und ist völlig unproblematisch. Mit **EXBASIC LEVEL II** erreichen Commodore Computer eine Leistungsfähigkeit, die bisher als unerreichbar galt. Zudem werden Programme für den TRS-80 mit **EXBASIC LEVEL II** endlich auch auf CBM/VC 20 lauffähig. Damit steht Ihnen ein riesiges Softwareangebot für Ihren Computer zur Verfügung. **EXBASIC LEVEL II** ist bereits weltweit über 10 000mal verkauft, wollen Sie es sich leisten, auf die neue Standardisierung zu verzichten!

EXBASIC LEVEL II bekommen Sie bei Ihrem örtlichen Computerfachhändler.

Das Anleitungsbuch
Zu EXBASIC LEVEL II bekommen Sie eine ausführliche, 120 Seiten starke deutsche Anleitung mit Einbauanweisung, Befehlserklärungen, LEVEL II Basic Kurs und vielen Beispielen geliefert. Im Unterschied zu manch anderer „Anleitung" handelt es sich hierbei nicht um eine schlecht übersetzte kurze Befehlsbeschreibung, sondern um ein didaktisch aufgebautes und schrittweise an die neuen Möglichkeiten von **EXBASIC LEVEL II** heranführendes Buch, geschrieben vom Autor der bei COMPUTER LIFE erschienenen VC 20/CBM-Buchreihe.

INTERFACE AGE Verlag GmbH,
Vohburgerstr. 1,
D-8000 München 21,
Tel. (0 89) 5 80 67 02

VC-20 Spiele Buch 1

Mit diesem Buch liegt ein weiterer Band der bei INTERFACE AGE (COMPUTER LIFE) erscheinenden Serie für Commodore Computer vor. Auch diesmal zeigen sich wieder alle Vorteile, die durch eine solche Spezialisierung erreicht werden. Zum einen laufen die Programme auf Anhieb. Man erspart sich das oft langwierige Anpassen von in "allgemeinem BASIC" geschriebenen Programmen an den eigenen Computer. Zum anderen können nur durch diese Spezialisierung wirklich alle Möglichkeiten des Computers voll ausgenutzt werden. So zeichnen sich auch alle 18 Spiele durch viel Liebe zum Detail aus. Die graphischen Zeichen kommen ebenso zum Einsatz wie die Möglichkeit der Tonerzeugung. Und wo BASIC gar nicht mehr schnell genug ist, da wird es durch ASSEMBLER ersetzt. Die Auswahl der Spielprogramme reicht vom Glücksspiel wie "Roulette" oder "Blackjack" über Actionspiele wie "Weltraumschlacht" oder "Rollerball" und Telespiele wie "Breakout" oder "Monster" bis hin zum "Ehevermittler". Wer lieber mit kühlem Kopf seinem Computer zeigen will, daß er doch der Schlauere ist, der greift zu "Rectus" oder "Gimmy-Nim". Aber - das ist noch lange nicht alles. In gut verständlicher und doch präziser Sprache erläutert der Autor in jedem der 18 Kapitel anhand des jeweiligen Spiels Tips und Tricks rund um das Programmieren. Da erfährt man, wie Bewegungen auf dem Bildschirm oder Musikstücke programmiert werden, was bei der Dimensionierung von Variablen zu beachten und wie der Tastaturpuffer zu benutzen ist, wie man Programme schneller und platzsparender machen kann und vieles mehr. Diese Fülle von Informationen läßt das Buch zu einem idealen BASIC-Kurs für alle werden, denen die Grundbegriffe nicht fremd sind und die ihren Computer besser kennenlernen wollen.