Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

KI: lernende Rechner

Derby auf dem Bildschirm

Tips: die Kalibrierung

Tandy MC-10

Heft 48

Ein wöchentliches Sammelwerk

computer kurs

Heft 48

Inhalt

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

Computer Kurs ist ein wöchentlich erscheinendes Sammelwerk. Die Gesamtzahl der Hefte ergibt ein vollständiges Computer-Nachschlagewerk. Damit Sie jede Woche Ihr Heft erhalten, bitten Sie Ihren Zeitschriftenhändler, Computer Kurs für Sie zu reservieren.

Zurückliegende Hefte
Ihr Zeitschriftenhändler besorgt Ihnen gerne zurückliegende Hefte. Sie können sie aber auch direkt beim Verlag bestellen.

Deutschland: Das einzelne Heft kostet DM 3,80. Bitte füllen Sie eine Postzahlkarte aus an: Marshall Cavendish Int. Ltd. (MCI), Sammelwerk-Service, Postgiroamt Hamburg 48064-202, Postfach 105703, 2000 Hamburg 1, Kennwort: Computer Kurs

Österreich: Das einzelne Heft kostet öS 30. Bitte füllen Sie eine Zahlkarte aus an: Computer Kurs, Wollzeile 11, 1011 Wien, Postscheckkonto Wien 7857201 oder legen Sie Ihrer Bestellung einen Verrechnungsscheck bei. Kennwort: Computer Kurs.

Schweiz: Das einzelne Heft kostet sfr 3,80. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

Abonnement

Sie können Computer Kurs auch alle 2 Wochen (je 2 Ausgaben) per Post zum gleichen Preis im Abonnement beziehen. Der Abopreis für 12 Ausgaben beträgt DM 45,60 inkl. Mwst., den wir Ihnen nach Eingang der Bestellung berechnen. Bitte senden Sie Ihre Bestellung an: Marshall Cavendish Int. Ltd. (MCI), Sammelwerk Service, Postgiroamt Hamburg 86853-201, Postfach 105703, 2000 Hamburg 1, Kennwort: Abo Computer Kurs. Bitte geben Sie an, ab welcher Nummer das Abo beginnen soll und ob Sie regelmäßig für jeweils 12 Folgen einen Sammelordner wünschen.

WICHTIG: Bei Ihren Bestellungen muß der linke Abschnitt der Zahlkarte Ihre vollständige Adresse enthalten, damit Sie die Hefte schnell und sicher erhalten. Überweisen Sie durch Ihre Bank, so muß die Überweisungskopie Ihre vollständige Anschrift gut leserlich enthalten.

SAMMELORDNER
Sie können die Sammelordner entweder direkt bei Ihrem Zeitschriftenhändler kaufen (falls nicht vorrätig, bestellt er sie gerne für Sie) oder aber Sie bestellen die Sammelordner für den gleichen Preis beim Verlag wie folgt:

Deutschland: Der Sammelordner kostet DM 12. Bitte füllen Sie eine Zahlkarte aus an: Marshall Cavendish International Ltd. (MCI), Sammelwerk-Service, Postgiroamt Hamburg 48064-202, Postfach 105703, 2000 Hamburg 1, Kennwort: Sammelordner Computer Kurs.

Österreich: Der Sammelordner kostet öS 98. Bitte füllen Sie eine Zahlkarte aus an: Computer Kurs Wollzeile 11, 1011 Wien, Postscheckkonto Wien 7857201 oder legen Sie Ihrer Bestellung einen Verrechnungsscheck bei. Kennwort: Sammelordner Computer Kurs

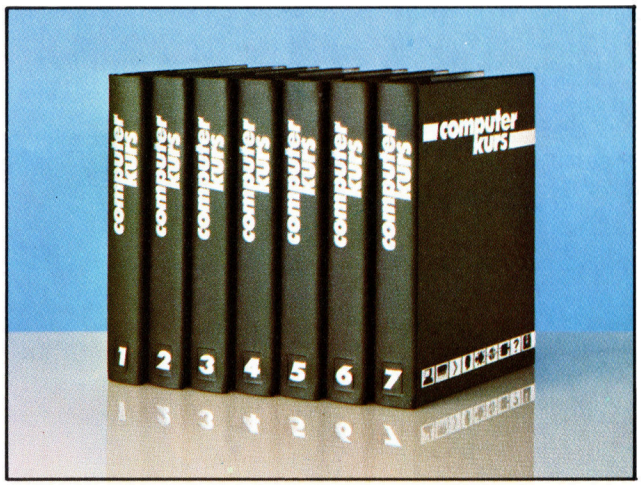
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

Alle 12 Hefte erscheint ein Teilindex. Die letzte Ausgabe von Computer Kurs enthält den Gesamtindex — darin einbezogen sind Kreuzverweise auf die Artikel, die mit dem gesuchten Stichwort in Verbindung stehen.

Redaktion: Winfried Schmidt (verantw. f. d. Inhalt), Elke Leibinger, Susanne Brandt, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

Vertrieb: Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1



Druck: E. Schwend GmbH, Schmollerstraße 31, 7170 Schwäbisch Hall

Perfekt gedruckt

Der LaserWriter von Apple arbeitet mit den modernsten Drucktechniken und erzeugt eine Schriftqualität, die es auf Microcomputern bisher noch nicht gab.

Die meisten Drucker für Microcomputer arbeiten mit einer Punktmatrix oder mit Typenrädern. Wenngleich diese beiden Techniken für die meisten Anwendungen völlig ausreichen, haben sie auch Nachteile. Matrixdrukker sind schnell und verfügen über eine breite Zeichenpalette. Ihr Druckbild genügt jedoch nur selten hohen Ansprüchen, da die acht bis zehn Nadeln der normalen Matrixdrucker eine recht grobe Zeichenstellung produzieren. Typenraddrucker erzeugen ein ausgezeichnetes Schriftbild, doch fehlt hier die Geschwindigkeit und Flexibilität der Punktmatrix. Wenn das Typenrad nicht auswechselbar ist, kann man nur in einer einzigen Schriftart drucken. Die Höchstgeschwindigkeit der Typenradmaschinen liegt bei etwa 80 Zeichen pro Sekunde, während Matrixdrucker in dieser Zeit leicht 200–300 Zeichen ausgeben.

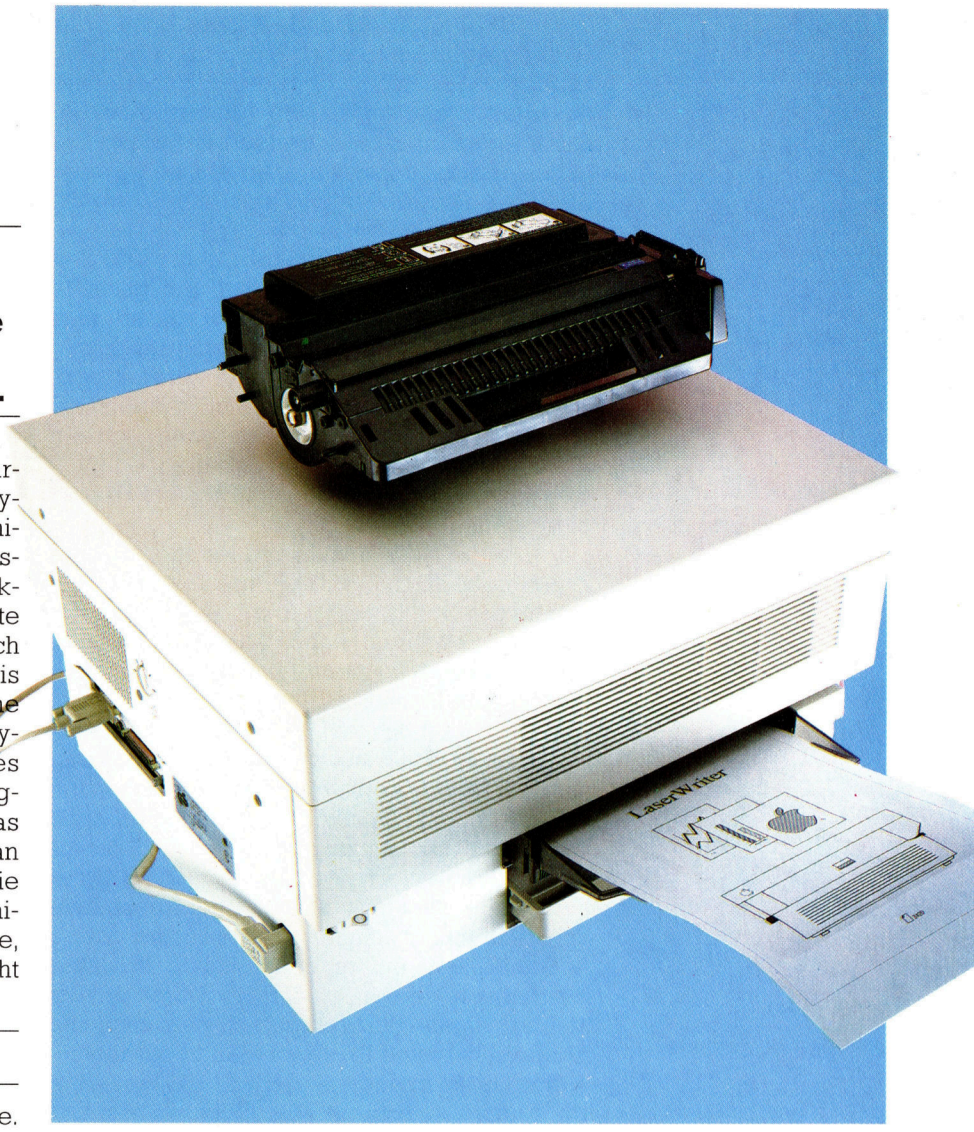
Hohe Geschwindigkeiten

Laserdrucker haben keinen dieser Nachteile. Sie erzeugen mit phänomenaler Geschwindigkeit ganze Seiten mit hoher Druckqualität. So produziert der auf die Groß-EDV ausgerichtete und rund 800 000 Mark teure Laserdrucker IBM 3800 Modell 3 jede Minute 215 Seiten Text.

Der LaserWriter von Apple ist nur ein kleines Abbild dieser Hochgeschwindigkeitsdrukker im IBM-Stil. Er druckt pro Sekunde 300 Zeichen von hoher Qualität und verfügt über mehrere Schriftarten, die über ein Menü angewählt werden. Der Drucker zeichnet auch Diagramme, Formulare und grafische Darstellungen, deren Qualität weit über der herkömmlicher Drucker liegt.

Das Gerät ist für die Arbeit mit dem Macintosh gedacht und wird über die normale serielle Hochgeschwindigkeitsschnittstelle für Drucker angeschlossen.

Die Arbeitsweise des LaserWriters unterscheidet sich grundlegend von der herkömmlicher Drucker. Während Matrix- oder Typenraddrucker ihre Druckköpfe über eine mit Papier belegte Walze bewegen und damit eigentlich eine Weiterentwicklung der Schreibmaschine sind, ähnelt der LaserWriter eher einer guten Fotokopiermaschine.



Der Drucker hat eine Größe von 290 × 420 × 475 mm. In dem Gehäuse befinden sich zwei Trommeln, von denen eine das Färbungsmittel (Toner) und die andere die Lasertechnik und einen rotierenden Zylinder enthält, der Schrift und Grafik auf das Papier überträgt.

Der von Canon entwickelte Mechanismus setzt die Druckseite zunächst in ein Bitmuster (Bit-Image) um. Das Bild wird dann auf den Zylinder übertragen, auf dessen Oberfläche sich Millionen elektroempfindlicher Punkte befinden. Diese Punkte werden vor der Übertragung ionisiert und damit positiv geladen. Der Laser läuft nun über die Oberfläche des Zylinders und „schreibt" so den Seiteninhalt darauf. Dabei werden alle vom Laserstrahl getroffenen Punkte neutralisiert. Nach Beendigung dieses Vorgangs wird das ebenfalls positiv geladene Färbungsmittel über die Zylinderoberfläche verteilt. Es setzt sich nur auf die neutralisierten Punkte, da die positiv geladenen Punkte das Färbungsmittel abstoßen.

Beim Einzug in die Maschine wird das Papier negativ geladen. Sobald das Blatt die Trommel passiert, zieht es die positiv gela-

Obwohl der Laser Writer für Microcomputer gedacht ist, hat er keine Ähnlichkeit mit den herkömmlichen Matrix- und Typenraddruckern. Seine Konstruktion gleicht eher den schnellen Laserdruckern der mittleren Datentechnik und der Groß-EDV, die Hunderttausende kosten. Der LaserWriter ist langsamer als seine großen Vorbilder, setzt aber die gleiche Technik ein.

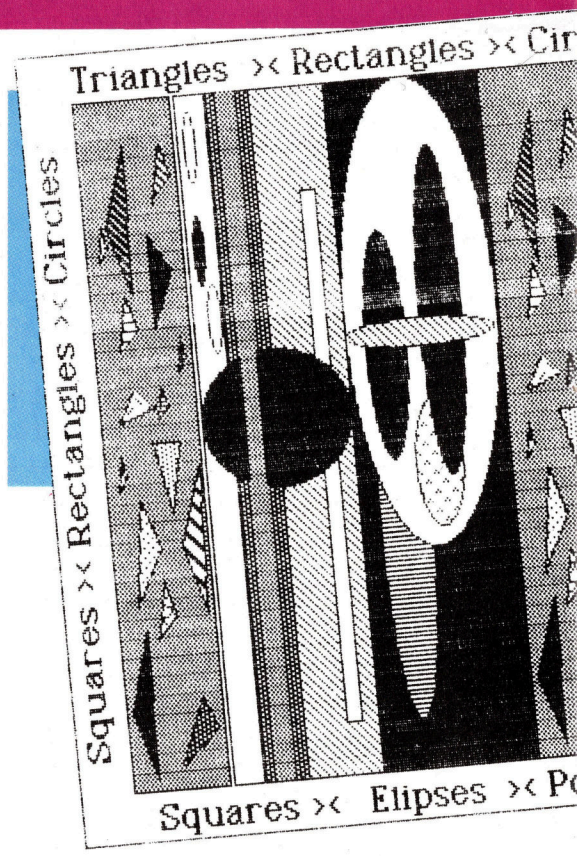
dene Färbung an. Nachdem zwei Rollen das Färbungsmittel mit einer Temperatur von 200 Grad auf das Papier übertragen haben, wird das Blatt ausgeworfen. Pro Minute kann der LaserWriter bis zu acht Seiten bedrucken. Die hohe Arbeitstemperatur der Maschine wird durch die Fixierrollen und nicht durch den Laser erzeugt.

Eine Farb-Kartusche reicht für ca. 3000 Seiten. Beim Toner-Austausch muß die gesamte Trommel ausgewechselt werden, da sie sich nicht wieder füllen läßt. Der Austausch kostet etwa 400 Mark und stellt somit einen erheblichen Kostenfaktor dar. Apple führt jedoch an, daß der Preis für eine Kopie bei etwa dreizehn Pfennig liegt und der LaserWriter so lediglich die gleichen Kosten verursacht wie ein Fotokopiergerät.

Durch die hohe Anzahl von Punkten auf der Zylinderoberfläche – 300 Punkte per Zoll – kommt die Druckqualität des LaserWriters der von Satzmaschinen nahe. Für eine A4-Seite beispielsweise kommen fast sieben Millionen Punkte zum Einsatz.

In bezug auf Schnelligkeit und Druckqualität ist der LaserWriter bei weitem nicht einzigartig auf dem Markt. Apple gelang es jedoch, seine eigene Technik in die Maschine einzubauen und damit den ersten „intelligenten" Drucker dieser Art herzustellen. Da der LaserWriter eine Seite zunächst als Bit-Map in seinem Speicher anlegt, mußte er wegen der hohen Auflösung mit einem beachtlichen Speicher ausgestattet werden. Zu diesem Zweck verfügt er über einen „Bordcomputer", der allein die Aufgabe hat, eine Druckseite vorzubereiten.

Das Gerät arbeitet mit dem 68 000-Prozessor von Motorola (auch im Macintosh eingesetzt), der mit 12 MHz getaktet wurde und über 1,5 MByte RAM und 500 KByte ROM verfügt. Im ROM sind die Zeichensätze von dreizehn Schriftarten gespeichert. Doch selbst zwei MByte können nicht die Bitmuster aller Zeichen und Anwendungsprogramme enthalten, die der LaserWriter benötigt. Um Speicherplatz zu sparen, enthält das ROM des Druckers daher nur die Umrisse der Zeichen und eine



Routine, die diese Linien beim Drucken ausfüllt. Das System spart damit nicht nur Speicherplatz, sondern kann die Umrisse auch auf jede gewünschte Größe bringen.

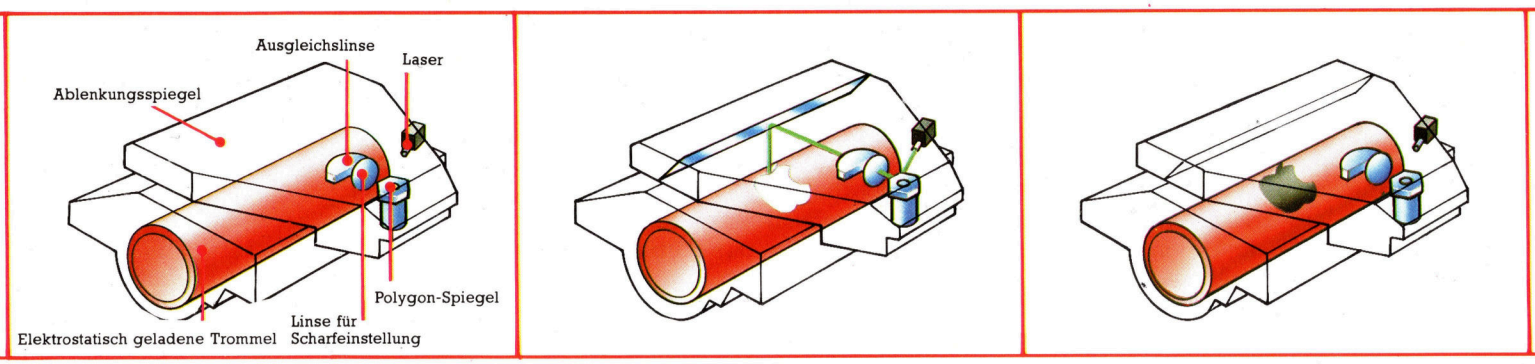
PostScript im ROM

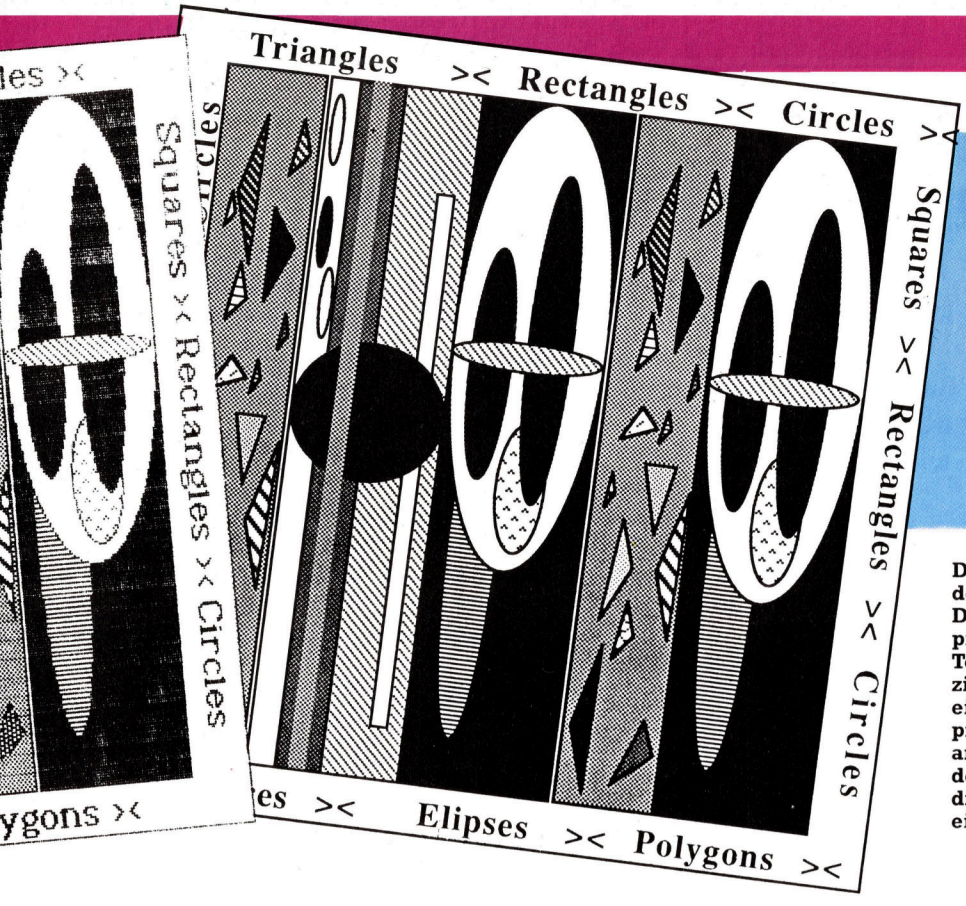
Im ROM liegt außerdem die speziell für Drukker entwickelte Programmiersprache PostScript. PostScript ist eine interaktive Programmiersprache, über die sich Stil und Aufbau der im ROM gespeicherten Umrisse verändern lassen. Mit den Befehlen dieser Sprache lassen sich auch Routinen schreiben, die Kreise etc. erzeugen, aus denen wiederum komplizierte Diagramme zusammengestellt werden. PostScript ähnelt FORTH, da man neue Prozeduren definieren kann und viel mit dem Stack arbeitet, um Daten zwischen Prozeduren auszutauschen. Die Arbeitsweise von PostScript unter-

Die Trommel wird elektrostatisch positiv geladen.

Der Laserstrahl wird ausgelöst, durch Spiegel auf die Trommel gelenkt und läßt dort das Bild aus neutralisierten Ionen entstehen.

Sobald die Oberfläche der Trommel an dem Behälter mit dem Färbungsmittel (Toner) vorbeiläuft, ziehen die „neutralisierten" Bereiche ...





Qualitätsvergleich

Dieses Bild wurde einmal auf dem Matrixdrucker (Macintosh Imagewriter) und einmal mit dem LaserWriter erzeugt. Abgesehen von den Ungenauigkeiten, die bei dem Matrixdrucker durch die unterschiedliche Farbdichte des Farbbandes entstehen, ist die höhere Auflösung des LaserWriters deutlich zu erkennen. Untersucht man den Druck des Imagewriters genauer, so zeigen sich in den Bereichen unbeabsichtigte Linien, an denen sich die Zeichenbereiche des Druckkopfes überschneiden. Bei dem LaserWriter gibt es dieses Problem nicht, da das Bild als Ganzes auf eine Trommel aufgetragen wird.

Die Bildfolge zeigt, wie der LaserWriter eine Darstellung auf das Papier bringt. Obwohl die Technik sehr kompliziert ist, beruht sie auf einfachen Funktionsprinzipien. Das System arbeitet mit den Methoden der Fotokopierer, die schon seit Jahren eingesetzt werden.

Apple LaserWriter

ABMESSUNGEN
290 x 420 x 475 mm

SCHNITTSTELLEN
RS422 und RS232C seriell

DRUCKAUFLÖSUNG
300 Punkte pro Zoll

ZENTRALEINHEIT
Motorola 68000 mit 12 MHz getaktet

SPEICHERKAPAZITÄT
1,5 MByte RAM, 500 KByte ROM

STÄRKEN
Der LaserWriter erzeugt Buchstaben und Diagramme von hoher Qualität.

SCHWÄCHEN
Da die maximale Papierhöhe nur etwas über dem A4-Format liegt, läßt sich das Gerät nur beschränkt für Grafik und Design einsetzen. Die Maschine ist sehr teuer.

scheidet sich erheblich von der anderer Grafiksprachen wie beispielsweise LOGO. Statt dem Drucker absolute Positionen mitzuteilen, nimmt PostScript eine Reihe Punkte (beispielsweise die Umrisse der Zeichen) und verteilt sie entsprechend der Proportionsangaben.

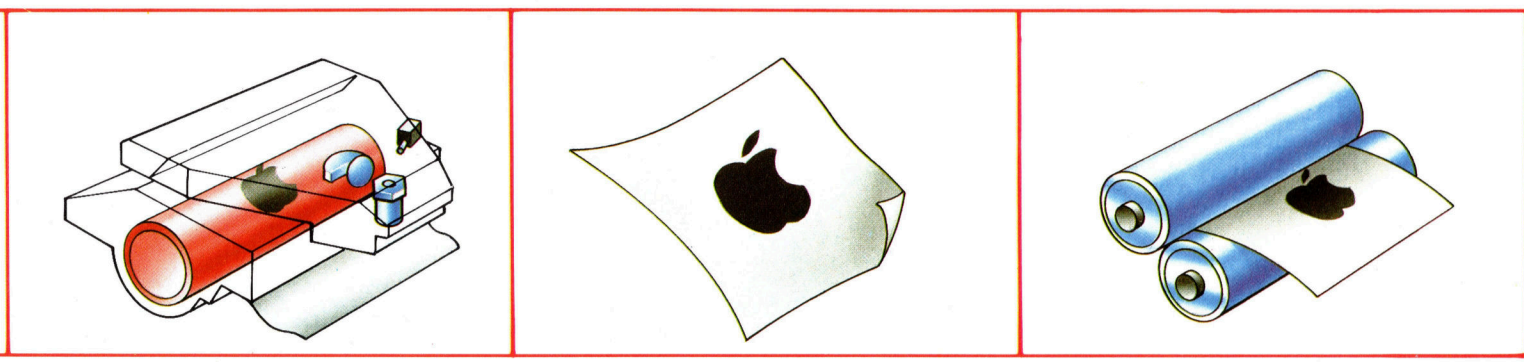
PostScript ist jedoch mehr als nur eine Zugabe – ohne diese Sprache kann der LaserWriter nicht optimal funktionieren. Für die Übertragung eines MBytes (die Information für eine vollständige Druckseite) würde der Macintosh rund 35 Sekunden brauchen. Die Programme der Systemdiskette, die zum Lieferumfang des LaserWriters gehört, übersetzen diese Information jedoch in das Quickdraw-Format des Macintosh. Die Daten werden dann über die serielle Leitung an den LaserWriter gesandt und dort in PostScript-Befehle umgewandelt. Der Computer des LaserWriters interpretiert nun diese Befehle und legt in seinem Speicher das Bitmuster dieser Seite an. Schließlich „zeichnet" der Laser mit diesem Bitmuster die Seite auf die elektrisch geladene Zylinderoberfläche. Durch die Komprimierung der notwendigen Informationen auf acht KByte verringert sich die Übertragungs- und Verarbeitungszeit drastisch.

Der LaserWriter hat zweifellos nicht nur die Entwicklung des Druckmechanismus bedeutend vorangebracht, sondern auch die Konstruktion von „Bordcomputern" und den Einsatz von PostScript. Der Preis des Gerätes ist natürlich für die meisten Besitzer eines Macintosh viel zu hoch (etwa 27 000 Mark). Da neue Techniken jedoch oft schon nach kurzer Zeit auf preiswerteren Maschinen verfügbar sind, wird es sicherlich in wenigen Jahren erschwingliche Laserdrucker geben. Aber nur wenn die Größe des Geräts verringert wird, ist es für Privatanwender sinnvoll.

... den positiv geladenen Toner an. Das negativ geladene Papier wird in die Trommel eingeführt, ...

... wo es den positiv geladenen Toner anzieht und damit das Bild auf die Seite überträgt.

Schließlich wird das Papier mit dem Toner zwischen zwei erhitzten Rollen hindurchgeführt, die den Toner fixieren.



Lernprogramme

Dies ist der zweite Teil unserer kleinen Serie, in der wir uns mit Ausbildungsprogrammen für Kinder befassen. Wir stellen zwei Programme der Firma Spinnaker vor und geben einen Überblick der von Macmillan Software in Zusammenarbeit mit Sinclair Research entwickelten Programme.

Die „Story Machine" von Spinnaker ist für Kinder zwischen fünf und neun Jahren gedacht. Es will Syntax-Regeln vermitteln, die Rechtschreibung verbessern und zum ausdrucksvolleren Schreiben anregen.

Das Programm enthält ein 52 Wörter umfassendes „Lexikon". Aus dem Wörterbuch baut das Kind nun einfache Sätze, die dann von den Charakteren auf dem Bildschirm in Form von Handlungen ausgeführt werden. So bewirkt die Eingabe des Satzes „Maurice küßt Blumen", daß auf dem Bildschirm ein Junge dargestellt wird (vom Benutzer zuvor als „Maurice" spezifiziert) und eine Figur, die eine Blume darstellt. Wenn beide dicht beieinander stehen, tauchen blinkende Herzen auf, die den Kuß signalisieren.

Wird ein Wort falsch geschrieben oder falsch benutzt, nimmt es der Computer nicht an. Er gibt statt dessen eine Botschaft, in der erklärt wird, was falsch ist, und fragt nach dem zu ändernden oder auszutauschenden Wort.

Hat man das Selberschreiben von Geschichten satt, bietet „Story Maker" zwei andere Optionen: Entweder teilt man sich das Schreiben mit dem Programm (wobei das Einsetzen von Wörtern wechselweise erfolgt) oder man läßt den Computer die Geschichte allein schreiben. Das ist möglich, weil das Wörterbuch in Teile aufgegliedert ist – Substantive, Verben, Präpositionen etc. – und über einen genauen grammatikalischen Algorithmus verfügt, der eigene Sätze zusammenstellen kann.

Die meisten der im Wörterverzeichnis enthaltenen Wörter sind ebenso wie die im Programm verwendeten Satzbau-Strukturen sehr einfach. Besonders die Verben beschreiben leicht ausführbare Aktionen wie etwa „hüpfen" und „gehen".

Das Programm enthält zwei fundamentale grammatikalische Regeln: Ein Satz muß aus Subjekt, Prädikat und Objekt in der richtigen Reihenfolge bestehen. Singular wie Plural-Formen müssen stimmig sein. Abgesehen von diesen Regeln ist die Syntax-Lehre nicht gerade zufriedenstellend – wegen der vielfältigen Unregelmäßigkeiten der englischen Sprache. So akzeptiert der Computer den Satz „Houses eat rocks", nicht aber den Satz „Girls eat rocks", solange nicht der bestimmte Artikel vor „Girls" steht.

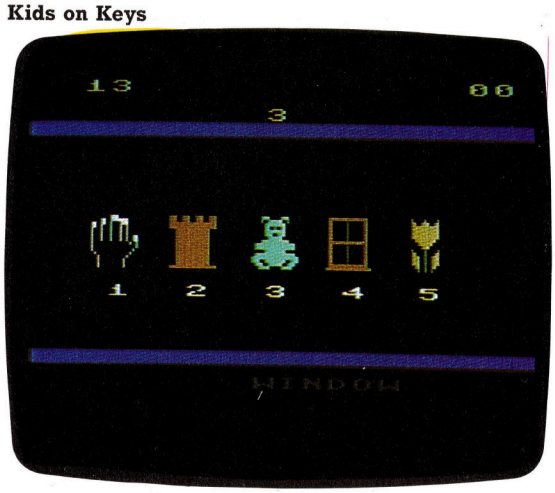
Grammatikfehler verboten

Im Programm wird die Regel, daß Singular- und Pluralformen nicht gemischt werden dürfen, nicht immer eingehalten. Das mag ein relativ kleiner Fehler sein, doch ein Programm, das Lehrzwecken dienen soll, muß immer exakt sein und darf keine Ausnahme zulassen.

„Kids On Keys" ist ein weiteres Spinnaker-Produkt für Kinder zwischen drei und neun Jahren. Programmzweck ist die Vermittlung von Fähigkeiten zur Wort-, Buchstaben- und Zahlen-Erkennung. Das Programm enthält drei verschiedene Spiele, die jeweils über unter-

Story Machine

Kids on Keys



schiedliche Schwierigkeitsgrade verfügen. Im ersten Spiel läuft ein Buchstabe über den Bildschirm. Das Kind muß die richtige Taste drükken, bevor der Buchstabe den unteren Bildschirmrand erreicht. Nach fünfzehn Buchstaben sinkt ein Ballon herab, der ein Wort enthält. Dieses muß richtig eingegeben werden, bevor der Ballon den Bildschirmrand erreicht. Das Spiel bietet so eine interessante Möglichkeit, das Kind mit der Computertastatur vertraut zu machen.

Im zweiten Spiel läuft eine Bildserie über den Bildschirm. Das Kind muß jeweils den Namen des Gegenstands eingeben, bevor dieser den Bildschirmrand erreicht. Darauf folgt eine „Bonus"-Runde, in der dasselbe Objekt herabsinkt. Dabei fehlt jedoch die Hälfte des Bildes, was die Identifizierung des Gegenstandes schwerer macht. In dieser Phase entwickelt sich das Programm zu einem Test- bzw. Reaktions-Spiel, da der Spieler eilig versucht, die richtigen Buchstaben einzugeben. Einer der Hauptnachteile des Programms besteht darin, daß falsche Eingaben nicht korrigiert werden können.

Die Schwäche des Programms ist, daß einige Sprites außerordentlich schlecht gezeichnet und schwer zu identifizieren sind. So mag ein Kind entscheiden, daß es sich bei einer bestimmten Darstellung um einen Mann handelt. Gibt es das Wort „Mann" in der richtigen Schreibweise ein, wird es feststellen, daß der Sprite weiter herabsinkt, weil das Programm ihn als „Bären" oder „Jungen" betrachtet. Das Kind könnte denken, daß die Rechtschreibung falsch war, da das Programm den Grund für die Fehleranzeige nicht erläutert (es verfügt auch nicht über eine entsprechende Analysemöglichkeit). Ferner: Hat das Sprite den unteren Bildschirmrand erreicht, gibt das Programm keine Auskunft darüber, wie die richtige Antwort hätte lauten müssen. Ein gutes Ausbildungsprogramm würde dem Benutzer nicht nur einen zweiten Lösungsversuch gestatten, sondern auch nach Eingabe mehrerer falscher Antworten die korrekte und eine dazugehörige Begründung liefern.

Im dritten Spiel werden fünf Bilder und ein Wort auf dem Bildschirm dargestellt. Das Kind soll nun dem Wort das richtige Bild zuordnen. Dieses Programm ist nicht so schnell wie die anderen, seine Grafiken aber sind ebenfalls recht dürftig.

Einfach lesen lernen

Das englische Softwarehaus Macmillan hat für den Spectrum eine Programmreihe mit dem Titel „Learn To Read" entwickelt. Die fünf Programme – gemeinsam mit Sinclair Research gestaltet – basieren auf Macmillans sehr erfolgreichem „Gay Way"-Lesebuch, das in englischen Grundschulen verwendet wird. Da Gay Way am einzelnen Schüler orientiert ist, hat es den Vorteil, daß der Benutzer mit seiner ganz „persönlichen" Geschwindigkeit lesen lernt. Die entsprechenden Computerprogramme sind eine folgerichtige Weiterführung, da das Lernen mit dem Computer ebenfalls individuell orientiert ist.

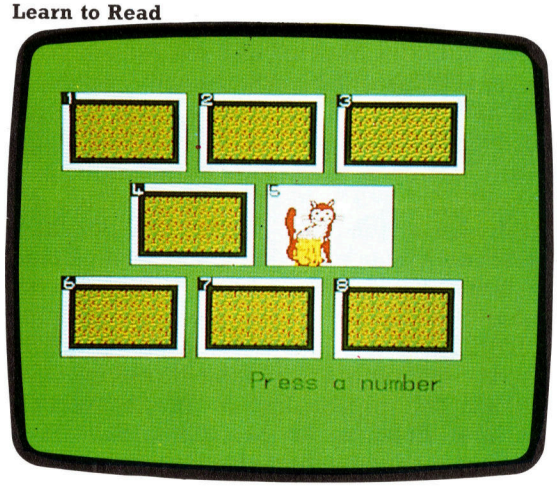
Aus den fünf Programmen ergibt sich ein vollständiger Kurs zum Lesenlernen. Im ersten Programm werden sechs Tiere vorgestellt (Deb, die Ratte, Sam, der Fuchs etc.), und sie erscheinen in allen Folgen wieder. Daraus ergibt sich ein inniges Vertrautheitsgefühl für das Kind.

Es wird davon ausgegangen, daß das Kind weder über Computer- noch über Lesevorkenntnisse verfügt. So zeigt beispielsweise das erste Programm fünf Optionen, und ein blinkendes Rechteck umgibt den Namen jeder Option. Der Benutzer wartet, bis sich der rechteckige „Cursor" von allein um die gewünschte Option legt, und drückt dann eine beliebige Taste. Zur Benutzung dieses Menüs ist keine weitere Anweisung erforderlich.

Nach Wahl einer Option zeigt der Computer, was zu tun ist, und fordert dann das Kind zur Antwort auf. Antwortet das Kind falsch, darf es mehrmals verbessern, bevor der Computer die richtige Antwort zeigt.

Learn to Read

Learn to Read



Kalibrierung

Unser Robot-Auto wurde bereits mit zwei Motoren und vier Tastsensoren ausgestattet, mit denen es seine Umwelt erkunden soll. Damit es vorgeschriebene Wege zurücklegen kann, muß es kalibriert werden. Danach kann mit einem kurzen Programm die Zusammenarbeit zwischen Motoren und Sensoren getestet werden.

Schrittmotoren eignen sich besonders gut für die digitale Steuerung, weil sie sich bei jedem empfangenen Impuls um einen genau festgelegten Winkel weiterdrehen. Die Ansteuerung der Motoren soll von der Umgebung des Roboters abhängen. Die Anzahl der für das Zurücklegen bestimmter Strecken und Winkel nötigen Impulse wird experimentell festgestellt. Wir brauchen zwei Mittelwerte für die Verhältnisse Impuls/Strecke bzw. Impuls/Winkel, die als Konstanten in die Steuer-Software übertragen werden. Später werden wir auch Programme entwickeln, mit denen der Roboter selbst ein Bild der Objekte entwirft, die seinen Bewegungsspielraum einengen. Die Werte, die für eine exakte Steuerung des Roboters erforderlich sind, sollen programmgesteuert errechnet werden.

Impuls-Strecken-Verhältnis

Das Verhältnis Impulse/zurückgelegte Strecke läßt sich mit einer kleinen Rechnung grob abschätzen: Ein Impuls dreht den Motor um 7,5 Grad weiter. Die Getriebeuntersetzung von 25:2 reduziert diesen Wert auf eine Achsenrotation von nur noch 0,6 Grad. Wenn das verwendete Lego-Rad einen Radius von 30 mm hat, ergibt sich: ein Impuls = 0,6/360 × 2 × Pi × 30 mm zurückgelegte Strecke. Ein Impuls erzeugt also eine Bewegung um 0,1 × Pi mm. Theoretisch ergibt sich die Maßzahl 3,183 für das Verhältnis Impuls/Strecke.

Mit unserem Kalibrier-Programm können Sie Ihren Roboter beim Durchfahren verschieden langer Wege testen. Bei jedem Versuch wird die Anzahl der Impulse und die rechnerische Streckenlänge auf dem Bildschirm angezeigt. Mit zwei 30 cm langen Linealen läßt sich die tatsächlich zurückgelegte Strecke ermitteln. Diese Meßwerte nutzt das Programm für eine Tabelle, die sich aus den Impulszahlen, der tatsächlich gefahrenen Strecke und der theoretischen Vorhersage zusammensetzt. Außerdem wird ein Mittelwert des Verhältnisses Impulse/Strecke berechnet, den Sie notieren sollten. Es könnte sich etwa ergeben, daß Ihr Roboter ein wenig weiter fährt als erwartet. Sie müssen das genaue Impuls/Streckenverhältnis Ihres Roboters kennen, damit zukünftige Programme auch richtig ablaufen.

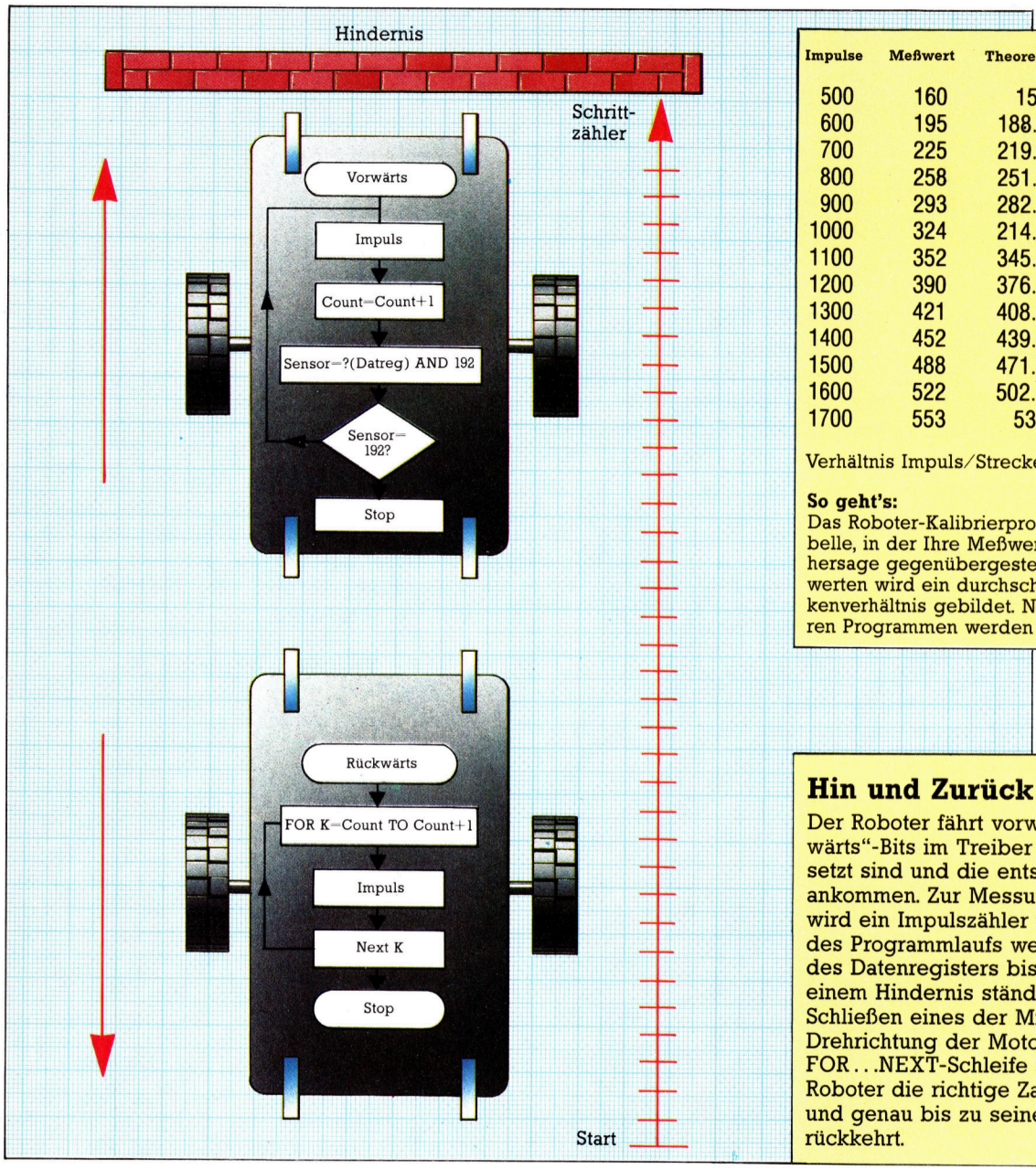
```
 10 REM **** BBC CALIBRATION ****
 20 DDR=&FE62:DATREG=&FE60
 30 ?DDR=15:REM LINES 0-3 OUTPUT
 50 forwards=4:backwards=2:DIM MD(12)
 60 FOR CC=500 TO 1700 STEP 100
 70 ?DATREG=0
 80 ?DATREG=(?DATREG OR 1) OR forwards
 90 PRINT CC,INT(CC*PI)/10
100 A$=GET$
110 FOR I=1 TO CC
120 PROCpulse
130 NEXT I
140 INPUT"MEASURED DISTANCE IN MM";MD((CC-500)/100)
150 NEXT CC
160 ?DATREG=0:T=0
180 PRINT"    PULSES","  MEASURED","  THEORET."
190 PRINT
200 FOR CC=500 TO 1700 STEP 100
210 PRINT CC,MD((CC-500)/100),INT(CC*PI)/10
220 T=T+CC/MD((CC-500)/100)
230 NEXT CC
240 PRINT:PRINT "PULSE TO DISTANCE RATIO:",T/12
260 END
270 DEF PROCpulse
280 ?DATREG=(?DATREG OR 8)
290 ?DATREG=(?DATREG AND 247)
300 ENDPROC
```

```
10 REM **** CBM 64 CALIBRATION ****
20 DDR=56579:DATREG=56577
30 POKE DDR,15:REM LINES 0-3 OUTPUT
50 FW=4:BW=2:DIM MD(12)
60 FOR CC=500 TO 1700 STEP 100
70 POKE DATREG,0
80 POKE DATREG,(PEEK(DATREG)OR 1)OR FW
90 PRINT CC,INT(CC*.1)/10
100 GET A$:IF A$="" THEN 100
110 FOR I=1 TO CC
120 GOSUB 270:REM PULSE
130 NEXT I
140 INPUT"MEASURED DISTANCE IN MM";MD((CC-500)/100)
150 NEXT CC
160 POKE DATREG,0:T=0
170 REM ** LINES 180-260 AS BBC VERSION **
175 REM ** BUT REPLACE PI BY .1 IN LINE 210 **
270 REM **** PULSE S/R ****
280 POKE DATREG,PEEK(DATREG)OR 8
290 POKE DATREG,PEEK(DATREG)AND 247
300 RETURN
```

Das Verhältnis Impuls/Winkel hängt mit dem Radabstand zusammen: Bei 140 mm ergibt sich für einen Vollkreis der Umfang von 140 mm × Pi. Ein einzelner Impuls erzeugt einen Winkel von 360 × 0,1 × Pi/(140 × Pi) Grad. Das Verhältnis Impuls/Winkel beträgt also 3,846.

Das genaue Messen eines Winkels ist nicht ganz einfach. Bei den meisten Anwendungen soll sich das Fahrzeug um 90 Grad (oder ein Vielfaches davon) drehen. Für ein rechtwinkliges Abbiegen ergibt sich rechnerisch der Wert von 346 Impulsen.

Zeichnen Sie zwei Linien im rechten Winkel auf einen Bogen Papier. Neben eine der Linien werden zwei Punkte als Startmarkierung für die Robot-Räder eingezeichnet. Nun wird der Roboter aufgestellt und das Programm für die 90-Grad-Kurve gestartet. Die FOR...NEXT-Schleife in Zeile 70 bestimmt die Anzahl der Impulse. Der im Programm benutzte Wert von 371 dient lediglich zum Experimentieren. Ver-



Impulse	Meßwert	Theoretischer Wert
500	160	157
600	195	188.4
700	225	219.9
800	258	251.3
900	293	282.7
1000	324	214.1
1100	352	345.5
1200	390	376.9
1300	421	408.4
1400	452	439.8
1500	488	471.2
1600	522	502.6
1700	553	534

Verhältnis Impuls/Strecke: 3.34767511

So geht's:
Das Roboter-Kalibrierprogramm erzeugt eine Tabelle, in der Ihre Meßwerte der theoretischen Vorhersage gegenübergestellt sind. Aus zwölf Meßwerten wird ein durchschnittliches Impuls/Strekkenverhältnis gebildet. Notieren Sie es, bei späteren Programmen werden die Zahlen gebraucht!

Hin und Zurück

Der Roboter fährt vorwärts, wenn die „Vorwärts"-Bits im Treiber des Schrittmotors gesetzt sind und die entsprechenden Impulse ankommen. Zur Messung der Fahrstrecke wird ein Impulszähler verwendet. Während des Programmlaufs werden die Sensor-Bits des Datenregisters bis zur Kollision mit einem Hindernis ständig abgefragt. Durch Schließen eines der Microschalter wird die Drehrichtung der Motoren umgeschaltet. Eine FOR...NEXT-Schleife sorgt dafür, daß der Roboter die richtige Zahl von Impulsen erhält und genau bis zu seinem Ausgangspunkt zurückkehrt.

ändern Sie diese Zahl so lange, bis die Räder nach dem Durchfahren der Kurve genau parallel mit der zweiten Linie stehen.

Das Linksabbiegen

Als nächstes wird die Richtung „right"(RT) in Zeile 60 durch „left"(LT) ersetzt, um sicherzustellen, daß Ihr Roboter auch im Gegenuhrzeigersinn richtig abbiegt. Ein Verdoppeln des Endwertes der FOR...NEXT-Schleife sollte eine Wendung um 180 Grad ergeben. Dabei müssen die Räder hinterher wieder auf dem Startpunkt zum Stehen kommen. Ist das nicht der Fall, sollten Sie die Räder justieren: Beide müssen den gleichen Abstand von der Fahrzeugmitte haben. Dann kann die Radposition auf den Achsen eingezeichnet werden, und die Räder fixiert man mit etwas Kleber.

```
10 REM **** CBM 64 TURN ****
20 DDR=56579:DATREG=56577
30 POKE DDR,15:REM LINES 0-3 OUTPUT
40 LF=6:RT=0
50 POKE DATREG,0
60 POKE DATREG,(PEEK(DATREG)OR 1)OR RT
70 FOR I=1 TO 371:GOSUB 90:NEXT I
80 POKE DATREG,0:END
90 REM **** PULSE S/R ****
100 POKE DATREG,PEEK(DATREG)OR 8
110 POKE DATREG,PEEK(DATREG)AND 247
120 RETURN
```

```
 10 REM **** BBC TURN ****
 20 DDR=&FE62:DATREG=&FE60
 30 ?DDR=15:REM LINES 0-3 OUTPUT
 40 left=6:right=0
 50 ?DATREG=0
 60 ?DATREG=(?DATREG OR 1)OR right
 70 FOR I=1 TO 371:PROCpulse:NEXT
 80 ?DATREG=0:END
 90 DEF PROCpulse
100 ?DATREG=(?DATREG OR 8)
110 ?DATREG=(?DATREG AND 247)
120 ENDPROC
```

```
10 REM **** CBM BUMPERS ****
20 DDR=56579:DATREG=56577
30 POKE DDR,15:PEM LINES 0-3 OUTPUT
40 FW=4:BW=2
50 POKE DATREG,(PEEK(DATREG)OR 1)OR FW
60 REM **** PULSE FORWARDS ****
```

```
65 CC=0
70 GOSUB 1000:CC=CC+1:REM PULSE
80 IF (PEEK(DATREG)AND 192)=192 THEN 70
90 REM **** GO BACK TO START ****
95 POKE DATREG,(PEEK(DATREG)AND 1)OR BW
100 FOR I=1 TO CC
110 GOSUB 1000:REM PULSE
120 NEXT I
130 POKE DATREG,0:END
1000 REM **** PULSE S/R ****
1010 POKE DATREG,(PEEK(DATREG)OR 8)
1020 POKE DATREG,(PEEK(DATREG)AND 247)
1030 RETURN
```

```
10 REM **** BBC BUMPERS ****
20 DDR=&FE62:DATREG=&FE60
30 ?DDR=15:REM LINES 0-3 OUTPUT
40 forwards=4:backwards=2
50 ?DATREG=(?DATREG OR 1) OR forwards
60 REM **** PULSE FORWARDS ****
65 count=0
70 REPEAT:PROCpulse:count=count+1
80 UNTIL(?DATREG AND 192)<>192
90 REM **** GO BACK TO START ****
95 ?DATREG=(?DATREG AND 1)OR backwards
100 FOR I=1 TO count
110 PROCpulse
120 NEXT I
130 ?DATREG=0:END
1000 DEF PROCpulse
1010 ?DATREG=(?DATREG OR 8)
1020 ?DATREG=(?DATREG AND 247)
1030 ENDPROC
```

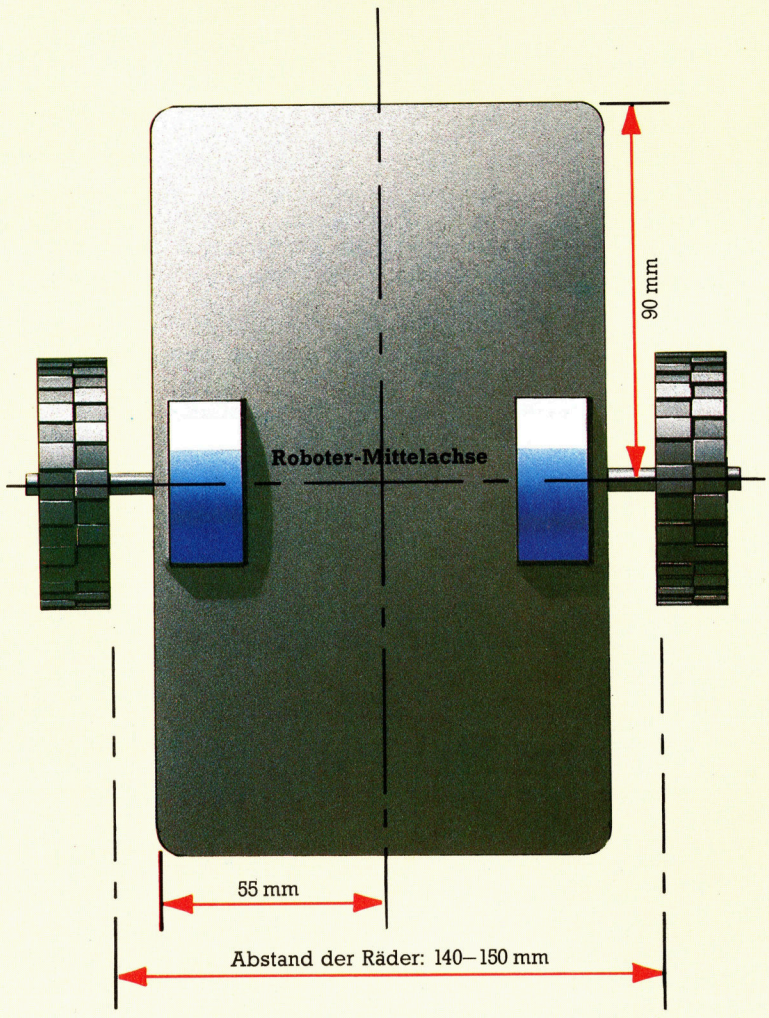
Geeignete Software kann die mit den Sensoren ermittelten Informationen über die Roboter-Umgebung auswerten und zur Steuerung nutzen. Ein einfaches Programm führt den Roboter beispielsweise beim Aufprall an einem Hindernis wieder zur Startposition zurück. Hier die Programmstruktur:

1. Datenrichtungsregister auf 15 setzen. Setzt Bit 0–3 auf Ausgabe, Bit 4–7 auf Eingabe.
2. Drehrichtung der Motoren auf vorwärts schalten.
3. Motor drehen, bis Bit 6 oder 7 auf Low geht, Impulse zählen und speichern.
4. Drehrichtung auf rückwärts schalten.
5. Anzahl der Impulse nach Größe der Variablen „count" an die Motoren schicken.
6. Datenregister auf Null, Programmende.

In diesem Programm werden die vom Steckerfeld weiter entfernten Sensoren als die vorderen angesehen. Sie sind durch zwei kurze Verbindungskabel zwischen den roten und blauen Buchsenpaaren auf der rechten Seite mit den Bits 6 und 7 gekoppelt. In Zukunft werden wir den D-Stecker auf der Oberseite des Gehäuses immer als vorn ansehen. Falls Ihr Roboter beim Programmablauf zuerst rückwärts fährt, müssen Sie nur den Gehäusedeckel abnehmen und um 180 Grad gedreht wieder festschrauben.

Radabstand einstellen

Eventuell müssen Sie Ihr Robot-Auto noch etwas nachjustieren, damit es sich genau kalibrieren läßt. Zuerst auf der Unterseite die Mittelachse markieren – am besten mit einem spitzen Nagel oder einem Filzstift. Der Abstand zwischen den Innenseiten der Räder sollte zwischen 140 und 150 mm liegen. Der Roboter soll beim Abbiegen um einen genau definierten Punkt drehen – beide Räder müssen also exakt den gleichen Abstand von der eingezeichneten Mittelachse haben. Justieren Sie den Abstand durch vorsichtiges Verschieben der Räder auf ihren Achsen.



Die Laufrichtung

Von den niederwertigen vier Bits des Datenregisters ist Bit 0 das „Ruhe"-Bit (normalerweise auf Eins), Bit 1 und 2 steuern die Laufrichtung des rechten und linken Motors. Bit 3 sorgt bei jedem Sprung von Low nach High für das Weiterdrehen beider Motoren um einen Schritt. Die Logikfunktionen AND und OR ermöglichen die Veränderung einzelner Bits, ohne den restlichen Registerinhalt zu beeinflussen. Die vier höherwertigen Bits sind durch das Datenrichtungs-Register auf Eingabe geschaltet und werden auf High gehalten. Sobald sich ein Microschalter schließt, geht das entsprechende Datenregister auf Low. Normalerweise haben Bit 6 und 7 bei der Stellung „Eingabe" den Wert 192 (128 + 64). Steht ein anderer Wert im Register, wird die Schleife für das Weiterfahren (Zeile 70–80) abgebrochen. Das geschieht, sobald einer oder beide Microschalter geschlossen sind. Die Anzahl der Impulse zu den Motoren ist beim Fahren gespeichert worden, so daß der Roboter durch einfaches Umschalten der Fahrtrichtung und erneutes Ansteuern mit der gespeicherten Zahl von Impulsen zum Ausgangspunkt zurückkehrt.

Auffällig ist, daß der Roboter rückwärts schneller fährt als vorwärts. Die Ursache sind unterschiedliche BASIC-Verarbeitungszeiten: Beim Vorwärtsfahren muß der Computer zwischen zwei Impulsen noch den Impulszähler steuern und prüfen, ob ein Sensor Berührung meldet. Diese Aktivitäten fallen beim Rückwärtsfahren fort.

Eine kurze Unterbrechung dieses Projekts soll Ihnen die Zeit zur Komplettierung Ihres Roboters geben. Die beiden nächsten Kursabschnitte behandeln die Ansteuerung von Servomotoren.

Geisterstunde

Im letzten Artikel über unser Abenteuerspiel-Projekt haben wir mit der Handhabung der speziellen Orte begonnen. In diesem Teil werden wir die Tunnel-Routine fertig ausarbeiten sowie eine weitere Routine entwerfen, die Geister im Wald erscheinen läßt.

Im vorangegangenen Abschnitt wurden die speziellen Eigenschaften der Tunneleingänge behandelt. An diesen Orten hat der Spieler die Möglichkeit, den Tunnel zu betreten oder den Weg zurückzugehen. Betritt der Spieler den Tunnel, wird die Unterroutine bei Zeile 4655 aufgerufen. Nun zu der Routine, die das Betreten des Tunnels handhabt. Sie ist nach bestimmten, vom Programmierer festgelegten Regeln geschrieben. Erstens kann der Spieler nur dann durch den Tunnel gehen, wenn er die Lampe hat, und zweitens muß er die Lampe rechtzeitig anzünden, um zu sehen, wohin er geht.

Da der Spieler auch im Tunnel Befehle geben muß, sollte die Unterroutine mit einem Abschnitt beginnen, der die Eingabe von Befehlen akzeptiert und diese zur weiteren Verarbeitung aufteilt. Wir können dem Spieler die Verwendung einiger normaler Befehle, wie TAKE, DROP, LIST oder END erlauben. Doch dabei ist Vorsicht geboten. Solange der Ortszeiger P aktiv ist, befindet sich der Spieler noch am Eingang des Tunnels, und er kann mit GO in bestimmte Richtungen gehen. Im Tunnel muß der GO-Befehl jedoch natürlich unterbunden werden.

Nach Aufruf der Befehls-Unterroutine wird durch einen GO-Befehl das Bewegungsflag MF gesetzt und der Wert von P geändert. Dies kann umgangen werden, indem P auf den Wert gesetzt wird, den es vor Aufruf der Befehls-Unterroutine hatte.

Nach Handhabung der normalen Befehle können die speziellen, für diese bestimmte Situation notwendigen Befehle behandelt werden. Der RETREAT-Befehl läßt sich einsetzen, um dem Spieler die Rückkehr zum Tunneleingang zu ermöglichen. Der einzige andere Befehl, der erlaubt wird, ist LIGHT (oder als Alternative USE). Bei Eingabe eines anderen Befehls wird die Meldung „I DON'T UNDERSTAND" ausgegeben, bevor eine neue Anweisung eingegeben werden kann.

Handelt es sich bei dem eingegebenen Befehl um LIGHT oder USE, müssen vor der Ausführung unbedingt die folgenden Tests ausgeführt werden:

1. Ist das angegebene Objekt gültig?
2. Trägt der Spieler das angegebene Objekt bei sich?
3. Ist das angegebene Objekt die Lampe?

Lauten die Antworten auf alle Fragen „Ja", so wird dem Spieler gestattet, durch den Tunnel zu gehen. Diese Objekt-Überprüfungen kommen Ihnen sicher bekannt vor. Sie sind in der Tat fast identisch mit denen, die in den TAKE- und DROP-Routinen verwendet werden. Daher können wir bereits geschriebene Unterroutinen zur Ausführung der Tests verwenden.

```
4700 REM ** ENTER TUNNEL **
4705 SN$="YOU ENTER THE TUNNEL BUT IT IS TOO DARK
TO"
4710 SN$=SN$+" FIND YOUR WAY.":GOSUB5500
4725 PRINT:INPUT"INSTRUCTIONS";IS$
4730 GOSUB2500:REM SPLIT INSTRUCTION
4732 :
4735 IF F=0 THEN 4725:REM INVALID INSTRUCTION
4740 OP=P:GOSUB3000:REM NORMAL INSTRUCTIONS
4745 IF MF=1THEN SN$="IT IS SO DARK THAT YOU CAN O
NLY SEE":P=OP
4747 IF MF=1THENSN$=SN$+" THE TUNNEL ENTRANCE":GOS
UB5500:MF=0:GOTO4725
4750 IF VF=1 THEN 4725:REM INSTRUCTION OBEYED
4755 IF VB$="RETREAT" AND P=4 THEN MF=1:P=6:RETURN
4760 IF VB$="RETREAT" AND P=1 THEN MF=1:P=9:RETURN
4762 IFVB$<>"USE"ANDVB$<>"LIGHT"THEN SN$="I DON'T
UNDERSTAND"
4765 IFVB$<>"USE"ANDVB$<>"LIGHT"THEN GOSUB5500:GOT
O4725
4777 :
4780 REM ** SEARCH FOR LAMP **
4790 GOSUB5300:REM VALID OBJECT ?
4795 OV=F:GOSUB5450:REM IS OBJECT HELD ?
4797 IF F=0 THEN SN$="THERE IS NO "+W$:GOSUB5500:G
OTO4725
4800 IF HF=0 THEN SN$="YOU DO NOT HAVE THE "+IV$(F
,1):GOSUB5500:GOTO4725
4810 REM ** IS OBJECT LAMP ? **
4815 IF F<>2 THEN SN$="THE "+IV$(F,1)+" IS NO USE"
:GOSUB5500:GOTO4725
4835 REM ** SUCCESS **
4840 SN$="YOU USE THE LAMP TO LIGHT YOUR WAY THROU
GH THE TUNNEL"
4845 SN$=SN$+" AND EVENTUALLY EMERGE FROM THE EXIT
 ":GOSUB5500
4850 IF P=1 THEN MF=1:P=4:RETURN
4855 IF P=4 THEN MF=1:P=1:RETURN
```

Zusätzlich zu speziellen Orten, wie zum Beispiel dem Tunneleingang, können zufällige Gefahren oder Ereignisse programmiert werden. Bis jetzt werden während der Entwicklung von „Haunted Forest" keine Geister eingeplant. Um die Spannung zu erhöhen, sollen sie zufällig erscheinen, während der Spieler durch den Wald geht. Für die Abwehr sind spezielle Handlungen notwendig. Bevor wir uns im Detail mit diesen Routinen befassen, soll untersucht werden, wie man die Zufallsroutinen in die Struktur des Hauptprogramms integrieren kann. Die Hauptschleife ruft eine Unterroutine bei Zeile 2700 auf, um zu testen, ob ein neuer Ort „speziell" ist oder nicht. Dies ist auch der beste Platz, um die folgenden Zeilen zum Aufruf der Geisterroutine einzubauen:

```
2707 REM ** RANDOM GHOST **
2710 IF P>4 AND RND (1)<0.1 THEN
     GOSUB 4290: RETURN
```

Zeile 2710 überprüft zuerst einmal, ob der aktuelle Ort nicht einer der speziellen Orte ist. Handelt es sich um einen normalen Ort, wird der RND-Befehl verwendet, um zu bestimmen, ob ein Geist auftaucht oder nicht. Dies geschieht mit einer Wahrscheinlichkeit von zehn Prozent. Da RND-Befehle keine wirklich zufälligen Werte produzieren, sondern nach einem bestimmten Muster vorgehen, verwenden wir beim Acorn B sowie beim Commodore 64 einen negativen Operanden und beim Spectrum den RANDOMISE-Befehl.

```
207 R=RND (-1)
```

Nach dem Aufruf der „Geister"-Routine betreten wir ein spezielles Szenario, in dem der Spieler mit dem Geist konfrontiert wird. Die Routine stellt eine Meldung dar, fragt nach einer Anweisung und teilt diese in Verb und Rest der Eingabe auf. Normale Befehle werden mit den Standard-Routinen gehandhabt, wobei jedoch auch hier der GO-Befehl ausgeschlossen wird – der Spieler erhält die Meldung, daß keine Bewegung möglich ist.

An diesem Punkt können neue Befehle verwendet werden. Wie auch bei den anderen Routinen zur Handhabung spezieller Orte ist die Qualität des fertigen Spiels davon abhängig, welcher Programmieraufwand in diese Routinen investiert wurde. Jeder sinnlose Befehl kann mit der Meldung „I DON'T UNDERSTAND" beantwortet werden. Mit zusätzlichem Aufwand lassen sich jedoch auch Befehle handhaben, die der Spieler vielleicht eingeben will, auch wenn sie in der Situation nichts nützen. Ein Beispiel hierfür finden Sie in der Geister-Routine.

Die Geister-Routine

Wird der Spieler mit einem Geist konfrontiert, so ist meist der erste Gedanke, den Geist zu töten (FIGHT oder KILL). Die Geister-Routine verarbeitet diese Befehle durch Aufruf einer speziellen Unterroutine. Hier wird eine Meldung ausgegeben, die besagt, daß diese Befehle dem Spieler nichts nützen, was doch interessanter aussieht als das übliche „I DON'T UNDERSTAND".

```
4290 REM **** RANDOM GHOST S/R ****
4295 SF=1:GC=0
4300 SN$="YOU FEEL A COLD SENSATION RUNNING THE LE
NGTH"
4305 SN$=SN$+" OF YOUR SPINE. SUDDENLY A WHITE APP
ARITION"
4310 SN$=SN$+" APPEARS FROM OUT OF THE TREES AND"
4315 SN$=SN$+" MOVES TOWARDS YOU":GOSUB5500:REM FO
RMAT
4320 :
4325 SN$="THE GHOST MOVES CLOSER":GOSUB5500
4330 GC=GC+1:IF GC>4 THEN GOSUB4455:REM
4335 PRINT:INPUT"INSTRUCTIONS";IS$
4340 GOSUB2500:REM SPLIT INSTRUCTION
4345 IF F=0 THEN 4325:REM NEXT INSTRUCTION
4350 OP=P:GOSUB3000:REM ANALYSE INSTRUCTION
4355 IF MF=1 AND VB$="GO"THEN GOSUB4400:GOTO 4325
4357 IF MF=1 AND VB$="LOOK" THEN GOSUB2000:GOSUB23
00:GOTO4325
4360 IF VF=1 THEN 4325:REM NEXT INSTRUCTION
4365 REM ** NEW INSTRUCTION WORDS **
4370 IF VB$="KILL" OR VB$="FIGHT" THEN GOSUB4425:G
OTO 4325
4375 :
4385 IF VB$="SING" THEN GOSUB4500:RETURN
4390 SN$="I DON'T UNDERSTAND":GOSUB5500:GOTO4325
4395 :
4400 REM ** ATTEMPT TO MOVE **
4405 SN$="YOU ARE TRANSFIXED WITH TERROR AND CANNO
T"
4410 SN$=SN$+" MOVE...YET":MF=0:GOSUB5500:P=OP
4415 RETURN
4420 :
4425 REM ** FIGHT OR KILL **
4430 SN$="THE GHOST IS A BEING OF THE
SUPERNATURAL"
4435 SN$=SN$+" AND LAUGHS AT YOUR FEEBLE ATTEMPTS"
4440 SN$=SN$+" TO INJURE HIM":GOSUB5500
4445 RETURN
4450 :
4455 REM ** DEATH **
4460 SN$="THE PAIN IN YOUR CHEST BECOMES UNBEARABL
E"
4465 SN$=SN$+" AND YOU SLUMP ONTO THE LEAFY FOREST
 FLOOR.":GOSUB5500
4470 SN$="YOUR SPIRIT RISES FROM YOUR INERT BODY"
4475 SN$=SN$+" AND YOU FLOAT AWAY INTO THE MIST TO
 JOIN"
4480 SN$=SN$+" THE OTHER TORMENTED SOULS OF THE"
4485 SN$=SN$+" HAUNTED FOREST.":GOSUB5500
4490 END
```

Letzte Rettung: Singen

Wird einer der normalen oder nutzlosen Befehle erkannt, berücksichtigt die Routine diese, bevor sie eine neue Anweisung fordert. Es befindet sich jedoch noch ein Haken in der Routine, da sie die Anzahl der vom Spieler eingegebenen Befehle während der Konfrontation mit dem Geist zählt. Werden mehr als vier Anweisungen eingegeben, versucht der Geist, den Spieler zu töten. Die einzige Chance des Spielers besteht darin, mit dem Befehl „SING" ein Lied zu singen. Nach Eingabe dieses Befehls erhält der Spieler drei Lieder zur Auswahl, von denen eines den Geist vertreibt. Wird das falsche Lied gewählt, geht die Seele des Spielers in die Heerschar derjenigen ein, die bereits im Haunted Forest ihr Leben lassen mußten:

```
4500 REM ** SING **
4505 SN$="YOU KNOW THREE SONGS. WHICH ONE WILL YOU
 CHOOSE ?":GOSUB5500
4510 SN$="1) THE THEME FROM 'GHOSTBUSTERS'":GOSUB5
500
4515 SN$="2) 'THERE'S A GHOST IN MY HOUSE'":GOSUB5
500
4520 SN$="3) 'WAY DOWN UPON THE SWANEE RIVER'":GOS
UB5500
4525 PRINT:INPUT"MAKE YOUR CHOICE";C$
4530 IF VAL(C$)>3 OR VAL(C$)<1 THEN PRINT:PRINT"IN
VALID":GOTO4525
4535 CR=INT(RND(1)*3)+1
4537 IF CR<>VAL(C$) THEN GOSUB4542:REM WRONG TUNE
4540 GOSUB4565:REM CORRECT
4542 REM **** WRONG TUNE S/R ****
4545 SN$="THE GHOST HAS A PARTICULAR HATRED OF"
4550 SN$=SN$+" THAT TUNE AND LUNGES AT YOU.":GOSUB
5500
4555 GOSUB 4455:REM DEATH
4560 :
4565 REM ** CORRECT TUNE **
4570 SN$="THE GHOST IS APPEASED BY YOUR RENDITION
OF THE TUNE"
4575 SN$=SN$+" AND VAPOURISES INTO THIN AIR":GOSUB
5500
4580 RETURN
```

Digitaya-Listing

```
2690 IF P=37 THEN2780:REM VECTOR TABLE
2700 IF P>7 THEN 2750:REM RANDOM BUG

2740 REM ** RANDOM BUG **
2750 RA=RND(TI)
2760 IF RA<0.05THEN GOSUB 5420:REM BUG
2770 RETURN
2780 REM ** VECTOR TABLE **
2790 SF=1
2800 SN$="YOU ARE MOVED AT HIGH SPEED TO A NEW LOC
ATION":GOSUB5880
2810 FORJ=1TO1000:NEXT:REM PAUSE
2820 P=INT(RND(TI)*40+7)
2830 MF=1:RETURN

4550 REM **** ALU ****
4560 SF=1
4570 RN=INT(RND(TI)*3+1)
4580 IF RN=1 THEN CD$="AND"
4590 IF RN=2 THEN CD$="OR"
4600 IF RN=3 THEN CD$="NOT"
4610 SN$="MOUNTED ON THE WALL THERE ARE THREE BUTT
ONS MARKED"
4620 SN$=SN$+" 'AND', 'OR' AND 'NOT'. ACCESS CAN B
E GAINED TO THE"
4630 SN$=SN$+" ACCUMULATOR BY PRESSING THE CORRECT
 BUTTON"
4640 GOSUB5880:REM FORMAT
4650 :
4660 REM ** INSTRUCTIONS **
4670 PRINT:INPUT"INSTRUCTIONS";IS$
4680 GOSUB1700:GOSUB1900:REM ANALYSE
4690 IF MF=1THEN RETURN:REM MOVE OUT
4700 IF VF=1THEN 4670:REM NEXT INSTRUCTION
4710 IFVB$="USE"OR VB$="PRESS"THEN4740
4720 PRINT"I DON'T UNDERSTAND":GOTO4670
4730 :
4740 REM ** VALID COMMAND **
4750 IF VB$="PRESS"THEN 4930
4760 REM ** COMMAND IS 'USE' **
4770 GOSUB5730:REM IS OBJECT VALID
4780 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO4670:REM
 NEXT INSTRUCTION
4790 :
4800 REM ** IS OBJECT CODE BOOK **
4810 IF F=7 THEN4850:REM OK
4820 SN$="YOUR "+IV$(F,1)+" IS OF NO USE":GOSUB588
0
4830 GOTO4670:REM NEXT INSTRUCTION
4840 :
4850 OV=7:GOSUB5830:REM IS CODE BOOK HELD
4860 IFHF=1THEN4900:REM OK HELD
4870 SN$="YOU DO NOT HAVE THE "+IV$(7,1)
4880 GOSUB5880:GOTO4670:REM NEXT INSTRUCTION
4890 :
4900 SN$="YOU OPEN THE CODE BOOK AND FIND THE WORD
 '"+CD$+"' WRITTEN INSIDE"
4910 GOSUB5880:GOTO4670:REM NEXT INSTRUCTION
4920 :
4930 REM ** COMMAND IS PRESS **
4940 IFNN$="AND"OR NN$="OR"OR NN$="NOT"THEN4970
4950 SN$="THERE IS NO "+NN$:GOSUB5880:GOTO4670:REM
 NEXT INSTRUCTION
4960 :
4970 REM ** RIGHT OR WRONG **
4980 IFNN$=CD$ THEN GOSUB5100:RETURN
4990 GOSUB5010:RETURN
5000 :
5010 REM ** WRONG S/R **
5020 SN$="WRONG, A TRAP DOOR OPENS AND YOU FIND YO
URSELF BACK"
5030 SN$=SN$+" BACK IN MAIN MEMORY"
5040 GOSUB5880:REM FORMAT
5050 IF RN=1 THEN P=39
5060 IF RN=2 THEN P=35
5070 IF RN=3 THEN P=29
5080 MF=1:RETURN
5090 :
5100 REM ** RIGHT S/R **
5110 SN$="THE GATEWAY TO THE ACCUMULATOR SWINGS OP
EN AND"
5120 SN$=SN$+" YOU PASS THROUGH":GOSUB5880
5130 P=30:MF=1:RETURN

5420 REM **** RANDOM BUG ****
5430 SF=1
5440 SN$="A LARGE AND UGLY BUG APPEARS FROM BEHIND
 A CHIP"
5450 SN$=SN$+" AND LUNGES TOWARDS YOU":GOSUB5880
5460 :
5470 REM ** INSTRUCTIONS **
5480 PRINT:INPUT"INSTRUCTIONS";IS$
5490 GOSUB1700:GOSUB1900:REM ANALYSE
5500 IFMF=1THENMF=0:PRINT"YOU CAN'T MOVE...YET":GO
TO5480
5510 IF VF=1THEN5480:REM NEXT INSTRUCTION
5520 IF VB$="KILL"ORVB$="FIGHT"THEN5550
5530 PRINT'I DON'T UNDERSTAND":GOTO5480
5540 :
5550 REM ** COMAND IS FIGHT/KILL **
5560 RA=RND(TI)
5570 IFRA<0.5 THEN GOSUB5600
5580 GOSUB5670:RETURN
5590 :
5600 REM **** KILLED BY S/R ****
5610 SN$="YOU FIGHT WITH THE BUG. IT SHOWERS YOU W
ITH SPURIOUS"
5620 SN$=SN$+" ERRORS AND THEY EAT INTO YOUR BRAIN
."
5630 SN$=SN$+" FINALLY YOU CAN TAKE NO MORE AND YO
UR HEAD EXPLODES."
5640 GOSUB5880
5650 END
5660 :
5670 REM **** YOU KILL S/R ****
5680 SN$="YOU FIGHT WITH THE BUG AND THOUGH IT IS
A HARD STRUGGLE"
5690 SN$=SN$+" YOU EVENTUALLY IRON IT OUT AND SURV
IVE.":GOSUB5880
5700 RETURN
```

BASIC-Dialekte

Spectrum

Ersetzen Sie bei beiden Programmen SN$ durch S$, IS$ durch T$, IV$(,) gegen V$(,), VB$ durch B$, CD$ durch C$ und NN$ durch R$.
Ersetzen Sie die folgenden Zeilen im Listing von Haunted Forest:

```
207 RAND
4815 IF F<>2 THEN LET S$=„THE“:LET
     A$=V$(F,1):GOSUB 7000
4816 IF F<>2 THEN LET S$=S$+“ IS
     NOT USE“:GOSUB 5500:GOTO 4725
```

Ersetzen Sie die folgenden Zeilen im Digitaya-Listing:

```
2750 LET RN=RND(1)
2820 LET P=INT(RND(1)*40+7)
4570 LET RN=INT(RND(1)*3+1)
4820 LET SN$=“YOUR “:LET A$=V$(F,1):
     GOSUB 8500
4825 LET SN$=SN$+“ IS OF NO USE“:
     GOSUB 5880
5560 LET RA=RND(1)
```

Acorn B

Ersetzen Sie die folgenden Zeilen im Listing von Haunted Forest:

```
207 RND(-TIME)
4535 CR=RND(3)
```

Ersetzen Sie die folgenden Zeilen im Listing von Digitaya:

```
2750 RN=RND(1)
2820 P=RND(40)+7
4570 RN=RND(3)
5560 RA=RND(1)
```

Adressenänderung

Wir untersuchen, wie die beiden Indexregister X und Y für die indizierte Adressierung eingesetzt werden, und zeigen an einigen Beispielen die Vorteile dieser Methode.

Das ursprüngliche Konzept des programmierbaren Computers basierte darauf, daß das Programm in dem gleichen Bereich (und im gleichen Format) wie seine Daten untergebracht sein sollte, damit es sich während des Ablaufs selbst modifizieren kann. Diese Möglichkeit war jedoch nicht für eine Umstellung der Befehle gedacht, sondern es sollten hauptsächlich Adressen geändert werden, von denen die Programme Daten abriefen. Stellen Sie sich nun vor, Sie sollten eine Tabelle mit mehreren tausend Zahlen bearbeiten, hätten dafür aber nur Befehle zur Verfügung, die, sehr zeitaufwendig, jeweils eine einzige Adresse ansprechen können.

Durch veränderbare Adressen ließe sich das Problem leicht lösen. Dabei würden in einer Schleife Befehle wiederholt werden, die sich bei jedem Durchlauf auf eine andere Adresse beziehen. Die Adreßänderung wird durch die Veränderung eines Registerwertes ausgelöst, der die Adresse beeinflußt. Dieses Konzept erscheint oft in BASIC-Programmen:

```
FOR I = 1 TO N
PRINT TABELLE(I)
NEXT I
```

In diesem Modul bezieht sich der PRINT-Befehl auf immer andere Daten, da sich das grundlegende Datenobjekt (TABELLE) über den ansteigenden Indexwert (I) bei jedem Durchgang verändert.

Das Grundprinzip der „Indizierten Adressierung" besteht darin, den Inhalt des Indexregisters auf eine Basisadresse zu addieren, um die aktuelle Adresse zu erhalten – das heißt die Speicherstelle, die angesprochen werden soll. Tritt dieser Befehl im Inneren einer Schleife auf, läßt sich dort auch das Indexregister inkrementieren oder dekrementieren, so daß die Tabellenwerte nacheinander abgerufen werden.

Der 6809 hat für diese Aufgabe nicht nur die Register X und Y, sondern auch S und U zur Verfügung. Unter besonderen Umständen läßt sich auch der Befehlszähler einsetzen. Weiterhin gibt es eine ganze Reihe von Indizierungsmöglichkeiten, die fast alle Programmanforderungen erfüllen. Da wir von nun an die verschiedenen Formate der Indizierung einsetzen werden, haben Sie ausreichend Gelegenheit, sich damit vertraut zu machen.

Die indizierte Adressierung wird durch den Zusatz ,X im Operandenfeld angezeigt (natürlich nur, wenn X das eingesetzte Indexregister ist). Indizierte Befehle haben folgendes Standardformat:

```
Opcode    Offset, Indexregister
  LDA     TABELLE1,X
  STA     TABELLE2,Y
```

Ein Offset von Null kann weggelassen werden:

```
Opcode    ,Indexregister
  LDA     ,X
  STA     ,Y
```

Sehen wir uns diesen Vorgang einmal in der Praxis an. Angenommen, von $3000 an ist eine Tabelle mit 64 Acht-Bit-Werten gespeichert, die wir nacheinander adressieren wollen. Dazu definieren wir die Basisadresse und reservieren mit Assembleranweisungen den Platz für die Tabelle:

```
          ORG   $3000
TABELLE   RMB   64
```

Diese Anweisungen setzen den Befehlszähler auf $3000, definieren den Anfang von TABELLE als $3000 und reservieren die nächsten 64 Bytes. Mit dem nächsten Programmteil adressieren wir die Bytes: Die neue ORG-Anweisung zeigt an, daß der Code in einem anderen Teil des Speichers abgelegt ist als die Daten. Mit dieser Vorsichtsmaßnahme verhindern wir, daß die Daten unbeabsichtigt den Programmcode überschreiben.

```
          ORG   $1000
COUNT     FCB   0
          LDX   #0
LOOP      LDA   TABELLE,X
```

Wir verändern nun den Wert des X-Registers:

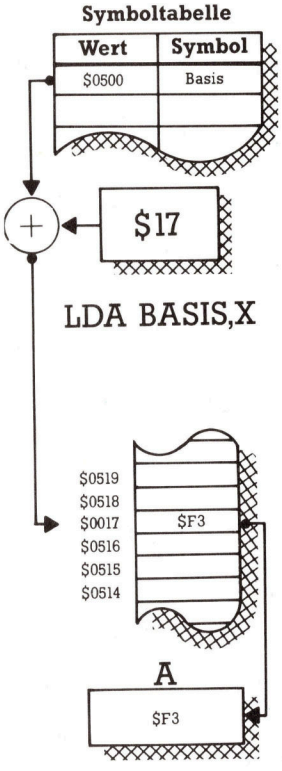
```
          TFR   X,D
          ADDD  #1
          TFR   D,X
```

Diese etwas umständliche Methode, X zu inkrementieren, läßt sich gut für In- und Dekremente einsetzen, die über Zwei liegen. Weitere Möglichkeiten sehen wir uns später an. Das letzte Teil des Codes inkrementiert den Zähler (COUNT) und stellt sicher, daß er nicht auf 64 steht (in diesem Fall ist das Programm beendet – die Schleife muß nicht nochmals durchlaufen werden):

```
          INC   COUNT
          LDB   COUNT
          CMPB  #64
          BLT   LOOP
```

Der Code läßt sich natürlich noch verbessern. Am praktischsten ist die Methode der „automatischen Inkrementierung". Der Befehl:

Das Symbol BASIS – mit $0500 initialisiert – ist die Adresse des ersten Bytes einer Wertetabelle. Der Befehl LDA BASIS,X setzt die Methode der indizierten Adressierung ein. Dabei wird der Inhalt des X-Registers auf den Wert von BASIS addiert und so die aktuelle Adresse errechnet, deren Inhalt in den Akkumulator geladen wird. Wenn sich dieser Befehl innerhalb einer Schleife mit dem Zähler X befindet, läßt sich die gesamte Tabelle Byte für Byte ansprechen. Da X ein 16-Bit-Register ist, kann die Schleife (in einem Acht-Bit-System wie dem 6809) den gesamten Speicherbereich ($0000 bis $FFFF) adressieren.



```
        LDA   TABELLE,X+
```

veranlaßt, daß der Wert von X automatisch nach jedem Gebrauch inkrementiert wird. Für eine Tabelle von 16-Bit-Werten wird X mit

```
        LDA   TABELLE,X++
```

um jeweils zwei inkrementiert. Unser ursprüngliches Programm nimmt nun Formen an:

```
LOOP    LDA   TABELLE,X+
        INC   COUNT
        LDB   COUNT
        CMPB  #64
        BLT   LOOP
```

Die Tabelle läßt sich mit der automatischen Dekrementierung auch von hinten nach vorn ansprechen. Diese Methode hat den Vorteil, daß der letzte Wert des X-Registers Null ist. Da die automatische Dekrementierung eines Indexregisters auch die Flags des Condition Code Registers setzt, können wir das Ende der Schleife direkt abfragen, ohne den Befehl CMP einsetzen zu müssen. Die gleiche Wirkung erzielt auch das Laden des Indexregisters mit einem negativen Wert, der so lange inkrementiert wird, bis Null erreicht ist. Bei jeder automatischen Inkrementierung werden die Flags des Condition Code Registers gesetzt, um das jeweilige Resultat anzuzeigen. Das Ergebnis Null setzt beispielsweise das Null-Flag, ein Übertrag das Übertrags-Flag und so weiter.

Vorsicht Bearbeitungsmodule!

Tests sollten möglichst nur über den Akkumulator durchgeführt werden, da in den meisten Programmen zwischen dem In- oder Dekrementierbefehl und dem Testbefehl Bearbeitungsmodule liegen, die das Condition Code Register verändern.

Selbst wenn Sie die Tabelle nicht von hinten nach vorn abfragen, sollte der Zähler rückwärts laufen, damit die Schleife bei Null endet. Bei der automatischen Dekrementierung ist wichtig, daß die Dekrementierung vor der Adressenberechnung ausgeführt wird, während die automatische Inkrementierung das Register nach der Berechnung der Adresse erhöht. Wenn X dabei 7 enthält und die TABELLE bei $1000 beginnt, dann lädt der Befehl LDA TABELLE,X+ erst den Inhalt der Adresse $1007 in den Akkumulator und inkrementiert danach X von 7 auf 8. LDA TABELLE,−X (beachten Sie, daß das Minuszeichen hier vor dem Registernamen steht) dekrementiert zunächst X von 7 auf 6 und lädt dann den Akkumulator mit dem Wert der Adresse $1006.

Wenn wir rückwärts durch die Tabelle gehen und der Zähler im Register B liegen soll, muß die Schleife folgendermaßen aussehen:

```
        LDX   #64
        LDB   #64
LOOP    LDA   TABELLE,-X
        DECB
        BGE   LOOP
```

In unserem ersten Beispielprogramm zählt eine einfache Schleife die Anzahl der negativen Werte einer Tabelle von Acht-Bit-Werten. Der Zähler im Akkumulator B dient dabei auch als Offset des festen Wertes im X-Register. Das zweite Programm zeigt, wie beide Indexregister zusammen und mit einem Null-Offset eingesetzt werden. Das Programm kopiert einen String von einer Speicherstelle (möglicherweise einem Eingabebuffer) in einen anderen Speicherbereich. Der String schließt mit einem Return-Zeichen ab. Seine Länge ist zwar nicht bekannt, kann jedoch maximal 255 Zeichen betragen. Nach der Speicherung wird das Return-Zeichen gelöscht und ein Byte, das die Länge des Strings anzeigt, an seinen Anfang gesetzt.

Im ersten Programm ist eine Tabelle von Acht-Bit-Werten von der Speicherstelle $3001 an untergebracht. Die Anzahl der Werte ist in $3000 gespeichert. Sie liegt unter 256. Das Programm zählt die Anzahl der negativen Werte in der Tabelle. Das zweite Programm kopiert einen String vom Eingabebuffer bei $3000 in den nächsten freien Stringbereich, dessen Adresse in $0010 liegt.

Zähler für negative Werte

```
TABELLE EQU  $3000          TABELLE enthält die Anzahl der Werte
        ORG  $1000
NEGS    FCS  0              NEGS soll die Anzahl der negativen Werte
        LDB  TABELLE        enthalten
        LDX  #TABELLE       Die Adresse der Tabelle wird in X
                            gespeichert. Den letzten Wert der Tabelle
LOOP    LDA  B,X            holen
        BGE  ENDLP          GOTO ENDLP, wenn der Wert positiv ist
        INC  NEGS           ELSE erhöhe den Zähler, wenn der Wert
                            negativ ist
ENDLP   DECB                Tabelle durchsuchen
        BGT  LOOP           Gibt es weitere Werte? Falls ja, GOTO LOOP
        SWI                 ELSE zurück zum Betriebssystem
        END
```

Kopiermodul für Strings

```
CR      EQU  13             ASCII-Wert des Zeichens RETURN
STRING1 EQU  $3000          Adresse des Eingabebuffers
STRING2 EQU  $0010          STRING2 enthält die Adresse des freien
        ORG  $1000          Speicherbereichs
        LDX  #STRING1       Die Adresse des Originalstrings wird in X
                            geladen
        LDY  STRING2        Die Adresse des Ziel-Strings (die in $10,$11
                            gespeicherte Adresse) wird in Y geladen
        TFR  Y,U            Die Adresse des Längen-Bytes in U
                            übertragen
        LDA  #0             Null im ersten Byte des Ziel-Strings
        STA  ,Y+            speichern
LOOP    LDA  ,X+            Das nächste Zeichen aus dem Eingabebuffer
                            holen
        CMPA #CR            Ist es das RETURN-Zeichen?
        BEQ  FINISH         Wenn ja, dann Ende
        STA  ,Y+            ELSE kopieren (und) Eins zur Länge
        INC  ,U             addieren
        BRA  LOOP           Nächstes Zeichen
FINISH  SWI                 Zum Betriebssystem zurückkehren
END
```

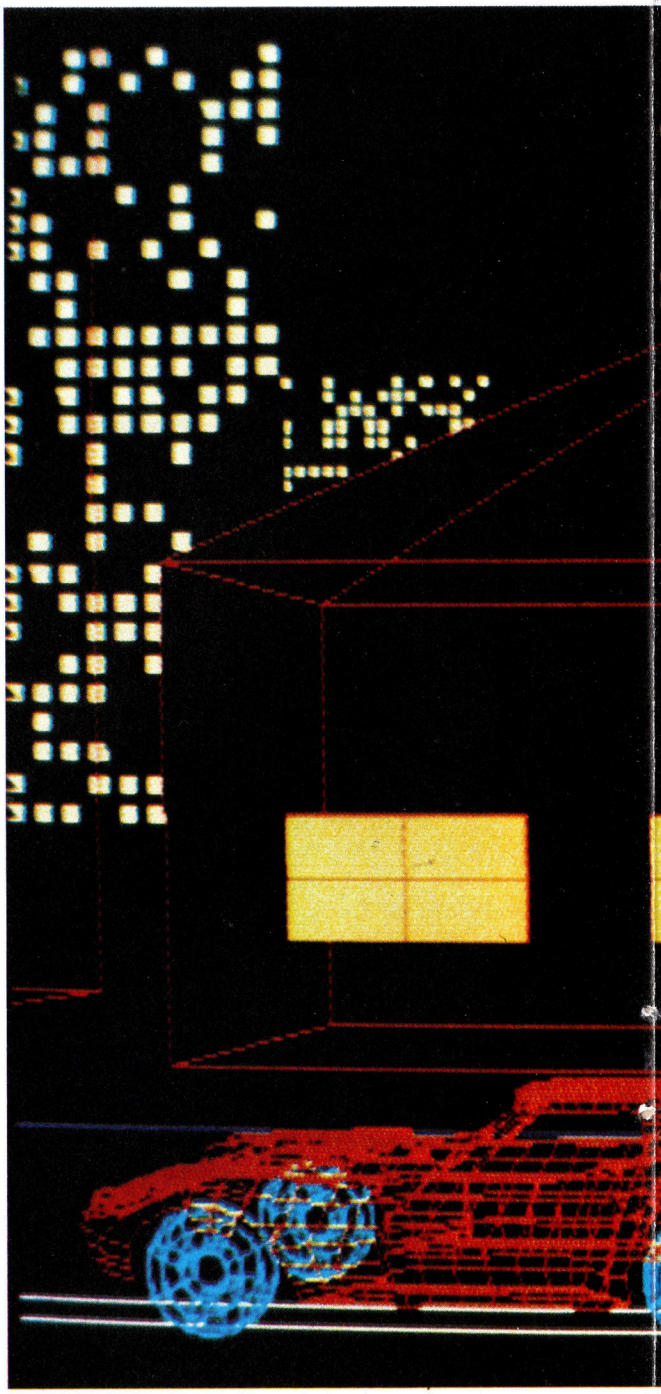
Realismus ist „in"

Hohe Auflösung läßt die Bilder wie Fotografien aussehen, und Drahtmodelle können in Filmsequenzen räumliche Bewegungen „erlebbar" machen.

Zu den Pionieren der Computer-Animation gehören Daniel Thalmann von der Universität Montreal und seine Frau Nadia Magnenat-Thalmann. Ihr 1982 der Öffentlichkeit vorgestellter Computer-Film „Dream Flight" schildert die Reise eines Wesens von einem anderen Stern zur Erde und dessen Erlebnisse auf unserem Planeten. Besonderer Wert wurde auf eine filmische Sichtweise gelegt, das heißt, die imaginäre Kamera gleitet in weichen Schwenks, Drehungen und rasanten Fahrten über, um und sogar durch die Szenerie, etwa in das Innere der Freiheitsstatue. Unterstützt wird diese computerhafte Darstellungsweise in „Dream Flight" durch die ausschließliche Verwendung von „wire-frames", den sogenannten „Drahtmodellen".

Der Artron 2000, auf dem der Tigerkopf (unten) entwickelt wurde, zählt zu den sogenannten Paintsystemen. Darunter versteht man Computer, deren größtes gestalterisches Potential auf dem grafisch-malerischen und nicht auf dem konstruktiven Bereich liegt. In den Artron lassen sich auch per Videokamera beliebige zwei- oder dreidimensionale Vorlagen eingeben und bearbeiten.

Große Ansprüche im grafisch-malerischen Bereich, verbunden mit einer hohen Auflösung, befriedigt auch das System von Genigraphics. Innerhalb der verschiedenen Ausbaustufen ist der Genigraphics 100 V auch animationsfähig. Für Druckvorlagen eignet sich besonders der Genigraphics SG 1, wie an der hier abgebildeten Uhr (rechts) sichtbar wird. Auch bei starker Vergrößerung sind keine störenden „Treppen" zu erkennen. Es entsteht ein realistischer Bildeindruck.



Auch bei starker Vergrößerung sind in den Grafiken des Genigraphics SG 1 keine störenden „Treppen" zu erkennen (siehe rechts oben, die Uhr). Es entsteht ein sehr realistisches Bild.

Effekte wie von Aquarellfarben lassen sich mit dem Artron 2000 realisieren, bei dem die elektronischen Farben regelrecht ineinanderlaufen können (unten).

computer kurs

Computer Welt

Hardware

Tips für die Praxis

PASCAL

PROLOG

Computer-Logik

BASIC

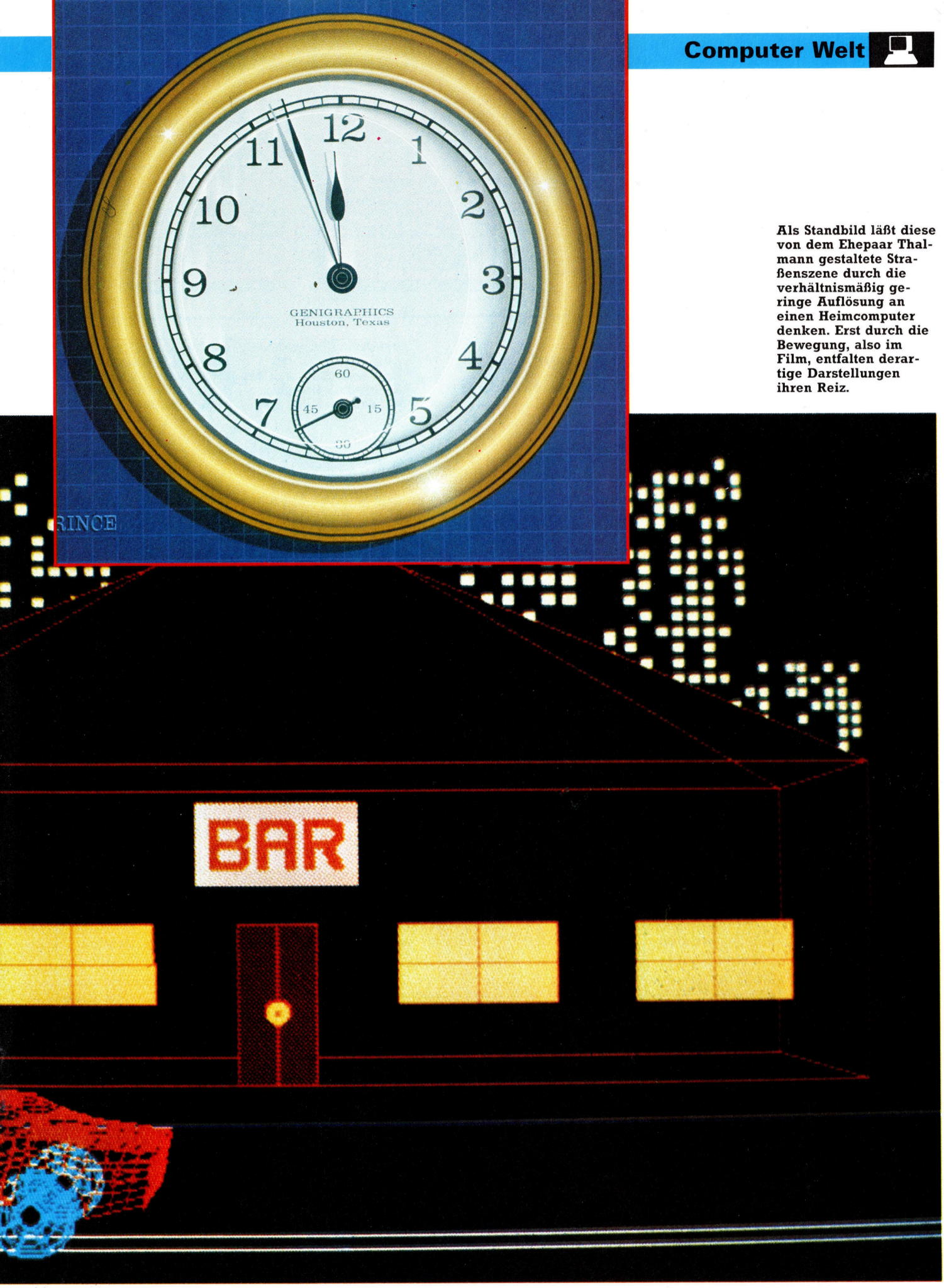
Bits & Bytes

Software

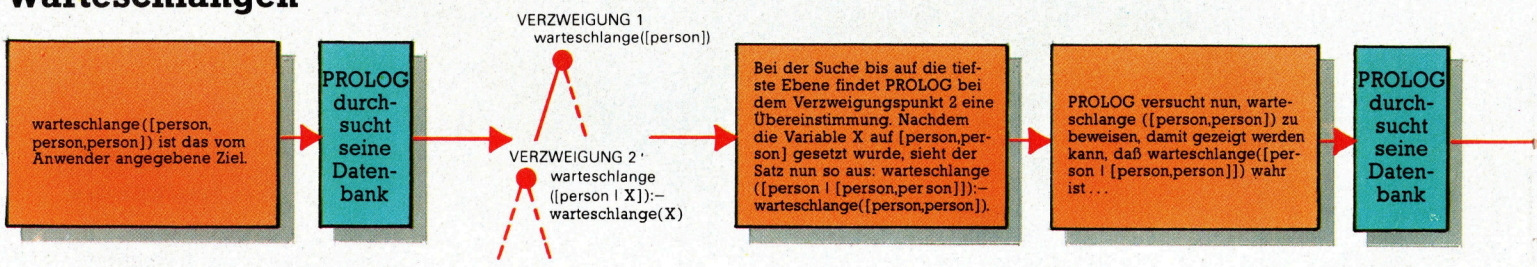
Peripherie

Alle zwölf Hefte erscheint ein solcher Teilindex. Der Gesamtindex erscheint mit dem letzten Heft — darin einbezogen sind Kreuzverweise auf die Artikel, die mit dem gesuchten Stichwort in Verbindung stehen.

Stichwörter, die auf die vorletzte Heftseite hinweisen, sind als VS und mit der betreffenden Heftnummer gekennzeichnet.



Als Standbild läßt diese von dem Ehepaar Thalmann gestaltete Straßenszene durch die verhältnismäßig geringe Auflösung an einen Heimcomputer denken. Erst durch die Bewegung, also im Film, entfalten derartige Darstellungen ihren Reiz.



Gesucht und gefunden

Das Ablaufdiagramm zeigt, wie PROLOG eine einfache Frage des Anwenders beantwortet. Beachten Sie, daß der Variablen X während der Programmausführung zwei unterschiedliche Werte zugeordnet werden.

In PROLOG erfolgt die Programmsteuerung nicht mit einer seriellen Reihenfolge, sondern mit einer Suchprozedur. In diesem Artikel untersuchen wir die Schritte, die zwischen der Definition eines Programmziels und seiner Ausführung liegen.

Während Sprachen wie BASIC oder PASCAL sequentiell gesteuert werden und streng von oben nach unten ablaufen (außer bei Schleifen oder GOTO-Befehlen), durchsucht PROLOG zunächst die Programmsätze bis in die tiefste Ebene.

Stellen Sie sich das Programm als Baum vor, an dessen Wurzeln das Ziel (die Behauptung) bewiesen werden soll. Die Verzweigungspunkte sind Teilziele, die Auskunft über die Wege zum Hauptziel geben. Ein Baum dieser Art läßt sich auf viele Arten durchsuchen. PROLOG fängt beim äußersten Zweig der linken Seite an und folgt ihm in die größtmögliche Tiefe. An jeder Verzweigung wird der Weg markiert, so daß PROLOG beim Erreichen der untersten Ebene (wenn es nicht weitergeht) auf den letzten Verzweigungspunkt zurückgehen kann. Von hier aus wird nun wieder der am weitesten links liegende (und noch nicht erforschte) Zweig gewählt etc.

Eine umfassende Suche berücksichtigt nun zwar alle Lösungswege, kann aber lange dauern. PROLOG erspart sich dabei jedoch alle überflüssige Arbeit. In der letzten Folge wurde gezeigt, daß PROLOG-Sätze Regeln darstellen, deren Hauptaussagen wahr sind, wenn alle Unteraussagen wahr sind:

```
aussage:—aussage1,aussage2,
aussage3. . .etc.
```

Dieser Satz ist vermutlich besser zu verstehen, wenn wir schreiben:

```
IF aussage1 wahr ist
   AND aussage2 wahr ist
   AND aussage3 wahr ist
   AND etc . . .
THEN ist aussage wahr.
```

Da die Unteraussagen mit AND verbunden sind, ist der gesamte Satz falsch, wenn sich eine der Aussagen nicht beweisen läßt. Wenn dieser Fall eintritt, geht PROLOG den Möglichkeiten dieses Weges nicht weiter nach.

Durch die Methode des Backtrakkings hat die Anordnung des Programmcodes kaum noch Bedeutung. Obwohl PROLOG großen Wert auf die „erklärende" Darstellung eines Problems legt, läßt sich in den Programmen aber auch die Struktur des Ablaufs deutlich erkennen. Der PROLOG-Satz:

```
marsmensch(X):—gliederzahl(X.7),
   kopfzahl(X.2),kann_programmieren
      in(X,cobol).
```

bedeutet: „X ist ein Marsmensch, wenn X sieben Glieder und zwei Köpfe hat und in COBOL programmieren kann". Das Programm würde folgendermaßen ablaufen: „Um beweisen zu können, daß X ein Marsmensch ist, beweise zunächst, daß er sieben Glieder hat, dann, daß er zwei Köpfe besitzt, und dann, daß er in COBOL programmieren kann".

Die Elemente von PROLOG sind weit weniger an Typendefinitionen gebunden als die der meisten anderen Sprachen. Die Variable Argument in dem Ausdruck pred(Argument) läßt sich einmal als Integer definieren, dann als Name (wie martin, venusianer, d24 etc.) und auch als Liste.

Jeder, der mit LOGO oder LISP vertraut ist, kennt den Listen-Datentyp und die besonderen Methoden der Listenverarbeitung. PROLOG-Listen werden von eckigen Klammern umschlossen und die Elemente durch Kommas getrennt. Beispiele für Listen sind [apfel,pfirsich,banane] und [a,f,e,g,r,x].

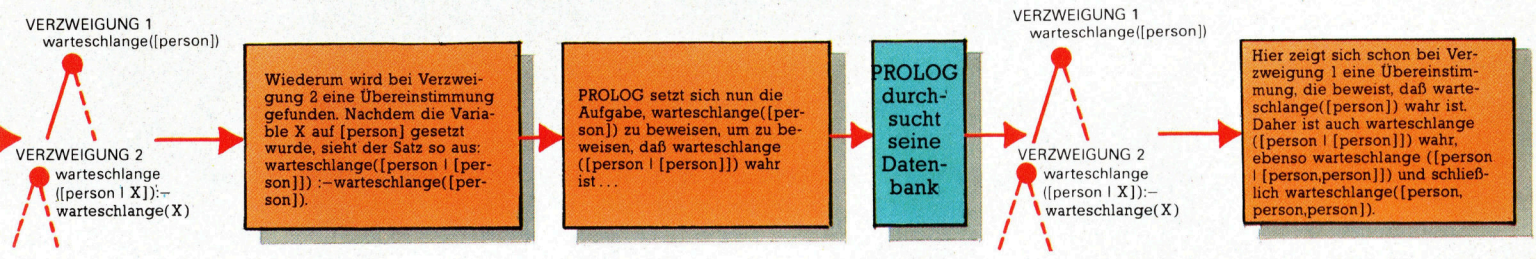
Listen zerteilen

Für die Bearbeitung werden die Listen Stück für Stück „auseinandergenommen", wobei immer das erste Element der Liste entfernt wird. [Kopf | Schwanz] beschreibt eine Liste mit dem Element Kopf und der Liste Schwanz. Jedes einzelne Listenelement kann ebenfalls eine Liste sein. Die Trennung „|" sieht in einer Liste von Früchten folgendermaßen aus: [apfel | [pfirsich, banane]]. Die Liste [z80] ist ein Spezialfall. Sie hat nur ein Element, das mit | in [z80 | []] unterteilt werden kann. Die Liste [] im zweiten Teil stellt eine leere Liste dar.

In PROLOG sind recursive Definitionen möglich. Zum Beispiel:

```
warteschlange([person]).
warteschlange([person | X]):—warte-
   schlange(X).
```

Zwei oder mehr Sätze, die (wie im vorstehenden Beispiel) den gleichen Kopf besitzen, werden Prozedur genannt. Die Prozedur warteschlange enthält einen Satz, der warteschlange als Liste mit einem Element (person)



definiert. Der zweite Satz informiert uns, daß warteschlange auch eine Liste mit dem Element person als Kopf und eine Liste mit dem Namen X als Schwanz haben kann. Aus der rechten Seite dieses Satzes läßt sich erkennen, daß X selbst eine Warteschlange sein muß.

Wenn wir PROLOG die Aufgabe stellen:

```
warteschlange([person,person,
    person]).
```

um herauszufinden, ob [person,person,person] eine Warteschlange ist, sucht das System in seinen Daten zunächst nach einem Satz, der dieser Aussage entspricht. Als erstes findet PROLOG warteschlange([person]). Da diese Listen nicht übereinstimmen, untersucht PROLOG den nächsten Satz warteschlange([person | X]):–warteschlange(X). Hier gibt es ebenfalls keine Übereinstimmung. PROLOG füllt nun die Variablen mit folgenden Werten:

```
warteschlange([person | [person,
    person]]):–warteschlange
        ([person,person]).
```

Um zeigen zu können, daß die Aussage des Kopfes wahr ist, muß das System nun beweisen, daß die Unteraussagen wahr sind. PROLOG nimmt daher warteschlange([person,person]) als Ziel und fängt an, die Sätze von Anfang an zu durchsuchen, um eine Übereinstimmung zu finden. Wieder gibt es bei warteschlange ([person]) keine Entsprechung, doch jetzt zeigt der zweite Satz

```
warteschlange([person |
    [person]]):–warteschlange
        ([person]).
```

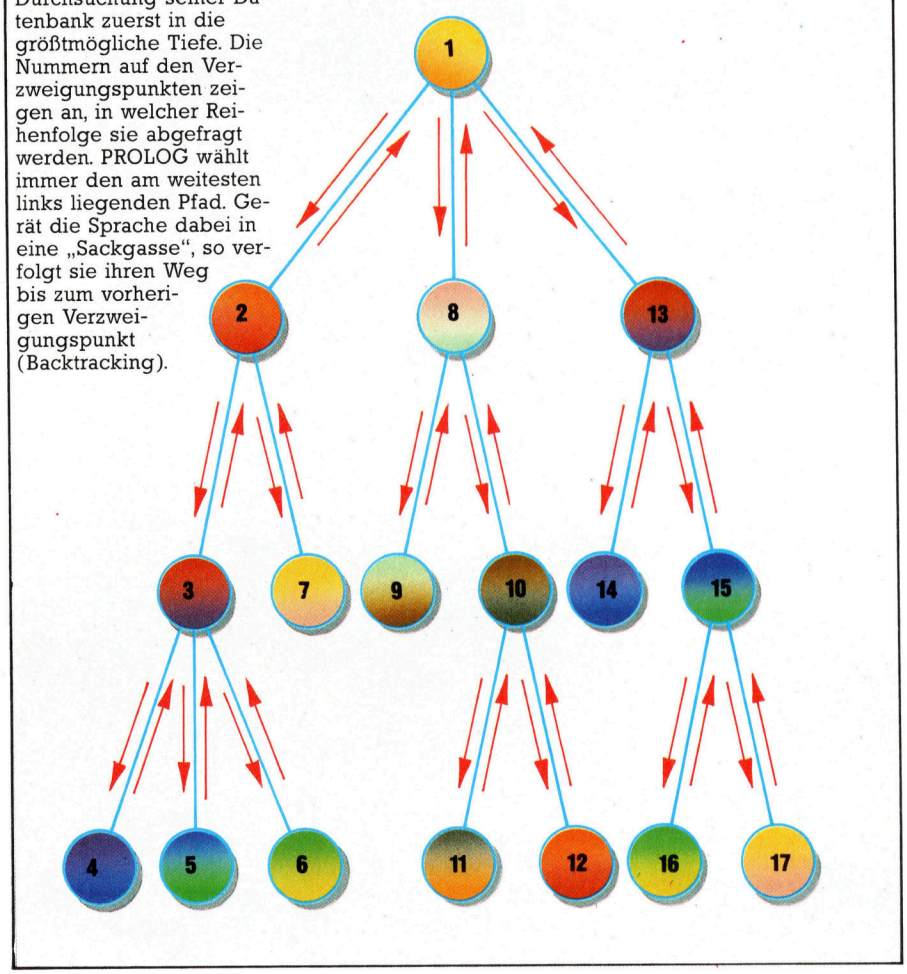
eine Übereinstimmung. Wichtig ist hierbei, daß die Variable X, die beim ersten Durchlauf auf [person,person] stand, auf [person] gesetzt wurde, obwohl der erste Wert von X erhalten geblieben ist.

Dies ist möglich, da Variablen beim Aufruf eines Satzes nur lokal eingesetzt werden. Jeder Aufruf von warteschlange([person | X]) scheint daher eine neue und einzigartige Variable zu verwenden.

Nachdem der passende Satz für den Kopf dieser Aussage gefunden ist, muß er bewiesen werden. Dies geschieht durch den Beweis der Unteraussage auf der rechten Seite des ":–"-Symbols, nämlich warteschlange([person]). Dafür werden die Sätze nochmals durchsucht, bis die direkte Entsprechung warteschlange ([person]) gefunden ist. Die Aussage warteschlange([person]) kann nun bewiesen werden. Damit ist auch die ursprüngliche Aussage warteschlange ([person,person,person]) wahr.

Die Prozedur warteschlange verdeutlicht eine ganze Reihe wichtiger Dinge. So kann beispielsweise die Anordnung der Sätze eine große Bedeutung haben. Durch den Einsatz zusätzlicher Sätze entstehen, wie bei einem logischen OR, alternative Möglichkeiten, ein Ziel zu beweisen. Obwohl die meisten PROLOG-Systeme einen Operatoren für OR enthalten, wird er eigentlich nicht gebraucht.

Immer linksherum
PROLOG geht bei der Durchsuchung seiner Datenbank zuerst in die größtmögliche Tiefe. Die Nummern auf den Verzweigungspunkten zeigen an, in welcher Reihenfolge sie abgefragt werden. PROLOG wählt immer den am weitesten links liegenden Pfad. Gerät die Sprache dabei in eine „Sackgasse", so verfolgt sie ihren Weg bis zum vorherigen Verzweigungspunkt (Backtracking).



Sieg oder Platz?

Königlicher Sport oder nur ein Tummelplatz für Gauner – über Pferderennen kann man unterschiedlicher Meinung sein. Ein Spiel-Klassiker der englischen Salamander Software läßt auch Nichtsportler am „Glück dieser Erde" teilhaben.

In jedem Stadium des Derbys können die Spieler das Rennprogramm anschauen (oberes Bild). Darin sind Details wie die Aufstellung und die Gewinnquoten der einzelnen Rennen verzeichnet. Wenn die Pferde für einen Wettbewerb feststehen, kann gewettet werden (zweites Bild): Zwischen fünf und 500 Pfund muß der Spieler jeweils setzen. Danach werden die Pferde an den Start gebracht, und das eigentliche Rennen beginnt. Das unterste Bild zeigt die Pferde beim Abstoppen hinter der Ziellinie – man sieht, es ging nur um wenige Längen . . .

Im Computerspiel „Classic Racing" muß man als Trainer für Rennpferde eine ganze Saison durchstehen. Man kann sowohl allein als auch mit bis zu fünf weiteren Teilnehmern spielen. Sind nicht genug Mitstreiter vorhanden, übernimmt der Computer ihren Platz, so daß in jedem Rennen sechs Pferde laufen. Die Dauer einer Saison kann gewählt werden – das Maximum sind sechzehn Derbytage mit jeweils sechs Rennen. Das Ziel ist natürlich, einen möglichst hohen Gewinn einzustreichen – entweder durch die Preisgelder für die drei schnellsten Pferde oder durch Wetten. Zwar muß auch das eigene Pferd im Rennen antreten, der Spieler kann aber auch auf den Sieg seiner Mitbewerber Wetten abschließen.

Im „Stall" stehen 16 Pferde, deren Leistung man zu Beginn der Saison noch nicht kennt. In den ersten Rennen sind die Preisgelder noch niedrig, ein günstiger Zeitpunkt, um das Können der eigenen Vierbeiner bei unterschiedlichen Distanzen und auf verschiedenen Böden genau kennenzulernen. Das beste Verfahren ist Ausprobieren. Wie verhalten sich die Pferde bei bestimmten Außenbedingungen? Ein Urteil kann nur nach exakten Beobachtungen und mit gründlichen Aufzeichnungen der jeweiligen Distanz, der Bodenbeschaffenheit, des Jockeygewichtes und natürlich der Ergebnisse gefällt werden. Leider bietet das Programm keine Möglichkeit für den direkten Ausdruck dieser Daten – der Notizblock darf also nicht fehlen!

Hat man seine sechs Pferde für den ersten Renntag gewählt, werden die Namen der Konkurrenten und das Gewicht ihrer jeweiligen Jockeys mitgeteilt. Der Rechner gibt für jedes Pferd eine Siegchance an – am Beginn der Saison geschieht dies eher willkürlich, mit ihrem Fortgang werden mehr und mehr vorhergehende Siege berücksichtigt. Das Abschließen von Wetten über Beträge zwischen fünf und 500 Pfund ist Pflicht – die Quoten sind oft mehr als großzügig bemessen. Doppelte Chancen winken beim Setzen auf einen Konkurrenten: Eine Gewinnchance besteht sowohl durch das Preisgeld, das nur der Sieger erhält, als auch bei einer Wette auf Platz.

Durch den Einsatz der eigenen Pferde unter ungünstigen Bedingungen lassen sich schöne Coups landen. So kann man etwa zweimal nacheinander ein Pferd über Distanzen und bei Bodenverhältnissen einsetzen, bei denen es sicher verliert. Wenn es dann im nächsten Rennen unter optimalen Bedingungen läuft, sind die Gewinnquoten für Siegwetten sehr günstig. Der Spieler sollte sich jedoch vor Überbeanspruchung seiner Favoriten hüten – wer ein Pferd in zu vielen aufeinander folgenden Rennen starten läßt, muß lernen, daß auch der beste Renner Schonung braucht, wenn seine Leistung über längere Zeit erhalten bleiben soll.

Das Rennen beginnt

Nach Abschluß der Wetten beginnt das Rennen selbst: Wie auf dem Rennplatz gehen die Pferde an den Start, wo sie in die jeweiligen Positionen gebracht werden. Nicht einmal das Geräusch der aufschlagenden Pferdehufe wurde bei diesem Programm vergessen. Die Bodenverhältnisse sind bei jedem Derby verschieden, auch die Distanzen variieren. Ist eines ihrer Pferde so schlecht, daß sie es an drei aufeinander folgenden Renntagen nicht starten lassen, wird es aus der Teilnehmerliste gestrichen – allerdings ist dann bei jedem weiteren Derby eine Strafe fällig.

Gegen Ende der Rennsaison wird der Wettbewerb härter. Alle Mitspieler kennen inzwischen die Leistung ihrer Pferde und lassen möglichst nur noch die Renner mit der besten Kondition starten. Aber auch die Gewinnchancen wachsen: Für die drei Ersten gibt es im letzten Rennen der Saison immerhin zusammen 90 000 Pfund Preisgeld.

Classic Racing ist eines der faszinierendsten Programme für den Oric. Die Darstellung der Rennen ist ausgezeichnet. Auch bei wiederholtem Spiel kommt keine Langeweile auf, weil man in jeder Saison eine neue Strategie finden muß. Außerdem – eine abgeschlossene Saison kann bis zu 250 000 Pfund an Preisgeldern und Wettgewinnen einspielen. Welcher echte Rennplatz kann da mithalten?

Classic Racing: Für den Oric-1/Atmos
Hersteller: Salamander Software, 17 Norfolk Road, Brighton BN1 3 AA
Autor: Paul Neal
Joysticks: Nicht notwendig
Format: Cassette

Lernende Rechner

Im Felde der Künstlichen Intelligenz setzt sich zunehmend die Meinung durch, daß der einzige Lösungsweg für viele Probleme die Entwicklung eines sogenannten „Kinder"-Systems sei – eines Systems also, das lernfähig ist.

Wenn ein Computersystem die Durchführung einer bestimmten Aufgabe innerhalb einer gewissen Zeit verbessert hat, ohne neu programmiert worden zu sein, kann man sagen, daß es gelernt hat. Wenn ein Evaluationsprozeß nicht vorliegt, kann man auch nicht von Lernen reden.

Ein Lern-Algorithmus sollte eine oder mehrere der folgenden Anforderungen erfüllen:

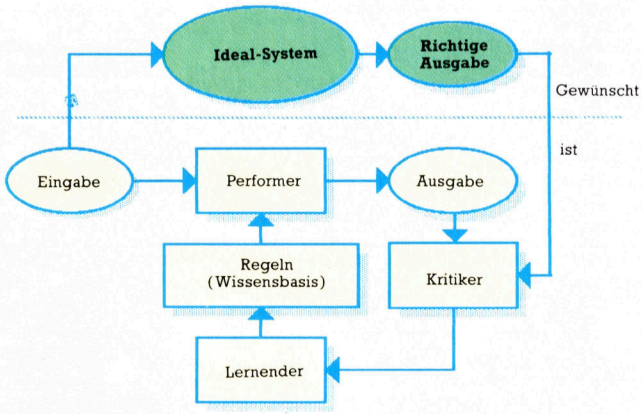
- eine Vielzahl von Problemen abdecken,
- genauere Lösungen anbieten,
- Antworten zu niedrigeren Kosten finden,
- programmiertes Wissen vereinfachen.

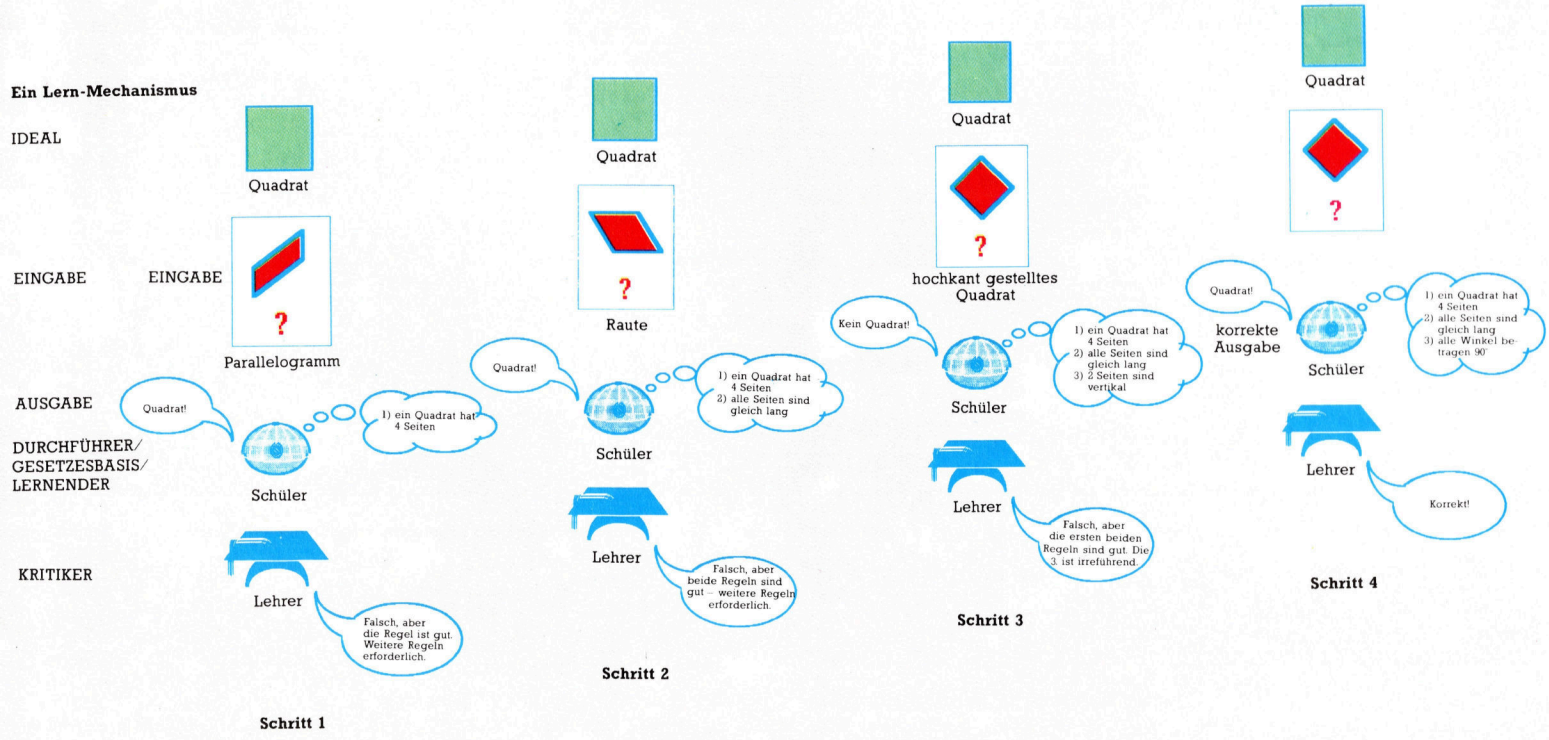
Der letzte Punkt setzt voraus, daß die Vereinfachung gespeicherten Wissens wertvoll ist, selbst wenn die Aufgabenlösung durch den Computer dadurch nicht verbessert wird. Diese Voraussetzung liegt vor, wenn das System mit einem Regelsatz beginnt und mit einem einfacheren endet.

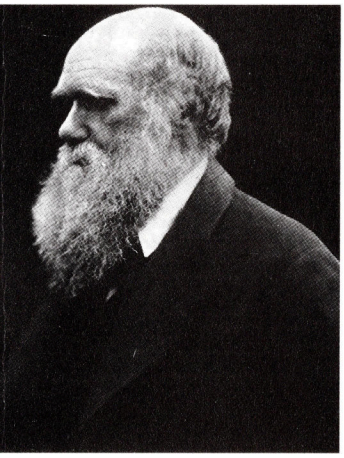
„Maschinen-Lernen" kann in vielen Bereichen stattfinden. Doch die erfolgreichsten Lernsysteme beschäftigen sich mit der Klassifizierung von Objekten. Ziel eines solchen Sy-

Kritischer Intellekt

Die Art, wie ein Kind lernt, ist komplex und noch nicht vollständig erforscht. Psychologen glauben, daß Kinder durch Bildung von Gesetzesstrukturen lernen, die „Schemata" genannt werden und auf Versuch und Irren basieren. Hypothesen (die „Regeln") werden getestet, und jene, die richtige Ergebnisse erzielen, werden behalten (= gelernt). In vielen Fällen wird der Lernprozeß von jemandem gesteuert, der das Wissen über ideale Ergebnise hat (dem „Kritiker"). Der Kritiker – hier als Lehrer dargestellt, hilft dem Kind („Lernender") beim Evaluieren und Bereinigen der internen Gesetzesreihe. KI-Lern-Systeme versuchen, dies nachzuahmen, indem eine Wissensbasis geschaffen wird, die in Verbindung mit einer Reihe von Beispielen und einer Evaluationsmethode verbessert werden kann.







Charles Darwins (1809–1882) Evolutionstheorie besagt, daß alle Arten (Lebensformen) sich als Reaktion auf ihre Umgebung durch natürliche Auslese verbessern bzw. anpassen. Nur die stärksten und am besten angepaßten Arten überleben und erzeugen so die nächste Generation, indem ihre Charakteristiken weitergegeben werden. Diese Grundidee fand erfolgreich im Bereich des maschinellen Lernens Anwendung. KI-Systeme konnten ihre Leistungsfähigkeit durch Evolution von Klassifizierungsgesetzen verbessern. Es ist kein Zufall, daß dieses System BEAGLE benannt wurde – nach dem Forschungsschiff, mit dem Darwin seine berühmte Expedition zu den Galapagosinseln unternahm.

stems ist die Begutachtung der Eingabedaten und die Klassifizierung, Interpretation und Identifikation derselben.

Man hat verschiedene Methoden ausprobiert, um die automatische Verbesserung von Programmausführungen zu erzielen. Zwei der einfachsten – Lernen durch Üben und Parameter-Regulierung – wurden erstmals von Arthur Samuel in seinen klassischen Lernstudien beim Damespiel angewandt. Zuvor hatten Rosenblatt und Selfridge Mustererkennungssysteme mit grundlegenden Lernfähigkeiten entwickelt, beispielsweise das Perzeptron.

Parameter-Regulierung ist eine Methode, bei der Programm-Koeffizienten und Parameter wiederholt reguliert werden, um die Ausführung zu verbessern. Dies ist eine einfache Form der Optimierung. Sie wurde in der Literatur zur angewandten Mathematik bereits erschöpfend behandelt. Lernen durch Üben ist völlig „unkreativ" und im Prinzip eine Speicher-Verdichtungstechnik. Dagegen sind die hier betrachteten Methoden zur Generalisierung imstande, folglich lernfähig, und können Antworten für Situationen finden, die zuvor nicht eingegeben wurden.

Jedes System, das für die Schaffung neuen Wissens entwickelt wird und beim Ablauf seine Durchführung verbessert, muß eine der folgenden Hauptkomponenten beinhalten:

- einen Satz von Datenstrukturen, der den derzeitigen Wissensstand des Systems darstellt (die „Regeln"),
- einen Aufgaben-Algorithmus (den „Performer"), der die Gesetze benutzt, um die Problemlösung zu steuern,
- ein Rückkopplungsmodul (die „Kritik"), das die tatsächlichen Resultate mit den gewünschten Zielen vergleicht,
- den Lernmechanismus selbst („Lernender"), der die Rückkopplung der Kritik benutzt, um die Regeln zu berichtigen.

Die zur Programmierung des Systemwissens gewählte Darstellungsmethode ist mindestens ebenso wichtig wie die Einzelheiten des Lern-Algorithmus. Deshalb muß vor dem Errichten eines Lernsystems sichergestellt sein, daß die beschreibende Sprache die Dinge wie benötigt ausdrücken kann. Angenommen, wir könnten dieses Problem lösen, so bliebe das Problem des Automatisierens, der internen Erzeugung genauer Beschreibungen.

Eine mögliche Problembetrachtungsweise ist die Suche durch alle Beschreibungen, die in einem gegebenen Kontext nützlich sein könnten. Ein KI-System, das Klassifikationen lernt, könnte mit einer Reihe von Parametern beginnen, die zur Erzeugung einer Klassifikation verwendet werden. Während der Lernphase würde das System weitere Beschreibungen durch Kombinieren der ursprünglichen Parameter generieren und evaluieren. Die Vielzahl der syntaktisch wichtigen erzeugten Beschreibungen wäre astronomisch. Und je ausdruckskräftiger die Beschreibungssprache, desto explosiver gestaltete sich das kombinatorische Problem. Deshalb muß ein Such-Leitweg gefunden werden, der die Mehrzahl der möglichen Beschreibungen, die irrelevant sind, ignoriert.

BEAGLE (Biological Evolutionary Algorithm Generating Logical Expressions) ist ein Computersystem, das Entscheidungsregeln mit Hilfe einer Datenbank produziert. BEAGLE arbeitet nach dem Prinzip „natürlicher Auswahl", wobei Regeln, die nicht zu den Daten passen, eliminiert und durch „Mutationen" besserer Regeln ersetzt werden. Oder aber neue Gesetze werden durch Verbindung zweier besser adaptierter Regeln geschaffen. Die Regeln sind Boolesche Ausdrücke, die durch Baumstrukturen dargestellt werden.

Als Basis zwei Programme

Die ursprüngliche Software bestand aus zwei PASCAL-Programmen, HERB (Heuristic Evolutionary Rule Breeder) und LEAF (Logical Evaluator And Forecaster).

HERB basiert auf drei Input-Dateien, die vom Anwender zu programmieren sind: ein Datenfile, ein „Payoff"-File und ein „Altgesetz"-File, das leer sein kann. Als Ausgabe wird ein neues Gesetzes-File produziert. Das Datenfile enthält den Trainings-Satz. Der Anwender muß auch eine „Payoff"-Matrix erstellen, die Werte oder Kosten korrekter und falscher Klassifizierungen definiert.

LEAF ist einfacher als HERB. Es ruft lediglich ein Datenfile im selben Format wie den Trainings-Satz ab und legt ein Gesetzes-File darüber. Es kann auf Abfrage unter anderem eine geordnete Liste der Bestandteile eines Datenfiles ausdrucken, die einer vorgegebenen Klasse (wie durch das Gesetzes-File definiert) am besten zu entsprechen scheinen, und eine weitere mit den am wenigsten passenden.

Der BEAGLE-Lern-Algorithmus besteht darin, daß die folgende Prozedur in mehreren „Generationen" wiederholt wird (ein kompletter Lauf durch den Trainings-Satz ist eine Generation).

1. Evaluiere jede Regel an jedem Beispiel in Übereinstimmung mit der „Payoff"-Matrix. Kürzere Regeln erhalten einen Bonus.
2. Ordne die Regeln entsprechend ihrer Wertigkeit in absteigender Folge und entferne die untere Hälfte der Liste.
3. Ersetze „tote" Regeln durch eine vorläufige Prozedur, die mit einem Paar zufällig gewählter überlebender Regeln kombiniert wird, indem Teile guter Regeln neu zusammengestellt werden.
4. Ändere einige zufällig gewählte Regeln (nicht aber die oberste) und benutze die Prozedur TIDY bei allen neuen Regeln, die für die nächste Generation bereit sind.

Die TIDY-Prozedur beschneidet gewisse

syntaktische Redundanzen, die erzeugt werden könnten. So etwa doppelte Negative und konstante Ausdrücke. Auf diese Art ist dann der „gestutzte" Regel-Baum mit demselben Satz von Regeln ausgestattet, aber kürzer und auch präziser in der Aussage.

Im nächsten Teil beschäftigen wir uns mit den Problemen der Implementierung einer einfachen BEAGLE-ähnlichen Lernmethode in BASIC.

Wetterbericht

Ein Beispiel für ein Lernsystem ist dieses, das den Wetterbericht eines Tages für die Vorhersage benutzt. Das System kann mit einer Reihe von Merkmalen beginnen, die das Wetter beschreiben – wie etwa Regenmenge, Menge des Sonnenscheins, maximale Windgeschwindigkeit und Luftdruck. Um die Regeln zu lernen, die der Maschine exakte Voraussagen ermöglichen, muß ein Übungssatz von Daten gesammelt werden. Bei jedem ist bekannt, ob es am darauffolgenden Tag geregnet hat oder nicht. So könnte eine typische Aufzeichnung sein:

Regen	0
Sonnenschein	7,2
Wind	22
Luftdruck	1017

Das System muß Regeln (im ersten Beispiel zufällig) erzeugen und den Übungssatz zur Überprüfung der Gültigkeit benutzen. Die Gesetze können beispielsweise als folgende Komponenten gespeichert werden:

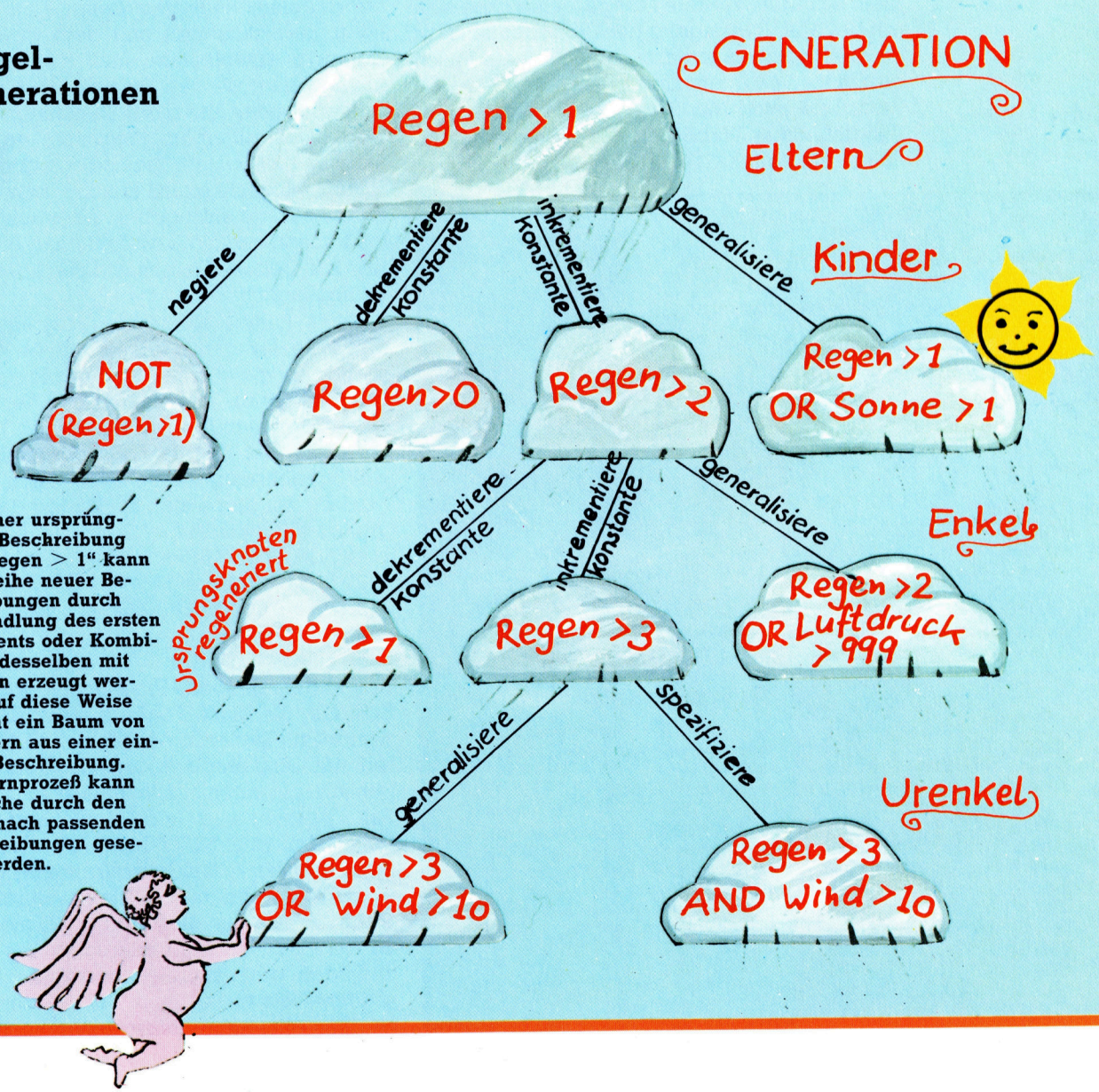
Merkmale	Wind
Konstanten	10
Vergleichsoperatoren	> oder <
Logische Operatoren	AND, OR und NOT

Diese werden zu Ausdrücken wie dem folgenden kombiniert:

Sonnenschein < 4 AND Luftdruck > 1000

Nach jedem Übungslauf werden die guten Gesetze gespeichert und die schlechten entfernt, um durch neue Gesetze, „geboren" aus den verbleibenden guten Gesetzen, ersetzt zu werden. Durch Evolution einer Gesetzesreihe, die ihre Leistungsfähigkeit bei jedem Übungslauf verbessert, sollte die Maschine ihr Wettervorhersagevermögen steigern können.

Regel-generationen



Aus einer ursprünglichen Beschreibung wie „Regen > 1" kann eine Reihe neuer Beschreibungen durch Umwandlung des ersten Statements oder Kombination desselben mit anderen erzeugt werden. Auf diese Weise entsteht ein Baum von Ablegern aus einer einzigen Beschreibung. Der Lernprozeß kann als Suche durch den Baum nach passenden Beschreibungen gesehen werden.

Fehlersuche

Methoden zum Erkennen und Verbessern von Fehlern sind für die Programmentwicklung unverzichtbar. Schon durch einfache Tippfehler kann das Programm abstürzen. Ebenso ernste Folgen können jedoch durch versteckte Mängel in der Logik oder beim Programmaufbau verursacht werden.

Fehlerquellen finden sich auf jeder Stufe der Programmentwicklung, bei der Festlegung der Programmeigenschaften ebenso wie bei der Ausarbeitung, der Eingabe oder auch beim Prüfen eines Programms. Schon die Vorüberlegungen bereiten manchmal einem späteren Fehler den Weg. Wenn der Einblick in das gestellte Problem fehlt, entstehen unter Umständen Programme, die etwas völlig anderes tun als beabsichtigt. Hier können die Methoden der strukturierten Programmierung Abhilfe schaffen. Weitere Mängel schleichen sich oft bei der Übertragung der Programmstruktur in die Programmsprache ein. Schon ein falsch getippter Variablenname hat verheerende Folgen. Und auch die Testphase ist nicht ungefährlich. Beim Verbessern eines Fehlers kann leicht ein neuer erzeugt werden.

Die meisten Programmfehler treten allerdings an den „Schnittstellen" auf, entweder zwischen zwei einzelnen Programmteilen oder auch zwischen Programm und Anwender. Beim Passieren einer solchen Schnittstelle sollten alle Werte darauf geprüft werden, ob sie der für das Programm geeigneten Daten-Kategorie angehören. Die Werte können entweder vor der Ausgabe oder im empfangenden Programmteil geprüft werden.

Um sicherzustellen, daß die von einem Unterprogramm weitergegebenen Daten im richtigen Bereich liegen und dem gewünschten Datentyp entsprechen, muß nach ihrer Herkunft gefragt werden. Stammen sie aus einem Datenfile oder aus einer Tastatureingabe? Im Gegensatz zu einem Programm arbeitet ein Mensch nicht immer methodisch und hat zahlreiche Möglichkeiten, falsche bzw. unangebrachte Werte einzugeben. Aber auch Datenfiles können zerstört bzw. fehlerhaft eingelesen werden. Genaue Tests sind daher auf jeden Fall unumgänglich.

Nicht immer stürzt ein Programm durch einen Fehler gleich ab. Wenn es doch geschieht, ist meist eine Regel der Programmiersprache (etwa die unerlaubte Verwendung eines Operators bei RESULT=FIRST$ + SECOND$) oder des Betriebssystems (etwa zuviele gleichzeitig offene Files) nicht beachtet worden. Auf den ersten Blick wirkt dieses Programm völlig korrekt:

```
10 FOR COUNTER = 1 TO 10
20 SUM = SUM + 1
30 PRINT COUNTER,SUM
40 GOTO 10
50 NEXT COUNTER
```

In Wahrheit führt das Programm jedoch in eine Schleife, die einen Fehler auslöst: BASIC verwendet zur Steuerung der FOR. . .NEXT-Schleifen ein Stapelregister, das zu Beginn jeder Schleife um einen bestimmten Wert (in diesem Fall 1) erhöht wird. In diesem Programm wird Zeile 50 mit dem NEXT-Befehl nie erreicht. Das Stapelregister (Stack) wächst immer weiter an – bis zum Programmabbruch durch „Stack Overflow". Fehler dieser Art sind meist leicht zu beheben. Sie sind allerdings nur mühsam zu finden, wenn sie in selten gebrauchten Programmteilen vorkommen. Für die Entdeckung ist dann eine sehr umfangreiche Suche nötig.

Strukturierte Fehlersuche

Fehler werden durch logisches Vorgehen vermieden bzw. schneller gefunden. Dieser Fragenkatalog ist ein Abriß der strukturierten Fehlersuche.

Variablen

1. Sind alle Variablennamen eindeutig? Haben Sie daran gedacht, daß manche Interpreter nur die zwei ersten Buchstaben des Variablennamens lesen?
2. Sind Variablen (speziell Schleifenzähler und Parameter) mehrfach verwendet worden, obwohl sie noch definierte Werte enthalten?
3. Sind die Array-Indizes ganzzahlig? Liegen sie in den erlaubten Grenzen?
4. Beginnen die Array-Indizes beim Element 0 oder beim Element 1?

Berechnungen

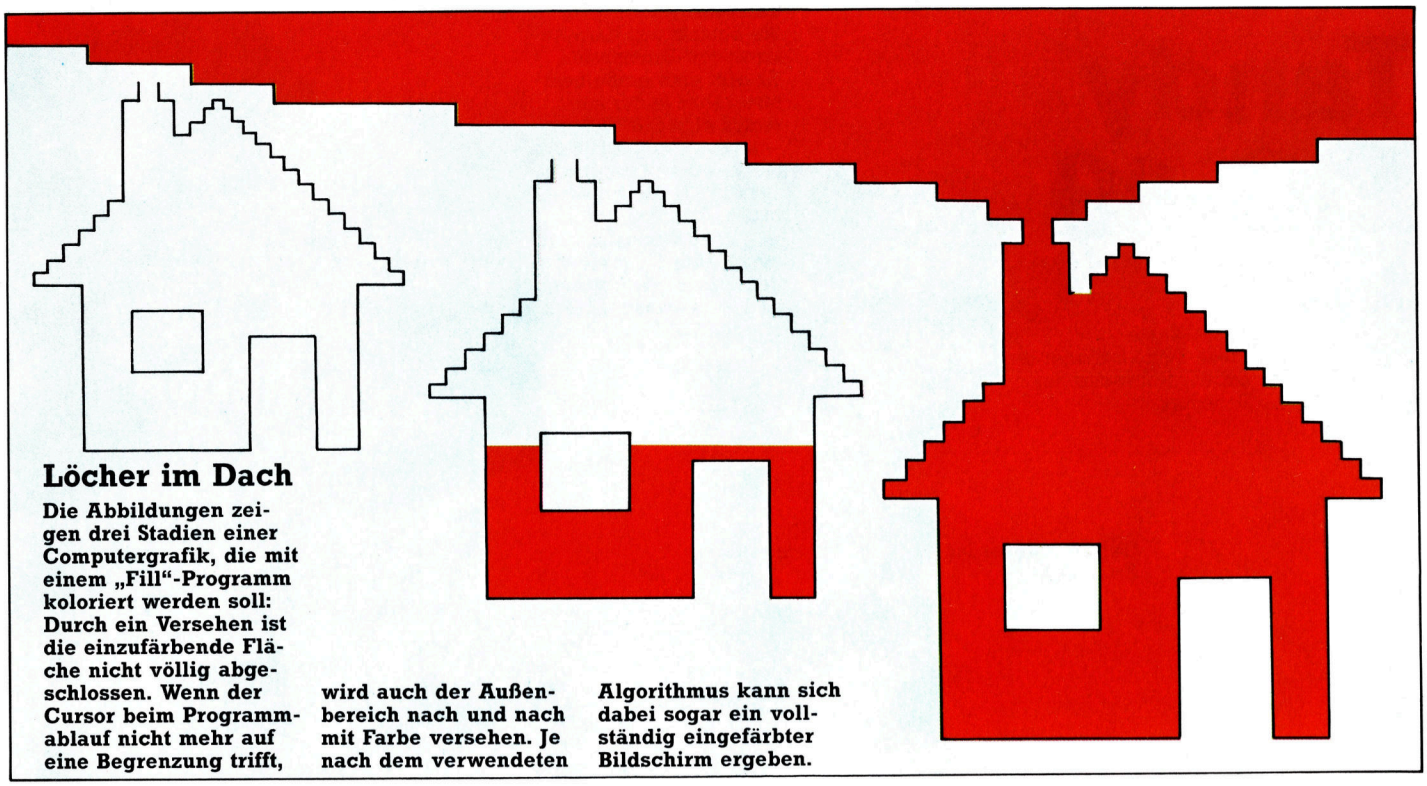
1. Ergeben Berechnungen Strings oder numerische Resultate? Sind die Ergebnisse String- oder numerischen Variablen zugeordnet?
2. Ergibt eine Berechnung Zahlen, die für Computerberechnungen zu groß oder zu klein sind? Kann das zu einer Division durch Null führen?
3. Sind Rundungsfehler von Bedeutung?
4. Werden alle Operationen in der richtigen logischen Reihenfolge ausgeführt?

Vergleichsoperationen

1. Werden Strings nur mit Strings und Ziffern nur mit Ziffern verglichen?
2. Kommt es bei den Strings auf teilweise bzw. vollständige Groß-/Kleinschreibung an?
3. Werden Strings von unterschiedlicher Länge verglichen? Spielt beim Vergleich der Unterschied zwischen den Zeichen oder die Differenz der Stringlänge die wichtigere Rolle?
4. Sind Boolesche- und Vergleichsoperatoren richtig zusammengestellt? A > B OR C beispielsweise ist ungleich A > B OR A > C!

Programmstruktur

1. Begrenzen Schleifen und Algorithmen den Wert irgendwelcher Variablen?
2. Hat jede Schleife und jede Routine nur einen Ein- und einen Ausgang?
3. Geht der Programmablauf nach einem unzutreffenden IF. . .THEN-Befehl zum nächsten Befehl oder in die nächste Programmzeile?
4. Was passiert, wenn bei einer Vielfach-Verzweigung keine einzige der Testbedingungen erfüllt ist?



Löcher im Dach

Die Abbildungen zeigen drei Stadien einer Computergrafik, die mit einem „Fill"-Programm koloriert werden soll: Durch ein Versehen ist die einzufärbende Fläche nicht völlig abgeschlossen. Wenn der Cursor beim Programmablauf nicht mehr auf eine Begrenzung trifft, wird auch der Außenbereich nach und nach mit Farbe versehen. Je nach dem verwendeten Algorithmus kann sich dabei sogar ein vollständig eingefärbter Bildschirm ergeben.

Besondes heimtückisch sind Fehler, die nur die Resultate eines Programms verfälschen, das ansonsten einwandfrei läuft. Als Beispiel haben wir ein „Fill"-Programm gewählt, das auf den Bildschirm gezeichnete Umrisse in einer bestimmten Farbe ausmalt. Das Programm orientiert sich dabei an den Grenzlinien. Beim Erreichen einer Linie läuft der Cursor in einer anderen Richtung weiter, bis er wieder an eine Umrißlinie stößt. Für einen korrekten Programmablauf müssen die Grenzlinien genau definiert und vollständig sein. Auch eine kleine Unterbrechung führt unvermeidlich zum „Auslaufen" der Farbe.

Fehlererkennung

Die BASIC-Dialekte der meisten Heimcomputer unterstützen die Fehlersuche durch differenzierte, eindeutige Fehlermeldungen und die Möglichkeit, ein abgestürztes Programm nach der Eingabe anderer Variablenwerte neu zu starten. Viele Dialekte verarbeiten auch Befehle wie ON ERROR GOTO: Der Programmablauf wird zu einer Routine für die Fehlerbehandlung „umgeleitet". Dafür fügt man beispielsweise diese Zeile in das Programm ein:

```
30 ON ERROR GOTO 20000:REM
    ROUTINEN ZUR FEHLERBEHANDLUNG
```

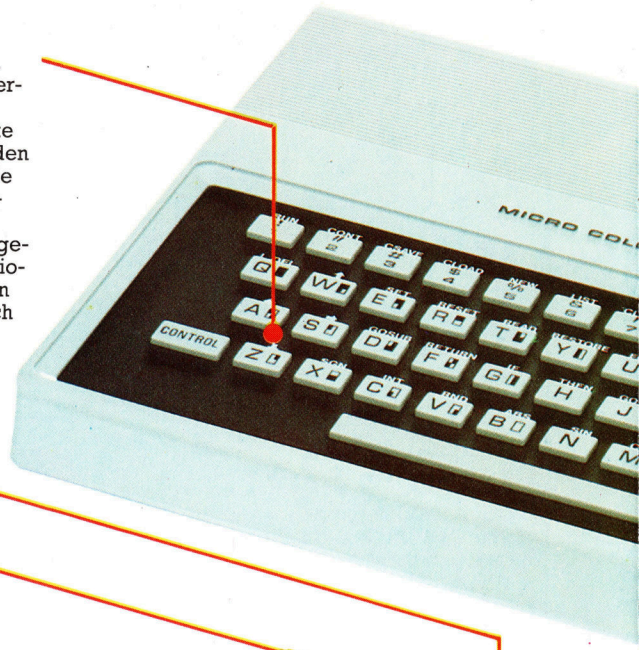
Tritt ein Fehler auf, würde das Programm auf Zeile 20000 verzweigen. ON ERROR setzt im allgemeinen auch noch zwei Variablen, eine für den Fehlertyp, die zweite für die Zeilennummer, in der der Fehler aufgetreten ist. Die Variablennamen wie auch die Werte des Fehlertyps sind je nach verwendetem Rechner unterschiedlich – genauere Angaben können Sie dem Handbuch Ihres Rechners entnehmen.

Gut durchdachte Programme kommen mit einer einzigen ON-ERROR-Routine aus. Natürlich kann eine solche Routine keine Syntaxfehler beheben oder das Überlaufen eines Stack-Registers verhindern. Der Nutzen liegt eher in einem sauberen Programmabschluß ohne offene Files. Außerdem erfährt der Anwender ungefähr, was den Fehler hervorgerufen hat.

Manche Fehler, etwa eine Division durch die Null, könnten zwar mit einer ON-ERROR-Routine behandelt werden, aber man sollte trotzdem darauf verzichten. Dafür sprechen mehrere Gründe:

- Der ON-ERROR-Befehl mit einem Sprung ins Hauptprogramm beinhaltet je einen Ein- und Ausgang – eine Verletzung der Regel für strukturierte Programmierung, die nur einen Ein- und Ausgang pro Teilprogramm vorsieht.
- Eine Division durch Null sollte dort verhindert werden, wo dividiert wird: Ein guter Programmierer vermeidet Algorithmen, die zu einem Systemabsturz führen könnten. Verlangsamen die dafür nötigen Vorkehrungen das Programm übermäßig, sollte man es entsprechend umschreiben.
- Fehlerbehandlungs-Routinen können schnell zu unhandlichen Ketten von IF...THEN...ELSE-Befehlen mit diversen Verzweigungen anwachsen. Sie sind mit den Zeilennummern des Restprogramms verbunden und müssen daher bei jeder Änderung eines Unterprogramms neu geschrieben werden. Zudem sind sie schwierig zu planen und noch schwieriger zu prüfen. Mängel bei der Fehlerbehandlung können den Programmablauf in völlig unvorhergesehener Weise stören.

Tandy MC-10



Tastatur
Sie besteht aus Hartkunststoff-Drucktasten. Es gibt eine große Leertaste, aber leider nur rechts eine SHIFT-Taste – an der entsprechenden Stelle links ist dafür die CONTROL-Taste angebracht. Das Anschlaggefühl ist durchaus angenehm, aber für professionelles Schnellschreiben ist die Tastatur natürlich ungeeignet.

Reset-Taste
Diese Taste befindet sich auf der Rückseite des Rechners.

Cassettenrecorder-Anschluß

RS232-Schnittstelle
Für vierpoligen DIN-Stecker; auf Pin 1 liegt der Empfangssignalpegel, Pin 2 überträgt die Empfangsdaten, Pin 4 die Sendedaten und Pin 3 ist Masse.

Platinenstecker
Außer einem Speicherausbau (16-KByte-Steckmodul) sollte diese Schnittstelle auch andere Erweiterungen zulassen.

ROM-Chip
Er enthält das Microsoft-BASIC.

CPU
Der hier verwendete Prozessor 6803 (aus der 6800er-Serie von Motorola) ist weniger bekanntgeworden als etwa der 6502 oder der Z80 – trotzdem ein sehr brauchbarer 8-Bit-Chip mit umfangreichem Befehlssatz.

Statisches RAM
Die vier KByte RAM verteilen sich auf zwei Chips. Fast 1 K davon ist aber schon für Bildschirmspeicher und Systemvariable vergeben.

Video-Prozessor 6847
Er dient zur Steuerung der Bildschirmausgabe. Der 6847 kann für die verschiedensten Ausgabeformate programmiert werden.

Obwohl dieser Rechner zur unteren Preisklasse gehört, bietet er eine gute Farbqualität und für den Einsteiger viele Möglichkeiten.

Der MC-10 von Tandy ist ein kompakter kleiner Computer, bei dem mit wenigen leistungsfähigen Chips eine Menge erreicht worden ist. Auch für die Benutzerfreundlichkeit ist einiges getan. Die Tastatur ist größer als die einiger anderer Rechner dieser Preisklasse und hat eine große Leertaste. BASIC-Befehle sind durch Betätigung nur einer Taste (bei gleichzeitigem Drücken von CONTROL) einzugeben. Nach der Inbetriebnahme steht die Tastatur automatisch auf „Großbuchstaben"; zum Umstellen muß man jeweils SHIFT 0 drücken.

Das Format der Bildschirmausgabe ist kleiner als bei den meisten Heimcomputern: Es sind nur 16 Zeilen zu 32 Zeichen darstellbar, und entsprechend grob ist auch die Grafikauflösung. Die Bilddarstellung kommt auch sonst etwas kurz, beispielsweise hinsichtlich der Farbwahlmöglichkeiten, obwohl die Farbqualität an sich sehr gut ist. Kurioserweise erscheinen Kleinbuchstaben nur als invertierte Großbuchstaben und Texte allgemein nur grün auf

Tandy MC-10

ABMESSUNGEN
210 × 178 × 51 mm

CPU
6803-Prozessor

TAKTFREQUENZ
4,4 MHz

SPEICHER
8 KByte ROM, 4 KByte RAM

BILDSCHIRMDARSTELLUNG
16 Zeilen zu 32 Zeichen, neun Farben (entweder Vorder- oder Hindergrund wählbar), 75 fest definierte Zeichen

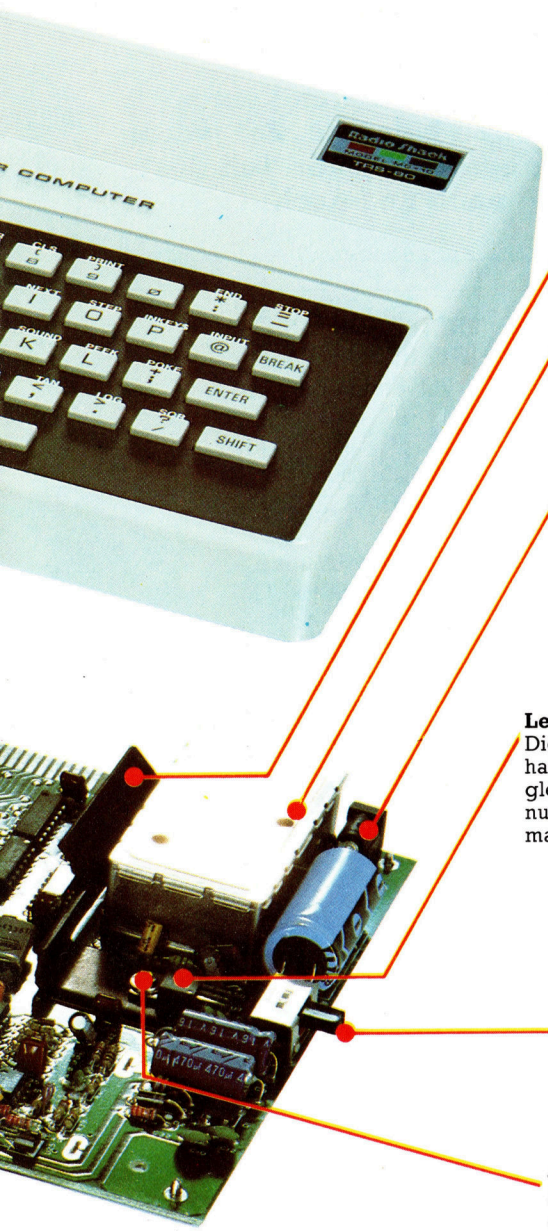
SCHNITTSTELLEN
RS232, Cassettenrecorder- und Fernseher-Anschluß

PROGRAMMIERSPRACHE
BASIC

MITGELIEFERTES ZUBEHÖR
Bedienungs- und BASIC-Anleitung, Antennenkabel

TASTATUR
48 Tasten

ANLEITUNG
Enthält nur wenige technische Details. Beigefügt ist ein Merkblatt mit den wichtigsten BASIC-Informationen.



Kühlkörper
Die Wärme wird über die große Oberfläche des Kühlkörpers an die Umgebung abgeführt.

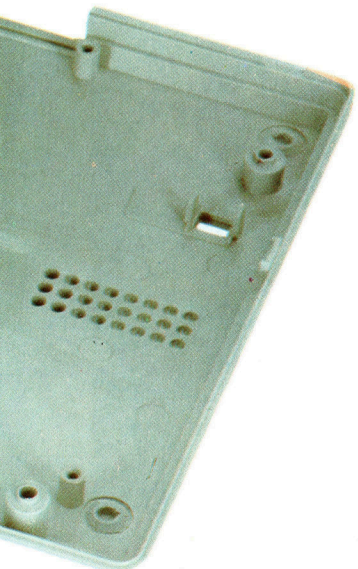
HF-Modulator
Zur Umsetzung des Signals vom Video-Prozessor in den Fernsehkanal. Mit einem speziellen Monitor-Ausgang ist der Rechner nicht ausgestattet.

Netzteilbuchse
Für den Anschluß eines externen Steckertransformators.

Leistungstransistor
Dieser Spannungsregler hat die Aufgabe, die gleichgerichtete Spannung vom Netztransformator konstant zu halten.

Betriebsschalter
Zum Ein/Ausschalten der Versorgungsspannung.

4,4-MHz-Quarz
Seine Frequenz wird für diverse Verwendungszwecke im System noch mehrfach unterteilt.



schwarz oder umgekehrt. Blockgrafiksymbole können zwar in neun verschiedenen Farben ausgegeben werden, jedoch nur auf schwarzem Untergrund.

Der einzige vorhandene Tonkanal läßt nur die Variation von Tonhöhe und -dauer zu. Der Tandy MC-10 ist mit einer Fernseher- und einer Cassettenrecorder-Schnittstelle sowie mit einem seriellen Interface (RS232) ausgestattet, das für einen Drucker oder für den Datenverkehr mit anderen Rechnern (auch im Rahmen eines MC-10-Netzwerks) verwendet werden kann.

An Spiele haben die Konstrukteure dieses Rechners wohl weniger gedacht, sonst hätten sie ihn mit mindestens einem Joystick-Eingang ausgerüstet. Auch an einen speziellen Grafik- oder Soundchip, der bei spielorientierten Rechnern meistens anzutreffen ist, wurde beim MC-10 nicht gedacht.

Offensichtlich wurden ursprünglich mehr Ausbaumöglichkeiten geplant, denn hinter einem verschraubten Deckel ist über einen Platinenstecker der Systembus zugänglich. Laut Herstellerangabe kann man dort als Speichererweiterung ein 16-KByte-RAM-Steckmodul unterbringen; das Handbuch gibt keinen Hinweis, was da vielleicht sonst noch angeschlossen werden könnte. Auch sonst enthält die Dokumentation nicht sehr viel konkrete technische Informationen. Abgesehen davon wurde der Text sehr unübersichtlich geschrieben.

Bei der Betrachtung der Rechnerdaten sollte man berücksichtigen, daß als Arbeitsspeicher nur 3142 Bytes für den Benutzer verfügbar sind, weil das 4-KByte-RAM auch den Bildschirmspeicher und etliche Systemvariablen beherbergt. Insgesamt ist der MC-10 in bezug auf das Preis/Leistungsverhältnis durchaus erwähnenswert.

Logischer Aufbau

In dieser Artikelserie werden wir uns mit der Entwicklung eines Grafik-Spiels für den Acorn B und den Electron befassen. Bei jedem Entwicklungsabschnitt des Spiels wird der entsprechende Teil des Programms aufgelistet, so daß Sie das Spiel nach und nach zusammenstellen können.

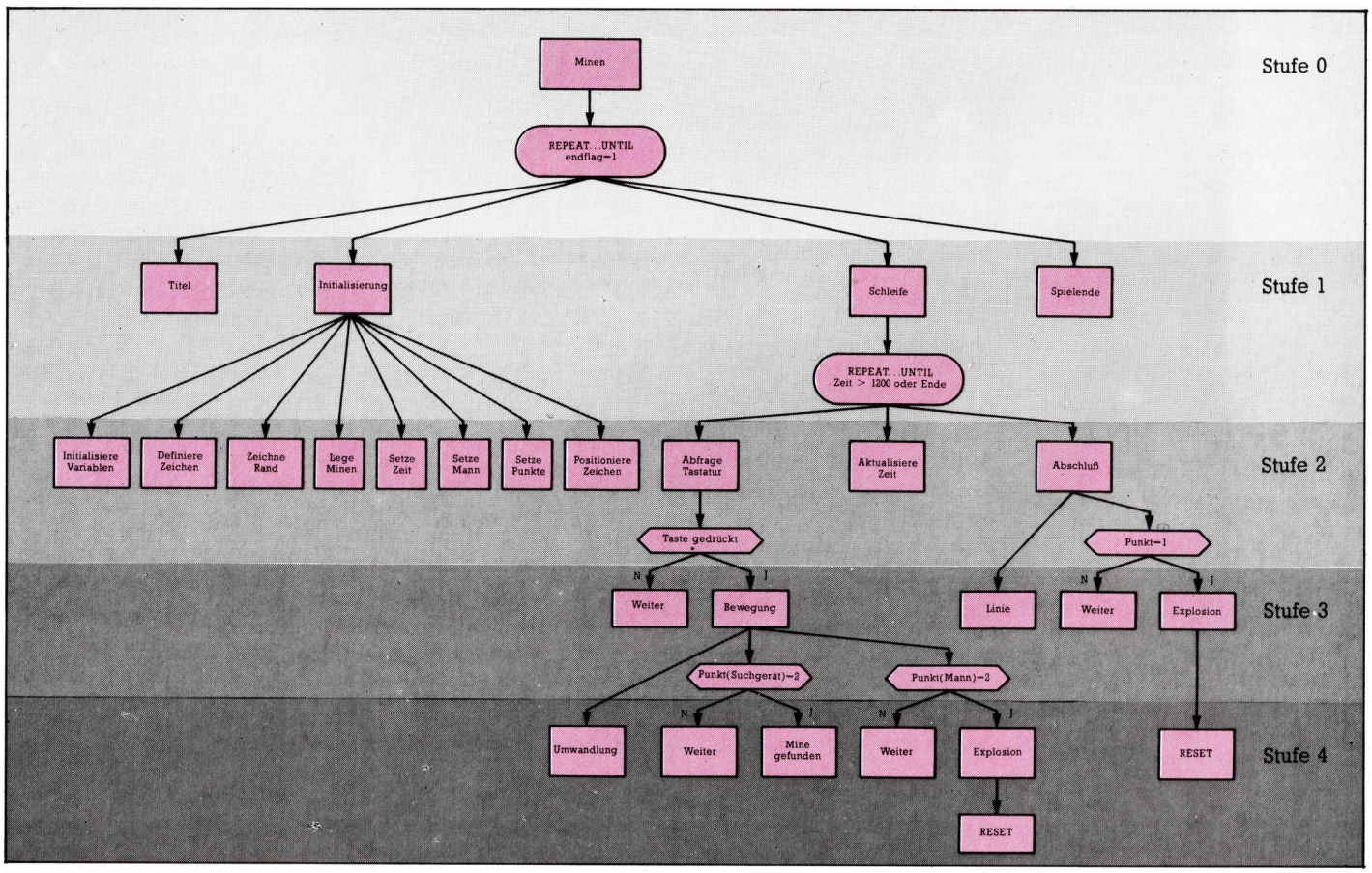
Das Acorn-BASIC hat gegenüber dem Standard-Microsoft-BASIC zwei große Vorteile: die hohe Ausführungsgeschwindigkeit und die Strukturierungsmöglichkeiten. Der Sinn der Entwicklung eines strukturierten Programms liegt darin, unabhängige Programmteile zu entwickeln, die einzeln getestet werden können, bevor sie zu einem großen Programm zusammengefügt werden. Bis zu einem gewissen Punkt kann jedes BASIC-Programm strukturiert werden, indem man Unterroutinen für einzelne Programmteile verwendet. Das Acorn-BASIC bietet jedoch spezielle Unterroutinen, genannt Prozeduren. Man kann sie sich als Programmblöcke vorstellen, die für eine spezielle Aufgabe entwickelt wurden. Stellen Sie sich beispielsweise ein Programm vor, das zwischen den Anweisungen jeweils eine Pause machen soll. Im Standard-BASIC könnte man hierzu eine Verzögerungsschleife verwenden:

```
10 PRINT "ERSTER ABSCHNITT"
20 FOR I=1 TO 100: NEXT I
30 PRINT "ZWEITER ABSCHNITT"
40 FOR I=1 TO 100: NEXT I
50 PRINT "DRITTER ABSCHNITT"
60 FOR I=1 TO 100: NEXT I
70 PRINT "VIERTER ABSCHNITT"
80 END
```

Die Zeitschleife als Unterroutine:

```
10 PRINT "ERSTER ABSCHNITT"
20 GOSUB 100
30 PRINT "ZWEITER ABSCHNITT"
40 GOSUB 100
50 PRINT "DRITTER ABSCHNITT"
60 GOSUB 100
70 PRINT "VIERTER ABSCHNITT"
80 END
100 REM ** UNTERROUTINE **
110 FOR I=1 TO 100: NEXT I
120 RETURN
```

Im Gegensatz zu Flußdiagrammen zeigt dieses Struktur-Diagramm mehr die Struktur der Prozeduren des Programms als den Programmfluß. Eine ovale Form zeigt den Beginn einer REPEAT. . .UNTIL-Schleife an. Die Rauten sind Entscheidungskästen – wenn ein Test negativ ausfällt, wird die Schleife fortgesetzt. Die Stufen-Nummern zeigen die Blockstruktur des Programms. Alle Schleifenanfänge und Prozedur-Aufrufe eröffnen einen neuen Programmabschnitt und eine niedrigere logische Stufe.



Wird die Unterroutine durch eine Prozedur ersetzt, erhält man folgendes Programm:

```
10 PRINT "ERSTER ABSCHNITT"
20 PROCdelay
30 PRINT "ZWEITER ABSCHNITT"
40 PROCdelay
50 PRINT "DRITTER ABSCHNITT"
60 PROCdelay
70 PRINT "VIERTER ABSCHNITT"
80 END
100 REM ** DEFINIERE PROCEDURE **
110 DEF PROCdelay
120 FOR I=1 TO 100: NEXT I
130 ENDPROC
```

Es gibt viele Ähnlichkeiten zwischen der Konstruktion einer Unterroutine und einer Prozedur. So stehen beispielsweise beide hinter der END-Anweisung, können jedoch vom Hauptprogramm aus aufgerufen werden. Der prinzipielle Vorteil einer Prozedur ist, daß sie per Namen und nicht über eine Zeilennummer aufgerufen wird. Die DEFinition der PROCedure kann irgendwo nach der END-Anweisung erfolgen.

Einsatz von Prozeduren

Wollen wir in unserem Beispielprogramm erreichen, daß die Zeitschleifen unterschiedlich lang sind, kann noch ein weiterer Vorteil der Prozeduren genutzt werden: die Möglichkeit, Werte an eine Prozedur zu übergeben. Im folgenden Beispiel sollen zwischen dem ersten und zweiten Abschnitt 100, zwischen dem zweiten und dritten 200 und zwischen dem dritten und vierten Abschnitt 175 Schleifendurchläufe eingefügt werden. Im Standard-BASIC müßte der Wert von I vor Aufruf der Unterroutine jeweils neu gesetzt werden. Bei der Verwendung von Prozeduren muß lediglich am Ende des Aufrufes der gewünschte Wert in Klammern eingegeben werden:

```
10 PRINT "ERSTER ABSCHNITT"
20 PROCdelay(100)
30 PRINT "ZWEITER ABSCHNITT"
40 PROCdelay(200)
50 PRINT "DRITTER ABSCHNITT"
60 PROCdelay(175)
70 PRINT "VIERTER ABSCHNITT"
80 END
100 REM ** DEFINIERE PROCEDURE **
110 DEF PROCdelay(N)
120 FOR I=1 TO N:NEXT I
130 ENDPROC
```

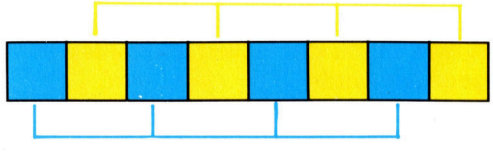
Um mehrere Werte zu übergeben, müssen diese durch Kommas getrennt werden.

Das Spiel, das wir entwickeln wollen, ist für einen Spieler gedacht und verwendet die Cursor-Tasten, um ein Minensuchgerät über ein Minenfeld zu bewegen. Für jede erfolgreich beseitigte Mine gibt es Punkte. Dabei gibt es einige Dinge, die dem Spieler die Arbeit erschweren. Hauptproblem ist Ihr Assistent, der jede Ihrer Bewegungen spiegelbildlich nachmacht. Dadurch müssen Sie immer aufpassen, daß er durch Ihre Bewegungen nicht auf eine Mine tritt. Außerdem gibt es eine Zeitbeschränkung auf zwei Minuten je Spiel. Im fertigen Spiel werden Sie vier Assistenten pro Spiel und die Auswahl zwischen neun Schwierigkeitsstufen (0–9) haben.

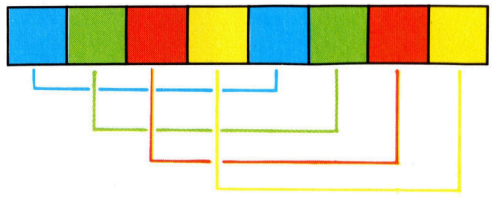
Das Struktur-Diagramm (links) gibt an, welche Prozeduren benötigt und wie diese zusammengesetzt werden. REPEAT. . .UNTIL-Schleifen werden in diesem Diagramm in ovaler Form dargestellt.

Bevor wir mit der Definition von Routinen zur Darstellung von Objekten beginnen können, müssen wir einen Bildschirm-Modus auswählen. Dabei sind drei Hauptfaktoren zu berücksichtigen: Auflösung, Farbe und Speicher. Bei einem kurzen Programm mag der Speicherplatzbedarf keine Bedeutung haben, doch das Programm, das wir hier entwickeln wollen, ist relativ umfangreich. Wir benötigen verschiedene Farben, damit Minen, Suchgerät und Assistenten leichter zu unterscheiden sind. Beim Acorn B stehen zwei Modi mit mittlerer Auflösung zur Verfügung. Modus 2 bietet 16 Farben, wogegen in Modus 5 nur vier Farben verwendet werden können. Betrachtet man das Bit-Muster, wird klar, warum Modus 5 weniger Speicher verbraucht als Modus 2.

Im Gegensatz zu vielen anderen Rechnern speichert der Acorn Farb- und Pixeldaten in einem Byte pro Bildschirmpunkt. In Modus 2 werden vier Bits zur Kennzeichnung der 16 Farben gebraucht. Somit kann ein Byte nur Farbinformationen für zwei Pixel speichern. Die Bits innerhalb der Bytes sind wie folgt angeordnet:



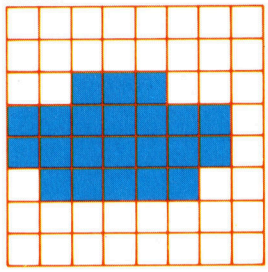
Da Modus 5 auf vier Farben begrenzt ist, werden nur 2 Bits je Farbinformation gebraucht. Ein Byte kann also vier Pixel repräsentieren:



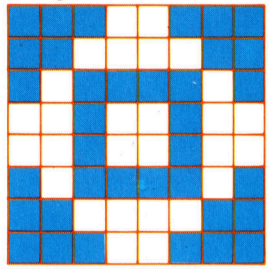
Beide Modi haben eine Auflösung von 160 × 256 Pixeln, woraus sich der Speicherplatzbedarf für einen Modus-2-Bildschirm ergibt: (160 × 256)/2=20 KByte. Modus 5 benötigt nur (160 × 256)/4=10 KByte Speicher. Wählt man Modus 5, stehen also zehn KByte zusätzlich zur Verfügung.

Die Konstruktion von anwenderdefinierten Zeichen ist auf dem Acorn sehr einfach. Jedes Zeichen besteht aus acht Zahlen, die die Dezi-

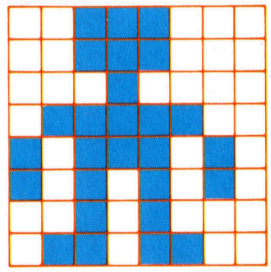
Mine



Suchgerät



Assistent



Das sind die anwenderdefinierten Zeichen. Die Pixel-Informationen sind in acht Byte unterteilt, wobei jedes Byte das binäre „Abbild" einer Zeile des Objekts repräsentiert.

maläquivalente jeder Reihe repräsentieren.

Mit dem Befehl VDU23 kann der Programmierer Zeichen mit ASCII-Codes von 226 bis 255 definieren. Der zweite Wert des VDU-Befehls gibt an, welchen Code Sie dem durch die letzten acht Zahlen definierten Objekt zuordnen wollen.

VDU23,224,0,0,56,254,254,124,0,0

Durch diese Anweisung wird CHR$(224) als Mine definiert. Zur Ausgabe des Zeichens wird die Anweisung PRINT CHR$(224) verwendet.

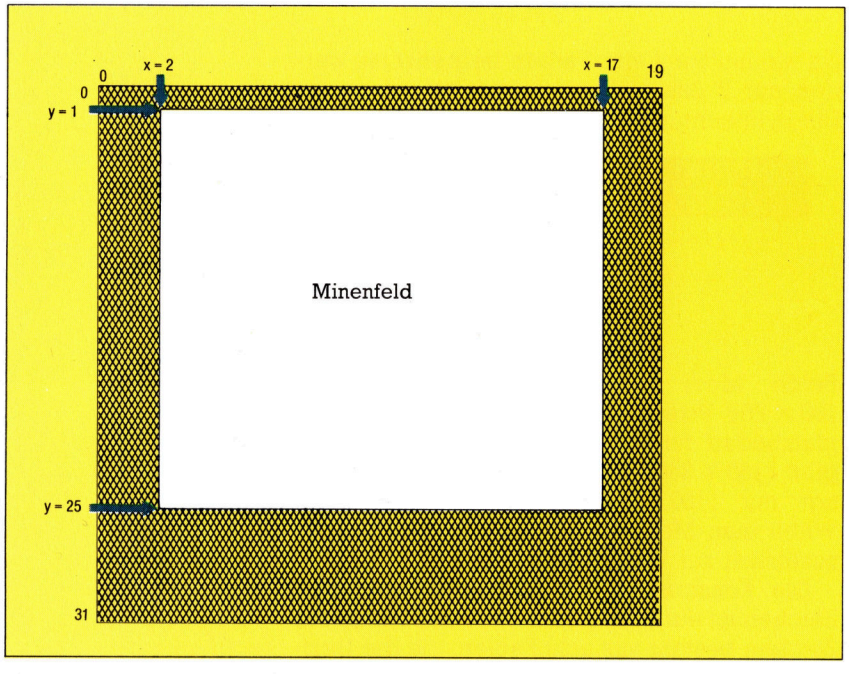
```
2380 DEF PROCdefine_characters
2390 REM ** MINES **
2400 VDU23,224,0,0,56,254,254,124,0,0
2410 REM ** MINE DETECTOR **
2420 VDU23,225,231,195,189,36,36,189,195,231
2430 REM ** ASSISTANT **
2440 VDU23,226,56,56,16,124,186,170,40,108
2450 ENDPROC
```

Bildschirmdarstellung

Sind die drei Objekte definiert, können sie auf dem Bildschirm ausgegeben werden. Am einfachsten geht das mit PRINT TAB(X,Y). In Modus 5 gibt es 32 Reihen mit je 20 Zeichen. Das bedeutet, daß X einen Bereich von 0 bis 19 und Y von 0 bis 31 hat. Das Minenfeld belegt den im Diagramm gezeigten Bereich.

Um Minen zufällig in dem Bereich „auszulegen", benutzen wir den RND(N)-Befehl. Ist N eine ganze Zahl, ergibt RND(N) als Ergebnis eine ganze Zahl zwischen 1 und N. Die horizontale Koordinate jeder Mine muß so gewählt werden, daß sie zwischen 2 und 17 liegt. RND(16) ergibt Zahlen von 1 bis 16, so daß RND(16)+1 die gewünschten Resultate bringt. Die folgende Prozedur produziert so viele Minen, wie durch den zuvor übermittelten Wert angegeben wird:

Das Minenfeld belegt 20 Reihen mit je 16 Zeichen. Die Minen werden während der Initialisierungsroutine zufällig „ausgelegt".




```
2560 DEF PROClay_mines (numer_mines)
2570 REM ** CHANGE COLOUR 2 TO GREEN **
2580 VDU19,2,2,0,0,0
2590 FOR I=1 TO number_mines
2600 PRINTTAB(RND(16)+1,RND(25));CHR$(224)
2610 NEXT I
2620 ENDPROC
```

In Modus 5 stehen normalerweise die Farben Schwarz, Rot, Gelb und Weiß zur Verfügung, entsprechend den Farbnummern 0, 1, 2 und 3. Man kann die Farben jedoch mit VDU19 ändern. Die Zahlen 0 bis 3 sind bekannt als logische Vordergrundfarben. Jede der 16 Farben des Acorn hat eine Nummer, die unabhängig vom verwendeten Modus ist. Diese werden aktuelle Farbzahlen genannt. Jeder der vier logischen Farben kann eine der 16 aktuellen Farben zugeordnet werden. Für unser Spiel färben wir die Minen mit Hilfe des VDU-Befehls grün (aktuelle Farbzahl 2), statt der normalerweise vorgegebenen Farbe Gelb.

Das Suchgerät und der Assistent erhalten ihre Startpositionen in der unteren linken und oberen rechten Ecke. Da wir die Positionen des Suchgerätes und des Assistenten später verändern wollen, verwenden wir die Variablen (xdet,ydet) für das Suchgerät und (xman,yman) für die Koordinaten des Assistenten.

```
2830 DEF PROCposition_chars
2840 COLOUR 1
2850 PRINTTAB (xdet,ydet);CHR$(225)
2860 PRINTTAB (xman,yman);CHR$(226)
2870 COLOUR 2
2880 ENDPROC
```

Die COLOUR-Befehle am Anfang und Ende der Prozedur bestimmen die logische Farbe der Zeichen. COLOUR 1 wählt die Farbe 1 (Rot) für das Suchgerät und den Assistenten sowie COLOUR 2 zur Darstellung der Minen in Grün. Nun müssen den Variablen xdet, ydet, xman und yman die entsprechenden Werte zugeordnet werden. Dies geschieht in einer anderen Prozedur, gemeinsam mit der Initialisierung weiterer Variablen.

```
2320 DEF PROCinitialise_variables
2330 xdet=2:ydet=25:xman=17:yman=1
2340 xstart=120:xfinish=1144
2350 zero$="000000"
2360 ENDPROC
```

Alle hier aufgeführten Prozeduren können mit Hilfe des folgenden kurzen Programms getestet werden:

```
5 MODE 5
10 COLOUR 2
20 PROCinitialise-variables
30 PROCdefine-characters
40 PROClay-mines(40)
50 PROCposition-chars
60 END
```

Im nächsten Teil dieses Kurses werden wir uns mit den Abfragen und Kontrollen der Bewegungen befassen.

Fachwörter von A bis Z

Greedy Method = Rabiate Methode
Programme lassen sich unter strategischen oder taktischen Gesichtspunkten entwickeln. Entweder werden vorübergehende Abweichungen von der geraden Linie aus taktischen Gründen toleriert, oder es wird stets der direkte Weg zum Ziel eingehalten. Das letztgenannte Verfahren wird im englischen Sprachgebrauch als „greedy" bezeichnet, womit auch schon die Nachteile dargelegt werden: Unwirtschaftlichkeit und die Unfähigkeit zu differenzieren, nämlich den möglichen Gewinn hinter einem scheinbaren Verlust zu sehen. Ein Schachprogramm beispielsweise, das nur mit direkten Algorithmen arbeitet, verzichtet auf wesentliche Details wie Stellungsspiel und Opferzüge. Aber Schnelligkeit und Agressivität allein haben keine Chance gegen einen Partner, der die taktischen Winkelzüge beherrscht.

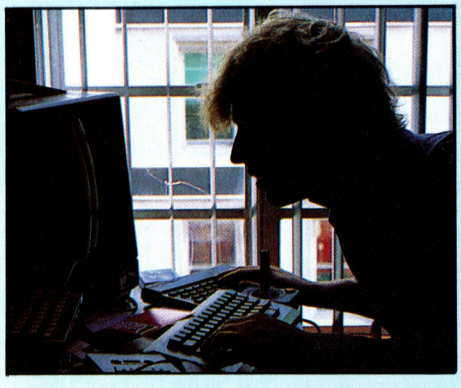
Grosch's Law = Groschs Gesetz
H. R. J. Grosch hat 1953 folgendes Gesetz für die Abhängigkeit der Leistungsfähigkeit eines Computers vom Preis verfaßt:

Leistung = const. * $(\text{Preis})^2$

In den fünfziger Jahren wurde diese Formel viel zitiert. Ein Computersystem, das dreimal soviel kostet wie ein anderes, müßte danach neunmal soviel leisten. Inzwischen hat die hochintegrierte Schaltungstechnik Groschs Gesetz jedoch vollständig unterminiert.

Hacker = Hacker
Ein Hacker ist im Fachjargon ein computerbegeisterter Mensch, der viel Zeit und Mühe investiert, um Schutzroutinen, Betriebssysteme und (via Telefon) fremde Computersysteme zu knacken. Die Hacker neigen zu einem großzügigen Umgang mit besitzrechtlichen Fragen, in einigen Fällen führt das bis zur Computerkriminalität. Die große Mehrheit beschränkt sich aber darauf, die Grenzen des eigenen Könnens bzw. des eigenen Rechnersystems zu erkunden. Ins Licht der Öffentlichkeit rückten die Hacker unter anderem durch den Film „War Games", in

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.



Die Ingenieurstudenten vom Massachusetts Institute of Technology machen sich angeblich alljährlich den Spaß, im Kontrollsystem eines Bostoner Bürohochhauses herumzuhackern, so daß dort das Licht stundenlang an- und ausgeht. Aber es bleibt nicht immer bei harmlosen Spielchen, und die Computerkriminalität steigt ständig.

dem ein einfallsreicher Amateur durch Einbruch in das Rechnernetz der US-Abwehr um ein Haar den dritten Weltkrieg ausgelöst hätte.

Half Duplex = Halbduplex
Ein Beispiel für den Halbduplex-Betrieb sind die Funkanlagen in Taxis und Einsatzfahrzeugen: Die Übertragung erfolgt in beiden Richtungen auf dem gleichen Kanal, der abwechselnd freigegeben wird.

Hamming Codes = Hamming Codes
Bei der Fernübertragung von Daten besteht immer die Gefahr, daß sich Fehler einschleichen. Deshalb haben die Informatiker eine ganze Reihe von Fehlersuch- und Fehlerkorrekturverfahren entwickelt. Für die Korrektur einzelner Bitfehler in binären Informationsblöcken stellen die „Hamming Codes" (um 1950 von R. W. Hamming bei der Firma Bell Telephone entwickelt) eine perfekte Lösung dar.

Angenommen, Sie wollen den Datenblock 0111 übertragen. Im Hamming-Code werden drei weitere Bits hinzugefügt, die so definiert sind, daß bestimmte Viererkombinationen aus den insgesamt sieben Bits immer eine gerade Anzahl von Einsen enthalten. Im obigen Beispiel müßte der Hamming-Code 100 lauten, der ganze Sieben-Bit-Block somit 0111100. Die Hamming-Kombinationen sind:

Datenwort 0111100		
Kombinationen	**Anzahl v. Einsen**	**Log. Resultat**
0.1.1.0	gerade	0
.11..00	gerade	0
...1100	gerade	0

Das logische Gesamtergebnis ist „000", gleichbedeutend mit „fehlerfrei". Stellen Sie sich nun vor, bei der Übertragung sei das dritte Bit von links „gekippt":

Datenwort 0101100		
Kombinationen	**Anzahl v. Einsen**	**Log. Resultat**
0.0.1.0	ungerade	1
.10..00	ungerade	1
...1100	gerade	0

Das logische Ergebnis 011 (binär) = 3 (dezimal) läßt erkennen, daß das dritte Bit von links umgefallen ist. Diesen Fehler kann die Empfangsstation automatisch korrigieren. Das Verfahren versagt nur, wenn mehr als eins von den sieben Bits fehlerhaft ist.

Bildnachweise
1317, 1340, 1341: Chris Stevens
1318, 1319: Kevin Jones, Caroline Clayton
1320, 1321, 1335, U3: Ian McKinnell
1323, 1324, 1328, 1332, 1333, 1339: Kevin Jones
1325, 1326, 1327: Liz Dixon
1330: Graphic Products, London
1331: Magnenat-Thalmann, Thalmann – University of Montreal, Prince – Genigraphics
1336: BBC Hulton Picture Library
1337: Uta Brandl
1342, 1344: Janet Barrance

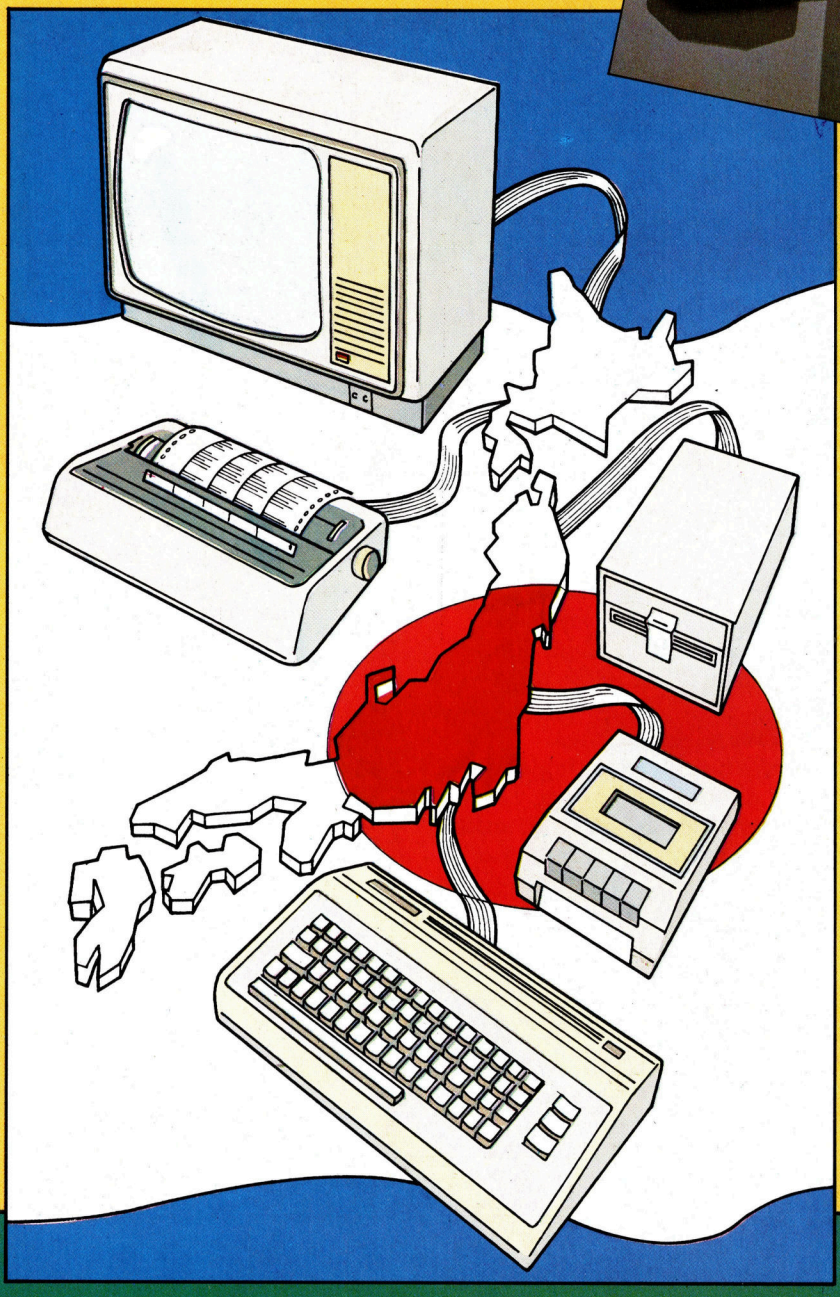
+ Vorschau +++ Vorschau +++ Vorschau

computer kurs

Heft 49

Atari 520 ST

Obwohl die letzte Zeit für die Computer-Hersteller nicht gerade ermutigend war, präsentierte Atari ein Gerät, das für Aufsehen sorgte: Für knappe 3500 Mark gibt es den 520 ST, der mit dem Motorola-68000-Prozessor ausgestattet ist, einem modernen 16-Biter.



Motoren-Einsatz

Jeder Gleichstrommotor kann über den Computer angesteuert werden. Besser ist der Einsatz von Schrittmotoren.

Nebenwirkungen

PROLOG bringt diesmal die „nicht-logischen" Eigenschaften wie „cut" und „fail".

Raketenmann

„Jet Pac" von Ultimate wurde ein Spiele-Hit für den Sinclair Spectrum. Auch die Commodore-Version hat viele Freunde gefunden.

Schwierigkeiten bereitet jedem Heimcomputer-Besitzer die fehlende Kompatibilität – es ist kaum möglich, von einem Rechner zum anderen zu wechseln und dabei die vorhandenen Peripherien sowie die Software zu nutzen. Japans Industrie brachte deshalb den MSX-Standard heraus: So soll erreicht werden, daß Hard- und Software generell austauschbar sind.