Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 44

6809 steuern

Commodore-Maus

Über das Suchen

Schneider 664

Ein wöchentliches Sammelwerk

computer kurs Heft 44

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

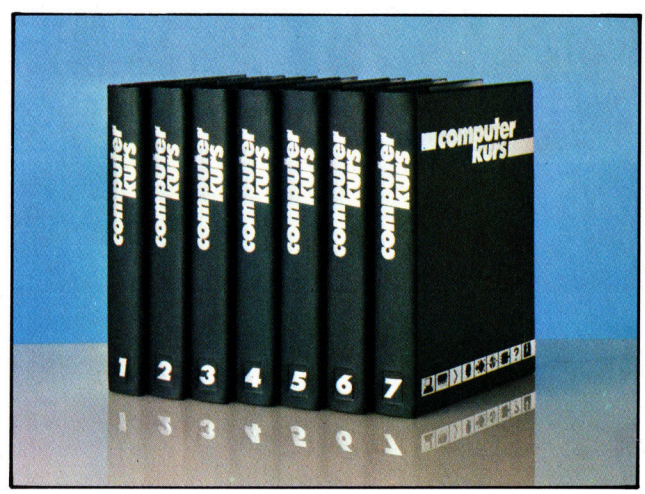
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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.

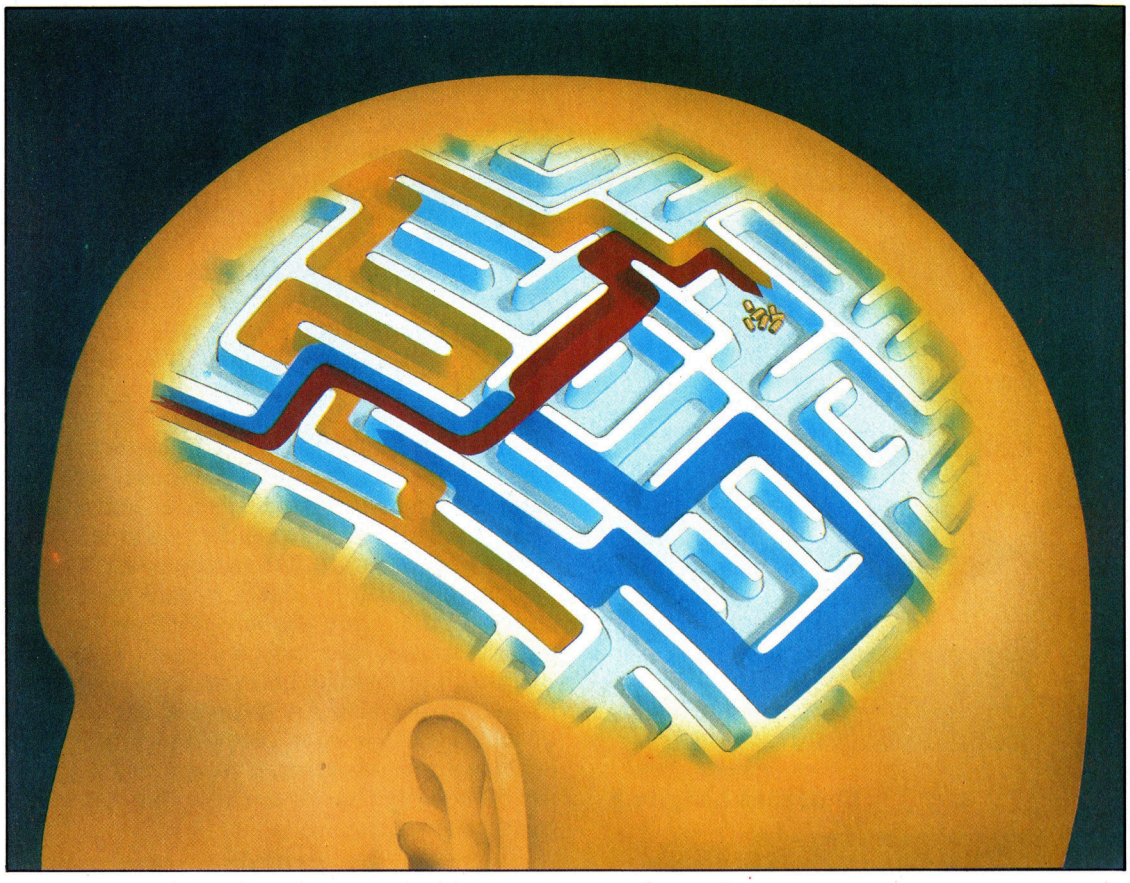
Redaktion: Winfried Schmidt (verantw. f. d. Inhalt), Elke Leibinger, Susanne Brandt, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

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



© APSIF, Copenhagen, 1982, 1983; © Orbis Publishing Ltd., 1982, 1983; © Marshall Cavendish Ltd., 1984, 1985;

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



Das Diagramm zeigt drei mögliche Wege durch ein Labyrinth unter Verwendung drei verschiedener Suchstrategien: Zufall, Systematik und Heuristik. In diesem Beispiel führt die heuristische Methode schneller zum Ziel als die beiden anderen, obwohl es möglich wäre, ein Labyrinth zu konstruieren, in dem das nicht unbedingt der Fall sein müßte. In der Praxis werden Labyrinthlösungen am besten durch Kombination heuristischer und systematischer Suchmethoden gefunden.

Heuristisch

Systematisch

Zufällig

Auf der Suche

Die umfassende Untersuchung des „Such"-Konzepts zur Lösung von Problemen ist zu einem wichtigen Bestandteil der Forschung in der „Künstlichen Intelligenz" geworden. Welche Vorstellungen bestehen in der Wissenschaft über die unterschiedlichen Suchtechniken?

Stellen wir uns vor, drei Ratten befänden sich in einem Labyrinth, in dem sich irgendwo eine Schale voller „köstlicher" Rattennahrung befindet. Eine Ratte streunt minutenlang im Labyrinth herum und schläft dann erschöpft ein. Der Untersuchungsleiter hatte dem Trinkwasser des Tieres Alkohol zugesetzt. Die zweite Ratte geht methodischer vor: Sie hält ihre linke Vorderpfote an die Wand und folgt bei Abzweigungen jeweils dem links liegenden Weg. Gelangt sie in eine Sackgasse, kehrt sie um. Unter Umständen erreicht sie ihr Ziel, doch bis dahin hat die dritte Ratte das Futter gefressen.

Die dritte Ratte verfügt über einen sehr guten Geruchssinn. Sie schnüffelt an mehreren Abzweigungen und entscheidet sich für den Weg, von dem sie annimmt, daß er sie schneller zu dem lockenden Duft führt. Es ist natürlich möglich, ein Labyrinth zu gestalten, mit dem diese Ratte zum Narren gehalten wird, doch man wird wohl zugestehen, daß diese Art der Suchstrategie die intelligenteste ist.

Im Feld der Künstlichen Intelligenz ist Suchen ein Schlüsselkonzept. Ob man nun 200 Faden tief in der Karibik mit einem Mini-Unterseeboot nach versunkenen Schätzen taucht oder sich im Zug mit der Lösung eines Kreuzworträtsels befaßt – man sucht etwas. Die diesem Bild entsprechende Problemlösungsmethode hat sich bei der KI als nützlich erwiesen. Unsere drei Ratten stehen als Beispiele für drei unterschiedliche Suchstrategien:

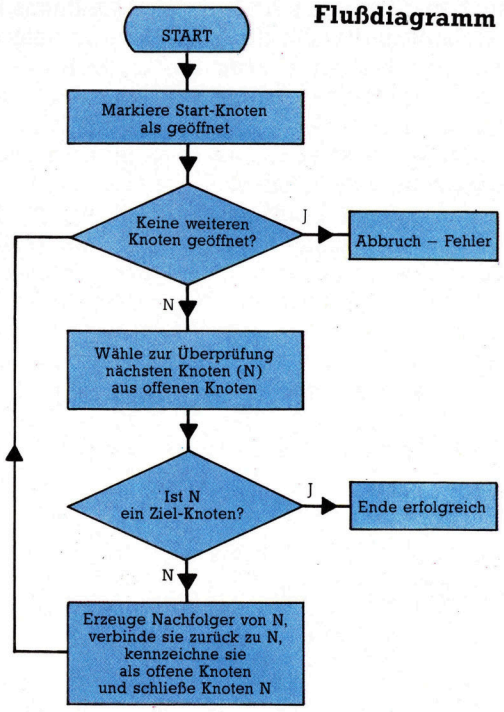
- Zufällige Suche (Torkeln eines stark Betrunkenen)
- Erschöpfende Suche (Systematisches Vorgehen)
- Heuristische Suche (Planvolles Vorgehen)

Die dritte Methode wird als intelligenteste bezeichnet, da sie mit weniger Mühe zum Ziel bzw. zur Lösung führt. Alle heuristischen Methoden setzen voraus, daß man anhand irgendwelcher Kriterien weiß, daß man sich dem Ziel nähert. Ohne dieses Wissen gibt es keine Intelligenz; man kann nur auf das systematische Vorgehen zurückgreifen.

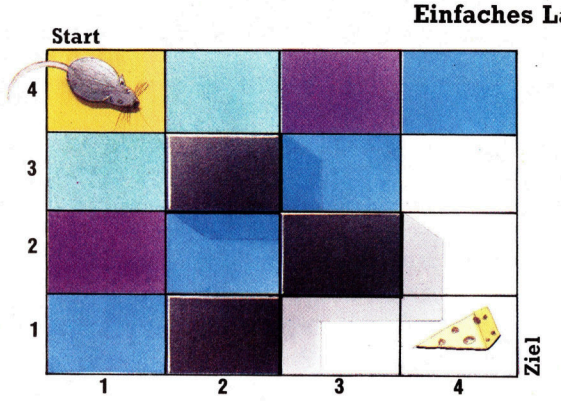
Methodische Suche

Bei den meisten KI-Suchmethoden bedient man sich der Darstellung von Bäumen, die auf dem Labyrinth-Modell basieren. Der hier gezeigte Baum wurde nach dem Labyrinth erzeugt, indem der Ausgangspunkt als Wurzelknoten betrachtet wurde. Die daraus resultierenden Abzweigungen wachsen durch Berücksichtigung aller in Frage kommenden Knoten. Zu beachten ist dabei, daß diagonale Bewegungen nicht erlaubt sind. Das Flußdiagramm stellt den Basis-Algorithmus für die Erzeugung der Baumstruktur dar.

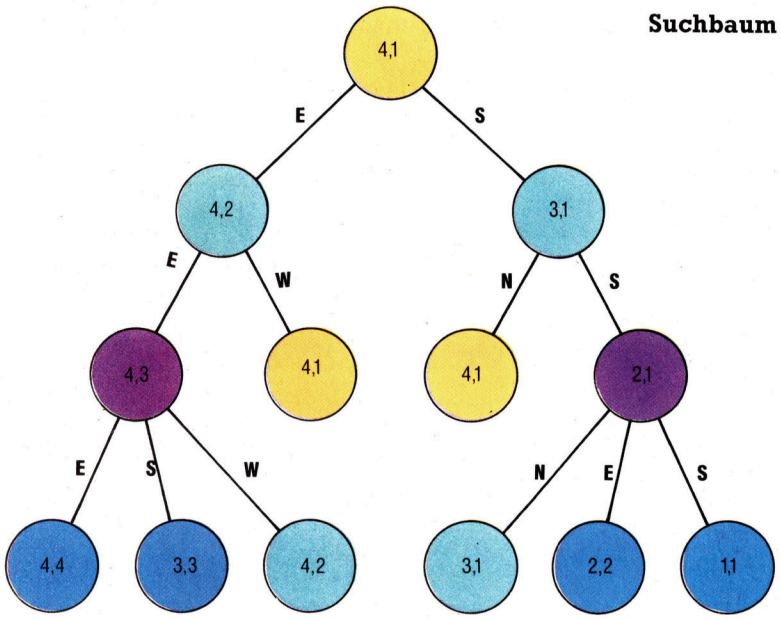
Flußdiagramm



Einfaches Labyrinth



Suchbaum



Den Weg durch ein Labyrinth zu finden ist ein typisches Suchproblem. Für die Beurteilung der Suchmethoden-Qualität gibt es zwei wichtige Fragen:

- Wie lange dauert es, einen Weg zu finden?
- Ist es der wirtschaftlichste Weg?

Ideal ist natürlich die Methode, die eine optimale Lösung (nicht die beste) in kürzest möglicher Zeit bringt. Die in der KI am häufigsten verwendeten Suchstrategien folgen einem Schema, bei dem die Gewichtung mehr oder weniger stark auf der Durchführungszeit liegt. Das abgebildete Algorithmus-Diagramm stellt eine Gruppe von Suchmethoden dar, aus denen eine bestimmte ausgewählt werden kann, indem man sich dafür entscheidet, wie die zentrale „Schachtel" gefüllt werden soll – oder aber, indem man entscheidet, welcher „Knoten" als nächster zu untersuchen ist. Die Bezeichnungen „offen" oder „geschlossen" beziehen sich auf den augenblicklichen Zustand jedes Knotens, der sich auf dem Weg vom Ausgangsort zum Ziel befindet. Die geöffneten Knoten stellen eine Art Warteliste dar, und die geschlossenen wurden bereits untersucht. Der Schlüssel zur effektivsten Suchweise liegt in der Reihenfolge, in der die Listenelemente abgearbeitet werden.

Von der Maus zum Käse

In dem einfachen Vier-mal-vier-Labyrinth sehen Sie oben links eine Maus, unten rechts den Käse. In jedem Rechteck kann die Maus zwischen vier Bewegungsrichtungen wählen (Nord, Süd, Ost oder West), obwohl einige nicht gangbar sind. Zeichnen wir die der Maus zugängigen offenen Wege auf, ergibt sich eine baumähnliche Struktur des Suchprozesses.

Aus dem Startrechteck (4,1) kann die Maus sich nur ostwärts nach (4,2) oder südwärts nach (3,1) begeben. Von (4,2) kann sie sich ostwärts oder westwärts und von (3,1) entweder nach Norden oder nach Süden bewegen. Geht die Maus nach Osten, dann nach Westen (oder nach Süden und dann nach Norden), wird sie zum Ausgangspunkt zurückkehren, was nicht sehr intelligent ist.

Eine systematische Suchmethode wird als „Breitensuche" bezeichnet. Hierbei erfolgt eine Untersuchung der der Ausgangsposition am nächsten befindlichen Knoten. Alle N-Züge werden dabei berücksichtigt (auf der Ebene N im Baum), bevor eine Bewegung N + 1 stattfindet. Es geht darum, eine Strecke zu finden, die die wenigsten Schritte beansprucht.

Will man dieses Verfahren verbessern, sind heuristische Informationen über die Entfernung des Ziels erforderlich. Dazu könnten wir die Straßenanordnung von Manhattan zugrunde legen: Auf Manhattan Island biegen die meisten Straßen rechtwinklig ab. Um von A nach B zu gelangen, muß man soundsoviel Blocks nach Norden oder Süden und soundso-

viel Blocks nach Osten oder Westen gehen. Ähnlich kann eine intelligente Maus im Labyrinth berechnen, wie viele Rechtecke das Ziel entfernt ist.

Durch das Wissen, wann eine Suche sich der Lösung nähert, kann die Suche beschleunigt werden. Die Ratte aus unserem einleitenden Beispiel folgt dabei einer einfachen Regel: Bewege dich immer näher aufs Ziel zu. Das ist die sogenannte „Bergsteiger-Strategie". – So benannt, weil es mit dem Finden eines Berggipfels im Nebel verglichen werden kann, an den man immer gelangt, wenn man bergauf geht. Das kann zwar ein schnellerer Weg als der der Breitensuche sein, garantiert aber nicht das Finden des optimalen Weges.

Eine bessere Methode ist der Kompromiß zwischen Breitensuche und einfachem Gipfelsuchen. Er wird als A* (Stern-)Algorithmus bezeichnet und wählt den nächsten zu untersuchenden Knoten auf der Basis von HD + SF, wobei HD die heuristische Schätzung der verbleibenden Entfernung und SF die bisher zurückgelegte Strecke darstellt. Je besser die Schätzung von HD, desto effektiver die Suche.

In unserem Beispielprogramm verwenden wir ein Programmraster, um alle drei Methoden einzusetzen, wobei nur geringfügige Modifikationen stattfinden. Die einzigen Unterschiede werden in der Wahl des Knotens deutlich, der als nächster untersucht werden soll:
Breitensuche – suche den niedrigsten SF
Gipfelsuche – suche den niedrigsten HD
A*-Algorithmus – suche niedrigsten HD + SF

Maze Search Program

```
 1000 REM ******************************
 1010 REM ** Listing 2.1 :            **
 1020 REM ** MAZE SEARCH PROGRAM      **
 1030 REM ******************************
 1040 MODE7
 1050 MH=17: MW=25: REM Maze Height & Width
 1060 SI=256 : REM tree limits.
 1070 WA=1: RR=2: FO=3: DN=4: BL=5
 1080 W1=1 : REM weight for SF
 1090 W2=2 : REM weight for HD
 1100 DIM M(MH+1,MW+1) : REM the maze
 1110 DIM C$(5) : REM maze characters
 1120 DIM P(SI),S(SI),N(SI),H(SI)
 1130 REM Path,Steps,Node,Heuristic-distance.
 1140
 1150 REM -- Rat in the Maze:
 1160 GOSUB 1360 : REM make the maze
 1170 NC=0 : REM no. of nodes examined
 1180 K=0 : REM counter
 1190 GOSUB 1660 : REM clear all paths
 1200 N(1)=2*MW+2 : REM 1st open node
 1210 S(1)=0: H(1)=FR-1 + (FC-1)
 1220 P(1)=0: REM no predecessor
 1230 REM -- Main loop:
 1240 REM **** MAIN LOOP ****
 1250 GOSUB 1770 : REM pick next node S
 1260 NC=NC+1
 1270 PRINT TAB(0,22);NC,SR,SC,H(S);"          "
 1280 GOSUB 1880 : REM generate successors
 1290 IF (FR<>SR OR FC<>SC) AND NC<300 THEN 1240
 1300 PRINT TAB(0,22);"Search finished!"
 1310 IF FR=SR AND FC=SC THEN GOSUB 2290 : REM ret
race steps
 1320 IF NC>300 THEN PRINT "Failed!"
 1330 PRINT NC;" nodes examined."
 1340 END
 1350 :
 1360 REM -- Routine to create maze:
 1370 FOR P=1 TO MH+1
 1380 FOR R=1 TO MW+1
 1390 IF RND(1) < 0.28 THEN M(P,R)=WA ELSE M(P,R)=
BL
 1400 IF P=3 OR R=3 THEN M(P,R)=BL
 1410 IF P=1 OR R=1 THEN M(P,R)=WA
 1420 IF P>MH OR R>MW THEN M(P,R)=WA
 1430 NEXT: NEXT
 1440 FR=2+INT(RND(1)*(MH-1)): REM food row
 1450 FC=4+INT(RND(1)*MW-3): REM food col
 1460 M(FR,FC)=FO
 1470 M(2,2)=RR : REM robot rat
 1480 C$(BL)=" "
 1490 C$(WA)=CHR$255 : REM blob
 1500 C$(DN)=" "
 1510 C$(RR)="R"
 1520 C$(FO)="f"
 1530 GOSUB 1560 : REM display it
 1540 RETURN
 1550 :
 1560 REM -- Maze display routine:
 1570 CLS
 1580 FOR R=1 TO MH+1
 1590 FOR C=1 TO MW+1
 1600 PRINT TAB(C,R); C$(M(R,C));
 1610 NEXT
 1620 PRINT
 1630 NEXT
 1640 RETURN
 1650 :
 1660 REM -- Tree-clearing routine:
 1670 DD=9999 : REM dead
 1680 FOR Q=1 TO SI
 1690 P(Q)=0
 1700 S(Q)=DD
 1710 N(Q)=0
 1720 H(Q)=DD
 1730 NEXT
 1740 NN=2 : REM next free node
 1750 RETURN
 1760 :
 1770 REM -- Pick best node S:
 1780 S=1: BN=DD
 1790 FOR I=1 TO SI
 1800 V=S(I)*W1 + ABS(H(I))*W2
 1810 IF V<BN AND H(I)>=0 THEN S=I: BN=V
 1820 NEXT
 1830 IF S=1 THEN PRINT TAB(0,20);"Exploring..
 1840 SR=INT(N(S)/MW)
 1850 SC=N(S)-MW*SR
 1860 RETURN
 1870 :
 1880 REM -- Routine to generate successors:
 1890 IF H(S)=0 THEN RETURN : REM done.
 1900 REM -- North:
 1910 Y=SR-1 : X=SC
 1920 IF Y>1 THEN GOSUB 2090
 1930 REM -- East:
 1940 Y=SR : X=SC+1
 1950 IF X<=MW THEN GOSUB 2090
 1960 REM -- South:
 1970 Y=SR+1 : X=SC
 1980 IF Y<=MH THEN GOSUB 2090
 1990 REM -- West:
 2000 Y=SR : X=SC-1
 2010 IF X>1 THEN GOSUB 2090
 2020 REM -- also close node S:
 2030 H(S) = -H(S)
 2040 IF H(S)>0 THEN PRINT "Ugh!"
 2050 PRINT TAB(SC,SR);" ";
 2060 M(SR,SC)=DN
 2070 REM blanks cell on screen.
 2080 :
 2090 REM -- Routine to open 1 node:
 2100 IF M(Y,X)=DN THEN RETURN
 2110 IF M(Y,X)=WA THEN RETURN
 2120 REM -- 1st find free location:
 2130 NX=0
 2140 REM ** FIND LOCATION LOOP
 2150 IF S(NN)<>DD THEN NX=NX+1: NN=NN+1
 2160 IF NN>SI THEN NN=1
 2170 IF NX>SI THEN PRINT "Full up!": STOP
 2180 IFS(NN)<>DD THEN 2140
 2190 REM -- Now open it:
 2200 XY=X + Y*MW
 2210 N(NN)=XY
 2220 P(NN)=S
 2230 S(NN)=S(S)+1
 2240 H(NN)=ABS(Y-FR) + ABS(X-FC)
 2250 PRINT TAB(X,Y);"+";
 2260 REM shows it on screen.
 2270 RETURN
 2280 :
 2290 REM -- Path-retracing routine:
 2300 ST=S(S)
 2310 FOR Q=1 TO 10000:NEXT Q:GOSUB1560
 2320 PRINT TAB(FC,FR);C$(FO);
 2330 REM ** PRINT PATH **
 2340 S=P(S) : REM parent node
 2350 XY=N(S) : REM coords.
 2360 Y=INT(XY/MW)
 2370 X=XY-Y*MW
 2380 M(Y,X)=RR : REM rat's footprint!
 2390 PRINT TAB(X,Y);"*";
 2400 IF S>0 THEN 2330
 2410 PRINT TAB(2,2);C$(RR)
 2420 PRINT TAB(0,22);"Path of ";ST;" steps."
 2430 PRINT NC;" nodes closed."
 2440 RETURN
```

Das Labyrinth befindet sich in einem zweidimensionalen Feld M (,), und die Datenstrukturen für den Suchbaum befinden sich in den Feldern P (), S(), N () und H(). Das Programm kombiniert „bisherige Kosten" und „geschätzte Kosten bis zum Ziel". Diese Faktoren lassen sich durch die Werte W1 und W2 in den Zeilen 1080 und 1090 ändern. Mit den gelisteten Werten folgt das Programm der Bergsteiger-Strategie. Die in diesem Programm verwendete heuristische Methode basiert auf dem „Manhattan-Beispiel". Durch Entscheidung für die heuristische Methode (W2 > W1) wird die Suche beschleunigt.

BASIC-Dialekte

Das Programm ist für den Acorn B. Für den C 64 und den ZX Spectrum sind folgende Änderungen erforderlich.

Commodore 64:
Ersetzen Sie die Anweisungen PRINT TAB (X,Y);"MESSAGE" durch
P=X:Q=Y:GOSUB 3000 :PRINT"MESSAGE"
3000 REM ** TAB ROUTINE **
3010 PRINT CHR$(19);
3020 PRINT LEFT$ (DWS,Q); TAB(P);
3030 RETURN

1040 FOR I=1 TO 25: DWS=DWS+CHR$ (17): NEXT I
1490 C$(WA)= CHR$(102)
1570 PRINT CHR$(147)

Spectrum:
Ersetzen Sie die Anweisung
TAB(X,Y); durch ATX,Y; und Zeile 1490 durch
1490 LET C$(WA)= CHR$(143)
Löschen Sie Zeile 1040.

Der Amstrad CPC 664 ist eine Weiterentwicklung des beliebten CPC 464. Statt eines Cassettendecks besitzt diese Maschine eine integrierte 3-Zoll-Diskettenstation im Hitachi-Standard. Die Cursortasten wurden vergrößert und lassen sich somit leichter bedienen. Das BASIC-ROM enthält außer den DOS-Kommandos zehn zusätzliche Befehle.

Viel Leistung für wenig Geld

Als Nachfolger des beliebten Schneider CPC 464 entwickelte Amstrad den CPC 664 mit eingebauter Diskettenstation. Die Größe des Arbeitsspeichers muß jedoch erweitert werden, wenn das Gerät auch in den Markt für kleine Bürocomputer vordringen soll.

Der Schneider CPC 464 entwickelte sich schon kurz nach seiner Vorstellung im Jahre 1984 zum Bestseller. Obwohl er keine technischen Neuerungen enthielt, war er – trotz stagnierender Verkaufszahlen des Micromarktes – ein großer Erfolg. Nur ein Jahr später führte die Firma ein zweites Gerät ein – den Schneider CPC 664. Er ist in vieler Hinsicht mit dem 464 identisch, besitzt statt des Cassettenrecorders jedoch ein eingebautes Diskettenlaufwerk.

Die Tastatur des CPC 664 ähnelt der des CPC 464. Die Tasten des Zehnerblocks hat Amstrad mit f1, f2 etc. bezeichnet. Ansonsten wurde nur die Größe der Cursor-Steuertasten verändert, die sich nun leichter bedienen lassen.

Die integrierte 3-Zoll-Diskettenstation im Hitachi-Standard hat das Cassettendeck ersetzt. Das eingebaute Diskettenlaufwerk ist kleiner als die externe Version, da es seinen Strom über den Monitor erhält und kein zusätzliches Netzteil benötigt.

Der CPC 464 brauchte wegen seines eingebauten Recorders keine zusätzliche Cassettenbuchse. Damit die für den CPC 464 entwickelte Software auch auf dem 664 laufen kann, hat das neue Gerät einen Cassettenrecorderanschluß.

Der Schneider CPC 464 besitzt einen einzigen Erweiterungsbus, der nicht nur für den Anschluß einer Diskettenstation ausgelegt ist, sondern generell als Schnittstelle zu Peripheriegeräten dient. Der CPC 664 verfügt nicht nur über diese Peripherieschnittstelle, sondern auch über eine zusätzliche Anschlußbuchse für ein zweites Laufwerk. Besonders bei dem Einsatz von CP/M ist eine zweite Diskettenstation sinnvoll. Selbstverständlich können sie auch unter CP/M mit nur einem Laufwerk arbeiten, doch müssen Sie dann während eines Programmablaufs öfter die Disketten tauschen.

Bei der Konstruktion der neuen Maschine wurde auch das BASIC-ROM um neue Befehle erweitert. Ferner gibt es jetzt natürlich Betriebssystembefehle für die Diskettensteuerung, die mit denen des DDI ROMs der externen Laufwerke identisch sind. Viele der neuen Befehle erweitern den stattlichen Befehlssatz zur Grafiksteuerung. Der wichtigste ist vermutlich die

FILL-Anweisung, die seltsamerweise in der ersten BASIC-Version fehlte. FILL füllt einen Bereich mit einer vom Programmierer festgelegten Farbe.

Ein weiterer Grafikbefehl (MASK) kann gepunktete Linien erzeugen. Er hat das Format MASK i,p, wobei i eine Ganzzahl zwischen 0 und 255 darstellt und p bestimmt, ob der erste Punkt geplottet werden soll. Wie schon der Name sagt, ist der Befehl eine „Maske", die mit einem Variablenfeld bitweise eine logische Operation ausführt. So erzeugt MASK 1,p Punkte in Abständen von acht Pixeln, MASK 17,p dagegen im gleichen Rhythmus zwei Punkte, und mit MASK 255,p entsteht eine durchgehende Linie.

Zusätzliche Grafikbefehle

MASK läßt sich auch in Verbindung mit den Befehlen GRAPHICS PEN und GRAPHICS PAPER einsetzen. Beim Ziehen einer Linie ist normalerweise die Hintergrundfarbe nicht sichtbar. Da jedoch bei gepunkteten Linien der Hintergrund wichtig sein kann, kann er mit GRAPHICS PAPER auf jede beliebige Farbe gesetzt werden. Auf die gleiche Weise legt GRAPHICS PEN die Vordergrundfarbe für Linien und Punkte fest.

Mit dem FRAME-Befehl läßt sich die grafische Anzeige weiterhin verfeinern. Oft entsteht bei einer Grafikdarstellung ein Flackern auf dem Bildschirm, da das System versucht, mitten in der Rasterabtastung ein neues Bild aufzubauen. FRAME unterbricht nun den Ablauf des BASIC-Programms so lange, bis die Rasterabtastung wieder am oberen Bildschirmrand anfängt und die neue Grafik erscheinen kann. Der Befehl erzeugt daher eine ruhigere Anzeige.

Mit dem Befehl COPY CHR$ lassen sich auf dem Bildschirm dargestellte Daten schnell von einer Stelle zur anderen bewegen. Dies ist besonders für kommerzielle Anwendungen interessant, die beispielsweise Zahlentabellen oder Adressen von einem Fenster an eine andere Stelle übertragen müssen.

Wir haben hier nur einige Beispiele des neuen BASIC-Befehlssatzes aufgeführt. Obwohl das ROM mit der vorherigen BASIC-Version kompatibel sein soll, können einige für den CPC 464 entwickelte Pakete auf dem 664 nicht funktionieren. Aus Gründen der Kompatibilität enthält das BASIC daher einen Block mit ROM-Einsprungadressen. Um ihre Programme schneller zu machen, haben einige Softwarehersteller diesen Adreßblock umgangen und die Routinen direkt aufgerufen. Da in der neuen ROM-Version viele dieser Routinen andere Adressen haben, funktioniert ein Teil der 464-Software nicht mehr auf dem 664.

Der Schneider CPC 664 zielt auf den Markt für Heimgeräte und für kleine kommerzielle Computer. Gemessen an seinem geringen Preis bietet er hervorragende Möglichkeiten. Beim Einsatz für kommerzielle Zwecke gibt es jedoch noch einige Probleme. Zwar hat Amstrad das Betriebssystem CP/M übernommen, doch sind die mit diesem Computer eingesetzten 3-Zoll-Disketten zur Zeit noch wenig verbreitet – Software fehlt noch.

Einige Standard-CP/M-Pakete wie Wordstar oder dBase II laufen nicht auf dem Schneider-Computer, da das Gerät unter CP/M nur 39 KByte für Anwendungen zur Verfügung hat. Es wurde jedoch schon eine Erweiterungsplatine angekündigt, durch die die gesamte Palette der CP/M-Software zugänglich wird.

Inzwischen ist ein weiterer Schneider-Computer erhältlich, der speziell für den kommerziellen Bereich entwickelt wurde. Er trägt die Bezeichnung CPC 6128 und ist mit 128 KByte ausgestattet. Dieser neue Computer verfügt ebenfalls über ein integriertes Diskettenlaufwerk und wird mit einem Softwarepaket ausgeliefert, das folgende Programme umfaßt: CP/M 2.2, CP/M Plus, DR. LOGO sowie das GSX-Grafikprogramm.

Schneider CPC 664

SCHNITTSTELLEN

Erweiterungsschnittstelle, Anschlußmöglichkeit für ein zweites Laufwerk, Buchsen für Cassettendeck, Ein-/Ausgabe, Joysticks, Drucker, 12 V- und 5 V-Eingang und Monitor.

MITGELIEFERTE SOFTWARE

Mitgeliefert wird CP/M 2.2 und DR. LOGO.

HANDBÜCHER

Das Handbuch enthält eine ausführliche Dokumentation des Amsoft-BASIC, DR. LOGO und CP/M. Es wurde aus den Handbüchern des CPC 464 und der DDI-Diskettenstation zusammengestellt.

STÄRKEN

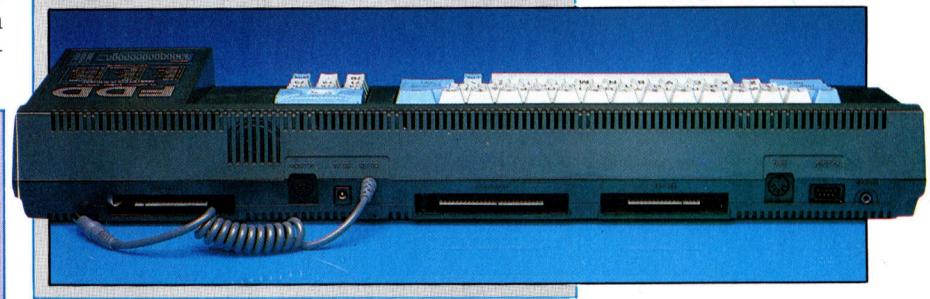
Der Schneider CPC 664 bietet viel Leistung für wenig Geld. Eine kleine kommerzielle Maschine dieser Art hätte vor nur wenigen Jahren noch ein Vielfaches gekostet.

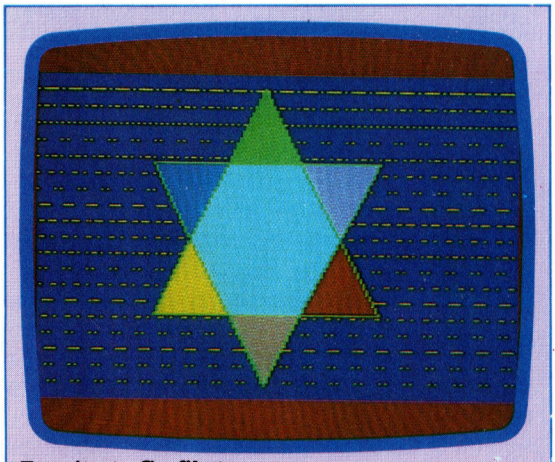
SCHWÄCHEN

Da dem Computer ausreichender Arbeitsspeicher fehlt, lassen sich die bekanntesten CP/M-Pakete nicht einsetzen. Die Einführung der angekündigten Erweiterungsplatine wird diesen Mangel jedoch beheben.

Verbindung zur Außenwelt

Auch bei den Schnittstellen gibt es einige Veränderungen. Rechts wurde eine Cassettenbuchse hinzugefügt, während sich an die Steckleiste links außen eine weitere Diskettenstation anschließen läßt. Damit steht die Erweiterungsschnittstelle für weitere Peripheriegeräte zur Verfügung.





Erweiterte Grafiksteuerung

Der Bildschirm zeigt drei neue Befehle des Schneider CPC 664. Die gepunkteten Linien im Hintergrund wurden mit dem Befehl MASK gezeichnet, die Spitzen des Sterns dagegen mit DRAW geplottet. Schließlich wurden die Flächen im Inneren des Sterns mit dem Befehl FILL eingefärbt.

Denksport

In diesem Artikel stellen wir Ihnen eine Art Gedächtnistest vor. Das Programm läßt eine von mal zu mal schwierigere Tonfolge erklingen, die Sie nachspielen müssen.

Die ersten Spiele, an die sich viele unter uns erinnern können, sind „Kettensätze". Jemand beginnt mit einem Wort wie zum Beispiel „Ich". Der Zweite sagt „Ich bin", der Dritte „Ich bin heute" usw. Vergißt ein Mitspieler ein Wort, so scheidet er aus. Die Aufgabe wird natürlich immer schwieriger, je länger der Satz wird und je weniger Sinn in der Wortfolge steckt.

Erstaunlicherweise ist gerade dieses Spiel unter Verwendung der Microelektronik in ein elektronisches Spiel umgesetzt worden. Der Prozessor steuert dabei eine Anzahl an Tasten, Lichtern und Tönen.

Bei der elektronischen Variante gibt das Gerät bestimmte Sequenzen von Tönen und Farben vor, die der Spieler durch Drücken der entsprechenden Tasten nachvollziehen muß. Danach wird die Sequenz um einen Ton erweitert und erneut abgespielt.

Die Umsetzung dieses Spieles auf den Computer mag ungewöhnlich erscheinen. Ein Spielzeug sollte schließlich transportabel und leicht zu benutzen sein. Doch selbst das beste Spiel wird einmal langweilig, und die Vielzahl an möglichen Programmen garantiert, daß der Computer nie uninteressant wird.

Unser Programm heißt „Follow That!" („Mach's nach") und basiert auf einem Spiel mit vier farbigen Lampen, jede mit ihrem eigenen Ton und einer Taste auf der Tastatur (numeriert von 1 bis 4). Das Programm läßt eine Lampe aufleuchten und erwartet, daß die richtige Taste gedrückt wird. Ist dies der Fall, werden zwei Lampen dargestellt usw.

Es gibt zwei Wege, aus dem Spiel auszuscheiden: wenn Sie zu lange für Ihre Antwort brauchen, oder wenn Sie im Verlauf des Spieles dreimal die falsche Taste drücken. Um das Spiel schwieriger zu machen, wird die Sequenz in dieser Version immer schneller dargestellt. Die Antwortzeit bleibt jedoch gleich. Eine Sequenz kann aus maximal 50 Lichtern bestehen.

Betrachten wir die Programmstruktur. Alle Farb- und Ton-Befehle sind in der folgenden Form am Programmende in Unterroutinen zusammengefaßt:

```
1000 Stelle Lampe Nummer a dar
1500 Erfasse Tastendruck 1, 2, 3 oder 4
2000 Generiere Geraeusch für Spielende
2500 Generiere Geraeusch als Warnung für
     falsche Antwort
6000 Stelle Meldung in Bildschirmzeile 20
     dar und warte, wenn noetig
```

Das Auslagern dieser Routinen aus dem Hauptprogramm hat zwei wesentliche Vorteile. Erstens werden computerspezifische Details wie Klang- und Farberzeugung zusammengefaßt, so daß eine Anpassung an andere Computer erleichtert wird. Zweitens besteht die Möglichkeit, die Routinen in einem ähnlichen Programm zu verwenden. Dasselbe Programm könnte beispielsweise verschiedene Variationen anbieten. Vielleicht wollen Sie einige selbstentwickelte Ideen integrieren, wie etwa die Darstellung mehrerer Bestergebnisse.

Mach's nach!

```
  10 REM Follow That! game
  30 LET h=0: LET n=0: LET w=3
  40 DIM c(4): DIM p(4): DIM a(50)
  50 LET p(1)=5: LET p(2)=8: LET p(3)=12: LET
 p(4)=15
  60 LET c(1)=1: LET c(2)=2: LET c(3)=4: LET
c(4)=6
  70 LET s$=" ": FOR i=1 TO 5: LET s$=s$+s$:
NEXT i
  80 LET b$=CHR$ (143): REM  a block
 100 REM *** Instructions page
 110 CLS : PRINT TAB (10);"Follow That!"
 120 PRINT : PRINT
 130 IF n>0 THEN  PRINT : PRINT "You managed
";n;" turns"
 140 IF n>h THEN  PRINT : PRINT "... A new re
cord !!!": LET h=n
 150 IF h>0 THEN  PRINT : PRINT "The best ye
t is ";h;" turns"
 155 PRINT : PRINT "Try to repeat the compute
r's    sequence of lights and sounds"
 160 PRINT "by pressing the keys 1 to 4"
 170 PRINT : PRINT "Press P to play, S to stop"
 180 LET a$=INKEY$: IF a$="" THEN  GO TO 180
 190 IF a$="s" OR a$="S" THEN  CLS : STOP
 200 IF a$<>"p" AND a$<>"P" THEN  GO TO 180
 205 REM  *** New game
 210 CLS : PRINT TAB (10);"Follow That!"
 220 FOR a=1 TO 4: GO SUB 1000: NEXT a
 230 LET n=0: LET m=0: RANDOMIZE
 240 REM *** Next turn
 250 LET n=n+1
 270 LET a(n)=INT (RND*4)+1
 280 IF m=w THEN GO SUB 2000: LET m$="*"+STR $(w)
+" wrong answers!": GO SUB 6000: GO TO 100
 285 LET m$="*Here it comes ...": GO SUB 6000
 290 FOR i=1 TO n
 300 LET a=a(i): GO SUB 1000
 310 FOR j=1 TO 100/n: NEXT j
 320 NEXT i
 330 LET m$="Follow that ...": GO SUB 6000
 340 LET i=1
 350 GO SUB 1500
 360 IF t=0 THEN  GO SUB 2000: LET m$="*Too s
low!": GO SUB 6000: GO TO 100
 370 IF a<>a(i) THEN  LET m=m+1: GO SUB 2500:
 GO TO 280
 380 LET i=i+1: IF i<=n THEN  GO TO 350
 390 IF n>50 THEN  LET m$="*You Win with 50 t
urns!": GO SUB 6000: GO TO 100
 400 LET m$="*Get ready to try again": GO SUB
 6000
 410 GO TO 250
1000 REM *** Light box a
1010 INK c(a)
1015 LET p=(a-1)*8+2
1020 FOR l=10 TO 14
1030 PRINT AT l,p;
1040 IF l=12 THEN  PRINT b$;b$;a;b$;b$;
1050 IF l<>12 THEN  PRINT b$;b$;b$;b$;b$;
1060 NEXT l
1070 BEEP 2/(n+1),p(a)
1080 PRINT AT 10,p;"      "
1090 PRINT AT 11,p;" ";b$;b$;b$;" ";
1100 PRINT AT 12,p;" ";b$;a;b$;" ";
1110 PRINT AT 13,p;" ";b$;b$;b$;" ";
1120 PRINT AT 14,p;"      "
1130 INK 0: RETURN
1500 REM *** Read a key
1510 LET t=250
1520 LET a$=INKEY$: IF a$="" THEN  LET t=t-1:
 IF t>0 THEN  GO TO 1520
1530 IF t=0 THEN  RETURN
1540 IF a$<>"1" AND a$<>"2" AND a$<>"3" AND a
$<>"4" THEN  GO TO 1520
1550 LET a=VAL (a$): GO SUB 1000
1560 RETURN
2000 REM *** Raspberry
2010 BEEP 3,0: RETURN
2500 REM Warning
2510 BEEP 1,0: RETURN
6000 REM *** Print m$
6010 PRINT AT 20,1;s$;AT 20,1;
6030 IF m$(1)="*" THEN  PRINT m$(2 TO ): FOR
z=1 TO 200: NEXT z
6040 IF m$(1)<>"*" THEN  PRINT m$
6050 RETURN
```

BASIC-Dialekte

Commodore 64

Ersetzen Sie folgende Befehle:
CLS durch PRINT CHR$(147),
LET A$=INKEY$ durch GET A$,
RANDOMIZE durch XX=RND(— TI),
(RND*4) durch (RND(1)*4),
M$(1) durch LEFT$(M$,1),
M$(2 TO) durch MID$(M$,2),
CHR$(143) durch CHR$(166);
PRINT AT L,C; durch PRINT
LEFT$(DN$,L+1)TAB(C); (Zeile 6010 wird zu 6010
PRINT LEFT$(DN$,21)TAB(1);S$;
LEFT$(DN$,21)TAB(1);
Ersetzen Sie DIM C(4) durch DIM C$(4),
in Zeile 1040 b$;b$;a;b$;b$; durch
BBZBB$ und
in Zeile 1100 b$;a;b$; durch BZB$;
Fügen Sie außerdem folgende Zeilen ein:

```
20 VL=54296:AD=54277:SR=AD+1:WF=AD-1:
NO=17:N1=NO:LF=AD-5:HF=LF+1
25 POKE AD,255:POKE SR,48:POKE VL,15
50 C$(1)=CHR$(31):C$(2)=CHR$(28):C$(3)=
CHR$(30):C$(4)=CHR$(158)
60 P(1)=51:P(2)=34:P(3)=64:P(4)=38
90 DN$=CHR$(17):FOR K=1 TO 5:
DN$=DN$+DN$:NEXT K:
DN$=CHR$(19)+DN$
1010 PRINT C$(A);
1015 P=(A-1)*9+3:Z$=RIGHT$(STR$(A),1)
1130 PRINT CHR$(144):RETURN
2010 SD=15:SP=4:N1=33:GOSUB 7000:RETURN
2510 SD=10:SP=10:N1=33:GOSUB 7000:RETURN
7000 REM *** BEEP SD,SP
7010 POKE VL,15:POKE WF,N1
7020 POKE LF,SP:POKE HF,SP:
FOR DD=1 TO SD*50:NEXT DD
7030 POKE HF,0:POKE LF,0:N1=NO:RETURN
```

Acorn B

Ersetzen Sie AT Y,X durch TAB(X,Y). Zeile 6010
beispielsweise lautet nun:
6010 PRINT TAB(1,20);S$;TAB(1,20)
Ersetzen Sie INKEY$ durch INKEY$(0)
Fügen Sie ferner die folgenden Zeilen ein:

```
20 MODE 2
25 COLOUR 135:CLS
60 C(1)=1:C(2)=2:C(3)=4:C(4)=5
80 B$=CHR$(35)
1010 COLOUR C(A)
1015 P=(A-1)*4
1070 SOUND 1,-10,P(A),40/(N+1):FOR DE=1
TO 2000/N:NEXT
1130 COLOUR 0:RETURN
2010 SOUND 1,-15,2,40
2510 SOUND 1,-15,40,40
```

Vertikaler Start

Im Programmierer-Jargon werden spezielle Applikationen für Ärzte, Rechtsanwälte, Designer, Journalisten, Fotografen und Lieferanten als „vertikale Software" bezeichnet. Häufig aber können auch allgemeinere Programme für spezielle Bereiche benutzt oder den Bedürfnissen des Endverbrauchers angepaßt werden.

Viele Anwendungen für Personal Computer sind entstanden, indem bestehende Software den aktuellen Problemen angepaßt wurde. Andere wiederum sind der „Spezial"-Software zuzuordnen. Man bezeichnet diesen Bereich als „Vertikalen Markt", da die Produkte jeweils nur für eine bestimmte Anwendergruppe entwickelt werden, etwa für Ärzte, Apotheker oder Psychologen. In dieser Artikelserie beschäftigen wir uns mit einer Reihe von Programmen für diesen Bereich, die interessante neue Aspekte der Microcomputer-Anwendung aufzeigen. Die folgenden Beispiele stellen dar, für welche Problemlösungen Software des „Vertikalen Marktes" entwickelt wurde.

Eine Gruppe von Eltern heroinabhängiger Jugendlicher im Londoner West End verwendet Caxton Softwares „BrainStorm"-Programm zur Planung einer Aufklärungskampagne über ihre Aktivitäten bei Ärzten, Sozialarbeitern und Regierungsstellen bzw. Behörden. Ein Restaurator in Kent setzt ein Practicalc-Spreadsheet ein, um zu analysieren, was seine Kunden bestellen, damit er sein künftiges Angebot entsprechend gestalten kann. Eine Baumschule in Sussex benutzt dasselbe Programm – für den C 64 – zur Lagerhaltung und Steuerung seiner Importe, um die Energiekosten zu optimieren. Ein Chirurg in einem Londoner Hospital setzte das Programm „Superbase" (Precision Software) ein, um so die Forschung und Heilung bei Krebs-Fällen zu beschleunigen.

Apotheker setzen Computer ein, um die Etiketten für Medikamente auszudrucken. Der Küchen-Gestalter Alan Batton in Warrington, Lancashire, arbeitet mit einem Programm, um Herde, Kühlschränke und andere Kücheneinrichtungsgegenstände auf einem Bildschirmgrundriß zu verschieben. Dieses Verfahren hat sich als so sinnvoll erwiesen, daß er das Küchen-Programm inzwischen sogar seinen Konkurrenten anbietet.

Dies sind nur einige neue Antworten auf die häufig gestellte Frage des künftigen Computerbesitzers: „Was kann ich damit denn tun?" Man schätzt, daß der durchschnittliche Computerbesitzer kaum zehn Prozent der Maschinenkapazität nutzt – und diese Zahl liegt vielleicht sogar noch zu hoch.

Es gibt zwei Wege, die Möglichkeiten eines Systems voll auszuschöpfen. Der eine besteht darin, neue Anwendungsbereiche für Standard-Software zu finden. Mike Ford, Berufsfotograf in Sheffield, nutzt das Lagerhaltungsmodul seines „Anagram Integrated Accounts"-Programms, um sein Fotoarchiv zu verwalten. So erstellt er eine Übersicht der Abzüge und Dias, die er angeboten hat. Das ist preiswerter, als Programme zu kaufen, die speziell für diese Aufgabe entwickelt werden.

Bibel-Studium

Alternativ besteht die Möglichkeit, nach einem Programm zu suchen, das für einen speziellen Anwendungszweck geschaffen wurde. Angenommen, man wäre Theologiestudent, dessen Schreibtisch von Wälzern, Konkordanzen, Kommentaren, Bibel-Wörterbüchern und dergleichen überquillt. Für diesen Bereich wird „The Word" Prozessor von Bible Research System angeboten, lauffähig auf dem IBM und kompatiblen. Zum Programm gehört die vollständige King-James-Übersetzung nebst Suchmöglichkeiten für die Erstellung von Kreuzverweisen auf Diskette.

Sind einem alle Übersetzungen biblischen Materials suspekt, kann man das Original mit einem „The Greek Transliterator" benannten Programm überprüfen. Damit verfügt man über das griechische Äquivalent jedes englischen Wortes oder Satzes und kann so jeden englischen Textbereich darstellen. Auf dieser Basis hat man Vergleichsmöglichkeiten aller Übersetzungen eines Wortes. Ein solch elektronisches Bibelstudium ist natürlich nicht billig.

Ein anderes Programm scheint auf den ersten Blick völlig anders gelagert zu sein. Einem Ingenieur, der Abwasserkanäle und Siele zu planen hat, wird das Programm MIDUSS – das McMaster Interactive Design of Stormwater Systems – bei der Größenbestimmung von Röhren, Kanälen und Sickergruben behilflich sein, das zudem die grafische Darstellung von Hydrogrammen auf einem hochauflösenden Bildschirm erlaubt.

Diese Art von „Wasserleitungsprogrammen" scheinen es Programmierern besonders angetan zu haben. Der Husky „Handheld"-Computer ist die Seele des CAMIL (Computer Aided Manhole and Location-)Pro-

gramms, das mittlerweile fast alle Wasseraufsichtsbehörden in England verwenden. Die Vermesser können vor Ort Daten eingeben und sie mittels Telefon an Großrechner weiterleiten, die darauf basierend Kanalisationspläne ausdrucken. Das Programm verfügt über Zugriffsmöglichkeiten zum Beispiel „auf alle gemauerten Abzugskanäle, die vor 1900 gebaut wurden". Und davon gibt's eine Menge.

Handelsvertreter werden die Programmreihe „Travelling" für nützlich befinden, die auf einem anderen Hand-held – dem NEC PC-8201 – läuft. Neben dem Grundprogramm, dem „Travelling Writer" – einer Textverarbeitung für Berichterstellung nebst Adreß- und Datenverwaltung –, gibt es den „Travelling Project Manager", den „Travelling Appointment Manager", den „Travelling Sales Manager" und den „Travelling Expense Manager", Programme also, mit denen der Handelsreisende Termine, Projekte, Verkäufe und Kosten mit Computerhilfe in den Griff bekommt.

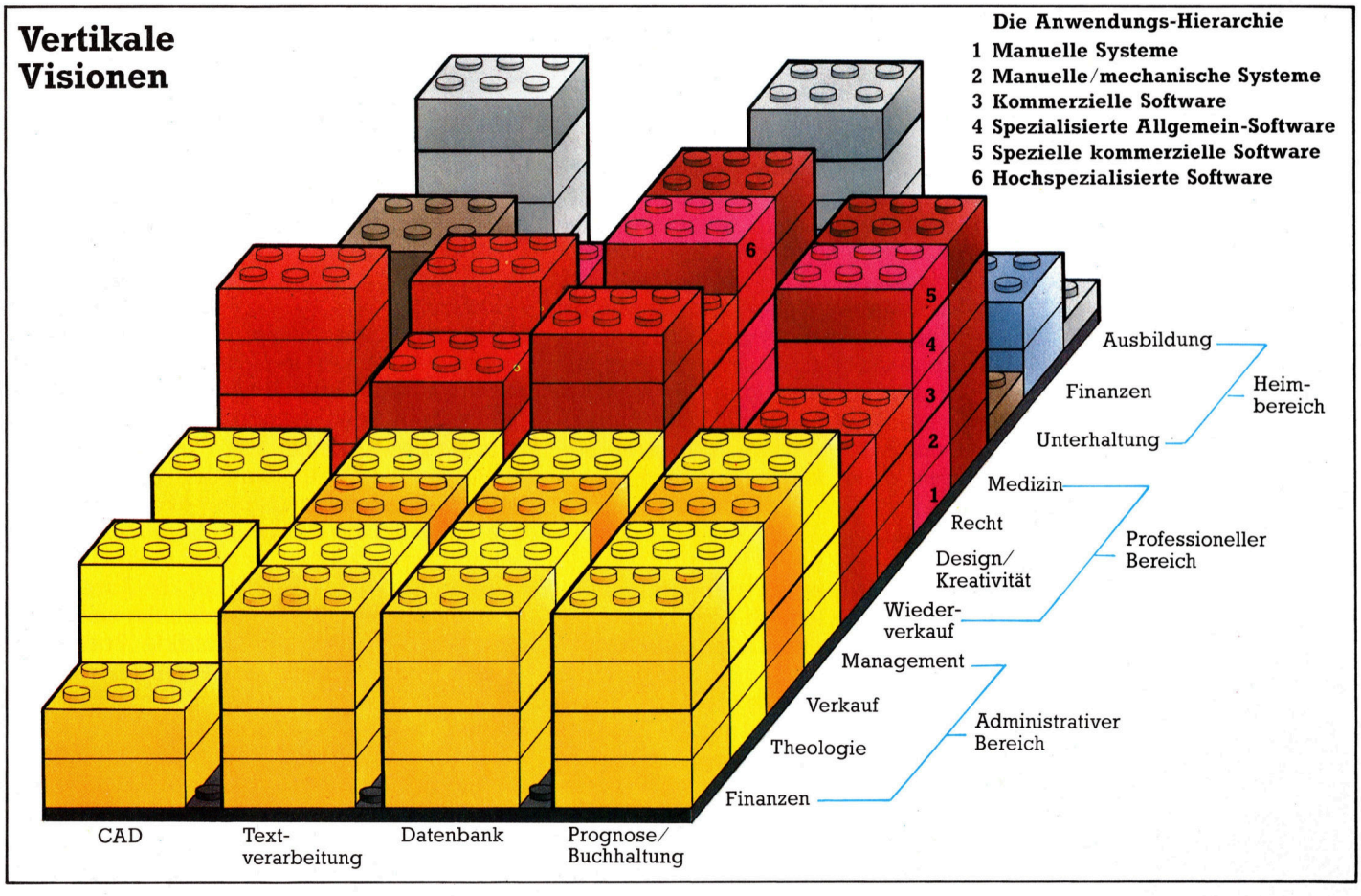
Vertreter können ihre Verkaufstechniken mit dem „Sales Edge"-Modul aus der „Human Edge"-Programmreihe verbessern. Sie werden von Thorn EMI für den IBM und Apricot angeboten. Das „Sales Edge"-Modul ermittelt die Verkaufsstärken und -schwächen des Benutzers auf der Basis einfacher Stimme-überein/Stimme-nicht-überein-Statements. Nach einer Reihe ähnlicher Fragen über den Kunden schlägt es eine Verkaufsstrategie vor, die alle Schachzüge (von der Gesprächseröffnung bis zum Abschluß) einer Verhandlung enthält. Auch das ist nicht billig.

Wer beruflich mit dem Geldwesen zu tun hat, wird möglicherweise bereit sein, 5000 Mark für „Forexias Forextend" zu investieren. Damit sind Studium und Analyse der Kursentwicklung von Dollar, Pfund, Schweizer Franken, japanischem Yen und der Deutschen Mark möglich. Das Programm bietet 37 verschiedene Darstellungen von Vergleichen, relativen Stärke-Indikatoren, Zinssätzen und handelsbedingten Veränderungen in der Zeit zwischen dem 1. Oktober 1983 und heute. Der Preis für das Programm mag hoch sein, ist aber weit weniger als das, was man bei Devisengeschäften aufgrund mangelhafter Information verlieren könnte.

Planungs- und Termin-Programme werden immer beliebter. Viele Computer verfügen über eingebaute Uhren. Die wenigsten jedoch unterbrechen die jeweils durchgeführte Tätigkeit, um Sie an einen wichtigen Termin zu erinnern. Dies aber kann der „Hewlett-Packard Schedule Planner", der für den HP-75C geliefert wird.

In der nächsten Folge dieser Serie sehen wir uns die Details des Programms „BrainStorm" an. Ferner werden wir uns mit weiteren speziellen Programmpaketen befassen.

Software wird geschrieben, um die Bedürfnisse bestimmter Benutzergruppen zu erfüllen. Das Gros der Software umfaßt deshalb Finanzverwaltungs-, Textverarbeitungs- und Datenbank-Programme. Von wachsender Bedeutung ist der CAD-Bereich (Computer Aided Design). Unser Diagramm zeigt diese Bereiche und ordnet sie bestimmten Nutzergruppen zu. Die vertikale Achse stellt den Komplexitätsgrad der Anwendung dar, wobei manuelle Methoden die Basis bilden, weiterführende Programme darüberliegen und maßgeschneiderte Software oder Spezial-Software sich auf der höchsten Ebene befindet.



Dynamische Daten

In dieser vorletzten Folge unserer PASCAL-Serie wird untersucht, wie sich mit Pointern, dem Heap und verketteten Strukturen Datenmengen von unbegrenzter Größe verarbeiten lassen.

Mit zwei Standardprozeduren (*new* und *dispose*) und einem speziellen Datentyp gewinnt PASCAL eine außerordentliche Flexibilität beim Einsatz dynamischer Speicherplatzverwaltung. „Pointer" sind Variablen, die statt Datenwerten (beispielsweise Ganzzahlen) „Zeiger" auf strukturierte oder unstrukturierte Datenobjekte enthalten. Zeiger werden wie Dateibuffer geschrieben (die auf den nächsten Datensatz der Datei zeigen, jedoch auf völlig andere Weise) und durch einen Aufwärtspfeil definiert. Dieser Pfeil muß vor einem Typennamen stehen, der dem Datentyp entspricht, auf den der Pointer zeigen soll:

```
TYPE
  GrossArray = ARRAY 1 . . 100 OF real;
  indirekt   = ↑ GrossArray;
```

ISO-PASCAL-Compiler verwenden dafür auch das Symbol (' @ '). Die Typendefinition reserviert eine Speicherstelle. Die Adresse eines großen Arrays wird dort allerdings erst gespeichert, wenn es mit der Prozedur *new* eingerichtet wurde. Bis zu diesem Zeitpunkt ist der Wert des Pointers – wie bei jeder anderen Variablen – undefiniert.

```
VAR
  Adresse : indirekt;
  Zahl    : integer;
```

reserviert den Speicherplatz für eine Maschinenadresse (16 Bits bei einem Acht-Bit-Rechner) und eine Ganzzahl. Beide Variablen wurden jedoch noch nicht auf einen bestimmten Wert initialisiert.

So wie wir Ganzzahlen vor dem Einsatz mit Null initialisieren, können wir auch Pointern den Wert *NIL* zuweisen. Das reservierte Wort NIL zeigt an, daß der Pointer gegenwärtig nicht benutzt wird. Sein Wert entspricht dem numerischen Wert Null. Die Variablen unseres Beispiels könnten daher folgendermaßen initialisiert werden:

```
Adresse := NIL;
Zahl    := 0;
```

Da (die Konstante) NIL eine Typenbezeichnung ist, wäre es eigentlich logisch, sie in einer Programmiersprache mit einem separaten Namen zu belegen.

Soll eine Pointer-Variable auf ein neues Datenelement zeigen, braucht nur die PASCAL-Prozedur *new* aufgerufen zu werden, die im Speicher dann den entsprechenden Platz reserviert und dessen Adresse in den Pointer schreibt. Wenn wir uns nun auf das Datenelement und nicht auf die Pointer-Adresse beziehen wollen, müssen wir den Pointer p „de-referenzieren" bzw. als p↑ schreiben. Beachten Sie, daß der Aufwärtspfeil nun wie bei dem Dateibuffer *hinter* dem Pointer-Namen steht und als „das Element, auf das p zeigt" bezeichnet wird.

Löschen des Pointers

Wenn Pointer-Daten, die mit *new* angelegt wurden, nicht mehr gebraucht werden, kann man die PASCAL-Prozedur *dispose* aufrufen, die das genaue Gegenteil von *new* bewirkt. Sie gibt den reservierten Speicherplatz wieder frei und weist dem Zeiger den Wert NIL zu. Das folgende Programm verdeutlicht diese Vorgänge. Aus Gründen der Einfachheit verwenden wir hier nur Ganzzahlen.

```
PROGRAM ZweiPlusZwei (output);
TYPE
  pointer = ↑integer;
VAR
  p1,
  p2        : pointer;
  Ergebnis : integer;
BEGIN
  new ( p1 );
  p1↑ := 2;
  new ( p2 );
  p2↑ := p1↑;
  Ergebnis := p1↑ + p2↑;
  WriteLn ( p1↑,'+',p2↑,'=',Ergebnis );
  dispose ( p1 );
  dispose ( p2 );
END.
```

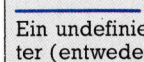
Diese etwas seltsame Art, zwei und zwei zu addieren, verdeutlicht zwei wichtige Punkte:

- Dynamische Variablen sind anonym – sie haben keinen Variablennamen (wie N), der irgendwo im Programm den Wert 2 enthält. Der Bezug auf die Datenelemente ist indirekt.
- Nach dem zweiten *dispose* existiert im Speicher nur noch die Ganzzahl *(Ergebnis)*; die dynamischen Daten gibt es nicht mehr.

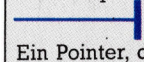
Wenn Sie unsere Serie über die Assemblersprache verfolgt haben, werden Sie sicher bemerkt haben, daß der indirekte Bezug der Pointer der indirekten Adressierung auf Maschinenebene entspricht. Zwischen diesen beiden indirekten Bezügen gibt es jedoch große Unterschiede. Zunächst sind in PASCAL nie die echten Adressen bekannt. Die einzige „absolute Adresse", die der PASCAL-Programmierer kennt, heißt NIL.

Auch können Sie in PASCAL den Speicher-

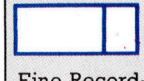
Pointer-Symbole



Ein undefinierter Pointer (entweder nach seiner Deklaration und vor einer Zuweisung, oder nach dispose).



Ein Pointer, der „nirgendwohin" zeigt (nach NIL-Zuweisung).



Eine Record-Einheit mit Datenfeld und Pointer.

platz beliebig oft belegen, freigeben und wiederverwenden, ohne je den Speicher „aufräumen" zu müssen. Da PASCAL die gesamte Speicherverwaltung übernimmt, interessiert nur, wieviel Speicherplatz noch frei ist. Diese Information läßt sich über die nicht standardmäßig vorgesehene Funktion *MemAvail* erfahren, – in einigen Versionen auch über die Funktion *Free*. Sie liefert die Anzahl der freien Speicherbytes. Auch die Funktion *SizeOf (Typenname)* ist sehr nützlich, da sie die Bytezahl des gewünschten Typs angibt.

PASCAL teilt den verfügbaren Arbeitsspeicher in zwei interne Strukturen – den Stack und den „Heap". Der Stack speichert beim Aufruf von Prozeduren und Funktionen die lokalen Daten, Rücksprungadressen und Ergebniswerte. Die dynamischen Daten werden dagegen dem Heap zugeordnet. Er arbeitet ähnlich wie der Stack, ist aber keine LIFO-Struktur (Last In, First Out). Der Heap fängt am anderen Ende des verfügbaren Arbeitsspeichers an und wächst in Richtung auf den Stack. Bei übermäßigem Gebrauch von *new* und/oder bei recursiven Programmen können Stack und Heap durchaus „zusammenstoßen". Dies läßt sich jedoch durch folgende Abfrage vermeiden:

```
IF SizeOf (Objekt) > Prozent * MemAvail
   THEN . .
```

Objekt ist der Typenname der Daten, die im Heap untergebracht werden sollen, und *Prozent* ein Wert im Bereich von 0,7, der dem Stack 30 Prozent des Speichers zur Verfügung stellt und dem Heap 70 Prozent. Wenn *MemAvail/Free* als „Hochwassermarkierung" eingesetzt wird, sollte auch eine Liste der nicht benötigten Pointer-Objekte angelegt werden.

Die eigentliche Stärke der Pointers zeigt sich erst beim Aufbau verketteter Strukturen. Für die String-Verarbeitung werden oft Arrays mit festen Längen (beispielsweise 80 Zeichen) eingesetzt. Ein in einem derartigen Array gespeichertes Schriftstück belegt jedoch auch für jede Leerzeile 80 Bytes.

PASCAL kann Schriftstücke mit variierenden Längen weitaus intelligenter speichern. Da PASCAL-Compiler zwischen Namen unterschiedlicher Länge unterscheiden können, ist hier die einfachste Struktur ein Record mit zwei Feldern: ein Feld für das Datenelement (in diesem Fall chars) und ein Pointer-Feld, das auf den nächsten Record der Liste zeigt.

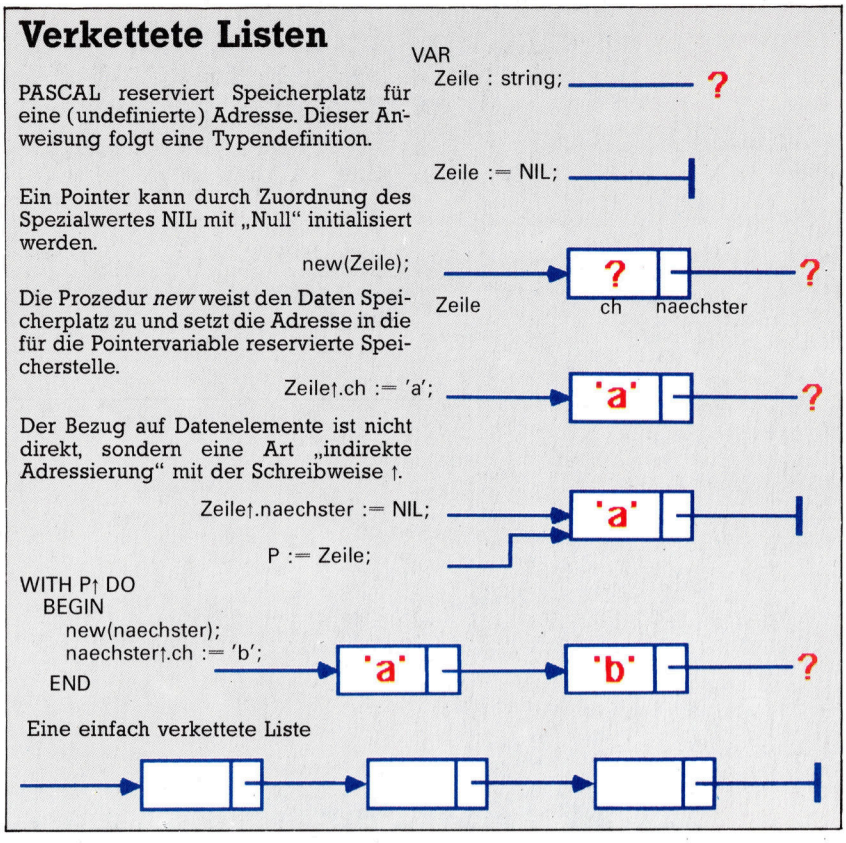
```
TYPE
   String   = ↑Zeichen;
   Zeichen = RECORD
                ch        : char;
                naechster : String
             END;
VAR
   Zeile : String;
BEGIN
   Zeile := NIL; etc.
```

Ein String mit der Länge Null wird durch die Zuordnung des Wertes NIL dargestellt. Jede andere Zeichenfolge erhält für die einzelnen Zeichen (char) einen neuen Record und einen *Pointer* auf den darauffolgenden Record. Da der Pointer des allerletzten Records auf NIL gesetzt wurde, läßt sich auch das Ende des Strings leicht feststellen. Mit dieser Prozedur lassen sich die Strings ausgeben:

```
WHILE Zeile <> NIL DO
   BEGIN
      write (Zeile↑.ch);
      Zeile := Zeile↑.naechster
   END
```

Beachten Sie, daß die Datenstruktur recursiv ist (sie definiert sich selbst). Da dem Pointerfeld kein vollständig definierter Typ zugeordnet werden kann, ist hier der Bezug auf den nächsten Record möglich.

Verkettete Listen



PASCAL bietet dem Programmierer die Möglichkeit, „verkettete Listen" einzusetzen. In diesen außerordentlich flexiblen Datenstrukturen zeigt jedes Datenelement auf das nächste Element der Liste. Die Listen können einfach verkettet sein (d. h. jedes Element zeigt auf das nächste) oder doppelt verkettet (d. h. die Pointer zeigen auf das nächste und das letzte Element der Liste) oder im Kreis verkettet (das letzte Element einer einfach verketteten Liste zeigt wieder auf das erste Element der Liste).

Das Programm Kreisliste

```
PROGRAM KreisListe ( input, output );
(* ---------------------------------------
- Dieses Programm setzt Records mit vermischten
- Daten in eine dynamische Kreisliste ein. Da
- die Daten in alphabetischer Reihenfolge nach
- dem Inhalt des Schlüsselfeldes (Name) einge-
- setzt werden, sind keine zusätzlichen Sortier-
- algorithmen notwendig.
---------------------------------------*)

CONST
  StringLaenge  = 25;
  Leerzeichen   = ' ';

TYPE
  Cardinal      = 0 . . MaxInt;
  StringGroesse = 1 . . StringLaenge;
  String        = PACKED ARRAY [ StringGroesse ]
                  OF char;
  Objekt        = RECORD
                    Name : String;
                    (* Platz fuer andere Felder *)
                    Schulden : Cardinal
                  END; (* Objekt *)
```

Mit diesem Programm können Sie Records mit gemischten Daten in eine dynamische Kreisliste einsetzen. Da die Daten nach einem alphabetisch geordneten Schlüsselfeld eingefügt werden, sind hier keine Sortieralgorithmen notwendig.

```
Pointer     = ↑Datensatz;
Datensatz   = RECORD
                Element : Objekt;
                naechstes : Pointer
              END; (* Datensatz *)
VAR
  Liste          : Pointer;
(* ------------------------1------------------------ *)
PROCEDURE Einlesen ( Kopf : Pointer );
VAR
  Daten :Objekt;
  OK    : boolean;
(* ------------------------2------------------------ *)
PROCEDURE LeerzeichenUeberspringen ( VAR F : Text );
  (* Leerzeichen am Anfang ignorieren *)
  VAR
    fertig : boolean;
  BEGIN
    fertig := EoF ( F );
      WHILE NOT fertig DO
        BEGIN
          fertig := ( F↑ > Leerzeichen ) OR EoLn ( F );

          IF NOT fertig THEN
            BEGIN
              get ( F );
              fertig := EoF ( F )
            END
        END
  END; (* LeerzeichenUeberspringen *)
(* ------------------------2------------------------ *)
PROCEDURE Zifferlesen ( VAR F   : Text;
                        VAR N   : Cardinal;
                        VAR OK  : boolean );
  (* eine nicht negative Zahl einlesen *)
TYPE
  einfach = 0 . . 9;

VAR
  Ziffern : SET OF char;
  fertig  : boolean;
(* ------------------------3------------------------ *)
  FUNCTION Wert ( Ziffer : char ) : einfach;
    (* char in numerischen Wert umwandeln *)
    BEGIN
      Wert := ord ( Ziffer ) - ord ( '0' )
    END; (* Wert *)
(* ------------------------3------------------------ *)
  FUNCTION Legal ( N      : Cardinal;
                   Ziffer : char ) : boolean;
  (* feststellen, ob groesser als MaxInt *)
  BEGIN
    IF N = MaxInt DIV 10
      THEN
        Legal := Wert ( Ziffer ) <= MaxInt MOD 10
      ELSE
        Legal := N < MaxInt DIV 10
  END; (* Legal *)
(* ------------------------3------------------------ *)
  BEGIN (* Zifferlesen *)

    Ziffern := [ '0' . . '9' ];
    OK      := NOT EoF ( F );

    IF OK THEN (* Wurde eine Ziffer gefunden ? *)
      OK := F↑ IN Stellen;

    N     := 0;
    fertig := NOT OK;

    WHILE OK AND NOT fertig DO
      BEGIN (* N auf Basis 10 aufbauen *)
        N := 10 * N + Wert ( F↑ );
        get ( F );
        fertig := EoF ( F );

        IF NOT fertig THEN
          fertig := NOT ( F↑ IN Ziffern );
        IF NOT fertig THEN
          OK := Legal ( N, F↑ )
      END;

    IF NOT EoF ( F ) THEN
      ReadLn ( F )

  END; (* Zifferlesen *)
(*------------------------2------------------------ *)
PROCEDURE ( ZeileLesen ( VAR S : String );
  (* Eine Zeile chars von der Tastatur lesen *)

  VAR
      Index    : 0 . . StringLaenge;
      Zeichen : char;
  BEGIN
    LeerzeichenUeberspringen ( input );
    Index := 0;

    WHILE NOT EoLn AND ( Index < StringLaenge ) DO
      BEGIN (* Ein char in den String setzen *)
        Index := succ (Index );
        read ( Zeichen );
        S [Index] := Zeichen
      END;

    IF Index < StringLaenge
      THEN (* mit Nullen auffuellen *)
        FOR Index := Index + 1 TO StringLaenge DO
          S [ Index ] := chr ( 0 )
      ELSE
        IF NOT EoLn THEN (* Zu viele chars *)
          WriteLn ( 'Achtung — Eingabe verkuerzt' );

    ReadLn

  END; (* ZeileLesen *)
(* ------------------------2------------------------ *)
PROCEDURE Einsetzen ( Liste   : Pointer;
                      Daten : Objekt );
  VAR
      Sucher,
      Verbindung : Pointer;
      Alpha      : String;

  BEGIN (* Daten in den Kopfsatz setzen *)
    Liste↑Element := Daten;
    Sucher        := Liste;
    Alpha         := Sucher↑.naechstes.Element.Name;

    WHILE Daten.Name < Alpha DO
      BEGIN (* Liste durchsuchen *)
        Sucher := Sucher↑.naechstes;
        Alpha  := Sucher↑.naechstes↑.Element.Name
      END;
          (* keine Ueberpruefung auf Doppelbelegung *)
    new (Verbindung); (* Weiteren Datensatz anlegen *)
    Verbindung ↑ . Element := Daten; (* Daten einfuegen *)
    Verbindung ↑ . naechstes := Sucher ↑. naechstes; (* und
      verbinden *)
    Sucher ↑. naechstes := Verbindung *( in der Liste *)
  END; (* Einsetzen *)
(* ------------------------2------------------------ *)
BEGIN (* Einlesen *)

  write ( 'Name ? ' );
  ZeileLesen ( Daten.Name );
    (* bei Leereintrag anhalten: *)
  WHILE Daten.Name [ 1 ]<> chr ( 0 ) DO
    BEGIN
      REPEAT
        REPEAT
          write ( 'Hoehe der Schulden ? ' : 20);

          IF EoLn ( input ) THEN
            ReadLn ( input );

        LeerzeichenUeberspringen ( input )
      UNTIL NOT EoLn ( input );

      Zifferlesen ( input, Daten.Schulden.OK );

      IF NOT OK THEN (* keine Ziffern *)
        WriteLn ( '*** FEHLER — ' : 20,
          'Bitte Neueingabe ***' );
      UNTIL OK;

      Einsetzen ( Kopf, Daten );
      WriteLn ( ' (Ende = RETURN)' : 40 );
      write ( 'Name ? ' );
      ZeileEinlesen ( Daten.Name )
    END
  END; (* Einlesen *)
(* ------------------------1------------------------ *)
PROCEDURE Anzeigen ( Liste : Pointer );
  VAR
      print : Pointer;
  BEGIN (* zuerst Kopfsatz ueberspringen *)
    print := Liste↑.naechstes;
    page ( output );
    WriteLn ( 'Gewuenschte Liste :' );
    WriteLn;
        (* noch mehr Daten zum Anzeigen ? *)
    WHILE NOT ( print = Liste ) DO
      BEGIN
        WITH print↑.Element DO
          WriteLn ( Schulden : 8, Name : 30 );
        WriteLn;
        print := print↑. naechstes
      END
  END; (* Anzeigen *)
(* ------------------------1------------------------ *)
BEGIN (* Kreisliste — Hauptprogramm *)
  new ( Liste ); (* Dummykopfsatz anlegen *)
  Liste↑.naechstes := Liste; (* auf sich selbst zeigen *)
  page ( output );
  WriteLn ( 'Daten eingeben: Erst Nachname, dann' );
  WriteLn ( 'die geschuldete Summe in (ganzen) Mark:' );
  WriteLn;
  Einlesen ( Liste ); (* Daten in die Liste schreiben *)
  Anzeigen (Liste ); (* In Reihenfolge ausgeben *)
  WriteLn ( '——— fertig ———' : 40 )
END.
```

Dschungelfieber

Ultimate machte sich mit hochwertigen Arcade-Action-Spielen für den ZX Spectrum einen Namen. In den neueren Veröffentlichungen sind grafische Perfektion, schnelle Action und Elemente strategischer Abenteuerspiele vereint. Ein Beispiel dafür ist das Spiel „Sabre Wulf".

Seit Arcadespiele nicht mehr so gefragt sind, wenden sich Software-Häuser jener neuen Spielform zu, die Elemente strategischer Action- und Abenteuerspiele miteinander vereint. Ultimates Sabre Wulf ist nach dieser Formel entstanden.

Ein Abenteuerspiel, in dem der Spieler den Helden – oft eine Figur aus der Science-Fiction- oder Fantasy-Literatur – durch unterschiedliche Umgebungen führen und dabei Aufgaben lösen und Rätsel klären muß, ist häufig langatmig. Die Aktion kann sogar für einen bestimmten Zeitpunkt unterbrochen werden, während der Spieler die Antwort auf ein scheinbar unlösbares Rätsel zu finden sucht. Ein Arcadespiel dagegen veranlaßt nicht zu längerem Nachdenken, sondern erfordert nur gutes Reaktionsvermögen und einen schnellen Finger am Feuerknopf.

Sabre Wulf versucht, die besten Elemente beider Spieltypen zu vereinen. Es ist im Grunde ein Labyrinthspiel, im Dschungelszenario angesiedelt und im weiteren Sinne dem Spielhallenklassiker „PacMan" verwandt. Der Held, Indiana-Jones ähnlich, muß seinen Weg durch ein sehr kompliziertes Labyrinth finden, hat Angreifern auszuweichen und muß Schätze sammeln, um Punkte zu bekommen. Ziel dieses Spiels ist es, vier Teile eines magischen Amuletts zu finden.

Die Dschungelszenerie ist hervorragend umgesetzt. Tiere, Pflanzen, Berge, Höhlen und Schätze sind detailliert dargestellt.

Die meisten Angreifer kann der Held mit einem schnellen Schlag des Schwerts ausschalten, wenn er ihnen im richtigen Augenblick und im rechten Winkel gegenübersteht. Doch einige der Gegner lassen sich nur mit speziellen Waffen abwehren, die bei der Reise durchs Labyrinth zu finden sind. Andere Gegenstände bringen Extra-Leben, was für den Spielablauf wichtig ist, da nur die besten Spieler lange unversehrt bleiben. Andere Gegenstände, etwa blühende Orchideen, können helfen oder stören. Abhängig von ihrer Farbe machen sie gegen Gefahr immun, verwandeln den Spieler in eine Pflanze, verdoppeln die Bewegungsgeschwindigkeit oder – und das verwirrt am meisten – bewirken eine Umkehrung der Spielsteuerung.

Generell kann man sagen, daß Sabre Wulf ein sehr gut gestaltetes Spiel ist, wenngleich es einige der üblichen Fehler aufweist, die auch andere Spielprogramme haben. Der anfangs sehr unterhaltsame Sound wird bald langweilig, und Ultimate hat im Programm leider keine Abstellmöglichkeit vorgesehen. Es wird zwar als Programm für wahlweise ein oder zwei Spieler bezeichnet, stellt sich tatsächlich aber als Solospiel mit zwei Punkteregistern dar. Enthalten ist die übliche Ultimate-„Ruhmeshalle", die Platz für sechs Höchstpunktzahlen bietet. Hier hat Ultimate sich für die in Spielhallen übliche Art der Namenseintragung entschieden: Die Initialen werden entweder durch Joystick oder über Tastatur eingegeben.

Ultimate hat bei der Entwicklung des Spiels berücksichtigt, daß es viele ergänzende Tastaturen für den Spectrum gibt. Trotzdem ist der Einsatz der Tastatur wenig befriedigend, da bei Sabre Wulf für Bewegung und Schwertkampf die Tasten Q, W, E, R und T benutzt werden. Das ist schwer zu lernen, da diese Tasten in einer Reihe liegen und deshalb die Bedeutung der Tasten leicht verwechselt werden kann.

Doch abgesehen davon ist Sabre Wulf ein hervorragendes Spiel, das wohlausgewogen Arcade-Action mit strategischen Abenteuer-Programmen paart.

Sabre Wulf: Für ZX Spectrum (48 K), Acorn B
Hersteller: Ashby Computers and Graphics Ltd., Ashby de la Zouch, Leicestershire LE6 5JU
Autoren: Ultimate
Joysticks: Kempston, Interface 2 und Cursorsteuerung
Format: Cassette

Der Bildschirm zeigt die Grafiken der Titelseite. Auf der Verpakkung des Programms ist diese Seite ebenfalls abgebildet. Sabre Wulf ist ein Labyrinthspiel mit einer geradezu genialen Vielfalt an hinterhältigen Charakteren. Die Umrandungen des Labyrinths sind mit Grafiken von herausragender Qualität dekoriert, die eine Dschungelszene darstellen.

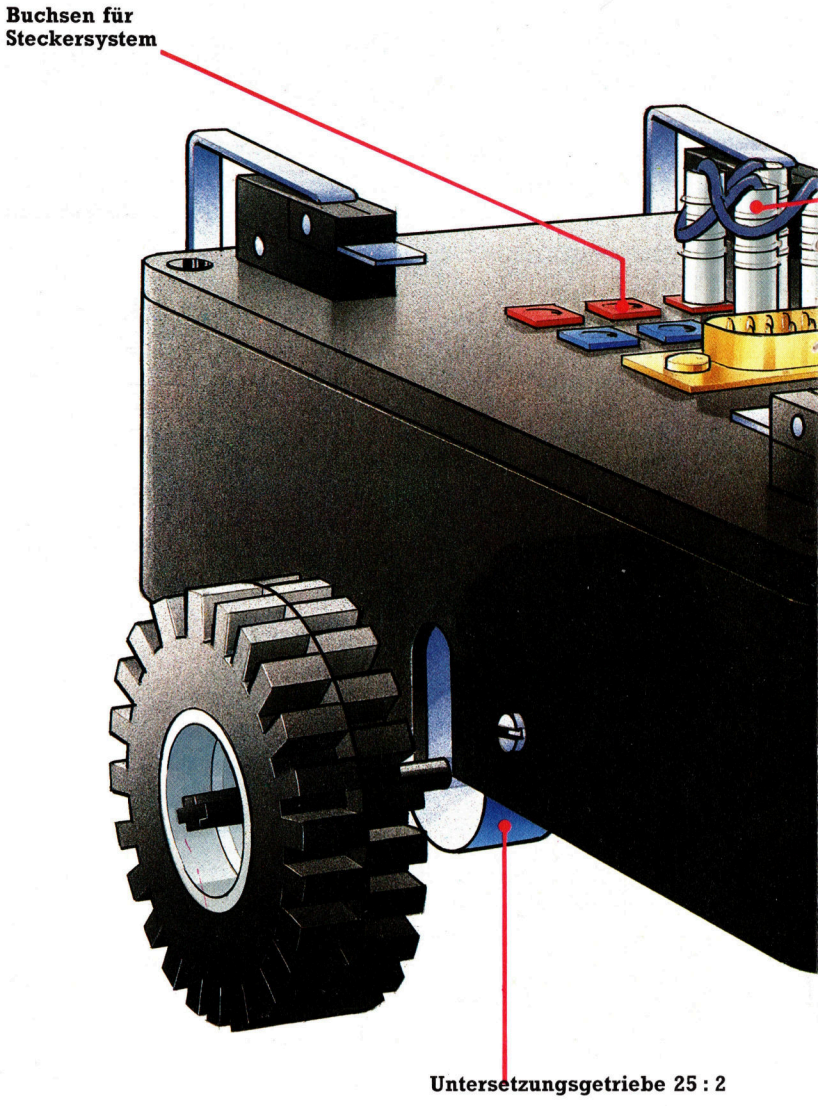
Neuer Anfang

Im Selbstbau-Kurs geht's mit einem neuen Projekt weiter voran. Ein Bodenroboter wird entwickelt, der sich exakt steuern läßt und sich durch Licht- und Berührungssensoren „intelligent" verhalten kann.

Das neue Projekt befaßt sich mit dem Bau eines mobilen Bodenroboters und dem Entwurf der dazugehörigen Software. Die beiden Räder des Roboters werden über Reduktionsgetriebe von Schrittmotoren bewegt. Diese ermöglichen eine genau definierte Bewegung in Schritten von 7,5 Grad, die in Verbindung mit den Untersetzungen bei Bedarf jedes Rad um nur 0,6 Grad weiterdrehen. Schrittmotoren eignen sich hervorragend für digitale Regelung, weil sie impulsgesteuert arbeiten – pro Impuls dreht sich der Anker nur um einen bestimmten Winkel. Die Ansteuerung soll über den User Port des Computers erfolgen. Damit der Roboter „intelligent" reagieren kann, ist der Einbau von Berührungs- und Lichtsensoren vorgesehen, mit denen er beispielsweise einer Linie folgen kann.

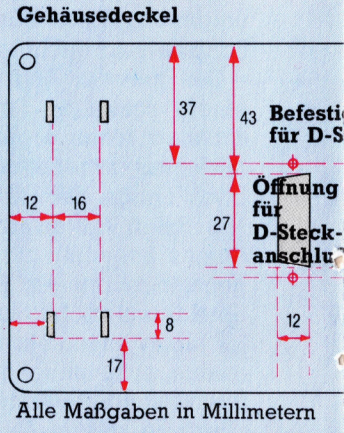
Vier Datenleitungen des User Ports werden für die Motorsteuerung gebraucht. Es stehen also nur noch vier Leitungen für die Eingabe der Sensordaten zur Verfügung. Um eine maximale Flexibilität zu erreichen, wird unser Robot-Auto deshalb mit einem Stecksystem ausgerüstet. Durch einfaches Umstecken der Verbindungskabel können so unterschiedliche Sensor-Kombinationen mit den vier Eingangsleitungen des User Port verbunden werden. Beispielsweise könnten in einem bestimmten Anwendungsfall alle vier Berührungstaster nötig sein, bei anderer Gelegenheit jedoch zwei Berührungstaster und zwei Lichtsensoren.

Durch die sensorischen Fähigkeiten und die präzise Steuerung des Robot-Autos ergibt sich die Möglichkeit, ausgefeilte Software einzusetzen, mit der sich der Roboter eine interne Karte seiner Umgebung erstellen kann. Das führt zu interessanten Fragestellungen bei der Planung des einzuschlagenden Weges und zu trickreichen Suchstrategie-Algorithmen. Wir beginnen mit der mechanischen Konstruktion. Der Zuschnitt der „Karosserie" und die Montage der Getriebe und D-Stecker muß möglichst exakt sein – stellen Sie Ihr handwerkliches Geschick auf die Probe!

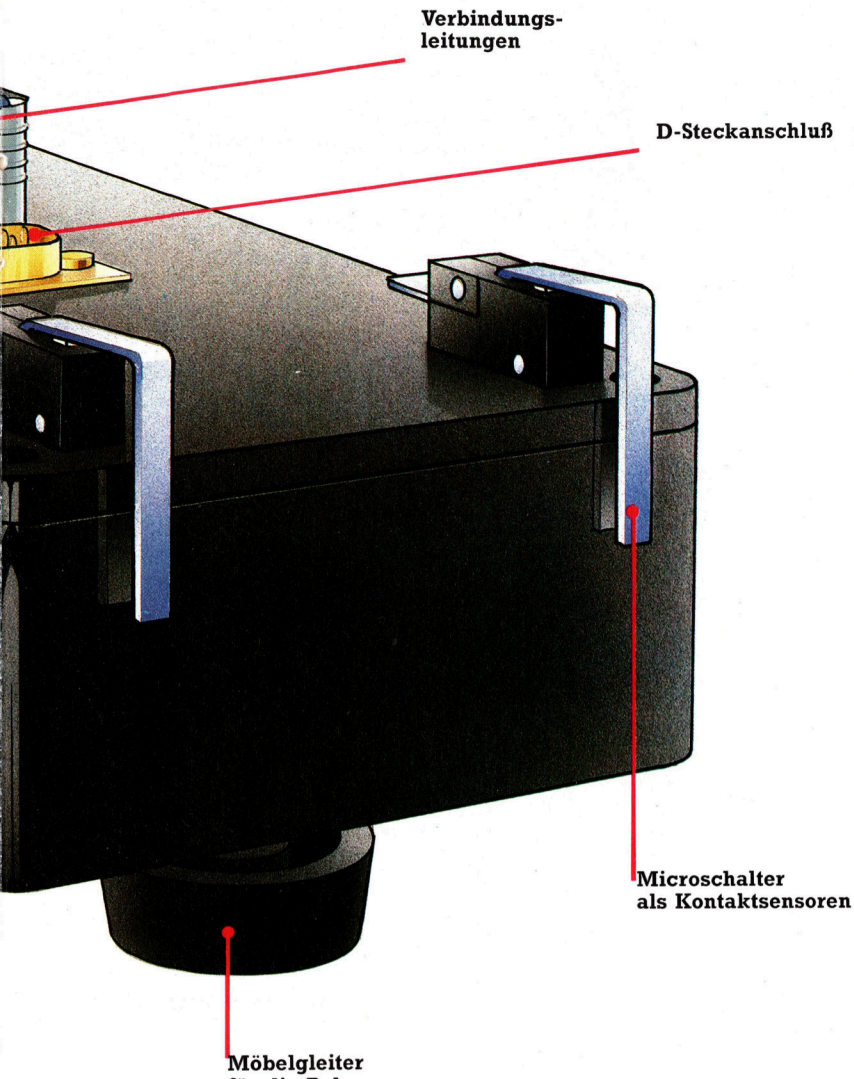


Erster Schritt

Schneiden Sie zuerst die eingezeichneten Löcher in das Kunststoffgehäuse der „Karosserie". In der Zeichnung sind alle Bohrungen und Schlitze mit genauen Maßen versehen. Durch die Öffnungen im Boden und an den Seiten führen später die Achsen der Räder. Die Befestigungslöcher für Motoren und Getriebe müssen auf beiden Seiten in exakt gleicher Höhe liegen. Die beiden Löcher im Boden nehmen die Füßchen auf, die das Robot-Auto in der Balance halten. Die Öffnung im Deckel wird mit einer D-förmigen Buchse für das Verbindungskabel zum Computer versehen. Die großen Löcher können Sie zuerst grob mit einem heißen Messer oder dem Lötkolben ins Gehäuse hineinschmelzen, die Feinarbeit wird danach mit einer kleinen Feile gemacht.



Alle Maßgaben in Millimetern

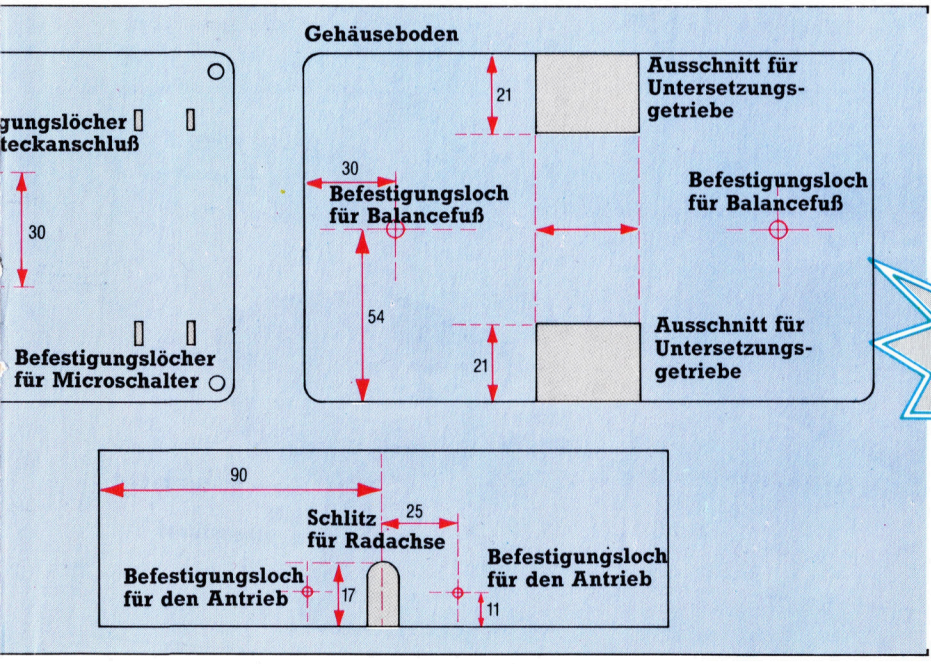


Liste der Bauteile

Anzahl	Bauteil
2	SAA-1027-Schrittmotor-Treiber
2	Schrittmotoren
2	Untersetzungsgetriebe 25:2
1	40109
3	16polige IC-Sockel
2	Widerstände, 100 Ohm
2	Widerstände, 270 Ohm, 0,5 Watt
2	Kondensatoren, 0,1 µF
1	Kondensator, 1000 µF, 25 Volt
1	Lochplatine, 24 Streifen à 50 Löcher
1	Verzinnte Schaltlitze
1	D-Stecker, 15polig
1	D-Buchse, 15polig
1	Gehäuse für D-Stecker
1	Netzteilbuchse, 2,1 mm ∅
1	IDC-Buchse, 20polig (Acorn B)
1	Platinenstecker, 24polig (C 64)
1	Gehäuse, 180 × 110 × 55 mm
1	Rolle Klebeband
2	Möbelgleiter
1	Pckg. Muttern, M5
1	Pckg. Schrauben, M5 × 25
2	Schrauben, M5 × 40
1	Pckg. Schrauben M3 × 20
1	Pckg. Muttern M3
2	Räder (Lego, 62 mm)
1	Satz Achsen (Lego)
4 Meter	Flachkabel, 12polig
1 Meter	Flachkabel, 20polig (Acorn B)
1	Gleichspannungs-Netzteil, 12 Volt, 1 A

Alle Teile zusammen können bis knapp 200 Mark kosten. Die Bauteile können Sie im Versandhandel oder in einem gut sortierten Elektronikladen kaufen. Schrittmotoren und Räder findet man oft auch in Geschäften für Modellbau-Artikel.

Haben Sie gelegentlich Probleme bei der Bauteilbeschaffung? Oft kann man sich aus dieser Lage mit ein wenig Phantasie befreien. Die meisten Mechanikteile lassen sich auch durch Bauelemente anderer Form und Größe ersetzen. Aber Vorsicht bei der Elektronik! Am sichersten ist es, wenn Sie die Bauanleitung zum Händler mitnehmen – dann ist es für ihn einfacher, ein vergleichbares Bauteil herauszusuchen.



ACHTUNG!

Das Robot-Auto verbraucht relativ viel Strom. Wenn das Netzteil gleichzeitig noch den Ausgangsbuffer versorgen müßte, würde das Fahrzeug wegen „Treibstoffmangel" einfach stehenbleiben – die Steuerung muß daher direkt über den User Port erfolgen. Das heißt aber: Hände weg, wenn Sie die Bauanleitung nicht genau einhalten können – andernfalls könnte Ihr Rechner schwere Schäden davontragen!

Zweiter Schritt

Motoren und Untersetzungsgetriebe werden einzeln gekauft und müssen erst zusammengebaut werden. Zum Getriebe gehören ein Distanzstück und ein kleines Zahnrad, das auf die Motorachse gesetzt wird. Sie können es auf der Achse mit etwas Superkleber sicher befestigen. Die Vertiefung auf der einen Seite des Zahnrades zeigt dabei vom Motor weg. Mit dem einen Ende des Distanzstükkes wird das Zahnrad wie in der Anleitung auf den richtigen Abstand vom Motor einjustiert. Danach sollte der Kleber zwei bis drei Stunden trocknen.

Dritter Schritt

Der Motor (mit montiertem Zahnrad) wird so befestigt, daß seine Zuleitungen auf der breiteren Seite des Untersetzungsgetriebes liegen. Die passenden Schrauben sind dem Getriebe beigelegt.

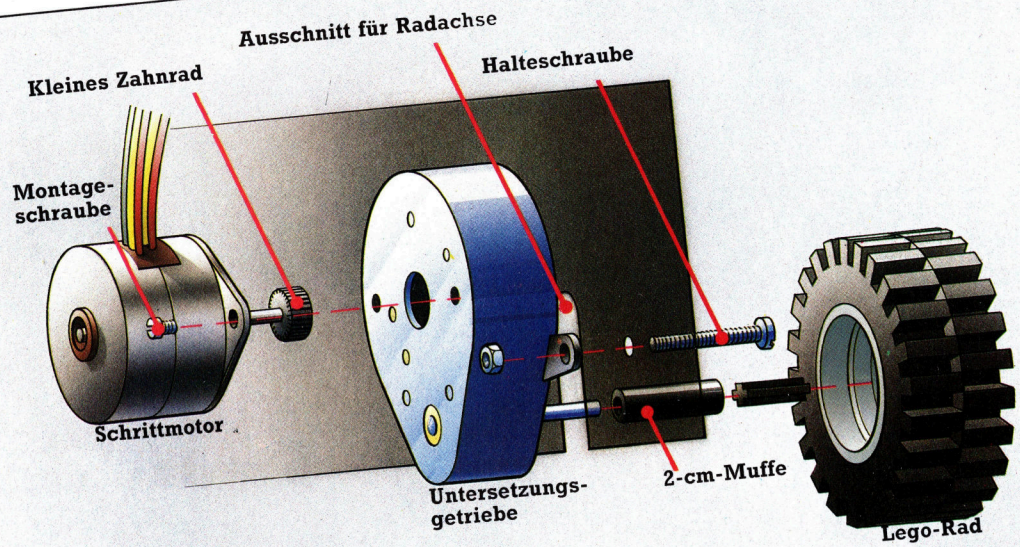
Die M5-Schrauben halten Antrieb und Karosserie zusammen. Damit sich die Räder zusammen mit dem Antrieb später noch verstellen lassen, wird nicht direkt verschraubt, sondern nur geklemmt.

Die Räder passen nur auf die entsprechend gekerbten Lego-Achsen. Schieben Sie den 20 mm langen Abschnitt eines Plastik-Kugelschreibers von geeignetem Durchmesser auf die Getriebeachse, und kleben Sie ihn fest. In die andere Öffnung dieser provisorischen Muffe wird dann eine der kurzen Lego-Achsen festgeklebt, auf die sich die Räder einfach aufstecken lassen.

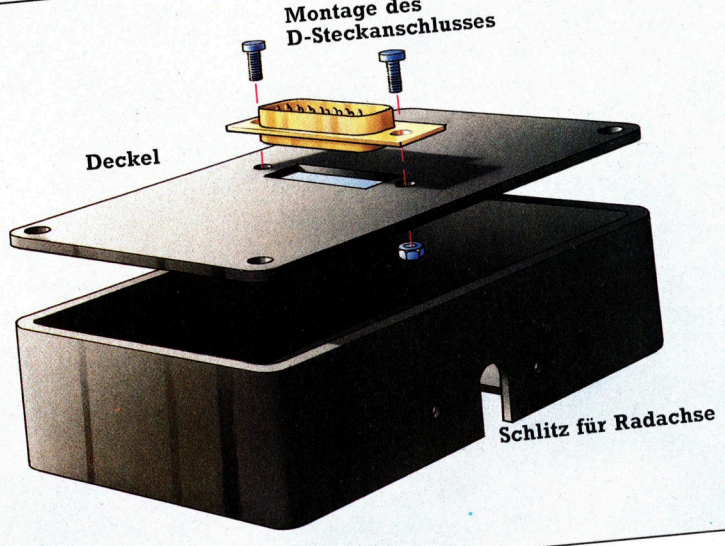
Vierter Schritt

Der (männliche) D-Stecker wird – die Stifte müssen nach oben zeigen – mit den M3-Schrauben am Gehäusedeckel befestigt. Danach montieren Sie beide Möbelgleiter mit M5-Schrauben, die im Inneren der Karosserie durch Muttern gesichert werden. Die Bodenfläche der Gleiter sollte ca. 3 cm von der Unterseite des Gehäuses entfernt liegen. Verwenden Sie zum Justieren Abstandhalter oder eventuell Kontermuttern.

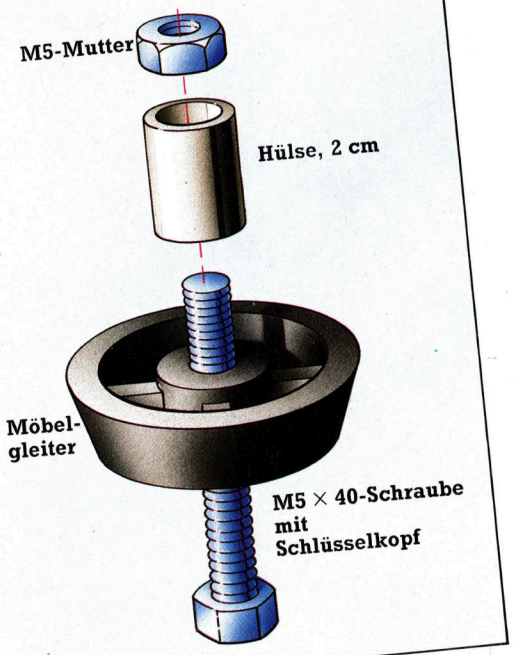
2 und 3



4



4



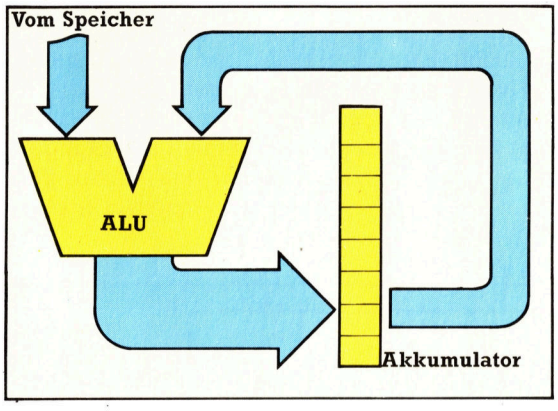
Logisches Finale

In jedem Computer wird der Datenfluß von einer zentralen Steuereinheit, der CPU, dirigiert. Die CPU ist auch für die Ausführung der Programmbefehle sowie für arithmetische und logische Berechnungen zuständig.

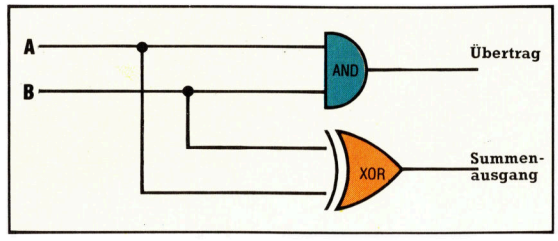
Das Arbeitsfeld der ALU (Arithmetik- und Logikteil der CPU) gliedert sich in zwei Gebiete: Rechenfunktionen, also Addition, Subtraktion und Inkrementierung (das schrittweise Erhöhen einer Zahl um den Wert Eins), und die Ausführung der drei wichtigsten Logikoperationen OR, XOR und AND.

Einige dieser Funktionen, etwa die Addition, beziehen sich auf zwei Operanden (Zahlen, die verarbeitet werden sollen), andere, wie die Inkrementierung, haben nur einen Operanden. Im letzteren Fall wird der benötigte Operand im Akkumulator, einem speziellen Register der CPU, zwischengespeichert. Bei zwei Operanden wird der fehlende aus dem Hauptspeicher abgerufen. Die beiden Zahlen werden nun durch den Schaltkreis der ALU „befördert", und die gewünschte Operation kann ausgeführt werden. Nach der Verarbeitung der Werte wird das Ergebnis wiederum in den Akkumulator geschrieben.

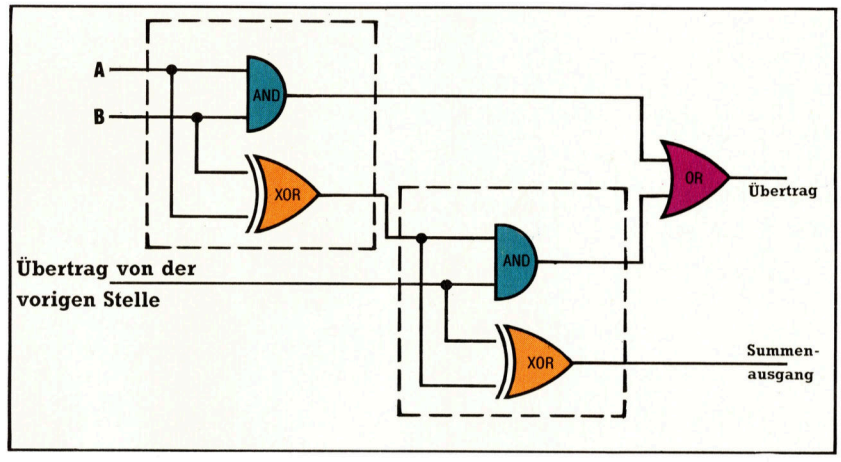
Diese Zeichnung verdeutlicht den Weg der Daten beim Passieren der ALU:



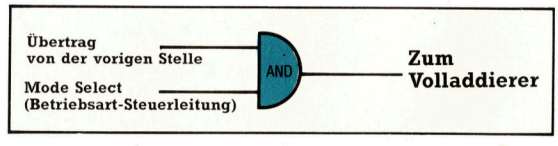
Zahlenwerte im Akkumulator und im Speicher werden mit acht Bits dargestellt. Diese acht Bits werden „parallel" verarbeitet, also alle gleichzeitig. Um die Arbeitsweise der ALU darzustellen, wollen wir eine Ein-Bit-Schaltung entwerfen, die alle sechs ALU-Funktionen ausführt. Das Bit des ersten Operanden nennen wir A, das des zweiten B. Grundlage unserer Ein-Bit-ALU ist ein Volladdierer. Wir haben ihn bereits in einem früheren Kursabschnitt aus zwei Halbaddierern aufgebaut, die mit AND-, OR- und NOT-Gattern realisiert wurden. Die Schaltung eines Halbaddierers kann durch Verwendung eines XOR-Gatters vereinfacht werden:



Zwei dieser Halbaddierer werden zu einem Volladdierer zusammengeschaltet:



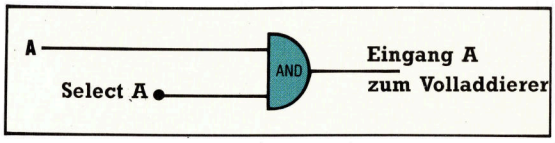
Damit der Volladdierer alle ALU-Funktionen ausführen kann, braucht er einige Zusatzschaltungen, die besondere Steuersignale einspeisen. Das wichtigste Steuersignal ist das „mode select"-Signal (Wahl der Betriebsart). Der Eingang „Übertrag von der vorigen Stelle" wird ausschließlich bei arithmetischen, nicht aber bei Logik-Operationen benötigt. Das „mode select"-Signal schaltet den Übertrags-Eingang dann über ein AND-Gatter ein oder aus:

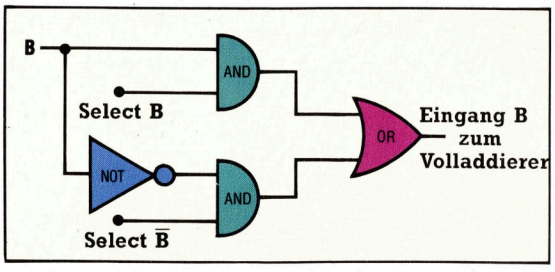


Bei arithmetischen Funktionen steht das „mode select"-Signal auf Eins, und der Übertrag wird durch das Gatter übertragen. Wenn er nicht gebraucht wird, steht „mode select" auf Null. Auf ähnliche Weise werden auch die anderen beiden Eingänge mit AND-Gattern versehen, um die Einspeisung von Bit A, Bit B oder beider Bits gleichzeitig zu steuern.

Für die Subtraktion durch Addition des Zwei-

erkomplements einer Zahl ist die Berechnung des Komplements erforderlich – alle Einsen müssen in Nullen, und alle Nullen in Einsen verwandelt werden. Zur Subtraktion wird also die Zugriffsmöglichkeit auf die Negation von Bit B gebraucht. Zu diesem Zweck wird Eingang B durch ein NOT-Gatter geschleust. Das dazugehörige „select"-Signal wird über ein weiteres AND-Gatter geliefert. Hier die gesamte Schaltung für die Eingabe von A, B mit den Steuersignalen:



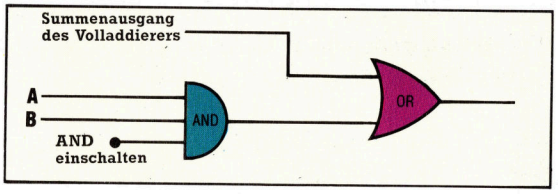


Durch Verwendung von vier Steuersignalen können alle arithmetischen Funktionen ausgeführt werden. Die Tafel gibt die jeweilige Kombinationen der Steuersignale an:

Funktion	Mode Select	Select A	Select B	Select $\overline{B}$
Addition	1	1	1	0
Subtraktion	1	1	0	1
A inkrementieren (Setzt 1. Übertrags-Eingang auf 1)	1	1	0	0
B inkrementieren	1	0	1	0

Bei Logikfunktionen ist der Übertragseingang unnötig – „mode select" auf Null schaltet ihn ab. Die logische XOR-Funktion ergibt sich jetzt am Summenausgang mit einem HI (Eins) auf Select A und Select B und einem LO (Null) als „mode select"-Signal.

Zur Realisierung der AND-Funktion sind getrennte Eingänge für A und B sowie ein besonderes AND-Select-Signal nötig:



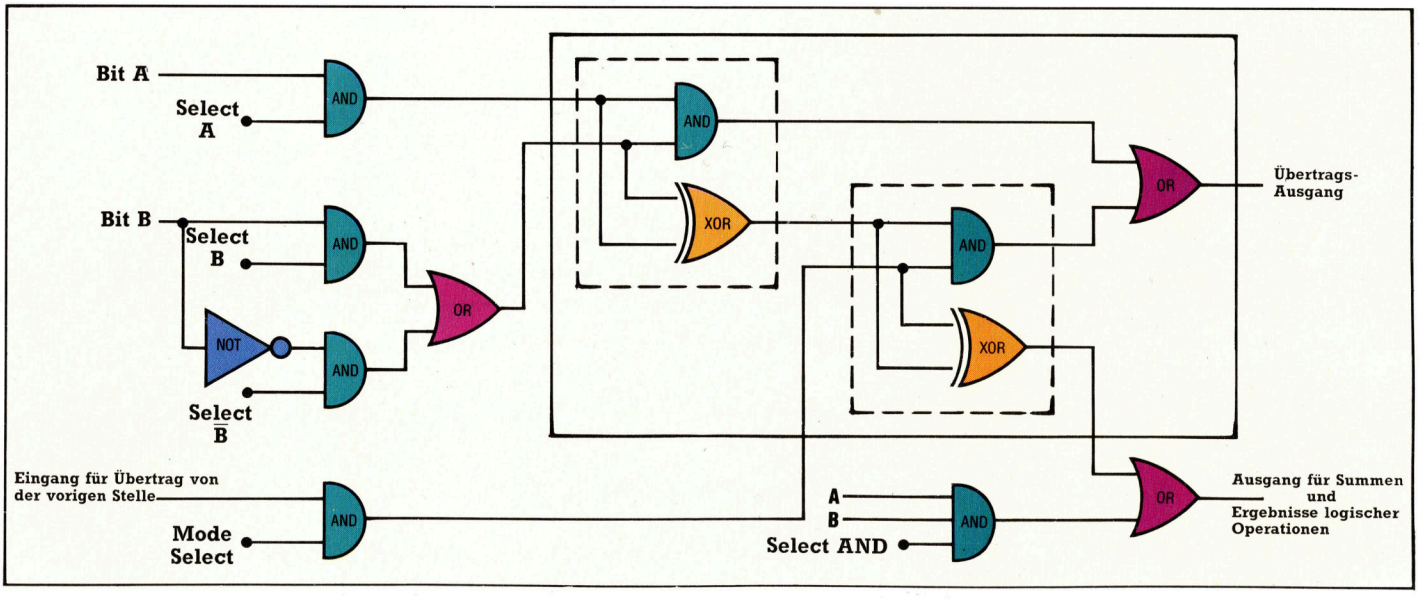
Als letztes fehlt noch die OR-Funktion. Dafür werden XOR- und AND-Ausgänge durch ein OR-Gatter verknüpft, wobei sich diese Wahrheitstafel ergibt:

A	B	Ausgabe	Funktion
0	0	0	
0	1	1	XOR-Funktion
1	0	1	
1	1	1	AND-Funktion

Die folgende Zusammenstellung zeigt, welche Kombination von Steuersignalen zu den einzelnen Logikfunktionen gehört:

Funktion	Mode Select	Select A	Select B	Select $\overline{B}$	Select AND
XOR	0	1	1	0	0
AND	0	0	0	0	1
OR	0	1	1	0	1

Unten auf dieser Seite ist die vollständige Schaltung der Ein-Bit-ALU mit dem zentralen Volladdierer und den Ergänzungen für die Steuersignale abgebildet. Zur Parallelverarbeitung von acht Bits sind acht dieser Schaltungen nötig. Der Übertragsausgang des achten Bits dient als Flag-Register des Prozessor-Status-Registers.



Am Tatort

Bis jetzt haben wir in unserem Abenteuer-Programm-Projekt eine Karte des Spielgeschehens sowie eine Hilfsroutine zur formatierten Textausgabe entwickelt. Jetzt können wir Routinen entwerfen, die die Orte beschreiben.

Die Grundbeschreibung jedes Ortes wird im Array LN$() gespeichert. Der Zugriff erfolgt über die Nummer des aktuellen Ortes. Bei „Haunted Forest" wird die gegenwärtige Position des Spielers in der Variablen P und die Beschreibung in LN$(P) gespeichert. Bei der Formulierung der Ortsbeschreibungen wurde darauf geachtet, daß gegebenenfalls „You are . . ." („Sie sind . . .") vorangestellt werden kann. Für einen Ort P kann also mittels der im letzten Artikel entwickelten Routine die Beschreibung formatiert und ausgegeben werden, indem „You are" mit dem entsprechenden Teil im Array LN$() kombiniert wird.

Zusätzlich zur Ortsbeschreibung muß der Spieler auch erfahren, ob sich dort eventuell Objekte befinden. Die im Spiel vorkommenden Objekte werden, gemeinsam mit ihrer Anfangsposition, in einem zweidimensionalen Array IV$(,) gespeichert. IV$(N,1) enthält also die Beschreibung des N'ten Objektes und IV$(N,2) seine Position. Zur Bestimmung, ob sich an einem Ort ein Objekt befindet oder nicht, muß dieses Verzeichnis durchsucht und die Objektposition mit der Nummer des gerade beschriebenen Ortes verglichen werden. Da es in Haunted Forest nur drei und in Digitaya acht Objekte gibt, kann mittels einer FOR. . .NEXT-Schleife linear gesucht werden.

In den Zeilen 2050–2080 befindet sich die Such-Schleife von Haunted Forest. Die Werte der zweiten Spalte des Arrays IV$(,) werden mit der Ortsnummer verglichen. Wird ein Objekt gefunden, stellt das Programm die entsprechende Beschreibung dar. Da an einem Ort mehrere Objekte sein können, muß die Konstruktion eines Satzes möglich sein, in dem die einzelnen Objekte, getrennt durch ein Komma, aufgelistet werden. Dies geschieht mittels SP$. Ein Flag F, mit dem Ursprungswert 0, wird auf 1 gesetzt, sobald ein Objekt gefunden wird. Beträgt der Wert nach der Suche noch 0, so gibt es an diesem Ort keine Objekte, und es wird eine Meldung ausgegeben.

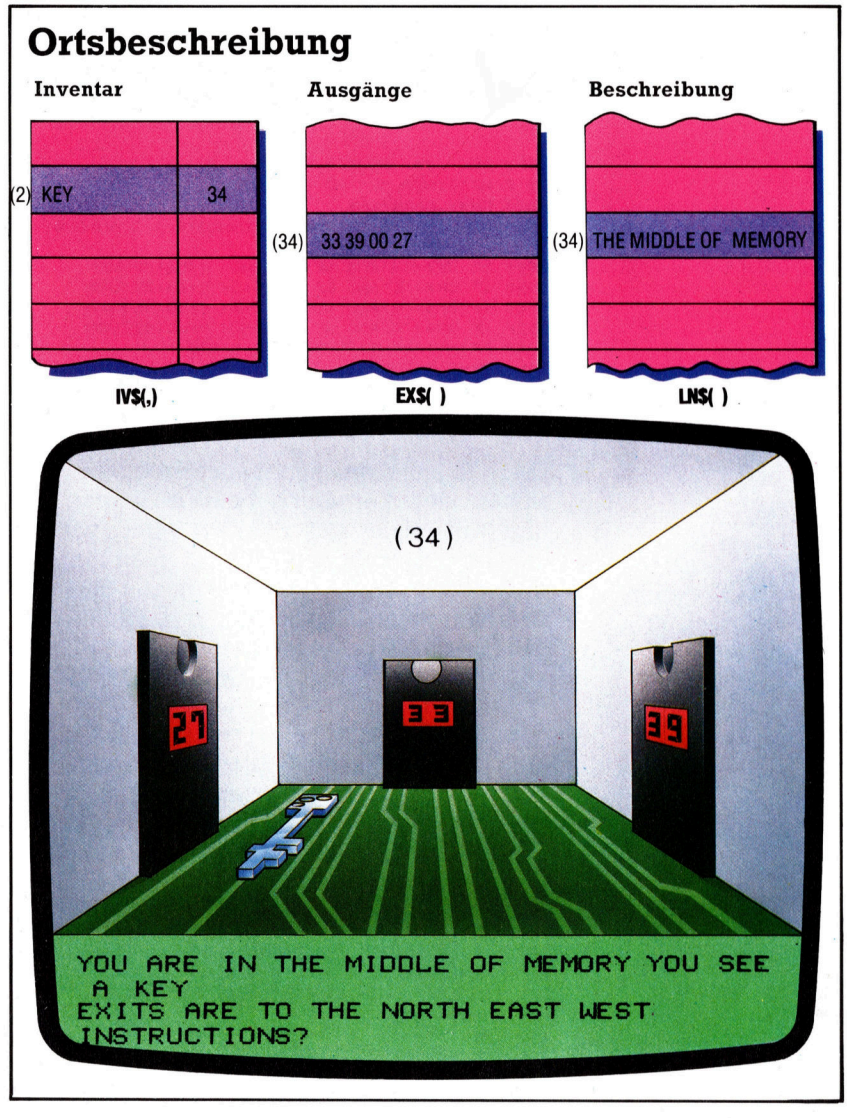
```
2000 REM **** DESCRIBE LOCATION ****
2010 SN$="YOU ARE "+LN$(P):GOSUB5500
2020 SN$="YOU SEE "
2030 REM ** CHECK INVENTORY FOR OBJ **
2040 F=0:SP$=""
2050 FOR I=1 TO 3
2060 IF VAL(IV$(I,2))<>P THEN 2080
2070 SN$=SN$+SP$+"A "+IV$(I,1):F=1:SP$=", "
2080 NEXT I
2090 IF F=0 THEN SN$=SN$+"NO OBJECTS"
2100 GOSUB5500:REM FORMAT OUTPUT
2110 RETURN
```

Die Daten, die Details über mögliche Ausgänge von einem Ort enthalten, befinden sich im Array EX$(). Jeder Wert dieses Strings besteht aus acht Stellen. Unterteilt man diese in Zweiergruppen, so erhält man (von links nach rechts) die Nummern der Orte im Norden, Osten, Süden und Westen. Durch diese Aufteilung stellt das Programm fest, welche Ausgänge möglich sind.

```
2300 REM **** DESCRIBE EXITS S/R ****
2310 EX$=EX$(P)
2320 NR=VAL(LEFT$(EX$,2))
2330 EA=VAL(MID$(EX$,3,2))
2340 SO=VAL(MID$(EX$,5,2))
2350 WE=VAL(RIGHT$(EX$,2))
```

Gibt es in einer Richtung keinen Ausgang, so

Die Details der Orte in unserem Abenteuerspiel werden in drei String-Arrays gespeichert. Sie enthalten Objekt-Namen und deren Positionen (IV$), mögliche Ausgänge (EX$) und Beschreibungen (LN$). EX$(34) kann beispielsweise die achtstellige Zahl 33390027 enthalten. Dies besagt, daß Ort 34 durch die Ausgänge nach Norden, Osten und Westen Verbindungen zu den Orten 33, 39 und 27 hat.



ist der zugeordnete Wert Null. Bevor der Satz „There are exits to the . . ." konstruiert werden kann, muß eine Vorabüberprüfung durchgeführt werden. Dies ist mittels eines logischen OR für alle vier Richtungsvariablen möglich, wobei man nur dann eine Null als Ergebnis erhält, wenn alle Richtungsvariablen Null sind. Ist dies nicht der Fall, überprüft die Routine jede einzelne Variable. Lautet eine Variable nicht Null, so wird die entsprechende Richtung an den Satz gefügt.

```
2355 IF(NR OR EA OR SO OR WE)=0 THEN RETURN
2360 PRINT:SN$="EXITS ARE TO THE "
2370 IF NR <>0 THEN SN$=SN$+"NORTH "
2380 IF EA <>0 THEN SN$=SN$+"EAST "
2390 IF SO <>0 THEN SN$=SN$+"SOUTH "
2400 IF WE <>0 THEN SN$=SN$+"WEST "
2410 GOSUB 5500:REM FORMAT
2415 PRINT
2420 RETURN
```

Jetzt, da die Routinen zum Beschreiben der Orte entwickelt sind, können wir Programmteile erstellen, die dem Spieler Aktionen in unserer Phantasiewelt erlauben. In einem anderen Artikel werden wir uns auch mit Algorithmen zur Analyse der Eingaben des Spielers befassen. Hier wollen wir uns nur mit den Befehlen zur Bewegung befassen, die in einem Wort, wie „NORTH" oder „SOUTH", eingegeben werden. Wird eine solche Anweisung an die Variable NN$ übergeben, sieht eine entsprechende Bewegungsroutine wie folgt aus:

```
3500 REM **** MOVE S/R ****
3510 MF=1:REM SET MOVE FLAG
3520 DR$=LEFT$(NN$,1)
3530 IF DR$<>"N"ANDDR$<>"E"ANDDR$<>"S"ANDDR$<>"W"
  THEN GOTO3590
3540 IF DR$="N"AND NR<>0 THEN P=NR:RETURN
3550 IF DR$="E"AND EA<>0 THEN P=EA:RETURN
3560 IF DR$="S"AND SO<>0 THEN P=SO:RETURN
3570 IF DR$="W"AND WE<>0 THEN P=WE:RETURN
3580 PRINT:PRINT"YOU CAN'T ";IS$
3585 MF=0:RETURN
3590 REM ** NOUN NOT DIRECTION **
3600 PRINT"WHAT IS ";NN$;" ?"
3610 MF=0:RETURN
```

Diese Routine verwendet nur den ersten Buchstaben des Bewegungsbefehls. Zuerst wird überprüft, ob es sich bei dem Befehl wirklich um eine Richtung handelt. Ist dies der Fall, wird sichergestellt, daß sich in der Richtung überhaupt ein Ausgang befindet. Wenn ja, erhält P – die Variable mit der aktuellen Spielerposition – den Wert von NR, EA, SO oder WE.

Bevor wir die hier entwickelten Unterroutinen verwenden können, müssen sie durch eine Wiederholungs-Schleife verknüpft werden. Das Flußdiagramm zeigt die notwendige logische Struktur. Obwohl es nicht die Endversion ist, genügt es doch zur Veranschaulichung der Gesamtstruktur. Fügen Sie nun die folgenden Programmzeilen ein.

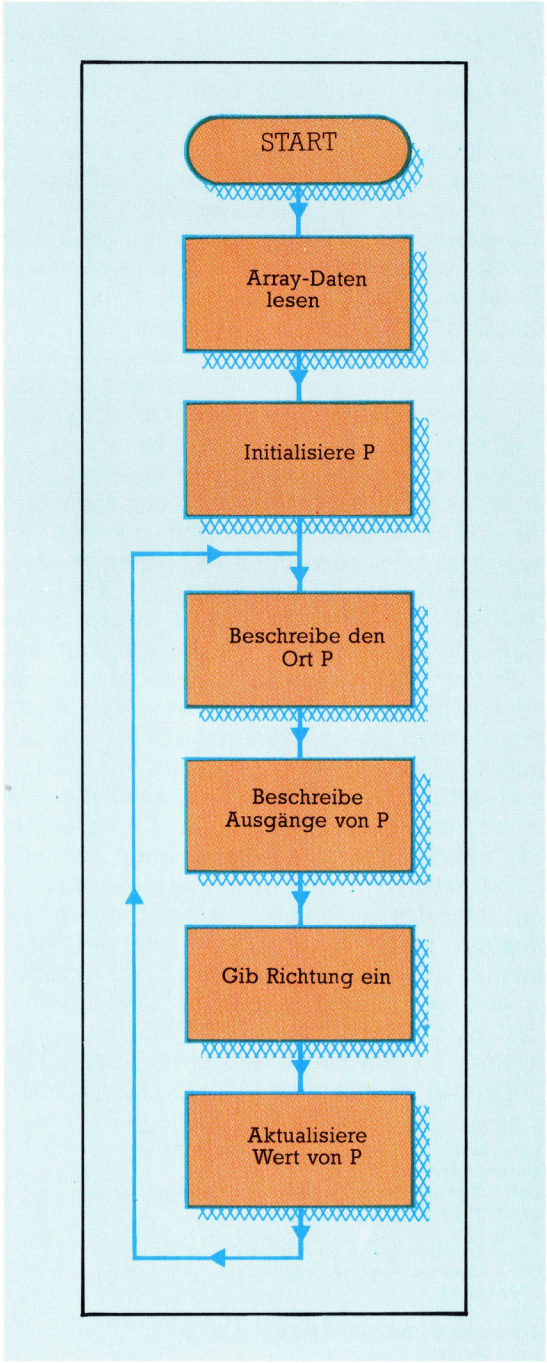
```
200 GOSUB6000:REM READ ARRAY DATA
210 P=INT(RND(TI)*10+1):REM START POINT
230 REM **** MAIN LOOP STARTS HERE ****
240 MF=0:REM MOVE FLAG
245 PRINT
250 GOSUB2000:REM DESCRIBE POSITION
255 GOSUB2300:REM DESCRIBE EXITS
260 PRINT:INPUT"INSTRUCTIONS";IS$
```

Geben Sie außerdem die folgenden Zeilen in die Hauptschleife ein:

```
270 NN$=IS$:GOSUB 3500:REM MOVE
280 GOTO 230:REM RESTART MAIN LOOP
```

Spectrum-Variationen

Da der Spectrum alle String-Arrays als Zeichenketten mit festgelegter Länge handhabt, entstehen Probleme, wenn man Teile eines Strings als Bestandteil eines größeren Satzes ausgeben will. Bei der Dimensionierung eines Arrays auf dem Spectrum bestimmt die letzte Zahl der Anweisung die Länge jedes Elements im Array. Beispielsweise dimensioniert DIM a$(3,2,20) ein 3 × 2-Array, wobei jedes Ele-



ment maximal 20 Zeichen umfassen kann. Ordnet man einem Element weniger als 20 Zeichen zu, wird der Rest mit Leerstellen aufgefüllt, wodurch viel Speicherplatz verschwendet wird. Um nun trotzdem String-Array-Variablen in Sätze einzufügen, müssen die Leerstellen entfernt werden. Für das Haunted-Forest-Listing müssen Sie die folgende Routine eingeben:

```
7000 REM **** SPECTRUM TRUNCATE ****
7010 FOR I=LEN(A$) TO 1 STEP -1
7020 IF A$(I TO I)<>"" THEN LET N=
     I:LET I=1
7030 NEXT I
7040 LET S$=S$+A$(TO N)
7050 RETURN
```

Beim Digitaya-Listing geben Sie dieselben Zeilen ein, doch verwenden Sie die Zeilennummern 8500 bis 8550.

Weitere Änderungen

Die Routine kürzt A$ durch Entfernung aller unnötigen Leerstellen, bevor A$ zu S$ hinzugefügt wird. Denken Sie daran, daß S$ die String-Variable ist, die zur formatierten Textausgabe verwendet wird. Vor Aufruf der Routine muß das in den Satz einzufügende Element in A$ übertragen werden. Aus diesem Grund sind bei den Spectrum-Versionen der beiden Programme noch folgende Änderungen notwendig:

Haunted Forest:

```
2010 LET S$="YOU ARE ":A$=L$(P):
     GOSUB7000:GOSUB 5500
2070 LET S$=S$+P$+"A": A$=V$(I,1):
     GOSUB7000:LET F=1:LET P$=","
```

Digitaya:

```
1450 LET S$="YOU ARE ":A$=L$(P):
     GOSUB8500:GOSUB5880
1500 IF VAL(V$(I,2)=P THEN LET
     S$=S$+P$+"A":A$=V$(I,1):
     GOSUB8500:LET F=1:LET P$=" "
```

Digitaya-Listing

Die Struktur von Digitaya ähnelt der von Haunted Forest. Fügen Sie die folgenden Programmzeilen in das bisher gezeigte Listing ein:

```
1100 GOSUB6090:REM READ ARRAY DATA
1210 PRINT:INPUT"INSTRUCTIONS";IS$
1120 P=47:REM START POINT
1130 :
1140 REM **** MAIN LOOP STARTS HERE ****
1150 :
1160 MF=0:PRINT
1170 GOSUB1440:REM DESCRIBE POSITION
1180 GOSUB1560: REM LIST EXITS
```

Integrieren Sie außerdem diese Zeilen:

```
1220 NN$=IS$:GOSUB 2000:REM MOVE
1230 GOTO 1140:REM RESTART MAIN
     LOOP
```

Beschreibung der Orte und Ausgänge

```
1440 REM **** DESCRIBE POSITION S/R ****
1450 SN$="YOU ARE "+LN$(P):GOSUB5880
1460 SN$="YOU SEE "
1470 REM ** SEARCH FOR OBJECT **
1480 F=0:SP$=""
1490 FOR I=1TO8
1500 IF VAL(IV$(I,2))=P THEN SN$=SN$+SP$+"A
"+IV$(I,1):F=1:SP$="  "
1510 NEXTI
1520 IF F=0 THENSN$=SN$+"NO OBJECTS"
1530 GOSUB5880:REM FORMAT
1540 RETURN
1550 :
1560 REM **** LIST EXITS S/R ****
1570 EX$=EX$(P)
1580 NR=VAL(LEFT$(EX$,2))
1590 EA=VAL(MID$(EX$,3,2))
1600 SO=VAL(MID$(EX$,5,2))
1610 WE=VAL(RIGHT$(EX$,2))
1620 IF(NR OR EA OR SO OR WE)=0THEN RETURN
1630 PRINT:SN$="EXITS ARE TO THE "
1640 IF NR<>0 THEN SN$=SN$+"NORTH "
1650 IF EA<>0 THEN SN$=SN$+"EAST "
1660 IF SO<>0 THEN SN$=SN$+"SOUTH "
1670 IF WE<>0 THEN SN$=SN$+"WEST "
1675 GOSUB 5890:REM FORMAT
1680 PRINT:RETURN
```

Bewegungs-Unterroutine

```
2000 REM **** MOVE S/R ****
2010 MF=1:REM MOVE FLAG SET
2020 DR$= LEFT$(NN$,1)
2030 IFDR$<>"N"ANDDR$<>"E"ANDDR$<>"S"ANDDR$<>
"W"TH EN2100
2040 IF DR$="N" AND NR<>0 THEN P=NR:RETURN
2050 IF DR$="S" AND SO<>0 THEN P=SO:RETURN
2060 IF DR$="E" AND EA<>0 THEN P=EA:RETURN
2070 IF DR$="W" AND WE<>0 THEN P=WE:RETURN
2080 PRINT"YOU CANT ";IS$
2090 MF=0:RETURN
2100 REM NOUN NOT OK
2110 PRINT"WHAT IS ";NN$;" ?"
2120 MF=0:RETURN
```

BASIC-Dialekte

Spectrum:
In beiden Programmlistings muß EX$() durch E$(), EX$ durch X$, SN$ durch S$, IS$ durch T$, LN$() durch L$, NN$ durch R$, SP$ durch P$ und DR$ durch D$ ersetzt werden. Beim Digitaya-Listing tauschen Sie bitte die folgenden Zeilen aus:

```
1580 LET NR=VAL(X$(TO 2))
1590 LET EA=VAL(X$(3 TO 4))
1600 LET SO=VAL(X$(5 TO 6))
1610 LET WE=VAL(X$(7 TO))
2020 LET D$=R$(TO 1)
```

Im Haunted-Forest-Listing ersetzen Sie bitte die folgenden Zeilen:

```
210 RANDOMISE:P=INT(RND(1)*10+1)
2320 LET NR=VAL(X$(TO 2))
2330 LET EA=VAL(X$(3 TO 4))
2340 LET SO=VAL(X$(5 TO 6))
2350 LET WE=VAL(X$(7 TO))
3520 LET D$=R$(TO 1)
```

Acorn B:
Tauschen Sie im Haunted-Forest-Listing die folgende Zeile aus:

```
210 P=RND(10)
```

Magische Maus

Aufgrund des Erfolges des Apple Macintosh mit seinem Maus-orientierten Betriebssystem hat die Microcomputer-Branche ihre anfänglichen Bedenken über Bord geworfen und bedient sich nun ebenfalls dieser Technik. Hier wird „Magic Mouse" von SMC Supplies für den Commodore 64 vorgestellt.

Angesichts der Beliebtheit von Maus-orientierten Betriebssystemen ist kaum noch zu glauben, daß es in Herstellerkreisen zunächst zahlreiche Debatten gab, ob diese Technik überhaupt Chancen hätte. Die vergleichsweise geringe Begeisterung für den LISA von Apple als erstem Maus-orientierten Micro schien die Meinung zu bestätigen, daß der Markt auf solche Systeme nicht anspreche.

Der spätere Erfolg des Macintosh brachte jedoch die Kritiker zum Schweigen und rechtfertigte die Apple-Philosophie. Inzwischen hat sich die Einschätzung soweit geändert, daß viele Hersteller Maus-orientierte Systeme entwickeln und Drittlieferanten Maus-Pakete für die Nachrüstung gängiger Rechner anbieten.

Die Magic Mouse von SMC Supplies ist eins der ersten Produkte dieser Art für den Commodore 64. Die mitgelieferte Software (Cassette und Diskette) umfaßt vier Anwendungsprogramme. Die Maus selbst ist mit 125x66x50 mm fast doppelt so groß wie die von Apple. Außerdem hat sie drei statt der üblichen zwei Bedienknöpfe. Diese farbigen Tasten erlauben (natürlich softwareabhängig) die Auslösung bestimmter Funktionen. Das Gehäuse enthält eine Hartgummikugel, die gegen ein Rollenpaar mit Codierscheiben gedrückt wird.

Anscheinend war zunächst ein Stahlkugel-System vorgesehen, das sich dann aber nicht bewährte, und die Markteinführung verzögerte sich. Die jetzige Version mit der Gummikugel arbeitet verläßlich.

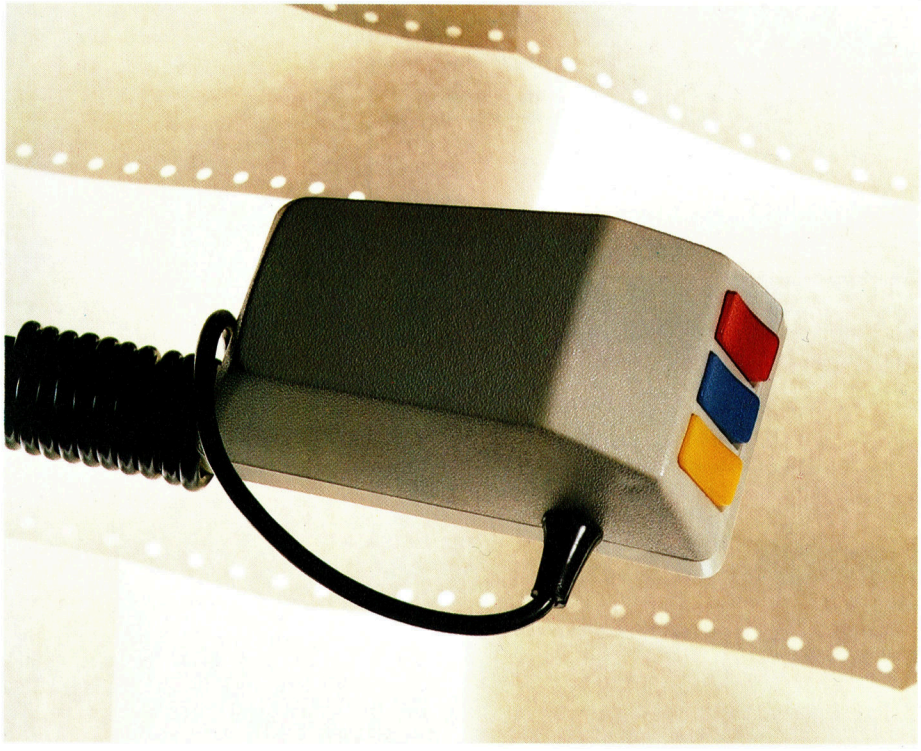
Das Software-Paket enthält die Programme Hi-Res-Designer (Hochauflösender Entwurf), Sprite-Designer (Sprite-Entwurf), Icon-Designer (Piktogramm-Entwurf) und Mouse-Controller (Maus-Steuerung).

Vor der Benutzung ist zunächst die Systemdatei zu laden und dann der Maus-Cursor zu kalibrieren. Beim Laden von der Diskette erscheint automatisch ein Hauptmenü, über das eins der vier Dienstprogramme anzuwählen ist. Bei der Cassettenversion ist die Systemdatei Bestandteil des Hi-Res-Designers.

Der Hi-Res-Designer ist ein „Malkasten"-Programm, das der Software für das Koala-Pad stark ähnelt. Nach dem Laden zeigt der Bildschirm eine Anzahl Kästchen mit verschiedenen Wahlmöglichkeiten wie BOX (Viereck), DRAW (Zeichnen) oder FILL (Einfärben). Die gewünschte Option wird mit Hilfe der Maus und anschließendem Druck auf die rote Taste aktiviert. Ähnlich bestimmen Sie Vorder- und Hintergrundfarbe, indem Sie den Cursor auf einen der 16 Töne der Farbskala setzen und die blaue Bedientaste drücken.

Das Icon-Programm und der Sprite-Designer unterscheiden sich eigentlich nur im Anwendungszweck. Wie die meisten derartigen Programme präsentiert der Sprite-Designer zunächst ein Raster mit 24 Zeilen und 21 Spalten, dessen einzelne Pixel dann mit dem Cursor durch Druck auf den roten Knopf belegt werden. Das Sprite selbst erscheint in einem Fenster in der rechten oberen Schirmecke, und rechts unten werden die Möglichkeiten für das Vergrößern, einen Farbwechsel, für den Wechsel zu einem neuen Sprite usw. angeboten.

Der Icon-Designer arbeitet in gleicher Weise mit einem Gitterraster und einem Fenster, in dem das Piktogramm erscheint. Der Unterschied liegt darin, daß die mit diesem Programm selbst definierten grafischen Symbole mit SAVE gespeichert und beliebig wiederverwendet werden können.



Die Magic Mouse ist eins der ersten Maus-orientierten Systeme für den Commodore 64. Obgleich eigentlich nicht als WIMP-Paket (Windows, Icons, Mice, Programming = Fenster, Bilder, Mäuse, Programmierung) gedacht, ist die Magic Mouse ein nützliches „Werkzeug". Das Kabel wird an den Joystick-Port des Commodore 64 angeschlossen.

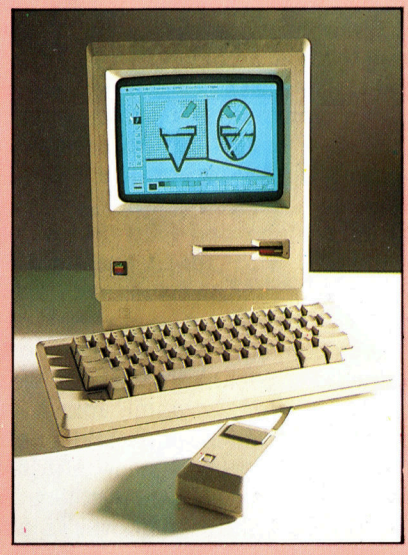
Das letzte Programm im Magic-Mouse-Paket, der Mouse-Controller, ist die Treiber-Software, die eine Maus-Steuerung in selbstgeschriebenen Programmen ermöglicht. Der Mouse-Controller bietet damit letztlich die meisten Anwendungsmöglichkeiten.

Der große Vorteil von Maus-orientierten Systemen gegenüber Digitalisiertabletts besteht in der stabilen Cursorlage, die eine exakte Linien- und Farbführung erlaubt. Fast alle Tabletts arbeiten mit einem Gitter, dessen Auflösung deutlich geringer als die des Bildschirms ist. Wenn sich der Griffel gerade zwischen zwei Gitterpunkten befindet, ist die Zuordnung für den Rechner nicht eindeutig, und es entsteht eine unpräzise Positionswiedergabe.

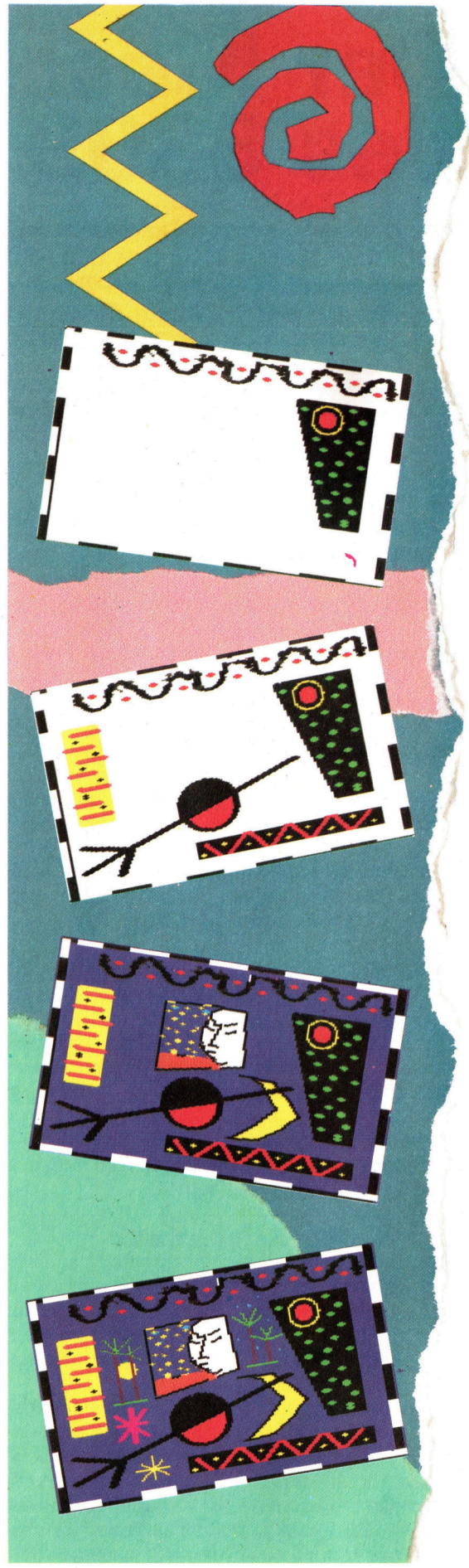
Trotzdem ist das System nicht ganz vollkommen. Der Hi-Res-Designer ist weniger benutzerfreundlich als einige andere Grafik-Pakete. Die RUB-Routine (Ausradieren) erlaubt beispielsweise nur eine pixelweise Fehlerkorrektur – das kostet Zeit und führt leicht zu neuen Fehlern, wenn etwas versehentlich gelöscht wird. Besser ist das Verfahren, alle Ergänzungen auf der Malfläche zu löschen, die seit dem letzten Menü-Aufruf hinzugefügt wurden.

Die mitgelieferte Anleitung ist ausgezeichnet. Für jedes Programm wird ausführlich erläutert, wie es arbeitet, und hinzu kommen Programmierhinweise mit Beispielen, die Ihnen den Einbau der fertigen Grafiken in Ihre eigene Software ermöglichen.

Obwohl die Magic Mouse von SCM durchaus nicht mit dem Macintosh-System vergleichbar ist, wurde die zugehörige Software offensichtlich als Programmierwerkzeug entwickelt, ähnlich wie bei der AMX-Maus. Zweifellos erwartet SMC, daß seitens kommerzieller Programmierer weitere Software produziert und damit das Interesse an diesem System verstärkt wird.



Der Apple Macintosh ist ein hervorragendes Beispiel für perfekte Maus-Steuerung.



Diese Bilder wurden mit dem Hi-Res-Designer aus dem Magic-Mouse-Softwarepaket angefertigt. Das Programm enthält alle Routinen, die bei derartigen Systemen gängig sind, wie LINE (Linienziehen), DRAW (Freihandzeichnen), FILL (Einfärben von Flächen) und CIRCLE (Kreise zeichnen). Bis zu 16 verschiedene Farbtöne stehen zur Verfügung, außerdem ein großes Angebot von Pinseltypen, womit Breite und Struktur von Strichen beliebig gewählt werden können.

Magic Mouse

SCHNITTSTELLE

Die Magic Mouse wird an den Joystick-Eingang des Commodore 64 angeschlossen.

SOFTWARE

Zum Lieferumfang gehören eine Cassette und eine Diskette mit vier Anwendungsprogrammen.

ANLEITUNG

Das Handbuch erläutert ausführlich die Inbetriebnahme der Maus, den Gebrauch der mitgelieferten Software und die Übernahme der erstellten Grafiken in eigene Programme.

STÄRKEN

Das Gerät ist solide gebaut. Der Benutzer hat mit dem Einfügen von Sprites und selbstdefinierten Grafiksymbolen in eigene Programme wenig Mühe.

SCHWÄCHEN

Insgesamt verfügt das Paket nicht über den weiten Anwendungsbereich anderer Maus-orientierter Systeme und schneidet gegenüber ausgefeilterer Software schlecht ab.

Der Maschinencode des 6809

Wir setzen unseren Maschinencodekurs mit einer Artikelserie über die Assemblersprache des 6809-Prozessors fort, der unter anderem die Heimcomputer Dragon und Tandy Color steuert. Zunächst wird untersucht, welche Rolle die Register dieses Prozessors spielen.

Microprozessoren haben im allgemeinen drei Hauptfunktionsbereiche: die Register (interne Speicher des Prozessors), die ALU (Arithmetik- und Logikeinheit, die einfache mathematische Vorgänge ausführt) und eine Steuereinheit, die veranlaßt, daß alle Vorgänge in der richtigen Reihenfolge und zur richtigen Zeit ablaufen.

Auf der untersten Funktionsebene reagieren Microprozessoren auf elektrische Impulse (die sie über externe Anschlußkontakte erhalten), indem sie entweder den Zustand ihrer internen Register ändern oder andere Impulse aussenden oder entgegennehmen. Wenn wir die Anwesenheit eines Stromimpulses als Eins ansehen und die Abwesenheit als Null, lassen sich diese Signale – die entweder zwischen Prozessor und Speicher oder nur im Prozessorinneren hin- und hergehen – als Zahlen eines binären Zahlensystems darstellen. Prozessoren werden programmiert, indem man sie mit einer Folge von Zahlen (bzw. Befehlen) versorgt, die sie dann wie gewünscht bearbeiten bzw. ausführen.

Acht-Bit-Prozessoren wie der 6809 können Binärzahlen mit einer Länge von acht Bits senden und empfangen. Sie umfassen damit die Dezimalzahlen 0 bis 255. Viele dieser Zahlen beziehen sich auf Speicheradressen. Diese werden auf den meisten Acht-Bit-Prozessoren als 16-Bit-Zahlen angegeben. Sie adressieren Speicherstellen von 0 bis 65535. Diese 16-Bit-Adressen verarbeitet der Prozessor natürlich nur in zwei Teilen zu je acht Bit.

Vier Hauptregister

Die Register eines Prozessors können unterschiedliche Formate haben. Einige sind für den internen Gebrauch reserviert und dem Programmierer nicht zugänglich. Der 6809 hat vier Hauptregister.

Am häufigsten wird der *Akkumulator* angesprochen, in dem auch die meisten Daten gespeichert und bearbeitet werden. Beispielsweise addiert der ADD-Befehl den Inhalt einer speziellen Speicherstelle auf die Daten, die sich gerade im Akkumulator befinden. Der Ergebniswert wird so in diesem wichtigen Register „akkumuliert" (angesammelt).

Mit dem *Indexregister* lassen sich Adressen verändern, so daß dann Tabellen und Daten leicht schrittweise angesprochen werden können. Ein Befehl, der sich über die *indizierte Adressierung* auf eine Speicherstelle bezieht, addiert den Inhalt des Indexregisters auf die angegebene Basisadresse und spricht so die *aktuelle Adresse* an. Für die schrittweise Bearbeitung von Datentabellen braucht nur die *Basisadresse* (das erste Datenelement der Tabelle) angegeben und das Indexregister inkrementiert zu werden. Da Indexregister normalerweise Adressen enthalten, haben sie im allgemeinen eine Länge von 16 Bits.

Der *Stack Pointer* enthält die oberste Adresse des *Stacks*, der sich besonders zur schnellen Zwischenspeicherung von Daten eignet. Der Stack wird häufig zur Speicherung des internen Prozessorregisters eingesetzt (etwa beim Aufruf von Unterroutinen), dessen Daten später wieder zurückgeladen werden müssen. Es können beliebig viele Register auf den Stack geschoben und wieder abgezogen werden. Der Stack Pointer enthält dabei die Speicheradresse, auf der die letzte Information abgelegt wurde. Da Stack Pointer sich auf Speicheradressen beziehen, sind sie im allgemeinen ebenfalls 16 Bits lang.

Das vierte Register – der *Programmzähler* – führt seine Funktion fast immer automatisch aus. Er enthält jeweils die Adresse des nächsten Befehls. Der Prozessor spricht diese Speicherstelle an, holt sich ihren Inhalt, interpretiert die Bedeutung und führt den Befehl aus. Normalerweise wird der Programmzähler bei jeder ausgeführten Anweisung automatisch inkrementiert. Verändert man aber seinen Inhalt (durch Addition oder Subtraktion oder durch Speicherung eines neuen Wertes), dann ändert sich auch der Programmablauf. Obwohl dieser Vorgang dem GOTO-Befehl ähnelt, werden auf Maschinenebene neue Adressen mit dem Sprungbefehl (JMP) angegeben und aktuelle Adressen mit dem Verzweigungsbefehl (BRA) verändert.

Die fünfte Registerart funktioniert völlig anders. Das *Condition Code Register* läßt sich als eine Reihe einzelner Bits beschreiben, die In-

Die 6809-Geräte mit der größten Verbreitung sind die Heimcomputer Dragon 32 und 64 und der Tandy Color Computer. Außerdem gibt es eine große Zahl von 6809-Entwicklungsgeräten, die in Universitäten und technischen Hochschulen im Einsatz sind.

formationen über bestimmte Zustände des Prozessors enthalten. Beispielsweise zeigt nach einem Prozessorvorgang eins dieser Bits an, ob das in einem internen Register abgelegte Ergebnis Null ist. So lassen sich Wertetabellen schrittweise abarbeiten, indem die Gesamtzahl der Werte in ein Register geladen und nach jedem Bearbeitungsvorgang von dieser Zahl eine Eins abgezogen wird. Erreicht dieses Register den Wert Null, dann enthält das Bit des Condition Codes die Information, daß alle Tabellenwerte abgearbeitet sind und der nächste Befehl angesprochen werden muß. Mit dieser Befehlsart lassen sich Bedingungen (IF-Anweisungen) und Schleifen (FOR . . .NEXT, WHILE . . . WEND und REPEAT . . .UNTIL) konstruieren.

Viele Prozessoren enthalten eine *Zero Page*, die normalerweise aus den ersten 256 Speicherstellen (Hex 0000 bis 00FF) besteht und mit Acht-Bit-Adressen angesprochen wird. Befehle mit diesem Adreßformat sind kürzer und werden auch schneller ausgeführt. Der Prozessor 6809 verfügt über ein *Direct Page Register* im Acht-Bit-Format, das die zusätzlichen acht Adreßbits der Zero-Page-Adressen enthält. Da sich der Wert dieses Registers verändern läßt, kann die Zero Page in einen beliebigen Speicherbereich verlegt oder sogar mehrfach eingerichtet werden.

Maschinencodeprogramme bestehen aus Folgen von Anweisungen, die sich aus Daten und Adressen zusammensetzen. Einige Programmierer können diese Werte direkt in numerischen Werten angeben – diese Methode ist jedoch für die meisten zu kompliziert. Maschinencodeprogramme lassen sich weitaus einfacher in der Assemblersprache schreiben, die die Befehle durch mnemotische Kürzel und die Adressen und Daten durch Labels ersetzt. Wenn wir beispielsweise die Daten einer Speicherstelle in den Akkumulator laden möchten, können wir

```
STORE FCB 0
```

schreiben, um im Speicher einen Platz zu reservieren, auf den wir uns mit STORE beziehen und auf dem nun eine Null gespeichert ist. FCB ist eigentlich kein Befehl, sondern eine Anweisung, die dem Übersetzungsprogramm (das den Quellentext der Assemblersprache in den Maschinencode überträgt) mitteilt, daß es für das Wort STORE eine bestimmte Adresse einsetzen soll. Der dort gespeicherte Wert läßt sich mit

```
LDA STORE
```

in den Akkumulator laden.

Programme in Assemblersprache müssen vor ihrem Ablauf von einem Spezialprogramm – dem *Assembler* – übersetzt werden. Assembler sind nicht allzu kompliziert, da eine Assembleranweisung dem daraus entstehenden Maschinencode fast direkt entspricht. Die Übersetzung tauscht nur die Kürzel gegen Zahlen aus und setzt statt der Namen die entsprechenden Werte und Adressen ein.

Bridge per Computer

Bridge findet – ähnlich wie Schach – auf allen Schwierigkeitsebenen seine leidenschaftlichen Spieler. Es läßt sich jedoch auch ausgezeichnet auf Computer umsetzen.

Bridgeprogramme lassen sich in zwei Kategorien einstufen: Es gibt Lehrsysteme und Spielpakete.

Auf dem Markt werden eine ganze Reihe guter Lehrsysteme angeboten. Dabei ist die Bridgemasterserie von Aufbau und Gestaltung her eine der besten. Versionen dieses Paketes laufen auf dem Spectrum, ZX81, Acorn B, Electron und Commodore 64. Konzipiert wurde die Serie von dem ehemaligen Weltmeister Terence Reese, Bridgekorrespondent des Observer und des London Standard, von dem auch eine Reihe von Bridgebüchern stammen.

Alle Lehrprogramme beruhen darauf, den Anfänger durch eine Folge von vorprogrammierten Blättern zu führen. Hierin unterscheiden sie sich auch von den Spielsystemen, die die Verteilung der Karten dem Zufall überlassen. Da das Lehrprogramm „weiß", wie die einzelnen Blätter aussehen, kann es den Spieler schrittweise in die vielen Regeln und Übereinkünfte einführen, die Bridge zu einem interessanten und intelligenten Spiel machen.

Der „Bridgemaster" zum Beispiel enthält zwei Programme: das „Complete Learning Package For The Beginner At Bridge", das für den Anfänger gedacht ist, sowie „Expert Bridge", mit dem der Profi seine Spieltechnik optimieren kann. Der erste Kurs beinhaltet zwei Programmcassetten, zwei Anleitungscassetten, ein dünnes Anwenderhandbuch und ein Taschenbuch von Reese, „Begin Bridge".

Lehrprogramme ohne Kommentarcassette müssen entweder ein sehr umfangreiches Handbuch bieten oder ausführliche Kommentare auf dem Bildschirm darstellen. Es gibt allerdings Bridgeprogramme, die davon ausgehen, daß der Anwender das Spiel schon kennt und nur darauf abzielt, seine Spieltechnik entscheidend zu verbessern.

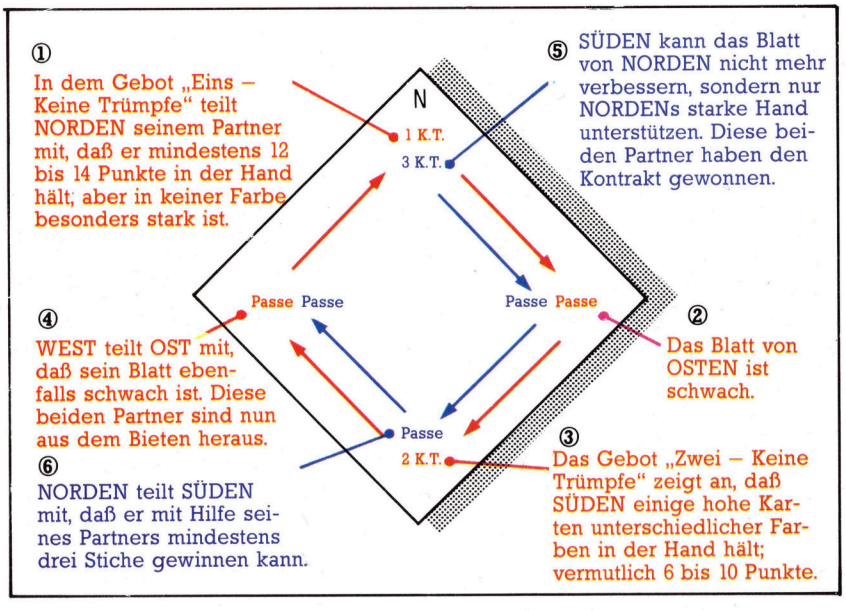
Alleinspieler und Strohmann

Bei einem Bridgespiel werden die Karten an vier Spieler verteilt, die sich in zwei Zweierteams aufteilen. Brideprogramme bezeichnen die Spieler im allgemeinen als Norden, Süden, Westen und Osten. Vor dem Spiel wird gereizt, wobei die Spieler die Spielstärke ihres Blattes angeben, ohne ihre Karten zu zeigen. Die Stärke eines Blattes wird entweder von der Zahl der hohen Karten oder der Anzahl Karten einer Farbe oder durch beides bestimmt. Der Gewinner des Reizens bestimmt den „Kontrakt". Damit wird die Trumpffarbe und die Anzahl der Stiche festgelegt, die der Gewinner erreichen muß. Der Gewinner des Reizens heißt „Alleinspieler" und seine Partner „Dummy" bzw. Strohmann – seine Karten werden aufgedeckt. Dem Reizen folgt das Ausspielen der Karten. Ob der Spieler Punkte gewinnt oder verliert, hängt nun davon ab, ob er die im Kontrakt festgelegte Anzahl Stiche erhält.

Alle Lehrprogramme enthalten das Reizen, bei dem der Spieler sowohl gewinnen als auch verlieren kann. Bei „Bridgemaster" gibt der Spieler ein Gebot ab, nachdem seine Karten auf dem Bildschirm dargestellt wurden. Danach zeigt der Computer sein Gebot an, das von dem des Spielers abweichen kann. Wie auch bei vielen anderen Lehrsystemen akzeptiert Reeses Programm nur das vorprogrammierte Gebot (oder Spiel). Dabei kann es durchaus passieren, daß ein völlig akzeptabler Spielvorschlag des Anfängers zurückgewiesen wird, wenn er mit dem vorprogrammierten Spielverlauf nicht übereinstimmt.

Außer der Anzeige der Blätter und dem Reizen steuert das Programm nun das eigentliche

Beim Reizen gibt es folgende Kartenwerte: As – 4; König – 3; Dame – 2; Bube – 1. Die roten Linien zeigen den ersten Durchgang an und die blauen den zweiten.



Spiel. Dabei sind die Karten häufig um ein Quadrat in der Bildschirmmitte angeordnet, dessen vier Seiten mit N, O (E), S, W beschriftet sind. Die Blätter von Norden und Süden werden vollständig dargestellt und die vier Farben in folgender Anordnung gezeigt: Pik, Herz (Coeur), Karo und Kreuz (Treff). Daneben erscheinen jeweils die Kartenwerte.

Sie sind immer Süden, wobei entweder Norden oder Süden der Dummy ist. Unabhängig davon, wer von beiden der Gewinner des Reizens ist, spielen Sie beide Blätter. Dies ist bei dem echten Bridgespiel natürlich nicht möglich. Dort können Sie den gesamten Abend spielen, ohne je einen Kontrakt zu gewinnen.

Dieser Unterschied ist für Lehrprogramme jedoch nicht wichtig. Zwar gibt es Verteidigungsspiele, in denen die gegnerische Partei das Reizen gewinnt, doch gibt es kein Programm, in dem der Alleinspieler Osten oder Westen ist. Wenn Sie „mauern", sich also um das Reizen „drücken" – was beispielsweise in dem Programm von CP-Software für den Spectrum möglich ist – und dabei bewußt erlauben, daß Osten oder Westen Alleinspieler wird, gibt das Programm eine Meldung aus, daß es unter diesen Voraussetzungen nicht spielen kann, und fordert Sie auf, über die R-Taste ein neues Blatt zu generieren.

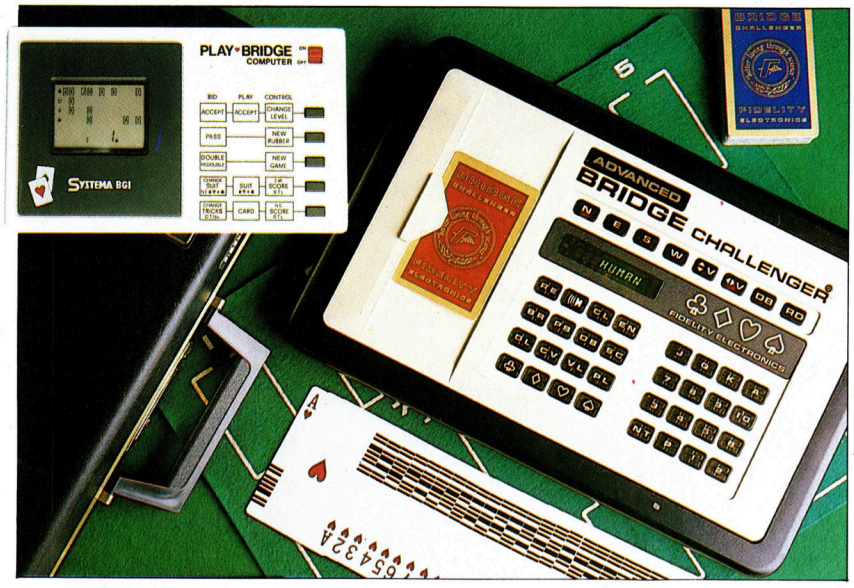
Nach Beendigung des Reizens wird das Ausspielen angezeigt. Je nach Gewinn des letzten Stichs spielt nun der Computer (für O oder W) oder der Spieler (für N oder S) die nächste Karte aus. Dabei erscheinen die Karten von O und W nacheinander auf dem „Tisch" – wie bei einem echten Spiel.

Computerspiele haben noch eine ganze Reihe weiterer Fähigkeiten. Der Bridgemaster beispielsweise gibt dem Spieler vor jedem Spiel die Wahl zwischen vier Alternativen: Bei „P" bestimmt der Spieler die auszuspielenden Karten für N und S, „A" spielt die Karten automatisch aus, „H" zeigt alle vier Blätter auf dem Bildschirm an, und „D" verteilt die Karten neu. Fast alle Bridgeprogramme verfügen über diese Möglichkeiten.

Analyse nach Spielende

In Verbindung mit Reeses Kommentar ist besonders das automatische Spiel für den Anfänger interessant, da er hier nicht durch das Ausspielen der eigenen Karten abgelenkt wird. Alle Spiele lassen sich wiederholen. Auch normale Spielprogramme haben diese Möglichkeit, doch dient sie hier nicht Lehrzwecken, sondern der Analyse nach beendetem Spiel.

Spielprogramme ohne Lehrsystem haben im wesentlichen die Aufgabe, die Spieltechnik eines erfahrenen Spielers zu verbessern. So enthält das Paket „Expert Bridge" der Bridgemasterserie einen Fortgeschrittenenkurs, der speziell auf Blätter mit hohen Karten, Schlemm-Reizen und Blockadespiele eingeht.



Ein weiteres Paket, das für den erfahrenen Spieler interessant ist, kommt von Alligata Software und wird für den Oric, Acorn B, Electron und Commodore 64 geliefert. Dieses Programm setzt eine genaue Kenntnis von Bridge voraus. Da kein Handbuch mitgeliefert wird, sollten Spieler mit schlechtem Gedächtnis die Spiele mitschreiben.

Das Bridgeprogramm von CP-Software für den Spectrum liefert sehr interessante Spiele. Zwar spielt es kein Blatt als Alleinspieler und kann daher auch keine Verteidigungsspiele trainieren, da aber 98 Prozent aller Bridgespieler Amateure sind, die jedes Blatt gleich gern spielen, fällt dieser Nachteil nicht sehr ins Gewicht.

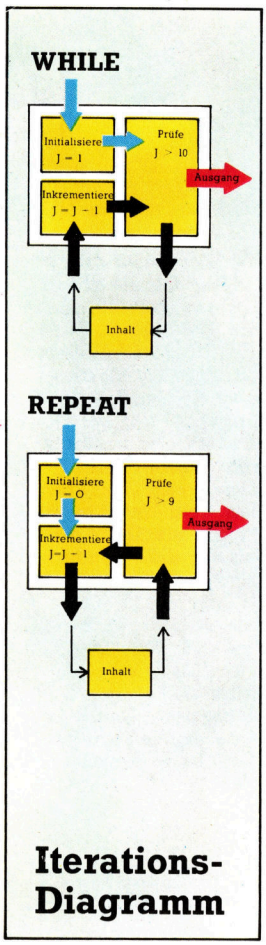
Beide Arten von Bridgeprogrammen – Lehr- und Spielsysteme – eignen sich ausgezeichnet zum Erlernen des Spiels oder zur Verbesserung schwacher Punkte. Bridge verlangt viel: ein gutes Gedächtnis, Fähigkeiten, zu analysieren und mit einem Partner zu spielen, zu wissen, wann Vorsicht und wann Risiko angebracht sind, ein „Poker Face" und einen kühlen Kopf. Die in diesem Artikel erwähnten Spiele können zwar ein Bridgespiel nicht perfekt simulieren, doch bieten sie interessante und anregende Übungsmöglichkeiten.

Die beiden im Bild gezeigten Geräte sind Bridgecomputer, rechts das Tischspiel „Advanced Bridge Challenger" von Fidelity Electronics. Der Bridge Challenger ist nur für erfahrene Spieler bestimmt und erfordert viel Übung und Konzentration. Sein Hauptnachteil ist die winzige Anzeige mit nur acht Zeichen. Die Maschine links im Bild ist der „Play-Bridge Computer" von Systema. Play-Bridge läßt sich in wenigen Minuten beherrschen. Er hat zwar kaum hochentwickelte Fähigkeiten, bringt aber viel Spaß.

Es gibt eine ganze Reihe von Bridgeprogrammen, die von Lehrsystemen bis zu Paketen reichen und die die Geschicklichkeit von fortgeschrittenen Spielern herausfordern. Das Bild zeigt (von links nach rechts) den Bridgemaster (für Spectrum, ZX81, Acorn B, Electron und Commodore 64), Bridge von Alligata Software (für Oric, Acorn B und C 64) und Bridge Player von CP-Software (für den Spectrum).

Schleifenstruktur

Flußdiagramme sind eine wichtige Technik bei der Programmgestaltung, doch die übliche Schreibform ist nicht immer ausreichend, vor allem dann, wenn es sich um komplizierte Schleifenstrukturen handelt.



Iterations-Diagramm

Wiederholung – oder Schleifenbildung – ist eine der grundlegenden Strukturen jeder Programmiersprache. Wir erwähnten bereits an anderer Stelle, daß immer dann im Algorithmus eine Schleife zu verwenden ist, wenn eine Entscheidung den Fluß der Steuerungsdaten umlenken soll.

Da Schleifen eine wichtige Strukturform darstellen und circa 60 Prozent der gesamten Prozessor-Aktivität ausmachen, ist es sehr wichtig, sie einmal detaillierter zu betrachten. Im Mittelpunkt der Betrachtung stehen ihre Einwirkung auf ein grundlegendes Programm, also eine Algorithmus-Struktur, und die verschiedenen Methoden, mit denen sie konstruiert und klassifiziert werden können.

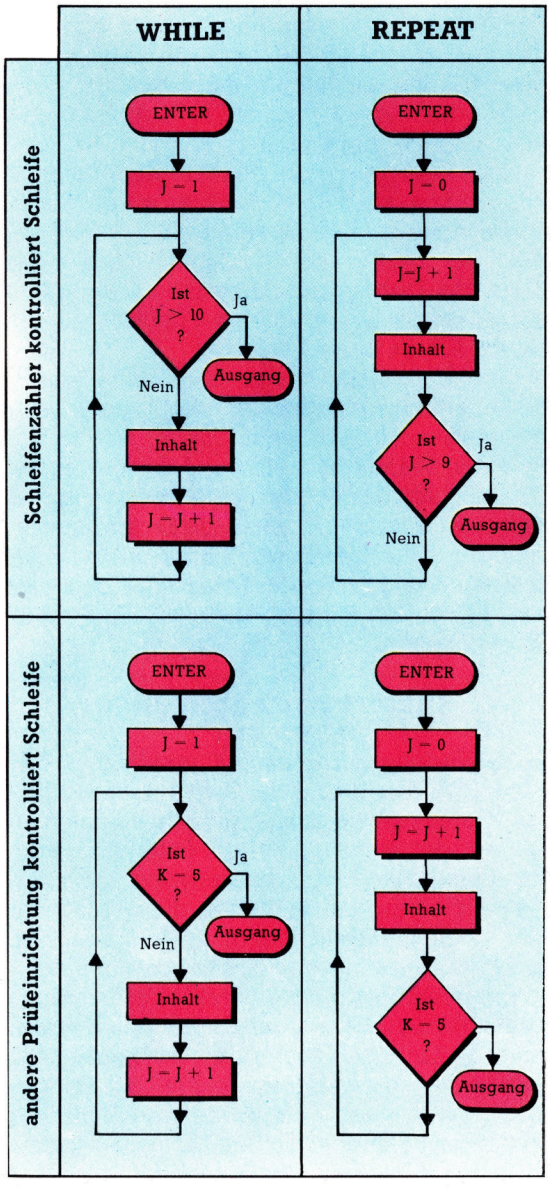
Schleifen werden häufig in zwei Kategorien eingestuft, abhängig von der Ähnlichkeit zu den beiden Hochsprachen-Schleifenstrukturen „REPEAT ... UNTIL" und „WHILE ... ENDWHILE", die beide in PASCAL verwendet werden. Die REPEAT-Schleife ist in zahlreichen BASIC-Versionen vorhanden. Die beiden Strukturen unterscheiden sich in der Position des Schleifenausgang-Tests: Bei einer REPEAT-Schleife erfolgt die Überprüfung am Ende der Schleife, wogegen sie bei einer WHILE-Schleife zu Beginn positioniert ist. Dieser erste Unterschied ist im Fluß-Diagramm deutlich zu sehen.

Eine andere Möglichkeit der Schleifen-Klassifizierung hängt davon ab, ob die Variable, die als Schleifenzähler dient, im Schleifenausgang-Test verwendet wird oder ob ein anderes Element zur Steuerung und Kontrolle des Schleifenendes benutzt wird. In einem Flußdiagramm konventioneller Art ist das nur undeutlich zu erkennen.

Klare Darstellung benötigt

Es gibt jedoch eine deutlichere Darstellungsform, „Iterations-Diagramm" genannt, die den Beginn einer Schleife klar kennzeichnet und die Verwirrung bei der Unterscheidung von Schleifen und Verzweigungen beseitigt. Sie besteht aus drei miteinander verbundenen Kästen. Der erste zeigt die Initialisierung des Zählers. Der zweite dient zum Zählen selbst, wogegen der dritte die Schleife beim Ausgang überprüft. REPEAT- und WHILE-Funktionen können in dieser Form dargestellt werden. Die Datenflußrichtung ist dabei verschieden. Der Test-Kasten zeigt an, ob die Schleifen zählerkontrolliert arbeiten oder nicht. Diese Punkte werden im Diagramm deutlich sichtbar.

In einer REPEAT-Schleife fließt die Kontrollfunktion folgendermaßen: „Initialisiere – Aussage – Prüfe – Aussage –Prüfe". Die WHILE-Schleife läuft so ab: „Initialisiere – Prüfe – Aussage – Prüfe – Aussage". Dies ist an der Richtung der Kontroll-Pfeile des Iterations-Diagramms zu erkennen.

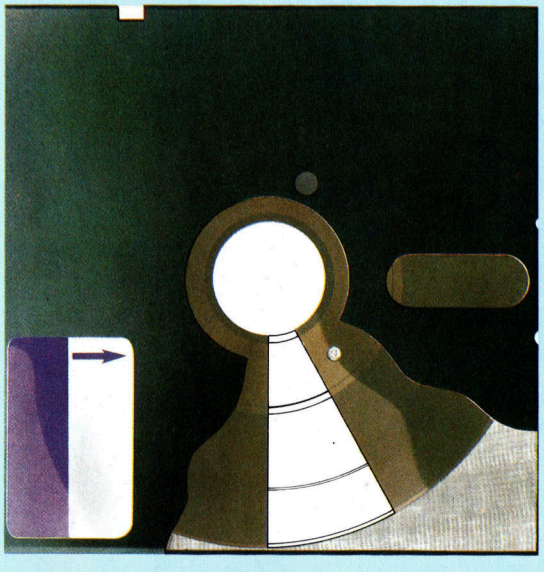


Fachwörter von A bis Z

Floppy Disk = Diskette
Im Laufe der Zeit sind verschiedene Diskettendurchmesser in Gebrauch gekommen: die 8-Zoll-Floppy, die 5 1/4-Zoll-Minifloppy sowie die 3-Zoll- und die 3 1/2-Zoll-Microfloppy. Die Datenträger bestehen aus ferromagnetisch beschichteten, weniger als 1/2 mm starken Kunststoffscheiben. Die Diskette befindet sich in einer Schutzhülle, die mit einem radialen Schlitz („Fenster") versehen ist. Beim Betrieb rotiert die Scheibe mit etwa 300 Upm, während der Schreib/Lese-Kopf über dem Fenster hin- und herfährt. So ist jeder Punkt der Oberfläche schnell erreichbar. Eine Aussparung im Hüllenrand dient als Schreibsperre: Vor dem Beschreiben der Diskette tastet das Laufwerk ab, ob diese Kerbe geschlossen oder frei ist.

Der Schreib/Lese-Kopf arbeitet wie bei einem Magnetbandgerät: Beim Schreiben werden die Magnetisierungsbezirke in der Plattenbeschichtung unterschiedlich ausgerichtet, und beim Lesen wird deren Orientierung abgetastet. Die Information wird auf konzentrischen Spuren (Tracks) aufgezeichnet, die ihrerseits in Sektoren unterteilt sind. Bei Disketten mit „doppelter Dichte"

Die Abbildung zeigt eine 5 1/4-Zoll-Diskette, wobei deutlich das Schreib/Lese-Fenster, das Indexloch und die Schreibsperren-Kerbe zu erkennen sind.



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.

(Double Density = DD) können bis zu 80 Spuren angelegt werden, bei „einfacher Dichte" (Single Density = SD) im allgemeinen 40 Spuren. Die Sektorgrenzen sind bei der sogenannten Hard-Sektorierung durch fotoelektrisch abtastbare Löcher (eins pro Sektor) am Innenrand der Diskette markiert. Bei „Soft-Sektorierung" gibt es dagegen nur ein einziges Indexloch am Anfang des ersten Sektors, und der Beginn jedes folgenden ist nur durch die Magnetisierung gekennzeichnet.

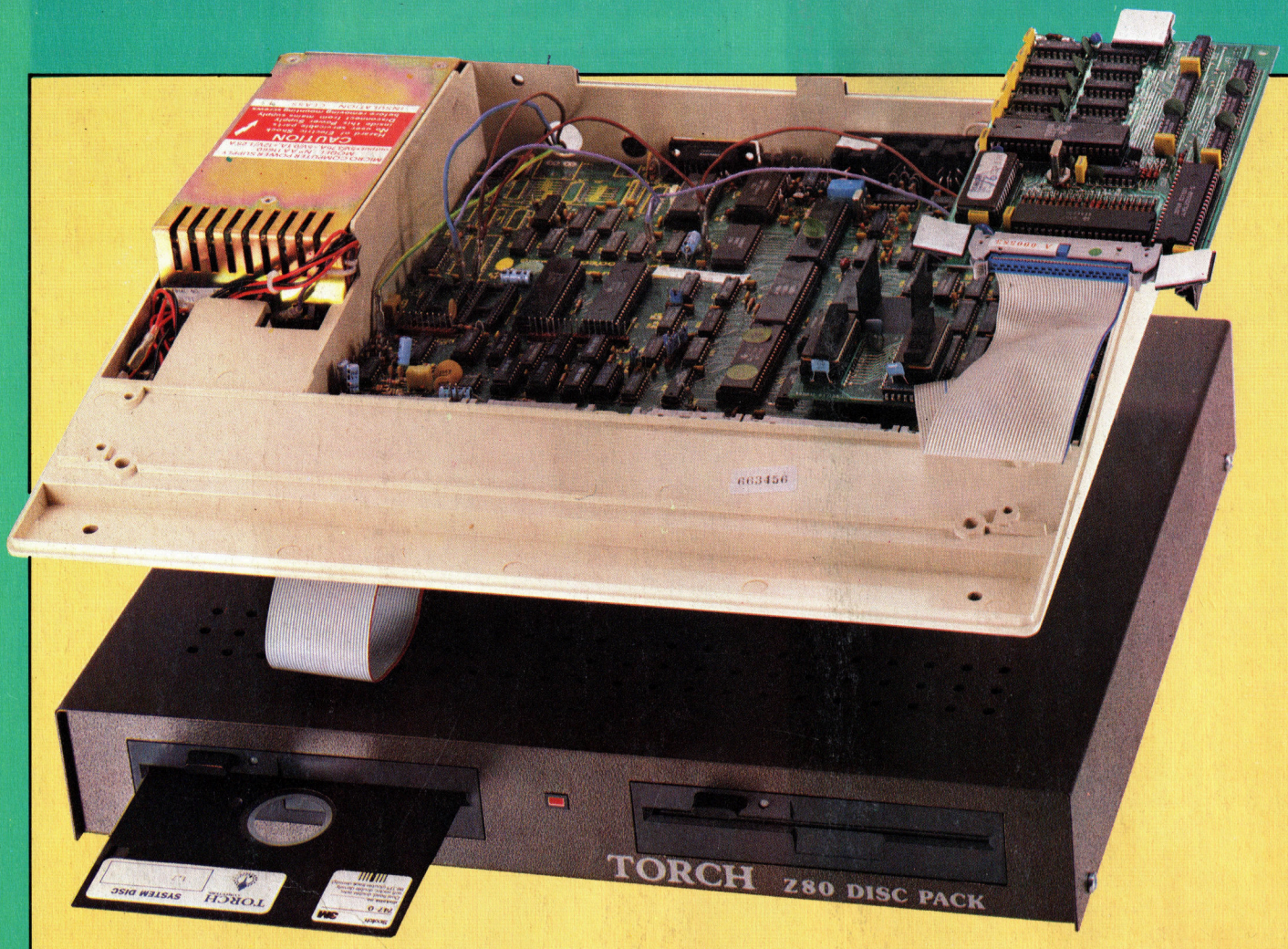
Flowchart = Flußdiagramm
Grafische Veranschaulichungen des Ineinandergreifens von Datenoperationen und Steuerungsvorgängen werden häufig als „Flußdiagramm" bezeichnet, obwohl vielfach nur Verknüpfungen statt des Programmflusses dargestellt sind. Ausdrücke wie „Prozeßdiagramm" oder „Datengrafik" wären im Grunde angemessener. Das übliche Flußdiagramm besteht aus Kästchen verschiedener Form, die Programmschritte wie Ein/Ausgabe, Entscheidungen und Wertzuweisungen beschreiben. Diese Kästchen werden durch Flußlinien mit Pfeilen verbunden, die die Abfolge der einzelnen Operationen verdeutlichen. Die meisten Autodidakten benutzen Diagramme dieser Art.

Jedoch verlieren rein grafische Lösungsansätze an Bedeutung gegenüber einer präzisen Festlegung der Anforderungen, der Datenspezifikationen und einer ergänzenden Funktionsbeschreibung.

Flow Control = Datenübertragungssteuerung
Beim Echtzeit-Datenverkehr sind die optimalen Übertragungsraten von Sender und Empfänger meist unterschiedlich. Damit langsame Empfangsgeräte nicht „überfüttert" werden, muß die Übertragung gesteuert werden – entweder vom Sender aus oder von einem Kontrollorgan im Übertragungsnetz. Das einfachste Verfahren ist die „End-to-End"-Steuerung, bei der die Kapazität des Empfängers die Übertragungsrate bestimmt. Eine andere Methode läuft unter der Bezeichnung „Hop-by-Hop" (etappenweise). Dabei wird die Datenrate für jede Teilstrecke des Übertragungsnetzes individuell durch die Kapazität des betroffenen Netzknotens bzw. Übermittlungskanals festgelegt.

Format = Format
In der Datenverarbeitung versteht man darunter im allgemeinen eine Anweisungsstruktur oder die Informationsstruktur auf Datenträgern. Bei Disketten beispielsweise gibt das Format an, wie Spuren und Sektoren auf der Magnetschicht angelegt sind. Aufgrund der unterschiedlichen Formatierung sind auf einem Laufwerk beschriebene Disketten von einem anderen meist nicht lesbar. Mit dem Aufkommen universeller Betriebssysteme wie CP/M und MS-DOS ist die Situation allerdings günstiger geworden, weil Software für diese Systeme oft vom gleichen Diskettenformat Gebrauch macht.

Bildnachweise
1205: Steve Cross
1206, 1210, 1213, 1218, 1219, 1220, 1223: Kevin Jones
1208, 1226: Crispin Thomas
1214, 1215, 1221, 1222: Liz Dixon
1217: Liz Heaney
1227: John Clementson, Dimension Graphics
1231: Ian McKinnell
U3: David Weeks



+ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +

computer kurs

Heft 45

Erweiterung

Das „Disc Pack“ (Foto oben) ist eine Erweiterung für den Acorn B: Zusätzliche 64K RAM, zwei doppelseitige Floppy-Laufwerke und CP/M-Kompatibilität.

Listenreich

Bei der Planung vielschichtiger Projekte werden Listen benötigt; „Brainstorm“ von Caxton Software leistet hier Hilfestellung.

Befehlseingabe

Weiter geht es mit unserem Abenteuer-Spiel: die Befehls-Routinen.

Gut organisiert

Der „Psion Organiser“ ist ein Taschencomputer mit vielen Möglichkeiten, Daten zu speichern sowie abzurufen, und ist einfach zu bedienen.