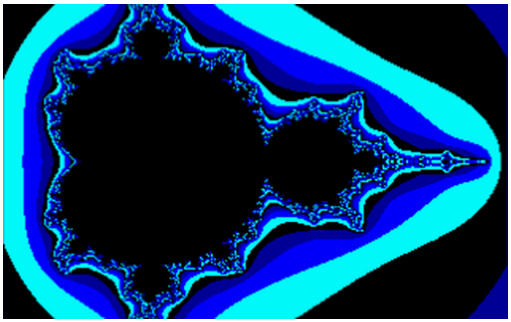
Matthias Uphoff

Das Software-Experiment

Experimentelle Mathematik, Simulationen
und künstliche Intelligenz
auf dem Schneider CPC



DMV-Verlag 1988 · HTML-Ausgabe 2002 · PDF-Ausgabe 2026

Die Artikelserie *Das Software-Experiment* erschien ab Sommer 1986 in der Zeitschrift *PC International*; 1988 gab sie der DMV-Verlag als Ringbuch mit Programmdiskette heraus. 2002 hat Matthias Uphoff die Texte eingescannt und als HTML-Seiten neu herausgegeben (siehe das Nachwort).

Diese PDF-Fassung wurde 2026 aus der HTML-Ausgabe gesetzt (`https://www.michael-wessel.info/retro/CPC/SoftEx/Inhalt.htm`). Der Text folgt ihr einschließlich der damaligen Rechtschreibung; offensichtliche Lesefehler der Texterkennung wurden berichtigt, soweit sie auffielen. Die Programme stehen auf der Diskette `SOFTEX.DSK`, die dort ebenfalls zu finden ist.

Satz: LaTeX, mit Claude Code (Anthropic) für Michael Wessel.

Inhaltsverzeichnis

Vorwort (1988) **vi**

Nachwort (2002) **viii**

1 Komplexe Grafik **1**
Herr Ober, bitte Zahlen! 2
Chaos im Computer 5
Kommt Zeit, kommt Grafik 8
Die Bedienung von KOMGRA.BAS 10

2 Das Spiel des Lebens **14**
Am Anfang war der Punkt... 15
Der Zellautomat 17
Das Spiel des Lebens 19
Evolution im Computer? 23

3 Das Ökologie-Experiment **25**
Das Wolf-Schaf-Gras-Modell 26
Fressen und gefressen werden 28
Das Öko-Programm 30
Katastrophen im Computer 31
Der ökologische Balanceakt 33
Software-Autor gesucht 34

4 Der Computer als Stratege **36**
Von Meistern und Mustern 37
Let's go bang! 39
Idiotisch oder genial? 40
Das Parameter-Tuning 45

5 Mit roher Rechengewalt **48**
Kalaha – die Guten ins Töpfchen... 50
Die Zugbewertung: Etwas kurzsichtig 53

Baumsuche mit Minimax 54
Rekursion – auch in BASIC! 56
Die Alpha-Beta-Baumschere 59

6 Der Computer lernt 62
Minischach: Klein, aber oho! 63
Schwer von Begriff? 65
Gepackte Daten: Dicht an dicht 66
Der Zuggenerator 69
Reduzierte Zuglisten 70
Der programmierte kleine Bruder 72

7 Der Computer als Wissenschaftler 76
Erkenntnis statt Programmierung 78
Modell gesucht 79
Der große Bluff: Wordmaster 81
Der allgemeine Problemlöser 84
Ohne Bier wird Gerd zum Tier! 85

8 Wortketten-Problem gelöst! 88
Wo liegt das Problem? 90
Schildkröte mit Rückwärtsgang 92
Der Weisheit letzter Schluß? 95
Alle auf einen Streich! 96

9 Entwicklungshilfe per Computer 98
1. Entwicklungshilfe als Thema des Erdkunde-Unterrichts 98
2. MORETTI im Schuleinsatz 99
3. Probleme der Sahelzone 100
4. Mathematisches Modell der Simulation MORETTI 103
5. Die Bedienung des Programms 103

10 Die Leser haben das Wort 108
Der eindimensionale Zellautomat 108
Neue Klänge mit dem CPC 110
Das wohltemperierte Klavier 112
Pascal läßt grüßen 114

Anhang **118**
1. Die Dateien des Software-Experiments 118
2. Der 664/6128-Emulator für den CPC 464 119

Vorwort (1988)

Ab Sommer 1986 wurde in der Zeitschrift PC International die Artikelserie *Das Software-Experiment* veröffentlicht, die schwerpunktmäßig die Themen experimentelle Mathematik, Simulationen und künstliche Intelligenz behandelte. Erklärtes Ziel war es dabei, den Leser nicht nur mit einer Anzahl Programms zum Abtippen zu versorgen, sondern ihn auch zum Mitmachen bzw. Mitdenken zu ermuntern.

Der große Erfolg der Serie war der Anlaß, das umfangreiche Material noch einmal in kompakter Form zu veröffentlichen und es damit allen CPC-Besitzern zugänglich zu machen, die ihren Rechner womöglich erst zu einem späteren Zeitpunkt erworben haben. Wie Sie sehen werden, stellt das Software-Experiment nicht nur eine außerordentlich vielseitige Programmsammlung dar, sondern bietet darüber hinaus eine geballte Ladung Know How, das in diesem Anleitungsbuch auf anschauliche und unterhaltsame Weise vermittelt wird.

Ein grafik- und musikfähiger Heimcomputer wie der CPC bietet eine Vielfalt kreativer Möglichkeiten, die zu einem großen Teil noch unerschlossen sind. Gerade die interessantesten Themen der Computergrafik und künstlichen Intelligenz erfordern jedoch ein fundiertes Wissen Über Datenstrukturen und Algorithmen, wenn man an dieser faszinierenden Entwicklung nicht nur als Konsument und Zuschauer, sondern auch als aktiver Forscher teilnehmen möchte. Deshalb werden in den folgenden Kapiteln die in den verschiedenen Programmen des Software-Experiments verwendeten Verfahrensweisen und Techniken ausführlich erläutert, um Anregungen für die Realisierung eigener Ideen zu geben.

Natürlich verfügt ein Heimcomputer nicht über die Kapazität moderner Großrechenanlagen, wie sie an Universitäten und wissenschaftlichen Instituten zu finden sind. Trotzdem lassen sich dem CPC mit seinem leistungsfähigen Basic, seiner Farbgrafik und den 64 KByte Speicher bereits erstaunliche Bilder und Intelligenzleistungen entlocken. Wie Sie sehen werden, sind die im *Software-Experiment* behandelten Themen alles andere als trivial; die bei der Schilderung von KI-Problemen beliebten Minimalbeispiele wie etwa das übersichtliche, aber dafür reichlich langweilige Tic-Tac-Toe-Spiel wer-

den Ihnen hier erspart bleiben.

Trotz des angestrebten Lerneffekts sollen beim Software-Experiment Spiel und Spaß nicht zu kurz kommen. Die ursprünglich auf den Abdruck als Listing zugeschnittenen Programme wurden noch einmal überarbeitet, so daß sie sich jetzt Ready to Run auf Ihrer Diskette bzw. Kassette befinden und ohne Einschränkungen auf allen CPC-Typen (464/664/6128) laufen.Nur für CPC 464-Besitzer ist bei einem Programm (MORETTI.BAS) der 664/6128-Emulator (EMU.BAS) notwendig, der als Bonus mitgeliefert wird. Die Beschreibung des Emulators finden Sie im Anhang.

Die meisten Programme können also auch ohne Kenntnis der theoretischen Grundlagen einfach nur als Spielprogramme genutzt werden. Da sie jedoch ursprünglich als flexible Experimentierflächen konzipiert wurden, enthalten sie im allgemeinen keine aufwendige Benutzerführung wie Hilfsbildschirme oder ähnliches. Aus diesem Grund empfiehlt es sich, in den dazugehörigen Kapiteln zumindest die Bedienungshinweise gründlich zu studieren.

Die Texte enthalten teilweise Hinweise, die sich auf bestimmte Programmzeilen bzw. -abschnitte beziehen. Es ist für das Verständnis förderlich, wenn Sie sich die Listings für das jeweilige Kapitel ausdrucken lassen, indem Sie nach dem Laden LIST #8 eingeben. Falls Sie keinen Drucker besitzen, so genügt es notfalls auch, die interessanten Abschnitte mit LIST auf den Bildschirm zu holen.

Nachwort (2002)

…und an dieser Stelle bricht das Vorwort ab – eine fehlende Seite war nicht mehr aufzufinden. Deshalb als Ersatz hier ein abschließendes Nachwort. Dass diese „Neuauflage“ der Artikelserie *Das Software-Experiment* in Form von HTML-Dateien überhaupt zustande gekommen ist, resultiert aus dem zufälligen(?) Zusammentreffen verschiedener Faktoren, die ich kurz aufzählen möchte:

Interessant ist es mitunter, in Google seinen eigenen Namen einzugeben und zu sehen, wo überall im Web man verzeichnet steht. Bei einem solchen Versuch traf ich auf einen Thread in einem CPC-Forum, in dem nach Artikeln und Programmen aus meiner Artikelserie geforscht wurde.

Einige Zeit später entdeckte ich, dass sehr gute CPC-Emulatoren als Freeware verfügbar sind. Das Ausprobieren schickte mich erst auf den Retro-Trip und dann in den Keller, wo nach einigem Herumwühlen drei CPC-Disks zum Vorschein kamen – eine davon enthielt sämtlichen Dateien zu der Artikelserie. Sie steckte in einer DIN A5-Ringbuch-Ausgabe des *Software-Experiments*, die der DMV-Verlag 1988 veröffentlicht hat, leider in äußerst bescheidener Qualität.

Nun wollte ich auch die alten Schätze noch mal in Aktion sehen. Da mir die Hardware-Ausrüstung fehlte, um die 3″-CPC-Disketten in Emulator-gerechte DSK-Imagefiles zu konvertieren, wandte ich mich auf gut Glück an einen der Thread-Teilnehmer… und fand in Michael Wessel einen sehr hilfsbereiten und interessierten Menschen, der vor allem noch echte CPCs besitzt. Mit diversen Hard- und Software-Problemen kämpfend, gelang ihm schließlich die Konvertierung der Disketten.

Derweilen gelangte ich zu der Überzeugung, dass auch die dazugehörigen Texte rettenswert wären. Also wurden 180 schlecht gedruckte Ringbuchblätter eingescannt und per OCR wieder in Textdateien verwandelt – ein mühsames und gleichzeitig reizvolles Unterfangen, das die Mentalität eines passionierten Museums-Restaurators erforderte. Eingefügt wurde noch etwas Bildmaterial – die Screenshots lieferte der CPC-Emulator *CaPriCe32* von Ulrich Doewich, auf dem sämtliche Programme des *Software-Experiments* problemlos liefen (mitunter war ich froh, dass der Emulator optional eine höhere

Geschwindigkeit erlaubt als das originale CPC-Tempo – was waren wir damals geduldig! Eine Stunde auf das *Apfelmännchen* warten…). Vielen Dank an dieser Stelle für die ausgezeichnete Freeware!

Die aktuelle Version des Emulators ist hier[1] zu finden. Um die Programme des *Software-Experiments* auszuführen, muß zunächst über den Menüpunkt **Disk/A-Insert Disk** die beiliegende Image-Datei **SOFTEX.DSK** geladen werden. Die Eingabe des Befehls **Cat** mit anschließendem RETURN zeigt den Inhalt der Diskette an. Der Start eines Programmes erfolgt dann durch Eingabe des Basic-Befehls **Run „Programmname"** und Drücken von RETURN. Es empfiehlt sich jedoch, zunächst über den Menüpunkt **Tools/Remap Controls** das „Y" und „Z" des PC-Keyboards dem „Y" und „Z" der CPC-Tastatur zuzuordnen, um die voreingestellte englische Belegung an deutsche Verhältnisse anzupassen. Da das Tastaturlayout des CPC insbesondere bei Sonderzeichen und Spezialtasten stark vom PC abweicht, ist es ratsam, sich das dem Emulator als GIF-Datei beiliegende Bild der CPC-Tastatur anzuschauen und mit den Eintragungen unter Tools/Remap Controls zu vergleichen, da man sonst einige Zeichen, die nicht auf den gewohnten PC-Tasten liegen, erst nach einigem Herumprobieren findet.

Betrachtet man die Programme zum *Software-Experiment* aus heutiger Sicht, so wirken sie in ihrer äußeren Erscheinungsform recht altbacken – die Bedienung ist gelinde gesagt gewöhnungsbedürftig. Aber die Ergebnisse, die sie liefern, sind zu einem guten Teil immer noch interessant und überraschend. Und wer sich für die Themenbereiche experimentelle Mathematik, Simulationen und künstliche Intelligenz interessiert, wird dem dazugehörigen Lesestoff auch heute noch einiges abgewinnen können. Allen, die sich aus Nostalgie oder ungebrochener Freude an der Computer-Bastelei noch einmal dem Software-Experiment zuwenden, wünsche ich unterhaltsame Stunden und viel Vergnügen!

Matthias Uphoff

[1] `http://www.classicgaming.com/caprice` (Stand 2002).

KAPITEL 1

Komplexe Grafik

Nicht nur in der Arbeitswelt erzeugt die moderne Computertechnik revolutionäre Veränderungen. Auch Wissenschaftlern und Künstlern stehen in zunehmenden Maße Hilfsmittel zur Verfügung, deren Möglichkeiten sich heute noch gar nicht vollständig abschätzen lassen.

So ist es zum Beispiel mit dem Hochleistungs-Computer CRAY X-MP möglich geworden, die Entstehungsgeschichte unseres Universums zu rekonstruieren und auf einem Bildschirm grafisch darzustellen. Wer schon einmal mit der Differentialgleichung zu tun hatte, die die Bewegung nur eines Planeten um die Sonne beschreibt, kann ermessen, welchen Rechenaufwand die Wechselwirkungen zwischen einer wahrhaft astronomischen Anzahl von Himmelskörpern erfordern.

Der Wissenschaftler früherer Zeiten, der praktisch nur über den Rechenschieber und die Logarithmentafel als Werkzeug verfügte, wäre damit hoffnungslos überfordert gewesen. Erst durch die Mikroelektronik ist es möglich geworden, komplizierte mathematische oder physikalische Systeme durch Simulationen experimentell zu erforschen. Besonders bemerkenswert ist dabei, daß die Ergebnisse dieser Experimente nicht nur in abstrakter Form als Zahlen oder Gleichungen vorliegen, sondern teilweise faszinierende Bilder ergeben. Wie ein Fernrohr oder ein Mikroskop erlaubt der Computer Einblicke in Welten und Dimensionen, die dem menschlichen Auge sonst verschlossen blieben. Mathematisch-logische Strukturen werden sichtbar, erscheinen als Farben und Formen auf dem Bildschirm, können experimentell und spielerisch verändert werden – kein Wunder also, daß die lange Zeit als trocken verschrieene Mathematik enorm an Attraktivität gewonnen hat.

Nicht nur Künstler entdecken hier neue Motive und Ausdrucksmöglichkeiten; auch Jugendliche entwickeln zunehmend ein lebhaftes Interesse an dieser Wissenschaft, die erst durch das Medium Computer im wahrsten Sinne des Wortes anschaulich geworden ist. Als erstes Software-Experiment

soll deshalb ein Programm vorgestellt werden, das mit einer relativ einfachen mathematischen Formel eine Vielzahl verblüffender Bilder erzeugt, die unter dem Namen *Chaos-Grafiken* oder *Apfelmännchen* bereits für einiges Aufsehen gesorgt haben.

Die Apfelmännchen-Grafiken entstehen durch die bildliche Darstellung der Eigenschaften komplexer Zahlenfolgen – was sind überhaupt komplexe Zahlen? Wie ist es möglich, daß eine mathematische Formel Strukturen erzeugt, die durch ihre geordnete Unregelmäßigkeit fast schon organischen Charakter haben? Um der Antwort auf diese Frage näher zu kommen, beginnen wir mit einem kleinen Ausflug in die Arithmetik, die Theorie der Zahlen:

Herr Ober, bitte Zahlen!

Am Anfang der Entwicklung des Zahlbegriffs standen die natürlichen Zahlen 1, 2, 3, ... natürlich wohl deshalb, da sie nicht aus theoretischen Überlegungen entstanden, sondern direkt aus der Anschauung. Der Urmensch trat aus seiner Höhle und begann zu zählen: 1 Säbelzahntiger, 2 Säbelzahntiger... – nun, wenn auch vielleicht nicht genau so, aber so ähnlich müssen die natürlichen Zahlen entstanden sein. Und für viele Zwecke reichen sie wirklich aus! Wir können sie addieren oder multiplizieren, und das Ergebnis wird wieder eine natürliche Zahl sein. Der Bereich der natürlichen Zahlen ist also hinsichtlich dieser Operationen abgeschlossen, wie der Mathematiker sagt.

Probleme tauchen jedoch auf, wenn wir subtrahieren oder dividieren wollen. Manchmal geht es gut, aber manchmal auch nicht, wie die Gleichung x+ 5 = 3 zeigt; sie hat im Bereich der natürlichen Zahlen keine Lösung. Um diesen unbefriedigenden Zustand zu beenden, erfanden die Menschen die negativen Zahlen, um sie mitsamt der Null zur Menge der ganzen Zahlen zusammenzufassen, oder um es im Computerlatein auszudrücken: Die Integerzahl war geboren.

Aber gleichzeitig hatte der Mensch damit auch den Bereich der Anschauung verlassen. Die negativen Zahlen stellen abstrakte Gebilde dar, mit denen sich zwar gut rechnen läßt, die aber unser Vorstellungsvermögen bereits auf eine harte Probe stellen. So gab es zum Beispiel während meiner Studienzeit einen Mathematikdozenten, der wegen seiner miserablen Vorlesungen sehr unbeliebt war. Im Laufe des Semesters schmolz sein Publikum zusehends zu-

sammen, bis eines Tages nur noch kümmerliche drei Studenten anwesend waren – und um es kurz zu machen: Die Vorlesung war wieder dermaßen unverständlich, daß insgesamt fünf von ihnen fluchtartig den Raum verließen. ‚So', sagte der Prof mit finsterer Miene, ‚wenn jetzt noch zwei reinkommen, dann ist keiner mehr da!'

Negative Zahlen sind also irgendwie doof, wie es einer meiner Nachhilfeschüler kurz und bündig zum Ausdruck brachte, aber sie sind auch praktisch, und haben sich deshalb bis heute halten können. Das gleiche gilt sicherlich auch für die rationalen Zahlen, die Menge der Brüche, mit deren Hilfe man problemlos fünf Kühe auf sieben Bauernhöfe verteilen kann – theoretisch zumindest -, aber leider sind damit noch längst nicht alle Probleme beseitigt. Man kann nämlich sehr einfach und schlüssig nachweisen, daß sich die algebraische Gleichung

$$x^2 = 2$$

nicht durch einen Bruch lösen läßt.

Angenommen, es wäre so, so müßte sich x in der Form p/q darstellen lassen, wobei wir jetzt voraussetzen wollen, daß dieser Bruch bereits so weit wie möglich gekürzt wurde. p und q sind damit teilerfremd, und daran ändert sich auch nichts, wenn wir den Bruch quadrieren. Andererseits erhalten wir nach dem Einsetzen die Aussage

$$p^2/q^2 = 2$$

Das Ergebnis kann aber nur 2 lauten, wenn sich der Bruch kürzen läßt, was im Widerspruch zu unserer Annahme steht, sie muß deshalb falsch sein. Also blieb den Mathematikern nichts anderes übrig, als für diese und ähnliche Fälle neue Zahlen zu erfinden, die irrationalen Zahlen. Was bedeutet aber eigentlich irrational? Nehmen wir also ein Fremdwörterlexikon zur Hand, und dort finden wir wahrhaftig ‚*mit dem Verstande nicht erfaßbar, gefühlsbetont, triebhaft*'.

Sie merken schon – jetzt wird's gefährlich! Zwar habe ich persönlich an der Wurzel aus 2 noch keine triebhaften Züge entdeckt, doch den Mathematikern müssen diese Zahlen irgendwie unheimlich gewesen sein, denn sie wurden flugs mit den rationalen Zahlen zur Menge der reellen Zahlen vereint, die Sie auf dem CPC als *REAL-Typ* wiederfinden. Reell heißt soviel wie *‚realisierbar, einwandfrei, anständig'*, Sitte und Ordnung wären also wieder

hergestellt, und man könnte sich jetzt in Ruhe anderen Dingen zuwenden, wenn es nicht noch ein kleines Problem gäbe:

$$x^2 = -1$$

Das ist wirklich eine harte Nuß: Welche Zahl ergibt mit sich selbst multipliziert −1? Mit den reellen Zahlen kommt man keinen Schritt weiter, und auch unser schlauer CPC ist hier überfordert:

```
PRINT SQR(-1)
```

entlockt ihm nur ein ratloses *Improper Argument*.

Doch an dieser nahezu unüberwindlichen Schwelle wagten die Mathematiker einen mutigen Schritt. ‚Gut', sagten sie sich, ‚wir kennen die Lösung nicht, aber wir wollen einfach mal annehmen, daß eine existiert', und nannten diese geheimnisvolle Zahl i, für die definitionsgemäß gilt

$$i^2 = -1$$

Der Buchstabe i steht für imaginär (scheinbar, unwirklich), ein treffender Name für eine absolut abstrakte Zahl, die keine Entsprechung mehr in der Wirklichkeit hat. Doch wie Versuche zeigen, kann man mit der imaginären Zahl alle gewohnten Rechenoperationen durchführen, ohne daß es zu Widersprüchen kommt. Die Erweiterung des Zahlenbegriffs ist damit zumindest formal in Ordnung. Insbesondere ist es möglich, die Zahl i durch Addition oder Multiplikation mit reellen Zahlen zu verknüpfen, wodurch Gebilde wie etwa 3+2*i entstehen, oder in allgemeiner Form

$$c = x + y \cdot i$$

Solche Zahlen c, die aus einer reellen Zahl und einem Vielfachen der imaginären Grundeinheit zusammengesetzt sind, nennt man komplexe Zahlen. Dem berühmten Mathematiker Gauß ist der Nachweis zu verdanken, daß damit unser Zahlensystem endgültig abgeschlossen ist; jede algebraische Gleichung hat also zumindest eine komplexe Lösung.

Es fragt sich nur, warum man sich überhaupt mit solchen Hirngespinsten beschäftigen soll, die auf den ersten Blick nicht den geringsten praktischen Wert haben. Doch ähnlich wie die negativen Zahlen stellen die komplexen

Zahlen trotz ihrer Unanschaulichkeit für manche physikalischen oder mathematischen Problemstellungen ein sehr effektives Werkzeug dar. So lassen sich zum Beispiel einige Differentialgleichungen wesentlich einfacher lösen, wenn man von vornherein bestimmte Variablen als komplexe Zahlen betrachtet.

Nicht umsonst stellen höhere Programmiersprachen für wissenschaftliche Anwendungen (z.B. FORTRAN) neben INTEGER und REAL auch den Variablentyp COMPLEX zur Verfügung. Obwohl das Locomotive-Basic keine Anweisung wie DEFCOMPLEX kennt, kann man jedoch auch mit dem CPC in die Gefilde der höheren Mathematik vordringen. Jede komplexe Zahl c=x+y*i läßt sich nämlich durch ein ganz normales Zahlenpaar (x,y) darstellen, wobei die eine den Realteil und die andere den Imaginärteil repräsentiert. Indem wir den Rechner die Komponenten separat bearbeiten lassen, ersparen wir ihm die ominöse Wurzel aus −1, die ja sofort einen Error auslösen würde.

Solch ein Zahlenpaar läßt sich natürlich hervorragend in einem Koordinatensystem darstellen, wobei die x-Achse der reellen Zahlengeraden und die y-Achse der imaginären Geraden entspricht. Zeichnet man vom Ursprung des Koordinatensystems einen Pfeil bis zu dem Punkt, der die komplexe Zahl repräsentiert, so kann man sich noch veranschaulichen, was man unter dem Absolutbetrag dieser Zahl versteht: Er ist identisch mit der Länge des Pfeils, die mit dem Satz des Pythagoras berechnet wird:

$$L^2 = x^2 + y^2$$

Auf diese Weise ist es sogar möglich, komplexen Zahlen eine Größe zuzuordnen, die für die Apfelmännchen-Grafik noch eine wichtige Rolle spielen wird.

Chaos im Computer

Ordnung ist das halbe Leben, so sagt ein Sprichwort – und Unordnung die andere Hälfte, ergänzt der erfahrene Programmierer, der nach intensiver Suche die Systemdiskette zwischen ein paar alten Listings im Papierkorb wiederfindet. Doch Schreck beiseite: Heutzutage macht sich in den Naturwissenschaften mehr und mehr die Überzeugung breit, daß hochkomplizierte Strukturen wie lebende Organismen nur durch das Zusammenspiel von Ord-

nung und Chaos, von Gesetz und Zufall entstehen konnten. Und diese Theorie wird insbesondere durch die Ergebnisse einiger Computersimulationen gestützt!

Die exakten Naturgesetze geben der Welt einen festen Rahmen. Sind alle Anfangsbedingungen eines Systems bekannt, so kann man mit ihrer Hilfe zumindest theoretisch in allen Einzelheiten berechnen, was passieren wird. Doch die moderne Physik hat nachgewiesen, daß im Bereich der Elementarteilchen Abläufe stattfinden, die unbestimmt sind, und über die höchstens statistische Aussagen gemacht werden können. Unsere Welt besitzt also einen eingebauten Zufallsgenerator, der zuverlässig verhindert, daß sie in gleichbleibenden Mustern stagniert.

Jeder Spiele-Programmierer weiß die RND-Funktion seines Computers zu schätzen, den vorprogrammierten Unbestimmtheitsfaktor, der für Abwechslung und Spannung sorgt. Bei Bedarf produziert er eine Folge von Zahlen, denen beim besten Willen kein System mehr anzusehen ist. Und trotzdem handelt es sich nicht um echten Zufall, sondern nur um eine sehr gute Simulation! Natürlich kann ein Computer sich nicht intuitiv Zahlen ausdenken, sondern muß sie auf irgendeine Weise berechnen.

Eine Folge von Pseudo-Zufallszahlen wird im allgemeinen iterativ erzeugt, d.h. jede Zahl entsteht aus der vorherigen durch ein bestimmtes Bildungsgesetz, das allerdings vom Betrachter kaum nachzuvollziehen ist, es sei denn, er schaut sich im ROM-Listing um. Eine einfache Rechenoperation wie das Quadrieren der Zahl reicht hier natürlich nicht, das Bildungsgesetz wäre noch zu offensichtlich. Wesentlich mehr künstliches Chaos entsteht schon, wenn man zusätzlich noch die erste Zahl der Folge als konstanten Wert subtrahiert, wie das folgende Kurzprogramm zeigt:

```
10 xstart=1.5: x=xstart
20 x=x*x-xstart: PRINT x
30 GOTO 20
```

Obwohl die recht strengen Kriterien für professionelle Zufallsgeneratoren durch dieses einfache System noch nicht erfüllt sind, ist das Ergebnis schon recht beachtlich, und es wird noch interessanter, wenn wir die gleiche Iterationsvorschrift auf komplexe Zahlen anwenden. Um das Gesetz noch einmal exakt zu formulieren: Wenn die komplexe Zahl C der Startwert ist,

$$C_1 = C$$

dann berechnen wir alle weiteren Zahlen nach der Vorschrift

$$C_{n+1} = C_n^2 - C$$

Trotz dieser einfachen Darstellung ergibt sich in der Praxis einige Rechnerei. Das Quadrat einer komplexen Zahl muß mit Hilfe der binomischen Formel ermittelt werden:

$$C^2 = (x + yi)^2 = x^2 + 2xyi + y^2i^2$$

Da i * i = −1 ergibt, können wir noch vereinfachen,

$$c^2 = x^2 + 2xyi - y^2$$

und dann Real- und Imaginärteil trennen,

$$\mathrm{Re}(c^2) = x^2 - y^2; \qquad \mathrm{Im}(c^2) = 2xy$$

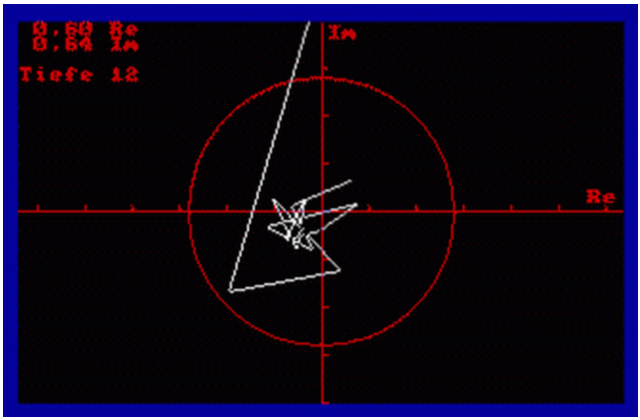
womit deutlich wird, daß diese Operation insgesamt eine Subtraktion und vier Multiplikationen erfordert. Das Subtrahieren zweier komplexer Zahlen ist glücklicherweise etwas weniger aufwendig; hier werden einfach die Real- und Imaginärkomponenten der beiden Zahlen separat voneinander abgezogen.

Um zu verstehen, wie diese Rechenoperation das Apfelmännchen und Konsorten erzeugen, sollten Sie sich zunächst das Programm *Komplexe Folgen* (KOMFOL.BAS) vornehmen. Nach dem Programmstart erscheint auf dem Bildschirm ein Koordinatenkreuz, das durch die reelle und imaginäre Achse der Zahlenebene gebildet wird. Zusätzlich ist ein Kreis und ein heller Punkt zu sehen, der mit den **Cursortasten** gesteuert wird und den Startwert repräsentiert. Oben links werden Real- und Imaginärteil des augenblicklichen Standortes angezeigt.

Wenn Sie nun mehrmals die SPACE-Taste drücken, berechnet das Programm nach der oben angegebenen Methode die jeweils nächste Zahl der Folge und zieht eine Linie zur neuen Position. Je nach Startwert ergibt sich ein wilder Zickzack-Kurs, der allerdings beendet wird, wenn das Abbruchkriterium

$$x^2 + y^2 \geq 8$$

erfüllt ist, also der Absolutbetrag einer Zahl der komplexen Folge außerhalb des Kreises mit dem Radius Wurzel 8 liegt. Mit DEL können sie das Programm neu starten und einen anderen Anfangswert probieren.



Die Ergebnisse dieser Experimente können sehr verschieden ausfallen. Einige Folgen verlassen schon nach wenigen Iterationen den Kreis, andere erst nach vielem Hin und Her (probieren Sie 0.24 / −0.64!), und manche, deren Anfangswert nahe genug bei 0 liegt, rennen sich fest und erreichen nie das Abbruchkriterium.

Bemerkenswert ist weiterhin, daß sich die Grundmuster in verkleinerter Form an verschiedenen Stellen der Grafik wiederholen. Der große Zusammenhang spiegelt sich im Kleinen wieder – ein wichtiger Grundgedanke der Esoterik und anderer, ganzheitlich orientierter Philosophien, der in diesen Bildern auf verblüffende Weise seinen Ausdruck findet.

Kommt Zeit, kommt Grafik

Wie wird die komplexe Grafik nun auf dem CPC realisiert? Gehen wir von den technischen Voraussetzungen aus: Der Grafikbildschirm stellt ebenfalls ein Koordinatensystem dar, das in MODE 1 insgesamt 320 * 200 Bildpunkte enthält.

Wenn wir einen Bereich der komplexen Zahlenebene auswählen und auf den Bildschirm projizieren, entspricht jeder Punkt einer Zahl. Sie stellt den

Startwert einer komplexen Folge dar, die berechnet wird, bis das Abbruchkriterium

$$x^2 + y^2 \geq 8$$

erreicht ist. Für ein komplettes Bild müssen also 64000 Folgen erzeugt werden, für jede Folge abhängig von der Iterationstiefe einige -zig komplexe Zahlen, und das auch noch in der zeitaufwendigen Fließkomma-Arithmetik. Es ist also leicht einzusehen, daß der Zeitaufwand für ein Bild mehrere Stunden betragen kann. Wie Experimente mit dem Programm KOMFOL.BAS zeigen, gibt es allerdings auch Folgen, die nie das Abbruchkriterium erreichen. Deshalb wird von vornherein eine maximale Tiefe als feste Grenze festgelegt, damit sich das Programm nicht in einer Endlosschleife verfängt.

Alles in allem zeigen die komplexen Folgen also ein recht chaotisches Verhalten, und genau das wollten wir ja auch erreichen. Doch jetzt der Clou: Eines Tages kam der Mathematiker Benoit Mandelbrot im IBM Research-Center New York auf die geniale Idee, die Punkte der komplexen Zahlenebene einzufärben, und zwar in Abhängigkeit von der Iterationstiefe bis zum Abbruch – und er staunte nicht schlecht, als er das Ergebnis sah: Das erste Apfelmännchen war geboren! Die feinen Strukturen am inneren Rand der Figur hielt er übrigens zunächst für Unschärfen, erzeugt durch Rundungsfehler. Erst die Berechnung vergrößerter Ausschnitte förderte die erstaunliche Tatsache zu Tage, daß diese Grafik eine unendliche Anzahl kunstvoller Unterstrukturen enthält.

Nach Ermittlung der Anzahl der Iterationsschritte bis zum Abbruch stehen in MODE 1 insgesamt 4 Farben zur Verfügung (PEN 0 bis PEN 3), um die Punkte in Abhängigkeit von der erzielten Rechentiefe mit dem PLOT-Befehl einzufärben. Für die maximale Tiefe kann z.B. PEN 0 verwendet werden; ansonsten sorgt die Formel Farbe = Tiefe MOD 4 dafür, daß sich die Farben zyklisch wiederholen.

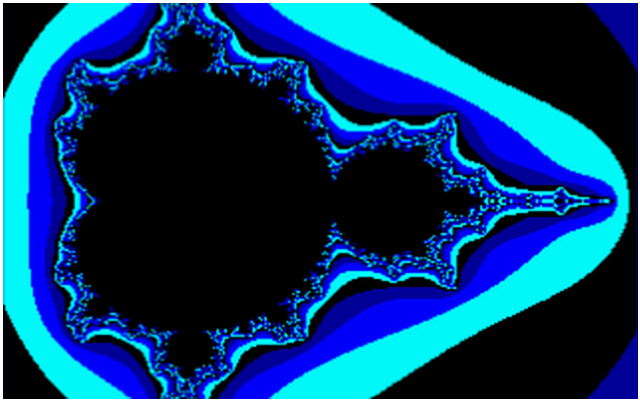
Die Berechnungsroutine für das Programm *Komplexe Grafik* (KOMGRA.BAS) wurde in Maschinensprache geschrieben, um eine hohe Geschwindigkeit zu erreichen. Da man den Algorithmus in dieser Form kaum wiedererkennen kann, finden Sie hier noch ein Basic-Listing, das im wesentlichen dem Maschinenprogramm entspricht. Für den praktischen Einsatz wäre es allerdings viel zu langsam.

```
10 MODE 1
20 xl = -0.7: xr = 2.1:REM linker und rechter Rand reell
30 yu = -1: yo = 1:    REM unterer und oberer Rand imaginär
40 MaxTiefe = 30:      REM maximale Iterationstiefe
50 dx = (xr - xl)/319: REM X-Schrittweite
60 dy = (yo - yu)/199: REM Y-Schrittweite
70 y = yu
80 FOR Zeile = 0 TO 199
90    x = xl
100   FOR Spalte = 0 TO 319
110     Tiefe = 0: re2 = x*x: im2 = y*y
120     WHILE (Tiefe < MaxTiefe) AND (re2 + im2 < 8)
130       re = re2 - im2 - x: im = 2*re*im - y
140       re2 = re*re: im2 = im*im: Tiefe = Tiefe + 1
150     WEND
160     IF Tiefe = MaxTiefe THEN f  = 0 ELSE f = Tiefe MOD 4
170     PLOT 2*Spalte,2*Zeile,f
180     x = x + dx
190   NEXT Spalte
200   y = y + dy
210 NEXT Zeile
```

Die Bedienung von KOMGRA.BAS

Und jetzt zur Bedienung des Programms *Komplexe Grafik*: Nach dem Programmstart erscheint auf dem Bildschirm ein kleines Menuefenster. Es handelt sich um ein *Pull Down-Menue*, das man jederzeit mit Hilfe der ENTER-Taste aufrufen oder verschwinden lassen kann, ohne daß die Grafik zerstört wird. Die einzelnen Menuepunkte werden mit Hilfe eines Pointers angewählt, der mit den Cursortasten oben und unten bewegt wird. Die COPY-Taste aktiviert die Funktion, auf die der Pointer zeigt.

Um das *Apfelmännchen* als Grundlage für weitere Bilder zu erzeugen, wählen Sie am besten direkt den Punkt *Zeichnen* an. Danach will das Programm noch die maximale Iterationstiefe wissen; geben Sie hier den Wert 30 ein und drücken ENTER. Der CPC wird sich daraufhin etwa eine Stunde lang mit dem Bild beschäftigen.



Wenn es fertig ist, erscheint wieder das Menuefenster, und Sie können mit der Funktion ***Speichern*** die Grafik nach Eingabe eines Dateinamens auf Kassette oder Diskette sichern. An den Namen hängt das Programm selbsttätig die Erweiterung .PIC (Picture) an, um das Binärfile als Bilddatei zu markieren. Auch beim *Laden* dürfen Sie die Kennzeichnung nicht mit eingeben, da sie das Programm von sich aus ergänzt! Beim Speichern auf Diskette sollte man sich unbedingt davon überzeugen, daß der Name korrekt ist und sich wirklich eine Diskette im Laufwerk befindet, da sonst das Bild durch die Fehlermeldung zerstört wird.

Beim Speichern werden übrigens alle wichtigen Bilddaten (Farben, Koordinaten, Tiefe) mit aufgezeichnet. Ein Trick erspart hier zusätzliche Parameterdateien: Die Werte stehen nämlich in einem nicht sichtbaren Bereich des Bildschirmspeichers! Nach dem Laden werden sie dann wieder gelesen und korrekt eingestellt. Dadurch ist es sogar möglich, ein unvollständiges Bild abzuspeichern und bei Gelegenheit weiterzeichnen zu lassen. Der Abbruch erfolgt mit ENTER, der CPC reagiert allerdings erst, wenn er mit der aktuellen Rasterzeile fertig ist, was mitunter etwas dauern kann. Eine unvollständige Grafik im Speicher wird vom Programm automatisch erkannt; nach Anwahl von ***Zeichnen*** setzt er sein Werk fort, ohne vorher dumme Fragen zu stellen.

Der Menuepunkt ***Farben*** ermöglicht es, mit Hilfe der Cursortasten oben und unten die vier Farbstifte (PEN) anzusteuern und dann mit den Tasten links bzw. rechts die dazugehörige Farbe (INK) zu ändern. Angezeigt wird

dabei laufend die Nummer der Farbe – siehe CPC-Anleitungsbuch. Mit ENTER kommen Sie zurück ins Hauptmenue.

Die Funktion ***Ausschnitt*** bietet die Möglichkeit, einen bestimmten Bereich der komplexen Zahlenebene herauszugreifen, um ihn entsprechend vergrößert zeichnen zu lassen. Wenn sich bereits ein Bild im Speicher befindet, so wird ein Rechteck auf dem Bildschirm erscheinen, das Sie mit den Cursortasten herumwandern lassen können. Für die Feinabstimmung sind die Cursortasten plus SHIFT bzw. CTRL zuständig; auf diese Weise wird die obere linke bzw. untere rechte Ecke des Ausschnitts pixelweise verschoben. Die Form und Größe des Rechtecks kann damit beliebig verändert werden. Es empfiehlt sich jedoch, ein Seitenverhältnis zu wählen, das ungefähr dem Bildschirmformat entspricht, damit beim Zeichnen nicht starke Verzerrungen entstehen.

Wer sich für die Koordinaten des Ausschnitts interessiert, kann die Werte einfach mit COPY abrufen; eine beliebige Taste läßt sie wieder vom Bildschirm verschwinden. Mit ENTER wird die Wahl des Ausschnitts abgeschlossen, und das Menue erscheint. Wenn Sie den Punkt ***Ausschnitt*** anwählen, ohne daß sich ein Bild im Speicher befindet, ergibt das Rechteck natürlich keinen Sinn. Das Programm erwartet in diesem Fall von Ihnen, daß Sie den Bereich durch die Eingabe von vier Zahlen bestimmen: linker Rand; rechter Rand; oberer Rand; unterer Rand

ENTER ohne eine Eingabe führt auch hier zurück ins Hauptmenue. Der Programmteil ***Zeichnen*** erzeugt immer den zuletzt eingestellten Ausschnitt. Wenn Sie nach dem Programmstart keinen Bereich vorgeben, wird, wie bereits erwähnt, automatisch das *Apfelmännchen* mit folgenden Koordinaten gemalt:

$$\text{links} = -0.7; \quad \text{rechts} = 2.1; \quad \text{unten} = -1.0; \quad \text{oben} = 1.0$$

Wenn das Programm ein neues Bild zeichnet, so wird das alte Bild einfach überschrieben. Es bleibt also noch auf dem Bildschirm, es sei denn, Sie wählen zunächst den Punkt ***Löschen***. Diese Funktion löscht das Bild, nicht aber einen vorher eingestellten Ausschnitt!

Für das nächste Bild sucht man sich am besten einen Bereich aus, der schon von vornherein eine interessante Struktur erkennen läßt. Von den Vergrößerungen kann man natürlich weitere Ausschnitte zeichnen lassen und auf diese Weise immer weiter in Bilddetails vordringen, soweit die Rechengenauigkeit des CPC reicht.

Insbesondere der innere Rand des Apfelmännchens gibt allerhand her. Wenn Sie nach Wahl des Menuepunktes ***Zeichnen*** eine größere Rechentiefe vorgeben (maximal 255), kommen weitere Formen zum Vorschein: Je größer die Tiefe, um so mehr Strukturen kann das Programm erfassen. Allerdings steigt damit auch die Rechenzeit für ein komplettes Bild enorm an und kann sich bis auf mehrere Stunden ausdehnen. Etwas Geduld braucht man also schon für dieses Abenteuer im Land der komplexen Zahlen!

Interessante Koordinaten zum Ausprobieren:

links	rechts	oben	unten	Tiefe
1.778	1.793	0.0069	−0.0069	30
0.038	0.103	0.834	0.845	60
0.090649	0.09416	0.84146	0.84378	60
0.09282	0.094321	−0.84301	−0.84329	90
0.7445	0.7459	−0.1121	−0.1132	200

Literaturhinweis:

- Becker/Dörfler: Dynamische Systeme und Fraktale, Vieweg
- Peitgen/Richter: The Beauty of Fractals, Springer-Verlag

KAPITEL 2

Das Spiel des Lebens

Leistungsfähige, grafisch orientierte Computersysteme gewinnen in der Industrie, Wissenschaft und Kunst zunehmend an Bedeutung. Die Entwicklung und Darstellung hochkomplexer Strukturen ist in vielen Fällen überhaupt erst durch den Einsatz bildverarbeitende Systeme möglich geworden. Und erstaunlich genug – oft liefern selbst rein zweckorientierte Forschungsprojekte ein Bildmaterial, das man eher einem genialen Künstler zutrauen möchte. Die im letzten Kapitel behandelten Chaos-Grafiken stellen sicherlich eines der erstaunlichsten Beispiele in dieser Hinsicht dar.

Doch bei aller Begeisterung für die Möglichkeiten der neuen Technologie sollte man nicht vergessen, daß sich in der Natur zahlreiche Kreationen finden lassen, neben denen die meisten Computergrafiken recht blaß aussehen. Haben Sie schon einmal Schneekristalle durch die Lupe oder ein Mikroskop betrachtet? Falls nicht, so sollten Sie es bei nächster Gelegenheit nachholen, oder zumindest lesen, was der Schriftsteller Thomas Mann über dieses Erlebnis berichtet:

Kleinodien, Ordenssterne, Brillantagraffen, wie der getreueste Juwelier sie nicht reicher und minuziöser hätte herstellen können... und unter den Myriaden von Zaubersternchen in ihrer unsichtigen, dem Menschenauge nicht zugedachten, heimlichen Kleinpracht war nicht eines dem anderen gleich.

Der unerschöpfliche Erfindungsgeist, mit dem die Natur milliardenfache Variationen des Themas sechsstrahlige Symmetrie erzeugt, erscheint nach wie vor als ein unbegreifliches Wunder. Und doch ist es inzwischen mit Hilfe intelligent ausgedachter Computersimulationen gelungen, zumindest einige der Grundprinzipien dieser Naturvorgänge zu erforschen und zu verstehen.

So konnte unter anderem gezeigt werden, daß ein einfacher Satz geeigneter Regeln (Naturgesetze), wiederholt angewendet auf bestimmte Grundelemente (Moleküle), mit fast zwingender Notwendigkeit Strukturen beliebiger Komplexität erzeugt. Aufgrund dieser Erkenntnisse sprechen manche

Forscher von einer Selbstorganisationsfähigkeit der Materie, eine Theorie, die zu weitreichenden Konsequenzen führt. Wie ist zum Beispiel das Leben auf unserem Planeten entstanden – ein einmaliger Zufall? Wenn die Materie unter der Einwirkung der Naturgesetze die Fähigkeit erlangt, sich selbst zu strukturieren, so ist vielleicht die individuelle Ausprägung der Lebensformen zufallsbedingt, nicht aber die Tatsache des organischen Lebens an sich. Es mußte mit hoher Wahrscheinlichkeit entstehen, sobald die entsprechenden Rahmenbedingungen gegeben waren!

Am Anfang war der Punkt...

Wie aus einfachsten Voraussetzungen außerordentlich komplexe Muster entstehen können, soll nun ein kleines Software-Experiment auf dem CPC demonstrieren, ein wahres Kabinettstück der mathematischen Experimentalkunst. Eine derart kunstvolle Filigrangrafik, wie sie das Programm *Reproduktionsmuster* (REPRO.BAS) erzeugt, haben Sie wahrscheinlich auf Ihrem Bildschirm noch nicht gesehen! Das Programm operiert mit den denkbar einfachsten Grundelementen, nämlich mit Bildpunkten, die entweder gesetzt (hell) oder zurückgesetzt (dunkel) sind. Dazu kommen noch zwei Gesetze, die iterativ (wiederholt) auf dem gesamten Bildschirm angewendet werden:

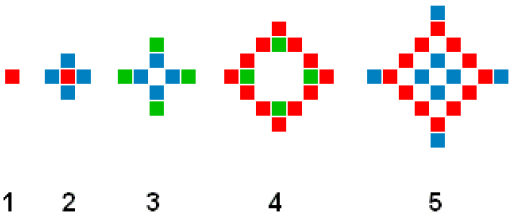
– Setze simultan alle Punkte, in deren Nachbarschaft (links, rechts, oben und unten) sich **genau ein** heller Punkt befindet.

Alle Punkte, die auf diese Weise bei einem Durchgang neu entstehen, werden als eine Generation bezeichnet. Mit dieser Definition kann die zweite Regel formuliert werden:

– Nachdem die Generation (n) erzeugt wurde, lösche alle Punkte der Generation (n-2).

Die Punkte gehen also nach einiger Zeit an Altersschwäche zugrunde; nach jedem Durchgang sind exakt die jeweils letzten zwei Generationen auf dem Bildschirm zu sehen. Solange die Regeln auf einen leeren Bildschirm angewendet werden, passiert natürlich überhaupt nichts. Doch ein einziger Punkt, der in Zeile 410 des Programms als Kristallisationskeim gesetzt wird,

bringt die Lawine ins Rollen. Das folgende Bild verdeutlicht das Prinzip und zeigt die Entwicklung der ersten fünf Generationen:



Das durch CALL &A000 aufgerufene Maschinenprogramm führt die Regeln aus und sorgt für die Bildschirmdarstellung. Übrigens wurden bei der Programmierung spezielle Techniken zur Erzeugung schneller Grafik benutzt, die direkt mit dem Bildschirmspeicher operieren. Die normalen Systemroutinen oder ein Basicprogramm wären hoffnungslos zu langsam, da pro Generation bis zu 384000 Punkte auf ihre Farbe getestet werden müssen!

Die Darstellung der Generationen durch verschiedene Farben hat nicht nur ästhetische Gründe. Sie ermöglicht es dem Programm, die Generation (n-2) zu erkennen, ohne eine Unmenge Koordinaten abzuspeichern. Weiterhin erlaubt die Farbdarstellung die quasi-synchrone Anwendung der Gesetze: Die während der Erzeugung der aktuellen Generation bereits gesetzten Punkte werden noch nicht als Nachbarn gezählt. Natürlich kann der Computer nicht wie gefordert alle Bildpunkte gleichzeitig erfassen, sondern nur nacheinander behandeln – eine Tatsache, die durch geeignete Programmiertricks ausgeglichen werden muß, um Störungen zu vermeiden.

Wer noch gerne etwas mit diesem System experimentieren möchte, kann eine veränderte Anfangssituation ausprobieren. Der in Zeile 410 erzeugte Punkt kann z.B. durch einen Buchstaben oder ein Grafiksymbol ersetzt werden. Sehr reizvoll sind auch Bilder, die aus mehreren auf dem Bildschirm verteilten Keimen entstehen. Obwohl die typische Charakteristik der erzeugten Grafiken erhalten bleibt, ergibt sich je nach Startfigur ein anderer Ablauf, und die Analogie zur Entstehung von Schneekristallen wird deutlich: Die Ausprägung der Form bestimmt der Kristallkeim (z.B. ein Staubkorn), die

charakteristische Symmetrie entsteht jedoch durch die Gesetzmäßigkeiten der molekularen Bindungskräfte.

Der Zellautomat

Bevor wir uns einem weiteren, auf ähnlichen Grundlagen beruhenden Software-Experiment zuwenden, soll zunächst daran erinnert werden, auf welche Weise der CPC Informationen verarbeitet. Wie Sie sicherlich wissen, enthält Ihr Computer einen Mikroprozessor als zentrale Steuerungseinheit. Er liest binäre Werte aus den Speicherzellen, verknüpft sie in seinen internen Registern nach bestimmten Regeln, und legt das Ergebnis wieder in anderen Speicherzellen ab.

Diese Art der Informationsverarbeitung nennt man sequentiell: Komplizierte Berechnungen werden ausgeführt, indem die dazu notwendigen Schritte nacheinander programmgesteuert abgearbeitet werden. Nach diesem Prinzip funktionieren praktisch alle Digitalcomputer, und nur die hohe Verarbeitungsgeschwindigkeit erzeugt mitunter den Eindruck, der Rechner könne mehrere Dinge gleichzeitig erledigen.

Der Computer ist damit ein getreues Abbild des sprachorientierten menschlichen Verstandes. Auch die verbale Informationsverarbeitung kann nur sequentiell erfolgen. Versuchen Sie einmal, bewußt zwei verschiedene Gedanken gleichzeitig zu denken: Sie werden sehen, es ist praktisch unmöglich.

Allerdings stellt der Verstand auch nur einen sehr kleinen Ausschnitt des informatischen Systems *Mensch* dar. Schon die verschiedenen Formen der Sinneswahrnehmung erfordern ein Verarbeitungssystem, das nach einem ganz anderen Prinzip arbeitet. Es ist ja ohne weiteres möglich, mehrere Dinge gleichzeitig zu sehen oder zu fühlen – mit anderen Worten, der Mensch kann eine große Anzahl verschiedener Daten parallel erfassen.

Der Verstand ist zwar in der Lage, die von den Sinnesorganen gelieferten Informationen auf einer abstrakten Ebene zu analysieren, doch für die unmittelbare Auswertung ist er effektiv zu langsam. Stellen Sie sich einmal vor, Sie müßten, um ein Haus zu erkennen, das Bild bewußt in ein Punktraster zerlegen, jedem Punkt einen Farb- und einen Helligkeitswert zuordnen, die Daten dann mit einem bereits gespeicherten Raster vergleichen und daraus einen Wert ermitteln, der Ihnen verrät, mit welcher Wahrscheinlichkeit es

sich um ein Haus handelt... des langen Satzes kurzer Sinn: Sie wären praktisch handlungsunfähig!

Dieses Beispiel macht deutlich, warum selbst die besten und schnellsten Computer trotz aller Bemühungen der Forscher nicht in der Lage sind, im menschlichen Sinne zu sehen oder zu hören: Ihr Arbeitsprinzip ist dazu einfach ungeeignet. Zwar werden inzwischen schon Versuche mit Multiprozessor-Systemen unternommen, doch auch hierbei ergeben sich erhebliche praktische Schwierigkeiten. So kann man z.B. nicht erwarten, daß ein Rechner mit 100 Prozessoren 100 mal so schnell wie ein normaler Computer läuft, da ein Teil der Kapazität gebraucht wird, um den Datentransfer zwischen den verschiedenen Einheiten zu koordinieren.

Interessanterweise existiert jedoch ein Ansatz zur Lösung dieser Probleme der bereits in der Vorcomputerzeit von dem Mathematiker John v. Neumann theoretisch untersucht wurde. Die schon 1936 von Alan M. Turing aufgeworfenen Frage nach der möglichen Existenz einer universellen Denkmaschine veranlaßte ihn zur gedanklichen Konstruktion eines sogenannten Zellautomaten. Dabei handelt es sich um einen homogenen, zellulär unterteilten Raum, in dem jede Zelle eine bestimmte Anzahl von Zuständen annehmen kann (z.B. verschiedene Farben). Auf alle Zellen wird synchron ein Satz von Umwandlungsregeln (Transition Rules) angewendet, deren Ausführung im einzelnen vom Zustand der jeweiligen Zelle und der angrenzenden Nachbarzellen abhängt.

John v. Neumann konnte theoretisch nachweisen, daß eine Konfiguration aus 200.000 Zellen mit jeweils vier Nachbarn und 29 Zustandsmöglichkeiten ausreicht, um im Prinzip jede denkbare Rechenoperation auszuführen. Solch ein Zellautomat enthält keinen zentralen Prozessor, sondern die Speicherzellen kommunizieren ohne Umschweife direkt miteinander. Mit jedem Taktimpuls, der im gesamten Raum Umwandlungen laut Regelkatalog bewirkt, werden eine große Anzahl von Operationen gleichzeitig ausgeführt. Informationen erscheinen dabei nicht mehr als Bits und Bytes, sondern als Raum- oder Flächenmuster, die sich überlagern und gegenseitig beeinflussen können – ein Prinzip, das die Natur auch bei der Konstruktion des menschlichen Gehirns angewendet hat.

Das Spiel des Lebens

Die wohl bekannteste Veranschaulichung des Zellautomaten-Prinzips ist das 1970 vom britischen Mathematiker Conway erfundene Game of Life. Wie in unserem vorherigen Experiment kann in diesem Spiel jede Zelle zwei verschiedene Zustände annehmen; sie ist also entweder bewohnt oder unbewohnt. Allerdings wurde die Nachbarschaft auch auf die diagonal angrenzenden Zellen erweitert – jede Zelle hat also acht Nachbarzellen – und die Umwandlungsregeln wurden so formuliert, daß die Abläufe recht realistisch die Folgen von Isolation, Überbevölkerung oder Kooperation in einer Gesellschaft lebendiger Wesen darstellen.

Die Regeln lassen sich durch zwei einfache Kommandos erfassen:

- Besetze synchron alle leeren Zellen, die genau drei Nachbarn haben.
- Lösche gleichzeitig (!) alle Zellen, die weniger als zwei oder mehr als drei Nachbarn haben.

Eine weitere Darstellungsmöglichkeit für diese Gesetze ist ein computergerecht gestalteter Regelkatalog in Tabellenform. Es gibt die Belegung einer Zelle in der Generation (n+1) in Abhängigkeit von ihrem momentanen Zustand und der Anzahl der Nachbarn an. Eine 1 bedeutet, daß die Zelle in der nächsten Generation bewohnt sein wird, eine 0 steht dagegen für eine leere Zelle:

Anzahl der Nachbarn	0	1	2	3	4	5	6	7	8
0 – Zelle wird zu	0	0	0	1	0	0	0	0	0
1 – Zelle wird zu	0	0	1	1	0	0	0	0	0

Auf den ersten Blick scheinen die Lebensbedingungen der Life-Einzeller also ziemlich schlecht zu sein, sie reagieren sehr empfindlich auf Einsamkeit oder Überbevölkerung. Einige Versuche zeigen jedoch recht schnell, daß gerade diese Einschränkungen dem System seine Dynamik verleihen und zu hochinteressanten Abläufen führen. Nicht umsonst hat dieses Spiel nach seiner Entstehung an Universitäten und wissenschaftlichen Instituten für allerhand Aufsehen und mitunter sogar Ärger gesorgt, da die Terminals der

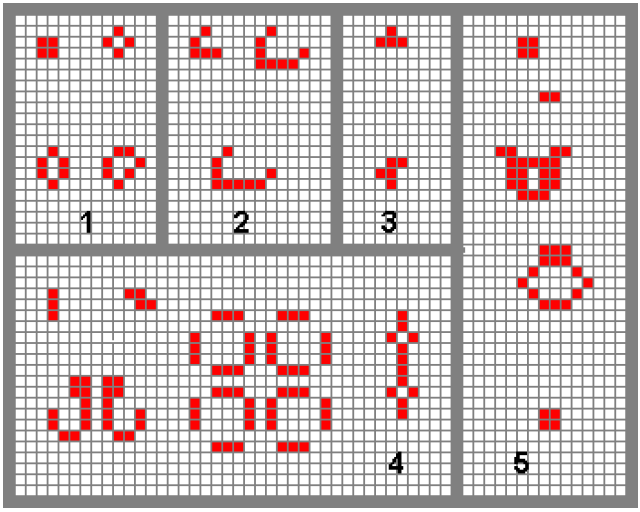
Rechenanlagen regelmäßig von Life-Fanatikern blockiert wurden – für jedermann verfügbare Heimcomputer gab es ja in den siebziger Jahren noch nicht.

Doch die Zeiten haben sich glücklicherweise geändert, und Sie können mit Hilfe des Programms Game of Life (LIFE.BAS) stundenlang auf Ihrem CPC experimentieren, ohne befürchten zu müssen, daß Sie im nächsten Augenblick das Betriebssystem wegen Überschreitung der Rechenzeit aus dem Rechner wirft.

In dem Programm werden die Zellen nicht wie bei den Reproduktionsmustern pixelweise, sondern etwas vergrößert dargestellt; insgesamt steht eine Fläche mit 50x80 = 4000 Zellen zur Verfügung. Die Umwandlungsregeln führt wieder ein kleines Maschinenprogramm aus, es schafft etwa zwei Generationen pro Sekunde. Besonderen Wert wurde auf eine extrem schnelle Bildschirmausgabe gelegt. In der Tat scheinen die Umwandlungen auf der gesamten Fläche gleichzeitig zu erfolgen.

Nach dem Programmstart erscheint zunächst das leere Spielfeld, das mit einem Karomuster vorstrukturiert ist, um eine genaue Analyse der einzelnen Abläufe zu ermöglichen. Um sich einen Eindruck von dem Treiben der Life-Einzeller zu verschaffen, können Sie zunächst die Taste Z drücken, worauf der Rechner eine Zufallsverteilung auf dem Bildschirm produziert. Warten Sie eine Weile, bis die Fläche einigermaßen voll ist, und brechen Sie dann den Vorgang mit ENTER ab. Mit Hilfe der SPACE-Taste können Sie nun die jeweils nächste Generation erzeugen, oder auch mit A die Generationswechsel automatisch ablaufen lassen. Auch diese Funktion wird mit ENTER abgebrochen.

Um gezielt die Eigenschaften bestimmter Konfigurationen zu untersuchen, kann mit E ein kleiner Editor angewählt werden. Der Cursor wird wie üblich mit den Pfeiltasten gesteuert; die Taste 1 belegt die Zelle auf Cursorposition, die Taste 0 löscht sie entsprechend. Um ganze Reihen von Life-Wesen zu produzieren, können Sie auch die Taste 1 gedrückt halten und den Cursor über den Bildschirm wandern lassen, er hinterläßt dann eine Spur von gefüllten Zellen. Mit ENTER wird der Editmodus wieder verlassen; mit L bei Bedarf die gesamte Fläche gelöscht.



Das obige Bild zeigt einige typische Life-Konfigurationen. Die mit 1 bezifferte Gruppe enthält vier stabile Figuren, die bei den Generationswechseln unverändert bleiben. Gruppe 2 zeigt drei sogenannte Gleiter. Bei diesen Figuren handelt es sich um Oszillatoren, die periodisch ihre alte Form wiedererlangen, sich dabei aber um ein kleines Stück weiterbewegen. Stationäre Oszillatoren mit einer Periode von 2 bis 15 Generationen finden Sie dagegen in der Gruppe 4 versammelt. Ein besonders eigenartiges Verhalten zeigt der 30-Generationen-Oszillator, der die Gruppierung 5 bildet: Er produziert, ohne dabei an Substanz zu verlieren, eine Serie kleiner Gleiter, die nach rechts abwandern.

Die Gruppe 3 enthält noch zwei Figuren, die ein expansives Verhalten zeigen. Während die obere Figur jedoch schon nach wenigen Generationen in vier stationäre Oszillatoren zerfällt, ist die untere Figur ohne weiteres in der Lage, den gesamten Bildschirm mit ihren Nachkommen zu füllen. Sie gilt als die kleinste Figur, die sich beliebig weit ausdehnt; bisher konnte kein definierter Endzustand ermittelt werden.

Wenn Sie die abgebildeten Muster alle gemeinsam testen wollen, entfernen Sie am besten die expansive Figur und die drei Gleiter aus dem Sortiment, um die anderen Figuren nicht zu stören. Ansonsten ist es aber sehr interessant, was sich bei einer Kollision verschiedener Muster abspielt: so-

wohl eine totale Auslöschung als auch die Entstehung neuer Figuren kann die Folge sein. Durch die eifrigen Bemühungen der vielen Life-Forscher ist es inzwischen sogar gelungen, Muster zu finden, die durch gegenseitige Beeinflussung logische Funktionen wie etwa NOT, AND, OR oder XOR simulieren – ein wichtiger Schritt in Richtung eines praktisch anwendbaren Zellautomaten.

Ein Problem, das die Realisierung eines auch nur kleinen Zellrechners auf dem CPC verhindert, ist leider die unvermeidliche Spielfeldbegrenzung. Selbst eine 320x200-Fläche wäre noch viel zu klein. Das Programm betrachtet alle Zellen außerhalb der Fläche als leer, und daraus ergeben sich empfindliche Störungen, sobald ein Muster an den Rand stößt. Auf interessante Weise arrangiert sich allerdings der Gleiter oben rechts mit dieser Beschränkung: Er wird reflektiert und verwandelt sich dabei in einen kleineren Gleiter!

Wie man bei eigenen Experimenten feststellen kann, führt die Begrenzung des Lebensraumes dazu, daß alle Konfigurationen nach einiger Zeit steckenbleiben; es bleiben nur noch stabile Muster und Oszillatoren übrig, die keine grundsätzlich neuen Entwicklungen mehr bewirken können. Doch dieses Phänomen hat noch eine andere Ursache: Im Game of Life ist der schöpferische Zufall nicht vorgesehen. Der gesamte Prozeß läuft deterministisch ab, d.h. mit einer gegebenen Anfangssituation steht praktisch schon in allen Einzelheiten fest, was passieren wird. Sie können aber einmal erforschen, was die folgenden zusätzlichen Programmzeilen bewirken:

```
981 f=1
982 re= INT (RND*50) + 1
983 sp= INT (RND*80) + 1
984 GOSUB 1060
```

Wenn Sie jetzt den automatischen Generationswechsel mit A anwählen, wird bei jedem Durchgang ein neues Wesen an einem zufälligen Ort geboren und bringt etwas Abwechslung in die Angelegenheit. Wie bereits im 1. Kapitel angedeutet, spielt der Zufall bei der Erzeugung dynamischer (lebendiger) Strukturen ja eine wichtige Rolle.

Und es gibt noch eine weitere sehr interessante Experimentiermöglichkeit: In den DATA-Zeilen 530 und 540 steht der vorhin bereits erwähnte Regelkatalog für das Game of Life. Hier können sie natürlich die Nullen

und Einsen auch anders verteilen und damit neue Umwandlungsgesetze gestalten. Die Abläufe, die sich aus einem veränderten Regelkatalog ergeben, stellen ein recht unerforschtes Gebiet dar und bieten noch viel Raum für aufregende Entdeckungen.

Evolution im Computer?

Nach dieser schon recht eindrucksvollen Simulation eines Zellautomaten drängt sich die Frage auf: Wieso ist eigentlich eine solche Maschine noch nicht gebaut worden? An der Hardware dürfte es nicht scheitern, die nötigen Chips wären mit dem heutigen technischen Stand sicherlich realisierbar. Doch das Problem liegt ganz woanders: Kein Mensch weiß, wie man einen Zellautomaten programmieren könnte.

Wie bereits erörtert wurde, beruht jede Sprache – und damit auch jede Programmiersprache – auf einer sequentiellen Logik. Zwar können wir auch über Ereignisse sprechen, die simultan stattfinden, jedoch nur, indem wir sie nacheinander beschreiben. Deshalb ist es sehr zweifelhaft, ob ein leistungsfähiger Zellautomat, der hunderte und tausende von Operationen gleichzeitig durchführt und dabei unzählige sich kreuz und quer gegenseitig beeinflussende Muster bildet, jemals mit den Mitteln eines sequentiell arbeitenden Verstandes beherrscht werden kann.

Bisher ist nur ein einziger Ausweg aus diesem Dilemma sichtbar: Die Maschine muß sich selbst programmieren! Wenn Sie Ihren CPC programmieren, so formulieren Sie mit Hilfe einer Programmiersprache einen Algorithmus, der Schritt für Schritt angibt, wie aus einer Situation A das Resultat B erzeugt wird. Wie einfach wäre es doch, wenn Sie den Computer einschalten und zu ihm sprechen könnten: ‚Hier mein elektronischer Freund, das ist ein Schachbrett, das sind die Figuren, und so gehen die Regeln... und du hast gewonnen, wenn du meinen König eroberst, bevor ich deinen erwische. Spiele so, daß du nach Möglichkeit gewinnst!‘

Sehen Sie – genau das müßte der Zellautomat können: Selbständig Lösungswege finden. Wie bereits gezeigt wurde, stellt es kein Problem dar, sich selbst organisierende Systeme zu erschaffen. Mit ein paar Umwandlungsregeln, die irgendwelche Muster erzeugen, ist es in diesem Fall jedoch nicht getan. Um gut Schach zu spielen, müßte die Maschine zielgerecht problembezogene Strukturen produzieren, ungünstige Muster verwerfen, Erfahrungen

sammeln und selektiv auswerten... mit anderen Worten, einen Lernvorgang, eine Evolution nachvollziehen.

Doch die Formulierung von Bewertungsschemata, die einen sich selbst organisierenden Prozeß zielgerichtet ablaufen lassen, ist bisher ein theoretisch nur ansatzweise gelöstes Problem. Hinweise liefert allerdings ein Forschungsbereich, der sich mit dem Grenzgebiet zwischen Biologie und Informatik beschäftigt. Welche Strategien hat die Natur bei der Entwicklung des Lebens angewendet? Welche statistischen Gesetzmäßigkeiten tragen dazu bei, daß sich bestimmte Arten entfalten, während andere aussterben?

Das nächste Kapitel des Software-Experiments wird noch weiteres Material zu diesem Thema liefern. Wer sich intensiver mit diesem Gebiet beschäftigen möchte, dem sei das Buch *Das Spiel – Naturgesetze steuern den Zufall* von Manfred Eigen/Ruthild Winkler (Piper Verlag) empfohlen. Es enthält zahlreiche Simulationsspiele, die sich auch auf einem Heimcomputer gut realisieren lassen.

KAPITEL 3

Das Ökologie-Experiment

Simulationen – Nachbildungen der Wirklichkeit – haben sich einen festen Platz unter den Standardanwendungen für Computersysteme erobert. Sei es bei der Ausbildung von Piloten für die Luft- oder Raumfahrt, bei wissenschaftlichen Forschungsprojekten oder in der wirtschaftlichen und militärischen Planung: In vielen Bereichen sind sie überhaupt nicht mehr wegzudenken.

Warum bisher keine Computersimulation die verantwortlichen Politiker und Militärs davon überzeugen konnte, wie gefährlich und unsinnig das Wettrüsten ist, sei allerdings dahingestellt. In dieser Folge des Software-Experiments werden wir uns mit einem anderen Thema beschäftigen, das ebenfalls sehr aktuell ist. Bereits in der letzten Folge wurde mit dem Game of Life ein Programm vorgestellt, das ansatzweise ein ökologisches System simuliert; und diese Idee soll noch etwas weiter verfolgt werden.

Bisher haben menschliche Eingriffe in die außerordentlich komplexe Dynamik ökologischer Systeme immer wieder zu der Erfahrung geführt, daß die Ergebnisse oft ganz anders aussehen, als es eigentlich geplant war. Insbesondere die langfristigen Folgen von zunächst sehr erfolgversprechenden Maßnahmen führen uns vor Augen, daß letztendlich der Mensch selbst betroffen ist, wenn er das empfindliche Gleichgewicht der Natur durch unbedachte Eingriffe zerstört. Der Einsatz von chemischen Mitteln verbessert die Ernteerträge – so sieht es in der Tat zunächst aus. Doch wenn sich nach einiger Zeit die bekämpften Schädlinge als resistent erweisen, ihre natürlichen Feinde dagegen scharenweise an dem Gift zugrunde gehen und die gesundheitlichen Folgen der Rückstände in den Nahrungsmitteln für den Menschen nicht mehr zu verantworten sind – was dann?

Die Dynamik ökologischer System wird durch eine Vielzahl ineinandergreifender Regelkreise und sich überlagernder Einflüsse bestimmt. Es ist deshalb kaum möglich, die Folgen von Eingriffen durch einfache, kausale

Wenn-Dann-Modelle zu erfassen. Was liegt also näher, als sich mit Hilfe von Computersimulationen einen Einblick in dieses so wohlgeordnete Chaos zu verschaffen? Die Geschwindigkeit und Speicherkapazität moderner Rechenanlagen erlaubt es, Entwicklungen, die sich in der Realität über Jahre und Jahrzehnte erstrecken, in wenigen Sekunden ablaufen zu lassen und dabei durch gezielte Veränderung der Systemparameter eine ganze Palette von Möglichkeiten zu erforschen.

Doch dabei ist Vorsicht geboten: Ein Abbild der Wirklichkeit im Speicher eines Computers ist immer eine reduzierte Wirklichkeit. So wie eine Landkarte zwar verschiedene Aspekte einer Landschaft wiedergeben kann, aber nie die Landschaft als Ganzes, so kann ein Computermodell nur eine Auswahl wichtiger Faktoren erfassen, und diese Auswahl muß der Programmierer treffen. Deshalb sollte man die Aussagekraft von Simulationen nicht überschätzen. Sie geben wie Landkarten einen Überblick, aber als Propheten taugen sie nicht. So wird ein Computer, der die Funktionen eines Kernreaktors simuliert, zwar zeigen können, auf welche Weise Störfälle möglich sind, nicht aber, daß sie unmöglich sind!

Das Wolf-Schaf-Gras-Modell

Auch viele für den CPC erhältliche Spiele stellen im Prinzip Simulationen dar, Sportspiele oder das Weltraumabenteuer Elite wären als Beispiel zu nennen. Doch während es hierbei im wesentlichen auf die Qualität der Grafik- und Soundeffekte ankommt, werden an wissenschaftliche Simulationsprogramme ganz andere Anforderungen gestellt. Sie sollen mit möglichst großer Exaktheit die Wirklichkeit quantitativ, d.h. in Form von Zahlenwerten wiederspiegeln. Dieses Material stellt für den Forscher eine geeignete Basis dar, um neue Erkenntnisse zu gewinnen oder bestimmte Theorien zu überprüfen. Auf den Uneingeweihten wirken lange Zahlenkolonnen oder verschlungene Bildschirmdiagramme jedoch im allgemeinen nicht besonders aufschlußreich.

Um mit solchen Programmen arbeiten zu können, sind schon recht solide mathematische Grundlagen erforderlich (z.B. Wachstumsfunktionen), deren Erörterung den Rahmen dieses Buches beiweitem sprengen würde. Deshalb werden wir als Einführung in den Themenbereich *Ökologie-Simulation* ein System behandeln, das zwar keinen Anspruch auf wissenschaftliche Exakt-

heit stellt, aber trotzdem eine ganze Reihe interessanter Experimente ermöglicht, und dabei auch etwas für's Auge bietet: Die Zeitraffer-Darstellung eines künstlichen kleinen Lebensraumes, in dem das Schicksal der Individuen nicht nur statistische Bedeutung hat, sondern auch einzeln beobachtet werden kann.

Ausgangspunkt ist also das im letzten Kapitel besprochene Game of Life, wobei jedoch einige wesentliche Änderungen vorgenommen werden. Zunächst soll jede Zelle des Spielfeldes nicht nur zwei, sondern vier verschiedene Zustände annehmen können. Um die Angelegenheit anschaulich zu gestalten, werden wir den Zuständen, repräsentiert durch die Zahlen von 0..3, sofort eine konkrete Bedeutung geben:

0 = leer 1 = Gras 2 = Schaf 3 = Wolf

Was nun während des Spielablaufs mit dem Inhalt einer Zelle passiert, hängt ähnlich wie beim Game of Life von der unmittelbaren Nachbarschaft ab; als Nachbarn sollen die vier direkt angrenzenden Zellen (oben, unten, links und rechts) gelten. Was mit einem Schaf passiert, das sich in der Nähe eines hungrigen Wolfes befindet, können Sie sich wahrscheinlich schon denken – doch zur exakten Formulierung der Spielregeln kommen wir gleich.



Zunächst ein weiterer wichtiger Unterschied zum Game of Life: Wie bereits erwähnt, ist das Modell von Conway streng deterministisch aufgebaut, es gibt dem Zufall keinen Raum. Über ein echtes ökologisches System kann man jedoch nur statistische Aussagen machen, also zum Beispiel über die

Häufigkeit oder Wahrscheinlichkeit bestimmter Ereignisse. Wann und wo ein Wolf über ein Schaf herfallen wird, läßt sich nicht genau berechnen, die exakten Koordinaten der Ereignisse in Zeit und Raum sind vollkommen unbestimmt.

Dieser Tatsache soll in unserem Ökologie-Experiment Rechnung getragen werden, indem die Ereignisse laut Regelkatalog nicht mehr simultan auf dem gesamten Spielfeld ausgeführt werden, sondern nacheinander, wobei der Rechner jedes Mal den Ort des Geschehens mit Hilfe der RND-Funktion auswürfelt. Daß sich ein Schaf in der Nähe eines Wolfes nicht unbedingt sicher fühlen kann, ist damit weiterhin klar; nur wann es gefressen wird, das steht in den Sternen.

Fressen und gefressen werden

Hier nun die Gesetze, die die Nachbarschaftsbeziehungen in unserem WSG-Modell regeln:

(1) Eine Leerzelle, in deren Nachbarschaft sich Gras befindet, wird durch ein neues Grasbüschel belegt.
(2) Eine mit Gras besetzte Zelle wird mit einem Schaf belegt, wenn sich mindestens ein Schaf in der Umgebung befindet.
(3) Ein Schaf wird durch einen neuen Wolf ersetzt, wenn sich mindestens ein Wolf in der Nachbarschaft befindet.
(4) Hat ein Schaf kein Gras in seiner Umgebung, so wird es durch eine Leerzelle ersetzt.
(5) Hat ein Wolf kein Schaf als Nachbar, so wird er ebenfalls durch eine Leerzelle ersetzt.
(6) In allen anderen Fällen findet keine Umwandlung statt.

In den Regeln (2) und (3) sind also Fressen und Vermehrung miteinander gekoppelt, was nicht unbedingt realistisch ist, aber zu sehr dynamischen Abläufen führt. Die Regeln (4) und (5) liefern Wölfe und Schafe dem Hungertot aus, wenn sie zu gierig werden.

Der CPC kann natürlich mit der sprachlichen Formulierung der Gesetze nicht viel anfangen, womit die Frage nach einer computergerechten Darstellung des Regelkatalogs auftaucht. Die Programmierung einer Fallunterschei-

dung mit Hilfe vieler IF..THEN-Statements ist zwar naheliegend, wird aber schnell unübersichtlich. Auch eine nachträgliche Änderung des Katalogs ist auf diese Weise nur sehr umständlich durchzuführen.

Für unsere Zwecke ist die im folgenden dargestellte Tabellenform wesentlich besser geeignet. Dieses System ist jedenfalls einfacher zu handhaben, als es auf den ersten Blick scheint: Die acht Spalten auf der rechten Seite und die mit Fragezeichen markierten Zeilen brauchen uns zunächst nicht interessieren, da sie nur für ein erweitertes System mit fünf Zellenzuständen eine Rolle spielen; und die Bedeutung der restlichen Eintragungen ist schnell geklärt.

Nachbarschaft	Gras	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0 = nicht vorhanden	Schaf	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 = vorhanden	Wolf	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	????	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Umwandlungs-	0 = Leer	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
anweisungen für	1 = Gras	1	1	2	2	1	1	2	2	0	0	0	0	0	0	0	0
die mittlere Zelle	2 = Schaf	0	2	0	2	3	3	3	3	0	0	0	0	0	0	0	0
	3 = Wolf	0	0	3	3	0	0	3	3	0	0	0	0	0	0	0	0
	4 = ????	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Der obere Teil der Tabelle stellt alle möglichen Nachbarschaftskonstellationen dar. Wer sich etwas mit Maschinensprache auskennt, hat sicherlich auf den ersten Blick festgestellt, daß hier die Binärzahlen von 0...15 versammelt sind, und zwar senkrecht geschrieben. Diese Anordnung hilft dem Rechner, nach der Analyse der Nachbarschaft die zuständige Spalte im Regelkatalog zu finden. Der untere Teil der Tabelle enthält die Umwandlungsanweisungen für die ausgewürfelte Zelle. Wenn der Rechner zum Beispiel ein Schaf erwischt, das von Gras und Wolf umgeben ist, so muß er in der sechsten Spalte von links nachschauen. In der für Schafe zuständigen Zeile 2 in der unteren Tabellenhälfte findet er dann am Kreuzungspunkt eine 3, das arme Schaf wird also durch einen neuen Wolf ersetzt.

Sie können sich anhand weiterer Beispiele davon überzeugen, daß die Tabelleneintragungen exakt die sechs Regeln für das WSG-Modell wiedergeben, wenn auch in recht abstrakter Form. Doch mit etwas Übung stellt die Codierung selbsterdachter Gesetze kein Problem dar. Für ambitionierte Programmierer wäre es vielleicht eine interessante Aufgabe, eine symbolische

Öko-Sprache zu erfinden und eine Art Compiler zu schreiben, der daraus die Tabellenform erzeugt.

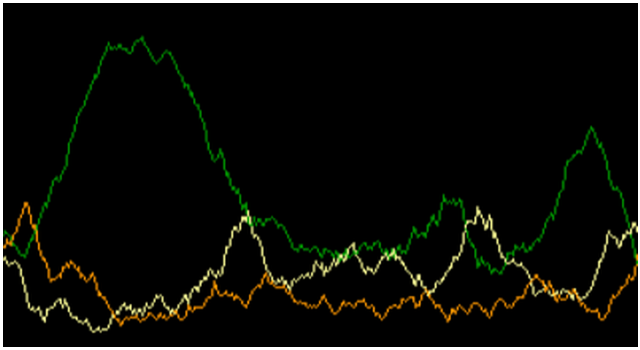
Das Öko-Programm

Das Simulationsprogramm **OEKOEXP.BAS** ist weitgehend modular aufgebaut und mit zahlreichen Kommentaren versehen, um auf einfache Weise Änderungen zu ermöglichen. Das Hauptprogramm zu Beginn hat nur die Aufgabe, die verschiedenen Programmteile in der richtigen Reihenfolge aufzurufen.

Im Initialisierungsteil ab Zeile 1000 werden die Arrays für das Spielfeld, den Regelkatalog usw. dimensioniert. Die eigentliche Definition des WSG-Modells erfolgt ab Zeile 2000. Für die Darstellung der verschiedenen Bewohner unseres Öko-Systems werden Grafikzeichen mit doppelter Höhe erzeugt, die ab Zeile 2150 mittels einiger Steuerzeichen zusammengesetzt werden – sogar die dazugehörige Farbe wird mit CHR$(15) gleich integriert. Damit es bei der Ausgabe dieser Superzeichen am rechten Bildschirmrand keine Probleme gibt, täuscht ein POKE im Initialisierungsteil dem CPC die Existenz von 40 Spalten in MODE 0 vor.

Auch die Regeltabelle wird in diesem Programmteil eingelesen; sie befindet sich in den DATA-Zeilen 2300 – 2340. Hier können Sie beliebige Änderungen vornehmen, um den Ablauf der Ökologie-Experimente zu beeinflussen. Das Unterprogramm ab Zeile 3000 bereitet den Spielablauf vor, indem es eine Zufallsverteilung von Wölfen, Schafen und Gras als Ausgangssituation erzeugt. Da es bei der Erforschung der Nachbarschaft passieren kann, daß das Programm über den Rand hinausschaut, wird dem Spielfeld intern noch ein Rahmen verpaßt, der mit der neutralen Ziffer 9 belegt ist.

Der eigentliche Spielablauf findet ab Zeile 4000 statt. Zunächst wird auf einen Tastendruck als Startsignal gewartet, und danach ein Zeitintervall von vier Sekunden (200 * 1/50 sec) für die regelmäßige Bestandsaufnahme festgelegt, die vom Unterprogramm in Zeile 6000 vorgenommen wird: Es merkt sich die jeweiligen Populationszahlen, so daß sie das Modul in Zeile 5000 zum Abschluß als Funktion der Zeit grafisch darstellen kann.



Das Unterprogramm ab Zeile 10000 steht den anderen Modulen zur Verfügung. Es führt nicht nur die Änderung der Zellinhalte aus, sondern bringt dabei auch die verschiedenen Zähler auf den neuesten Stand. Als Parameter werden die Koordinaten (X,Y) einer Zelle und der neue Inhalt in der Variablen neu Übergeben.

Katastrophen im Computer

Nach dem Programmstart, der Erzeugung der Anfangsverteilung und dem Beginn des Spielablaufs wird sich ein recht munteres Treiben auf dem Bildschirm entfalten; zusätzlich geben noch drei Zähler im unteren Bildschirmbereich über die aktuellen Populationszahlen Auskunft. Ein weiterer Zähler zeigt laufend die Anzahl der ausgewürfelten Ereignisse an. Wie sich schon nach wenigen Versuchen herausstellt, ergeben sich vollkommen andersartige Abläufe als beim Game of Life. Während das im letzten Kapitel beschriebene Simulationsspiel eine Neigung entwickelte, nach einiger Zeit in festen Formen zu stagnieren, führt die zufallsgesteuerte Dynamik des WSG-Modells leicht zu Katastrophen, bei denen eine oder mehrere Arten vollkommen aussterben.

Zwar sind die WSG-Gesetze so formuliert, daß sie extremen Entwicklungen entgegenwirken: Eine starke Vermehrung der Schafe zieht zum Beispiel auch eine entsprechende Vermehrung der Wölfe und Reduzierung der Nahrung nach sich. Da aber die räumliche Verteilung der Lebewesen in unserem Spiel eine starke Rolle spielt, setzen die Regelmechanismen oft erst verzögert ein, und es entstehen mehr oder minder starke Schwankungen der

Populationszahlen, was sich nach dem Abbruch mit ENTER sehr schön aus den gezeichneten Kurven ersehen läßt. Insbesondere bei einem beschränkten Spielfeld von nur 200 Zellen kommt es dabei leicht zu Fluktuationskatastrophen. Wie stark in der Tat die Elastizität eines ökologischen Systems von der Größe des Lebensraumes abhängt, soll nun ein weiteres Experiment zeigen.

Zu diesem Zweck muß mit Hilfe der MERGE-Anweisung die Erweiterung 1 (OEKOERW1) in das Hauptprogramm integriert werden. Sorgen Sie dafür, daß sich das Hauptprogramm im Speicher befindet, und geben Sie dann **MERGE „OEKOERW1"** ein, worauf die neuen Programmteile dazugeladen werden. Auch die noch folgenden Ergänzungen Lassen sich auf diese Weise einbauen. Vorher muß jedoch immer das Originalprogramm **OEKOEXP.BAS** neu geladen werden, da sich die Erweiterungen untereinander nicht vertragen!

Die Erweiterung 1 zerlegt das Spielfeld durch eine Autobahn in zwei voneinander isolierte Teile. Dazu tritt noch eine neue Regel in Kraft:

(7) Schafe und Wölfe in der Nachbarschaft der Autobahn werden überfahren, also durch ein Leerfeld ersetzt.

Für die Darstellung der Autobahn wird intern die Ziffer 4 als fünfter Zellenzustand benutzt, ein Beispiel dafür, daß sich mit dieser Codierung nicht nur Lebewesen darstellen lassen. Der neue Regelkatalog für das erweiterte System enthält in der letzten Zeile natürlich nur Vieren, da die Autobahn nicht umgewandelt werden kann.

Die linken acht Spalten entsprechen den ursprünglichen WSG-Gesetzen, die rechten acht Spalten spiegeln die Änderungen wieder, die sich in direkter Nahe der Autobahn ergeben. Wie ein paar Versuche zeigen, ist zumindest in einem der beiden Teilbereiche meist schon nach kurzer Zeit das ökologische Gleichgewicht zerstört. Die unmittelbare Wirkung der Autobahn spielt dabei noch nicht einmal die entscheidende Rolle. Viel schlimmer wirkt sich die herabgesetzte Toleranzbreite für die natürlichen Schwankungen des System aus. Ein kleines System ist mit einer Differenz von +/- 20 Tieren natürlich viel eher überfordert als ein großer Lebensraum – ein statistischer Effekt, der bei der Landschaftsplanung leicht übersehen wird.

Der ökologische Balanceakt

Auf der anderen Seite stellt sich natürlich die Frage, welche Maßnahmen geeignet sind, um solche Katastrophen zu verhindern. Zwei Ansätze sollen im Folgenden untersucht werden:

1. Die Verbesserung der Energieversorgung

Unser WSG-Modell ist ein sogenanntes dissipatives System, das zu einer Aufrechterhaltung ständig Energie umsetzt. Diese Energie wird durch das nachwachsende Gras repräsentiert; ohne Graswuchs haben sowohl Schafe als auch Wölfe keine Überlebenschance. Eine Verbesserung der Energieversorgung läßt sich sehr einfach erreichen, indem man in die erste DATA-Zeile des Regelkatalogs durchweg Einsen schreibt, wonach die Regel (1) folgende Gestalt annimmt:

(1) Ein Leerfeld wird auf jeden Fall mit neuem Gras belegt.

Damit kann das Gras also an jeder beliebigen Stelle neu wachsen (z.B. durch Samenflug), die Nachbarschaft anderer Pflanzen ist nicht mehr notwendig. Einige Tests zeigen schnell, daß auf diese Weise in der Tat mehr Stabilität gegeben ist; das Spiel läuft im Schnitt wesentlich länger katastrophenfrei. Allerdings sind solche idealen Verhältnisse nur bedingt realistisch: Ein derart ungehemmtes Pflanzenwachstum wird im allgemeinen nur zeitweise, also im Frühling oder Sommer stattfinden.

2. Der menschliche Eingriff

Eine kritische Situation tritt bei dem WSG-Modell oft schon kurz nach Beginn auf. Die gute Durchmischung der Lebewesen in der Startverteilung führt zu einem starken Anwachsen der Wolfspopulation und einer Reduzierung der Vegetation, wodurch prompt die Schafe in eine Krise kommen. Doch Hilfe naht: Wir werden mit der Erweiterung 2 ein paar Jäger ins Feld schicken, um den Bestand an Wölfen knapp zu halten. Gehen Sie zu diesem Zweck genau wie bei der ersten Erweiterung vor: Laden Sie das Hauptprogramm **OEKOEXP.BAS** neu und geben Sie dann **MERGE „OEKOERW2“** ein. Durch neue DATA-Zeilen für den Regelkatalog werden folgende Gesetze eingeführt:

(-) Eine Leerzelle in der Nachbarschaft eines Wolfes wird mit einem Jäger belegt.

(-) Ein Wolf, der einen Jäger als Nachbarn hat, wird durch ein Leerfeld ersetzt.

(-) Ein Jäger, in dessen Nachbarschaft sich kein Wolf befindet, verschwindet wieder.

Die Folgen dieser Ergänzung sind jedoch sehr drastisch: In den meisten Fällen werden die fleißigen Jäger recht schnell ein Aussterben der Wölfe bewirken, was zwar den Schafen im ersten Moment gut tut, aber nicht dem Gras. Der Bildschirm wird regelrecht kahlgefressen. Nur die daraufhin einsetzende Hungerkatastrophe verhindert mitunter eine totale Einöde.

Wie Sie sehen, ist es also gar nicht so einfach, ein System zu gestalten, daß sich dynamisch und trotzdem stabil verhält. Wer den Ehrgeiz entwickelt, durch Herumbasteln am Regelkatalog seine Idealökologie zu entwickeln, sollte dabei beachten, daß sich die Tendenz eines System im allgemeinen erst nach mehreren Durchläufen erkennen läßt. Durch den starken Einfluß des Zufalls zeigt ein einzelner Spielablauf immer nur, was passieren kann, aber nicht, was passieren wird! Aus diesem Grund ist unser Programm zwar recht spannend, aber in der vorliegenden Form nicht geeignet, um wissenschaftliche Theorien zu entwickeln: Es fehlt die Reproduzierbarkeit der Versuchsergebnisse.

Software-Autor gesucht

Andererseits bietet unser Öko-Experiment jedoch einen Ansatz, der bei wissenschaftlichen Programmen meist nicht gegeben ist: Es läßt sich zu einem recht interessanten Spielprogramm ausbauen. Bislang bestand zwar nur die Möglichkeit, die Regeln zu definieren und sich den Ablauf passiv anzuschauen, doch eine dritte Erweiterung soll zeigen, wie eine direkte Eingriffsmöglichkeit programmiert werden kann.

Der Programmteil **OEKOERW3**, der genau wie die vorherigen Ergänzungen mit **MERGE** eingebunden wird, bringt einen Jäger auf das Spielfeld, den Sie mit Hilfe der **Cursortasten** herumwandern lassen können. Bereits mit Wölfen oder Schafen besetzte Zellen kann er natürlich nicht betreten, und da er weiterhin das Gras unter seinen Füßen zertrampelt, sollten Sie ihn etwas behutsam durch die Landschaft bewegen.

Auf der anderen Seite stehen ihm jedoch auch einige Aktionsmöglichkeiten zur Verfügung. Wenn Sie die SPACE-Taste gedrückt halten und zusätzlich

die **Cursortasten** betätigen, wird je nach Lage der Dinge in der entsprechenden Richtung ein Leerfeld mit Gras bepflanzt oder auch ein Wolf oder Schaf abgeschossen. Doch nichts ist ohne Risiko: Wie Sie der letzten DATA-Zeile entnehmen können, ist der Jäger auch als Futter für die Wölfe vorgesehen. Wenn er sich in der Nachbarschaft eines Wolfes befindet und in diesem Moment zufällig ausgewürfelt wird, ist es um ihn geschehen!

Auf diese Weise können Sie also regulierend in den Ablauf Ihrer Ökologie-Experimente eingreifen. Für einen echten Spiele-Hit fehlt dem Programm zwar noch eine spannungsvolle Motivation, doch das läßt sich mit einigen zusätzlichen Ideen ändern. Hier ein paar Anregungen:

Benutzen Sie mehrere Regelkataloge, die periodisch oder zufallsgesteuert ausgetauscht werden, um Jahreszeiten oder Seuchen zu simulieren.

- Statten Sie den Jäger mit begrenzten Mitteln aus (Graspflanzen, Munition), die möglichst sinnvoll zu Erhaltung des Ökologischen Gleichgewichts eingesetzt werden müssen.
- Gestalten Sie eine Landschaft, in der die Zellen eine unterschiedliche Lebensqualität haben. Zu diesem Zweck könnte z.B. ein zusätzliches Array lq(x,y) definiert werden, das jeder Zelle des Spielfeldes einen Wert zuweist, der auf die Überlebenschancen der Bewohner Einfluß hat.
- Bevölkern Sie das Spielfeld mit weiteren Pflanzen- und Tierarten.

Der letzte Punkt ist allerdings programmtechnisch nicht leicht in den Griff zu bekommen, da die Menge M der Eintragungen in der Regeltabelle mit der Anzahl Z der möglichen Zellenzustände nach der Formel

$$M = Z \cdot 2^{(z-1)}$$

exponentiell anwächst. Hier müßte man also eventuell noch einen geeigneteren Algorithmus finden.

Ergänzende Literatur:

Hartmut Bossel: Umweltdynamik, tewi Verlag
Hans Rauch: Modelle der Wirklichkeit, Heise Verlag

KAPITEL 4

Der Computer als Stratege

Ein beliebtes Test- und Übungfeld für die Methoden der künstlichen Intelligenz stellen nach wie vor die strategischen Spiele dar. Eine überschaubare Anzahl Spielfiguren, ein genau definierter Satz Regeln, so etwas mag der Computer, da er hier ungestört von den oft nicht kalkulierbaren Ereignissen der realen Welt sein Können beweisen kann.

Trotzdem werden in diesem Kindergarten der künstlichen Intelligenz bereits Anforderungen gestellt, die nicht von schlechten Eltern sind. Gefragt ist insbesondere

- kombinatorisches Denken, d.h. die Fähigkeit, mögliche Abläufe im voraus zu berechnen und die Konsequenzen abzuschätzen.
- strategisches Denken, d.h. die Fähigkeit, zielstrebig praktikable Pläne zu entwickeln und durchzuführen.

Keine Frage, daß Maschinen mit diesen Fähigkeiten nicht nur für den schachspielenden Computerfreak interessant sind; auch Militärs und Manager finden hier ein willkommenes Werkzeug, um sich noch effektiver gegenseitig zu bedrohen oder Geld abzuluchsen. Doch die traurige Tatsache, daß maschinelle Intelligenz, gepaart mit menschlicher Dummheit, eine absolut katastrophale Mischung ergeben kann, soll uns nicht daran hindern, in den nächsten Kapiteln des Software-Experiments einige der Grundlagen zu erforschen, die Computer in gewiefte Strategen verwandeln – im Vertrauen darauf, daß Sie, lieber Leser, zu intelligent sind, um die dabei gewonnenen Erkenntnisse im Rahmen eines ausgeklügelten Selbstmord-Systems einzusetzen.

Jedes der Spielprogramme, die in den nächsten Kapiteln zur Sprache kommen, demonstriert ein bestimmtes Arbeitsprinzip und wird dem Charakter des Software-Experiments entsprechend nicht nur als Spielpartner geeignet sein, sondern zum dran drehen und Experimentieren einladen; wie immer

sind Sie zur Mitarbeit aufgerufen. Und soviel sei jetzt schon versprochen: Wir werden keine Trivialspiele wie etwa Tic-Tac-Toe behandeln, das bei dieser Gelegenheit meist herhalten muß, sondern Strategiespiele, die Ihnen mit Sicherheit einiges zu denken geben werden.

Unser erstes Studienprojekt wird das japanische Gobang-Spiel sein (auch als Go-Moku bekannt), eines jener reizvollen archaischen Spiele, die mit einfachen Mitteln sehr interessante und komplexe Abläufe erzeugen und deshalb für die Programmierung besonders gut geeignet sind. Bevor wir uns jedoch diesem Thema zuwenden, werden wir zunächst einen Blick darauf werfen, wie eigentlich der Mensch strategisch spielt.

Von Meistern und Mustern

Nach landläufiger Meinung handelt es sich bei Schachmeistern um Gehirnakrobaten, die in Windeseile eine Vielzahl möglicher Züge vorausberechnen können und aus diesem Grund dem durchschnittlichen Amateurspieler weit überlegen sind. Doch entspricht das wirklich den Tatsachen? Recht überraschend ist zumindest die Antwort, die der amerikanische Schachmeister Marshall auf die Frage gab, wieviel Züge er im allgemeinen vorausberechnet: ‚Zwei, aber dann zwei Gute!' Zunächst einmal fühlt man sich auf den Arm genommen – soll das alles sein? Doch in der Tat hat sich herausgestellt, daß Schachmeister zwar durchaus in der Lage sind, bestimmte Zugfolgen sehr weit zu durchdenken, aber dabei nur eine kleine Auswahl der vielen möglichen Züge in Betracht ziehen, nämlich wie gesagt nur die guten.

Für den Schachprogrammierer ist das sehr frustrierend: Wie soll er diesen schwer zu definierenden Instinkt für die Qualität bestimmter Züge auf seine Maschine übertragen? Hinsichtlich der kombinatorischen Fähigkeiten haben die modernen Schachmaschinen durchaus schon Meisterniveau erreicht, doch es fehlt nach wie vor der Blick für die Situation, der dem Meister auf Anhieb verrät, welche Züge überhaupt betrachtenswert sind.

Unerwartete Schützenhilfe erhält der Programmierer jedoch von den Psychologen, die ebenfalls ein lebhaftes Interesse dafür entwickeln, wie solche geistigen Leistungen zustande kommen. Sie haben herausgefunden, daß der Schachmeister weniger einzelne Figuren und deren Zugmöglichkeiten betrachtet, sondern bestimmte Teilstrukturen auf dem Schachbrett wiedererkennt. Diese Muster werden z.B. durch die Bauernketten in einer bestimm-

ten Eröffnung gebildet, durch die Gruppierung der Offiziere auf der Königsseite oder die Stellung der Figuren im Zentrum. Der Meister sieht diese Muster als Ganzes, so wie wir ein Wort unmittelbar verstehen können, ohne es erst in seine Buchstaben zu zerlegen, und weiß aus Erfahrung, welche Fortsetzungen in den speziellen Situationen besonders erfolgversprechend sind.

Gut, sagt sich der Schachprogrammierer, dann machen wir das genauso. Zunächst schaffen wir ein Festplattenlaufwerk an, um genügend Speicherplatz zu haben, dann bekommt der Rechner eine Liste aller relevanten Teilstrukturen, die beim Schach auftreten können, und zu guter Letzt engagieren wir einen Spezialisten, der die dazugehörigen guten Züge eintippt – fertig ist der elektronische Großmeister! Und dieses Konzept sieht in der Tat sehr erfolgversprechend aus: Leistungsfähige Algorithmen zur Identifizierung von Mustern (Pattern Recognition) sind bereits erforscht und bekannt, an Speicherplatz mangelt es heutzutage auch nicht mehr, und wenn dann noch die Geschwindigkeit und die daraus resultierende Kombinationsstärke moderner Rechner dazukommt… ja, dann wäre wirklich bald ein Computer Schachweltmeister, wenn es nicht noch ein entscheidendes Hindernis gäbe.

Erinnern wir uns noch einmal an die Analogie zur Sprache. Um den Sinn eines Wortes zu erfassen, genügt es meist nicht, das Wort zu identifizieren, sondern wir müssen auch die spezielle Bedeutung des Wortes innerhalb eines Satzes oder sogar eines ganzen Textes erfassen, den Sinnzusammenhang. Es stellt im Prinzip kein Problem dar, einem Computer ein ganzes Wörterbuch einzutrichtern und ihm alle grammatikalischen Regeln beizubringen, aber das heißt noch lange nicht, daß er jetzt die Sprache versteht!

Während der Speicherplatz für die Wörter, Deklinationen, Konjugationen usw. allemal ausreicht, erreicht die Anzahl der möglichen Sinnbeziehungen zwischen verschiedenen Wörtern astronomische Dimensionen und ist programmtechnisch nur sehr schwer in den Griff zu bekommen. Und genau hier liegt der Hund begraben: Der Schachmeister kennt nicht nur eine große Anzahl von Teilstrukturen (Schachwörter), er kann sie auch zueinander in Beziehung setzen und ihre Bedeutung innerhalb eines größeren Zusammenhangs erkennen, und das macht ihm bis heute kein Computer nach.

Aus dieser Tatsache ergibt sich ein wesentlicher Unterschied in der Verfahrensweise bei menschlichen und elektronischen Schachspezialisten: Der Computer muß alle Züge (und meistens noch tausende von Folgezügen) probeweise ausführen, bevor er sie anhand der resultierenden Stellung beurtei-

len kann; der Großmeister beurteilt erst die Stellung und rechnet dann nur ein paar Kombinationen durch. Es liegt auf der Hand, welches Verfahren effektiver ist.

Let's go bang!

Bevor wir dazu kommen, wie sich solche Probleme in der Praxis darstellen, hier zunächst die Spielregeln: Gobang wird ursprünglich auf einem 19x19 Feld gespielt. Kenner dieses ehrwürdigen Spiels mögen verzeihen, daß es für unser Programm auf computergerechte 16x16 Felder zurechtgestutzt wurde; die Programmierung in Assembler wurde auf diese Weise besonders einfach.

Zwei Spieler setzen nun abwechselnd verschiedenfarbige Plättchen auf das Spielfeld, und zwar mit dem Ziel, horizontal, vertikal oder in einer der beiden Diagonalrichtungen eine ununterbrochene Reihe aus fünf eigenen Steinen zu bilden. Gewonnen hat, wer es als erster schafft. Es kommt also darauf an, Züge zu finden, die möglichst in mehreren Richtungen etwas bewirken, also z.B. gleichzeitig eine eigene Reihe ergänzen und eine gegnerische Reihe blockieren. Dabei gilt es, einen guten Überblick zu bewahren – vielleicht ist es ganz sinnvoll, wenn Sie jetzt schon das Programm **GOBANG.BAS** starten und ausprobieren, um sich einen Eindruck von dem Spiel zu verschaffen; das Verständnis für die folgenden Ausführungen wird dadurch erleichtert.

Nach dem Programmstart wird zunächst das Spielfeld gezeichnet, danach können Sie entscheiden, wer anfängt. Wenn Sie am Zug sind, erscheint ein Cursor, der wie üblich mit den Pfeiltasten gesteuert wird. Bewegen Sie ihn einfach zu dem Feld, auf das Sie setzen wollen und drücken Sie dann ENTER, worauf ein roter Stein erscheint (Ihre Farbe) und das Programm seinen nächsten Zug berechnet. Wenn nach 120 Zügen kein Gewinner feststeht, gilt die Partie als unentschieden; nach dem Spielende startet ein beliebiger Tastendruck eine neue Partie. Falls Sie als Gobang-Anfänger zunächst einige saftige Niederlagen hinnehmen müssen, lassen sie sich nicht allzusehr frustrieren: Das Programm ist zu schlagen – aber wie, das müssen Sie schon selbst herausfinden!

Und jetzt zur Programmierung: Was den Programmierer zunächst einmal erschreckt, ist die große Anzahl der Zugmöglichkeiten bei Gobang. Während in Schachstellungen durchschnittlich 40 Züge zur Auswahl stehen, sind

es bei diesem Spiel zumindest in der Anfangsphase mehr als 200, und das bedeutet im Klartext, daß der Computer seine eigentliche Stärke, nämlich die Vorausberechnung aller möglichen Zugfolgen bis zu einer gewissen Tiefe, in diesem Fall nicht ausspielen kann.

Eine kurze Rechnung zeigt sofort die Hoffnungslosigkeit dieses Unterfangens: Um nur zwei Züge weit im voraus zu denken, müßte das Programm auf dem 16x16-Brett zu Beginn insgesamt 256x255x254x253 Stellungen untersuchen, und das wäre selbst für ein schnelles Maschinenprogramm zuviel. Zwar sind aus der Schachprogrammierung Methoden bekamt, um die Anzahl der zu untersuchenden Stellungen zu reduzieren (Alpha/Beta-Algorithmus, mehr dazu im nächsten Kapitel), doch auch das würde bei Gobang nur bedingt helfen – wer hat schon Lust, eine Stunde oder länger auf den nächsten Zug des Rechners zu warten?

Deshalb müssen wir uns wohl oder übel auf die Methode der Meister besinnen und einen Weg finden, den Wert eines Zuges auf Anhieb zu bestimmen, ohne die möglichen Fortsetzungen probeweise durchzuspielen. Von Vorteil ist immerhin, daß es nur zwei verschiedene Arten von Spielsteinen gibt, und die ziehen glücklicherweise nicht auf dem Brett herum, sondern bleiben brav an ihrem Ort.

Idiotisch oder genial?

Der Mensch denkt bei der Spielbewertung qualitativ und spricht in diesem Zusammenhang von idiotischen, schlechten, annehmbaren, guten oder sogar genialen Zügen. Doch mit dieser sprachlichen Klassifizierung kann ein Rechner nur wenig anfangen, seine Domäne sind die Zahlen, und deshalb muß sich der Programmierer Gedanken darum machen, wie die Qualität durch Zahlenwerte dargestellt werden kann. Wäre das Programm in der Lage, alle Konsequenzen eines Zuges bis zum Ende der Partie zu berechnen, so könnte das so aussehen:

$$-1 = \text{verloren} \qquad 1 = \text{gewonnen} \qquad 0 = \text{unentschieden}$$

und ein einfacher arithmetischer Vergleich stellt dann klar, welche Züge besser oder schlechter sind.

Doch wie bereits erwähnt, ist das endgültige Spielergebnis für den Computer wegen der Vielzahl von Zugmöglichkeiten absolut außer Sichtweite,

und deshalb müssen wir uns darauf beschränken, gewisse Teilerfolge zu bewerten, die erfahrungsgemäß eine Bedeutung für den Spielausgang haben. Wenn wir ein Leerfeld betrachten, das für einen potentiellen Zug in Frage kommt – nennen wir es einmal Zielfeld – ist intuitiv klar, daß die Qualität des Zuges irgendetwas damit zu tun hat, wie sich bereits gesetzte Steine um dieses Zielfeld herumgruppieren. Sie bilden ein Muster mit bestimmten Merkmalen, und diese Merkmale geben uns einen Hinweis darauf, ob es sich lohnt, dort einen weiteren Stein hinzusetzen.

Für die Programmierung bietet sich deshalb folgendes Verfahren an: Der Rechner untersucht die Umgebung des Zielfeldes auf das Vorhandensein günstiger Merkmale und vergibt dafür – je nachdem, ob er sie vorfindet oder nicht – nach einem bestimmten System Punkte. Diese Punkte werden anschließend addiert, und der beste Zug ist letztendlich der, dessen Zielfeld die höchste Bewertung erhält. Die knifflige Aufgabe ist dabei weniger die Programmierung der Merkmalserkennung, sondern vielmehr die Aufstellung einer effektiven Bewertungsfunktion, die bestimmt, für welche Merkmale es wieviel Punkte gibt. Hier kann der Programmierer nur sein eigenes Spielverständnis in die Waagschale werfen und durch viele Versuche herausfinden, welches System zu einer akzeptablen Spielstärke führt.

Das Vergnügen, an einer Bewertungsfunktion herumzubasteln, können Sie gleich selbst genießen: CPC-Gobang erlaubt es nämlich, alle relevanten Faktoren beliebig zu ändern. Doch zunächst soll erläutert werden, welche Merkmale überhaupt in Rechnung gezogen werden.

Betrachten wir zu diesem Zweck eine beliebige unvollständige Fünferreihe auf dem Spielfeld, in der unser leeres Zielfeld enthalten ist. Sollten sich in dieser Reihe bereits gegnerische Steine befinden, so ist sie für uns nicht weiter von Belang, da sich mit ihr das Endziel, nämlich die Bildung einer Fünferreihe aus eigenen Steinen, nicht mehr erreichen läßt: Dafür vergeben wir kategorisch 0 Punkte. Anderenfalls wird uns jedoch interessieren, wieviele eigene Steine bereits in diesem Abschnitt liegen – je mehr, desto besser. Doch wieviele Punkte geben wir dafür? Bewährt hat sich folgende Formel:

$$P = \text{Anzahl}^2$$

Für vier bereits vorhandene eigene Steine gibt es also 16 Punkte (und der Zug gewinnt die Partie!), für drei Steine 9 Punkte usw. Das ist schon ein guter Ansatz, doch er genügt noch lange nicht. Zum Beispiel spielt es auch

eine Rolle, wie die eigenen Steine in dem betrachteten Fünferabschnitt verteilt sind. Schauen Sie sich einmal die folgenden Situationen an, in denen das Zielfeld durch X, eigene Steine durch 0 und Leerfelder durch ein Minuszeichen markiert sind:

```
a) 0 0 - 0 X
b) 0 0 0 - X
c) - 0 0 0 X
```

In allen Fünferreihen sind bereits drei eigene Steine vorhanden, doch in der Reihe a liegen sie getrennt, während sie in Reihe b und c im Zusammenhang auftreten. Offensichtlich ist die Reihe c am wertvollsten, da der Zug einen beidseitig offenen Vierer erzeugt und damit zum Gewinn der Partie führt – es sei denn, die Reihe stößt rechts an den Rand oder einen gegnerischen Stein. Die Reihen a und b können dagegen durch einen Zug des Gegners sofort wieder entwertet werden. Dieses Beispiel verdeutlicht, daß neben der Anzahl der Steine drei zusätzliche Merkmale beachtet werden müssen:

1) Liegen die Steine getrennt oder im Zusammenhang? Um diesen Sachverhalt zu erfassen, wird die Variable ZS definiert:

ZS = 1 bei Zusammenhang, ZS = 0 sonst

wobei das Zielfeld bereits vorausschauend als besetzt gilt. Auch in der folgenden Situation ist also ZS = 1:

```
- 0 X 0 -
```

2) Liegt das Zielfeld in der Nähe eines eigenen Steines? Entsprechend wird die Variable NH besetzt:

NH = 1 bei Nähe, NH = 0 sonst

3) Ist die betrachtete Fünferreihe an beiden Enden offen oder an einer Seite blockiert? Hier kommt die Variable OF ins Spiel:

OF = 1 wenn offen, OF = 0 sonst

Insgesamt sieht die Bewertungsfunktion damit so aus:

$$P = \text{Anzahl} \cdot \text{Anzahl} + \text{ZS} + \text{NH} + \text{OF}$$

Als Test können Sie die Punktzahl für die folgende Fünferreihe ermitteln (+ steht für einen gegnerischen Stein, X ist das Zielfeld). Sie sollten dabei auf fünf Punkte kommen (Anzahl=2, ZS=0, NH=1, OF=0):

```
+ 0 - - 0 X
```

Beachtet werden muß allerdings noch, daß ein Zielfeld an insgesamt fünf sich überlappenden Fünferreihen beteiligt ist. Betrachten Sie den folgenden Ausschnitt (+ ist wieder ein gegnerischer Stein oder ein Feld außerhalb des Spielbretts):

```
+ + - - 0 X 0 - 0 - -
```

Wenn wir alle Fünferreihen bewerten, in denen das Zielfeld enthalten ist, erhalten wir von links nach rechts die Punktzahlen 0, 6, 7, 11 und 6. Es ist naheliegend, als resultierenden Wert die maximale Punktzahl zu verwenden, also die 11 Punkte für die Fünferreihe, die mit dem fünften Feld von links beginnt:

$$\text{Wert} = \text{MAX}(\text{P1}, \text{P2}, \text{P3}, \text{P4}, \text{P5})$$

Damit sollte das Programm schon in der Lage sein, zielbewußt zusammenhängende Reihen mit möglichst vielen eigenen Steinen aufzubauen. Doch das allein genügt leider nicht; es gilt ja auch, gewinnträchtige Aktionen des Gegners im Keim zu ersticken! Zu diesem Zweck brauchen wir jedoch nur die gesamte Bewertung zu wiederholen, aber mit vertauschten Rollen: Jetzt gelten die eigenen Steine als feindlich und die Steine des Gegners als Freunde. Nachdem wir auf diese Weise die Situation durch die Augen des Gegenspielers betrachtet haben, können wir der Punktwertung entnehmen, wie gerne er im nächsten Zug einen Stein auf das Zielfeld setzen möchte. Falls sich herausstellt, daß er sehr gerne tun würde, ist es natürlich ratsam, in weiser Voraussicht dort einen eigenen Stein zu plazieren, um diese Aktion zu verhindern.

Damit erhalten wir also für jedes Zielfeld zwei Bewertungen, einen Wert für den Spieler, der gerade am Zug ist (Wert1), und einen Wert für den Gegner (Wert2). Wie setzen wir jetzt diese Ergebnisse zueinander in Beziehung? Mit Sicherheit ist es besser, eine eigene Fünferreihe zu vervollständigen und damit zu gewinnen, als eine Fünferreihe des Gegners zu verhindern – Angriff ist die beste Verteidigung! Deshalb addieren wir zu Wert1 noch einen oder zwei Punkte als Offensivbonus OB und wählen danach das größere Ergebnis als Bewertung:

$$\mathrm{BW} = \mathrm{MAX}(\mathrm{Wert1} + \mathrm{OB}, \mathrm{Wert2})$$

Doch damit sind wir noch nicht am Ende angelangt: Der aufmerksame Leser wird an dieser Stelle mit Recht bemängeln, daß ein sehr wichtiger Aspekt des Gobang-Spieles bisher noch nicht berücksichtigt wurde. Der Reiz des Spieles entsteht ja gerade dadurch, daß jeder Zug gleich in mehrere Richtungen Auswirkungen hat. Die Bewertungsfunktion in der vorliegenden Form analysiert jedoch nur jeweils eine Richtung, also z.B. eine horizontale Reihe.

Also wenden wir das eben beschriebene System nacheinander auf Reihen in allen Richtungen an – horizontal, vertikal und zweimal diagonal – und untersuchen damit praktisch ein sternförmiges Muster, in dessen Zentrum sich das Zielfeld befindet. Insgesamt erhalten wir auf diese Weise vier Bewertungen BW1...BW4, und wieder taucht die Frage auf, wie diese Ergebnisse miteinander verknüpft werden sollen. Doch um es kurz zu machen, hier ein System, daß sich recht gut bewährt hat: Die Resultate für die verschiedenen Richtungen werden zunächst in absteigender Folge sortiert, so daß gilt:

$$\mathrm{BW1} \geq \mathrm{BW2} \geq \mathrm{BW3} \geq \mathrm{BW4}$$

Danach werden die Werte gewichtet, indem jeder mit einem bestimmten Faktor multipliziert wird, und anschließend addiert:

$$\mathrm{Gesamtwert} = \mathrm{F1} \cdot \mathrm{BW1} + \mathrm{F2} \cdot \mathrm{BW2} + \mathrm{F3} \cdot \mathrm{BW3} + \mathrm{F4} \cdot \mathrm{BW4}$$

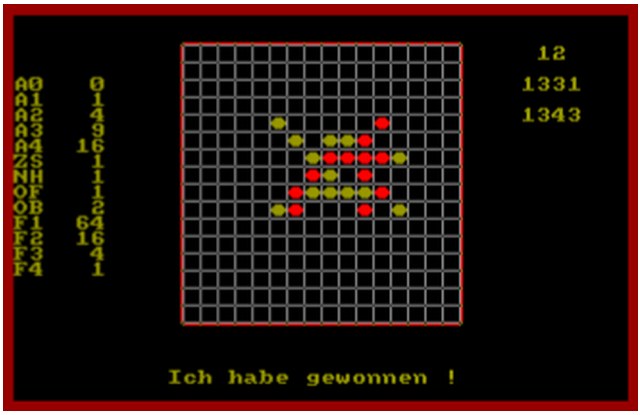
Die Wahl geeigneter Faktoren F1...F4 erfordert einiges Fingerspitzengefühl; unser CPC-Programm spielt jedenfalls mit folgenden Werten schon recht ordentlich:

$$\mathrm{F1} = 64, \quad \mathrm{F2} = 16, \quad \mathrm{F3} = 4, \quad \mathrm{F4} = 1$$

Nach diesem Verfahren braucht der Rechner also nur jedem leeren Feld einen Gesamtwert zuzuordnen, und das Feld mit dem höchsten Wert erhält dann den Zuschlag – unser Programm zieht!

Das Parameter-Tuning

Der letzte Abschnitt hat deutlich gezeigt, wie komplex eine effektive Stellungsanalyse schon bei diesem relativ einfachen Spiel werden kann. Immerhin ist es erstaunlich, wie gut das Programm spielt, ohne die möglichen Folgen der Züge vorauszuberechnen. Leider gibt es noch keine Theorie, die Anhaltspunkte für den Aufbau von Bewertungsfunktionen liefert; auch die in unserem Fall benutzten Punktwertungen beruhen letztendlich auf experimentell erprobten Annahmen. Es ist also sehr wahrscheinlich, daß sich die Spielstärke durch ein Parametertuning weiter steigern läßt – doch damit sind Sie jetzt an der Reihe!



CPC GOBANG stellt für die Einstellung der Bewertungs-Parameter einige Hilfsmittel zur Verfügung: Wenn Sie am Zug sind, können Sie durch COPY die Anzeige der Zugbewertung ein- oder ausschalten. Rechts auf dem Bildschirm erscheint dann für jedes Zielfeld, auf dem sich der Cursor gerade befindet, die entsprechende Bewertung und verrät Ihnen, was der Rechner von Ihren Plänen hält. Weiterhin zeigt das Programm in der Zeile darunter die Bewertung der eigenen Züge an.

Nach Drücken von TAB werden dagegen sämtliche im letzten Abschnitt besprochenen Parameter auf der linken Seite ausgegeben. Um nun einen oder mehrere Werte probeweise zu ändern, begeben Sie sich einfach mit dem Cursor in die entsprechende Zeile, geben den neuen Wert ein und schließen mit ENTER ab. Diese Manipulation können Sie jederzeit auch während einer Partie vornehmen, mit ENTER kehren Sie dann ins Spielgeschehen zurück.

In der Parametertabelle finden Sie zuoberst die Werte A0…A4, die die Punktzahl für eine entsprechende Anzahl eigener Steine in einer unvollständigen Fünferreihe bestimmen. ZS, NH und OF geben an, wieviele Punkte die dazugehörigen Strukturmerkmale erhalten, falls Sie in einer Reihe auftauchen, und OB steht für den Offensivbonus. Den Abschluß bilden die Faktoren F1..F4, mit denen die Ergebnisse für die vier Richtungen gewichtet werden.

Insgesamt stehen also 13 Werte zur Verfügung, an denen Sie drehen können, wodurch eine Reihe interessanter Experimente möglich wird. So ist es zum Beispiel sehr instruktiv, bei ZS, NH und/oder OF eine Null einzutragen, worauf der Rechner diese Merkmale nicht mehr beachtet und ziemlich chaotisch spielt. Auf diese Weise können Sie die Bedeutung der verschiedenen Werte für das Spielverhalten des Programmes erforschen und versuchen, es zu verbessern. Besonders interessant ist es natürlich, nach einer Niederlage des Computers die Ursachen zu analysieren und durch geeignete Änderungen dafür zu sorgen, daß der Fehler in Zukunft vermieden wird.

Zum Abschluß noch ein paar Anregungen und Informationen für eigene Aktivitäten: Nicht nur die Spielstärke, sondern auch der Bedienungskomfort des Programms läßt sich noch verbessern. So könnte man z.B. ein Array einbauen, in dem sich der Rechner alle Züge merkt, um eine Zugzurücknahme oder ein Nachspielen der Partie zu ermöglichen. Auch Zugvorschläge lassen sich mit den vorhandenen Unterprogrammen realisieren. Deshalb hier für alle Bastler die Bedeutung der drei Assemblerroutinen:

CALL &A000 löscht das gesamte Spielfeld, das zeilenweise im Speicher ab Adresse &A300 abgelegt wird. In der internen Darstellung bedeutet eine 0 ein leeres Feld, eine 1 einen Stein des Rechners und eine 2 einen Stein des (menschlichen) Gegners.

CALL &A00D berechnet die Bewertung für ein Feld der Spielfläche. Dazu muß vor dem Aufruf in der Speicherstelle &A1D4 die Spalte und in &A1D5 die Reihe übergeben werden (beide Werte im Bereich 0…15); weiterhin wird der Routine über die Adresse &A1CF mitgeteilt, für welchen Spieler (1 oder

2) die Bewertung durchgeführt werden soll. Nach dem Aufruf befindet sich in &A1D2 das Lowbyte und in &A1D3 das Highbyte des errechneten Wertes. Weiterhin kann der Speicherstelle &A1D0 entnommen werden, ob ein Gewinnzug des Rechners (Bit 1 gesetzt) oder des Gegners (Bit 2 gesetzt) vorliegt. Die Routine überprüft nicht, ob ein Feld bereits besetzt ist!

CALL &A022 ermittelt den Zug mit der maximalen Wertung; auch hier muß vor dem Aufruf über die Speicherstelle &A1CF festgelegt werden, wer gerade am Zug ist. Nach dem Aufruf findet man in &A1D4 die Spalte und in &A1D5 die Reihe des entsprechenden Feldes; die Bewertung und die Gewinnflags werden wie in der vorigen Routine übergeben.

Literaturhinweis: Der Algorithmus für die Bewertungsfunktion wurde dem Buch *Pascal at Work and Play* von Richard S. Forsyth entnommen (Chapman and Hall, London / New York), das noch weitere Leckerbissen dieser Art bietet.

KAPITEL 5

Mit roher Rechengewalt

Das erklärte Ziel der künstlichen Intelligenz ist die Simulation menschlicher Denkvorgänge. Doch ob sich dieses Ziel erreichen läßt, darüber streiten sich die Gelehrten. Sicherlich hat die KI-Forschung schon einige Teilerfolge vorzuweisen, doch unerreicht ist bis heute die ungeheure Flexibilität und Vielseitigkeit des menschlichen Denkens. Alle existierenden angeblich intelligenten Programme beherrschen nur ein sehr begrenztes Wissensgebiet, und der Schachcomputer, mit dem man sich nebenbei über Gott und die Welt unterhalten kann, wird wohl noch einige Zeit auf sich warten lassen.

Aber selbst bei einem Programm, das die Fähigkeiten eines menschlichen Experten erreicht oder sogar übertrifft, sind Zweifel angebracht, ob es sich dabei wirklich um eine intelligente Leistung handelt. Diesen Zweifel entwickelt im allgemeinen weniger der Anwender, der ja nur die Ergebnisse zu sehen bekommt, sondern vielmehr der Programmierer, der seiner Maschine Byte für Byte beibringen muß, wie das Problem gelöst wird. Er weiß, daß die eindrucksvollen Ergebnisse oft nur durch relativ simple Prozeduren erzielt werden, die in irgendwelchen Schleifen mit einem Affenzahn einige hundert Male durchlaufen werden – was hat das mit Intelligenz zu tun?

Auch wenn ein Mensch und eine Maschine bei einer Aufgabe zum gleichen Ergebnis kommen, so ist doch die Methode meist unterschiedlich. Dazu gleich ein Beispiel: Wie würden Sie an das folgende Problem herangehen (ohne Computer)?

Gesucht sind zwei zweistellige ganze Zahlen, deren Produkt 999 beträgt.

Offenbar ist eine gewisse Portion Intelligenz notwendig, um diese Aufgabe ohne überflüssige Rechnerei zu lösen. In der Tat existiert eine einfache Methode, mit der man das Ergebnis sehr schnell ermitteln kann… verraten wird sie hier natürlich nicht! Falls Sie zu den Auserwählten gehören, die das Verfahren auf Anhieb entdecken, seien Sie nicht allzu deprimiert, wenn der

CPC das Ergebnis Ihrer Intelligenzleistung ebenfalls in Sekundenschnelle findet, und zwar mit einem ziemlich primitiven Vierzeiler:

```
10 FOR x% = 10 TO 99
20   FOR y% = 10 TO 99
30     IF x%*y% = 999 THEN PRINT x%; y%: END
40 NEXT y%, x%
```

Zwar verbringt der Computer hier die meiste Zeit mit vollkommen unsinnigen Berechnungen, doch die hohe Ablaufgeschwindigkeit macht diesen Nachteil mehr als wett. Das Problem wird einfach mit roher Rechengewalt gelöst, und nicht umsonst hat dieser Programmtyp in der Fachliteratur den Beinamen *Brute Force* erhalten.

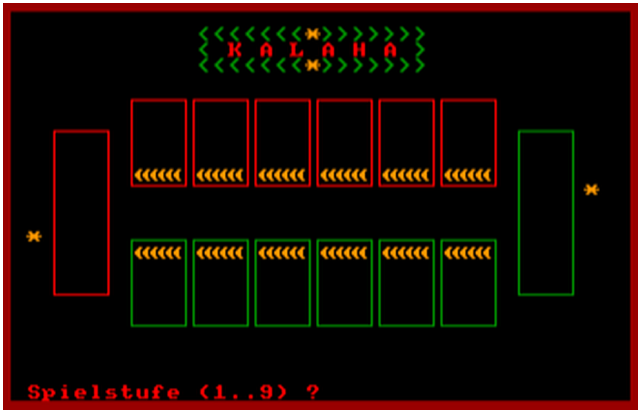
Dieses Beispiel zeigt jedenfalls deutlich, daß die Leistungen von Menschen und Maschinen auf vollkommen verschiedenen Voraussetzungen beruhen. Der menschliche Verstand ist vergleichsweise langsam, aber vielseitig; Computer sind ungeheuer stur, dabei jedoch sehr schnell und präzise. Eine effektive Methode zur Problemlösung muß natürlich diesen unterschiedlichen Hardwareeigenschaften gerecht werden. Auch im Bereich der strategischen Spiele spiegelt sich dieser Unterschied wieder. Programme, die nach der menschlichen Methode vorgehen, untersuchen nur eine kleine Auswahl sinnvoller Züge, müssen jedoch sehr viel strategisches Wissen enthalten und es vor allem korrekt anwenden. Die Spielstärke hängt sehr stark davon ab, wie differenziert das Programm, bei der Analyse vorgeht: Es muß in der Lage sein, eine große Anzahl verschiedener Stellungstypen und Merkmale zu unterscheiden, um ihre Bedeutung im Gesamtzusammenhang richtig einzuschätzen.

Doch wie bereits im letzten Kapitel angedeutet, bringt die Programmierung einer vielseitigen Bewertungsfunktion erhebliche Problem mit sich. Wer versucht hat, das Gobang-Programm durch veränderte Parameter zu verbessern, kann davon sicherlich ein Lied singen: Wie man es auch dreht und wendet – es finden sich immer wieder Situationen, in denen die Zugbewertung nicht greift, d.h. wichtige Tatsachen falsch einschätzt oder einfach übersieht. Eine verfeinerte und damit aufwendigere Bewertungsfunktion könnte zwar in einigen Fällen Abhilfe schaffen, stellt aber den Programmierer vor sehr komplizierte Aufgaben und verlängert natürlich auch die Rechenzeit.

Trotz intensiver Bemühungen ist es bis heute nur ansatzweise gelungen, eine menschliche Vorgehensweise mit ausreichender Effektivität auf einen Computer zu übertragen. Deshalb werden wir uns in diesem Kapitel mit einem anderen Programmtyp beschäftigen, dessen Leistungsfähigkeit mehr auf der Rechengeschwindigkeit beruht als auf feinsinnigen Analyseverfahren – und es ist in der Tat erstaunlich, welche ‚Intelligenz' ein kleiner Z 80-Prozessor mit 4 MHz Taktfrequenz entwickelt, wenn er so richtig loslegen darf! Als Demonstrationsobjekt wird uns diesmal ein Spiel dienen, das zur Abwechslung nicht aus Asien, sondern aus Afrika stammt.

Kalaha – die Guten ins Töpfchen...

Für eine Partie Kalaha braucht man 72 Bohnen und 14 Töpfe. Zu Beginn liegen sechs Bohnen in jedem Topf, nur die beiden Sammeltöpfe an der linken und rechten Seite bleiben leer. Jeder der beiden Spieler hat Zugriff auf die sechs Töpfe, die sich auf seiner Seite befinden. Gezogen wird, indem man aus einem seiner Töpfe die Bohnen entnimmt und sie auf die nachfolgenden Töpfe reihum im Gegenuhrzeigersinn verteilt. Beginnt zum Beispiel der Rechner mit seinem fünften Topf (obere Reihe, von rechts gezählt), so liegen danach in seinem sechsten Topf sieben Bohnen, in seinem Sammeltopf eine Bohne und in den Töpfen 1..4 des Gegners (untere Reihe, von links gezählt) jeweils sieben Bohnen.



Dazu kommen noch zwei Sonderregeln.

- (1) Landet die letzte Bohne in einem eigenen leeren Topf, so erobert man alle Bohnen im gegenüberliegenden gegnerischen Topf und legt sie in seinen Sammeltopf.
- (2) Fällt die letzte Bohne genau in den eigenen Sammeltopf, so darf man noch einmal ziehen.

Ansonsten wird abwechselnd gezogen. Gewonnen hat, wer zuerst mehr als 36 Bohnen in seinen Sammeltopf gebracht hat. Falls ein Spieler nicht mehr ziehen kann, weil seine Töpfe alle leer sind, so hat er verloren. Aus dem Sammeltopf kann nicht gezogen werden, die dort befindlichen Bohnen liegen fest.

Nach dem Start des Programms KALAHA.BAS möchte der Rechner zunächst die gewünschte Spielstufe wissen – je höher die Stufe, um so stärker spielt das Programm, benötigt dafür allerdings auch mehr Rechenzeit. Durch eine weitere Eingabe bestimmen Sie, wer die Partie beginnt. Falls Sie am Zug sind, erscheint ein Pfeil auf dem Bildschirm, der durch die Cursortasten links und rechts gesteuert wird. Durch Drücken von ENTER bzw. RETURN können Sie den Topf anwählen, aus dem Sie ziehen wollen, der Rechner verteilt danach die Bohnen reihum. Auch wenn das Programm seinen Zug ermittelt hat, erscheint zunächst ein Pfeil, der den entsprechenden Topf markiert. Erst nach ENTER wird der Zug ausgeführt, so daß Sie sich in Ruhe anschau-

en können, was Ihr elektronischer Spielpartner sich da ausgedacht hat. Nach Abschluß der Partie startet eine beliebige Taste ein neues Spiel.

Ihre ersten Versuche unternehmen Sie am besten in der ersten oder zweiten Spielstufe, um nicht gleich zu Beginn demoralisiert zu werden. Bereits in der dritten Stufe reagiert das Programm ausgesprochen ungemütlich und ist ab Stufe 6 nur noch sehr schwer zu schlagen. Auch wenn man die Partie eine Zeit lang ausgeglichen halten kann, findet der Rechner irgendwann raffinierte Zugkombinationen, die ihn deutlich in Vorteil bringen – und spätestens hier wird jeder Programmierfreak neugierig, wie diese Intelligenzleistung zustande kommt.

Einen kleinen Hinweis liefert immerhin die Anzeige der Stellungsbewertung in der linken oberen Bildschirmecke: Die erste Zahl gibt während des Suchvorgangs den bislang besten Zug an, die zweite Zahl den dazugehörigen Erwartungswert. Bei einem positiven Wert sieht sich das Programm im Vorteil, bei einem negativen Wert im Nachteil, und der Wert 0 deutet auf eine ausgeglichene Stellung hin. Falls eine Bewertung von +99 erscheint, so hat der Rechner einen sicheren Gewinnweg entdeckt, bei −99 sieht er dagegen keine Chance mehr und gibt auf.

Wie Sie aus dem Listing ersehen können, wird die Zugberechnung durch ein Maschinenprogramm vorgenommen. Den Datazeilen kann man natürlich nicht mehr ansehen, wie das Programm dabei vorgeht. Deshalb werden wir es jetzt durch eine BASIC- Routine ersetzen, die zwar längst nicht so leistungsfähig ist, dafür aber anschaulich das Prinzip demonstriert. Gehen Sie zu diesem Zweck auf folgende Weise vor:

- Löschen Sie die Programmzeile 90, 1140 und alle Zeilen ab 5000.
- Ergänzen Sie die Basic-Zugberechnung *Zug Computer V1*, indem Sie sie mit **MERGE „ZUGCOMV1“** dazuladen.

Die aus dem Hauptprogramm entfernte Zeile 1140 sorgt normalerweise für die Einhaltung der Sonderregel (2). Die folgenden Betrachtungen werden jedoch wesentlich verständlicher, wenn beide Spieler streng abwechselnd ziehen. In Kauf nehmen müssen wir dabei, daß Kalaha ohne diese Regel einiges an Spielwitz einbüßt. Wer also nur spielen will, ist mit der Grundversion besser beraten.

Für das Verständnis der folgenden Ausführungen ist es sehr günstig, wenn Ihnen ein Listing der Basic-Zugberechnung vorliegt. Sie können es mit LIST

5000- auf den Bildschirm bzw. mit LIST 5000-,#8 auf dem Drucker ausgeben lassen.

Die Zugbewertung: Etwas kurzsichtig

Wie ein erster Versuch zeigt, spielt die BASIC-Version V1 zwar ausgesprochen schwach (die Eingabe der Spielstufe ist hier ohne Bedeutung), doch dafür ist die Vorgehensweise sehr leicht zu durchschauen: Alle möglichen Züge des Rechners (maximal 6) werden probeweise ausgeführt und danach die Differenz aus der Anzahl der Bohnen im Sammeltopf des Rechners und des Gegners gebildet – je größer das Ergebnis, um so besser der Zug. Da diese Zugbewertung einem ausgeglichenen Spielstand den Wert 0 zuordnet, handelt es sich im Gegensatz zu der Gobang-Bewertungsfunktion im letzten Kapitel um eine symmetrische Funktion

Das Spielfeld wird intern durch das Array in(14,9) repräsentiert. Die Variablen in(1,0)…in(14,0) enthalten die Anzahl der Bohnen, wobei der Gegner die Töpfe 1…6 und der Rechner die Töpfe 8…13 belegt; Nr. 7 und Nr. 14 sind die Sammeltöpfe. Da wir noch einige Hilfsspielfelder benötigen, wurde das Array mit einem zweiten Index versehen, also mehrstöckig aufgebaut. Mit diesen Informationen und einigen ergänzenden Kommentaren sollte das Listing keine Rätsel mehr aufgeben.

Zeile 5020:

Die Bewertung wird mit –99 initialisiert, dem denkbar schlechtesten Ergebnis. Sie bleibt erhalten, falls das Programm keinen legalen Zug findet, und signalisiert in diesem Fall dem Hauptprogramm, daß es an der Zeit ist, aufzugeben.

Zeile 5040:

Falls der Topf, aus dem gezogenen werden soll, leer ist, so werden die folgenden Schritte übersprungen.

Zeile 5050-5080:

Um den Zug auszuführen, wird das Spielfeld auf ein Hilfsspielfeld kopiert und bei dieser Gelegenheit umgedreht. Durch die vertauschten Seiten erhält der Rechner passen zur FOR…NEXT-Schleife die Topfnummern 1…6. Eine Laufvariable von 8…13 würde es zwar auch tun, aber da

wir dieses Verfahren für die kommenden Versionen ohnehin brauchen, wird es der Einfachheit halber schon jetzt eingeführt.

Zeile 5090-5140:
Hier wird der Zug ausgeführt. Die Bohnen werden dem Topf entnommen, in die Variable anz (Anzahl) übertragen und durch die FOR…NEXT-Schleife auf die folgenden Töpfe verteilt.

Zeile 5150:
Falls die letzte Bohne in einen eigenen leeren Topf gefallen ist, wird die Sonderregel (1) durchgeführt.

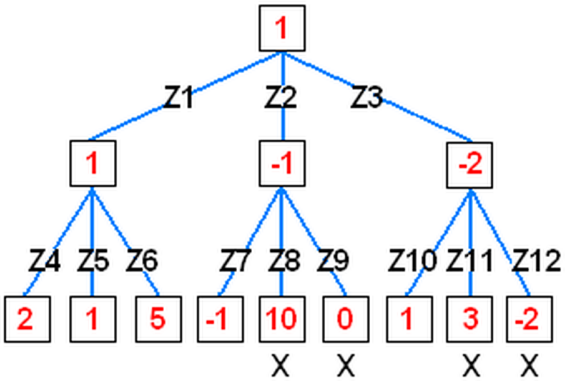
Zeile 5160:
Züge die zum Gewinn des Gegners führen, werden übergangen.

Zeile 5170-5180:
Der Zug wird bewertet und in der Variablen bestzug notiert, falls er das bislang beste Ergebnis übertrifft.

Solch eine einfache Zugbewertung hat natürlich den Vorteil, daß sie sehr schnell berechnet werden kann. Nur macht sich immer wieder katastrophal bemerkbar, daß die Antwort des Gegners in keiner Weise berücksichtigt wird. Was nützt ein stolzes Ergebnis von +3, wenn der Gegenzug die Bewertung auf –10 herabdrückt? Diese Art von Kurzsichtigkeit, in der Fachliteratur auch als Horizonteffekt bekannt, läßt sich durch eine statische Zugbewertung nur schlecht vermeiden. Nötig ist deshalb eine dynamische Bewertung, bei der auch die Gegenzüge probeweise ausgeführt werden.

Baumsuche mit Minimax

Wie wird eine solche dynamische Methode nun in der Praxis realisiert? Anstelle der Zugbewertung muß in Zeile 5170 ein weiteres Unterprogramm aufgerufen werden, das den besten Gegenzug ermittelt. Der Gegner wird natürlich versuchen, möglichst wenige Bohnen in den Sammeltopf des Rechners zu bringen, aber dafür den eigenen Sammeltopf so weit es geht füllen. Wenn wir wie gehabt die Differenz zwischen diesen beiden Werten heranziehen, ist der beste Zug des Gegners genau der, bei dem die Bewertung minimal wird. Der Spielbaum im folgenden Diagramm verdeutlicht das Verfahren.



Angenommen wird dabei, daß dem Rechner und seinem Gegner jeweils drei Zugmöglichkeiten zur Verfügung stehen. Wenn der Rechner den Zug Z1 auf der linken Seite ausführt, müssen die Antworten Z4…Z6 überprüft werden, und wie aus dem Diagramm ersichtlich wird, kann der Gegner die Bewertung höchstens auf 1 herabdrücken. Dieses Minimum stellt auch gleichzeitig den Wert des Rechnerzugs dar, die Bewertung wird also quasi in die vorherige Ebene heraufgereicht.

Wählt der Rechner den Zug Z2, so kann in einer der Varianten sein Vorsprung sogar auf 10 Bohnen hochschnellen – doch das Risiko ist zu hoch: Ein versierter Gegner wird natürlich mit Z7 antworten und sich damit einen Vorsprung von einer Bohne sichern. Die endgültige Bewertung für den Zug Z2 beträgt also −1, und damit ist er trotz der verführerischen Aussicht von +10 effektiv schlechter als der Zug auf der linken Seite. Das Programm sucht also letztendlich das Maximum der heraufgereichten Minima, wodurch klar wird, warum dieses Verfahren kurz und bündig den Namen Minimax erhalten hat.

Ohne Zweifel wird ein Programm mit dieser Methode besser spielen als mit der kurzsichtigen statischen Bewertung. Im Auge behalten muß man jedoch, daß die statische Bewertung nicht aufgehoben, sondern nur aufgeschoben wurde. Wenn das Programm auf diese Weise den besten Zug seines Gegners ermittelt, ist also nach unseren bisherigen Erfahrungen nicht unbedingt sicher, ob es sich wirklich um den besten Zug handelt. Die Gefahr von Fehleinschätzungen ist zwar vermindert, aber noch längst nicht beseitigt!

Doch was hindert uns daran, auch die beste Antwort auf den Zug des Gegners mit der dynamischen Minimax-Methode zu berechnen? Wir müssen ja nur den Spielbaum um eine weitere Stufe ergänzen, in der die möglichen Antworten das Rechners ausprobiert werden. Dazu benötigen wir eine weitere Unterprogrammebene, in der nach bewährtem Schema wiederum der optimale Zug des Rechners (und das dazugehörige Maximum gesucht wird. Doch damit noch nicht genug: Auch diesen Zug könnte man wieder dynamisch ermitteln…

Doch um es kurz zu machen: Theoretisch ist mit dem Minimax-Verfahren zwar eine Untersuchung des kompletten Spielbaumes bis zum Spielende möglich, doch in der Praxis wird dieses Vorhaben durch die explosionsartig anwachsende Anzahl der Verzweigungen in den tieferen Ebenen verhindert. Wenn man annimmt, daß in einer Kalaha-Partie durchschnittlich fünf Zugmöglichkeiten vorhanden sind, so erfordert eine Minimax-Analyse bis zur n-ten Ebene die Bewertung von 5^n Endpositionen. Eine Vorausberechnung von 12 Zügen (sechs für den Rechner und sechs für den Gegner) erzeugt bereits 244 Millionen Endstellungen, und hier tauchen selbst für sehr leistungsfähige Computer unüberwindliche Grenzen auf. Dabei ist der Verzweigungsfaktor bei Kalaha noch vergleichsweise harmlos: In Schachstellungen sind z.B. durchschnittlich 40 verschiedene Züge möglich.

Auch wenn die Maschine noch so schnell ist – irgendwo muß also eine Grenze gesetzt und die Stellung statisch abgeschätzt werden. Klar ist jedoch, daß ein Programm um so stärker spielt, je weiter es mit seiner Analyse im Spielbaum vordringt. Natürlich steigt damit auch die Rechenzeit an, oder anders gesagt: Die Spielstärke bei einem vorgegebenen Zeitlimit hängt unmittelbar von der Geschwindigkeit des Computers ab.

Rekursion – auch in BASIC!

Wie wird die Minimax-Baumsuche nun programmtechnisch realisiert? Zunächst scheint es so, als müßte man für jede Ebene ein separates Unterprogramm schreiben, wobei erst das letzte die statische Bewertung durch Differenzbildung vornimmt. Doch der versierte Programmierer erkennt sofort, daß diese Unterprogramme alle sehr ähnlich aussehen, und fragt sich natürlich, ob man nicht auch mit einer einzigen Subroutine auskommen könnte, die sich dann schlauerweise selbst aufruft.

In der Tat existiert ein als Rekursion bekanntes Verfahren, mit dem sich dieses Problem sehr elegant lösen läßt Obwohl die Grundvoraussetzung für rekursive Programmstrukturen – nämlich die Möglichkeit verschachtelter GOSUBs – in BASIC gegeben ist, tauchen hier jedoch einige Schwierigkeiten auf. Schauen Sie sich zunächst die folgende stark vereinfachte Struktur an:

```
100 st=st+1
110 FOR zug=1 TO 6
120   'Spielfeld kopieren'
130   'Seiten vertauschen'
140   'Zug ausführen'
150   IF st=stmax THEN 'statistische Bewertung' ELSE GOSUB 100:
      ↪  'dynamische Bewertung'
160   IF 'neuer Bestwert' THEN 'Zug und Bestwert merken'
170 NEXT zug
180 st=st-1: RETURN
```

Hier wurde vorsorglich eine Abbruchbedingung eingebaut: Sobald die Suchtiefe st eine vorgegebene Grenze stmax erreicht, nimmt das Programm die endgültige Bewertung vor und beendet die Rekursion. Natürlich muß die Variable st bei jedem Aufruf um 1 erhöht und jedem RETURN um 1 erniedrigt werden, damit das Programm jederzeit weiß, auf welcher Ebene es sich gerade befindet. Weiterhin wird an dieser Stelle deutlich, welchen Vorteil der Programmteil *Seiten vertauschen* bietet: Er sorgt automatisch dafür, daß immer abwechselnd Züge des Rechners und des Gegners ausgeführt werden, so daß wir für beide Seiten mit einem Unterprogramm auskommen.

Ärger wird es jedoch mit der Schleifenvariablen zug geben, da sie nach jedem GOSUB 100 neu initialisiert wird und damit den Zählerstand auf der vorherigen Ebene durcheinanderbringt. Hier macht sich unangenehm bemerkbar, daß BASIC, im Gegensatz zu Pascal oder LOGO, keine lokalen Variablen kennt, die trotz Namensgleichheit eindeutig nur einem Unterprogramm bzw. einer Unterprogrammebene zugeordnet sind. Ähnliche Überlegungen gelten übrigens für die Variablen bestwert und bestzug: Auch hier braucht jede Ebene ihre eigene Schublade.

Was also tun? Die erste Idee ist natürlich, ein Array zug(st) zu benutzen,

```
110 FOR zug(st) = 1 TO 6
```

und das sieht zwar sehr schön aus, funktioniert aber leider nicht, da das Locomotive-Basic keine indizierten Variablen in FOR… NEXT-Schleifen

gestattet: Syntax Error! Glücklicherweise steht uns mit WHILE...WEND noch eine andere Schleifenkonstruktion zur Verfügung, die das Problem zwar nicht ganz so elegant löst, aber dafür indizierte Variablen akzeptiert:

```
105 zug(st) = 6
110 WHILE zug(st) > 0
:
:
:
165   zug(st) = zug(st) - 1
170 WEND
```

Doch jetzt genug der grauen Theorie! Das Unterprogramm *Zug Computer V2* realisiert alle bisher erläuterten Ideen; Sie können es mit **MERGE „ZUG-COMV2“** anstatt der alten V1-Version in das Programm integrieren. Auch hier sollten Sie sich mit LIST 5000-,#8 einen Ausdruck erzeugen, damit die folgenden Ausführungen verständlich werden.

Die maximale Suchtiefe stmax wird durch die Eingabe der Spielstufe nach dem Programmstart bestimmt; allerdings sollten Sie zunächst keinen höheren Wert als 3 vorgeben, da sonst die Rechenzeit ungemütlich lang wird. In der Spielstufe 1 spielt das Programm zwar genauso schwach wie die Version V1, doch schon in der zweiten Stufe ist eine deutliche Steigerung zu bemerken: Der Rechner stolpert nicht mehr in jede Falle, da ihn die Vorausberechnung des Gegenzuges meist schon verrät, was Sie im Schilde führen. In der dritten Stufe erweist er sich schließlich als recht zäher Gegner, und man muß sich schon anstrengen, um noch zu gewinnen.

Die Struktur des neuen Programm-Moduls entspricht weitgehend der Version V1. Dazu noch ein paar ergänzende Hinweise:

Zeile 5020-5050:
Dieser Vorspann initialisiert die Suchtiefe st mit 0, ruft das eigenliche rekursive Unterprogramm auf und organisiert die Variablenübergabe an das Hauptprogramm.

Zeile 5110-5140:
Das Spielfeld wird kopiert und umgedreht. Natürlich hat jede Ebene ihr eigenes Hilfsspielfeld in(topf,st).

Zeile 5150-5210:
Der Zug wird ausgeführt.

Zeile 5230:

Nach dem rekursiven Aufruf wird die Bewertung des besten Zuges von der nächsttieferen Ebene übernommen. Da das Umdrehen des Spielfeldes auch die beiden Sammeltöpfe vertauscht, erscheint die Bewertung mit einem falschen Vorzeichen, was korrigiert werden muß.

Zeile 5240:

Zunächst verwirrt hier, daß entgegen dem Minimax-Prinzip auch bei Zügen des Gegners scheinbar ein Maximum gesucht wird. Da jedoch, wie gesagt, bei jedem rekursiven Aufruf das Vorzeichen der Bewertungsdifferenz in(7,st) – in(14,st) umgedreht wird, ist das Maximum auf jeder zweiten Suchebene in Wirklichkeit ein Minimum… als ob es nicht schon kompliziert genug wäre, mag mancher Leser hier bemängeln – aber die elegante Lösung mit nur einem Unterprogramm hat eben ihren Preis. Zum Verständnis der ganzen Angelegenheit trägt jedenfalls die Anzeige der aktuellen Suchtiefe in der linken oberen Bildschirmecke bei: Hier können Sie beobachten, wie das Programm im Spielbaum herumklettert.

Die Alpha-Beta-Baumschere

Ein großer Nachteil des Minimax-Verfahrens ist leider das starke Anwachsen der Rechenzeit mit jeder zusätzlichen Suchebene. Bereits in Stufe 4 wird sie unerträglich lang. Deshalb soll noch ein ergänzendes Verfahren vorgestellt werden, das es auf überraschend einfache Weise ermöglicht, große Bereiche des Spielbaumes zu ignorieren und trotzdem das gleiche Ergebnis wie bei einer vollständigen Suche zu erhalten.

Werfen wir zu diesem Zweck noch einmal einen Blick auf das Bild, das den Spielbaum darstellt. Nachdem das Programm den Zug Z1 mit allen Konsequenzen durchgerechnet hat, erwartet es also zumindest einen Wert von +1 und notiert sich das Ergebnis in der Variablen bestwert(1). Danach wird Z2 untersucht, und bereits die Betrachtung des ersten Gegenzuges Z7 stellt klar, daß der Gegner auf jeden Fall das Ergebnis –1 erreichen kann. Da wir ja aus dieser Ebene das Minimum der Bewertungen nach oben übertragen, wird die Bewertung durch die Untersuchung der weiteren Gegenzüge Z8 und Z9 vielleicht noch schlechter, aber mit Sicherheit nicht besser werden.

Der aus der Untersuchung von Z1 stammende Erwartungswert kann also durch Z2 nicht übertroffen werden, und da diese Tatsache bereits nach

der Bewertung des Gegenzuges Z7 feststeht, können die Züge Z8 und Z9 mitsamt allen weiteren Verzweigungen ignoriert werden, was in dem Diagramm durch ein X angedeutet wird. Für den Zug Z3 gelten ähnliche Überlegungen: Zwar wird der Erwartungswert durch Z10 erreicht, aber nicht verbessert, so daß auch hier weitere Varianten keine Rolle mehr spielen – es sei denn, man möchte das Programm zwischen gleichwertigen Zügen nach einem Zufallsprinzip wählen lassen.

De Erwartungswert für Züge des Rechners wird im allgemeinen mit dem griechischen Buchstaben Alpha bezeichnet; er entspricht unserem bestwert(st) bei ungeradzahligem st. Ein Zug gilt als widerlegt, wenn Alpha durch die Bewertung eines Gegenzuges unterboten wird; das Abschneiden der weiteren Gegenzüge wird *Alpha-Cutoff* genannt. Entsprechend läßt sich dieses Verfahren auch auf die dynamische Bewertung der Gegenzüge übertragen: Der Erwartungswert für den Gegner wird mit Beta bezeichnet, und der Cutoff erfolgt, falls die Antwort des Rechners diesen Wert überbietet (der Gegner sucht ja ein Minimum!).

Trotz dieser komplizierten Überlegungen ist erstaunlich wenig Aufwand nötig, um den Alpha-Beta-Algorithmus in unser Programm einzubauen. Ergänzen Sie im Unterprogramm *Zug Computer V2* die folgende Zeile:

```
5245 IF -wert <= bestwert(st-1) THEN zug(st) = 0
```

Durch Herabsetzen des Zugzählers auf 0 wird die WHILE...WEND-Schleife bei nächster Gelegenheit verlassen, alle weiteren Züge bleiben unberücksichtigt. Auch hier muß natürlich der durch das Umdrehen des Spielfeldes erzeugte Vorzeichenwechsel beachtet werden. Weiterhin besteht die Gefahr, daß das Programm in der ersten Ebene (st=1) auf den nicht definierten bestwert(0) zugreift. In der ersten Ebene dürfen natürlich keine Züge abgeschnitten werden, was durch die folgende Ergänzung zuverlässig verhindert wird:

```
5020 st = 0: bestwert(0)= -99
```

Nach erneutem Programmstart können sie anhand der Tiefenanzeige beobachten, wie große Bereiche des Spielbaumes übergangen werden, wodurch das Programm erheblich schneller wird. Sogar die Stufe 4 bietet jetzt erträgliche Rechenzeiten. Erwähnt werden muß allerdings noch, daß die Effektivität des Alpha-Beta-Algorithmus stark von der Reihenfolge abhängt,

in der die Züge untersucht werden. Je eher der beste Zug auftaucht, um so mehr kann der Spielbaum reduziert werden. Wenn z.B. in dem im Diagramm dargestellten Spielbaum die Suche von rechts nach links erfolgt, so erhalten wir keinen einzigen Cutoff! Aus diesem Grund werden die Züge bei vielen Programmen zunächst einmal durch eine relativ flache Suche abgeschätzt und dann vorsortiert, bevor die eigentliche Analyse beginnt – eine der vielen Verfeinerungen der Minimax-Methode, mit denen man problemlos ein ganzes Buch füllen könnte.

In diesem Kapitel müssen wir uns jedoch auf die wesentlichen Grundlagen beschränken, die praktisch bei allen strategischen Spielprogrammen eine Schlüsselrolle einnehmen. Immerhin vermitteln die vorgestellten Beispiele in BASIC eine deutliches Bild von der Funktionsweise des Kalaha-Maschinenprogrammes, wenn auch die Sonderregel 2 noch einige besondere Tricks erfordert, die hier leider nur angedeutet werden können: Wenn ein Spieler noch einmal am Zug ist, wird das Spielfeld natürlich beim Aufruf der nächsten Ebene nicht umgedreht, und auch der Vorzeichenwechsel der Bewertungsfunktion entfällt. Weiterhin darf in diesem Fall auch kein Alpha-Beta-Schnitt erfolgen... alles etwas kompliziert, aber machbar! Zusätzlich wurde das Maschinenprogramm noch durch einen Zufallsgenerator ergänzt, der bei gleichwertigen Zügen die Entscheidung trifft und für Abwechslung sorgt.

Die BASIC-Versionen der Zugberechnung wurden geschrieben, um Ihnen Material in die Hände zu geben, mit dem Sie nach Herzenslust experimentieren können. Scheuen Sie sich also nicht, Veränderungen vorzunehmen und eigene Ideen auszuprobieren: Nur durch die Praxis gewinnt man die nötige Erfahrung, um selbst ‚intelligente' Programme zu schreiben.

KAPITEL 6

Der Computer lernt

Wie in der letzten Folge deutlich wurde, fällt es schwer, einem Programm Intelligenz zuzusprechen, dessen Vorgehensweise man vollständig durchschaut hat. Zwar spielt das Kalaha-Programm clever genug, um in den höheren Spielstufen den meisten Menschen überlegen zu sein, doch zu offensichtlich wird die Rolle des Computers als elektronischer Knecht, der nur eine Folge vorgegebener Anweisungen ausführt. Jeder Taschenrechner kann schließlich besser mit Zahlen umgehen als ein Mensch, ohne deshalb als intelligent zu gelten.

Was unseren bisherigen Programmbeispielen fehlte, war die Fähigkeit zu wirklich eigenständigen geistigen Leistungen, die sogar für den Programmierer zu überraschenden Ergebnissen führen. Zwar erwecken manche Programme besonders in der Entwicklungsphase durchaus den Eindruck, als ob sie über eine gehörige Portion Eigenleben verfügen, doch im allgemeinen ist das mehr auf die Unvollkommenheit des Programmierers als auf maschinelle Intelligenz zurückzuführen. Ein gründliches Debugging bereitet diesen ersten Anzeichen einer Roboterrebellion meist ein schnelles Ende.

Was dem Normalprogrammierer ein Greuel ist – nämlich ein Programm, das in Eigeninitiative Speicherinhalte oder sogar sich selbst verändert – gilt dagegen im Bereich der künstlichen Intelligenz als erstrebenswertes Ziel. Lernfähigkeit heißt das Thema, das nach wie vor für die KI-Forscher eine große Herausforderung darstellt, und das nicht nur aus wissenschaftlichem Ehrgeiz. Lernende Programme könnten z.B. erheblich dazu beitragen, die enormen Kosten der Softwareentwicklung und -installation zu senken, indem sie sich selbständig an unterschiedliche Anforderungen der Anwender anpassen. Der Programmierer stände damit nicht mehr vor dem Dilemma, entweder ein hochspezialisiertes Programm zu schreiben, das vielleicht nur für eine einzige Anwendung zu gebrauchen ist, oder aber ein vielseitiges Programm, das jedoch sehr kompliziert und unhandlich wird.

Trotz dieser attraktiven Aussichten ist die Realisierung solcher Systeme noch nicht besonders weit fortgeschritten. Auch wenn teilweise schon recht überzeugende Demonstrationen lernfähiger Programme existieren (z.B. elektronische Ratten, die von selbst den Weg durch ein Labyrinth finden), so sind die Anwendungsgebiete jedoch meistens noch zu akademisch, um eine praktische Bedeutung zu gewinnen. Weiterhin brauchen viele dieser Experimentalprogramme teure Großrechenanlagen, um effektiv zu arbeiten, und auch eine gewisse psychologische Hemmschwelle sollte man nicht unterschätzen: Maschinen, die über vorgegebene Anweisungen hinaus Eigeninitiative entwickeln, haben schon etwas Unheimliches an sich. Zahlreiche schaurige Visionen in Science Fiction-Romanen geben einen deutlichen Hinweis auf die bei vielen Menschen vorhandenen Ängste.

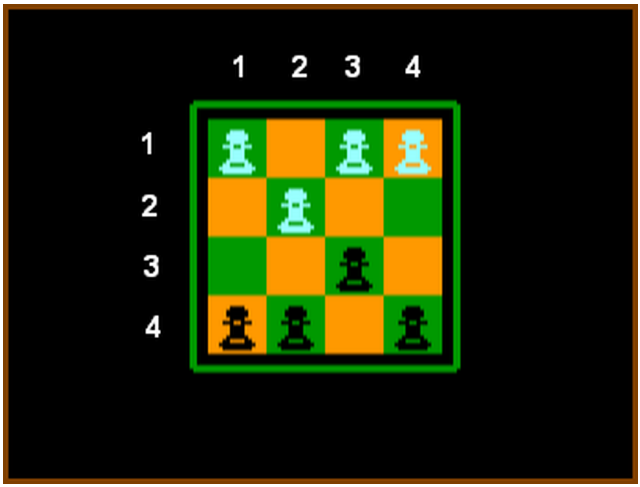
Untersucht man jedoch die verschiedenen Voraussetzungen, die ein Programm erfüllen muß, um zumindest annähernd eine Lernfähigkeit im menschlichen Sinne zu realisieren, so macht sich schnell Ernüchterung breit. Zu kompliziert ist die Palette der menschlichen Wahrnehmungs- und Verarbeitungsmöglichkeiten, um sie einfach auf eine Maschine übertragen zu können, und nach dem heutigen Stand der Technik sind selbständig denkende machthungrige Roboter nach wie vor in den Bereich der Fiktion zu verweisen. Viel realistischer ist dagegen die Bedrohung, die von einem simplen Programmfehler in einem militärischen Abwehrsystem ausgehen kann!

Minischach: Klein, aber oho!

Um die Möglichkeiten und Probleme maschineller Lernfähigkeit zu untersuchen, dient uns in dieser Folge ein Mini-Schachspiel, das extra zu diesem Zweck erfunden wurde. Das Spielgeschehen findet auf einem 4x4-Brett statt, auf dem zu Beginn wie im Bild ersichtlich acht Bauern aufgestellt werden.

Wie bei dem großen Original gehen sie jeweils einen Schritt geradeaus in Vorwärtsrichtung; eine andere Figur schlagen können sie nur diagonal. Sobald ein Bauer die letzte Reihe erreicht, verwandelt er sich in eine Dame, die sich in alle Richtungen beliebig weit bewegen kann. Um die Zugmöglichkeiten und damit auch den Speicherbedarf unseres lernfähigen System nicht unnötig auszuweiten, wurde allerdings ein Schlagzwang für die Damen eingeführt: Sie müssen also eine andere Figur schlagen, falls sie kön-

nen. Verloren hat ein Spieler, wenn er nicht mehr ziehen kann, weil er keine Figuren mehr besitzt, oder seine Bauern alle blockiert sind. Die Partie gilt als remis, wenn vier Züge lang (zwei auf jeder Seite) kein Bauer bewegt und keine Figur geschlagen wurde.



Zu Beginn fragt das Programm, ob bereits eine Datei auf Datenträger mit den Ergebnissen vorangegangener Partien existiert. Falls ja, so wird sie geladen, anderenfalls führt der Rechner sofort seinen ersten Zug aus. Daß er die Partie immer beginnen darf, hat übrigens seinen guten Grund: In einer früheren Testversion spielte das Programm als Nachziehender nur mit den schwarzen Steinen und sorgte dabei – wie es sich für ein lernfähiges Programm gehört – gleich für eine große Überraschung. Nachdem es eine erhebliche Anzahl saftiger Niederlagen eingesteckt hatte, behauptete es plötzlich bereits nach meinem ersten Zug ‚Ich habe verloren' und wollte um keinen Preis weiterspielen. Nicht etwa, daß mein CPC jetzt beleidigt war. Er hatte nur entdeckt, daß der Anziehende das Spiel bei optimaler Strategie auf jeden Fall gewinnt! Also wurde der Spieß umgedreht, und in der vorliegenden Version muß das Programm beweisen, daß es diesen Gewinnweg selbständig herausfinden kann.

Doch jetzt zur weiteren Bedienung: Sie können einen Zug eingeben, indem Sie den Cursor zunächst zur gewünschten Figur steuern und ENTER oder COPY drücken. Begeben Sie sich dann mit dem Cursor zu dem Zielfeld;

ein zweiter Druck auf die ENTER- oder COPY-Taste führt den Zug aus. Illegale Züge werden nicht akzeptiert, das Programm reagiert in diesem Fall nur mit einem Warnton. Beachten Sie, daß kein Bauernzug möglich ist, wenn Ihre Dame eine gegnerische Figur schlagen kann! Wenn Sie nach dem Partieende die Frage ‚Noch ein Spiel' mit N beantworten, so schreibt das Programm alle bislang vorliegenden Erkenntnisse unter dem Namen **MSCHACH.DAT** auf Datenträger und wird danach beendet.

Schwer von Begriff?

Nach den ersten Partien gegen das Minischach-Programm scheint diese Frage wirklich einige Berechtigung zu haben. Munter verliert es bei konsequentem Gegenspiel eine Partie nach der anderen, ohne dabei bemerkenswerte Fortschritte zu machen, und der Fachmann ahnt bereits, was hier abläuft: Die Zugauswahl erfolgt rein zufällig. Bei genauer Beobachtung kann man allerdings feststellen, daß das Programm nie zweimal auf die gleiche Weise verliert. Früher oder später wird es auch zu Situationen kommen, in denen das Programm aufgibt, bevor es endgültig mattgesetzt wird; es erkennt den unvermeidlichen Verlust bereits im voraus.

Diese Lerneffekte werden um so deutlicher, je konsequenter Sie bestimmte Varianten bevorzugen (soweit es der Zufallsgenerator zuläßt). Ergänzend sei hier weiterhin bemerkt, daß das Programm seine Erfahrungen auch auf spiegelbildliche (d.h. an der vertikalen Mittelachse gespiegelte) Situationen anwenden kann. Trotzdem läßt es sich nicht verheimlichen, daß es im Vergleich zum Menschen nervtötend langsam lernt. Daß eine Verlustpartie grundsätzlich nicht wiederholt wird, gibt natürlich etwas Hoffnung: Früher oder später muß sich die Spielstärke dabei ja verbessern, es fragt sich nur, nach wieviel Partien.

Im weiteren Verlauf dieses Kapitels wird noch eine Programmänderung besprochen werden, die Ihnen einen großen Teil der Lehrtätigkeit abnimmt. Bevor wir dazu kommen, soll jedoch genau erläutert werden, wie das Programm überhaupt funktioniert, welche Ursachen seine geistige Trägheit hat und was beim Menschen anders läuft. Zu diesem Zweck beginnen wir mit der internen Darstellung des Spielbretts, die sehr einfach ist: Es handelt sich um ein Array namens b(sp,re), in dem folgende Werte stehen können:

−1	Randfeld außerhalb
0	leer
1	weißer Bauer
2	weiße Dame
3	schwarzer Bauer
4	schwarze Dame

Anstatt des eigentlich nur notwendigen 4x4-Arrays wird also ein 6x6-Feld mit Umrandung benutzt, die bei der Untersuchung der Zugmöglichkeiten sehr hilfreich ist, wie sich noch zeigen wird. Werfen wir jetzt einen Blick auf das Minischach-Listing, das Sie sich auf dem Drucker oder Bildschirm ausgeben lassen sollten, um die weiteren Erläuterungen verfolgen zu können. Die Aufgabe des Initialisierungsteils ist im wesentlichen nur die Dimensionierung verschiedener Felder und die Vorbereitung der Grafikelemente. Nach dem Spielstart ab Zeile 450 wird zunächst der Zugzähler zz und der Remiszähler rz (er überwacht die Einhaltung der Remisregel!) mit 0 initialisiert und das Spielbrett b(x,y) mit der Anfangsverteilung belegt.

GOSUB 1250 sorgt für die Bildschirmausgabe, und die Zuweisung spieler=1 bestimmt, daß der Rechner immer beginnt. Falls Sie anfangen wollen, muß hier spieler=2 stehen. Die sich aus dieser Spielversion ergebenden Dateien sollten allerdings separat geführt werden, da sonst auf lange Sicht ein *Memory full* droht. Der eigentliche Spielablauf wird in den Zeilen 600-670 kontrolliert, indem abwechselnd die Unterprogramme *Zug Computer* und *Zug Gegner* aufgerufen werden. Nach Abschluß der Partie enthält die Variable erg das Spielergebnis; in Abhängigkeit von diesem Wert schließt sich eine Auswertung an, zu der wir noch kommen. Zunächst werden wir etwas tiefer in die Datenstrukturen des Programms eindringen.

Gepackte Daten: Dicht an dicht

Eines ist klar: Wer etwas lernen will, braucht ein gutes Gedächtnis, und wie man weiß, hat der Mensch deren sogar zwei: Ein Kurzzeit- und ein Langzeitgedächtnis. Diese Trennung ermöglicht es, selektiv zu lernen, d.h. nur wirklich wichtige Daten werden nach einer Auswertung langfristig bereitgehalten. Gerade bei einem Microcomputer mit begrenztem Speicherplatz ist es außerordentlich ratsam, dieses Verfahren zu kopieren; doch zunächst

müssen wir klären, welche Daten überhaupt für eine Speicherung in Frage kommen.

Als erstes kommen z.B. die verschiedenen Spielsituationen in Betracht, die im Laufe der Partien auftreten. Schließlich soll sich das Programm ja merken, in welchen Positionen es durch welche Züge gewonnen bzw. verloren hat. Damit deuten sich aber auch schon die ersten Speicherplatzsorgen an: Ein 4x4-Integerarray zur Darstellung des Brettes benötigt insgesamt 16x2=32 Bytes, das macht bei 1000 verschiedenen Positionen fast 32 KByte aus, und die kommen bei Minischach locker zusammen! Wie man sieht, empfiehlt es sich, bei speicherintensiven Problemen solche Überlegungen vor der Programmerstellung durchzuführen, um vor unangenehmen Überraschungen sicher zu sein.

Was also tun? Eine hilfreiche Technik ist das sogenannte Packen der Daten, bei der eine Kodierung gesucht wird, die möglichst wenig Speicherplatz erfordert. Intuitiv ist klar, daß eine Integervariable, die Zahlenwerte von −32768 bis 32767 darstellen kann, mit einem Bereich von 0 bis 3 glatt unterfordert ist: Hier stehen insgesamt 16 Bit (mit Vorzeichen) zur Verfügung, die noch wesentlich mehr Informationen aufnehmen können. Da unser Spielfeld genau 16 Felder umfaßt, bietet es sich an, zunächst einmal bitweise zu verschlüsseln, welche von ihnen belegt (Bit gesetzt) oder leer sind (Bit zurückgesetzt). Die Startposition sieht dann zeilenweise gelesen so aus:

```
psn1 = 1111 0000 0000 1111
```

Doch das allein reicht natürlich nicht. Zusätzlich muß notiert werden, um welche Figuren es sich handelt. Auf dem Spielbrett sind den verschiedenen Figurenarten die Ziffern von 1 bis 4 zugeordnet; subtrahiert man hier den Wert 1, so ergibt sich ein Bereich von 0 bis 3, für dessen Darstellung man nur zwei Bit braucht:

Figur	dez.	bin.
1 (weißer Bauer)	0	00
2 (weiße Dame)	1	01
3 (schwarzer Bauer)	2	10
4 (schwarze Dame)	3	11

Damit können wir die Figuren (ohne Berücksichtigung der Leerfelder) einfach in der Reihenfolge aufzählen, in der sie von links oben nach rechts unten auftreten. Da sich mit Sicherheit nie mehr als acht Steine auf dem Brett tummeln, kommen wir auch hier mit 16 Bit aus, die für die Startposition so aussehen:

```
psn2 = 00 00 00 00 10 10 10 10
```

Wie man sich anhand weiterer Beispiele vor Augen führen kann, ist die gesamte Spielsituation durch diese beiden 16-Bit-Werte eindeutig erfaßt. Obwohl wir für die Darstellung jetzt nur noch 4 anstatt 32 Bytes benötigen (also 2 Integervariablen), gehen keine Informationen verloren! Der einzige Nachteil liegt mitunter darin, daß bei einer Neubearbeitung gepackter Daten diese erst wieder dekodiert werden müssen, was natürlich etwas Zeit kostet.

Die praktische Realisierung des Packens zeigen die Programmzeilen 1400 – 1520 im Rahmen des Unterprogramms *Zug Computer*. Damit es mit dem Vorzeichenbit keinen Arger gibt, werden die gepackten Daten zunächst in Realvariablen (p1! und p2!) aufgenommen. Anschließend sorgt dann die UNT-Funktion für eine Umwandlung in vorzeichenbehaftete Integerwerte (psn1 und psn2). Wie Sie dem Listing entnehmen können, wird die ganze Prozedur danach noch einmal wiederholt, und zwar mit rückwärts laufendem Spaltenindex, wodurch die kodierte Form des gespiegelten Bretts entsteht (psx1 und psx2).

Im weiteren Verlauf tritt ein kleines Maschinenprogramm in Aktion, dem neben den eben besprochenen Variablen noch die Startadressen zweier Arrays übergeben werden, die einen wichtigen Bestandteil des Langzeitgedächtnisses bilden: In l.psn1(i) und l.psn2(i) (das kleine l steht für Langzeit) merkt sich das Programm alle entscheidenden Situationen vorangegangener Partien in kodierter Form. Die Assemblerroutine schaut praktisch nur nach, ob die aktuelle Brettposition dort schon verzeichnet ist. Ist das der Fall, so wird in Zeile 1570 die Variable flag auf –1 gesetzt; i enthält dann die Nummer, unter der die Position vermerkt ist. Xflag wird gesetzt, falls die spiegelbildliche Position gefunden wurde.

Damit erfüllt unser Programm bereits eine wesentliche Voraussetzung für Lernfähigkeit: Es kann sich erinnern. Doch die Erkenntnis, daß eine Spielsituation schon einmal da war, hilft bei der Zugauswahl natürlich nicht weiter. Es fehlt ein Hinweis darauf, welche Züge sich an dieser Stelle bereits als günstig oder ungünstig erwiesen haben.

Der Zuggenerator

Viele Programm für strategische Brettspiele enthalten einen sogenannten Zuggenerator. Dieser Programmteil hat die Aufgabe, eine Liste aller legalen Züge anzufertigen, aus der das Hauptprogramm dann seine Auswahl trifft. Weiterhin können Zuglisten für die Überprüfung von Eingaben benutzt werden: Ist ein Zug nicht in der Liste enthalten, so ist er illegal und wird zurückgewiesen.

Auch unser Minischach-Programm enthält einen solchen Regelexperten, der gleich noch genauer zur Sprache kommt. Zunächst stellt sich aber wieder die Frage nach einer günstigen Datenstruktur für die Darstellung der Zuglisten. Wenn Sie noch einmal einen Blick auf die Darstellung des Schachbrettes werfen, können Sie sich sofort klarmachen, daß ein Zug vollständig durch vier Zahlen von 1 bis 4 charakterisiert ist: Von (Spalte,Reihe) nach (Spalte,Reihe). In dem Programm sind übrigens grundsätzlich die Startkoordinaten in den Variablen sp,re und die Zielkoordinaten in x,y enthalten.

Wer die vorangegangenen Ausführungen über das Packen von Daten aufmerksam verfolgt hat, kann sich hier leicht ausrechnen, daß zur Speicherung dieser vier Zahlen 4x2=8 Bits (d.h. 1 Byte) vollkommen ausreichen; wir erhalten also pro gepacktem Zug einen Wert von 0 bis 255. Und jetzt ein besonderer Trick: Dieser Wert kann ohne weiteres mit Hilfe der CHR$-Funktion in ein ASCII-Zeichen verwandelt werden, so daß wir die Zugliste als String darstellen können! Dem Unterprogramm ab Zeile 2570 können Sie entnehmen, wie dieser Vorgang in der Praxis aussieht.

Die Verarbeitung der Zugliste als String bietet einige handfeste Vorteile. So weist Z.B. ein Leerstring sofort darauf hin, daß ein Spieler verloren hat, da er keinen Zug mehr ausführen kann. Weiterhin läßt sich mit Hilfe der INSTR-Funktion außerordentlich bequem feststellen, ob ein Zug in der Liste enthalten und damit legal ist. Auch Manipulationen durch MID$ und ähnliche Funktionen sind möglich, um z.B. als schlecht erkannte Züge aus einer Liste zu streichen.

Den einzigen Nachteil dieser Methode werden besonders CPC 464-Besitzer bei vollem Speicher mitunter erleben, wenn der Rechner eine Garbage Collection durchführt, um seine Strings mitsamt unseren Zuglisten aufzuräumen: Er stellt zu diesem Zweck alle laufenden Aktivitäten ein und ist während der Zeit auch nicht weiter ansprechbar. Die Stringverwal-

tung des 664/6128 wurde in dieser Hinsicht verbessert, braucht dafür aber auch zwei zusätzliche Bytes Speicherplatz pro Stringvariable.

Doch jetzt, wie versprochen, ein genauerer Blick auf den Zuggenerator ab Zeile 2060. Zu Beginn wird hier eine Variable seite definiert, die das Erstellen der Zuglisten sowohl für die weißen als auch für die schwarzen Steine ermöglicht, in Abhängigkeit davon, wer gerade am Zug ist. Danach wird das Spielfeld auf Damen hin abgetastet, und, soweit welche vorhanden sind, ein Unterprogramm aufgerufen, das eventuell mögliche Schlagzüge in die Zugliste schreibt – sie sind ja verpflichtend! Deshalb geht es auch gleich zurück (RETURN), falls die Zugliste nach dieser Prozedur nicht mehr leer ist. Anderenfalls wird das Spielfeld noch einmal durchgecheckt, um Bauernzüge und nicht-schlagende Damenzüge zu registrieren.

Die drei Unterprogramme ab Zeile 2220 erkunden die verschiedenen Bewegungsmöglichkeiten der Figuren. Für die Ermittlung der Damenzüge sind insbesondere zwei bereits im Initialisierungsteil erzeugte Arrays dx(ri) und dy(ri) nützlich; sie enthalten für alle acht Bewegungsrichtungen einen sogenannten Richtungsoffset, der fortlaufend zu den Koordinaten der Startposition addiert wird, bis ein Hindernis (z.B. der mit –1 belegte Rand) der Zugstraße eine Ende setzt. Wer Minischach mit veränderten Spielregeln probieren möchte, kann hier folgende Änderungen vornehmen:

```
FOR ri = 0 TO 3: 'verwandelt die Damen in Türme
```

oder

```
FOR ri = 4 TO 7: 'verwandelt die Damen in Läufer.
```

Reduzierte Zuglisten

Kehren wir jetzt zu dem Modul *Zug Computer* zurück. Was passiert hier ab Zeile 1590, nachdem unser Programm mit Hilfe des Maschinenprogramms versucht hat, sich zu erinnern? Falls die aktuelle Position noch nicht im Langzeitgedächtnis enthalten ist, so wird der eben besprochene Zuggenerator aufgerufen, um die dazugehörige Zugliste zu ermitteln. Andernfalls wird eine bereits fertige Zugliste l.zliste$(i) benutzt, die ebenfalls Bestandteil des Langzeitgedächtnisses ist und – wie Sie sich wahrscheinlich schon denken können – um alle Züge reduziert wurde, die sich bei vorherigen Partien in

dieser speziellen Situation als verlustträchtig erwiesen haben. Sie enthält also nur gute Züge bzw. Züge, über die noch nichts bekannt ist. Sollte in dem einen oder anderen Fall eine leere Liste in Erscheinung treten, so weiß das Programm, daß es verloren hat (erg=2), und es geht zurück ins Hauptprogramm.

Ansonsten wird das Spiel jedoch mit einigen Eintragungen ins Kurzzeitgedächtnis fortgesetzt, das dazu dient, den Ablauf einer Partie in allen Einzelheiten zu protokollieren. Die Variablenfelder beginnen alle mit k, benutzen den Zugzähler zz als Index und haben im einzelnen folgende Bedeutung:

k.psn1(zz), k.psn2(zz): Die Brettposition vor Ausführung eines Zuges in gepackter Form.

k.zliste$(zz): Die zur Auswahl des Zuges benutzte Zugliste.

k.index(zz): Diese Variable erhält den Wert −1, falls die aktuelle Position noch nicht im Langzeitgedächtnis vermerkt ist. Anderenfalls gibt sie an, unter welchem Index die Eintragung gefunden wurde.

k.znr(zz): Gibt die Position des gewählten Zuges in der Zugliste an; sie wird in Zeile 1670 nach einem Zufallsprinzip ermittelt. Falls Sie möchten, daß das Programm systematisch lernt, so können Sie die Zeile einfach in k.znr(zz)=1 abändern; das Programm führt dann stur immer den ersten in der Liste vorhandenen Zug aus.

Was nach der Eintragung ins Kurzzeitgedächtnis passiert, ist schnell geklärt: Der ausgewürfelte Zug wird mit Hilfe von MID$ der Zugliste entnommen, in einen Zahlenwert verwandelt und anschließend bitweise in vier Koordinaten zerlegt, mit denen das Unterprogramm *Zug ausführen* den Rest erledigt. Danach wird nur noch der Zugzähler zz erhöht und es geht zurück ins Hauptprogramm: Der Gegner ist an der Reihe.

Und damit kommen wir zu der Auswertung nach Ende einer Partie. Schauen wir uns zunächst an, was passiert, wenn der Rechner die Partie gewonnen hat (ab Zeile 810). Es ist sofort klar, daß die Zugliste, die zu der Position vor dem Gewinnzug gehört, genau auf diesen einen Zug reduziert werden kann; einen besseren wird das Programm auf keinen Fall finden. Die einzige Frage ist nur noch, ob die Position schon im Langzeitgedächtnis vermerkt ist oder nicht. Im ersten Fall wird die Manipulation der Zugliste direkt dort vorgenommen, im zweiten Fall muß zunächst eine neue Eintragung vorgenommen und die Variable mp (Memory Pointer), die immer auf den nächsten freien Platz in Langzeitgedächtnis zeigt, um 1 erhöht werden. Es gelangen also nur Positionen vom Kurz- ins Langzeitgedächtnis, bei denen ein konkre-

ter Anlaß zur Reduzierung der Zugliste vorliegt. Für alle anderen Positionen können die vollständigen Listen ja jederzeit vom Zuggenerator erzeugt werden.

Der Memory-Pointer, d.h. die Anzahl der bereits gespeicherten Positionen und Zuglisten, wird übrigens laufend in der linken oberen Bildschirmecke angezeigt. Sollte er sich bedrohlich dem Wert 2000 nähern, so müssen Sie vorsorglich die Dimensionierung des Langzeitgedächtnisses in Zeile 150 erhöhen.

Etwas komplizierter ist die Auswertung, falls der Rechner eine Partie verliert (ab Zeile 870). Klar ist sofort, daß der letzte Zug aus der Zugliste eliminiert werden muß, doch dabei kann ein Spezialfall auftreten. Falls dieser Zug der einzige in der Liste ist, d.h. keine Alternativzüge mehr übrig bleiben, so heißt das automatisch, daß auch der vorherige Zug des Rechners als Verlustzug zu betrachten ist – er führt ja, falls der Gegner es will, zu einer Position mit einer leeren Zugliste! Ist dann der vorherige Zug ebenfalls ohne Alternative, so muß die Auswertung auf den vorvorigen Zug ausgedehnt werden, usw. Auf diese Weise kann es passieren, daß durch einen einzigen Partieverlust ganze Zugfolgen bis hin zum Partieanfang eliminiert werden.

Zum Abschluß bleibt noch, welche Auswertung bei einem Remis in Frage kommt. Entscheidend ist dabei, ob das Programm lernen soll, zu gewinnen, oder nicht zu verlieren – im ersten Fall muß das gleiche Verfahren wie bei einer Verlustpartie angewendet, im zweiten Fall dagegen die Auswertung der gewonnenen Spiele übernommen werden. Im Programm wurde ein weiser Kompromiß geschlossen: Es findet überhaupt keine Auswertung statt! Wenn Sie wollen, können Sie aber durch ein entsprechendes GOTO in Zeile 940 die Remis-Routine über die eine oder andere Auswertung umleiten.

Der programmierte kleine Bruder

Damit haben Sie nun alle für das Lernverhalten des Rechners wichtigen Programmteile kennengelernt, und wir werden uns jetzt der Frage zuwenden, warum der Lernvorgang vergleichsweise langsam stattfindet, obwohl sich in der Datenstruktur durchaus Parallelen zum menschlichen Gedächtnis finden lassen. Doch mit dem Speichern von Positionen und Eliminieren von Verlustzügen ist es offensichtlich nicht getan. Insbesondere besitzt der Mensch zusätzlich die Fähigkeit, aus seinen Erfahrungen allgemeine Regeln

abzuleiten, wie z.B.

Es ist schlecht, wenn der Gegner eine Dame erhält,

und kann die Regeln auch wieder auf konkrete Spielsituationen anwenden. Es liegt auf der Hand, daß diese Fähigkeit einen Lernvorgang außerordentlich beschleunigt, da die bereits gewonnenen Erfahrungen auch auf Situationen angewendet werden können, die bislang noch nicht aufgetreten sind.

Damit berühren wir ein Gebiet, mit dem Elektronengehirne ihre Schwierigkeiten haben. Regeln wie das obige Beispiel werden im allgemeinen durch Bewertungsfunktionen erfaßt (siehe Kapitel 4). Daraus resultiert praktisch die Forderung nach einem Programm, das im Verlauf des Lernvorgangs die Bewertungsfunktion oder zumindest die Gewichtsfaktoren einer vorgegebenen Formel selbst verändert. Doch bislang gibt es in dieser Hinsicht kaum vorzeigbare Ergebnisse, und die wenigen Experimentalprogramme kranken meist am gleichen Problem wie unser Beispiel: Sie lernen zu langsam. Zwar beherrscht unser Programm immerhin eine Art der Verallgemeinerung, nämlich die Anwendung seines Wissens auf spiegelbildliche Situationen, doch das ist sehr wenig.

Auf welch absurde Weise sich der Mangel an Abstraktionsfähigkeit bemerkbar machen kann, soll die folgende kurze Szene verdeutlichen: In tagelanger, aufopferungsvoller Lehrtätigkeit haben Sie das Minischach-Programm soweit gebracht, daß es allen Ihren Gemeinheiten und taktischen Schlichen gewachsen ist. Auch bei stärkstem Gegenspiel gelingt es kaum noch, eine Partie gegen den Rechner zu gewinnen. Stolz auf die Erfolge Ihres Schülers wollen Sie das Programm beenden und den gesamten Lerninhalt auf Datenträger schreiben lassen, da stürmt Ihr kleiner Bruder in den Raum. Spaßeshalber erklären Sie ihm die Regeln und laden ihn zu einer Partie ein. Und siehe da: Obwohl der Kleine oft mit den denkbar dümmsten Zügen operiert, verliert das ach so gelehrige Programm plötzlich wieder eine Partie nach der anderen – was ist passiert?

Ganz klar: Sie haben dem Programm zwar beigebracht, wie man gegen einen guten, aber nicht, wie man gegen einen schlechten Spieler gewinnt! Da das Programm nicht verallgemeinert, nützen ihm die Erfahrungen aus den vorherigen Partien überhaupt nichts, wenn es auf vollkommen unbekannte Stellungen stößt. Also legen Sie dem Knirps nahe, weitere Partien zu spielen, um das Programm auch in dieser Beziehung sturmfest zu machen, doch

leider hat der nach 10 Minuten die Nase voll und zieht von dannen. Was nun?

Doch bevor Sie in Versuchung kommen, selber den Dummi zu spielen, hier eine bessere Lösung: Warum soll der Computer nicht diese Rolle übernehmen? Das Ganze stellt überhaupt kein Problem dar. Sie müssen nur die Erweiterung *kleiner Bruder* mit **MERGE „BRUDER"** dazuladen, und danach lernt der Computer, indem er gegen sich selbst spielt, eine Partie nach der anderen! Nach Drücken von DEL wird das Gemetzel bei nächster Gelegenheit abgebrochen, und das Programm schreibt wie gewohnt sein Langzeitgedächtnis auf Datenträger.

Auf den ersten Blick scheint hier wirklich ein alter KI-Traum (oder ein Trauma?) in Erfüllung zu gehen: Ohne menschliches Zutun erarbeitet sich die Maschine eine Qualifikation, die sie irgendwann in die Lage versetzen wird, sich in der Realität (d.h. gegen menschliche Gegner) zu bewähren. Schaut man sich jedoch an, was auf dem Bildschirm abläuft, so stehen einem die Haare zu Berge: Da ja beide Kontrahenten ihre Züge zufallsgesteuert wählen, kommen kaum sinnvolle Zugfolgen zustande. Bei dem Führer der schwarzen Steine bleibt das auch so, während der Führer der weißen Steine ständig dazulernt; er müßte also nach einer ausreichenden Anzahl von Partien die Mehrzahl der Spiele gewinnen. Lassen Sie Ihrem CPC ruhig zwei oder drei Nächte Zeit, und probieren Sie, was bei diesem Langzeitexperiment herauskommt.

Das Programm des Autors operiert inzwischen mit einer Datei, die auf Disk ca. 30 KByte belegt und 1950 gespeicherte Positionen umfaßt. Damit gewinnt es in der Tat die meisten Partien gegen den kleinen Bruder, und auch erfahrene Gegner werden mitunter schon deutlich abserviert. Entstanden ist die Datei in einigen harten Lehrstunden und durchrechneten Nächten; die Kombination beider Methoden hat sich als recht sinnvoll erwiesen.

Trotzdem sollte man nicht verschweigen, daß das Programm in der vorliegenden Form von seiner Leistungsfähigkeit her kaum befriedigt. Die Erwartungen, die bei der Planung in das Projekt gesetzt wurden, erfüllt es nur teilweise. Aber trotzdem gab es gute Gründe, es im Software-Experiment vorzustellen. Einerseits demonstriert es interessante Programmiertechniken (z.B. gepackte Daten, Zuggenerator), die auch bei anderen Gelegenheiten nützlich werden können; andererseits ist es recht aufschlußreich, dem Programm bei seinen Lernbemühungen zuzuschauen. Und vielleicht hat ja der

eine oder andere Leser noch eine Idee, wie man das Programms verbessern kann?!

KAPITEL 7

Der Computer als Wissenschaftler

Hier als Einstimmung zunächst ein Zitat aus der Frankfurter Rundschau vom 3.11.1986:

„Während das Verhältnis des Vatikans zur Naturwissenschaft geklärt zu sein scheint – immerhin ist Galileo Galilei von Johannes Paul II. rehabilitiert worden, 350 Jahre nach dem Prozeß – ist im Verhältnis zu den Humanwissenschaften vielfach noch eine galileische Situation gegeben. Davon zeugt die jüngste Stellungnahme des Vatikans zur Homosexualität bei Männern und Frauen. ‚Einzig in der Ehe kann der Gebrauch der Sexualkraft moralisch gut sein. Deshalb handelt eine Person, die sich homosexuell verhält, unmoralisch', lauten die beiden Kernsätze dieses Dokuments."

Die Entwicklung der modernen naturwissenschaftlichen Denkweise begann im 17. Jahrhundert, und als ihr geistiger Vater wird im allgemeinen Galileo Galilei angesehen. Wie man weiß, provozierte er damals mit seiner Behauptung, die Sonne und nicht etwa die Erde wäre der Mittelpunkt unseres Planetensystems, einigen Ärger. Die Vertreter der herrschenden Lehrmeinung verfügten jedenfalls über ausreichende Druckmittel, um ihn zum Widerruf zu veranlassen. Galileis berühmter Ausspruch auf dem Totenbett ‚...und sie bewegt sich doch' ist vielleicht nachträglich erfunden worden, steht aber symbolisch für die Tatsache, daß die zu seiner Zeit revolutionären Erkenntnisse trotz alledem nicht mehr aufzuhalten waren.

Die Legende berichtet, daß Galilei seine Widersacher aufforderte, durch das von ihm entwickelte Fernrohr zu schauen, um sich mit eigenen Augen von den Gesetzmäßigkeiten der Planetenbewegung zu überzeugen. Doch die Pharisäer und Schriftgelehrten lehnten ab: ‚Warum sollen wir durch dein Rohr blicken, wo wir doch genau wissen, daß nicht sein kann, was du sagst...' Als Kinder des Weltraum-Zeitalters blicken wir heute verständnislos auf die-

se Mischung von Dummheit und Arroganz und fragen uns, wie so etwas möglich sein konnte – doch halt! Sind Sie, lieber Leser, bereit, für einen Moment eine selbstkritische Haltung einzunehmen? Dann denken Sie einmal über die folgenden Fragen nach: Woher wissen Sie eigentlich, daß sich die Erde um die Sonne dreht? Haben Sie es in der Schule gelernt, in einem Buch gelesen oder im Fernsehen gehört? Mal ganz ehrlich: Hätte man Ihnen auf diese Weise nicht auch die umgekehrte Meinung eintrichtern und Sie zu einem überzeugten Vertreter eines geozentrischen Weltbildes erziehen können, ohne daß Sie es jemals in Frage gestellt hätten?

Man muß sich in diesem Zusammenhang vor Augen halten, daß es zu Galileis Zeiten gang und gäbe war, alle wissenschaftlichen Erkenntnisse aus der Bibel oder den Schriften des klassischen Altertums abzuleiten. Galileis Faux Pas bestand weniger darin, den herrschenden Lehrmeinungen zu widersprechen; von Bedeutung war vielmehr, daß er die damals gültige wissenschaftliche Methodik von Grund auf ablehnte. Als Gegenpol zur Deduktion, der Anwendung vorgegebener Gesetzmäßigkeiten auf Einzelfälle (Naturphänomene), führte er das wissenschaftliche Experiment als Frage an die Natur ein, ob denn die in den Theorien enthaltenen Behauptungen überhaupt stimmen. Die Theologen seines Zeitalters sahen darin einen Versuch, die Theologie den Naturwissenschaften unterzuordnen und schickten Galilei die Inquisition auf den Hals, da sie verständlicherweise keine Lust hatten, ihre dominierende Position aufzugeben.

Die wissenschaftliche Methode im Sinne Galileis beinhaltet also eine ausgesprochen kritische Denkweise, die eine hohe geistige Beweglichkeit erfordert. Für sie gibt es keine gottgegebenen Gesetze, die als heilige Kühe im Marmortempel der Wissenschaft vor sich hin muhen. Jede Behauptung (ob aus der Bibel oder dem Fernsehen) stellt nur einen Anlaß dar, weitere Fragen zu stellen, die natürlich dazu führen können, daß der ursprüngliche Denkansatz total über den Haufen geworfen wird. Um diesen Vorgang noch einmal zusammenzufassen: Aus der Beobachtung einzelner Naturphänomene wird auf bestimmte Gesetzmäßigkeiten geschlossen (Induktion). Aus diesen Gesetzen läßt sich wiederum ableiten, wie sich die Natur in anderen Fällen verhalten wird (Deduktion). Experimentell wird geklärt, ob diese Voraussagen mit der Realität übereinstimmen. Ist das nicht der Fall, so muß die Theorie entsprechend modifiziert und als Ausgangspunkt für weitere Versuche benutzt werden. Aus diesem ständigen Wechselspiel von Induktion und Deduktion entsteht wissenschaftliche Erkenntnis, Theorie und Praxis arbeiten

gleichberechtigt Hand in Hand.

Erkenntnis statt Programmierung

Galilei gewann seine Erkenntnisse durch eine ständige Auseinandersetzung mit der Realität; er lehnte es ab, die Realität mit Gewalt so zu interpretieren, daß sie zu einem vorprogrammierten Wissen kompatibel wurde. Seine Methode stellt praktisch eine systematisierte Form des Lernens dar, und hier wird der KI-Programmierer hellhörig: Methode? System? Wo diese Begriffe auftauchen, ist häufig ein Algorithmus nicht mehr fern, ein in Einzelschritte auflösbares Verfahren, das auch von einer Maschine ausgeführt werden kann.

Und damit sind wir wieder bei der Frage angelangt, die uns bereits im letzten Kapitel beschäftigt hat: Wie kann ein Computer zu einem selbständigen Lernprozeß angeleitet werden? Immerhin hatten wir mit dem Minischach-Programm Teilerfolge vorzuweisen, die aber insgesamt nicht besonders befriedigten. Die Lernerfolge wurden praktisch nur durch eine gut organisierte Datenspeicherung erreicht, und das im Schneckentempo, da unser Programm aus seinen Erfahrungen keine allgemeingültigen Gesetzmäßigkeitenableiten konnte.

Bevor wir uns jedoch dem Problem zuwenden, einem Computer zumindest einen Schimmer echter Erkenntnisfähigkeit zu vermitteln, soll der Begriff ‚Lernen' an dieser Stelle genauer betrachtet werden. In gewisser Weise kann ja schon die Programmierung eines Computers, also das Abspeichern einer Folge von Anweisungen, als Lernvorgang aufgefaßt werden. Doch diese Art des Lernens, die praktisch nur eine ausgeprägte Merkfähigkeit verlangt, ist für die künstliche Intelligenz nicht besonders interessant. Gefragt sind vielmehr Systeme, die aufgrund ihrer Erfahrungen von sich aus ihr Verhalten ändern, und das erfordert nicht nur ausreichend Speicherplatz, sondern auch eine hochentwickelte Auffassungsgabe, um die Folgen verschiedener Entscheidungen wahrzunehmen und zu verarbeiten.

Auch beim Menschen wird dieser Unterschied deutlich. Mit Sicherheit erfordert es mehr Intelligenz, eine Sprache zu lernen, als Vokabeln zu lernen: Im ersten Fall müssen komplexe Sinnzusammenhänge erfaßt werden, im zweiten Fall reicht ein gutes Gedächtnis. Es ist immer wieder faszinierend, mitzuerleben, mit welcher naturgegebenen Intelligenz sich ein kleines

Kind die Sprache aneignet; erschreckend ist andererseits die Erkenntnis, daß sich Menschen in vieler Hinsicht wie Computer programmieren lassen.

Ein gutes Beispiel dafür bietet die Werbung, die durch vielfache Wiederholung und unterschwellige Beeinflussung unser Konsumverhalten steuern soll. Daß diese Art der Programmierung offenbar gut funktioniert, läßt sich allein daraus ablesen, daß die Wirtschaft Jahr für Jahr unverdrossen Milliarden investiert, um z.B. brave Hausfrauen davon zu überzeugen, daß sie ohne RENOL ein schlechtes Gewissen ihrer Familie gegenüber haben müssen.

Erinnern wir uns noch einmal an die unbequemen Fragen zu Beginn dieses Kapitels. Mit Sicherheit haben Sie bereits vieles in Ihrem Leben gelernt. Doch wie sieht es aus: Beruht das Gelernte auf Erkenntnissen, die durch Erfahrung und gezielte Fragen gewonnen wurden, oder auf einer durch Erziehung oder Medien vermittelten Programmierung? Wie auch immer das Ergebnis Ihrer Untersuchung ausfallen mag – diese Gegenüberstellung vermittelt ein Bild davon, worauf es bei einer ‚künstlich intelligenten' Lernfähigkeit ankommt.

Modell gesucht

Nach wie vor ist die Realität mit allen ihren Aspekten zu komplex, um als Objektraum für die Programmierung künstlicher Intelligenz in Frage zu kommen. Alle Versuche beziehen sich deshalb auf einen genau definierten (und meist sehr kleinen) Ausschnitt der Realität oder auf ein stark vereinfachtes Modell realer Vorgänge. Wie bereits in den letzten Kapiteln deutlich wurde, eignen sich Spiele zu diesem Zweck ganz ausgezeichnet. Sie beschränken sich einerseits auf eine überschaubare Menge von Objekten (Spielfiguren) und Operationen (mögliche Spielzüge), enthalten aber auf der anderen Seite eine ausreichende Anzahl Unsicherheitsfaktoren (Züge des Gegners), auf die das Programm mit einer gewissen Selbständigkeit reagieren muß.

Auch die Anwendung wissenschaftlicher Methoden läßt sich sehr schön anhand eines Spiels demonstrieren, das vor einigen Jahren unter dem Titel ‚Superhirn' oder ‚Mastermind' in den Handel kam und sehr populär wurde. Im Prinzip geht es darum, einen geheimen Code zu knacken, der aus einer verdeckten Reihe farbiger Steckknöpfe besteht. Für jeden Code, den man ausprobiert, erhält man vom Gegenspieler eine Trefferquote, die über die Anzahl der richtigen Farben und richtigen Positionen Auskunft gibt. Durch

logische Schlußfolgerungen und eine effektive Versuchsanordnung soll man dem Code möglichst schnell auf die Schliche kommen.

Wir werden uns in dieser Folge mit einer besonders reizvollen Variante dieser Spielidee beschäftigen, bei der mit dem gleichen Verfahren ein geheimes Wort gesucht wird – also das ideale Spiel für Hacker! Als Code oder Versuche kommen natürlich nur im Sprachgebrauch übliche Wörter in Frage, und man muß sich vor Spielbeginn auf eine bestimmte Wortlänge einigen. Hier gleich ein Beispiel: Gesucht ist das Wort HAUS; das Experiment DACH erhält dann als Trefferquote zwei richtige Buchstaben (A und H) und eine richtige Position (A steht an der richtigen Stelle). Im Folgenden werden wir die Ergebnisse kurzerhand so notieren,

```
DACH # +
```

mit zwei Markierungen für zwei richtige Buchstaben, wobei das Doppelkreuz für die richtige Position steht. Zu beachten ist, daß doppelt vorkommende Buchstaben nur dann zwei Treffer ergeben, wenn sie auch im Geheimwort doppelt vorhanden sind. Der geheime Code HEXE kann z.B. folgende Versuchsergebnisse erzeugen:

```
HUHN #    ESEL + +
```

Um sich dem geheimen Wort zu nähern, ist eine Auswertung der Trefferquoten unumgänglich, wobei man meist eine Reihe von Fällen durch logische Schlüsse ausklammern kann. Teilweise ist das sehr einfach: Wenn ein Wort keinen einzigen Treffer erhält, so kommen die Buchstaben in dem geheimen Code nicht vor. Etwas schwieriger wird es jedoch, wenn die ersten beiden Versuche das folgende Ergebnis bringen:

```
EHRE #    WORT +
```

Hieraus geht eindeutig hervor, daß das geheime Wort kein R enthält – können Sie herausfinden, warum? Noch wesentlich komplizierter gestalten sich Schlußfolgerungen, die aus der gemeinsamen Betrachtung von drei oder mehr Versuchen entstehen, und oft ist man sogar auf Vermutungen angewiesen. Schauen Sie sich die folgenden Ergebnisse an:

```
BERG # #    WERT # #
```

Es drängt sich der Verdacht auf, daß die Kombination ER in der Mitte für die beiden richtigen Positionen verantwortlich ist; wie sollte es anders sein? Also hauen wir mit dem dritten Versuch in die gleiche Kerbe,

```
HERZ #
```

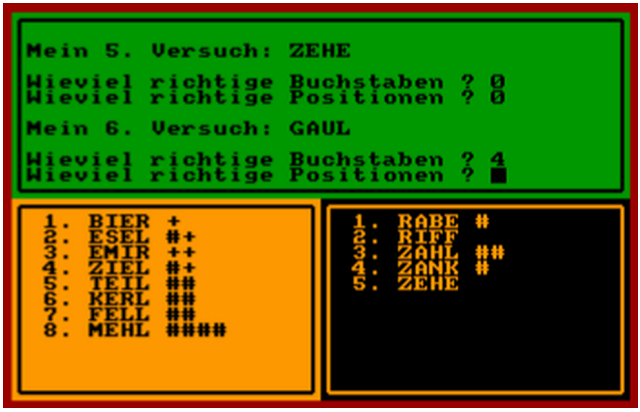
und erhalten nur eine richtige Position! Hier kann man ansatzweise nachempfinden, wie einem Wissenschaftler zumute ist, wenn durch das Ergebnis eines Experiments eine schon sicher geglaubte Theorie den Bach runter geht. Das geheime Wort in diesem Fall war übrigens BART...

Um sich ein vollständiges Bild von dem Charakter dieses Spiels zu verschaffen, suchen Sie sich am besten einen Partner und spielen ein paar Fälle durch. Dabei wird schnell klar, daß ein blindes Herumprobieren überhaupt nichts bringt. Eine gute Methode besteht jedoch darin, aufgrund der vorherigen Versuchsergebnisse eine Theorie über die Zusammensetzung des geheimen Wortes aufzustellen, wobei man mitunter bewußt in Kauf nimmt, daß sie auch falsch sein könnte. Der nächste Versuch wird dann wie in unserem letzten Beispiel dieser Arbeitshypothese entsprechend

formuliert; die Trefferquote gibt Auskunft darüber, ob die Annahmen berechtigt waren. Mit dieser naturwissenschaftlichen Strategie hat man eine klare Linie vor Augen, die relativ sicher zum Ziel führt. Alles, was uns jetzt noch fehlt, ist ein intelligentes Programm, das den geheimen Code ebenfalls knacken kann.

Der große Bluff: Wordmaster

Zunächst taucht jedoch die Frage auf, ob wir uns nicht ein bißchen viel vorgenommen haben. Offensichtlich muß das Programm ja hochkomplizierte logische Schlüsse bewältigen können, Annahmen aufstellen und wieder verwerfen... brauchen wir hier nicht mindestens einen Großrechner und spezialisierte Programmiersprachen wie LISP oder PROLOG? Doch weit gefehlt – WORDMASTER demonstriert eindrucksvoll, daß auch ein Microcomputer in reinem BASIC dieses Problem lösen kann!



Nach dem Programmstart mit **RUN „WMASTER“** erwartet Sie das Hauptmenue, das unter anderem den Punkt *Spielen* anbietet. Wer hier sofort zugreift, wird jedoch eine Enttäuschung erleben: WORDMASTER verlangt zunächst eine Wortliste; ohne einen gewissen Grundwortschatz ist das Programm nicht aktionsfähig. Auf der Programmdiskette bzw. -kassette stehen Ihnen zwei fertige Wortlisten zur Verfügung **(LISTE4.DAT, LISTE5.DAT)**, die Sie mit dem Menuepunkt 3 laden können. Eine enthält vier- und die andere fünfbuchstabige Wörter. Aufgenommen wurden nur im deutschen Sprachgebrauch übliche Hauptwörter ohne Umlaute und ß, weiterhin keine Eigennamen oder geographische Namen. Diese Einschränkungen haben sich in der Praxis als sinnvoll erwiesen; wie Sie sehen werden, ist das Spiel auch damit schon schwierig genug!

Der Menuepunkt 2 dient dazu, eine Wortliste einzugeben, zu verändern oder anzuzeigen; auf dem Bildschirm erhalten Sie alle notwendigen Hinweise. Zu beachten ist, daß bei der Eingabe einer neuen Liste das erste Wort die Länge aller weiteren Wörter bestimmt, abweichende Eingaben werden im Folgenden zurückgewiesen. WORDMASTER sortiert automatisch neue Wörter alphabetisch ein, so daß sich bei umfangreichen Listen nach jeder Eingabe eine kleine Verzögerung ergibt. Maximal kann eine Liste 1000 Wörter umfassen. Wer diese Grenze überschreiten will, muß die Dimensionierung des Arrays w$() in Zeile 100 und 980 erhöhen. Die Wortlänge sollte nicht mehr als sechs Buchstaben betragen, da sonst im Hauptprogramm Probleme mit der Anzeige auftauchen.

Nach der Ruckkehr ins Hauptmenue ist WORDMASTER hoffentlich gut gefüttert und zufrieden, es kann losgehen. Der gesamte Spielablauf findet im Dialog mit dem Programm statt, so daß eine umfangreiche Anleitung überflüssig ist. Wenn Sie sich entscheiden, die Partie zu beginnen, so müssen Sie zunächst ein Wort raten, das sich WORDMASTER ausgedacht hat. Neben der Eingabe Ihrer Versuche haben Sie jederzeit die Möglichkeit, mit den Cursortasten oben und unten die Wortliste zu durchsuchen, falls Ihnen nichts mehr einfällt. Im Notfall können Sie sich sogar durch Eingabe eines Fragezeichens das geheime Wort verraten lassen, doch dann werden Ihnen automatisch 10 Versuche angerechnet, und Sie haben die Runde mit großer Wahrscheinlichkeit verloren.

Falls Sie ein Wort eingeben, das WORDMASTER noch nicht kennt, so möchte das Programm wissen, ob das Wort neu in die Liste aufgenommen werden soll; nur in diesem Fall wird es als Versuch akzeptiert. Hüten Sie sich davor, unsinnige Buchstabenkombinationen zu verwenden, denn WORDMASTER paßt sich Ihrer Wortwahl an und benutzt die Eingabe bei nächster Gelegenheit womöglich als geheimen Code! Wenn das neu gewonnene Wissen erhalten bleiben soll, so muß natürlich nach Beendigung des Spiels die erweiterte Wortliste mit Punkt 4 des Hauptmenues neu abgespeichert werden. Auf diese Weise vergrößert das Programm mit jeder Spielrunde seine Wissensbasis.

Ihre Versuche und Trefferquoten werden in einem separaten Bildschirmfenster protokolliert. Bemühen Sie sich auf jeden Fall, mit möglichst wenigen Versuchen auszukommen! Nachdem Sie das gesuchte Wort gefunden haben, ist WORDMASTER an der Reihe, und Sie dürfen sich ein geheimes Wort ausdenken. Wie Sie sehen werden, findet das Programm den Code meistens sehr schnell, und wenn es dabei weniger Versuche braucht, haben Sie die Runde verloren. Andererseits kann es natürlich passieren, daß WORDMASTER das Wort noch nicht kennt, und in ausgleichender Gerechtigkeit werden dann dem Programm 10 Versuche aufgebrummt. Allerdings lernt es auch in diesem Fall dazu!

Die Versuchsergebnisse für WORDMASTER bestehen aus zwei Zahlen (richtige Buchstaben/richtige Positionen). Falls Sie nach der ersten Eingabe DEL drücken, so kann diese korrigiert werden; DEL vor der ersten Eingabe führt zum Neubeginn des Ratevorgangs. Dieser Abbruch ist nur sinnvoll, wenn Sie einen Eingabefehler in einem vorherigen Versuch entdecken, der WORDMASTER natürlich daran hindert, den Code zu entschlüsseln. Unbe-

merkte Eingabefehler werden vom Programm im allgemeinem nach Beendigung der (erfolglosen) Suche gefunden und angezeigt.

WORDMASTER behält also auf jeden Fall die Übersicht, und wenn man sich anschaut, wie zielstrebig das geheime Wort ermittelt wird, so scheint das Programm in der Tat die vorhin beschriebene wissenschaftliche Strategie spielend zu bewältigen. Doch ehrlich währt am längsten: Gegen allen Anschein kann es weder logische Schlüsse ziehen noch Theorien aufstellen und überprüfen – das einzige, was WORDMASTER wirklich leistet, ist eine überzeugende Simulation dieser Fähigkeiten!

Wer sich an Kapitel 5 (Mit roher Rechengewalt) erinnert und die Anzeige des Denkvorgangs auf dem Bildschirm kritisch betrachtet, wird bald erkennen, daß hier nur ein schneller Such-Algorithmus am Werke ist... aber stop! Damit das Spiel noch eine Weile seinen Reiz behält, soll es diesmal ausnahmsweise Ihnen überlassen bleiben, das genaue Verfahren herauszufinden. Die Programmzeilen 2880 – 3090 dürften für eine ausgiebige Analyse einige Hinweise geben, die hier jedoch nicht mehr zur Sprache kommen können, da wir noch Platz für ein anderes heißes Thema brauchen:

Der allgemeine Problemlöser

Bevor Sie mit dieser Überschrift den Verdacht bestätigt finden, daß der Autor dieser Zeilen nun endgültig einem galoppierenden Größenwahn verfallen ist, hier ein kurzer Rückblick in die Geschichte der künstlichen Intelligenz, der alles erklärt: Zu Beginn der Computerära war der Optimismus trotz der nach heutigem Standard erbärmlichen Hardware ungebrochen, und in den fünfziger Jahren wurde allen Ernstes an Programmen gearbeitet, die nicht nur spezielle Aufgaben, sondern jedes beliebiges Problem lösen sollten. Und ob Sie es glauben oder nicht: Die Bemühungen hatten sogar Erfolg! 1957 entstand der in Fachkreisen berühmte *General Problem Solver* (GPS) von Newell, Shaw und Simon. Damit drängt sich natürlich die Frage auf, warum in aller Welt dieses Programm nicht eingesetzt wurde, um endlich die großen Probleme der Menschheit zu lösen – doch dazu kommen wir gleich. Zunächst soll in Grundzügen dargestellt werden, mit welchem Ansatz dieses softwaremäßige Universalgenie arbeitet.

Alle Problem entstehen im Prinzip aus einer Differenz zwischen einem gegebenen Ist-Zustand und einem erwünschten Soll-Zustand und lassen sich

damit auf die Frage ‚Wie komme ich von A nach B' reduzieren. Sie sind zu Hause, wollen aber gerne nach Honolulu, oder sind ledig und möchten heiraten, oder haben kein Geld, brauchen aber mindestens zwei Riesen für Ihren neuen Wunsch-PC... einen Moment, das werden wir gleich haben: In der Welt, auf die sich das Problem bezieht, gibt es im allgemeinen nützliche Objekte (Verkehrsmittel, Zeitschriften, Sparkassen) und dazu passende Operatoren (Taxi bestellen, Heiratsanzeige aufgeben, Bank überfallen), mit denen sich der Zustand A transformieren, d.h. in einen anderen Zustand überführen läßt.

Natürlich kann man nicht erwarten, daß ein einziger Operator sofort den gewünschten Zustand B herstellt, sondern es müssen oft ganze Ketten von Operatoren in der richtigen Reihenfolge angewendet werden. Doch keine Sorge, die findet der GPS schon heraus. Wir müssen dem Programm nur vorher begreiflich machen, auf welche Weise welche Operatoren mit welchen Objekten auf welche Zustände wirken...

Sehen Sie – und genau deshalb hat sich der GPS nie durchsetzen können. Die Kodierung und Aufbereitung aller relevanten Daten ist nämlich so zeitaufwendig, daß sich der Beitrag des Programms zur Lösung des Problem dagegen sehr bescheiden ausnimmt, oder mit anderen Worten: Ohne Computer lassen sich die Probleme meist schneller lösen. Sinnvoll wäre der GPS nur, wenn er die Daten selbständig erfassen könnte, doch in dieser Hinsicht sind die Experten heute sehr pessimistisch. Alles in allem muß das Projekt deshalb als Fehlschlag bezeichnet werden, hat aber trotzdem der gesamten KI-Forschung wesentliche Impulse vermittelt. Außerdem ist die Idee einfach zu interessant, um sie hier unter den Tisch fallen zu lassen. Eine Demonstration der GPS-Arbeitsweise auf einem CPC ist nun zwar etwas zu hoch gegriffen – hier wäre endgültig der Umstieg auf PROLOG fällig – doch wie sieht es mit einem speziellen Problemlöser aus, der zumindest im Ansatz einige der Ideen verwirklicht? Zu diesem Zweck kommen uns die WORDMASTER-Wortlisten wie gerufen:

Ohne Bier wird Gerd zum Tier!

Vermutlich haben Sie den folgenden Aufgabentyp schon einmal in der Rätselecke einer Illustrierten gesehen:

```
GERD .... .... .... WOLF
```

In jeder Vollmondnacht verwandelt sich Gerd unweigerlich in einen Wolf – wie geht das vor sich? Es wird jeweils ein Buchstabe ausgetauscht, so daß ein neues gültiges Wort (aus der Liste) entsteht, und damit wären bereits die Operatoren für unser Problem definiert. Als Differenz zwischen dem Ist- und dem Soll-Zustand kann die Anzahl der unterschiedlichen Buchstaben gelten. Da die Differenz zu Beginn vier Buchstaben beträgt und die Aufgabe in vier Schritte gelöst werden soll, ist klar, daß jede Umwandlung die Differenz um 1 reduzieren muß; aus diesem Grund ist die Aufgabe auch nicht besonders schwierig.

Wesentlich härtere Probleme bringt jedoch eine andere Transformation mit sich, die in der Realität recht häufig vorkommt, nämlich die Umwandlung von GELD in BIER. Durch Probieren kann man sich davon überzeugen, daß sich die Differenz auf Anhieb nicht verringern läßt, da keiner der Buchstaben aus BIER in GELD hineinpaßt, d.h. zu einem existierenden Wort führt. Mit Sicherheit brauchen wir mehr als vier Schritte und müssen uns auf Umwege einlassen. Wieviel Schritte werden also mindestens notwendig sein? Diese Frage ist nicht ohne weiteres zu beantworten. Bekannt ist bisher eine Lösung, die sechs Zwischenstationen erfordert. Sie beginnt mit der Umwandlung von GELD in HELD – versuchen Sie mal, den Rest zu finden! Sehr behilflich war dem Autor dabei ein kleines Programm, das eine der WORDMASTER-Listen nach vierbuchstabigen Wörtern durchsucht, die sich von dem eingegebenen Wort genau in einem Buchstaben unterscheiden.

```
10 OPENOUT "X": MEMORY HIMEM-1: CLOSEOUT
20 OPENIN "LISTE4.DAT"
30 a=0: DIM w$(1000)
40 WHILE NOT EOF: a=a+1: INPUT#9,w$(a): WEND
50 CLOSEIN
60 INPUT "Wort: ",a$: a$=UPPER$(a$)
70 FOR i=1 TO a
80    dif=0
90    FOR j=1 TO 4
100     IF MID$(a$,j,1) <> MID$(w$(i),j,1) THEN dif=dif+1
110   NEXT j
120   IF dif=1 THEN PRINT w$(i);"
130 NEXT i
140 PRINT: PRINT: GOTO 60
```

Die Arbeit mit diesem Programm ist sehr aufschlußreich, da sie einiges über die Struktur der Wortliste verrät. Es gibt hier Inseln, also einzelne Wörter (z.B. AUTO) und kleine Wortgruppen, die keine weitergehende Transformation zulassen, oder auch Sackgassen, d.h. Wörter, die nur einen einzigen Umwandlungspartner haben und keine Verkettung in mehrere Richtungen erlauben. Die Liste erscheint unter diesem Aspekt als eine Art Wortlabyrinth, durch das sich der bierdurstige Forscher mühsam hindurchtastet. Sicherlich läßt sich damit eine neue und interessante Spielidee entwickeln, doch bleiben wir bei unserem Problemlöser: Gesucht ist ein Programm, das a) Aufgaben dieser Art löst, und b) Aufgaben dieser Art konstruiert!

Wenn wir es bei einfachen Aufgaben mit vier Schritten (siehe Gerd, der Werwolf) belassen, so scheint solch ein Programm zwar nicht unbedingt trivial, aber immerhin realisierbar zu sein, da mit der Reduzierung der Buchstabendifferenz eine klare Bedingung für jeden Schritt vorliegt. Haarig wird es jedoch bei unserem GELD-BIER-Problem: Was soll hier der Suche eine Richtung geben? Wie kann ein Programm Inseln und Sackgassen erkennen, Endlosschleifen verhindern?

Diese offenen Fragen wurden in der Ausgabe 1/87 von PC International an die Leser weitergereicht, in der Hoffnung, daß der eine oder andere versierte Datenbastler zumindest eine Teilantwort finden würde. Und die Reaktion übertraf in der Tat alle Erwartungen: Lesen Sie im nächsten Kapitel, was geschah…

KAPITEL 8

Wortketten-Problem gelöst!

In der Januarausgabe 1987 der Zeitschrift PC International wurde den Lesern die Aufgabe gestellt, ein Programm zu entwickeln, das zwei beliebige Wörter durch eine Wortkette verbindet, wobei jeweils nur ein Buchstabe ausgetauscht werden darf. Als besonders harte Nuß hatten wir das Problem präsentiert, auf diese Weise GELD in BIER zu verwandeln und weiterhin die Frage aufgeworfen, ob es überhaupt möglich ist, mit einem Mikrocomputer wie dem CPC Rätsel dieser Art generell zu lösen. Die Leserschaft ließ jedoch nicht lange auf eine Antwort warten: Bereits kurz nach Erscheinen von Heft 1/87 trafen die ersten Programm zu diesem Thema in der Redaktion ein, und nach einigen Tests und Experimenten mit dem Material blieb nur noch die Feststellung: Das Problem ist in der Tat gelöst!



Man sollte wirklich nicht das in der Mikrocomputerszene vorhandene kreative Potential unterschätzen: Hier werden unbehelligt von ökonomischen Zwängen oder wissenschaftlichen Zielsetzungen auf spielerische Weise Problem gelöst, die Profis mitunter nur ein müdes „Dazu brauche ich einen größeren Rechner“ entlocken. Herr Raab aus Nimtofte, Dänemark, der sich ebenfalls mit dem Wortketten-Problem beschäftigt hat, schrieb dazu: „Die tägliche Arbeit mit dem Computer läßt einem kaum Zeit für dergleichen, und die meisten Computersysteme heutzutage sind so komplex, daß man sich keine Experimente erlauben kann. Da ist der Computer zu Hause viel besser. Er ist zwar etwas langsam, aber dafür kann ich damit machen, was ich will, wenn das Gerät nicht gerade von meinen Söhnen beschlagnahmt ist.

Und wer unvorsichtigerweise den Ehrgeiz der Microcomputer-Freaks herausfordert, muß wissen was er tut: Gerade die Firma Commodore, die mit ihrem C 64 einiges zu dieser Entwicklung beigetragen hat, gehörte vor einiger Zeit zu den Leidtragenden, als die Wundermaschine Amiga der Öffentlichkeit präsentiert wurde. Wer kennt nicht den berühmten „Bouncing Ball“, der als bildschirmfüllendes Makro-Sprite neue Maßstäbe bei animierter Grafik setzte? Natürlich dauerte es nicht lange, bis der lästige Konkurrent Atari auf seinem ST mit einer getreuen Kopie der karierten Superkugel aufwarten

konnte. Gut, beeilte man sich bei Commodore mitzuteilen, man möge aber bitte beachten, daß der Atari damit bereits zu 90% ausgelastet sei, während der Amiga dank seiner Spezialchips ohne weiteres noch gleichzeitig ein Präludium von Bach und eine Partie Schach spielen könne.

Das Argument hatte etwas für sich. Als jedoch wiederum nur kurze Zeit später auf allen möglichen Mikros bis hin zum Sinclair Spectrum der hüpfende Ball auftauchte, zog man es bei Commodore vor, der weiteren Entwicklung mit weiser Zurückhaltung zu begegnen. Auch für den CPC existiert übrigens ein verblüffendes „Bouncing Ball"-Programm, bei dem gleich eine ganze Legion von Bällen auf dem Bildschirm vor einem bewegten Hintergrund (!) herumhüpft. Dieses Grafik-Demo, das in PC International 7/88 veröffentlicht wurde, muß man einfach sehen, um es zu glauben!

Das alles schmälert zwar nicht die Leistungsfähigkeit des Amiga als Grafikmaschine, zeigt aber, daß die 8-Bit-Zwerge immer noch nicht ausgereizt sind. Gerade die hardwaremäßigen Beschränkungen scheinen die Programmierer zu immer neuen Taten anzuregen, und von daher hätte man es sich eigentlich denken können: Die im 7. Kapitel angedeuteten Zweifel, ob eine Lösung des Wortkettenproblem auf einem CPC überhaupt möglich ist, waren für die PC International-Leser eine echte Herausforderung.

Wo liegt das Problem?

An dieser Stelle soll zunächst die Lösung des GELD/BIER-Rätsels verraten werden:

```
GELD - HELD - HERD - HERR - HEER - TEER - TIER - BIER
```

So geht's – und wenn man das Wort HIER zuläßt, kommt man sogar noch mit einem Schritt weniger aus. Um solch ein Rätsel zu lösen braucht der Computer genau wie ein Mensch einen gewissen Wortschatz. Welchen Einfluß der Umfang und die Zusammensetzung der Wortliste hat, beschreibt Herr Mühlenweg aus Krefeld: „Ob eine Transformation einfach (z.B. GERD-WOLF) oder schwierig ist, hängt nur vom Vokabular des Rechners ab. Fehlt beispielsweise das Wort GOLD, so wird aus dem einfachen ein schwieriges oder sogar unlösbares Problem".

Ein sehr großer Wortschatz ist allerdings noch keine Garantie für überragende Fähigkeiten bei der Suche nach Wortketten. Schließlich muß der

Rechner ja geeignete Wörter herausfiltern. Gerade das GELD /BIER-Problem leistet dabei einigen Widerstand. Nach welchem System kann man zum Beispiel erkennen, daß die Umwandlung in HELD einen geeigneten Ansatz darstellt? Herr Raab stellt die grundlegenden Schwierigkeiten auf sehr anschauliche Weise dar: „Zur Lösung des Problems braucht man ein Maß dafür, wie weit man zu jedem Zeitpunkt von dem gesetzten Ziel, also dem Wort BIER, entfernt ist. Die Anzahl der unterschiedlichen Buchstaben im Zielwort und einem Vergleichswort ist ein natürliches Maß für diesen Abstand. Stellt man sich das Zielwort als Berggipfel vor, so kann man die Differenz als Luftlinienabstand ansehen. Leider haben wir kein Maß für den Weg, den wir zurücklegen müssen, wenn wir nicht fliegen können, sondern auf einen Pfad angewiesen sind, d.h. auf benachbarte Wörter".

Der Computer als Pfadfinder in unwegsamen Gelände – es wird klar, daß es wohl kaum ohne einen effektiven Suchalgorithmus geht, der den Pfad durch Versuch und Irrtum ermittelt. Das erste Programm, das auf diese Weise eine Lösung anstrebt, traf bereits Anfang Januar ein und stammt von Herrn Klaas Wedemeyer aus Hamburg. Es benutzt ein sich selbst aufrufendes (rekursives) Unterprogramm, das zu dem Startwort ein benachbartes Wort (ein Buchstabe verschieden) aus der Liste sucht, zu diesem Wort wieder ein benachbartes Wort, und so weiter. Gerät das Programm dabei in eine Sackgasse, so kehrt es automatisch zum vorherigen Schritt zurück und probiert eine andere Fortsetzung. Endlosschleifen werden verhindert, indem jeder Kandidat daraufhin überprüft wird, ob er schon in der Wortkette vorkommt.

KI-Experte Martin Schlöter von PASCAL International hatte auch sofort den passenden Fachausdruck parat, als ich ihm das Programm zeigte. „Diese Methode, bei der der Computer in seiner eigenen Spur zurückwandert, um eine andere Abzweigung zu finden, heißt Backtracking und ist in der Sprache PROLOG gleich eingebaut", erläuterte er und murmelte dann noch etwas, daß sich wie „sowas macht man auch nicht in BASIC" anhörte. Bevor er sich wieder in die höheren Regionen der Programmierkunst zurückzog, gelang es mir jedoch, ein kleines LOGO-Programm abzustauben, daß die Rekursion sehr anschaulich grafisch darstellt:

```
to Baum :Länge :Winkel :Tiefe
if :Tiefe 0 then [stop]
fd :Länge
```

```
rt :Winkel / 2
Baum :Länge * 0.9 :Winkel :Tiefe - 1
lt :Winkel
Baum :Länge * 0.9 :Winkel :Tiefe - 1
rt :Winkel / 2
bk :Länge
end
```

Schildkröte mit Rückwärtsgang

Das LOGO-Programm malt also einen Baum und nutzt dabei die Tatsache aus, daß jeder Ast mit seinen Verzweigungen wieder als Baum betrachtet werden kann; die Routine braucht sich also nur selbst aufzurufen. Starten Sie, falls vorhanden, Ihren LOGO-Interpreter, tippen Sie das Listing ab und geben Sie dann z.B. ein:

```
Baum 50 60 4
```

Der erste Parameter bestimmt die Länge der Äste, der zweite den Öffnungswinkel bei einer Verzweigung und der dritte die Rekursionstiefe. Betrachtet man den Weg der Schildkröte auf dem Bildschirm so kann man sich gut vorstellen, wie der Suchvorgang in der Wortliste abläuft: Hat das grafisch talentierte Reptil einen Endpunkt im Baum erreicht, so klettert es wieder abwärts, um an einer anderen Stelle weiterzumalen, an der noch eine Verzweigung fehlt.

Das Prinzip sieht auf den ersten Blick vielversprechend aus; die ersten Versuche mit einer umfangreichen Wortliste verliefen jedoch enttäuschend: Da ja alle denkbaren Folgen ohne Einschränkung untersucht werden, verirrt sich der Rechner ziemlich schnell in endlosen Wortketten. Zwar garantiert das Verfahren, daß alle existierenden Lösungen gefunden werden (egal wie lang), braucht dazu jedoch sehr viel Zeit. Außerdem interessiert eine Lösung in 20 Schritten kaum, wenn es auch eine in 6 Schritten gibt. Was also tun? Doch die Frage war bald beantwortet: Nur kurze Zeit später traf das Listing von Herrn Mühlenweg ein, mitsamt einigen weiteren Ideen.

In seinem Programm wird der Benutzer zu Beginn gefragt, wieviele Schritte die Kette maximal enthalten soll, wodurch zwar eine längere Lösung hinter dem Suchhorizont verschwindet, aber dafür die Rechenzeit kontrollier-

bar wird. Zusätzlich sieht der Autor noch eine weitere Möglichkeit, die Rechenzeit zu verkürzen: „Wenn keine Aussicht mehr besteht, daß in den verbleibenden Suchbaumetagen (bis zum vorgegebenen Suchhorizont) noch alle ungleichen Buchstaben ausgetauscht werden können, kommt das betreffende Wort nicht mehr in Frage. Der Suchaufwand wird hierdurch sicherlich reduziert, ohne daß sich das Ergebnis ändert“

Befinden sich z.B. zwei Schritte vor dem Ende der Kette noch drei falsche Buchstaben in einem Wort, wird es überhaupt nicht mehr in Betracht gezogen. Diese zielgerichtete Suche vermeidet in der Tat eine Menge Irrwege. Störend macht sich nur bemerkbar, daß das Programm immer wieder auf Umwege hereinfällt, wie etwa

```
HASE - NASE - VASE
```

Von HASE nach VASE hätte auch ein Schritt gereicht, wie man sofort sieht. Doch auch hier gibt es Abhilfe: Es muß nur abgefragt werden, ob sich ein Wort vom vorvorigen Wort der Kette in nur einem Buchstaben unterscheidet. Ist das der Fall, so liegt offensichtlich ein Umweg vor, und das Wort wird verworfen. Zwar vermeidet diese Abfrage nicht alle Umwege (sie können sich ja über mehr als drei Schritte erstrecken), eliminiert aber den häufigsten Fall wirkungsvoll. Im Prinzip könnte man diesen Test auf weitere Wörter der Kette ausdehnen, es fragt sich nur, ob dadurch auf Dauer nicht mehr Zeit verbraucht als eingespart wird.

So weit, so gut – faßt man diese Ideen zusammen, so erhält man bereits ein recht brauchbares Programm. Das BASIC-Programm **WKETTE1.BAS** zeigt, wie die verschiedenen Elemente realisiert werden. Als Arbeitsgrundlage dient eine Liste vierbuchstabiger Wörter (LISTE4.DAT), die bereits im letzten Kapitel von WORDMASTER benutzt wurde und vom Wortketten-Programm in den Zeilen 190 – 250 in das Array wort$ geladen wird, bevor der Benutzer das Rätsel eingibt.



Nach dem Start mit **RUN „WKETTE1“** erwartet das Programm zunächst drei Eingaben von Ihnen, die jeweils mit RETURN bestätigt werden: Das vierbuchstabige Startwort, das Endwort, und die Anzahl Schritte, die maximal zur Bildung der Wortkette erlaubt sind.

Die Hauptarbeit leistet das rekursive Unterprogramm ab Zeile 450. Die aktuelle Suchebene ist in der Variablen schritt vermerkt; sie wird bei jedem Aufruf um 1 erhöht. Weiterhin existiert für jede Ebene ein Zeiger, der das gerade untersuchte Wort in der Liste (d.h. im Array wort$) markiert. Bevor nun ein Kandidat als nächstes Glied der Kette akzeptiert wird, muß er einige strenge Prüfungen über sich ergehen lassen. Fällt er durch, so springt das Programm sofort nach Zeile 890, wo der Zeiger auf das nächste Wort gerichtet wird. Ist das Ende der Wortliste erreicht, so wird die Variable schritt um 1 vermindert, und das Programm kehrt auf die vorherige Ebene zurück.

Erfüllt der Kandidat jedoch alle Kriterien, so wird er in Zeile 840 in das Array kette$ eingetragen. Beträgt die Differenz zum Zielwort (zdif) an dieser Stelle nur noch 1, so liegt eine Lösung vor, die mit GOSUB 960 angezeigt wird; andernfalls ruft die Routine sich mit GOSUB 450 selbst auf, um die Kandidaten für den nächsten Schritt zu untersuchen. Auf dem Bildschirm kann man dabei sehr schön beobachten, wie das Programm die Wortketten auf- und notfalls auch wieder abbaut, falls sie nicht zum gewünschten Ziel führen.

Der Weisheit letzter Schluß?

Nein, das ist unser Programm mit Sicherheit nicht. Um es gut lesbar zu gestalten, wurde bewußt darauf verzichtet, mit allen Tricks und Kniffen eine maximale Geschwindigkeit zu erreichen. Ambitionierte Bastler können hier mit Sicherheit noch einiges herausholen. Aber abgesehen von programmtechnischen Fragen: Ist dieser Algorithmus wirklich die einzige Methode, um das Wortkettenproblem zu lösen? Weitere Leserzuschriften beweisen, daß es durchaus noch Alternativen gibt. Sie beruhen auf der Idee, die Wortliste im Speicher gleich von vornherein so zu strukturieren, daß eine besonders effektive Suche möglich wird.

Herr Maleschka aus Hagen schickte uns z.B. ein Pascal-Programm mit folgender Idee: „Die Daten werden im Speicher in sogenannten Inseln abgelegt. Das Besondere an den Wörtern einer Insel ist, daß man von jedem Wort einer Insel zu jedem anderen Wort der Insel mit Sicherheit einen Weg findet." Gleich bei der Eingabe eines neuen Wortes wird untersucht, ob es in den bereits bestehenden Inseln einen Nachbarn gibt. Ist das der Fall, so wird es dort einsortiert. Finden sich auf mehreren Inseln Nachbarn, so bildet das Wort quasi eine Brücke zwischen den Wortgruppen, und sie werden zusammengefaßt. Der Vorteil liegt auf der Hand: Das Programm kann sehr schnell feststellen, ob es überhaupt eine Lösung gibt (Start- und Zielwort müssen sich ja gemeinsam auf einer Insel befinden!) und kann bei der Suche die überseeischen Gebiete vollkommen außer acht lassen.

Bei der Strukturierung von Daten tut sich BASIC allerdings recht schwer. Abgesehen von den unflexiblen Arrays werden keine Hilfsmittel geboten, im Gegensatz zu Pascal: Mit Records, Pointern und verketteten Listen stellen komplexe Datenstrukturen kein Problem dar. Man kann allerdings auch die andere Richtung wählen und in die Tiefen der Assemblerprogrammierung herabsteigen, wo sich Daten ebenfalls sehr flexibel verwalten lassen – zwar nicht so komfortabel und ohne Sicherheitsgurte, aber dafür sehr flott. Herr Gerald Steffens aus Harsewinkel hat diesen Weg eingeschlagen und ein Programm eingesandt, daß in der Tat beeindruckende Leistungen zeigt. Es ordnet die Wortliste im Speicher in Form eines Graphen an: Zu jedem Wort existiert eine Gruppe von Zeigern, die die Speicheradressen der benachbarten Wörter angeben. Schreibt man alle Wörter auf ein großes Blatt Papier und verbindet die benachbarten Wörter durch Linien, so hat man die Struktur bildlich vor Augen.

Den Graphen muß der Rechner nur einmal zu Beginn aufbauen (außer es kommen neue Wörter hinzu), und danach geht die Post ab: Da das Programm die benachbarten Wörter nicht mehr mühsam suchen muß, sondern gleich einen entsprechenden Hinweis im Speicher findet, kann es sich in Windeseile durch den Graphen hindurchhangeln. Weiterhin hilft ein fundierter theoretischer Hintergrund bei der Programmierung: „Die reine mathematische Idee, die übrigbleibt, wenn man das Problem analysiert, findet sich in der Graphentheorie und Kombinatorik wieder“, schreibt Herr Steffens und führt in diesem Zusammenhang den Algorithmus von Dijkstra an, den er auch in seinem Programm als Grundlage benutzt hat.

Alle auf einen Streich!

Leider reicht der Platz nicht, um diese Methode ausführlich zu diskutieren, deshalb nur soviel: Es handelt sich im Prinzip um einen Labyrinth-Algorithmus, der mit nachtwandlerischer Sicherheit den kürzesten Weg im Graphen ermittelt, und zwar nicht nur zu einem vorgegebenen Zielwort, sondern gleich zu allen erreichbaren Punkten. Es ist schon beeindruckend, wie das Programm nach Eingabe des Startwortes eine Liste aller überhaupt nur möglichen Zielwörter ausspuckt, mitsamt der minimalen Schrittzahl, in der sie erreicht werden können. Nach Eingabe eines der Wörter wird dann prompt die dazugehörige Kette geliefert. Natürlich hat man auf diese Weise auch gleich eine große Anzahl potentieller Aufgaben zur Auswahl – also das ideale Programm für eine Rätselzeitschrift!

Das Programm besteht aus zwei Teilen. Zuerst wird der Lader für den Maschinencode mit **RUN „WKETTE2“** gestartet, worauf dieser dann automatisch das Hauptprogramm KETTEV2 nachlädt. Hier noch ein paar Erläuterungen zu den Punkten des Hauptmenues:

Laden: Es kann wahlweise eine Wortliste im Binär- oder im ASCIIFormat geladen werden. **LISTE4.DAT** auf Ihrer Diskette/Kassette ist eine ASCII-Datei.

Speichern: Die Wortliste wird nach Angabe einer Dateinummer binär abgespeichert.

Ausgabe: Die Liste wird auf dem Bildschirm ausgegeben.

Wort einfügen: Ein neues Wort kann eingegeben werden.

Korrektur: Ein falsches Wort wird gegen das richtige ausgetauscht.

Einstieg ins Suchprogramm: Dient zur Eingabe des Start- und Zielwortes.

Zurück ins Hauptmenue kommt man jederzeit durch Drücken von ENTER ohne eine vorherige Eingabe.

Damit liegt nun ein nahezu perfektes Wortketten-Programm vor, und es bleibt eigentlich nur noch, allen Lesern zu danken, die – genannt oder ungenannt – in ihren Beiträgen das Problem von allen Seiten beleuchtet, analysiert und schließlich gelöst haben. Durch ihre tatkräftige Mithilfe ist dieses Kapitel des Software-Experiments erst möglich geworden.

KAPITEL 9

Entwicklungshilfe per Computer

von H. Brendjes / J-D. Kaufmann

Sollen Computer im Schulunterricht eingesetzt werden? Hier sind die Meinungen der Pädagogen gespalten: In einer Welt, in der Kinder ohnehin der Allgegenwart elektronischer Medien ausgeliefert sind, sollten sie nicht auch noch in der Schule vor dem Bildschirm sitzen, warnen die Kritiker. Kinder und Jugendliche müssen sich ohnehin früher oder später mit der Computertechnologie auseinandersetzen, halten die Befürworter dem entgegen, warum soll das nicht auf sinnvolle Weise im Unterricht geschehen? Doch wie man es auch dreht und wendet: Die Frage wird letztendlich nicht durch theoretische Erwägungen, sondern durch praktische Erfahrungen im Schulalltag geklärt werden.

Doch die Entwicklung steckt noch in den Kinderschuhen; geeignete Software ist bisher kaum verfügbar. Experimentierfreudigen Lehrern bleibt deshalb meistens nichts anderes übrig, als die Programme selbst zu schreiben. Um Material zur Verfügung zu stellen und Erfahrungen zu vermitteln, veröffentlichen wir den folgenden Bericht, der die Arbeit mit einer Computersimulation im Erdkunde-Unterricht beschreibt.

1. Entwicklungshilfe als Thema des Erdkunde-Unterrichts

Das Unterrichtsthema Entwicklungshilfe bedarf seinerseits einer Hilfe zur Entwicklung. Zunehmend sind Lehrer wie Schüler überfordert, wenn es darum geht, die für dieses Lernfeld gesteckten Ziele zu erreichen. Lehrer wie

Schüler stehen im Widerspruch zwischen der theoretischen Behandlung von Entwicklungsbemühungen und der Praxis, nämlich der häufigen Darstellung von Verarmung in der Dritten Welt durch die Medien gegenüber.

Die Problematik des Themas *Dritte Welt* liegt in seiner Komplexität. Sie spiegelt sich allein schon in dem Unvermögen wieder, zentrale Begriffe wie Unterentwicklung oder Entwicklung exakt und allgemeingültig zu definieren. Einseitige Erklärungsversuche von Unterentwicklung und ebenso einseitig begründete Strategien der Entwicklungshilfe bringen allenfalls Halbwahrheiten hervor. Unterentwicklung und Entwicklung sind jedoch durchweg weitverzweigte Problem, die ein vielschichtiges Vorgehen erfordern und deren Veranschaulichung durch Medien wie Text und Grafik zwangsläufig unzureichend sein muß.

Wer Entwicklungsprobleme im Unterricht verständlich machen will, steht vor einer doppelten Schwierigkeit: Entwicklungsländer und ihre Probleme lernt der Schüler nicht durch eigene Anschauung und Erfahrung kennen, sondern nur durch indirekte und ausschnittsweise Vermittlung über Medien. Was Hunger in einem Sahelland bedeutet, kann er auch dann nicht nachempfinden, wenn er freiwillig einige Tage auf alle Nahrungsmittel verzichtet. Hilfe für diese Probleme reduziert sich für den Schüler in aller Regel auf Nahrungsmittelhilfe und das klassische Bohren von Brunnen.

Die Beurteilung der Auswirkungen dieser Maßnahmen auf das Umfeld erfordert ein Begreifen der netzwerkartigen soziologischen und ökologischen Strukturen, die auf herkömmliche Weise einem Schüler nicht vermittelt werden können. Hier bietet sich der Einsatz von Computern im Schulunterricht geradezu an, um mit Hilfe eines geeigneten mathematischen Modells die Situation unter Zugrundelegung gewisser Vereinfachungen und Annahmen zu simulieren.

2. MORETTI im Schuleinsatz

Im Sommer 1985 lief in den dritten Fernsehprogrammen eine Sendung, in der sechs Teilnehmer eines Volkshochschulkurses ein Entwicklungshilfeprojekt in einer Computersimulation durchführen sollten. Die Kommunikation mit dem Rechner erfolgte ausschließlich über ausgedruckte Tabellen und einen Moderator, der die Tastatur bediente. Die Idee, einen derart komplexen Zusammenhang mit Hilfe eines Rechners durchspielen zu lassen, fanden wir

durchaus überzeugend; die praktische Durchführung im Schulunterricht erforderte jedoch einige Änderungen.

Der Umgang mit dem Programm/Rechner war so zu gestalten, daß kein Moderator/Lehrer zur Verständigung mit dem Rechner erforderlich ist, sondern jeder Schüler allein oder in einer kleinen Gruppe (bis 3 Schüler) dieses leicht bewältigen kann. Auch waren die Daten zur Situationsanalyse (Ist-Zustand) sowie zur Planung der Maßnahmen für die nächsten Jahre (Soll-Zustand) so darzustellen, daß sie sich Schülern der Klassen 9 bis 11 in einer verständlichen, ansprechenden Form präsentieren. Nicht zuletzt erschienen uns die im Programm eingearbeiteten Zusammenhänge teilweise noch zu stark vereinfacht bzw. unvollständig.

In den kalten norddeutschen Wintermonaten 1985/86 haben wir (zwei Realschullehrer und Hobby-Informatiker) in rund 100 Mannstunden ein Programm für den Schneider CPC gestrickt, das unseren Anforderungen entsprach und das anschließend im Unterricht einer 10. Klasse (Erdkunde Wahlpflichtkurs) durch den dort unterrichtenden Kollegen erprobt wurde. Dieser Kollege hatte zunächst eine ausgeprägte Computer-Phobie, wurde dann von uns aber zu seinem Glück gedrängt: Er setzte es im Rahmen seiner Unterrichtseinheit etwa 10 Stunden lang ein und war danach ebenso wie seine Schüler begeistert.

Das Programm MORETTI.BAS soll nun stets zum Einüben theoretisch erarbeiteter Inhalte beim Thema Entwicklungshilfe eingesetzt werden. Der Einsatz des Computers in der Schule ist aus unserer Sicht sehr fragwürdig, wenn ein Thema den Schülern besser auf andere Weise nahegebracht werden kann. In diesem Fall aber erschließt die Arbeit mit dem Computer ganz neue Bereiche; nur so ist es möglich, die Entwicklungen mathematisch beschreibbarer System auf anschauliche Weise zu studieren. Der entscheidende Vorteil liegt darin, daß Schüler nunmehr theoretische Zusammenhänge praktisch und spielerisch erfahren und anwenden können; manche Zusammenhänge offenbaren sich sogar erst durch den handelnden Umgang mit der Simulation.

3. Probleme der Sahelzone

Die Sahelzone Westafrikas als Übergangsraum zwischen der Wüste Sahara und der tropischen Savanna ist seit langer Zeit eine Kontaktzone zwi-

schen nomadischer und seßhafter Bevölkerung. Das ermöglicht auf der einen Seite den Austausch von Anbauprodukten und Produkten der Viehwirtschaft, führt aber durch den doppelten Nutzungsanspruch zu schwerwiegenden Problemen in der landwirtschaftlichen Entwicklung. Insbesondere gilt das, wenn Dürreperioden die Nutzungsmöglichkeiten des Lebensraumes beschränken. Als lebenswichtige Frage erweist sich das Verhältnis von Ökopotential und tatsächlicher Nutzung der natürlichen Ressourcen. Dabei hat sich gezeigt, daß eine landwirtschaftliche Übernutzung, sowohl in Anbaubereich als auch in der Viehwirtschaft, schwerwiegende Schäden hervorruft und eine Ausdehnung von Arealen mit wüstenhaften Bedingungen begünstigt. (1)

3.1. Niederschläge

Die Schwankungsbreite der Niederschläge ist sehr hoch, sie beträgt teilweise mehr als 50% in Dürrejahren. Man darf daher bei der Beurteilung der zu erwartenden Niederschläge nicht vom absoluten Jahresmittelwert ausgehen. Die Zahl der trockenen Jahre überwiegt die der feuchteren Jahre deutlich. Entscheidendes Merkmal der Sahelzone ist die kurze sommerliche Regenzeit mit hoher Variabilität der Regenmengen und die Gefahr, daß mehrere aufeinanderfolgende trockene Jahre den Grundwasserspiegel absinken lassen und so eine Dürrekatastrophe auslösen.

3.2. Viehwirtschaft

In günstigen Jahren findet eine starke Vermehrung der Herden statt. Das führt zu einer Überweidung: Die Weidefläche wird durch Abgrasen derart zerstört, daß eine natürliche Regeneration des Pflanzenbestandes ausbleibt. Das Gebiet wird so zur Wüste, wodurch sich das Nahrungsangebot für das Vieh vermindert. In Dürrejahren verhungern dann große Teile des in besseren Zeiten angewachsenen Viehbestandes, weil ein rechtzeitiger Verkauf und damit eine Kapitalverbesserung unterbleibt. Ursache dafür ist ein historisch begründetes Eigentumsverständnis: Der Eigentümer kann über das Vieh nicht nach eigenem Gutdünken verfügen, sondern er ist dafür im Auftrag seiner Ahnen verantwortlich, und das heißt natürlich Besitzstandswahrung als Nachweis besonderer Macht und Stellung. So ist zum Beispiel in Nordghana der Aufbau einer Fleischindustrie versucht worden, aber trotz vorhandener Herden haben die Bauern nur wenige Tiere verkauft, was für die Fabrik nicht ausreichte.

3.3 Grundwasser und Brunnen

Die Wasserversorgung im Sahel ist dort durch Verwüstung gefährdet, wo eine Überzahl von Brunnen die Grundwasservorräte rasch schwinden läßt. Es bilden sich dabei ringartig um die Brunnen auftretende Flächen, in denen keine Vegetation mehr möglich ist: Der Grundwasserspiegel sinkt, außerdem werden die Pflanzen durch die sich an den Brunnen konzentrierenden Herden abgefressen und zertrampelt. Auf diese Weise entstehen um Tiefbrunnen Kahlflächenringe mit bis zu 32 km Durchmesser. Auch in noch weiterem Umkreis stirbt ein Teil der Vegetation an Wassermangel.

3.4 Bevölkerungsentwicklung

Die schnelle Zunahm der Bevölkerung in den Entwicklungsländern ist eines der größten Hindernisse, um den Kreislauf der Armut zu durchbrechen. Nach den gegenwärtigen Schätzungen wird die Weltbevölkerung von zur Zeit etwa 4,7 Milliarden Menschen bis zum Jahr 2000 auf etwa 6,1 Milliarden Menschen anwachsen. Da rund 80% dieses Zuwachses in den Entwicklungsländern stattfindet, werden statt der gegenwärtig gut 70% zur Jahrtausendwende 78% der Weltbevölkerung in Entwicklungsländern leben. Die Ursachen dieser Bevölkerungsexplosion liegen sowohl in den steigenden Geburtenziffern als auch in den fallenden Sterberaten. Durch die Verbreitung der Errungenschaften der modernen Medizin, die Bekämpfung der Seuchen durch Massenimpfungen und Verbesserung der hygienischen Verhältnisse konnte die durchschnittliche Lebenserwartung der Bevölkerung in den Entwicklungsländern auf 55 Jahre gesteigert werden (zum Vergleich: in Mitteleuropa ca. 70 Jahre). Familienplanung ist ein entscheidender Beitrag, die Bevölkerungsexplosion in den Griff zu bekommen. (2)

3.5 Handel auf Märkten

Märkte werden in zentralen Orten und größeren Dörfern alle paar Tage abgehalten. Das bedeutet aber nicht, daß dort ein reger Handel nach marktwirtschaftlichen Gesichtspunkten stattfindet, wie wir es gewohnt sind. Vieh, Geflügel oder Getreide (Hirse) werden nur verkauft, um den Erlös zum Erwerb lebensnotwendiger Güter anderer Art zu verwenden, die nicht selbst produziert werden können. Angebot und Nachfrage auf dem Markt beeinflussen kaum die Produktion von landwirtschaftlichen Gütern.

4. Mathematisches Modell der Simulation MORETTI

Alle im vorangegangen Abschnitt aufgeführten Aspekte und ihre wechselseitigen Abhängigkeiten und Rückkopplungen sind von uns durch mathematische Gleichungen beschrieben worden. Sie bilden das Herzstück der Simulation.

Bei der Konstruktion dieser Gleichungen haben wir uns zum einen durch einen Experten für die Landwirtschaft der dritten Welt fachkundig beraten lassen, zum anderen haben wir durch viele Berechnungen und Probeläufe von Programmteilen das Verhalten des Modells getestet. Dabei zeigte sich, daß die sehr oft in anderen Programmen verwendeten linearen Beziehungen nicht zu wirklichkeitsgetreuen Ergebnissen führen. So wird die Erhöhung der Brunnenzahl von 2 auf 4 andere Auswirkungen haben als die Erhöhung von 4 auf 6 oder von 7 auf 9. Weiterhin hängt die Vegetationsmenge noch von anderen Parametern ab: Rinderzahl, Regen, Grundwasserstand, Vegetationsfläche und Verdunstung.

5. Die Bedienung des Programms

An dieser Stelle zunächst ein wichtiger Hinweis für CPC 464-Besitzer: Das Programm **MORETTI.BAS** nutzt den erweiterten Basic-Befehlssatz des CPC 664/6128. Um es trotzdem auf Ihrem Rechner laufen lassen zu können, müssen Sie die 664/6128-Emulation installieren, die sich auf Ihrer Diskette / Kassette befindet und den erweiterten Basic-Befehlssatz auch auf dem CPC 464 zur Verfügung stellt. Das geschieht, indem Sie **RUN „EMU“** eingeben und dann warten, bis die Meldung *Basic 1.1 installiert* auf dem Bildschirm erscheint. Danach können Sie dann das Moretti-Programm normal mit **RUN „MORETTI“** starten. Weitere Informationen zum Emulator finden Sie im Anhang.

Im Simulationsprogramm MORETTI geht es darum, eine Gruppe von Menschen in der Savanne Afrikas, eben die Moretti, durch geeignete Maßnahmen zu einer größeren Stabilität der Lebensbedingungen zu führen, d.h. dafür zu sorgen, daß die Ernährung ausreichend ist, genügend Wasser vorhanden ist und Mensch und Tier gegen Seuchen ausreichend unempfindlich werden. Zu diesem Zweck steht ein Kapital von 500.000 DM zur Verfügung, mit dem der Entwicklungshelfer gut haushalten muß.

Durch sinnvollen Einsatz dieser Mittel soll er die Menschen in die Lage versetzen, im weiteren ohne Zuschüsse auszukommen und notwendige Geldmittel durch den Verkauf von Rindern zu erlösen. Gelingt das nicht und sinkt die Bevölkerungszahl oder die Rinderzahl unter eine bestimmte Grenze, so ist das Projekt gescheitert, was dann auch angezeigt wird.

Die Parameter sind durch das oben erwähnte mathematische Modell miteinander verknüpft und bieten alle Möglichkeiten zur Durchführung eines eigenen Konzeptes, das natürlich alle Fehler vergangener Entwicklungshilfepolitik enthalten kann: Erschöpfung des Grundwassers durch Tiefbrunnen, Beschränkung auf Nahrungsmittelhilfe, medizinische Versorgung ohne Geburtenregelung etc. Ein Projekt verläuft erfolgreich, wenn es gelingt, die Moretti über 30 Jahre in ihrer zahlenmäßigen Größe etwa konstant zu halten, und zwar möglichst bei verbesserten Lebensbedingungen (Rinder je Einwohner = Lebensstandard) und einer relativ großen Unempfindlichkeit gegen natürliche Einwirkungen (Seuchen, Trockenperioden).

Zur weiteren Erläuterung des Vorgehens im Programm sind im folgenden die verschiedenen Bildschirme dargestellt und kommentiert; es kann über den Hauptschirm (Chronologie) stets zu allen Schirmen gewechselt werden. Bei der Arbeit mit dieser Simulation ist häufig die Unterrichtsstunde eher zu Ende als das Projekt; aus diesem Grund kann zu jeder Zeit der aktuelle Stand mit der Taste S auf einer Diskette gespeichert werden, um in einer anderen Stunde nach dem Laden via L mit dem letzten Stand fortzufahren.

5.1 Hauptschirm, genannt Chronologie (Taste C)

Der obere Teil enthält die wesentlichen Informationen der jeweils letzten drei Jahre, so daß Tendenzen erkannt und beurteilt werden können. Beim Grundwasserstand bedeutet +0.0 den Mittelwert. Abweichungen davon ergeben sich durch Regen, Trockenheit und den Betrieb von Brunnen. Der untere Teil gibt eine Kurzübersicht über die für das laufende Jahr getroffenen Maßnahmen, um so in Verbindung mit den anderen Informationen eine Grundlage zur Planung des folgenden Jahres zu schaffen. Die Taste TAB blendet die Tastenbelegung zur Anwahl anderer Bildschirme ein. Groß- und Kleinschreibung ist dabei bedeutungslos; die Betätigung nicht belegter Tasten wird vom Programm ignoriert und führt nicht zu Fehlreaktionen.



5.2 Entwicklungshilfemaßnahmen (Taste M)

In diesem Schirm können verschiedene Maßnahmen für das jeweils folgende Jahr geplant werden, die unterschiedlich intensiv mit entsprechend unterschiedlichen Auswirkungen sind. Dieser Schirm dient sowohl als Übersicht und Erläuterung der Maßnahmen, als auch zu deren Veränderung (Taste V). Bei der Bekämpfung der Rinderseuche, dem Gebrauch von Antikonzeptiva (Mittel zur Empfängnisverhütung) sowie der medizinischen Versorgung ist zu beachten, daß beim Erhöhen der Intensitätsstufe jeweils die Grundkosten pro Stufe anfallen, so daß ein häufiger Wechsel der Stufen entsprechend viel Kapital verbraucht. Zum Kauf/Verkauf von Rindern bzw. zum Hirsekauf wird in einen anderen Schirm gewechselt, der die Marktsituation darstellt. Angebot und Nachfrage sowie die Preise verändern sich von Jahr zu Jahr. Alle hier getroffenen Entscheidungen werden erst wirksam, wenn im Hauptschirm (Chronologie) in das Folgejahr übergewechselt wird.



5.3 Rindermarkt von Dorso Duro

Der Rindermarkt dient dazu, nach Seuchen die Herden zu vergrößern bzw. bei entsprechend großen Herden Mittel für notwendige Maßnahmen zu erlösen. Der Übergang in diesen Schirm erfolgt, wie auch im nächsten Punkt, beim Verändern automatisch.



5.4 Hirsemarkt von Grana Nera

Hier kann Hirse gekauft werden, die als Nahrungsmittelhilfe dient. Ist Hirse gekauft, jedoch nicht benötigt worden, so wird sie teilweise gegessen und verfüttert oder gelagert, wobei ein Teil der eingelagerten Hirse verdirbt, so daß der Vorrat langsam abnimmt.

5.5 Darstellung der kritischen Werte (Taste K)

Dieser Schirm bietet eine Hilfe bei der Einschätzung der Lage der Moretti. In zwei Balkendiagrammen wird dargestellt, wie gut die Nahrungsgrundlage für Menschen bzw. Rinder ist. Bei Werten unterhalb des Existenzminimums stirbt ein Teil der Tiere bzw. Moretti. Die Versorgung der Rinder ist eine wesentliche Entscheidungshilfe beim Kauf oder Verkauf von Tieren; so ist etwa der Kauf von Rindern sinnlos, wenn schon die vorhandenen Tiere zu wenig Futter haben.

Die Grafik für die Versorgung der Menschen berücksichtigt nur die Zahl der Rinder als Nahrungsgrundlage, so daß bei vorhandenen Hirsevorräten oder rechtzeitigem Kauf von Hirse Werte unterhalb des Existenzminimums überbrückt werden können, ohne daß es Hungertote gibt.

5.6 Grafik der zahlenmäßigen Entwicklung (Taste G)

Hauptaufgabe dieser Grafik ist es, die zahlenmäßige Entwicklung der Menschen und Rinder zu zeigen und damit sowohl den Verlauf des Projekts über den gesamten Zeitraum von 30 Jahren zu dokumentieren, als auch im Überblick langfristige Entwicklungen und deren Ursachen sichtbar zu machen. Dazu dient speziell die Einblendung farbig abgesetzter Rechtecke auf der Zeitachse, die die Zeitpunkte des Auftretens von Seuchen bei Menschen (unterhalb der Achse) und Tieren (oberhalb der Achse) markieren und dadurch eine Erklärung für den Verlauf der beiden Kurven liefern. Durch den Vergleich mehrerer Grafiken nach Ablauf von 30 Jahren lassen sich sehr gut die Auswirkungen verschiedener Entwicklungshilfestrategien aufzeigen und deren Wirksamkeit diskutieren.

Literaturverzeichnis:

(1) Der Bundesminister für wirtschaftliche Zusammenarbeit: Leben am Rande der Sahara, Rautenstrauch-Jöst-Museum, Köln 1981

(2) Der Bundesminister für wirtschaftliche Zusammenarbeit: Politik der Partner, 6. Auflage 1983

Der Bundesminister fÜr wirtschaftliche Zusammenarbeit: Zusammenarbeit mit Entwicklungsländern: So sieht die Praxis aus, 4. Auflage 1984

B. Rinder: Unterrichtsskizze Dürrekatastrophe im Sahel. In: Geographie im Unterricht, 6/80; Friedrich Verlag, Seelze

KAPITEL 10

Die Leser haben das Wort

In diesem Kapitel finden Sie zwei Leserbeiträge zum Software-Experiment, die stellvertretend für zahlreiche weitere Zuschriften anläßlich dieser Serie stehen, und als Zugabe die CPC-Version eines Beitrags, der ursprünglich in der Zeitschrift PASCAL International veröffentlicht wurde.

Der eindimensionale Zellautomat

Bereits im zweiten Kapitel wurden sich selbst organisierende Systeme und Zellautomaten behandelt. Eine besonders interessante und zudem grafisch sehr reizvolle Variante dieses Themas haben wir Herrn Brinkmann aus Ankum zu verdanken.

Das Programm *Zellautomat* (**ZAUTOMAT.BAS**) ist ein Hilfsmittel für die grafische Darstellung eindimensionaler Zellautomaten. Im Gegensatz zum Game of Life, das sich auf einer Fläche ausbreitet, entwickelt sich der eindimensionale Zellautomat entlang einer Linie. Das Schicksal einer Zelle in der folgenden Generation wird dabei vom Zustand der linken und rechten Nachbarzelle und der betrachteten Zelle bestimmt. Indem man Generation auf Generation untereinander anordnet, ist es möglich, die ganze Entwicklung in einem Bild sichtbar zu machen. Die Logik bzw. die Regeln des jeweiligen Zellautomaten können vom Benutzer frei definiert werden.

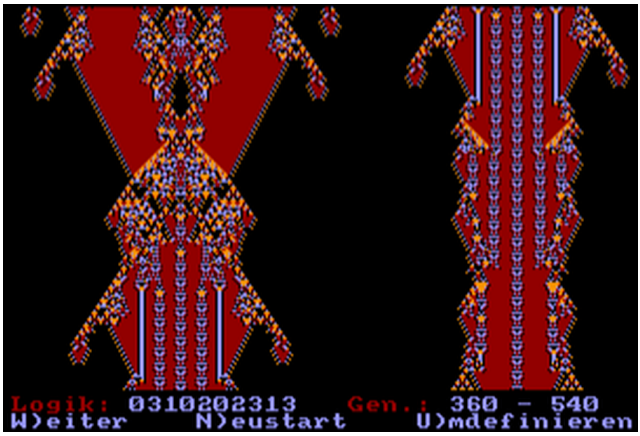
Theoretisch sind 4^{10} (mehr als 1 Million) verschiedene Zuordnungen denkbar. Davon sind allerdings die meisten uninteressant. Solche Zellautomaten können nicht nur hübsche Grafiken erzeugen, sondern auch als Modell für physikalische Prozesse wie z.B. turbulente Strömungen dienen. Ebenso vermutet man, daß einige der eindimensionalen Zellautomaten genau wie das Game of Life universale Rechner sind.

Die Zellen (hier 320 pro Zeile) können die Werte 0, 1, 2, und 3 annehmen und werden auf dem Bildschirm durch einen Punkt in der entsprechenden

PEN-Farbe dargestellt. Um die nächste Generation zu berechnen, sind jeweils drei nebeneinander liegende Werte zu addieren. Dem Ergebnis wird in einer Zuweisungstafel eine Zahl zugeordnet, die den Zustand der mittleren Zelle in der nächsten Generation bestimmt. Dazu ein Beispiel:

Summe der drei Zellenwerte:	0	1	2	3	4	5	6	7	8	9
Neuer Wert der mittleren Zelle:	0	3	1	0	2	0	2	3	1	3

Das folgende Bild zeigt, wie sich diese Zuweisungstafel grafisch auswirkt. Das angegebene Beispiel ist der Zellautomat, der nach dem Start des Programms zu Verfügung steht. Da die obere Zeile der Tabelle immer gleich bleibt, wird durch die Angabe der unteren Zeile der Zellautomat vollständig definiert.



Und jetzt einige Hinweise zur Bedienung: Nach dem Start des Programms ZAUTOMAT.BAS erscheint am oberen Bildrand ein Pfeil, am unteren Bildrand die X-Position dieses Pfeils, und der Computer erwartet die Eingabe der Anfangspunkte. Wird die COPY-Taste gedrückt, so erscheint direkt über dem Pfeil in der obersten Bildschirmzeile ein Punkt in der Farbe von PEN 1. Durch mehrmaliges Drücken erhält er dann die Farbe 2, 3 oder wird gelöscht. Der Pfeil selbst wird durch die Cursortasten links bzw. rechts gesteuert. Nach der Eingabe der Anfangspunkte können Sie den Zellautomaten mit

der RETURN-Taste starten. Der Aufbau des Bildes läßt sich jederzeit durch das Drücken einer beliebigen Taste unterbrechen, und Sie gelangen in ein kleines Menue. Hier stehen die folgenden drei Punkte zur Verfügung:

W)eiter: Die Ausgabe eines begonnenen Bildes wird fortgesetzt bzw. die Ausgabe der nächsten 10 Generationen begonnen.

N)eustart: Hier gelangen Sie wieder in den Eingabemodus für die Anfangspunkte.

U)mdefinieren: In diesem Unterpunkt können Sie ein neues Logikmuster definieren, indem Sie wie bereits erläutert die untere Zeile der Zuweisungstafel ohne Kommas und Zwischenzeichen eingeben. Die folgende Tabelle zeigt noch einige interessante logische Muster zum Ausprobieren. Mit der selbst zu definierenden Zuweisungstafel und den Anfangsbedingungen sind geradezu unerschöpfliche Experimentiermöglichkeiten gegeben

Summe der drei Zellenwerte:	0	1	2	3	4	5	6	7	8	9
Neuer Wert der	0	2	3	0	1	0	3	2	3	1
mittleren Zelle:	0	2	3	0	2	1	3	2	3	1
	0	2	1	1	3	3	2	2	0	3
	0	2	1	1	0	3	2	0	1	3
	0	1	2	1	3	0	2	3	2	2
	0	1	2	1	3	1	0	0	0	0

Neue Klänge mit dem CPC

Nicht so groß wie erhofft war leider die Resonanz auf die im Rahmen des Software-Experiments gegebene Anregung, Methoden und Programme zu erfinden, die mathematische Strukturen auf dem CPC nicht nur sichtbar, sondern auch hörbar machen. Das ist eigentlich verwunderlich, da schon die Grundlagen der Musiktheorie voller Mathematik sind. Herr Winkelmann aus Hannover zeigt in dem folgenden Beitrag, wie sich mit dem Computer auf mathematische Weise neue Wege der Klangerzeugung beschreiten lassen, für die konventionelle Musikinstrumente gar nicht oder nur eingeschränkt benutzbar sind:

Die gewohnte (europäische) Musik teilt eine Oktave in zwölf Schritte, die sogenannten Halbtonschritte auf. Daraus ergibt sich die Frage, ob man eine Oktave nicht auch in eine andere Anzahl von Schritten einteilen kann. Um diese Idee zu realisieren, muß man jedoch zunächst einmal wissen, mit welcher mathematischen Gesetzmäßigkeit sich die gewohnten Tonschritte berechnen lassen. Bei der Oktave ist das noch relativ einfach; sie entspricht der doppelten Frequenz, somit muß also die Tonperiode im Soundbefehl (zweiter Parameter) halbiert werden. Dazu ein kurzes Beispielprogramm:

```
100 INPUT "Grundperiode:",grund
110 oktave=grund/2
120 REM Grundperiode spielen
130 SOUND 7,grund,200,7
140 REM Oktave spielen
150 SOUND 7,oktave,200,7
160 REM Beide gleichzeitig
170 SOUND 1,grund,300,7
180 SOUND 2,oktave,300,7
```

Bei den Halbtonschritten ist die Berechnung auch nicht viel schwieriger, wenn man weiß, daß bei temperierter Stimmung zwischen den Frequenzen eine einfache mathematische Beziehung besteht:

Grundton	$440 \cdot 2^{0/12}$ Hertz
kleine Sekunde	$440 \cdot 2^{1/12}$ Hertz
große Sekunde	$440 \cdot 2^{2/12}$ Hertz
kleine Terz	$440 \cdot 2^{3/12}$ Hertz

...und so weiter. Der Zähler im Exponenten entspricht also der Anzahl der Halbtonschritte. Damit wäre das Problem gelöst, und man kann ein kurzes Programm schreiben, das die Halbtonschritte hörbar macht. Dabei ist jedoch zu beachten, daß zwischen den Tonperioden im SOUND-Befehl des CPC und den Tonfrequenzen ein reziprokes Verhältnis besteht. Man darf deshalb nicht mit den Potenzen von zwei multiplizieren, sondern muß durch diese dividieren, um eine aufsteigende Tonfolge zu erhalten. Diese Überlegungen führen zu folgendem Programm:

```
100 grundton = 284:REM Kammerton a
110 FOR halb = 0 TO 12
120   leiterton= grundton/2^(halb/12)
130   Sound 7,leiterton,100,7
140 NEXT halb
```

Um nun aber die Oktave in eine andere Anzahl Töne als die gewohnten zwölf zu unterteilen, ersetzt man die 12 in den Zeilen 110 und 120 einfach durch die gewünschte Anzahl. So kommt man zu dem folgenden Programm:

```
100 INPUT "Unterteilung:",anz.ton
110 grundton = 284
120 FOR schritt = 0 TO anz.ton
130   leiterton = grundton /2^(schritt/anz.ton)
140   SOUND 7,leiterton,100,7
150 NEXT schritt
```

Mit Hilfe dieser Anregung sollte es jedem möglich sein, sich eine Tabelle mit ungewöhnlichen Tonschritten zu berechnen und damit zu experimentieren. Besonders interessant ist die Einteilung der Oktave in 24 Schritte, weil man einerseits weiterhin die gewohnten Halbtöne zur Verfügung hat, andererseits aber auch dazwischenliegende Vierteltöne.

Das wohltemperierte Klavier

Neben der heute bei Instrumenten wie Klavier oder Gitarre üblichen temperierten Stimmung, die eine Oktave in zwölf gleiche Halbtonschritte unterteilt, gibt es bei einigen Instrumenten wie z.B. der Geige immer noch die sogenannte reine Stimmung, bei der die Frequenzen der Töne bestimmte ganzzahlige Verhältnisse ergeben. Diese Art der Stimmung hat zur Folge, daß ein Geiger den Ton *Es* anders greift als den Ton *Dis*, obwohl dies in der temperierten Stimmung die gleichen Töne sind. Die heute übliche temperierte Stimmung ist eigentlich nur eine gute und praktische Annäherung an die ursprüngliche reine Stimmung; sie hat es zum Beispiel erst möglich gemacht, ein Instrument wie das Klavier zu konstruieren, auf dem sich alle Tonarten gleichberechtigt benutzen lassen.

Jedoch bietet sich auch die reine Stimmung für Experimente an. Als Einstieg soll das Programm *Frequenzverhältnisse* (**SOUND1.BAS**) dienen, das

die Verhältnisse zwischen den Frequenzen hörbar macht. Nach Eingabe einer Grundfrequenz und eines Verhältnisses wird die zweite Frequenz berechnet. Die beiden Töne werden zunächst nacheinander und darauffolgend im Zusammenklang gespielt.

Beim Experimentieren mit diesem kurzen Programm stellt man schnell fest, daß einfache Zahlenverhältnisse Konsonanzen (harmonisch zusammenklingende Töne) ergeben; je einfacher das Verhältnis, desto schöner der Klang. Und tatsächlich ergeben sich in reiner Stimmung für die gebräuchlichen Konsonanzen folgende Frequenzverhältnisse:

Prime	1:1	Große Terz	5:4
Oktave	2:1	Kleine Terz	6:5
Quinte	3:2	Große Sexte	5:3
Quarte	4:3	Kleine Sexte	8:5

Der Grad der Konsonanz nimmt hier von oben nach unten ab. Im Mittelalter wurden nur solche Intervalle als konsonant empfunden, bei denen Zähler und Nenner des Verhältnisses zwischen 1 und 4 lagen, also die Prime, die Quarte, die Quinte und die Oktave. Inzwischen werden auch die anderen vier aufgeführten Intervalle als Konsonanzen wahrgenommen. Durch die häufige Verwendung von Septakkorden und ähnlichen Dissonanzen in der heutigen Musik wird das Konsonanzempfinden wahrscheinlich weiter abflachen und der Unterschied zwischen Konsonanz und Dissonanz in Zukunft immer mehr verwischt werden.

Auch hier lassen sich mit dem Computer prinzipiell neue Wege beschreiten. So könnte man zum Beispiel mit der Unterteilung der Oktave in 24 Töne einen Dreiklang aufbauen, der zwischen Dur und Moll liegt. Beim Dur-Dreiklang liegt auf einem Grundton eine große Terz und eine reine Quinte (z.B. C-E-G), beim Moll-Dreiklang hingegen liegt auf dem Grundton eine kleine Terz und eine reine Quinte (z.B. D-F-A). Wenn man nun einen Dreiklang aufbaut aus einem Grundton, einer reinen Quinte und dem Ton zwischen kleiner und großer Terz, so erhält man einen Akkord, der zwischen Dur und Moll liegt. Hörbar wird er durch das Programm **SOUND2.BAS**. Hier liegt sicherlich noch ein weites Feld für eigene Experimente!

Pascal läßt grüßen

Nach diesem Abstecher in die Welt der Klänge zurück zur Grafik: Der folgende Beitrag aus eigener Feder hätte hervorragend in die Serie *Das Software-Experiment* hineingepaßt, entstand aber leider zu spät. Er wurde anstattdessen in der Ausgabe 10/87 der Zeitschrift PASCAL International veröffentlicht und demonstriert, wie sich mit erstaunlich einfachen Mitteln verborgene Strukturen im Pascalschen Dreieck sichtbar machen lassen. Hier soll nun das Versäumte mit der entsprechenden Basic-Version (**TRIANGLE.BAS**) nachgeholt werden, womit wir wieder an das Thema des ersten Kapitels anknüpfen und sich der Kreis schließt.

Am Anfang war das Nichts: Ein unendliches Zahlenfeld aus lauter Nullen, vollkommen homogen, in jeder Hinsicht symmetrisch.

```
0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0
```

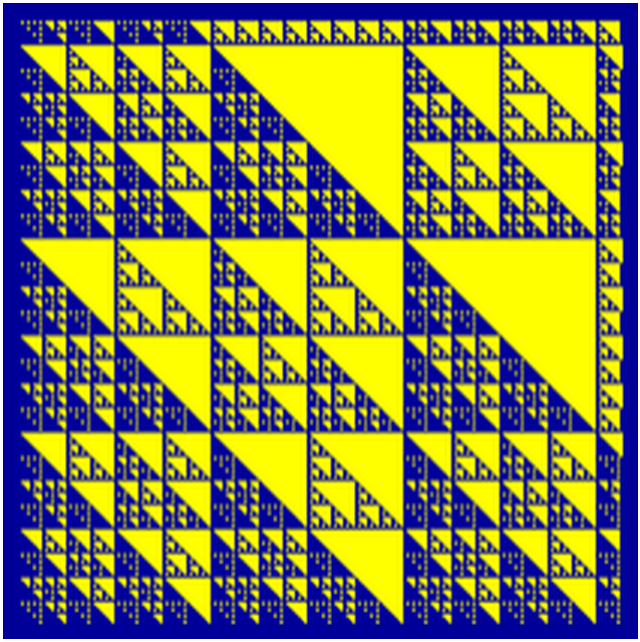
Doch hinter dieser monotonen Gleichförmigkeit verbarg sich ein Gesetz, das alle Zahlen zueinander in Beziehung setzte, ein unsichtbares Kraftfeld:

Jede Zahl ist gleich der Summe der links und rechts darüberliegenden Zahl,

und die Nullen gehorchten dem Gesetz auf absolut nichtssagende Weise, ohne ihm Ausdruck verleihen zu können. Doch genau wie unser physikalischer Raum seine Eigenschaften erst dann offenbart, wenn er Materie enthält, so zeigte das Zahlenfeld eines Tages seinen wahren Charakter, als sich unvermittelt eine kleine unscheinbare EINS dorthin verirrte, die grenzenlose Harmonie des sich selbst Entsprechenden mit einem Schlag durchbrach und quasi als Kondensationskeim eine Lawine von Zahlen auslöste, die sich unaufhaltsam bis in alle Unendlichkeit über die Ebene ergoß:

```
0   0   0   0     1     0   0   0   0
  0   0   0     1     1     0   0   0
0   0   0   1     2     1   0   0   0
  0   0   1     3     3     1   0   0
0   0   1   4     6     4   1   0   0
  0   1   5    10    10     5   1   0
0   1   6   15    20    15   6   1   0
```

Auf einmal gab es größer und kleiner, gerade und ungerade, rechts und links, Richtung und Gegenrichtung… kurz gesagt, die Ebene hatte eine konkrete Struktur erhalten. Und diese Struktur ist dem Mathematiker wohlbekannt: Es handelt sich um das Pascalsche Zahlendreieck, das sich insbesondere nützlich erweist, um die Koeffizienten eines binomischen Ausdrucks der Form $(a+b)^n$ zu berechnen. Doch Zahlentabellen und Formeln können uns kaum vermitteln, was bei diesem mathematischen Schöpfungsakt wirklich entstanden ist, genausowenig, wie ein Notenblatt die Musik zum Erklingen bringt, die sie beschreibt. Obwohl die darstellende Mathematik eine recht alte Disziplin ist, hat sie eigentlich erst mit dem Computer ihr Musikinstrument erhalten, das mathematische Strukturen sicht- und hörbar macht und uns wie ein Mikroskop Einblicke in Welten verschafft, die uns sonst verschlossen blieben.



Trotzdem ist der programmtechnische Aufwand zur Erzeugung solcher Bilder oft erstaunlich gering. Das Programm **TRIANGLE.BAS** stellt im Prinzip nur fest, welche Zahlen des Pascalschen Dreiecks durch einen vorgegebenen Wert teilbar sind und markiert sie auf dem Bildschirm, durch einen Punkt – und das ist schon genug, um die in der Struktur verborgene Ordnung und Harmonie sichtbar zu machen. Um den Bildschirm gut auszunutzen, wird das Zahlenfeld um 45 Grad gekippt und auf einen quadratischen Ausschnitt begrenzt. Da die Zahlen des Dreiecks ziemlich schnell anwachsen und den Rechenbereich und die Genauigkeit der CPC-internen Zahlendarstellung überfordern, rechnet das Programm nur mit dem durch MOD ermittelten Divisionsrest weiter, wodurch sich das grafische Ergebnis jedoch nicht ändert.

Nach dem Programmstart brauchen Sie nur einen ganzzahligen Wert größer 1 einzugeben und RETURN zu drücken, worauf der CPC dann das dazugehörige Bild berechnet. Wenn es fertig ist, wartet der Rechner auf einen beliebigen Tastendruck, um danach einen neuen Teiler in Empfang zu nehmen.

Wer mit dem Programm etwas experimentieren möchte, kann versuchen, dem Divisionsrest in der Variablen r2(spalte) verschiedene Farben zuzuordnen. Die Zeile 220 könnte dann beispielsweise so aussehen (MOD 4 begrenzt nur den Zahlenbereich auf gültige PEN-Nummern):

```
220 PLOT spalte*2+120,Zeile*2,r2(spalte) MOD 4
```

Der Algorithmus entstammt dem Buch *Kunst! Computer?* von KarlHeinz Koch (tewi-Verlag), das neben vielen philosophischen Bezügen und kritischen Anmerkungen zur Computerkunst noch zahlreiche weitere Experimente dieser Art enthält. Zwar sind die meisten Programm in Atari-Basic (800 XL) und für die Ausgabe auf einem Matrixdrucker geschrieben. Die ausführliche Dokumentation erleichtert es jedoch, die angewendeten Verfahren auf den CPC zu übertragen.

Anhang

1. Die Dateien des Software-Experiments

Die folgende Liste gibt darüber Auskunft, in welchem Kapitel die Dateien auf der Software-Experiment-Diskette beschrieben werden und welche Bedeutung sie haben:

Dateiname	Kapitel	Bedeutung
KOMFOL.BAS	1	Demonstriert komplexe Zahlenfolgen
KOMGRA.BAS		Erzeugt komplexe Grafiken (Fraktale)
REPRO.BAS	2	Erzeugt mehrfarbige Reproduktionsmuster
LIFE.BAS		Game of Life, Beispiel für Zellautomaten
OEKOEXP.BAS	3	Hauptprogramm für das Ökologie-Experiment
OEKOERW1 (*)		Erweiterungen zum Öko-Experiment (MERGE)
OEKOERW2 (*)		
OEKOERW3 (*)		
GOBANG.BAS	4	Programm für das Gobang-Brettspiel
KALAHA.BAS	5	Programm für das Kalaha-Bohnenspiel
ZUGCOMV1 (*)		Experimentalversionen der Zugberechnung in Basic (MERGE)
ZUGCOMV2 (*)		
MSCHACH.BAS	6	Lernfähiges Mini-Schachprogramm
BRUDER (*)		Modifikation für Mini-Schach (MERGE)
WMASTER.BAS	7	Wordmaster Wortrate-Programm
LISTE4.DAT		Wortlisten für Wordmaster (4 u. 5-buchstabige Hauptwörter)
LISTE5.DAT		

Dateiname	Kapitel	Bedeutung
WKETTE1.BAS	8	Wortkettenprogramm 1. Version
WKETTE2.BAS		Lader für Wortkettenprogramm 2. Version
KETTEV2		Maschinencode, wird von WKETTE2.BAS nachgeladen.
MORETTI.BAS (**)	9	Entwicklungshilfesimulation
ZAUTOMAT.BAS	10	Demonstration eines eindimensionalen Zellautomaten
SOUND1.BAS		Demoprogramme für neue Formen der Klangerzeugung
SOUND2.BAS		
TRIANGLE.BAS		Grafische Struktur des Pascalschen Dreiecks
EMU.BAS	Anh.	Ladeprogramm für den 664/6128-Emulator
EMU.BIN		Maschinencode für den Emulator

(*) Um einen Fehler in der MERGE-Funktion des CPC 464 zu umgehen, sind alle Basic-Programmteile, die nachträglich dazugeladen werden, im ASCII-Format gespeichert. Falls Sie auf dem CPC 464 diese Dateien auf eine andere Diskette kopieren wollen, müssen Sie sie deshalb mit SAVE „Dateiname“,A neu abspeichern. Auf dem CPC 664/6128 hat das Aufzeichnungsformat keine Bedeutung.

(**) Das Programm MORETTI.BAS läuft auf dem CPC 464 nur, wenn vorher der 664/6128-Emulator mit RUN „EMU“ installiert wurde. Um eine einwandfreie Zusammenarbeit mit dem Emulator zu gewährleisten, wurde das Programm im ASCII-Format abgespeichert, woraus eine relativ lange Ladezeit resultiert. Es ist aber möglich, das Programm normal (d.h. ohne die A-Option) neu abzuspeichern, worauf es schneller geladen wird, in dieser Form jedoch nur auf Ihrem Rechnertyp lauffähig ist.

2. Der 664/6128-Emulator für den CPC 464

Der Emulator ermöglicht es, den erweiterten Befehlssatz des Locomotive-Basic 1.1 auch auf dem CPC 464 zu benutzen. Das Programm besteht aus

zwei Teilen: Durch den Start von EMU.BAS wird das Maschinenprogramm EMU.BIN in den Basic-Interpreter des CPC 464 eingehängt und sorgt für die Behandlung der neuen Befehle. **Bevor Sie den Emulator mit RUN „EMU“ installieren, sollten Sie den Rechner mit CTRL-SHIFT-ESC in den Einschaltzustand versetzen!** Nach kurzer Ladezeit erscheint die Meldung *BASIC 1.1 installiert* auf dem Bildschirm. Der Emulator bleibt bis zum nächsten Zurücksetzen oder Ausschalten des Rechners aktiv.

Nahezu alle für den CPC 664/6128 geschriebenen Basic-Programme laufen mit Hilfe des Emulators auf dem CPC 464. Probleme können sich jedoch ergeben, wenn die Programme einen Maschinencode-Anteil enthalten, der den gleichen Speicherbereich wie der Emulator benutzt (ab &A200). Weiterhin löst der Befehl CLEAR INPUT in einem auf dem 664/6128 geschriebenen Programm trotz Emulator auf dem CPC 464 einen Syntax Error aus, bei dem wie gewohnt die entsprechende Zeile auf dem Bildschirm angezeigt wird. Drücken Sie in diesem Fall ENTER, um die Zeile unverändert wieder einzugeben, und starten Sie das Programm neu. Der Emulator sorgt automatisch für eine Anpassung. Dieses Problem tritt nicht auf, wenn das 664/6128-Programm mit SAVE „Name“,A im ASCII-Format abgespeichert wurde.

Hier nun die Liste der neuen Kommandos:

CLEAR INPUT

Löscht den Tastaturpuffer und verhindert, daß während eines Programmablaufs eingegebene Zeichen ungewollt durch INPUT oder INKEY$ übernommen werden.

COPYCHR$(#⟨*window*⟩)

Diese Funktion gibt das Zeichen an der aktuellen Cursorposition im angegebenen Textfenster zurück.

CURSOR ⟨*Schalter1*⟩, ⟨*Schalter2*⟩

Schaltet den Cursor während eines Programmablaufs aus oder ein. Die Wirkung der Schalter beschreibt die folgende Tabelle:

`CURSOR 0,0`	aus/aus bei INPUT
`CURSOR 0,1`	aus/ein bei INPUT
`CURSOR 1,0`	aus/aus bei INPUT
`CURSOR 1,1`	ein/ein bei INPUT

DEC$ ⟨*numerischer Ausdruck*⟩**,** ⟨*Formatstring*⟩

Gibt den Wert des angegebenen numerischen Ausdrucks als formatierte Zeichenkette zurück. Die Angabe des Formats erfolgt wie bei PRINT USING (siehe CPC Anleitungsbuch). Beispiel:

```
a$ = DEC$(PI,"#.###"): PRINT a$
```

Die Funktion ist eigentlich schon auf dem CPC 464 vorhanden, verlangt aber aufgrund eines Fehlers im Interpreter immer zwei offene Klammern nach DEC$.

FILL ⟨*Farbstift*⟩

Füllt eine beliebig umrandete Fläche mit der angegebenen Farbe. Als Umrandung oder Begrenzung gelten nur Linien, die mit der aktuellen Grafikfarbe (GRAPHICS PEN) oder der Füllfarbe gezeichnet wurden. Der Füllvorgang beginnt an der aktuellen Position des Grafikcursors, der also vorher mit MOVE in die Fläche gesetzt werden muß.

FRAME

Wartet auf den Strahlrücklauf des Monitors und erleichtert damit die Programmierung flackerfreier animierter Grafik.

GRAPHICS PAPER ⟨*Farbstift*⟩

Bestimmt die Farbe des Grafikhintergrundes bei Textausgaben auf Grafikcursorposition (nach TAG) und bei Anwendung des MASK-Befehls.

GRAPHICS PEN ⟨*Farbstift*⟩**,** ⟨*Hintergrund*⟩

Wählt die Schreibfarbe und den Hintergrundmodus für Grafik. Der zweite optionale Parameter hat folgende Bedeutung:

0 = normal überschreiben, 1 = transparent.

MASK ⟨*Bitmaske*⟩

Eine Bitmaske ist ein Wert von 0…255, der den Status von jeweils acht benachbarten Bildpunkten beim Zeichnen einer Linie bestimmt. Damit können beliebig unterbrochene beziehungsweise gestrichelte Linien erzeugt werden.

PEN ⟨*#window*⟩**,** ⟨*Farbstift*⟩**,** ⟨*Hintergrund*⟩

Wie beim CPC 464, nur kann jetzt mit dem letzten Parameter der Hintergrundmodus für Textausgaben bestimmt werden (0= normal, 1= transparent). Von den letzten beiden Werten kann wahlweise einer entfallen.

MERGE und CHAIN MERGE „Dateiname“

Diese Kommandos funktionieren jetzt auch bei Diskettenbetrieb einwandfrei.

Parameterübergabe bei CALL und RSX-Befehlen:

Beim CALL-Befehl und bei RSX-Kommandos können Strings direkt ohne den Umweg über den Klammeraffen (@) übergeben werden. Beispiel: |ERA, „TESTPROG.BAK“

Zusätzliche Grafikparameter:

– Alle Grafikbefehle werden auf das Format **KOMMANDO x, y, Farbstift, Schreibmodus** erweitert. Das gilt für die Befehle MOVE, MOVER, PLOT, PLOTR, DRAW, DRAWR. Der optionale Parameter *Schreibmodus* gibt an, wie die neuen Bildpunkte mit dem Hintergrund verknüpft werden:

0 = Überschreiben, 1 = XOR-Modus, 2 = AND-Modus, 3 = OR-Modus