Bartel · Kraas
Schrüfer

Das grosse

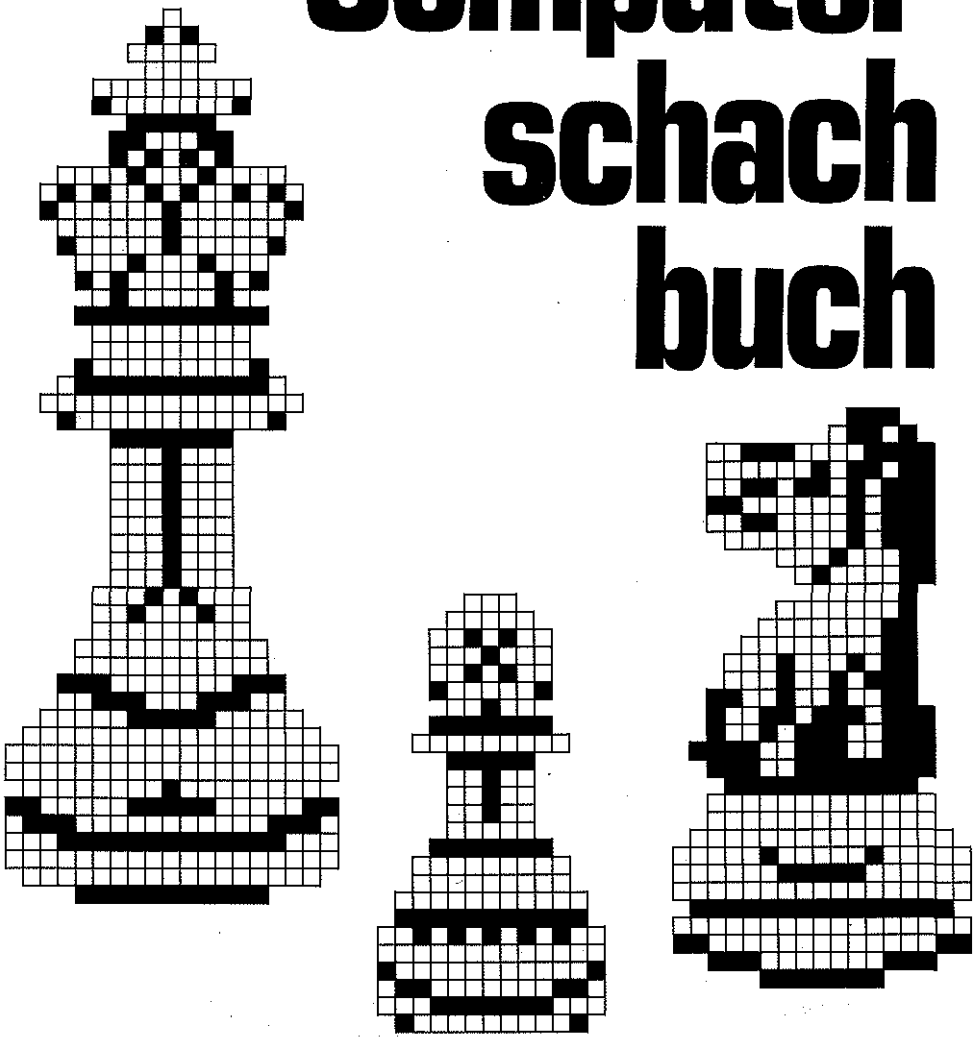
Computer schach buch

EIN DATA BECKER BUCH

Bartel · Kraas
Schrüfer

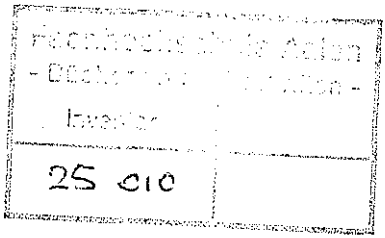
Das grosse

Computer schach buch



EIN DATA BECKER BUCH

ISBN 3-89011-117-3



Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Inhaltsverzeichnis

0. Vorwort

Schach ist das Spiel der Könige. Und Schach ist der König unter den Spielen. So sagt man. Woran liegt das? Nach der mathematischen Spieltheorie ist es nämlich nicht mehr als ein "endliches Zwei-Personen-Nullsummenspiel mit vollständiger Information"; also ein ziemlich einfacher Fall. In der Praxis ist Schach jedoch eine unendlich vielfältige Mischung aus Kunst, Logik, Psychologie, Kampf und Kombination. Millionen Menschen spielen es, Millionen beschäftigen sich mit Schach und Hundertausende besitzen einen Schachcomputer oder spielen Schach auf ihrem Mikrocomputer.

Dieses Buch wird Sie also mit den elektronisch-kybernetischen Aspekten des Schachspiels vertraut machen. Sie werden lernen, wie Schach überhaupt auf den Computer kam. Wir werden Ihnen die Theoretiker und Praktiker des Computerschachs vorstellen und versuchen, Ihnen die Grundlagen der Strategiespiele zu vermitteln.

Die gelernten Begriffe, Methoden und Algorithmen können Sie dann praktisch an einem Schachprogramm in BASIC nachvollziehen. Dabei steht nicht im Vordergrund, Ihnen ein superschnelles und extrem spielstarkes Programm zum Abtippen anzubieten - das wäre in BASIC gar nicht möglich - sondern zu zeigen, wie die Mehrzahl der kommerziellen und ein Großteil der experimentellen Programme arbeitet. Deshalb nimmt die Bildschirmausgabe der Such- und Bewertungsfunktionen auch einen überproportional großen Raum ein.

Vermutlich werden Sie nach der Beschäftigung mit unserem Programm schon so viel gelernt haben, daß Ihnen Ihr Schachcomputer bzw. Ihr gekauftes Schachprogramm kaum noch Rätsel aufgibt. Trotzdem wollen wir Ihnen auch noch zeigen, wie Sie mehr mit Computerschach anfangen können.

Wenn Sie wollen, können Sie erfahren, wie Sie jedes Programm in Schwierigkeiten bringen oder sogar nach Belieben schlagen

können - ganz gleich auf welcher Stufe Sie spielen. Sind Sie weniger vom Gewinnen-Wollen beseelt, finden Sie immerhin noch etliche Tips, die Ihnen helfen, beim Spiel mit dem Computer mehr zu lernen.

Wenn Ihnen im Verlauf der Lektüre unbekannte Begriffe und Namen von Leuten, die Sie nicht kennen, begegnen, können Sie im umfangreichen Anhang nachschlagen. Was Sie dort nicht finden, entnehmen Sie dann den Büchern, die wir im Literaturverzeichnis anbieten.

Wenn Sie 'Das große Computerschach-Buch' durchgearbeitet haben, werden Sie viel mehr über Schach wissen, Ihre Spielstärke verbessern können und jedem Schachprogramm mit reichlich Selbstvertrauen entgegentreten.

Wir hoffen, daß neben diesen Nutzeffekten auch noch eine Menge Spaß beim Lesen entsteht und daß Sie uns mit Kritik, Verbesserungsvorschlägen und Anregungen helfen werden, dieses Buch ständig weiterzuverbessern.

Düsseldorf, Salzgitter, Lehrte im November 1985

Rainer Bartel, Hans-Joachim Kraas, Günther Schrüfer

Anmerkungen und Danksagungen:

Das Programm DEMOSCHACH wurde von Hans-Joachim Kraas und Günther Schrüfer ursprünglich auf einem Commodore SX-64 geschrieben und mit BASIC 64 kompiliert. Auf den IBM PC wurde es von Rainer Bartel gebracht und dort mit dem Microsoft QuickBASIC-Compiler kompiliert. Die Anpassung an die Schneider CPC-Rechner führte Helmut Retzlaff durch (Dankeschön!).

Alle Schachdiagramme und ein Teil der übrigen Abbildungen wurden von Dirk Schaun (Vielen Dank!) mit dem Programm ProfiPainter auf dem Schneider CPC 464 erstellt und mit dem Schneider-Drucker NLQ ausgedruckt.

Unser besonderer Dank gilt dem Team vom DATA BECKER-Lektorat (Roland Heine, Helmut Retzlaff, Jürgen Steigers, Thomas Vervost, Claus Wagner und - last but not least - Brigitte Witzer), das aus einem undurchsichtigen Haufen Manuskriptblätter erst dieses Buch machte, Cäcilia Jordan, die den Buchtitel gestaltete und allen Menschen, die bei der Produktion des Buches ihre Hände im Spiel hatten.

Am innigsten müssen wir uns bei unseren Frauen und Kindern bedanken, die uns über vier Monate nur noch als mehr oder weniger entnervte Autoren erlebten und uns trotzdem nach Kräften unterstützten.

1. Wie kam Schach auf die Maschine? - Geschichte des Computerschach

Bevor wir auch nur ein Wort über die Frage verlieren können, wie Schach auf die Maschine kam, müssen wir uns erst einmal mit dem Problem beschäftigen, wie Schach in die Köpfe der Menschen kam. Dazu brauchen wir glücklicherweise nicht die ganze Geschichte des Spiels durchwandern - bis zum Ende des 18. Jahrhunderts gab es so gut wie keine Theorie, kaum Ansätze von Systematik, und nur sehr wenige Partien aus dieser Zeit sind überliefert. Zwar hatte bereits im 17. Jahrhundert der spanische Mönch Ruy Lopez eine umfangreiche Analyse von Eröffnungen geschrieben (die Spanische Eröffnung heißt ihm Angelsächsischen im zu Ehren Ruy Lopez), doch war dieses Werk von eher akademischem Interesse für Adel und Klerus, auf die das Schachspiel seinerzeit beschränkt war.

Michail Botwinnik, Schachweltmeister (mit Unterbrechungen) von 1948 bis 1963, datiert den Beginn des modernen Schach mit dem Gewinn der Weltmeisterschaft durch Wilhelm Steinitz gegen Hermann Zuckertort 1871. Steinitz meinte, im Schach müsse wie in der Natur alles eine Ursache haben, es sei notwendig, seinem Spiel bestimmte, definierbare Prinzipien zugrunde zu legen.

Erst mit diesem Wandel zur Systematik, dessen Ergebnisse sich bis heute in einem Gebirge von Schachliteratur niederschlagen, gewann Schach eine faßbare Gestalt. Ob Eröffnung, Mittel- oder Endspiel: jeder Teil einer Schachpartie ist heutzutage gründlich analysiert und systematisiert.

Wer sich zum Spitzenspieler, zum Großmeister oder gar Weltmeister aufschwingen will, kommt ohne eine permanente Beschäftigung mit 'der Theorie', wie Schachspieler es ausdrücken, nicht aus. Vorbei sind die Zeiten, als wilde und legendenumwobene Charaktere wie La Bourdonnais, Philidor

oder MacDonnell den inoffiziellen Titel eines Weltmeisters unter sich ausmachten. Aber noch nicht angebrochen sind die Zeiten, in denen ein Computerprogramm nach der Krone greift.

1.1 Die ersten Schachautomaten - Türke, Mephisto und andere

Nehmen wir es vorweg: die in diesem Kapitelchen besprochenen Maschinen sind nicht die Vorläufer des Computerschachs. Trotzdem ist es ganz amüsant zu wissen, wie alt die Idee der Schachautomaten eigentlich ist.

Am Ende des 18. Jahrhunderts war die Feinmechanik ein beliebtes Amüsement für gelangweilte Könige und ihren Hofstaat. Die Mechaniker selber waren oft adlig und betrieben ihr Handwerk als 'Hobby', wie wir es heute nennen würden. So z.B. der sächsische Baron von Kempelen, der so nützliche Dinge erfunden hatte wie einen Tisch, der sich automatisch selbst deckte; das Geschirr war an der Unterseite der Tischplatte befestigt und wurde auf Knopfdruck durch Federkraft an die Oberseite geschwenkt. Auch einen Automaten, der mit der Feder in einer wunderschönen Handschrift Sonette zu Papier brachte, stammte aus seiner Werkstatt. Dabei meinte es Kempelen durchaus ernst mit diesen Automaten, sein 'Türke', der sich letztendlich als 'getürkt' herausstellte, zählte für ihn eher zu den unwichtigen Erfindungen. Zu seinem Unglück war nur diese Schachmaschine für ihn so etwas wie ein kommerzieller Erfolg.

1770 präsentierte er dem staunenden Wiener Hof diesen Schachautomaten, den er den 'Türken' nannte. Bis heute ist nicht geklärt, ob diese Erfindung wirklich von Kempelen selber stammte oder nicht vielleicht auf weit ältere Vorbilder aus dem orientalischen Raum zurückging. Es gibt Theorien, die besagen, der Ur-'Türke', den Kempelen von einem Landfahrer gekauft habe, sei tatsächlich (ohne Tricks) in der Lage gewesen, eine Partie Schach zu spielen, der Verkäufer habe aber, aus Ärger über den geringen Preis, den er bekam, die Mechanik weit-

gehend zerstört, so daß Kempelen nichts anderes übrigblieb, als sich eines Tricks zu bedienen.

Jedenfalls arbeitete der geniale Baron mit einer Handvoll Tricks, wie sie heute noch die Illusionisten verwenden. Der Automat bestand aus einer ziemlich großen Kiste (etwa anderthalb Meter im Quadrat und 90 Zentimeter hoch) mit einem Schachbrett auf der Oberseite, an der eine orientalische gekleidete Figur saß. Vor jeder Vorstellung überzeugte von Kempelen das Publikum davon, daß sich außer allerhand Zahnrädern und Federwerken nichts in der Kiste befand, in dem er hier und da eine Klappe öffnete, Teile der Verkleidung hochschwenkte und so tat, als sei alles wirklich nur Mechanik.

In Wirklichkeit saß ein nicht zu groß geratener menschlicher Spieler im Gehäuse, der seine Gegner reihenweise schlug. Leider ist der Name dieses versteckten Meisters nicht überliefert, daß es sich aber um einen Zwerg gehandelt haben soll, wie üblicherweise behauptet wird, gehört ins Reich der Fabel, denn in späteren Jahren tat sich unter anderem der Schachmeister Allgaier als versteckter Spieler hervor; und von diesem ist nicht bekannt, daß er kleinwüchsig gewesen sei.

Friedrich der Große war der erste Käufer des 'Türken', er bezahlte eine für damalige Verhältnisse horrende Summe für den 'Automaten', spielte allerdings nur in Anwesenheit des Erfinders einige Partien und verlor, als er den Schwindel entdeckt hatte, das Interesse an der Kiste, die danach zwanzig Jahre in irgendeinem Abstellraum verstaubte.

Die Wahrheit über den 'Türken' hatte sich jedoch nicht sehr verbreitet, im Gegenteil: die Legende gehörte sozusagen zum Standardrepertoire aller Schachspieler der damaligen Zeit. Auch Napoleon, der sich selbst für einen passablen Spieler hielt, aber auch fast nur gegen seine Geliebten und Untergebenen antrat, ließ sich nach der Eroberung Berlins 1806 durch die französische Armee den Automaten vorführen. 1809 kam es zu einer Partie Napoleons gegen den 'Türken', in dessen Innerem sich der erwähnte Meister Allgaier befand - natürlich verlor der große Korse gegen einen der größten Spieler seiner Zeit.

Danach ging die immerhin schon vierzig Jahre alte Maschine auf eine Tournee durch Europa, die ein großer kommerzieller Erfolg wurde; bis 1837 konnten die Menschen in allen großen Städten Europas sehen, wie ein mechanischer Spieler die jeweiligen Lokalmatadoren zur Strecke brachte. Später wurde der 'Türke' im Museum ausgestellt und fiel 1854 einem Feuer zum Opfer.

Schon 1789 hatte der clevere Baron erste Nachahmer gefunden, die aber Maschinen produzierten, die weit weniger raffiniert waren als der 'Türke'. Man bedenke, daß die Puppe ja tatsächlich die Figuren mechanisch gesteuert führte und daß die Übermittlung der gegnerischen Züge ins Innere ebenfalls mechanisch stattfand. Ein Wunderwerk der damals möglichen Technik war der 'Türke' also auf jeden Fall.

Zwei andere berühmte Vertreter dieser ersten Generation von Schachautomaten waren 'Ajeeb' und 'Mephisto', die Mitte des 19. Jahrhunderts in England entstanden und beide mehr oder weniger geschickte Kopien des 'Türken' waren; 'Mephisto' war weniger erfolgreich und wurde schon nach ein paar Jahren wieder verschrottet, während man 'Ajeeb' noch 1929 in London bewundern konnte.

Baron von Kempelens Schachautomat hat die Phantasie vieler Schriftsteller angeregt und auch Spekulationen über die unbegrenzten Möglichkeiten der Technik (im 19. Jahrhundert, wohlgemerkt!) ausgelöst, das alles ist vergangen; geblieben sind die Spuren, die diese Maschine in der deutschen Sprache hinterlassen hat: Von einer Zeitungsmeldung, die eine erfundene Geschichte wiedergibt, sagt man, sie sei - getürkt.

Weit ernsthafter, dafür aber auch unscheinbarer, waren die Bemühungen des spanischen Mathematikers Torres y Quevedo, der um 1890 mit einem mechanischen Schachautomaten experimentierte. Dieser Apparat - den übrigens keiner, der über ihn geschrieben hat, je zu Gesicht bekam - soll schon ein wenig Ähnlichkeit mit einem modernen Schachcomputer gehabt haben; zumindest besaß er ein Schachbrett, auf dem das Ausgangs- und das Zielfeld der zu ziehenden Figur angezeigt wurde. Der Haken

an der Sache: Torres' Maschine war lediglich fähig, mit den weißen Figuren ein König-Turm-Endspiel gegen den schwarzen Monarchen siegreich zu bestreiten, allerdings nicht immer auf dem kürzesten Wege.

Diese erste wirkliche Schachmaschine zeigt merkwürdige Parallelen zur Entwicklung der Computer auf: Etwa zur gleichen Zeit wie Torres arbeitete das sagenhafte Mathematik-Genie Charles Babbage, freundlich betreut von der nicht minder legendären Lady Ada Lovelace, an seiner mechanischen Rechenmaschine, die - obwohl nie vollendet - schon eine ganze Menge Grundbegriffe der digitalen Datenverarbeitung vorwegnahm.

Nach Torres' Erfindung war erst einmal 60 Jahre lang Ruhe in Sachen Schachmaschinen. Erst mit der Erfindung der elektronischen, digitalen Rechenmaschine durch Konrad Zuse (um 1941) begannen die Spekulationen über nichtmenschliche Schachspieler erneut.

1.2 Die Theoretiker des Computerschach - von Neumann, Shannon, Turing

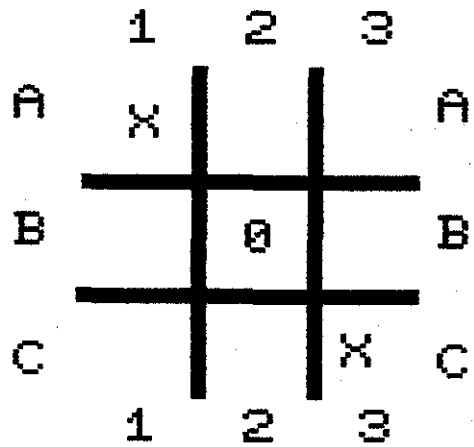
Konrad Zuse, über dessen gewaltigen Anteil an der Entwicklung des Computers sich insbesondere die englischsprachige Literatur gerne ausschweigt, hatte 1941 mit dem Z3 die erste einsatzfähige, programmgesteuerte Rechenanlage gebaut und - schon in der Planungsphase - über die Möglichkeiten eines Schachprogrammes spekuliert und sich bereits Gedanken über die notwendigen Algorithmen gemacht. Zwar blieb Zuse auch nach dem Krieg Praktiker auf dem Gebiet der Rechner, ein Schachprogramm hat er jedoch nie geschrieben.

a) Spieltheorie

Ur-Väter aller Schachalgorithmen sind jedoch die Mathematiker von Neumann und Morgenstern, die 1944 ihre Spieltheorie vor-

legten. Diese besagt zunächst, daß man alle denkbaren Spiele klassifizieren kann und zwar nach ihren mathematischen Gegebenheiten. Das Schachspiel gehört dabei zu den bereits erwähnten "endlichen Zwei-Personen-Nullsummenspielen mit vollständiger Information". Ein anderes, leichter verständliches Spiel, das zu derselben Kategorie gehört, ist das berühmte "Tic-Tac-Toe".

Zwei Spieler (daher ein Zwei-Personen-Spiel!) setzen auf einem drei mal drei Felder großen Spielbrett abwechselnd ihnen zugeordnete Steine. Nach maximal neun (3 * 3 = 9) Zügen ist das Spiel beendet; es ist also endlich (im Gegensatz zu mancher Skatrunde z.B., die sich über Tage hinziehen kann). Gewonnen hat derjenige Spieler, dem es gelingt, eine senkrechte, waagerechte oder diagonale Dreierreihe mit seinen Steinen zu besetzen. Schaffen es beide Spieler nicht, endet das Spiel unentschieden.



Aber was hat es mit der ominösen Nullsumme auf sich? Einfach ausgedrückt: der Gewinner gewinnt das, was der Verlierer verliert, bei Unentschieden gewinnen bzw. verlieren beide dasselbe. Dazu müssen wir uns ansehen, wie ein solches Spiel überhaupt abläuft:

Nr.	Zug
1	B2 / A1
2	C2 / A2
3	A3 / C1
4	B1 / B3
5	C3

Unentschieden!

Wenn wir die verschiedenen Züge nach ihrem Ergebnis bewerten, können wir eine Tabelle aufstellen, die zunächst in jedem Feld den Wert enthält, den wir dem Zug geben. Auf jeden Zug gibt es einen Gegenzug, den wir ebenfalls bewerten. Die Differenz dieser beiden Werte tragen wir in diese Tabelle ein. Ein (fiktives) Beispiel:

Stellung:
```
..X
OX.
O..
```

Matrix aus der Sicht von "X":

X/O	A1	A2	A3	B1	B2	B3	C1	C2	C3	min.
A1	..	-08	..	..	..	22	..	16	20	-08
A2	-74	..	..	..	..	12	..	-02	-02	-74
A3	..	..	..	..	..	..	..	..	..	
B1	..	..	..	..	..	..	..	..	..	
B2	..	..	..	..	..	..	..	..	..	
B3	-80	04	..	..	..	..	..	00	02	-80
C1	..	..	..	..	..	..	..	..	..	
C2	-80	00	..	..	..	04	..	..	-02	-80
C3	-80	-02	..	..	..	-06	..	00	..	-80

Was sagt nun die Matrix aus? Aus jedem Feld ist ersichtlich, welche Strategie bei welcher Gegenstrategie zu welchem Ergebnis führt. In unserem Beispiel sehen wir die Bewertung aus der Position des 'Kreuzers'. Der insgesamt höchstmögliche Wert

in dieser Stellung beträgt 08. Da aber Weiß mit optimalem Gegenspiel von Schwarz rechnen muß, wird es den Zug wählen, der bei bestem Gegenzug den höchsten Wert erreicht oder, anders ausgedrückt, die Zeile nehmen, in der der höchste Minimalwert steht; also A1 oder C3, denn in beiden Fällen kann 'Kreuz' eine Bilanz von 0 erreichen. Alle anderen Züge können von Schwarz so beantwortet werden, daß aus weißer Sicht die Bilanz negativ wird.

b) Minimax-Theorem

Und hier setzt das Theorem von v. Neumann an: Es besagt, daß, wenn Weiß jeweils das Zeilenminimum maximiert und Schwarz das Spaltenmaximum minimiert, beide Spieler das Optimum dessen erreichen, was sie vom Spiel erwarten können.

Das dritte Merkmal des Schach in der Spieltheorie ist der Begriff "mit vollständiger Information". Das bedeutet, daß zu jedem Zeitpunkt jeder Spieler alle Informationen über die möglichen Züge des Gegners hat. Alles klar, könnte man meinen, dann muß man also nur für jede Stellung eine Gewinnmatrix berechnen und schon findet man den besten Zug. Genau das war von Neumanns Auffassung vom Schach: "Problem theoretisch geklärt, Problem uninteressant." Seine Aussage zum Schach war einfach die: "Da es sich um ein endliches Zwei-Personen-Nullsummenspiel mit vollständiger Information handelt, muß kein Spiel mit dem Gewinn eines der beiden Teilnehmer enden."

Daß das Remis eigentlich das natürlichste Ergebnis einer Schachpartie ist, haben uns ja schon so manche Großmeister bei ihren Weltmeisterschaftskämpfen vorgeführt. Andererseits war dieses Minimax-Theorem, wie es die Mathematiker nannten, auch eine große Ermutigung für die Computerwissenschaftler, da es schließlich auch bewies, daß bei vollständiger, korrekter Minimaximierung aller Zugfolgen kein Spieler verlieren muß. Und weil die Computer damals, als sie noch mit Relais oder Röhren betrieben wurden, auch schon sehr genau rechneten, schien klar, daß sie auch erfolgreich Schach spielen könnten.

Leider würde eine simple Gewinnmatrix beim Schach ziemliche Dimensionen annehmen; allein in der Eröffnung stehen Weiß insgesamt 20 Züge zur Verfügung (8 mal Bauerneinzelschritt + 8 mal Bauerndoppelschritt + 4 mal Springerzug). Schwarz hat in seinem ersten Zug die gleiche Anzahl Gegenzüge; macht eine Matrix von 400 Feldern Umfang. Im Mittelspiel hat jede Seite durchschnittlich 38 Zugmöglichkeiten pro Stellung, macht eine Matrix von 1.444 Feldern. Aus jeder Ausgangsstellung entstehen wiederum neue Strategiemöglichkeiten, so daß eine vollständige Minimax-Prozedur über längere Zugfolgen selbst vom Computer in angemessener Zeit nicht mehr zu bewältigen ist.

c) Computerschach wird Wissenschaft

Böse betrachtet, war das Minimax-Theorem der einzige nennenswerte Beitrag der Spieltheoretiker zum Computerschach. Den praktischen Anfang des königlichen Spiels auf Elektronenrechnern kann man - welch seltener Glücksfall! - ziemlich exakt bestimmen: Am 9. März 1949 erschien ein epochaler Artikel in einem Wissenschaftsmagazin: "Programmieren eines Digital-Rechners zum Schachspiel" von Claude Shannon. Dieser Pionier im Computerschach arbeitete zu der Zeit in der Forschungsabteilung von Bell Telephone in New Jersey, die erhebliche Anstrengungen unternommen hatte, die Computerwissenschaft für ihre Zwecke nutzbar zu machen.

Shannon entwickelte (basierend auf Wieners Kybernetik-Theorien und seinen Kenntnissen der Rechner-Technologie) ein bis heute gültiges Konzept für die Schachprogrammierung. Seine Ideen in die Tat umzusetzen, blieb ihm nicht vergönnt; zu schwach waren die Leistungen der damaligen Computer und obendrein konnten sie nur in reinem Binär-Code programmiert werden - komplexe Programme ließen kaum realisieren.

Shannon ging davon aus, daß Schach ein ideales Versuchsfeld für Experimente mit maschineller Intelligenz sei, da alle denkbaren Ereignisse des Spiels in unzweideutige mathematische

Termen zu fassen seien. Sein erster Ansatz war die Frage: Wie kann der Computer ein Schachbrett darstellen?

Er schlug vor, daß das Brett durch 64 'Worte' zu beschreiben sei, die jeweils 64 Bits umfassen; also ein 'Wort' pro Feld. Jedes Wort kann als eine Art Postfach angesehen werden, daß alle Informationen über den Zustand des Feldes enthält. Shannon hatte außerdem die Idee, Züge von Figuren auf dem Feld dadurch zu erzeugen, daß man jeder Figur eine bestimmte mathematische Operation zuordnet, die - wenn sie ausgeführt wird - als Ergebnis die Koordinaten des Zielfeldes ergibt.

Bei Shannon sah das in der Urfassung so aus:

56	57	58	59	60	61	62	63	(8)
48	49	50	51	52	53	54	55	(7)
40	41	42	43	44	45	46	47	(6)
32	33	34	35	36	37	38	39	(5)
24	25	26	27	28	29	30	31	(4)
16	17	18	19	20	21	22	23	(3)
08	09	10	11	12	13	14	15	(2)
00	01	02	03	04	05	06	07	(1)
A	B	C	D	E	F	G	H	

Ein Bauer hat bei Spielbeginn jeweils zwei mögliche Züge, die sich mit einer einfachen mathematischen Operation ausdrücken lassen:

V = Nummer des Von-Feldes;
N = Nummer des Nach-Feldes.

N1 = V + 10
N2 = V + 20

Auch ein Bauernschlagzug läßt sich leicht darstellen:

N1 = V + 9
N2 = V + 11

Nach dieser Methode kann jeder pseudolegale Zug erzeugt werden, also jeder Zug, mit dem Figuren den Regeln entsprechend bewegt werden.

Moderne Brettdarstellung weicht nur wenig von Shannons Konzept ab; erstens numeriert man die Felder anders und zweitens legt man 'Postfächer' für 64 plus 72 gleich 136 Felder an, indem man um das eigentliche 8-mal-8-Brett eine Doppelreihe von Pseudo-Feldern zieht, die mit Pseudo-Figuren besetzt sind. So erreicht man, daß eine Figur nicht aus dem Brett 'herausspringen' kann. Auch sein Vorschlag, die Figuren durch Zahlenwerte (positive für weiße, negative für schwarze Steine) zu repräsentieren, wird nach wie vor benutzt, auch hier mit Abwandlungen.

Die zweite Frage, die sich Shannon stellte war:
Wie kann der Computer den besten Zug finden?

Und hier leistete Shannon seinen wichtigsten Beitrag zur Theorie des Computerschach: Er klassifizierte alle denkbaren Vorgehensweisen in zwei unterschiedliche Typen:

- Typ A = Brute Force;
- Typ B = selektive Suche.

Brute Force (also rohe Kraft) bedeutet, daß der Computer in jeder Position alle seine legalen Züge erzeugt und bewertet, dann die gegnerischen und wieder seine Gegenzüge, usw. Die Betonung liegt auf alle. Geht man wieder von durchschnittlich 38 legalen Zügen pro Stellung aus, kann man leicht ausrechnen, daß eine Suche über 'nur' 4 Halbzüge (ein Halbzug ist jeweils der Zug einer Seite) schon 2.085.136 Züge betrachten müßte. Diese exponentielle Explosion möglicher Positionen kann heute noch kaum ein Computer bewältigen.

Shannon selbst hat schon Vermutungen angestellt, wie die Zahl der zu berechnenden Züge verkleinert werden kann, diese aber nicht konkretisiert. In der modernen Schachprogrammierung, die überwiegend mit Typ-A-Programmen arbeitet, richtet sich ein

großer Teil der Bemühungen auf das Reduzieren der betrachteten Züge. Aber dazu später mehr.

Erfolgversprechender erschienen Claude Shannon Typ-B-Programme. Hier sollten von vornherein nur sogenannte 'plausible' Züge betrachtet werden. Die Schwierigkeit war - und ist immer noch - die Festlegung, welche Züge denn plausibel seien.

Wichtige Techniken wie der Spielbaum, die Stellungsbewertung und die Zugwahl (per Minimax-Prozedur) hat Shannon bereits geplant und beschrieben.

Ein zweiter wichtiger Vorreiter in Sachen Computerschach war der große britische Mathematiker Alan Turing. Auch ihn interessierte Schach in erster Linie als Versuchsfeld für die Künstliche Intelligenz. Seine Ideen ähneln denen von Claude Shannon sehr; er ging sogar weiter als dieser, indem er ein Programm schrieb, das die notwendigen Techniken enthielt. Da er aber weder die Geduld, noch die konkreten Möglichkeiten besaß, das Programm wirklich auf einem Rechner zu implementieren, ließ er ganze Partien von Hand (!) nach dem Programm berechnen.

d) Turing-Test erfolgreich absolviert?

Übrigens ist Schach das einzige Gebiet, auf dem bisher der Turing-Test erfolgreich bestritten wurde. Dieser Test beruht auf folgendem Postulat:

"Wenn ein Mensch je einen Dialog mit zwei verschiedenen Partner führt und hinterher nicht sagen kann, welcher der Gesprächspartner ein Mensch und welcher ein Computer war, muß man davon ausgehen, daß der Computer über Intelligenz verfügt."

Die vorgeschlagene Testanordnung funktioniert so:
Der Testkandidat sitzt in einem Raum und kann sich per Fernschreiber mit zwei Partnern, die räumlich getrennt und nicht

sichtbar sind, unterhalten. Er stellt den beiden Partnern Fragen, die diese beantworten.

Aus den Antworten soll der Proband herausfinden, welcher seiner Gesprächspartner ein Mensch, welcher ein Computer ist, ohne diese konkrete Frage ("Bist Du ein Computer?") zu stellen.

Bisher ist der Test in der beschriebenen Anordnung nicht gelungen, d.h. kein Programm konnte so eindeutig antworten, daß ein Kandidat eine sichere Zuordnung geben konnte.

Bei einem Simultan-Schachspiel, das der deutsche Großmeister Dr. Helmut Pfleger 1983 in Hamburg durchstand, hatten die Veranstalter drei Schachcomputer eingeschmuggelt, deren Züge über Minifunkgeräte an ihr 'Bedienungspersonal' weitergegeben wurden. Insgesamt wurden 28 Partien gespielt, Dr. Pfleger gewann 22, spielte fünfmal Remis und verlor einmal. Als man ihm das geheime Unternehmen eröffnete und fragte, welcher Gegner seiner Meinung nach ein Computer gewesen sei, mußte er passen. Selbst Experten, denen man die Partie-Notationen vorlegte, konnten nicht eindeutig entscheiden, wo Künstliche Intelligenz im Spiel war; lediglich Garry Kasparow fand auf Anhieb heraus, daß eine der Remis-Partien von einem Computer gespielt sein mußte.

Kehren wir zurück zu Shannons und Turings Tagen; damals waren die Aussichten für ein spielstarkes Computerprogramm noch sehr schlecht. Erst die Versuche, Programme wirklich auf Großrechnern zu implementieren, brachten ein wenig Fortschritt.

1.3 Die Ära der Großrechner - MAC HACK, CHESS, KAISSA

Zunächst mußten sich die schachbegeisterten Programmierer mit den schwachen Leistungen der Rechner auseinandersetzen. Wissenschaftler, die im Atombomben-Zentrum der USA in Los

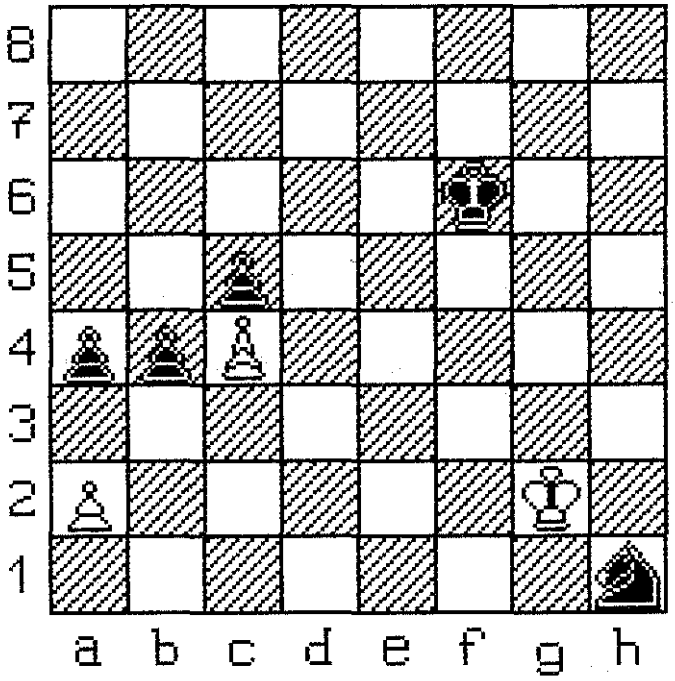
Alamos mit einem Computer namens MANIAC (was soviel heißt wie 'Der Verrückte', ein treffender Name ...) arbeiteten, versuchten Turings Ansätze zu verwirklichen. Leider war der Rechner viel zu langsam; selbst ein kastriertes Schach mit sechs mal sechs Feldern - man spielte ohne Läufer und mit je sechs Bauern - brauchte Stunden, um einen Zug zu finden.

Immerhin konnte auf dem Nachfolger MANIAC 2 ein 'echtes' Schachprogramm installiert werden, das aber erbärmlich schlecht gespielt haben soll. Wer weiß, wie stark Schach auf dem Computer heute schon wäre, wenn die Programmierer mehr Ehrgeiz in Sachen Schach gezeigt hätten.

Das erste Programm, das alle Regeln beherrschte und eine Partie korrekt zu Ende spielen konnte, wurde um 1955 auf einer IBM 704 geschrieben. Edward Lasker, ein Internationaler Schachmeister und Autor zahlreicher Schachbücher, spielte gegen dieses noch namenlose Programm und bescheinigte ihm 'passable Amateurstärke'. Dieses Programm bediente sich - notgedrungen - der Shannon-B-Strategie; es betrachtete in jeder Stellung nur die sogenannten 'plausiblen' Züge und konnte, wegen der Beschränkungen durch den Rechner, auch nur zwei Halbzüge tief blicken.

Herbert Simon von der Carnegie-Mellon Universität in Pittsburgh sorgte mit seinem CP-1 für entscheidende Fortschritte: So war das Programm in einer Hochsprache codiert und erreichte in der Eröffnung und dem Mittelspiel recht gute Erfolge. Die Schwächen lagen - und das ist bei vielen Programmen heute noch der Fall - im Endspiel. Dort beging es ernste Fehler, die ein menschlicher Gegner leicht ausnutzen konnte. Dabei stieß man erstmals auf den sogenannten 'Horizonteffekt'.

Bauen Sie einmal Ihr Schachbrett folgendermaßen auf:



Schwarz gewinnt leicht, wenn der König mit Ke5 zentralisiert wird.
(Fortsetzung: 2. Kh1: Kd4 3. Lg8 Kc3 4. Kg1 b3 und mindestens ein Bauer geht zur Dame)

Ganz falsch wäre natürlich 1. ... b3, denn nach 2. ab3: ab3: 3. Lb3: ... hält Weiß Remis. Komischerweise spielen (fast) alle Schachprogramme auf niedriger Stufe diesen Unglückszug ... b3. Wie kommt das?

Das Programm findet nach 1. ... b3 zwei mögliche, erfolgversprechende Varianten für Weiß:

a) 1. ... b3 2. ab3: ab3: 3. Kh1: ... und
b) 1. ... b3 2. ab3: ab3: 3. Lb3: ...

Das Programm 'sieht' den Verlust des Springers auf h1, aber auch gleichzeitig den eines Bauern. Da es - nach der Minimax-Methode - davon ausgeht, daß Weiß den jeweils besten Zug wählt, nimmt es an, Weiß würde den Springer in jedem Fall im nächsten Zug nehmen, da er einen höheren Wert hat als der Bauer.

Weiter wird nicht gerechnet, dieser vierte Halbzug stellt den Horizont dar, über den hinaus das Programm nicht sehen kann.

Gerade im Endspiel mit seinen oft zehn, zwölf oder mehr Halbzügen tiefen Kombinationen macht sich dieser Horizonteffekt bemerkbar. Wir werden im Kapitel 5 darauf noch näher eingehen.

a) Greenblatts MAC HACK

Einen Schritt weiter ging 1966 (wir lassen hier ein paar Zwischenstufen weg) ein Student namens Richard Greenblatt, der am Massachusetts Institute of Technology in Boston, kurz MIT genannt, an einem Projekt zur Erforschung der Künstlichen Intelligenz beteiligt war. Allerdings hat sein Programm MAC HACK VI nicht unmittelbar damit zu tun. In erster Linie ging es ihm darum, die bekannten Algorithmen mit den besten Werkzeugen zu verwirklichen.

Dazu stand ihm eine PDP-6 (mit 256 KB RAM!) zur Verfügung. Geschrieben wurde das Programm mit einem Macro-Assembler

(MIDAS), der eine Fülle von Software-Tools zum Debugging, zur Strukturierung der Daten und zur Verknüpfung von Programm-Modulen bot. Im Grunde war Greenblatt der erste Programmierer, der an ein Schachprogramm so ambitioniert heranging, wie man es bis dahin nur bei der Entwicklung von hochkomplexen wissenschaftlichen Anwendungen gewohnt war.

So ist auch nicht weiter verwunderlich, daß MAC HACK innerhalb von - sage-und-schreibe - nur drei Monaten fertig war. Dabei erwiesen sich manche Einrichtungen, die heute zum Standard aller bedeutenden Schachprogramme gehören, als äußerst nützlich:

1) optische Anzeige des Schachbrettes und der Notation;
2) Eingabe der Züge mit der üblichen Notationsform;
3) optische Anzeige der Bewertung der Züge;
4) Protokolle über Auswahl und Bewertung;
5) statistische Funktionen über Rechenzeit und
6) Anzahl der berechneten Züge.

Greenblatt benutzte diese, auf den ersten Blick nur dem Komfort dienenden Funktionen hauptsächlich zur Beobachtung des Programms. Erstmals wurde nämlich auch ein Schachprogramm anhand gespielter Partien verbessert und dazu waren Einblicke in die 'Denkvorgänge' des Rechners unumgänglich.
MAC HACK repräsentiert die klassische Form eines Typ-B-Programmes. Ein sehr großer Teil des Codes dient dazu, herauszufinden, welche Züge (die 'plausiblen') überhaupt bewertet werden sollen.

Dieses Programm nahm im Januar 1967 an der Schachmeisterschaft des Staates Massachusetts teil und erreichte ein Remis (bei vier Niederlagen in fünf Partien). Dieses Unentschieden ist bekannt als die erste Partie eines Computers gegen einen Menschen, die das Programm nicht verlor. Schauen wir uns aber einmal die erste Partie an, die ein Computerprogramm gegen einen menschlichen Spieler unter Turnierbedingungen spielte - und verlor.

Gespielt am 21. Januar 1967
Weiß: Spieler (ca. ELO 2190)
Schwarz: MAC HACK VI

01	g2-g3	e7-e5	30	Td3-d2	Te2xd2
02	Sg1-f3	e5-e4	31	Dc3xd2	Sg6-e5
03	Sf3-d4	Lf8-c5	32	Tf1-d1	Dd7-c7
04	Sd4-b3	Lc5-b6	33	Lg2-d5	Kg7-g6
05	Lf1-g2	Sg8-f6	34	b3-b4	Lc5-b6
06	c2-c4	d7-d6	35	Dd2-c2	Se5-c6
07	Sb1-c3	Lc8-e6	36	Ld5-e6	Sc6-d4
08	d2-d3	e4xd3	37	Td1xd4	Lb6xd4
09	Lg2xb7	Sb8-d7	38	Dc2xf5+	Kg6-g7
10	e2xd3	Ta8-b8	39	Df5-g4+	Kg7-h6
11	Lb7-g2	0-0	40	Dg4xd4	Dc7-e7
12	0-0	Le6-g4	41	Dd4-h4+	Kh6-g6
13	Dd1-c2	Tf8-e8	42	Le6-f5+	Kg6-g7
14	d3-d4	c7-c5	43	Dh4xh7+	Kg7-f8
15	Lc1-e3	c5xd4	44	Dh7-h8+	Kf8-f7
16	Sb3xd4	Sd7-e5	45	Dh8-a8	De7-c7
17	h2-h3	Lg4-d7	46	Da8-d5+	Kf7-g7
18	b2-b3	Lb6-c5	47	Kh2-g2	Dc7-e7
19	Ta1-d1	Dd8-c8	48	h3-h4	Kg7-h6
20	Kg1-h2	Se5-g6	49	g3-g4	Kh6-g7
21	Le3-g5	Te8-e5	50	h4-h5	De7-e2
22	Lg5xf6	g7xf6	51	h5-h6+	Kg7-f8
23	Sc3-e4	f6-f5	52	h6-h7	De2xf2
24	Se4-f6+	Kg8-g7	53	Kg2xf2	Kf8-e7
25	Sf6xd7	Dc8xd7	54	h7-h8D	a7-a6
26	Sd4-c6	Tb8-e8	55	Dd5-e6++	
27	Sc6xe5	Te8xe5			
28	Dc2-c3	f7-f6			
29	Td1-d3	Te5-e2			

Wir haben bewußt auf eine Kommentierung der Partie verzichtet, um MAC HACK VI nicht aus heutiger Sicht zu be- bzw. verurteilen, was ihm sicher nicht gerecht würde. Immerhin schlägt er sich in dem Turnier noch ganz beachtlich, verliert

einmal nur sehr knapp im Endspiel und erreicht gegen einen der menschlichen Spieler (ca. ELO 1400) ein Remis.

MAC HACK wurde bis Mitte der siebziger Jahre kontinuierlich verbessert und lief auf allen Rechner der PDP-Serie im Time-Sharing. Damit bekam die Idee des Schachspiels auf Computern erheblichen Auftrieb, denn an diesen Rechnern arbeiteten vorwiegend Programmierer, die auch verstanden, was sie da spielten. Leider trat MAC HACK nie bei den US-Computerschachmeisterschaften und den Weltmeisterschaften an, so daß bis heute nicht so recht einzuordnen ist, wie stark dieses Programm im Vergleich zu anderen wirklich war.

Immerhin wurde MAC HACK eine außergewöhnliche Ehrung zuteil: Der US-Schachverband ernannte das Programm wegen seiner Verdienste um das Computerschach zum Ehrenmitglied!

b) Slate and Atkin: CHESS 4.0 und Nachfolger

Larry Atkin hatte schon 1968 als Student an der Northwestern University an einem ziemlich schwachen Schachprogramm gewerkelt, als David Slate - ein recht guter Schachspieler - davon hörte und sich beteiligte. Das war 1969 und das Ergebnis nannte sich schlicht CHESS 2.0. Bis 1970 wurde daraus CHESS 3.0, das an den amerikanischen Computerschachmeisterschaften teilnahm und gewann. Das Programm wurde vollständig in Assembler auf einer CDC 6400 geschrieben.

Auch CHESS 3.5 war bei den Meisterschaften 1971 siegreich, während die Version 3.6 1972 gewann. Schon während dieser zwei Jahre hatten Slate und Atkin begonnen, einen neuen, von der 3er-Serie abweichenden Ansatz zu untersuchen; CHESS 3.6 und seine Vorläufer waren nämlich - wie MAC HACK VI - Programme vom Typ Shannon B, die nur die jeweils 'plausiblen' Züge untersuchen. Mit der letzten Version schien ihnen dieses Konzept ausgereizt und sie begannen CHESS 4.0, das mit der Shannon-A-Strategie arbeiten sollte.

Das Konzept war gut und die implementierten Algorithmen für die Suche und Bewertung ausgefeilt. CHESS 4.0 sowie die Versionen 4.5 und 4.6 dominierten das Computerschach von 1974 bis fast 1978; CHESS 4.6 wurde 1977 in Toronto sogar Computerschach-Weltmeister.

Wie gut CHESS 4.5 war, zeigt vielleicht folgende Blitzpartie, die anläßlich einer Schauveranstaltung gespielt wurde. Nach den vereinbarten Regeln konnte der menschliche Spieler über 5 Minuten Gesamtbedenkzeit verfügen, während das Programm auf eine durchschnittliche Rechenzeit von 5 Sekunden pro Zug eingestellt wurde. Der Gegner war immerhin mit einer ELO-Zahl von ca. 2000 bewertet.

Gespielt im September 1979 in Houston, Texas
Weiß: Spieler
Schwarz: CHESS 4.5

01	e2-e4	Sb8-c6	14	Sf5-g3	Lh1-f3
02	d2-d4	d7-d5	15	Sb1-c3	f6xg5
03	e4-e5	f7-f6	16	f4xg5	d5-d4
04	f2-f4	Lc8-f5	17	Lc1-f4+	Kc7-c6
05	g2-g4	Lf5-e4	18	Lf4-e5	Lf3-d5
06	Sg1-f3	Sc6xd4	19	Sc3xd5	Db6xb2
07	Sf3xd4	Le4xh1	20	Ta1-d1	Kc6xd5
08	e5-e6	a7-a6	21	Df7-f5	Kd5-c6
09	Lf1-d3	c7-c5	22	Df5-e4+	Kc6-b6
10	g4-g5	Dd8-d6	23	De4-d5	Db2-b4+
11	Dd1-h5+	Ke8-d8	24	Ke1-e2	h7-h6
12	Dh5-f7	Kd8-c7	25	Dd5-d7	h6xh5
13	Sd4-f5	Dd6-b6			

Und Weiß verliert durch Überschreiten der Bedenkzeit. Beeindruckend ist, wie CHESS sich 'ruhig und gelassen' gegen den scharfen Angriff von Weiß wehrt - und das bei sehr begrenzter Rechenzeit.

c) Die Levy-Wette

Noch interessanter sind die Partien, die CHESS 4.5 gegen den schottischen Internationalen Meister David Levy spielte. Aber bevor wir Ihnen eine der Partien aus dem Jahre 1977 zum Nachspielen anbieten, wollen wir erzählen, wie es zu der Wette kam.

David Levy ist einer der Pioniere des Computerschach, das zeigt sich allein schon an der Tatsache, daß er seit den Tagen der ersten Computerschach-Turniere immer wieder als Turnierleiter engagiert wird. Darüber hinaus ist Levy einer der Gründer und Sprecher der ICCA (International Computer Chess Association), dem Verband aller Schachprogrammierer und Freunde des Computerschachs.

Jedenfalls kam es 1968 anläßlich einer Cocktailparty zu einer heißen Diskussion über die Schachfähigkeiten von Computern zwischen Levy und zwei Professoren, die meinten, es würde keine zehn Jahre mehr dauern, bis ein Computerprogramm ihn, David Levy, würde schlagen können. Man schloß eine Wette ab und die Kontrahenten setzten je 250 Dollar ein. Ein Jahr später erhöhte Seymour Papert (der Erfinder der Computersprache LOGO) die Summe auf 750 Dollar, die in den folgenden Jahren durch weitere Mitwetter auf insgesamt 1250 Dollar anstieg.

Die Wette war zwar sehr medienwirksam, doch fand sich bis 1977 keine Gelegenheit, einen klärenden Wettkampf durchzuführen. Am 1. April 1977 trat David Levy in Pittsburgh, Pennsylvania zu einem Match über zwei Partien gegen CHESS 4.5 an. Die Regeln waren für ihn günstig: Da CHESS das Match nur gewonnen hätte, wenn es beide Partien siegreich beendet hätte, brauchte Levy nur eine Partie zu gewinnen (oder einfach beide remisieren).

Levy gewann die erste Partie (und damit das Match) einfach:

Gespielt am 1. April 1977 in Pittsburgh, PA
Weiß: CHESS 4.5 auf CYBER 176
Schwarz: David Levy

01	e2-e4	c7-c5	23	Ta1-b1	Sc8-b6
02	Sg1-f3	d7-d6	24	Tf3-f1	Tf8-b8
03	d2-d4	c5xd4	25	Tb1-d1	f7-f6
04	Sf3xd4	Sg8-f6	26	a3-a4	a6-a5
05	Sb1-c3	g7-g6	27	b4-b5	c6xb5
06	f2-f3	Lf8-g7	28	a4xb5	Tb8-c8
07	Lc1-e3	0-0	29	Td1-d3	Tc8-c5
08	Dd1-d2	Sb8-c6	30	Td3-g3	Ta8-c8
09	Lf1-c4	a7-a6	31	Tf1-f3	a5-a4
10	Sd4xc6	b7xc6	32	h2-h4	a4-a3
11	0-0	Sf6-d7	33	f5xg6	h7xg6
12	f3-f4	Sd7-b6	34	Tf3-e3	Ld7-e6
13	Lc4-e2	Lc8-e6	35	h4-h5	g6-g5
14	b2-b3	Sb6-c8	36	Sc3-d5	a3-a2
15	a2-a3	Dd8-a5	37	Te3-a3	Le6xd5
16	b3-b4	Da5-c7	38	e4xd5	Tc5xc2
17	f4-f5	Le6-d7	39	Le2-d1	Tc2-d2
18	Le3-h6	Dc7-b6+	40	Kh1-h2	Tc8-c1
19	Kg1-h1	Db6-d4	41	Ld1-b3	a2-a1D
20	Dd2xd4	Lg7xd4	42	Ta3xa1	Tc1xa1
21	Tf1-f3	Ld4-g7	43	Tg3-e3 und aufgegeben	
22	Lh6xg7	Kg8xg7			

Levy spielte hier ganz bewußt nach dem Motto "Wer wenig tut, macht wenig falsch" und ließ CHESS einfach auflaufen. Klar, David Levy als einer der besten Kenner des Computerschachs wußte um die Unfähigkeit eines Programmes, langfristige Pläne zu schmieden, und nutzte diese Tatsache aus.

Ein Jahr später - die Wetter hatten ihre Einsätze verdoppelt, so daß es Levy immerhin um 2500 Dollar ging - sollten die zehn Jahre der Wette abgelaufen sein. Es kam also zum alles entschei-

denden Match - diesmal gegen CHESS 4.6, die aktuellste Version. Die neuen Regeln besagten, daß das Programm den Menschen dann besiegt habe, wenn es aus sechs Partien mehr als 3 Punkte erreicht hätte; es mußte also wenigstens eine Partie gewinnen, wenn die anderen fünf alle Remis blieben.

Die erste Partie ist eine der bis dahin interessantesten Schachspiele zwischen einem Computer und einem Menschen. Levy selbst beschreibt die Situation des Wettkampfes und dieser ersten Partie so (aus ICCA Newsletter Vol.1, No.2, September 1978):

"Ich nahm Platz in einem fast schalldichten Glaskasten und spielte auf einem Spezialbrett, das David Cahlander von CDC erfunden hatte. Das Brett ermöglichte die direkte Übertragung der Züge vom Computer; wenn das Programm seinen Zug gefunden hatte, leuchteten die 'von'- und 'nach'-Felder auf, so daß ein Mensch die Figuren setzen konnte.

Das Match begann mit einem sehr aufregenden Spiel. Kurz nach der Eröffnung übersah ich die Tatsache, daß ein Springeropfer, das ich für nutzlos angesehen hatte, in Wirklichkeit zerschmetternd war. Den Zug des Programmes, den ich übersehen hatte, war '... Dg5', der das Opfer nicht nur sinnvoll, sondern für mich sogar ziemlich verheerend macht. Ich überredete das Programm zum Damentausch, um einen möglichen Mattangriff abzuwehren und so erreichten wir ein Endspiel, in dem ich drei Bauern weniger hatte. An dieser Stelle begann das Programm zu patzen und nach einer haarsträubenden Verteidigung erreichte ich noch das Remis."

Diese Partie zeigt eine ganze Menge der Stärken und Schwächen von Schachprogrammen, wie sie auch heute noch anzutreffen sind.

Gespielt am 26. August 1978 in Toronto, Kanada
Weiß: David Levy
Schwarz: CHESS 4.6

01	g2-g3	d7-d5	07	Lc1-b2	Dd8-e7
02	Lf1-g2	e7-e5	08	a2-a3	e5-e4
03	d2-d3	Sg8-f6	09	Sf3-e1	0-0
04	Sg1-f3	Sb8-c6	10	d3-d4	Lc5-d6
05	0-0	Lc8-d7	11	e2-e3	Sf6-g4
06	b2-b3	Lf8-c5			

Levy hat durch frühzeitiges Verlassen bekannter Eröffnungsvarianten das Programm dazu gezwungen, die Eröffnungsbibliothek zu verlassen; d.h., daß CHESS nicht mehr die besten Züge einfach aus dem Speicher abrufen kann, sondern - wie in Mittel- und Endspiel - jede Variante berechnen und bewerten muß.

Die Stellung sieht nicht besonders spannend aus, es ist ein Bild, das man mit vertauschten Farben aus der Indischen Verteidigung kennt. Schwarz hat zwei gut plazierte Läufer und freibewegliche Türme auf der Grundreihe; Weiß dagegen hat sich beide Fianchetto-Läufer von Bauern blockieren lassen, kann den Springer auf E1 nicht bewegen und steht etwas unbeweglich; mit dem Bauernzug h2-h3 käme wieder Bewegung ins weiße Spiel - und hier hat Levy zwar die Möglichkeit Sg4xe3 gesehen, aber für sinnlos gehalten.

12 h2-h3 Sg4xe3

mit Angriff auf die weiße Dame.

13 f2xe3 De7-g5

die Pointe, die David Levy übersehen hatte. Was bleibt ihm übrig? Die Bauern auf E3 und G3 sind angegriffen, letzterer gleich zweimal und beide zugleich können nicht gedeckt werden - CHESS bekommt für den geopferten Springer nicht nur zwei Bauern zurück, sondern auch einen starken Angriff auf den König.

14 g3-g4 ...

noch die beste Verteidigung.

14 ... Dg5xe3+

da hat er den Salat! Der Bauer ist futsch und die beiden schwarzen Läufer zeigen bedrohlich in Richtung weißer König. Eigentlich ist das Spiel für Weiß schon fast verloren - säßen sich zwei Menschen gegenüber, hätte der Anziehende vielleicht schon resigniert und aufgegeben. Aber David weiß, wo Goliath CHESS seine Schwächen hat.

15 Tf1-f2 Ld6-g3

Es kommt immer schlimmer!

16 Dd1-e2 ...

Eine Möglichkeit, den Angriff des Schwarzen zu stoppen, ist hier der Damentausch; geht Schwarz darauf ein, ist der erste Schwung ziemlich gebremst und das Opfer (fast) widerlegt.

16 ... De3xf2+
17 De2xf2 Lg3xf2+
18 Kg1xf2 f7-f5

Die Damen sind zwar auf Umwegen getauscht, aber Schwarz hat noch mehr in petto.

19 g4xf5 Sc6-e7

Das ist zu schlapp! Schlägt der Turm auf F5, bleibt es für Weiß gefährlich (19. ... Tf8xf5+ 20. Kf2-e3 Ta8-f8 21. Sb1-d2 Ld7-e8 etc.).

20	c2-c4	Tf8xf5+
21	Kf2-g1	c7-c6
22	Sb1-c3	Tf5-h5
23	Kg1-h2	Ta8-f8
24	Sc3-d1	Se7-g6
25	Ta1-c1	Ld7xh3
26	Lg2xh3	Tf8-f1
27	Se1-g2	Tf1-f3

So, jetzt hat der weiße Läufer auf H3 keine Chance mehr; der Springer G2 kann ihn nicht mehr decken und der andere auf D1 auch nicht - der Tausch Läufer gegen Läufer unter Zugewinn eines Bauern für Schwarz ist perfekt. Bemerkenswert an dieser Kombination ist, daß die Pointe des Zuges 25. ... Ld7xh3 erst im 28. Zug liegt, das Programm also über sieben(!) Halbzüge hinweg rechnen mußte. Levy versucht mit seinem nächsten Zug diese Pointe noch um einen weiteren Halbzug zu verschieben, in der Hoffnung, den schwarzen Turmzug nach H3 über den Horizont des Programmes hinauszudrücken.

28	c4xd5	Th5xh3+
29	Kh2-g1	c6xd5
30	Tc1-c8+	Sg6-f8
31	Lb2-c3	Tf3-d3
32	Sd1-e3	Th3xe3
33	Sg2xe3	Td3xe3

Eine kleine Bilanz: beide Seiten haben neben ihren Königen noch je eine Leichtfigur (Läufer bzw. Springer) und je einen Turm, Schwarz aber hat drei volle Bauern mehr als Weiß! Die Position ist für beide Seiten nicht besonders überwältigend zu nennen, so daß Schwarz mit einem extremen Vorteil ins Endspiel geht. Noch einmal: würde ein Mensch die schwarzen Figuren führen, hätte Weiß sicher spätestens an dieser Stelle aufgegeben; Meister Levy rechnet jedoch mit der Endspielschwäche des Programmes.

34	Lc3-b4	Te3-f3
35	Tc8-d8	h7-h6
36	Td8xd5	Tf3xb3
37	Td5-d8	Tb3-f3
38	Td8-a8	g7-g5
39	d4-d5	h6-h5
40	d5-d6	Kg8-g7
41	Ta8xa7	...

Und der erste Mehrbauer fällt.

41	...	Tf3-f7
42	Ta7-a5	Kg7-f6

Damit ist der Turm von der F-Linie abgeschnitten.

43	Lb4-c3+	Kf6-g6

Das hätte er gleich haben können und ein Tempogewinn durch den weißen Läuferzug ist es auch nicht.

44	Ta5-e5	Tf7-f3
45	Lc3-b4	Tf3-f4
46	Te5-e7	Tf4-f7

Aha! Tauschen will er! Pustekuchen. Stattdessen fällt Mehrbauer Nummer 2.

47	Te7xe4	Tf7-d7
48	Te4-e7	...

Jetzt möchte Weiß die Türme vom Brett haben - um dann den Bauer auf E7 zur Dame zu machen ...

48	...	h5-h4
49	Kg1-g2	g5-g4
50	Kg2-h2	b7-b6

Er hat die Aussichtslosigkeit seines Bauernvormarsches auf dem Königsflügel eingesehen und fängt einfach irgendetwas anderes an.

51 Kh2-g2 ...

Das Remis läßt schön grüßen!

51 ... Td7-d8
52 a3-a4 Sf8-d7
53 a4-a5 Sd7-f6
54 a5xb6 ...

Und eine letzte schöne Springergabel holt diesen Bauern zurück - oder etwa nicht?

54 ... Sf6-d5
55 b6-b7 Sd5xe7
56 d6xe7 Td8-h8
57 b4-d6 ...

kostet den schwarzen Turm das Leben.

57 ... Kg6-f6
58 b7-b8D Th8xb8

Wo sind die vielen Mehrbauern geblieben?

59 Ld6xb8 Kf6xe7
60 Lb8-f4 Ke7-f6
61 Lf4-d2 Kf6-g6
62 Ld2-e1 Kg6-g5
63 Le1-f2 Kg5-h5
64 Lf2-e1 ...

Und so weiter, und so weiter. Schlimmstenfalls gibt Weiß seinen Läufer für die zwei Bauern und dann stehen sich die Monarchen ganz ohne Armeen gegenüber - ein klassischer Fall von Remis.

Diese Partie, obwohl noch in der 'Frühzeit' des Computerschach gespielt, enthält alle Elemente, die man immer beobachten kann, wenn Menschen gegen Computer antreten. Mit dem Unterschied, daß heute jeder Mikrocomputer-Besitzer das David-Levy-Erlebnis haben kann, wenn er gegen eines der guten Programme spielt.

CHESS erlebte noch die Versionen 4.7 und 5.0, die aber lange nicht so erfolgreich waren wie die Vorläufer. Das lag einfach daran, daß die Konkurrenz auch nicht schlief; zahlreiche Programmierer in den Vereinigten Staaten forschten zum Thema Schach und in manchen Instituten wurde eine Menge Zeit und Geld in die Entwicklung von Schachprogrammen gesteckt.

Auch in anderen Ländern machten sich fähige Leute ihre Gedanken - neben den westeuropäischen Ländern (unter anderem in der Bundesrepublik, Holland, Schweden, Frankreich, Großbritanien) spielte die Sowjetunion von Anfang an eine wichtige Rolle; z.B. der Ex-Weltmeister Michail Botwinnik, der seit Beginn der siebziger Jahre an seinem Über-Super-Schachprogramm PIONIER arbeitet, aber bis heute kein spielfähiges Programm erstellen konnte. Die zählbaren Erfolge sowjetischer Schachprogramme liegen vielmehr in der Frühzeit des Computerschaches.

d) Das erste Weltmeisterprogramm: KAISSA

In den Jahren 1966 und 1967 fand ein Schachwettkampf statt, der einiges Aufsehen erregte: Ein Programm der Stanford Universität spielte 4 Partien gegen das Programm ITEP, geschrieben von Michail Domskoj und Wladimir Arlasarow am Kybernetik Institut der Universität Moskau. Dieses Match der Supermächte endete zugunsten des sowjetischen Programms, wobei die Qualität der Partien eher bescheiden war; das Stanford-Programm hatte die schlechte Angewohnheit, wegen seiner selektiven Suche wichtige Züge einfach zu übersehen, und so geriet es in allen Partien schnell in Nachteil und konnte lediglich einmal Remis halten.

Aus dem ITEP wurde nach 1971 das Programm KAISSA, das sich auch gegen menschliche Gegner recht achtbar schlug. 1972 spielte es ein Zwei-Partien-Match gegen die Leser der PRAWDA und endete mit 0,5 zu 1,5. Auch der damalige Weltmeister Boris Spasski mußte bei einem Schaukampf die Stärke des Programmes anerkennen.

KAISSA - benannt nach der Göttin des Schach - war für seine Zeit mit Sicherheit eines der modernsten Programme; es zählt zum Shannon-A-Typ, rechnet also über etwa 5 bis 7 Halbzüge tief alle Varianten durch und schließt eine Ruhesuche an, die sich den Abtauschvarianten und versteckten Drohungen widmet. Dies sind Eigenschaften, die heute fast jedes Schachprogramm - ob auf Großrechnern oder Mikros - besitzt. KAISSA war mit einer ganzen Menge Schachwissen ausgestattet, so daß es besonders in positioneller Hinsicht über ganz erstaunliche Fähigkeiten verfügte. Einziges Handicap: KAISSA hatte eine unausrottbare Neigung zu Patzern - menschliche Gegner konnten diese Fehler in der Regel leicht aufdecken, andere Computer taten sich schwerer.

1974 wurde KAISSA Weltmeister unter den Schachprogrammen, wobei es von den Turnierregeln (Schweizer System) begünstigt wurde: Es mußte weder gegen den Zweitplazierten CHESS 4.0, noch gegen den Dritten RIBBIT antreten. KAISSA gewann seine vier Partien recht leicht und durfte sich als erstes Schachprogramm Weltmeister nennen.

Bei der zweiten Computerschach-Weltmeisterschaft 1977 in Toronto war es umgekehrt: CHESS 4.6 gewann den Titel mit 4 zu 0, mußte aber wieder nicht gegen KAISSA spielen. Dieses mit Spannung erwartete Zusammentreffen wurde kurze Zeit später im Rahmen eines Schaukampfes nachgeholt - CHESS gewann beide Partien. Interessant ist die Tatsache, wie 'menschenähnlich' die Programme spielten. Besonders deutlich wird das bei der folgenden Partie:

Gespielt am 8. August 1977 in Toronto, Kanada
Weiß: KAISSA (UdSSR)
Schwarz: CHESS 4.6 (USA)

01	e2-e4	Sb8-c6	24	Kg1-g2	Kg8-g7
02	Sg1-f3	e7-e6	25	Sf3-g5	Td7-d2
03	d2-d4	d7-d5	26	Te1-g1	Td2-c2
04	Lf1-d3	d5xe4	27	b2-b3	Sg6-e5
05	Ld3xe4	Lc8-d7	28	Tg1-h1	Tc2xa2
06	0-0	Sg8-f6	29	Th1-h4	Se5-d3
07	Tf1-e1	Sf6xe4	30	Sg5-h3	Ta2-b2
08	Te1xe4	Lf8-e7	31	g4-g5	Kg7-g8
09	c2-c4	f7-f5	32	Sh3xf4	Tb2xf2+
10	Te4-e1	0-0	33	Kg2-g3	Tf2xf4
11	Sb1-c3	f5-f4	34	Th4xf4	Sd3xf4
12	Dd1-d3	Dd8-e8	35	Kg3xf4	Kg8-f7
13	g2-g3	f4xg3	36	b3-b4	Kf7-e6
14	h2xg3	De8-f7	37	Kf4-e4	a7-a6
15	Lc1-f4	g7-g5	38	Ke4-f4	Ke6-d6
16	d4-d5	e6xd5	39	Kf4-e4	c7-c5
17	Sc3xd5	g5xf4	40	b4xc5+	Kd6xc5
18	Sd5xe7+	Sc6xe7	41	Ke4-d3	a6-a5
19	Dd3x7	Se7-g6	42	Kd3-c3	a5-a4
20	Dd7xf7+	Tf8xf7	43	Kc3-d3	Kc5-b4
21	g3-g4	Tf7-d7	44	Kd3-c2	Kb4xc4
22	Ta1-d1	Ta8-d8	45	aufgegeben	
23	Td1xd7	Td8xd7			

Zwei Dinge fallen sofort ins Auge: Erstens spielen beide Programme sehr materialbewußt, d.h. sie rechnen Abtauschvarianten besonders tief durch, und zweitens, wie schwach Weiß das Endspiel behandelt.

Wenn man davon ausgeht, daß diese beiden Programme zum Ende der siebziger Jahre die Spitze des Computerschach darstellen und trotzdem äußerst endspielschwach sind, wird man die Anstrengungen der Programmierer verstehen, das Endspiel zu verbessern. Wie gesagt, die grundsätzlichen Algorithmen sind seit jener Zeit bekannt und werden seitdem in fast allen Programmen angewendet; eine weitere Verbesserung versprach

man sich von schnelleren Rechnern mit größerer Speicherkapazität, die in der Lage wären, allein auf Grund ihrer hardwareseitigen Voraussetzungen tiefer zu rechnen und mehr Varianten zu speichern. So kamen die Computergiganten ins Spiel.

1.4 Die Superriesen - BELLE und CRAY BLITZ

1978 schockte ein Programm die Computerschachwelt: BELLE. Zunächst war es ein Programm unter vielen, das bereits 1972 von Ken Thompson für PDP 11-Rechner geschrieben wurde und bei den Meisterschaften der Jahre 1973 und 1974 mit geringem Erfolg teilnahm. Interessant ist, daß BELLE unter UNIX lief und auch vom 'Vater' dieses Betriebssystems, Ken Thompson, geschrieben war.

Thompson verfügte in den folgenden Jahren über nahezu unbeschränkte Mittel bei der Weiterentwicklung von BELLE; sein Arbeitgeber, die Bell Thelephone Laboratories versprachen sich von seiner Arbeit Aufschlüsse auch über die Programmierung anderer komplexer Anwendungen.

Es war schnell klar, daß eine reine Softwareverbesserung bestenfalls einen Gleichstand mit den anderen Programmen geben würde. Thompson griff also zur denkbar aufwendigsten Lösung: einem reinen Schachrechner. Dieses Zusammenspiel von anfangs 25 Chips brachte zunächst eine erhebliche Beschleunigung bestimmter Funktionen und wurde nach und nach erweitert.

a) BELLE: eindeutig auf Schach getrimmt

Das BELLE-Programm des Jahres 1978 verfügt über einen Schachprozessor mit 325 Chips, die mit einem spezifischen Mikrocode programmiert werden; die Steuerung dieses Prozessors

übernimmt ein PDP 11-Rechner, der in C programmiert ist. Das Programm ist 90 Kilobytes lang, dazu kommen 5 KB Daten und eine Tabelle von 128 KB.

1980 wurde BELLE in einer Version Computerweltmeister, die bereits 1700 Chips für Schachzwecke enthält. Diese Spezialisierung macht BELLE zu einem Brute-Force-Programm (bzw. -Rechner), das im Mittelspiel ungefähr 8 Halbzüge tief rechnet, im Endspiel aber zwischen 10 und 30 (!) Halbzüge tief sieht. Diese Endspielstärke war es in erster Linie, die BELLE in den Jahren 1980 bis 1983 nahezu konkurrenzlos machte.

Sehen wir uns einmal an, wie BELLE mit dem vormaligen Weltmeister CHESS 4.7 umspringt:

Gespielt am 4. Dezember 1978 in Washington D.C., USA
Weiß: BELLE
Schwarz: CHESS 4.7

01	e2-e4	Sb8-c6	26	Dg6xe6+	Tg7-f7
02	d2-d4	d7-d5	27	De6-h6	Tf7-g7
03	Sb1-c3	e7-e6	28	Dh6-h8+	Kg8-f7
04	Sg1-f3	Lf8-b4	29	e5-e6+	Kf7xe6
05	e4-e5	Sg8-e7	30	Dh8xg7	Lb7xg2
06	Lc1-d2	Se7-f5	31	Th1-h6+	Ke6-d7
07	Sc3-e2	Lb4-e7	32	0-0-0	Lg2-d5
08	c2-c3	0-0	33	Sd2-e4	Kd7-c8
09	Se2-f4	f7-f6	34	Th6-h8	Ld5xe4
10	Lf1-d3	f6xe5	35	Td1xd8+	Le7xd8
11	d4xe5	g7-g5	36	Dg7-e7	Kc8-b7
12	g2-g4	Sf5-g7	37	De7xe4+	Kb7-a7
13	Sf4-g2	b7-b6	38	Th8-g8	Ta8-b8
14	Dd1-e2	Lc8-b7	39	g4-g5	Ld8-e7
15	Th1-g1	a7-a5	40	Tg8xb8	Le7xg5+
16	a2-a4	Kg8-h8	41	f2-f4	Lg5xf4+
17	h2-h3	Kh8-g8	42	De4xf4	Ka7xb8
18	Tg1-h1	h7-h6	43	Kc1-d2	Kb8-b7
19	h3-h4	d5-d4	44	Kd2-d3	Kb7-c8
20	h4xg5	Sc6-b4	45	b2-b4	a5xb4
21	g5xh6	Sb4xd3+	46	Df4xb4	Kc8-d7
22	De2xd3	d4xc3	47	Db4-b5+	Kd7-d8
23	Dd3-g6	c3xd2+	48	Kd3-e4	aufgegeben
24	Sf3xd2	Tf8-f7			
25	h6xg7	Tf7xg7			

Diese Partie zeigt deutlich, wie überlegen BELLE schon damals war; herrliche Kombinationen mit verblüffenden, mehrere Halbzüge entfernten Pointen, ein ziemlich zwingendes Endspiel und nicht zuletzt eine Eröffnungsbibliothek mit über 350.000 Varianten machten BELLE fast unschlagbar; Gefahr drohte eigentlich nur vom anderen Giganten...

b) Auf dem größten Rechner: CRAY BLITZ

Auch hier gab es das Programm BLITZ schon einige Zeit, bevor sich der Erfolg einstellte. Robert Hyatt, Albert Gower und Harry Nelson hatten damit 1976 an den amerikanischen Computerschachmeisterschaften teilgenommen und den vierten Platz erreicht.

BLITZ ist ebenfalls ein Shannon-A-Programm und BELLE in vielen Aspekten ähnlich. Es ist mit zahlreichen Bewertungsfunktionen ausgestattet und da liegt auch der Haken: Je größer die Zahl der Bewertungsfunktionen ist, desto mehr Zeit muß für die Suche aufgewendet werden. Oder mit anderen Worten: man kann noch so viele, noch so schöne Bewertungsalgorithmen - bei BLITZ hat man es mit mehreren Tausend Statements zu tun - einbauen, sie nutzen nur dann etwas, wenn der Rechner schnell genug ist, sie auch zu durchlaufen. 1978 war BLITZ softwareseitig an einer Grenze angelangt und so beschloßen die Entwickler, ihr Programm auf dem damals mächtigsten Großrechner, der CRAY ONE laufen zu lassen.

Dieser Computer arbeitet mit mehreren Prozessoren, deren Zusammenspiel von einem der leistungsfähigsten Prozessoren überhaupt gesteuert wird. Mit dieser gewaltigen Kraft schafft BLITZ die Berechnung von bis zu 20.000 Stellungen pro Sekunde. Das Programm selbst ist 200 KB lang, dazu kommt eine Tabelle von über 120 Megabytes(!).

Diese Speichergigantomanie wird von Programmen wie BELLE oder BLITZ ausgenutzt, um möglichst viele Informationen über die Stellung und die Bewegungsmöglichkeiten der Figuren in Form von Bitmustern zu speichern, z.B. die Standorte aller weißen Bauern, die Bewegungsmöglichkeiten der schwarzen Dame etc.

Was soll das? Es werden Unmengen von Mustern in umfangreichen Tabellen abgelegt und miteinander verglichen - und das Ergebnis? Nun, mit diesen speicheraufwendigen Tabellen

werden zwei Funktionen eines Schachprogrammes erheblich beschleunigt: der Zuggenerator und die Suche.

Wie wir bei dem Kapitel über Shannon gesehen haben, muß bei einem Zuggenerator normalerweise für jede Position immer wieder jede Zugmöglichkeit jeder Figur berechnet werden, das kostet viel Zeit. Beim Verfahren mit den Bitmustern ist das nicht nötig; für jede Figur muß das vorhergehende Bitmuster nur jeweils verändert werden, um die neuen Zugmöglichkeiten zu erhalten. Und weil die Prozessoren nichts besser und schneller können als solche Operationen auf Bitebene, geht die Zuggenerierung sehr schnell.

Die Suche wird dadurch beschleunigt, daß die durchsuchten Züge ebenfalls durch einfache logische bzw. arithmetische Operationen erzeugt werden können - wie wir gesehen haben, führen einfache logische Verknüpfungen nach den Regeln der Boole'schen Algebra zu sehr aussagekräftigen Bitmustern.

Allerdings wird die Aussagekraft nur dann erreicht, wenn sehr viele Bitmuster erzeugt und verarbeitet werden; rechnen wir einmal zusammen, welche Muster man wenigstens braucht:

Je 1 für den Standort jeder Figur jeder Farbe plus 1 für die Bauernformation jeder Farbe, macht zusammen 14 Stück.
Je 1 für die Bewegungsmöglichkeit jeder Figur jeder Farbe, macht zusammen 32 plus je 16 für die Schlagzüge der Bauern.
Alle logischen Verknüpfungen der ersten beiden Gruppen, zusammen 14 mal 48, das sind 672.

Wie gesagt, das ist das absolute Minimum, belegt aber auch schon 8 mal 672 Byte gleich 5376 Byte. Nun können auch noch bestimmte Züge in bestimmten Stellungen als Bitmuster gespeichert werden, um die Generierung bereits berechneter Züge zu sparen - für jede dieser Positionen braucht man wieder alle 5376 Bytes. Man sieht, Speicherplatz kann man bei dieser Methode gar nicht genug haben.

Die Schachprogramme wie BELLE und CRAY BLITZ arbeiten mit diesem Verfahren. Wie man sich leicht vorstellen kann,

lassen sich solche Programme nicht mehr im Time-Sharing fahren. Wenn sie Partien spielen, sind die Rechner voll und ganz mit Schach beschäftigt, die Kosten sind immens; eine Partie von BLITZ auf der CRAY ONE kostet, wenn man nur den Preis für die Rechenzeit betrachtet, mehr als eine halbe Million Dollar (fünf Stunden bei rund 30 Dollar pro Sekunde Rechenzeit)!

Sehen wir uns einmal einen Kampf der Giganten an. Es handelt sich um eine Partie im Kampf um die US-Computerschach-Meisterschaft 1981, bei der BELLE Meister wurde und BLITZ den zweiten Platz belegte. BELLE war zu der Zeit absolut auf der Höhe, BLITZ dagegen lief noch nicht hundertprozentig zufriedenstellend - immerhin aber so gut, daß es außer in der Partie gegen BELLE immer gewann.

Gespielt am 9. November 1981 in Los Angeles, USA
Weiß: CRAY BLITZ
Schwarz: BELLE

01	e2-e4	e7-e5
02	Sg1-f3	Sb8-c6
03	Lf1-c4	Sg8-f6
04	Sf3-g5	...

Die Preußische Verteidigung. Sie gilt als ziemlich zweischneidige Sache, weil Weiß bei sehr agressivem Spiel schon sehr früh das Zentrum aufgibt und eigentlich nicht mehr als einen gewonnenen Bauer bei zerrütteter Stellung erreicht. Die Ambivalenz dieser Eröffnung zeigt sich darin, daß bei ruhigem Spiel genau der gegenteilige Effekt auftritt - eine recht ruhige Stellung entsteht.

Allerdings gibt es allerlei taktische Scharmützel, die aus der Preußischen Verteidigung rasante Mittelspiele entstehen lassen. Vermutlich hat BLITZ diese Eröffnung gespeichert, weil die Programmierer annehmen, Schwarz müsse auf diese Züge hin seine Eröffnungsbibliothek verlassen. Aber dem ist nicht so...

04 ... d7-d5
05 e4xd5 Sc6-a5

Springer am Rand ist zwar eine Schand', aber hier die beste Verteidigung!

06 Lc4-b5+ c7-c6
07 d5xc6 b7xc6
08 Dd1-f3 ...

Das ist ziemlich ungewöhnlich; Weiß hätte normalerweise Le2 gespielt.

08 ... Ta8-b8

Schwarz für den Bauern weniger ein schönes offenes Spiel.

09 Lb5xc6+ Sa5xc6

Vielleicht hätte Weiß hier besser auf den Bauern verzichtet.

10 Df3xc6+ Sf6-d7
11 d2-d3 ...

Zwischenbilanz: Weiß hat zwei Bauern mehr und eine noch gar nicht so zerrüttet aussehende Stellung - ob die Theorie (siehe oben) widerlegt ist?

11 ... Lf8-e7
12 Sg5-e4 Lc8-b7

So schnell ändert sich das Bild. Weiß muß jetzt um seine Dame fürchten; Schwarz hat alle Bewegungsmöglichkeiten für seine Figuren und kann im nächsten Zug rochieren.

13	Dc6-a4	Dd8-c7
14	Sb1-c3	Lb7-c6
15	Da4-c4	Dc7-c8
16	Sc3-d5	Lc6xd5
17	Dc4xd5	Dc8xc2

Ein folgenschwerer Einbruch.

18	0-0	f7-f6
19	f2-f4	Sd7-b6
20	Dd5-a5	...

Und wieder steht die weiße Dame sehr merkwürdig. Vermutlich ist etwas dran an der Regel, daß die Dame nicht zu früh ins Spiel gebracht werden soll (siehe 8. Zug).

20	...	Dc2xd3

So, das materielle Gleichgewicht ist hergestellt und Schwarz steht besser.

21	Da5xa7	0-0
22	Da7xe7	...

Eine Dame sieht rot! Aber was bringt dieser Amoklauf? Schwarz holt sich den Springer auf E4 und eventuell noch den schönen Bauern auf der F-Reihe; warten wir es ab...

22	...	Dd3xe4
23	De7-e6+	Kg8-h8
24	f4xe5	f6xe5
25	Tf1xf8+	Tb8xf8

Achtung! Matt droht mit Dame auf E1!

26	h2-h3	De4-e1+
27	Kg1-h2	h7-h6
28	De6xb6	Tf8-f1

Es sieht so aus, als sei Weiß einfach zu gierig; schon wieder droht Matt mit Turm auf H1.

29	Db6-d8+	Kh8-h7
30	Dd8-d3+	e5-e4
31	Dd3xf1	...

Nun hat sie ausgehaucht. Eine Verzweifelungstat der weißen Monarchin, die sich für das Leben ihres Königs opfert. Wahrscheinlich vergebens, denn Turm plus Läufer plus einen Mehrbauern wiegt die Dame kaum auf.

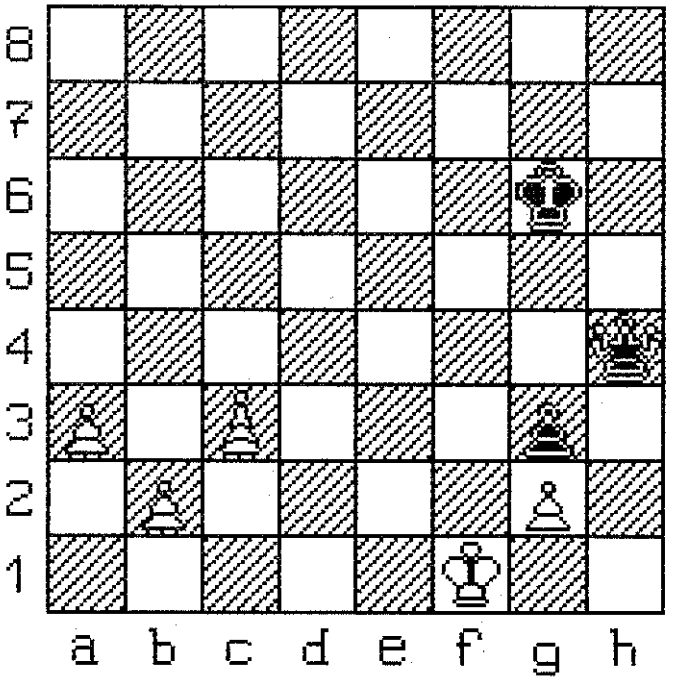
31	...	De1xf1
32	a2-a3	e4-e3

Will dieser Bauer sich etwa verwandeln? Das zu verhindern, kostet mindestens eine weiße Figur!

33	Lc1xe3	Df1xa1

Wäre BLITZ vernunftbegabt, hätte es spätestens nach dieser Demütigung aufgegeben - so aber geht es weiter bis zur bittren Neige.

34	Le3-d4	h6-h5
35	Ld4-c3	g7-g5
36	Lc3-e5	Da1-e1
37	Le5-c3	De1-f2
38	Kh2-h1	g5-g4
39	h3xg4	h5xg4
40	Kh1-h2	Df2-h4+
41	Kh2-g1	g4-g3
42	Kg1-f1	Kh7-g6



Und hier hatten die Operatoren ein Einsehen und machten dem ungleichen Kampf ein Ende (übrigens setzt Schwarz in spätestens 9 Zügen Matt - probieren sie es aus!).

Inzwischen - seit 1983 - ist CRAY BLITZ Computerschachweltmeister und wird seinen Titel bei den nächsten Meisterschaften (1986 in Köln) verteidigen müssen. Dabei ist die

Konkurrenz schärfer geworden; Programme, die dem Weltmeister gefährlich werden können, kommen allerdings aus einer ganz unvermuteten Ecke.

1.5 Aufstand der Mikros - SARGON und die anderen

Es ist wie in der Naturgeschichte: Die riesigen Dinosaurier (Programme, die auf Mainframe-Computern laufen) beherrschen das Bild, aber ganz unten beginnen die ersten Säugetiere (Programme, die auf Mikrocomputern laufen) sich bemerkbar zu machen. Nach und nach werden sie besser und schaffen es, mit ihrer Schnelligkeit und Flexibilität, den Großen Angst einzujagen.

1977 ist das Jahr, in dem erstmals ein Schachprogramm auf einem Mikrocomputer 'anständige' Partien spielte: SARGON war geboren. Seine Eltern, Dan und Kathe Spracklen, hatten zunächst gar keine Ambitionen mit dem Programm; im Gegenteil, als Dan im September mit der Arbeit begann, besaßen sie nicht einmal einen Mikro, sondern schrieben große Teile des Programms in einem Pseudocode für einen nichtvorhandenen, gedachten Rechner. Schließlich erwarb das Ehepaar einen Z80-Computer (Wave Mate Jupiter) und erlernte dessen Maschinencode.

Das Übertragen des bereits vorhandenen Pseudocodes war der geringere Teil der Arbeit; schwierig und zeitraubend verlief das Debugging, also die Fehlersuche. Ende Januar 1978 war es dann so weit: SARGON beherrschte die Regeln und wählte nur noch wirklich legale Züge - mehr aber auch noch nicht.

Doch dann - in einem Kraftakt sondergleichen - schafften es die beiden, das Programm zum ersten Schachturnier für Mikrocomputer im März 1978 fertigzustellen. SARGON gewann das Turnier leicht mit 5 zu 0 Punkten und wurde berühmt. Das Programmlisting wurde wenig später in Buchform veröffentlicht.

Aber das war erst der Anfang; SARGON hat sich laufend verändert und taucht unter verschiedenen Namen in verschiedenen Schachcomputern auf, ist als SARGON II eines der verbreitetsten Schachprogramme für APPLE und C 64 und schickt sich an, als SARGON III das Programm für IBM-PC (und Kompatible) sowie für den APPLE Macintosh zu werden.

SARGON II und 2.5 (nebst diversen Versionen) findet man in folgenden Schachcomputern:

BORIS, BORIS 2.5, Modul MORPHY für BORIS (Applied Concepts); CHAMPION SENSORY CHALLENGER, ELITE, SENSORY 9, PRESTIGE usw. (Fidelity) - Das 'usw.' soll besagen, daß sich danach die Spur von SARGON verwischt; ob und in welchem Umfang SARGON in den unzähligen Weltmeisterschafts- und Wettkampfprogrammen von Fidelity enthalten ist, werden erst die Archäologen des Computerschach entziffern können.

Vielleicht enthält ja gerade Ihr Schachcomputer SARGON und sie wußten es gar nicht. Wegen dieser enormen Verbreitung ist SARGON vermutlich eines der ganz wenigen Schachprogramme, die ihren Entwicklern auch finanziell etwas gebracht haben. SARGON ist allerdings nicht nur weit verbreitet, sondern auch spielstark; zwar ist es von der Struktur her ziemlich konventionell (Shannon-A-Typ mit Alpha-Beta-Algorithmus), dafür aber auch mit sehr viel nützlichem Schachwissen ausgestattet. Während die früheren Versionen über sehr kleine Eröffnungsbibliotheken verfügten, ist SARGON III (zumindest in der PC- und der Macintosh-Version) mit 68.000 Halbzügen gesegnet.

SARGON fühlt sich im Mittelspiel in offenen, taktisch bestimmten Positionen am wohlsten und kann sehr gefährliche Verwicklungen sauber durchspielen. Sehen wir uns an, wie SARGON im Spiel gegen den scheinbar übermächtigen Gegener BELLE bestand:

Gespielt am 30. Oktober 1979 in Detroit, USA
Weiß: SARGON 3
Schwarz: BELLE

01	Sb1-c3	d7-d5	07	Se2-c3	Lf8-b4
02	e2-e4	Sg8-f6	08	Lc1-d2	Lb4xc3
03	e4xd5	Sf6xd5	09	b2xc3	Lc8-f5
04	Sc3xd5	Dd8xd5	10	c3-c4	Dd5-d4
05	Sg1-e2	Sb8-c6	11	Ld2-e3	Dd4-c3+
06	d2-d3	e7-e5	12	Le3-d2	Dc3-a3

So, damit ist das Mittelspiel wohl eröffnet. Die Eröffnung ähnelt in mancherlei Hinsicht der Skandinavischen Verteidigung da; Schwarz steht recht beweglich, muß aber auf die Dame achten; Weiß ist etwas bedrängt.

13	g2-g4	...

Ist das schon der Verzicht auf die kurze Rochade? Ein scharfer Zug ist es allemal.

13	...	Lf5-d7
14	Lf1-g2	0-0
15	0-0	f7-f5
16	Dd1-b1	...

SARGON wird frech: Der Bauer auf B7 ist angegriffen, außerdem wird die schwarze Dame belästigt.

16	...	Ta8-b8
17	g4xf5	Ld7xf5
18	Db1-b5	a7-a6
19	Db5-d5+	Tf8-f7
20	Tf1-b1	...

Wieder eine der latenten Drohungen gegen die schwarze Dame, die sich in ihrer Ecke gar nicht wohlfühlt.

20	...	Tb8-d8
21	Dd5-f3	Sc6-d4

Nun wird die weiße Dame gejagt - solche intensiven Verfolgungen von Schwerfiguren sind typisch für Schachprogramme; es geht in der Regel um materiellen Vorteil, positionelle Aspekte spielen zwar auch eine Rolle, langfristige Pläne schmieden können sie jedoch nicht. Im konkreten Fall würde ein Mensch als Führer der schwarzen Steine vielleicht einen - auch mit Opfern verbundenen - Angriff auf den weißen König anzetteln; die Dame könnte über D6 eingreifen, der Springer versuchen, ein Schach auf E2 anzubringen usw.

22	Df3-d1	c7-c6
23	Ld2-g5	Td8-f8
24	Lg5-e3	Da3-c3
25	Tb1-c1	Dc3-b2

Typisch! Schwarz spielt auf Bauerngewinn. Dabei bietet sich doch ein zerstörerischer Angriff Richtung F2 geradezu an. Mal sehen, ob sich die Freßgier auszahlt.

26	a2-a4	Lf5-g6
27	Ta1-b1	Db2-a3
28	Tb1-a1	Da3-c3

Man will doch wohl nicht schon ins Remis-Geschäft kommen?

29	Le3-d2	Dc3-b2
30	Ta1-b1	Db2-a2
31	Tb1-a1	...

Und immer lockt das Weib. Einem menschlichen Spieler wäre wahrscheinlich schon der Geduldsfaden gerissen, er hätte die schwarze Dame über C5, D6 oder E7 längst auf den anderen Flügel gebracht; BELLE dagegen ist stur.

31	...	Da2-b2
32	Ld2-e3	Kg8-h8

Jetzt fällt beiden nichts mehr ein. SARGON könnte nun die Initiative übernehmen.

33	Ta1-b1	Db2-a3
34	Tb1-a1	Da3-c3
35	Le3-d2	Dc3-b2

Und nun rückt Remis durch Zugwiederholung in greifbare Nähe.

36	Ta1-b1	Db2-a2
37	Tb1-a1	Da2-b2
38	Ld2-e3	...

SARGON will nicht. Mit Tb1 wäre das Unentschieden besiegelt. Ein toller Erfolg für das Mikroprogramm gegen den Weltmeister; doch SARGON möchte nun herausfinden, ob es nicht sogar gewinnen kann.

38	...	Tf7-e7
39	Ta1-b1	Db2-a3
40	Tb1-a1	Da3-c3
41	Le3-d2	Dc3-b2
42	Ta1-b1	Db2-a2

Und schon wieder sind die beiden dicht am Remis.

43	Ld2-b4	...

Endlich passiert was! Ob es gut ist?

43	...	Tf8-f7
44	Tb1-a1	Da2-b2
45	Ta1-b1	Db2-a2
46	Lb4xe7	...

Weiß der Programmierer warum, aber erst jetzt nimmt Weiß den Qualitätsvorteil wahr. Man sagt von SARGON, daß es in geschlossenen Mittelspielpositionen nicht weiß, was es anfangen soll - bei dieser Partie sieht es wirklich so aus.

46	...	Tf7xe7
47	Tb1-a1	Da2-b2

Beide ratlos?

48	Kg1-f1	Te7-f7
49	Ta1-b1	Db2-a2
50	Dd1-e1	...

Ein schwerer Fehler! Weiß möchte den Bauern e5 haben und dafür seinen auf C2 hergeben; der schwarze Springer auf C2 ist aber sehr viel stärker und der schwarze Angriff mit der Dame auf der 2. Reihe so viel gefährlicher, daß auch die weiße Drohung, mit der Dame auf der 8. Reihe Schach zu geben, harmlos wird.

50	...	Sd4xc2
51	De1xe5	Lg6xd3+
52	Kf1-g1	Da2-a3

Deckt das Feld f8, wo der Turm das Schach unterbrechen soll, wenn die weiße Dame auf e8 oder b8 geht.

53	De5-b8+	Tf7-f8
54	Db8xb7	Sc2-d4
55	Tb1-a1	Da3-c5
56	Tc1-d1	Ld3xc4
57	Kg1-h1	Lc4-d5

So, Schwarz will alles tauschen, um dann im Endspiel mit einem Mehrbauern zu gewinnen.

58	Lg2xd5	Dc5xd5+	62	Ke1-d2	Tf8-d8+
59	Kh1-g1	Sd4-e2+	63	Kd2-c2	Se2-d4+
60	Kg1-f1	Dd5-f3	64	Kc2-d3	Sd4-b5+
61	Kf1-e1	Df3xf2+	65	Db7-d7	...

Der letzte Schnaufer ...

65	...	Td8xd7+
66	Kd3-c4	Df2-c2+
67	Kc4-b4	Dc2-c3++

Man sollte allerdings jetzt nicht in den Glauben verfallen, SARGON sei eben nur ein Mikroprogramm, denn erstens hat sich BELLE auch nicht gerade mit Ruhm bekleckert und zweitens hat mit dem FIDELITY-Experimentalprogramm Elite XC ein anderes Mikroprogramm Rache genommen: Im Juni 1985 gewann es die Erste Offene US-Meisterschaft für Schachprogramme; BELLE wurde Fünfter und konnte gegen Programme wie MEPHISTO Modular (ein Seriengerät trat an!) und NOVAG Y (ein Experimentierprogramm) nur ein Remis erreichen.

Woran liegt es, daß die Mikros so gut geworden sind?
Zwei Tatsachen müssen näher betrachtet werden:

1. Die Mikrocomputer von heute sind leistungsfähiger als die Großrechner der siebziger Jahre.

2. Die Algorithmen in Schachprogrammen sind bekannt und weitgehend ausgereift.

Wenn man also einen leistungsfähigen Mikroprozessor nimmt und die bekannten Algorithmen darauf programmiert, erhält man ein Programm, daß genauso spielstark ist wie die Programme auf den größeren Systemen. Das führt zu der Frage, ob es überhaupt noch eine Weiterentwicklung in der Schachprogrammierung geben kann.

Es kann. Einerseits ist die Rechengeschwindigkeit immer noch ein Qualitätsmerkmal - wenn ein Computer z.B. hundertmal so viele Positionen pro Sekunden berechnen kann wie sein Gegner, kann es über den Daumen gepeilt etwa zwei Halbzüge tiefer rechnen. Auf der anderen Seite kann viel Speicherplatz für aufwendigere Tabellen benutzt werden - und das nicht nur in der Eröffnung. Bei geschickter Anwendung verringert das Plus an

Speicherplatz die Rechenzeit wiederum, was dazu führt, daß ein bißchen tiefer gerechnet werden kann.

Zukunftsträchtig ist auch die Verwendung von Multiprozessor-Systemen; hier werden verschiedene Aufgaben an mehrere Prozessoren delegiert, die von einem Zentralprozessor koordiniert werden. Die Versuche mit solchen Monstern laufen und es stellt sich für die Entwickler nur die Frage, wieviele Prozessoren noch sinnvoll miteinander kombiniert werden können.

Und auf der Software-Seite? Hier liegt der Fortschritt tatsächlich nur im Detail. Solange Programme vom Typ Shannon-A (Brute Force) geschrieben werden, müssen Algorithmen wie Alpha-Beta-Prozedur, Forward Pruning, Ruhesuche und heuristische Verfahren angewandt werden, um erfolgreich zu sein.

Das BASIC-Schachprogramm in diesem Buch repräsentiert in dieser Hinsicht den Stand der Technik: Alle Verfahren, die sich in BASIC realisieren lassen, sind implementiert und optimiert - wenn Sie sich mit diesem Programm auseinandersetzen, werden Sie verstehen lernen, wie 90% aller Schachprogramme - vom Weltmeisterprogramm bis zum billigen Programm für den Homecomputer - funktionieren.

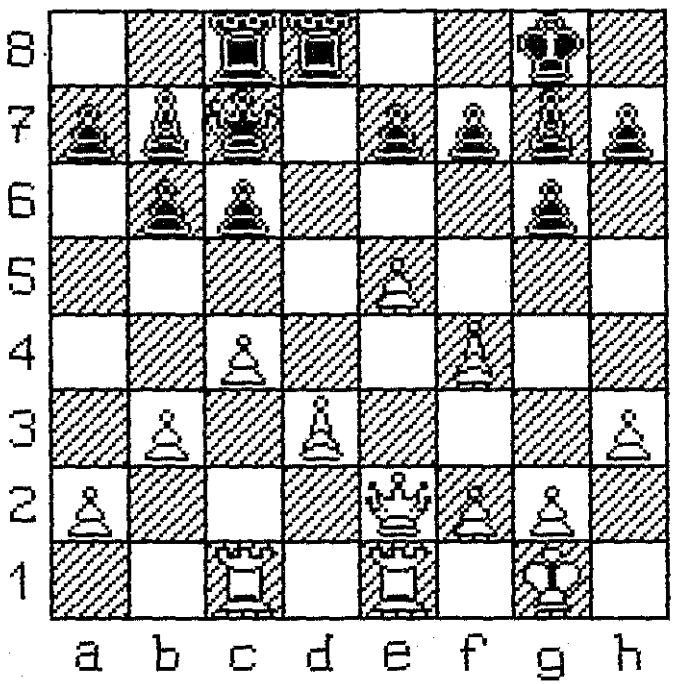
In diesem Sinn stellt das Große Computerschach-Buch den augenblicklichen Stand der Entwicklung dar. Wenn man Ex-Weltmeister Michail Botwinnik glauben darf, ist der entscheidende Durchbruch noch nicht geschafft: "Erst wenn Computerprogramme zielgerichtete Pläne entwickeln und diese durchführen können, nähert sich das Computerschach dem menschlichen Spiel."

Manchmal hat man allerdings das Gefühl, die Rechner wären schon so weit - ein Beispiel, das bei der Partie eines der Autoren gegen SARGON III auf einem Commodore PC entstand, nährt den Verdacht.

Nach einigem Hin- und Hergeschiebe ergab sich folgende Stellung:

Gespielt am 2. November 1985
Weiß: SARGON III
Schwarz: Rainer Bartel

Die Stellung:



Ziemlich langweilig - bis hierher war auch noch nichts Aufregendes passiert. Aber dann dachte sich SARGON etwas aus:

18 Te1-f1 ...

Nanu, sollten die Türme nicht besser auf der D-Linie verdoppelt werden?

18 ... Td8-d4
19 Lf4-g3 c6-c5

So, Schwarz steht ganz schön: Die Türme werden sich auf der D-Linie verdoppeln, der Läufer auf B7 hat die Diagonale und auch der Läufer auf G7 beherrscht die Diagonale.

20 f2-f4 Tc8-d8
21 f4-f5 ...

Was will er? Mit Bauer nach E6 die schwarze Dame demaskieren und dann mit f7:+ den König bedrohen? Das wäre kein Problem (21. ... e7 22. e6 Dc6 23. e6 e6 24. Df2 Tf8 - Gefahr vorbei). Nur auf F5 zu schlagen, wäre nicht so gut.

21 ... Lg7-f8
22 f5xg6 h7xg6
23 Tf1xf7 ...

Das ist ein Hammer! Natürlich darf der Turm nicht genommen werden (23. ... Kf7: 24. e6+ ... mit Dameverlust!). Aber was dann?

23 ... Dc7-c6

Droht mit Schach auf G2.

24.	Tf7-f2	e7-e6
25	Ld3xg6	Td4-d2
26	Lg6-f7+	Kg8-h8
27	De2-h5+	Kh8-g7
28	Dh5-g6+	Kg7-h8
29	Dg6-g8++	

SARGON spielt, als ob er im 18. Zug den Plan gefaßt hätte, dem Schwarzen mit einem Angriff auf F7 den Garaus zu machen. Analysiert man seine Züge, stellt man fest, daß das Programm nie tiefer als 4 Halbzüge blicken mußte, um die Fortsezung zu finden. Trotzdem: Bravo, Computer!

2. Wie lernt der Computer gewinnen - kleine Theorie der Strategiespiele

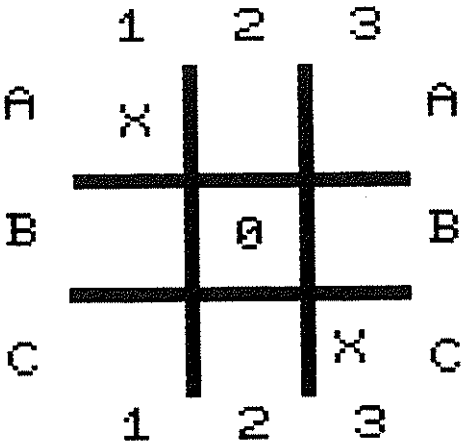
Strategie hat heutzutage sehr wenig mit Krieg zu tun; man spricht von sozialen Strategien, Marktstrategien, sogar Wissenschaftler, die ein neues Medikament erforschen, legen sich für ihre Arbeit ein Strategie zu recht.

Der Duden definiert den Begriff so:
"Strategie, die: Kriegskunst; genau geplantes Vorgehen"

Strategiespiele sind folgerichtig solche, bei denen man nur gewinnen kann, wenn man sorgfältig plant und die gefundenen Pläne konsequent in die Tat umsetzt. Das ein Schachspieler ohne solche Handeln über ein Stümper-Dasein nicht hinauskommt, liegt auf der Hand; was macht aber der arme Computer, den man dazu zwingen will, ein Strategiespiel erfolgreich zu absolvieren? Wo er doch bekanntermaßen nicht dazu in der Lage ist, Pläne zu schmieden? Nun, er muß angeleitet werden.

Als erstes muß er sich vorstellen können, was die Figuren überhaupt sind, wie groß das Spielfeld ist, wo die Figuren stehen und was Zweck des Spiels ist. Im 3. und 4. Kapitel werden wir dies alles einem Computer beibringen, der daraufhin das Schachspiel beherrscht; an dieser Stelle wollen wir zunächst bei den Grundlagen beginnen und sozusagen als Fingerübung ein anderes, sehr simples Spiel auf den Computer bringen: TIC TAC TOE.

Es ist eines der ältesten Spiele überhaupt - schon die alten Ägypter gaben ihren Toten kleine Täfelchen mit, auf denen das klassische Symbol:



zu sehen ist. Die Regeln haben sich seitdem wohl kaum verändert.

1) Gespielt wird auf einem Brett mit 3 mal 3 Feldern.
2) Die Spieler setzen abwechselnd.
3) Steine dürfen nur auf leere Felder gesetzt werden.
4) Gewonnen hat, wer 3 eigene Steine auf 3 aneinander grenzenden Feldern hat.
5) Das gilt für die Senkrechten, Waagerechten und Diagonalen.
6) Ein Spiel dauert maximal neun Züge lang.
7) Hat nach neun Zügen kein Spieler gewonnen, endet die Partie unentschieden.

Üblicherweise werden die Steine durch Kreuze ("X") und Kringel ("O") dargestellt.

Was muß das Programm zuerst wissen?

2.1 Kein Brett im Rechner - die Spieldarstellung

Das Programm muß zunächst einen Begriff vom Spielfeld bekommen; für jedes Feld des Brettes muß eine Speicherstelle reserviert sein, die je nachdem ein Stein der einen oder anderen

Partei oder gar keiner auf dem Feld ist, einen anderen Wert enthält. Man könnte versuchen diese Variable zweidimensional als

BR(3,3)

zu dimensionieren. Für spätere Aufgaben ist ein eindimensionales Array brauchbarer. Also fängt das Programm so an:

```
1010 DIM BR(9)   :  REM BRETT
```

Diese Variable muß initialisiert werden, denn genau wie das Brett zu Anfang leer ist, enthält auch BR(9) zuerst nur Nullen.

```
1110 FOR I = 1 TO 9
1120 :   BR(I) = 0
1160 NEXT
```

Und wie sehen für das Programm die Steine aus? Das wird bereits bei der Anfangsabfrage festgelegt:

```
1510 CLS   :   REM  PRINT CHR$(147);
1520 PRINT  "WER SOLL ANFANGEN?"
1530 INPUT  "(M)ENSCH ODER (C)OMPUTER ";IN$
1540 IF  IN$ = "M"  THEN  F = 5  :   GOTO  2000
1550 IF  IN$ = "C"  THEN  F = 1  :   GOTO  2000
1560 GOTO  1510
```

Eine 1 soll also einen Stein des Computers repräsentieren, eine 5 den des Spielers. Diese Werte haben wir gewählt, weil man bei dieser Konstellation sehr leicht feststellen kann, ob eine Dreierreihe vorhanden ist. Schauen wir uns einmal folgende Tabelle an. Sie enthält alle möglichen Varianten der Besetzung einer Dreierreihe und dazu die Summe der entsprechenden Variablen:

```
X.. = 1  /   .X. = 1  /   ..X = 1
O.. = 5  /   .O. = 5  /   ..O = 5
XX. = 2  /   .XX = 2  /   X.X = 2
OO. =10  /   .OO =10  /   O.O =10
XO. = 6  /   OX. = 6  /   X.O = 6
O.X = 6  /   .XO = 6  /   .OX = 6
XXX = 3 (alle Felder besetzt)
OOO =15 (alle Felder besetzt)
XXO = 7  /   OXX = 7  /   XOX = 7
OOX =11  /   XOO =11  /   OXO =11
```

Man sieht, daß bei der gewählten Darstellung der Steine immer eindeutig festgestellt werden kann, wie eine Dreierreihe besetzt ist. Würde man - was auch naheliegt - 1 und -1 wählen, ergäbe sich z.B. folgendes Dilemma:

```
XXO = (2*1)+(1*(-1))=1 aber auch
X.. = (1*1)+(2*0)=1
```

2.2 Was darf das Programm - die Spielregeln

Drei Programmteile beinhalten die Spielregeln:

- das Hauptprogramm;
- die Abfrage auf das Spielende und
- der Zuggenerator.

Das Hauptprogramm sorgt dafür, daß die Spieler abwechselnd ziehen:

```
2010 Z = Z + 1                          :  REM  ZUGZAEHLER
2020  IF  Z = 10  THEN  2600            :  REM  SPIELENDE
2030  IF  F = 5  THEN   GOSUB  3000    :  REM  SPIELERZUG
2040  GOSUB  2500                       :  REM  SIEG?
2050  IF  F = 1  THEN   GOSUB  4000    :  REM  COMPUTERZUG
```

```
2060  GOSUB  2500                    :  REM  SIEG?
2070 F = 6 - F                       :  REM  FARBWECHSEL
2080  GOTO  2010                     :  REM  NAECHSTER ZUG
```

Solange die Maximalzahl an Zügen (nämlich 9) nicht erreicht ist, bleibt das Programm in seiner Hauptschleife. Je nachdem, welchen Wert F (=Farbe) annimmt, wird eines der beiden Unterprogramme für den Spieler- bzw. Computerzug angesprungen; die Formel für den Farbwechsel funktioniert so:

```
Für F=1 wird durch      Für F=5 wird durch
      F=6-F                   F=6-F
       F=5                     F=1
```

In den Zeilen 2040 und 2060 wird ein Unterprogramm angesprungen, daß prüft, ob der gerade erfolgte Zug zum Sieg einer Partei geführt hat.

```
2510  GOSUB  7000
2520  IF  A < 50  AND  B < 50  THEN  RETURN
2530  GOSUB  6000
2540  IF  A > 50  THEN   PRINT  "COMPUTER HAT GEWONNEN!"
      :   END
2550  IF  B > 50  THEN   PRINT  "MENSCH HAT GEWONNEN!"
      :   END
```

Hier taucht der Sprung zur Zeile 7000 auf, den wir aber zunächst nicht untersuchen wollen. Das Programm beachtet aber auch die Regel für das Unentschieden:

```
2010 Z = Z + 1                       :  REM  ZUGZAEHLER
2020  IF  Z = 10  THEN  2600         :  REM  SPIELENDE
```

Wenn die Maximalanzahl an Zügen überschritten ist und bislang keine Seite gewonnen hat, verzweigt das Programm zur Zeile 2600 - das Spiel endet unentschieden.

```
2610 PRINT "UNENTSCHIEDEN ..." : END
```

Bevor wir uns den Zuggenerator ansehen, müssen wir uns mit der Eingabe des Spielerzuges beschäftigen.

```
3010 CLS : REM PRINT CHR$(147);
3020 GOSUB 6000 : REM BRETTAUSGABE
3030 INPUT "DEIN ZUG ";IN$ : REM ZUGEINGABE
3040 IF LEN (IN$) < > 2 THEN 3010
3050 Z1$ = LEFT$ (IN$,1) : REM ENTSCHLUESSELN
3060 Z2$ = RIGHT$ (IN$,1)
3070 Z2 = VAL (Z2$) : IF Z2 > 3 THEN 3010
3080 Z1 = ASC (Z1$) - 65
     : IF Z1 < 0 OR Z1 > 2 THEN 3010
3090 ZU = (3 * Z1) + Z2
3100 IF BR(ZU) < > 0 THEN 3010 : REM ZUG LEGAL?
3110 BR(ZU) = F : REM ZUG AUSFUEHREN
3120 RETURN
```

Für den menschlichen Spieler ist es komfortabler, denn Zug in einer Notation, die aus Zeilen- und Spaltenkoordinate besteht, einzugeben. Für das Programm muß diese Eingabe entschlüsselt werden. Eingegeben wird ein String, bestehend aus zwei Zeichen. Ob das zweite Zeichen eine Ziffer ist, wird nicht gesondert überprüft, nur Zahlen, die größer als 3 sind, werden zurückgewiesen. Das erste Zeichen stellt die Zeilenkoordinate dar uns muß einer der Buchstaben A,B oder C sein. Um das festzustellen, benutzen wir den Zahlenwert des ASCII-Codes dieses Zeichens:

A=65, B=66 und C=67 (gilt für die meisten Computer).

Daraus ergibt sich, daß

A für die Reihe 0,
B für die Reihe 1 und
C für die Reihe 2 steht.

Die Entschlüsselung wird dadurch unkompliziert. Wenn das Programm weiß, welches Feld der Spieler besetzen möchte, prüft es in Zeile 3100, ob das Feld leer ist, dann ist nämlich BR(ZU)=0.

Das Programm muß schon vor der Suche nach dem besten Zug wissen, welche Felder es überhaupt besetzen darf. Das errechnet der Zuggenerator.

```
8110  FOR  I = 1  TO  9
8120  : ZL(I) = 0
8130  NEXT
8140 ZZ = 0
```

Mit diesen Zeilen wird die Liste, die später die Ziffern für alle besetzbaren Felder enthält, initialisiert. Auch die Variable, die angibt, wie lang die Zugliste ist, muß auf 0 gesetzt werden.

```
8210  FOR  I = 1  TO  9
8220  :  IF  BR(I) < > 0  THEN  8250  :   REM  BESETZT?
8230  : ZZ = ZZ + 1                   :   REM  INDEX HOEHER
8240  : ZL(ZZ) = I                    :   REM  ZUG EINTRAGEN
8250  NEXT
8260  RETURN
```

Nun werden alle neun Felder nacheinander abgetastet - jedes leere Feld wird mit seiner laufenden Nummer in die Liste ZL(ZZ) eingetragen, ZZ wird für jeden neuen Eintrag um 1 erhöht. Ein Beispiel:

Die Stellung

```
X.O
OOX
...
```

ergibt die Zugliste

1. 2
2. 7
3. 8
4. 9

2.3 Vorwärts und zurück - das Entscheidungsverfahren

Was heißt das eigentlich: Das Programm soll lernen zu gewinnen? Wir wollen das Programm dazu bringen, Züge auszuführen, die den Gewinn herbeiführen können. Bis jetzt könnte der Teil für den Computerzug ja so aussehen:

```
4010  CLS                 :  REM  PRINT CHR$(147);
4020  GOSUB  6000         :  REM  BRETTAUSGABE
4030  GOSUB  8000         :  REM  ZUGGENERATOR

> 4040 ZU =   INT (RND(0) * ZZ) + 1
> 4050 BR(ZU) = F
> 4460  RETURN
```

Die Zufallszahl RND(0) würde einen beliebigen Zug aus der Liste nehmen und ausführen. Sie können sich leicht davon überzeugen, daß mit diesem Auswahlverfahren das Programm leicht zu schlagen ist.

Was muß es denn wissen, um 'gute' Züge zu machen? Das Programm muß Positionen, die aus der bei Beginn der Suche

vorliegenden Stellung entstehen, bewerten können - ihnen einen Wert zuordnen können, der besagt, ob ein Zug zum Gewinn führt.

Das ist für einen Gewinn in der nächsten Stellung recht einfach zu finden. Wir schreiben zunächst folgende Bewertungsfunktion:

```
7210  FOR  I = 1  TO  8
7220  : SI(I) = 0
7230  NEXT
```

In der Variablen SI(8) werden die Summen aller möglichen Dreierreihen erfaßt, die sich folgendermaßen berechnen:

```
7310 SI(1) = BR(1) + BR(2) + BR(3)
7320 SI(2) = BR(4) + BR(5) + BR(6)
7330 SI(3) = BR(7) + BR(8) + BR(9)
7340 SI(4) = BR(1) + BR(4) + BR(7)
7350 SI(5) = BR(2) + BR(5) + BR(8)
7360 SI(6) = BR(3) + BR(6) + BR(9)
7370 SI(7) = BR(1) + BR(5) + BR(9)
7380 SI(8) = BR(3) + BR(5) + BR(7)

> 7390  RETURN
```

Wir haben oben schon gesehen, daß die Summen der jeweils drei relevanten Felder eindeutig auf die Besetzung hinweisen. Hier wird es nun deutlich; ist einer der Werte SI(1) bis SI(8) gleich 3, ist eine Gewinnstellung für den Spieler erreicht, ist SI(1) bis SI(8) gleich 15, hat der Computer gewonnen.

Der Programmteil Computerzug könnte also so aussehen:

```
4010  CLS                 :  REM  PRINT CHR$(147);
4020  GOSUB  6000         :  REM  BRETTAUSGABE
4030  GOSUB  8000         :  REM  ZUGGENERATOR

> 4040  FOR  I = 1  TO  ZZ
> 4050 ZU = ZL(I)
> 4060 BR(ZU) = F
> 4070  GOSUB 7000
> 4080 J = 1
> 4090  IF SI(J) < > 3  THEN  4110
> 4100  RETURN
> 4110 J = J + 1
> 4120 BR(ZU) = 0
> 4130  IF J < 8  THEN  4090
> 4140  NEXT
```

Das Programm würde den Zug ausführen, der unmittelbar zum Gewinn führt. Leider entsteht diese Situation selten, so daß nach dieser Bewertung noch einmal der Zufallszug gewählt werden müßte, wenn kein Gewinnzug gefunden wurde.

Wir führen den Suchbaum ein. Dieser Begriff bezeichnet eine Methode, mit der aus einer Ursprungsstellung alle möglichen Folgestellungen erzeugt werden, aus denen dann wieder die möglichen Folgezüge erzeugt werden usw. bis zu einer vorher definierten 'Tiefe'. Man spricht von Tiefe, weil ein Suchbaum sich von der Wurzel aus nach unten verzweigt. Stellen wir uns einmal den denkbar einfachsten Suchbaum für eine TIC TAC TOE-Stellung vor:

Ursprungsstellung:

```
XOX
OXO
.O.
```

Folgestellungen:

```
XOX      XOX
OXO      OXO
XO.      .OX
```

Beide Fortsetzungen führen zum Gewinn für den "X"-Spieler. Dieser Suchbaum hat die Tiefe 1, weil 1 Folgezug untersucht wird. Das Verfahren, einen Zug weiterzublicken, haben wir ja schon in unserem bisher entwickelten Suchprogramm verwendet; wir erzeugen ja alle legalen Züge, führen sie probehalber aus, stellen fest, ob sie einen Gewinn herbeiführen und nehmen sie wieder zurück.

Nicht viel komplizierter ist die Anwendung eines Suchbaums der Tiefe 2. Zuerst werden in der Urspungsstellung alle vom Zuggenerator in die Zugliste eingetragenen Züge ausgeführt, dann wird in den enststandenen Stellungen der Zuggenerator aufgerufen und die Züge aus der Zugliste wieder ausgeführt. Die dann vorliegenden Stellungen werden bewertet.

In der Praxis werden für jeden Zug, der in der Ursprungsstellung möglich ist, alle möglichen Folgezüge ausgeführt, bevor der nächste ausgeführt wird. Als Ablaufplan sieht dies so aus:

```
(Stellung retten)
(Zuggenerator aufrufen)
(Zugliste retten)
(Zuglisten-Index retten)
(Zugzähler auf 1 setzen)
(1) (Zug ausführen)
:   (Zuggenerator aufrufen)
:   (Zugzähler auf 1 setzen)
:   (2) (Zug ausführen)
:   (Zug bewerten)
:   (Wert merken)
```

```
:   (Zug zurücknehmen)
:   (Zugzähler erhöhen)
:   (falls Zugzähler kleiner Maximum --> zurück zu (2)
(Zug zurücknehmen)
(Zugzähler erhöhen)
(falls Zugzähler kleiner Maximum --> zurück zu (1)
(Ende)
```

Mit diesem Verfahren bekämen wir je einen Wert für jede Stellung, die sich bei Tiefe 2 ergibt. Und so sähe das als Listing aus:

```
4110  FOR  I = 1  TO  9
4120 ST(0,I) = BR(I)  :   REM  STELLUNG RETTEN
4130  NEXT
4140  GOSUB  8000      :   REM  ZUGGENERATOR
4150  FOR  N = 1  TO  ZZ
4160 ZG(N) = ZL(N)     :   REM  ZUGLISTE RETTEN
4170  NEXT
4180 YY = ZZ           :   REM  ZUGLISTEN-INDEX RETTEN

4210  FOR  N = 1  TO  YY  :   REM  SCHLEIFE TIEFE 1
4220 ZU = ZG(N)           :   REM  ZUG AUS LISTE
4230 BR(ZU) = F           :   REM  ZUG AUSFUEHREN

4330  :    GOSUB  8000              :   REM  ZUGGENERATOR
4340  :    FOR  M = 1  TO  ZZ
4350  :    :    ZU = ZL(M)          :   REM  ZUG AUS LISTE
4360  :    :    BR(ZU) = F          :   REM  ZUG AUSFUEHREN
4370  :    :     GOSUB  7000        :   REM  BEWERTUNG

4390  :    :    W = GW              :   REM  WERT MERKEN
4400  :    :    BR(ZU) = 0          :   REM  ZUG ZURUECK
4410  :    NEXT
```

Halt! Eines haben wir vergessen: Wenn wir Züge der Tiefe 2 erzeugen, müssen wir so tun, als setzten wir die gegnerischen Steine. Wir müssen die Farbe wechseln.

```
4310  :    F = 6 - F                    :  REM  FARBWECHSEL
```

Was nutzen uns aber die schönsten Werte, wenn wir sie nur für Tiefe 2 wissen? Es wäre besser, wenn wir aus den Werten für die Tiefe 2 darauf schließen könnten, welche Stellung der Tiefe 1 die beste ist.

Dazu führen wir die Minimax-Prozedur ein. Sie folgt aus dem Minimax-Theorem und das ist ganz einfach: Man kann davon ausgehen, daß der Gegner in einer Situation den für ihn jeweils besten Zug macht. Ein guter Zug des Gegners ergibt einen niedrigeren Wert für die Stellung, als ein schlechter, weil die Stellungswerte immer aus der Sicht des jeweils Ziehenden gesehen werden. Ein Beispiel (mit fiktiven Werten):

```
X am Zug

Tiefe 0:                    X.O
                            O.X
                            X.O


Tiefe 1:        a) XXO      b) X.O      c) X.O
                   O.X         OXX         O.X
                   X.O         X.O         XXO


Tiefe 2:        XXO  XXO    XOO  X.O    XOO  X.O
                OOX  O.X    OXX  OXX    O.X  OOX
                X.O  XOO    X.O  XOO    XXO  XXO

Wert:           -11   -9    +1    0     -1   +1
```

Jetzt kann der Spieler mit den "X"en davon ausgehen, daß in Tiefe 2 der Gegner jeweils den Zug macht, der für ihn am

besten ist, also den Wert am niedrigsten hält - man sagt: Der Gegner minimiert.

Die Minimumwerte sind in Tiefe 2:

Für Stellung a)	-11
Für Stellung b)	0
Für Stellung c)	-1

Diese Werte können wir der Tiefe 1 zuordnen. Da ein eigener Zug zu den Stellungen der Tiefe 1 führt, wird der Spieler den für ihn besten Zug wählen; d.h. den Zug, der den für ihn besten Wert erzielt. Im Beispiel also den Zug, der zu Stellung b) führt, wodurch der Wert 0 mindestens erreicht wird. Der Spieler hat bei Tiefe 1 maximiert.

Ins Programm eingebaut sieht das so aus (hier nicht als einzelne Zeilen gezeigt, sondern im Zusammenhang mit den Zeilen für den Suchbaum):

```
4210  FOR  N = 1  TO  YY  :   REM  SCHLEIFE TIEFE 1
4220 ZU = ZG(N)           :   REM  ZUG AUS LISTE
4230 BR(ZU) = F           :   REM  ZUG AUSFUEHREN
4240  FOR I = 1 TO 9
4250 ST(1,I) = BR(I)      :   REM  STELLUNG MERKEN
4260  NEXT
4300  REM  * TIEFE 2 *
4301  REM
4302  :
4310  :    F = 6 - F                    :   REM  FARBWECHSEL
4320  :    W = 0                        :   REM  WERT INIT.
4330  :    GOSUB  8000                  :   REM  ZUGGENERATOR
4340  :    FOR  M = 1  TO  ZZ
4350  :    :    ZU = ZL(M)          :   REM  ZUG AUS LISTE
4360  :    :    BR(ZU) = F          :   REM  ZUG AUSFUEHREN
4370  :    :     GOSUB  7000        :   REM  BEWERTUNG
4380  :    :     IF  GW > W  THEN  4400 :   REM  BESTER WERT?
4390  :    :    W = GW              :   REM  WERT MERKEN
4400  :    :    BR(ZU) = 0          :   REM  ZUG ZURUECK
```

```
4410  :     NEXT
4420  :     BW(N) = W                    :   REM  WERT MERKEN
4430  :     FOR  I = 1  TO  9
4440  :   :   BR(I) = ST(0,I)        :   REM  STELLUNG REKON.
4450  :     NEXT
4460  :     F = 6 - F                    :   REM  FARBWECHSEL
4470  NEXT
4495  :
4496  :
4497  :
4498  :
4499  REM
4500  REM  * MINIMAX *
4501  REM
4502  :
4510 BZ = ZG(1)                    :   REM  BESTEN ZUG INIT.
4520 W = 0                         :   REM  WERT INITIALISIEREN
4530  FOR  N = 1  TO  ZZ
4540  IF  BW(N) < W  THEN  4570    :   REM  NEUER BESTER WERT?
4550 W = BW(N)                     :   REM  BESTEN WERT MERKEN
4560 BZ = ZG(N)                    :   REM  BESTEN ZUG MERKEN
4570  NEXT
4580 BR(BZ) = F                    :   REM  ZUG AUSFUEHREN
4590  RETURN
```

Damit hat das Programm bei einer Suche mit der Tiefe 2 den für sich optimalen Zug gefunden. Solange nur zwei Züge tief gesucht wird, kann man die Tiefen einfach ineinander verschachteln - will man tiefer suchen, muß man zu rekursiven Routinenaufrufen greifen. Das lohnt sich aber bei einem Spiel wie TIC TAC TOE nicht. Erst bei so komplexen Spielen wie Schach muß mit größerer Tife gesucht werden. Wie man das macht, wird im Kapitel 3. erklärt.

Unser TIC TAC TOE-Programm ist auch so stark genug - es wird normalerweise jedes Spiel mindestens unentschieden halten. Dafür muß allerdings noch die Bewertungsfunktion verbessert werden, bisher wird ja nur festgestellt, ob eine Gewinnstellung vorliegt.

Bei dem gezeigten Teil der Bewertung werden die 8 möglichen Dreierreihen bereits aufsummiert. Es sollen nun aus den sich ergebenden Werten Schlüsse gezogen werden. Dabei ist es sinnvoll, alle potentiellen Dreier für beide Farben getrennt zu berechnen, die Werte für das Programm werden in A summiert, die für den Spieler in B. Der Wert GW ein der Stellung ergibt sich aus der Differenz von A und B.

```
7110 A = 0  :  B = 0
7120 GW = 0

7610 GW = A - B
```

Welche Werte sollen nun aus welchen Summen für einen Dreier errechnet werden. Empirische Untersuchungen haben folgendes Rezept ergeben:

1) Der Dreier ist von den Steinen einer Farbe besetzt = 100
2) Der Dreier ist noch möglich, zwei Felder sind belegt = 8
3) Der Dreier ist noch möglich, ein Feld ist belegt = 2
4) Der Dreier ist noch ganz unbelegt = 1

Berechnen wir einmal folgende Position:

```
XO.
OO.
...
```

Die Dreier:

1) XO. --> Für A: 0 Punkte / Für B: 0 Punkte
2) ... --> Für A: 1 Punkt / Für B: 1 Punkt
3) OO. --> Für A: 0 Punkte / Für B: 8 Punkte
4) X.O --> Für A: 0 Punkte / Für B: 0 Punkte
5) O.O --> Für A: 0 Punkte / Für B: 8 Punkte
6) ... --> Für A: 1 Punkt / Für B: 1 Punkt
7) X.. --> Für A: 2 Punkte / Für B: 0 Punkte
8) ..O --> Für A: 0 Punkte / Für B: 2 Punkte

SUMMEN --> Für A: 4 Punkte / Für B: 20 Punkte

GW = A - B GW = -16

Programmiert ergeben sich daraus die Zeilen:

```
7410 FOR I = 1 TO 8
7420 : IF SI(I) = 0 THEN A = A + 1 : B = B + 1
7430 : IF SI(I) = 1 THEN A = A + 2
7440 : IF SI(I) = 2 THEN A = A + 8
7450 : IF SI(I) = 3 THEN A = A + 100
7460 : IF SI(I) = 5 THEN B = B + 2
7470 : IF SI(I) = 10 THEN B = B + 8
7480 : IF SI(I) = 15 THEN B = B + 100
7490 NEXT
```

Wir können aber noch einen zweiten Aspekt in die Bewertung einfliessen lassen, nämlich die Belegung der Felder durch den jeweiligen Spieler. Betrachtet man sich einmal, welche Felder einen Dreier ergeben können, wird deutlich, daß manche Felder wertvoller sind als andere:

Die acht möglichen Dreier bestehen aus folgenden Feldern:

1) 1+2+3
2) 4+5+6
3) 7+8+9
4) 1+4+7
5) 2+5+8
6) 3+6+9
7) 1+5+9
8) 3+5+7

Die Felder sind ungleich oft in den Dreiern vertreten:

Feld 1 ist in 3 Dreiern.
Feld 2 ist in 2 Dreiern.
Feld 3 ist in 3 Dreiern.
Feld 4 ist in 2 Dreiern.
Feld 5 ist in 4 Dreiern.
Feld 6 ist in 2 Dreiern.
Feld 7 ist in 3 Dreiern.
Feld 8 ist in 2 Dreiern.
Feld 9 ist in 3 Dreiern.

Dementsprechend verteilen wir folgenden Feldwerte:

404
080
404

Oder im Listing:

```
1020 DIM FW(9)    :  REM FELDWERTE
1110 FOR I = 1 TO 9
1120 :    READ FW(I)
1160 NEXT
1200 DATA 4,0,4,0,8,0,4,0,4
```

Diese Feldwerte werden in der Bewertungsfunktion einfach aufaddiert:

```
7510 FOR I = 1 TO 9
7520 : IF BR(I) = 1 THEN A = A + FW(I)
7530 : IF BR(I) = 5 THEN B = B + FW(I)
7540 NEXT
```

Mit diesem Verfahren erhält jede Position eine signifikante Bewertung.
Eigentlich ist unser TIC TAC TOE damit fertig. Allerdings haben wir nicht alle Programmzeilen hier im Text gezeigt und erläutert. Deshalb folgt gleich das komplette Listing am Stück.

Sicher ist diese Programm nicht besonders elegant oder schnell, Sie werden aber beim Abtippen nachvollziehen können, wie die grundlegenden Algorithmen für ein Strategiespiel auf Computern funktionieren. Sollten Sie im Kapitel 3. über die theoretischen Grundlagen der Schachprogrammierung Verständnisschwierigkeiten haben, können Sie zu diesem Kapitel zurückkehren und versuchen, anhand des TIC TAC TOE-Programmes die Algorithmen besser zu verstehen.

Und wenn Sie Lust haben können Sie es auch noch verbessern. Programmierübungen an diesem einfachen Beispiel werden Ihnen helfen, das Schachprogramm ebenfalls nach eigenen Vorstellungen zu verändern. Dazu einige Vorschläge:

- installieren Sie eine größere Suchtiefe.
- probieren Sie weitere Bewertungsroutinen.
- schreiben Sie das Programm für 4 mal 4 Felder um.

Das vollständige Listing TIC TAC TOE:

```
1000  REM  *** VARIABLEN ***
1001  REM
1002  :
1010  DIM  BR(9)    :   REM  BRETT
1020  DIM  FW(9)    :   REM  FELDWERTE
1030  DIM  BW(9)    :   REM  BEWERTUNG
1040  DIM  SI(8)    :   REM  DREIER
1050  DIM  ST(1,9)  :   REM  STACK
1095  :
1096  :
1097  :
1098  :
1099  REM
1100  REM  * INITIALISIEREN *
1101  REM
1102  :
1110  FOR  I = 1  TO  9
1120  :     BR(I) = 0  :  BW(I) = 0  :   READ  FW(I)
1130  :     FOR  J = 0  TO  1
1140  :     :    ST(J,I) = 0
1150  :     NEXT
1160  NEXT
1170  FOR  I = 1  TO  8
1180  :    SI(I) = 0
1190  NEXT
1200  DATA  2,0,2,0,6,0,2,0,2
1295  :
1296  :
1297  :
1298  :
1299  REM
1300  REM  * STEINE AUF DEM BRETT *
1301  REM
1302  :
1310 BR$(0) = "."
1320 BR$(1) = "X"
1330 BR$(5) = "O"
```

```
1495  :
1496  :
1497  :
1498  :
1499  REM
1500  REM  *** ANFANGSABFRAGE ***
1501  REM
1502  :
1510  CLS   :   REM  PRINT CHR$(147);
1520  PRINT  "WER SOLL ANFANGEN?"
1530  INPUT  "(M)ENSCH ODER (C)OMPUTER ";IN$
1540  IF  IN$ = "M"  THEN  F = 5  :   GOTO  2000
1550  IF  IN$ = "C"  THEN  F = 1  :   GOTO  2000
1560  GOTO  1510
1995  :
1996  :
1997  :
1998  :
1999  REM
2000  REM  *** HAUPTPROGRAMM ***
2001  REM
2002  :
2010 Z = Z + 1                        :   REM  ZUGZAEHLER
2020  IF  Z = 10  THEN  2600          :   REM  SPIELENDE
2030  IF  F = 5  THEN   GOSUB  3000   :   REM  SPIELERZUG
2040  GOSUB  2500                     :   REM  SIEG?
2050  IF  F = 1  THEN   GOSUB  4000   :   REM  COMPUTERZUG
2060  GOSUB  2500                     :   REM  SIEG?
2070 F = 6 - F                        :   REM  FARBWECHSEL
2080  GOTO  2010                      :   REM  NAECHSTER ZUG
2495  :
2496  :
2497  :
2498  :
2499  REM
2500  REM  *** SIEG?
2501  REM
2502  :
2510  GOSUB  7000
2520  IF  A < 50  AND  B < 50  THEN  RETURN
```

```
2530  GOSUB  6000
2540  IF  A > 50  THEN   PRINT  "COMPUTER HAT GEWONNEN!"
      :   END
2550  IF  B > 50  THEN   PRINT  "MENSCH HAT GEWONNEN!"
      :   END
2595  :
2596  :
2597  :
2598  :
2599  REM
2600  REM  *** UNENTSCHIEDEN?
2601  REM
2602  :
2610  PRINT  "UNENTSCHIEDEN ..."  :   END
2995  :
2996  :
2997  :
2998  :
2999  REM
3000  REM  *** SPIELERZUG ***
3001  REM
3010  CLS   :   REM  PRINT CHR$(147);
3020  GOSUB  6000                    :   REM  BRETTAUSGABE
3030  INPUT  "DEIN ZUG ";IN$         :   REM  ZUGEINGABE
3040  IF   LEN (IN$) <  > 2  THEN  3010
3050 Z1$ =  LEFT$ (IN$,1)            :   REM  ENTSCHLUESSELN
3060 Z2$ =  RIGHT$ (IN$,1)
3070 Z2 =  VAL (Z2$)  :   IF  Z2 > 3  THEN  3010
3080 Z1 =  ASC (Z1$) - 65
     :   IF  Z1 < 0  OR  Z1 > 2  THEN  3010
3090 ZU = (3 * Z1) + Z2
3100  IF  BR(ZU) <  > 0  THEN  3010 :  REM  ZUG LEGAL?
3110 BR(ZU) = F                      :   REM  ZUG AUSFUEHREN
3120  RETURN
3995  :
3996  :
3997  :
3998  :
3999  REM
4000  REM  *** COMPUTERZUG ***
```

```
4001  REM
4010  CLS                 :   REM  PRINT CHR$(147);
4020  GOSUB  6000         :   REM  BRETTAUSGABE
4030  PRINT  "COMPUTER RECHNET"
4095  :
4096  :
4097  :
4098  :
4099  REM
4100  REM  * VORBEREITUNGEN *
4101  REM
4102  :
4110  FOR  I = 1  TO  9
4120 ST(0,I) = BR(I)  :   REM  STELLUNG RETTEN
4130  NEXT
4140  GOSUB  8000      :   REM  ZUGGENERATOR
4150  FOR  N = 1  TO  ZZ
4160 ZG(N) = ZL(N)     :   REM  ZUGLISTE RETTEN
4170  NEXT
4180 YY = ZZ           :   REM  ZUGLISTEN-INDEX RETTEN
4195  :
4196  :
4197  :
4198  :
4199  REM
4200  REM  * TIEFE 1 *
4201  REM
4202  :
4210  FOR  N = 1  TO  YY  :   REM  SCHLEIFE TIEFE 1
4220 ZU = ZG(N)           :   REM  ZUG AUS LISTE
4230 BR(ZU) = F           :   REM  ZUG AUSFUEHREN
4240  FOR I = 1 TO 9
4250 ST(1,I) = BR(I)      :   REM  STELLUNG MERKEN
4260  NEXT
4300  REM  * TIEFE 2 *
4301  REM
4302  :
4310  :    F = 6 - F                    :   REM  FARBWECHSEL
4320  :    W = 0                        :   REM  WERT INIT.
4330  :    GOSUB  8000                  :   REM  ZUGGENERATOR
```

```
4340  :     FOR  M = 1  TO  ZZ
4350  :     :    ZU = ZL(M)            :   REM  ZUG AUS LISTE
4360  :     :    BR(ZU) = F            :   REM  ZUG AUSFUEHREN
4370  :     :     GOSUB  7000          :   REM  BEWERTUNG
4380  :     :     IF  GW > W  THEN  4400 :   REM  BESTER WERT?
4390  :     :    W = GW                :   REM  WERT MERKEN
4400  :     :    BR(ZU) = 0            :   REM  ZUG ZURUECK
4410  :     NEXT
4420  :     BW(N) = W                  :   REM  WERT MERKEN
4430  :     FOR  I = 1  TO  9
4440  :    :   BR(I) = ST(0,I)       :   REM  STELLUNG REKON.
4450  :     NEXT
4460  :     F = 6 - F                  :   REM  FARBWECHSEL
4470  NEXT
4495  :
4496  :
4497  :
4498  :
4499  REM
4500  REM  * MINIMAX *
4501  REM
4502  :
4510 BZ = ZG(1)                  :   REM  BESTEN ZUG INIT.
4520 W = 0                       :   REM  WERT INITIALISIEREN
4530  FOR  N = 1  TO  ZZ
4540  IF  BW(N) < W  THEN  4570  :   REM  NEUER BESTER WERT?
4550 W = BW(N)                   :   REM  BESTEN WERT MERKEN
4560 BZ = ZG(N)                  :   REM  BESTEN ZUG MERKEN
4570  NEXT
4580 BR(BZ) = F                  :   REM  ZUG AUSFUEHREN
4590  RETURN
4995  :
4996  :
4997  :
4998  :
4999  REM
6000  REM  *** BRETTAUSGABE ***
6001  REM
6002  :
6010  CLS    :   REM  PRINT CHR$(147);
```

```
6020  PRINT "  123"
6030  PRINT "A ";BR$(BR(1));BR$(BR(2));BR$(BR(3));" A"
6040  PRINT "B ";BR$(BR(4));BR$(BR(5));BR$(BR(6));" B"
6050  PRINT "C ";BR$(BR(7));BR$(BR(8));BR$(BR(9));" C"
6060  PRINT
6070  PRINT
6080  RETURN
6995  :
6996  :
6997  :
6998  :
6999  REM
7000  REM  *** BEWERTUNG ***
7001  REM
7002  :
7095  :
7096  :
7097  :
7098  :
7099  REM
7100  REM  * WERTE INITIALISIEREN *
7101  REM
7102  :
7110 A = 0  :  B = 0
7120 GW = 0
7195  :
7196  :
7197  :
7198  :
7199  REM
7200  REM  * DREIER-WERTE INITIALISIEREN *
7201  REM
7202  :
7210  FOR  I = 1  TO  8
7220  : SI(I) = 0
7230  NEXT
7295  :
7296  :
7297  :
7298  :
```

```
7299  REM
7300  REM  * DREIER-WERTE BERECHNEN *
7301  REM
7302  :
7310 SI(1) = BR(1) + BR(2) + BR(3)
7320 SI(2) = BR(4) + BR(5) + BR(6)
7330 SI(3) = BR(7) + BR(8) + BR(9)
7340 SI(4) = BR(1) + BR(4) + BR(7)
7350 SI(5) = BR(2) + BR(5) + BR(8)
7360 SI(6) = BR(3) + BR(6) + BR(9)
7370 SI(7) = BR(1) + BR(5) + BR(9)
7380 SI(8) = BR(3) + BR(5) + BR(7)
7395  :
7396  :
7397  :
7398  :
7399  REM
7400  REM  * DREIER-WERTE ADDIEREN *
7401  REM
7402  :
7410  FOR  I = 1  TO  8
7420  :  IF  SI(I) = 0  THEN  A = A + 1  :  B = B + 1
7430  :  IF  SI(I) = 1  THEN  A = A + 2
7440  :  IF  SI(I) = 2  THEN  A = A + 4
7450  :  IF  SI(I) = 3  THEN  A = A + 50
7460  :  IF  SI(I) = 5  THEN  B = B + 2
7470  :  IF  SI(I) = 10  THEN  B = B + 4
7480  :  IF  SI(I) = 15  THEN  B = B + 50
7490  NEXT
7495  :
7496  :
7497  :
7498  :
7499  REM
7500  REM  * FELDWERTE ADDIEREN *
7501  REM
7502  :
7510  FOR  I = 1  TO  9
7520  :  IF  BR(I) = 1  THEN  A = A + FW(I)
7530  :  IF  BR(I) = 5  THEN  B = B + FW(I)
```

```
7540  NEXT
7595  :
7596  :
7597  :
7598  :
7599  REM
7600  REM  * WERT ERRECHNEN *
7601  REM
7602  :
7610 GW = A - B
7620  RETURN
7995  :
7996  :
7997  :
7998  :
7999  REM
8000  REM  *** ZUGGENERATOR ***
8001  REM
8002  :
8095  :
8096  :
8097  :
8098  :
8099  REM
8100  REM  * ZUGLISTE INITIALISIEREN *
8101  REM
8102  :
8110  FOR  I = 1  TO  9
8120  : ZL(I) = 0
8130  NEXT
8140 ZZ = 0
8196  :
8197  :
8198  :
8199  REM
8200  REM  * ZUGLISTE ERSTELLEN *
8201  REM
8202  :
8210  FOR  I = 1  TO  9
8220  :  IF  BR(I) < > 0  THEN  8250  :   REM  BESETZT?
```

```
8230  : ZZ = ZZ + 1                    :   REM  INDEX HOEHER
8240  : ZL(ZZ) = I                     :   REM  ZUG EINTRAGEN
8250  NEXT
8260  RETURN
```

3. Schachprogrammierung

In diesem Abschnitt beschäftigen wir uns vor allem mit den Prinzipien der Schachprogrammierung. Wir betrachten die grundlegenden Methoden.

Auch hier hat die Technik des Programmierens eine große Bedeutung. Bevor wir uns der Schachprogrammierung selber widmen, beschäftigen wir uns mit einigen wichtigen Programmiermethoden, wie z.B. modularer Programmierung und Rekursion.

Schachprogramme müssen effizient geschrieben sein, damit sie schnell sind und viele Varianten untersuchen können. Dieses Thema zieht sich wie ein roter Faden durch dieses Kapitel, ob wir nun den Zuggenerator, die Hilfsroutinen oder die Bewertungsfunktion betrachten.

Den eigentlichen Kern diese Abschnitts bilden die Beschreibung der Suche und der Bewertungsfunktion. Diese beiden Moduln enthalten die Intelligenz des Programms - die Bewertungsfunktion das Wissen um das Schachspiel und die Suche das Wissen um die kombinatorische Bedeutung der Varianten.

3.1 Programmieren - Kunst und Technik

Schachprogramme sind natürlich recht umfangreich. Dies stellt erhebliche Anforderungen an den Programmierer. Wenn er völlig unsystematisch an die Aufgabe herangeht, wird er sich unweigerlich verheddern. Beschäftigen wir uns also mit dem Thema "Programmentwicklung".

3.1.1 Programmentwicklung

Bei größeren Aufgaben kann man die Programmentwicklung in Phasen aufteilen. Da ist zunächst die *Problemanalyse.* Man betrachtet die Aufgabenstellung, versucht sie in Teilaufgaben zu

zerlegen und macht sich grundsätzliche Gedanken zur Lösungsmethode.

In der nächsten Phase fallen die ersten Entscheidungen. In Abhängigkeit von gewissen Rahmenbedingungen wird die Programmiersprache gewählt. Die Grobstruktur des Programms wird festgelegt. Dies betrifft sowohl *Datenstrukturen* (Variablen) als auch die *Moduln* (Zeilenbereiche, die Teilaufgaben lösen) und deren Abhängigkeiten. So können z.B. bestimmte Moduln erst dann arbeiten, wenn andere die notwendigen Daten berechnet haben.

Nun folgt die *Implementierung*, das eigentliche Erstellen des Programms. (Dabei sollte man vor dem Eintippen die Programmzeilen auf Papier bringen. Viele Dinge lassen sich dort besser überblicken und leichter ändern.) Bei großen Programmen muß man sich auch über die Reihenfolge, in der man die Moduln schreibt, Gedanken machen. Jedes neue Modul muß sich testen lassen. Daher müssen zunächst die Basismoduln geschrieben werden. Nach und nach können dann "höhere" Moduln, die die bereits fertigen benutzen, hinzugefügt werden.

Diese Aufteilung ist sicherlich etwas willkürlich. Die drei Phasen gehen ineinander über. Man sollte sich aber nie in speziellen Teilproblemen verlieren, ohne das Ganze zu sehen.

3.1.2 Programmorganisation

Bei der Programmentwicklung müssen wir immer auf die Lesbarkeit, die Verständlichkeit des Programms achten. So sollten Variablennamen etwas über ihre Bedeutung verraten. In BASIC, wo nur zwei Buchstaben signifikant sind, ist das schwierig. Aber wir können dafür sorgen, daß bestimmte Abkürzungen immer die gleiche Bedeutung haben.

Auch der Programmtext muß logisch aufgebaut sein. Dies nennt man *modulares Programmieren*. Wir müssen Teilaufgaben so programmieren, daß sie in zusammenhängenden Zeilenbereichen gelöst werden. Wichtig sind dabei *Routinen* - Zeilenbereiche, die

durch ein GOSUB am Anfang angesprungen und am Ende des Bereichs durch ein RETURN verlassen werden. Ein Modul besteht dann aus mehreren aufeinanderfolgenden Routinen.

Der Vorteil besteht darin, daß wir bei Änderungen, Verbesserungen, meistens nur ein Modul oder auch nur eine Routine bearbeiten müssen. Dies sind relativ kleine Programmteile, die wir überschauen können. Wenn wir die Variablen logisch anordnen, so sind bei Änderungen kaum Seiteneffekte zu befürchten, es werden also andere Programmteile nicht fehlerhaft.

Dies alles ist relativ schwierig, denn die eigentliche Forderung lautet: vor dem Programmieren die Aufgabe völlig durchschauen, die Lösung ganz logisch aufbauen. In der Praxis werden wir immer Dinge übersehen, und wenn wir Pech haben, stehen sie im Widerspruch zu unseren bisherigen Vorstellungen. Dann ist die Versuchung groß, Flickschusterei zu betreiben und die Logik außer acht zu lassen.

3.1.3 Der Stack

a) Rekursion

Eine häufige Programmstruktur, die nicht nur bei Spielprogrammen auftritt, ist die *Rekursion*. Höhere Programmiersprachen wie PASCAL, ADA, oder C haben diese im Gegensatz zu BASIC schon berücksichtigt. Rekursion kann aber auch hier durch einen *Stack* realisiert werden.

Was ist Rekursion? Beim Schachprogramm tritt sie an folgender Stelle auf: nehmen wir an, wir hätten eine Routine "Untersuche Stellung" geschrieben, die den Wert, die Güte einer Stellung bestimmt. (Routine: ein Zeilenbereich des Programms, der durch ein GOSUB angesprungen wird und mit einem RETURN verlassen wird. Mit einem GOTO darf weder in Routinen hinein- noch herausgesprungen werden.) Diese Routine untersucht zunächst die Stellungsmerkmale und bildet sich so ein Urteil. Nun kann es sein, daß dies nicht ausreicht und einige Züge gesondert analysiert werden müssen, um Klarheit zu

schaffen. Dann muss "Untersuche Stellung" diese Züge nacheinander "ausführen", also die daraus resultierenden Stellungen erzeugen, und diese dann untersuchen lassen. Dies sollte naheliegenderweise wieder durch die Routine "Untersuche Stellung" ausgeführt werden. Es müßte also ein erneutes GOSUB zum Anfang der Routine gemacht werden, bevor das erste durch ein Return abgeschlossen ist. Dies nennt man einen *rekursiven Aufruf*, alles zusammen *Rekursion.*

Bei der Rekursion taucht aber ein Problem auf: beim zweiten Aufruf werden die gleichen Variablen wie beim ersten benutzt, es werden ja dieselben Programmzeilen ausgeführt. Dann rechnet die zuerst aufgerufene Routine nach Abschluß des zweiten Aufrufs (durch RETURN) mit anderen Variablenwerten als vor dem rekursiven Aufruf weiter. Dies darf nicht passieren. Hieraus folgt: alle Werte, die durch den rekursiven Aufruf verändert werden können und die hinterher noch benötigt werden, müssen vor dem GOSUB gerettet (weggespeichert) werden und nach dem entsprechenden RETURN wieder erzeugt werden. Variablen, die so behandelt werden müssen, nennt man *rekursive Variablen.*

Beim Schachspiel ist die Stellung ein Beispiel für eine rekursive Variable. Überlegt das Programm an einem Zug, so muß es die resultierende Stellung intern aufbauen. Ist es damit fertig, muß aber wieder die ursprüngliche Stellung vorliegen.

Hinzu kommt, daß Rekursion tief geschachtelt sein kann. Die durch den zweiten Aufruf (das zweite GOSUB) rekursiv aufgerufene Routine kann ja erneut sich selbst aufrufen. Dann müssen die rekursiven Variablen zweimal (an verschiedene Stellen!!) weggespeichert werden, damit die Werte nicht verlorengehen. Die Datenstruktur (Variablenmenge), die dies leistet, nennt man *Stack.*

Die Anzahl der rekursiven GOSUBs, die noch nicht durch ein RETURN abgeschlossen wurden, nennt man *Tiefe* der Rekursion. Eine wichtige Sache muß noch erwähnt werden. Die Rekursion muß *terminieren.* Es dürfen nicht laufend rekursive

GOSUB's aufeinanderfolgen. Dann würde das Programm nie enden. Wir benötigen also ein *Abbruchsbedingung*.

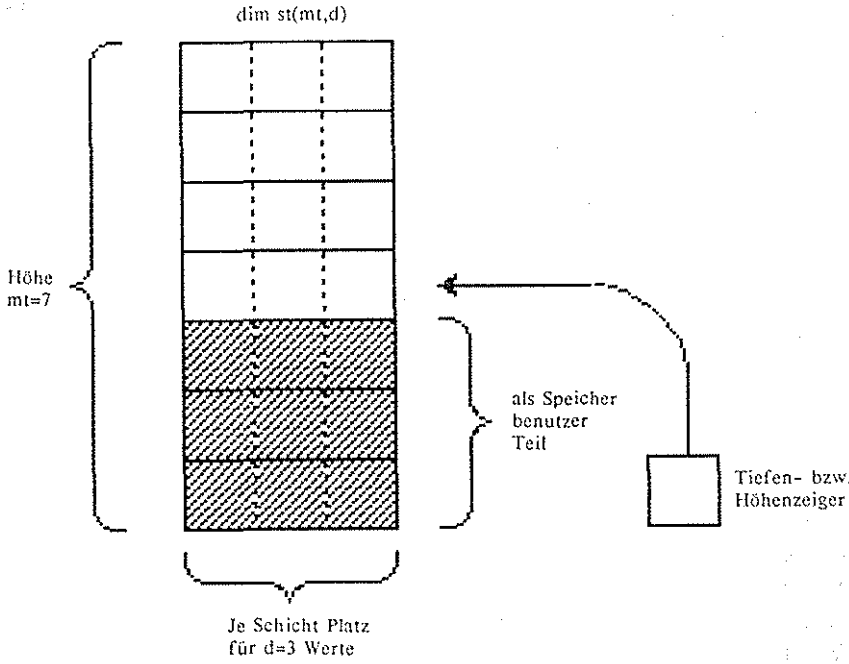
Bemerkung: Rekursion kann nicht nur auftreten, wenn eine Routine sich selbst aufruft, sondern auch, wenn zwei Routinen zusammenwirken. Ruft eine Routine A eine Routine B auf, und diese dann wieder Routine A, so entsteht auf indirektem Weg ebenfalls Rekursion - wieder ist Routine A doppelt aktiviert.
Dies führt zu dem sinnvollen Begriff *Aktivierung*: Den Aufruf einer Routine nennt man auch Aktivierung. Genauer: den speziellen Aufruf mit bestimmten Variablenwerten und bestimmtem Stackzustand nennt man Aktivierung.

b) Der Stack

Wie bereits gesagt, soll der Stack die Werte der rekursiven Variablen vor einem rekursiven Aufruf (GOSUB) speichern und anschließend wieder bereitstellen. Dies muß bei jeder zulässigen Tiefe der Rekursion möglich sein.

Logisch gesehen brauchen wir eine Art "Stapel". Vor dem GOSUB legen wir die rekursiven Werte auf den Stapel; anschließend, nach dem RETURN, tragen wir sie wieder in die Variablen ein und entfernen sie vom Stapel.

Zur Realisierung des Stacks muß man in BASIC Arrays benutzen. Der erste Index des Arrays gibt die Tiefe t an, der möglicherweise vorhandene zweite stellt für jeden rekursiven Wert einen Speicherplatz bereit. Z.B.: dim st(mt,d). Mt ist dabei die maximal zulässige Rekursions-Tiefe (die Obergrenze für t), d die Zahl der Werte.
Auf den Stapel übertragen heißt das:
Die maximale Stapelhöhe ist mt, jede Stapelschicht kann d Werte aufnehmen.



Zusätzlich benötigen wir eine ganzzahlige Variable, hier t. Diese gibt die aktuelle Rekursions-Tiefe an. Mit dem Stack führen wir nur zwei Operationen durch: entweder wir packen Informationen darauf, oder wir entfernen sie. Ist die aktuelle Tiefe t, so werden beim draufpacken die Stackelmente st(t,1), st(t,2) bis st(t,d) gesetzt und anschließend t um eins erhöht. Beim Entfernen läuft es genau umgekehrt: erst wird die Tiefe t erniedrigt, dann werden die oben gennannten Elemente benutzt und gelöscht.

Arbeiten (d.h. verändern) dürfen wir nur mit den "obersten" Elementen das Stacks, also mit st(t,1) bis st(t,d).

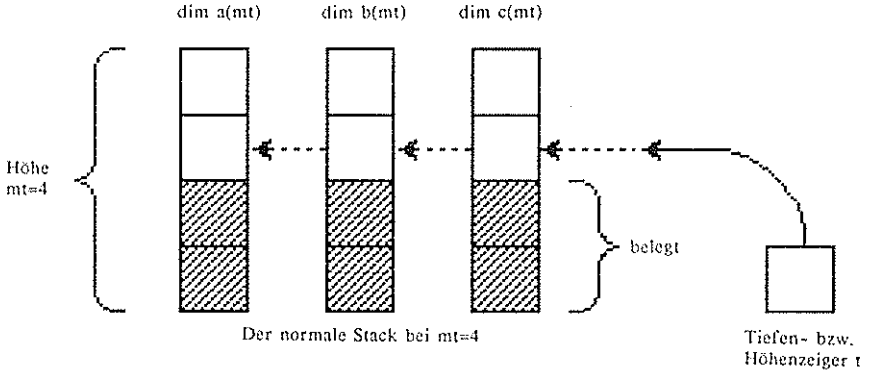
Auf den Stapel übertragen ist t die aktuelle Höhe.

Es sollen drei Ideen für die Benutzung des Stacks diskutiert werden.

b1) Der normale Stack

Diese Form des Stacks wird durch höhere Programmiersprachen unterstützt. Nehmen wir an, wir haben eine rekursive Prozedur mit den drei rekursiven Variablen a, b und c. Dann wird zusätzlich die Variable t eingeführt, die die Rekursionstiefe angibt. Bei jedem rekursiven Aufruf (GOSUB) wird t um eins erhöht, nach dessen Abschluß (RETURN) um eins erniedrigt. Drei Arrays werden eingeführt: dim a(mt), b(mt), c(mt). Bei jedem Vorkommen im Programmtext werden die Variablen a, b und c durch a(t), b(t) und c(t) ersetzt. Folglich werden von verschieden tiefen Aktivierungen der rekursiven Prozedur verschiedene Speicherplätze benutzt.

Der Stack wird also mit drei Arrays und einem gemeinsamen Tiefenzeiger t realisiert.



Der normale Stack bei mt=4

Nachteil dieser Lösung: viel Schreibarbeit für die Klammern, viel Rechenzeit für das laufende Indizieren. Ferner müssen meistens die Variablen vor einem GOSUB initialisiert werden.

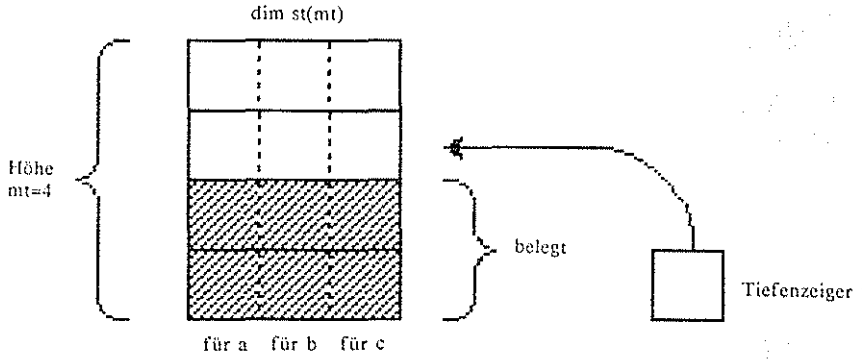
b2) Der Speicher-Stack

Nehmen wir wieder das Beispiel aus b1. Auch hier gebe t die Tiefe der Rekursion an. Diesmal wird nur ein Array eingeführt: dim st(mt,3). Vor jedem rekursiven Aufruf (GOSUB) werden a,b und c weggespeichert:

st(t,1)=a:st(t,2)=b:st(t,3)=c ,

und nach dessen Abarbeitung auf die alten Werte gesetzt:

a=st(t,1):b=st(t,2):c=st(t,3) .



Diese Art des Stacks wird in unserem Programm verwendet. Nachteil dieser Lösung ist, daß die Werte zweimal umgespeichert werden müssen.

b3) Der inkrementelle Speicher-Stack

Eigentlich handelt es sich hier um den gleichen Stack wie bei Teil b2. Sind aber nicht nur einfache Variablen sondern auch Arrays rekursiv , ist es unnötig, das ganze Array abzuspeichern, wenn nur ein Teil durch die Rekursion verändert wird. Ein Beispiel ist das Brett beim Schachspiel. Durch einen Zug werden

meist nur zwei Felder verändert, warum also alle speichern? Das würde Rechenzeit und Speicherplatz kosten. Wenn nur die Veränderungen protokolliert werden, nennt man das Vorgehen "inkrementell".

Gegenüber dem normalen Stack gewinnen wir noch einen großen Vorteil. Dort müssten wir beim rekursiven Aufruf das Brett für die nächste Tiefenstufe initialisieren, hier müssen nur die tatsächlich betroffenen Felder behandelt werden. Damit sparen wir Rechenzeit, und der Nachteil des Speicher-Stacks wird zum Vorteil. Wir müssen nur wenige Werte zweimal umspeichern; die meisten müssen wir nicht einmal initialisieren.

Der einzige neue Nachteil ist das etwas aufwendige Programmieren.

c) Ein Beispiel

Folgendes Problem sei gegeben:

> Vier Punkte, p1, p2, p3 und p4, seien zum Teil mit Einbahnstraßen verbunden. Die Frage ist: gibt es einen Weg von p1 nach p4?
> Das Array dim w%(4,4) gibt die Verbindungen an. Existiert ein Weg von pi nach pj (i=1, 2, 3, 4; j=1, 2, 3, 4), so hat w%(i,j) den Wert -1, sonst den Wert 0.

Wir wollen folgende (nicht sonderlich geschickte, aber rekursive) Lösungsidee verfolgen:

> Ein Weg vom Standort s nach p4 existiert genau dann, wenn entweder
>
> - s schon p4 ist (Abbruchsbedingung)
> - oder von s aus ein Punkt pj erreichbar ist, von dem aus p4 erreichbar ist.

Also gehen wir so vor:

> Wir prüfen, ob wir schon am Ziel sind. Wenn nicht, untersuchen wir nacheinander für alle erreichbaren Punkte, ob über sie p4 erreichbar ist.

In BASIC können wir folgende Routine schreiben:

```
100 REM S SEI STANDORT. IST P4 VON "S" AUS ERREICHBAR?
110 REM WENN JA, WIRD "JA%" AUF -1, SONST AUF 0 GESETZT.
120 JA%=0
130 IF S=4 THEN JA%=-1: GOTO300: REM SCHON AM ZIEL?
140 FOR I=1 TO 4: REM FUER ALLE ERREICHBAREN PUNKTE
150 IF NOT W%(S,I) THEN GOTO 260
200 S=I: REM REKURSIVEN AUFRUF VORBEREITEN, INITIALISIEREN
210 GOSUB 100: REM REKURSIVER AUFRUF
250 IF JA% THEN GOTO 300: REM GEFUNDEN
260 NEXT I
300 RETURN
```

Wir haben leider Fehler gemacht: die Rekursion nicht beachtet. Einmal terminiert die Routine nicht. Wenn die Einbahnstaßen einen Kreis bilden, sucht die Routine auch im Kreis. Es wird laufend GOSUB 100 gemacht, zum RETURN kommen wir aber nicht.

Dies kann einfach gelöst werden. Wenn ein Weg existiert, so ist seine Länge maximal 3; höchstens 4 Punkte werden berührt. Wir führen die Variable t ein, die die Rekursionstiefe protokolliert und ggf. die Suche beendet:

```
100 REM S SEI STANDORT. IST P4 VON "S" AUS ERREICHBAR?
110 REM WENN JA, WIRD "JA%" AUF -1, SONST AUF 0 GESETZT.
120 JA%=0
130 IF S=4 THEN JA%=-1: GOTO 300: REM SCHON AM ZIEL?
135 IF T=4 THEN GOTO 300: REM ZU TIEF?
140 FOR I=1 TO 4: REM FUER ALLE ERREICHBAREN PUNKTE
150 IF NOT W%(S,I) THEN GOTO 260
190 T=T+1: REM DIE REKURSIONS-TIEFE
200 S=I: REM REKURSIVEN AUFRUF VORBEREITEN, INITIALISIEREN
```

```
210 GOSUB 100: REM REKURSIVER AUFRUF
220 T=T-1: REM DIE REKURSIONS-TIEFE
250 IF JA% THEN GOTO 300: REM GEFUNDEN
260 NEXT I
300 RETURN
```

Der zweite Fehler ist, daß wir rekursive Variablen nicht zurücksetzen. Hierzu führen wir den Stack ein. Welche Variablen sind rekursiv?

w%: Nein, wird nicht verändert
t: Nein, dient der Verwaltung des Stacks
s: Ja, muß nach rekursivem Aufruf wieder alten Wert haben
i: Ja, wie eben
ja%: Nein, wird nur einmal verändert, dann sind wir fertig

c1) Lösung mit normalem Stack

```
10 DIM S(4),I(4): REM ARRAYS FUER DIE REKURSIVEN VARIABLEN.
100 REM S(T) SEI STANDORT. IST P4 VON "S(T)" AUS ERREICHBAR?
110 REM WENN JA, WIRD "JA%" AUF -1, SONST AUF 0 GESETZT.
120 JA%=0
130 IF S(T)=4 THEN JA%=-1: GOTO 300: REM SCHON AM ZIEL?
135 IF T=4 THEN GOTO 300: REM ZU TIEF?
140 FOR I(T)=1 TO 4: REM FUER ALLE ERREICHBAREN PUNKTE
150 IF NOT W%(S(T),I(T)) THEN GOTO 260
190 T=T+1: REM DIE REKURSIONS-TIEFE
200 S(T)=I(T-1): REM REKURSIVEN AUFRUF VORBEREITEN, INITIALISIEREN
210 GOSUB 100: REM REKURSIVER AUFRUF
220 T=T-1: REM DIE REKURSIONS-TIEFE
250 IF JA% THEN GOTO 300: REM GEFUNDEN
260 NEXT I(T)
300 RETURN
```

Die Arrays s und i bilden zusammen mit t den Stack. Leider geht diese Lösung in BASIC nicht, da in FOR-Schleifen nur einfache Variablen vorkommen dürfen.

c2) Lösung mit Speicher-Stack

```
10 DIM ST(4,2): REM ARRAY FUER DEN STACK
100 REM S SEI STANDORT. IST P4 VON "S" AUS ERREICHBAR?
110 REM WENN JA, WIRD "JA%" AUF -1, SONST AUF 0 GESETZT.
120 JA%=0
130 IF S=4 THEN JA%=-1: GOTO 300: REM SCHON AM ZIEL?
135 IF T=4 THEN GOTO 300: REM ZU TIEF?
140 FOR I=1 TO 4: REM FUER ALLE ERREICHBAREN PUNKTE
150 IF NOT W%(S,I) THEN GOTO 260
160 REM REKURSIVE VARIABLEN RETTEN
170 ST(T,1)=S
180 ST(T,2)=I
190 T=T+1: REM DIE REKURSIONS-TIEFE
200 S=I: REM REKURSIVEN AUFRUF VORBEREITEN, INITIALISIEREN
210 GOSUB 100: REM REKURSIVER AUFRUF
220 T=T-1: REM DIE REKURSIONS-TIEFE
230 REM ALTE WERTE EINSETZEN
240 S=ST(T,1)
245 I=ST(T,2)
250 IF JA% THEN GOTO 300: REM GEFUNDEN
260 NEXT I
300 RETURN
```

Damit haben wir eine Routine, die die Aufgabe löst. Ergänzen wir folgendes Rahmenprogramm, so sollte das Problem mit Rekursion gelöst sein.

```
20 S=1: T=1: REM RAHMENPROGRAMM
30 DIM W%(4,4): "W% EINLESEN"
40 GOSUB 100
50 PRINT "EIN WEG EXISTIERT";
60 IF NOT JA% THEN PRINT " NICHT";
70 PRINT "."
80 END
```

3.1.4 Besonderheiten von BASIC

BASIC hat wie jede Programmiersprache bestimmte Eigenheiten, und die betreffen auch das Schachprogramm. Darauf gehen wir hier ein. Dies ist aber keine Einführung in BASIC!

a) Die Variablen-Typen

BASIC kennt drei Arten einfacher Variablen:

Gleitpunktzahlen,	Bsp.: a
Ganze Zahlen,	Bsp.: a%
Strings,	Bsp.: a$

Die ersten beiden Typen interessieren uns besonders. Ganze Zahlen haben gegenüber Gleitpunktzahlen den Vorteil, daß sie zur Darstellung des Wertes nur zwei Byte belegen, und den Nachteil, daß sie etwas langsamer interpretiert werden. Beim Schachprogramm ist aber Speicherplatz ein entscheidender Punkt, und so benutzen wir, wenn immer möglich, ganze Zahlen. (Und beim Schachprogramm ist das fast immer möglich!) Bei compilierten Versionen können diese auch schneller bearbeitet werden.

In vielen Fällen benötigen wir nur ein Byte (8 Bit), um einen Wertebereich darzustellen. In BASIC können wir das nicht ausnutzen, wenn man das Programm aber auf Assembler übertragen will, sollte man darauf achten.

b) Arrays

"dim br%(144)" ist die Definition eines Feldes, dessen einzelne Felder br%(1) bis br%(144) ganze Zahlen darstellen. Wir benutzen nie den Index 0 (Bsp.: br%(0)), da in einigen BASIC-Versionen dies nicht erlaubt ist (im Gegensatz zu Commodore-BASIC). Dies kostet bei mehrdimensionalen Arrays (Bsp.: dim st%(8,44)) allerdings einigen Speicherplatz.

c) Logische Werte

In einer BASIC-Anweisung wie "IF A=1 THEN ..." tritt der logische Wert "A=1" auf. Je nachdem, ob er "wahr" oder "falsch" ergibt, wird der "THEN"-Teil ausgeführt oder nicht.
Logische Werte werden intern durch Zahlen dargestellt: der -1 entspricht "wahr" und der 0 entspricht "falsch". Hin und wieder nutzen wir dies aus, indem wir ganzen Zahlen logische Werte zuweisen, z.B.: B%=(A=1) oder kürzer B%=A=1. B% erhält dann den Wert -1, wenn "A=1" erfüllt ist, sonst den Wert 0.

Gleichwertig zur obigen IF-Anweisung wäre also

B%=A=1: IF B% THEN ...",

sofern b% eine sonst nicht benutzte Variable ist.

Gelegentlich rechnen wir auch mit logischen Werten, und dann kann auch die Anweisung "B%=-1" auftauchen. B% wird hier der Wert "wahr" zugewiesen.

d) Mischung mit Umgangssprache

Die Listings in diesem Kapitel sollen verschiedene Methoden demonstrieren und nicht, wie man BASIC programmiert. Um das Wesentliche herauszustellen, müssen wir technische Details außen vor lassen. Daher benutzen wir für einfache, aber unwesentliche Teilaufgaben Umgangssprache:

Bsp.: IF "KEIN ZUG MEHR DA" THEN GOTO 200"

Selbstverständlich ist das in Kapitel 4 abgelistete Programm frei von solchen Vereinfachungen, es ist also lauffähig, nur in diesem Kapitel wollen wir die Methoden herausstellen.
Die meisten umgangssprachlich beschriebenen Teilaufgaben werden im Programm durch GOSUB-Aufrufe realisiert.

3.2 Aufbau eines Schachprogramms - wie aus dem Baukasten

3.2.1 Die Bestandteile

Aus welchen Teilen müßte nun ein Schachprogramm bestehen? Dies läßt sich leicht beantworten, wenn wir uns Überlegen, wie ein Mensch Schachspielen lernt.

Zunächst benötigen wir natürlich ein Schachbrett mit den zugehörigen Figuren, damit wir die Stellung auch im Blick haben. Dem entspricht bei einem Schachprogramm die *interne Stellungsdarstellung*, also eine Menge von Datenstrukturen, die eine Stellung eindeutig darstellen.

Weiterhin müssen wir natürlich die Regeln kennenleren. Da gibt es einmal die Zugregeln, die bestimmen, in welcher Weise wir die Steine setzen dürfen. Dem entspricht im Schachprogramm der *Zuggenerator*. Dieser berechnet aufgrund der intern dargestellten Stellung die möglichen Züge.

Weitere Spielregeln geben Auskunft über den Wert einer Stellung, also darüber, ob und wer gewinnt. Diese Regeln sind meistens recht kompliziert. So ist ein Spieler dann Matt, wenn sein König im Schach steht und dies durch keinen Zug geändert werden kann. Dies wird zusammen mit dem Suchalgorithmus gelöst.

Doch dieses Wissen genügt nicht. In vielen Stellungen ist es sehr schwer, zu erkennen, wer besser steht. Auch in solchen Situationen muß ein Schachspieler sich ein Urteil bilden. Dies übernimmt die *Bewertungsfunktion*.

Schließlich muß ein Schachspieler auch Varianten durchrechnen, überlegen, was er und sein Gegner nach konkreten Zügen machen können. Dies übernimmt der *Suchalgorithmus*. Dieser kann auch feststellen, ob ein Spieler Schachmatt oder Patt ist.

Soweit zum inhaltlichen Aufbau. Ein Schachprogramm muß natürlich auch gewisse Servicefunktionen erfüllen, wie die Ein-

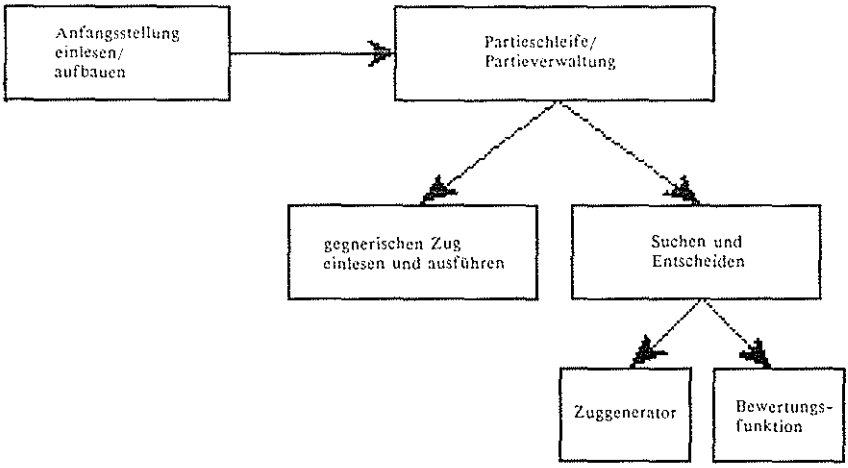
und Ausgabe von Stellungen und Zügen. Dies leistet das *Ein-Ausgabe-Modul.*

Schließlich benötigen wir noch einige *Hilfsroutinen.* Diese können beispielsweise feststellen, ob ein König im Schach steht, oder sie berechnen Zusatzinformationen über Züge.

Natürlich müssen wir auch eine ganze Partie spielen können, also abwechselnd mit dem Gegener ziehen, Zeitbegrenzungen einhalten und schließlich die Partie beenden. Dies erledigt im Programm die *Partieverwaltung.*

3.2.2 Die Abhängigkeiten

Wie wird nun der Programmablauf bei dieser Einteilung sein? Zunächst wird sicherlich die Anfangsstellung eingelesen werden. Anschließend muß die Partieschleife aktiviert werden. Diese muß abwechselnd einen Zug des Gegners - unter Benutzung des Ein-Ausgabe-Moduls - einlesen und die Suche des Programms aktivieren. Hierzu wird sie den Such- und Entscheidungsalgorithmus aufrufen. Dieser benutzt sicherlich Zuggenerator und Bewertungsfunktion. Dies wird auch durch die folgende Abbildung dargestellt.



3.2.3 Das Grundkonzept

Eine wichtige Frage müssen wir noch klären. Wie intelligent soll unser Programm werden? Wieviel Schachwissen soll es kennen? Wie genau soll es Stellungen beurteilen? Soll es für jeden Sonderfall spezielle Komponenten enthalten?

Wenn wir uns Schachliteratur besorgen, 1000-seitige Endspielbücher und dicke Mittelspielbücher betrachten, wissen wir, welche Aufgabe vor uns liegt. Alles werden wir nicht programmieren können. Wir werden sorgfältig abwägen müssen, welche Merkmale das Programm kennen muß und welche nicht. Zusätzliche Kenntnisse bedeuten nicht nur zusätzlichen Programmieraufwand, sondern auch, daß das Programm langsamer wird. Weniger Wissen kann mehr sein, wenn das Programm dann tiefer suchen kann.

Auch die Suche ist ein wichtiger Punkt. Natürlich muß das Programm auch Zugfolgen, Varianten berechnen. Doch welche Züge soll es untersuchen, welche nicht? Wenn ein Großmeister fünf oder sechs Züge vorausberechnet, kann er dies nur, weil er sich auf die sinnvollen Varianten beschränkt und die schlechten ignoriert.

Schon 1949 hat sich Claude Shannon mit diesem Thema beschäftigt. Er hielt zwei Programmarten für denkbar: das Typ-A-Programm würde alle Züge bis zu einer festen Tiefe untersuchen, das Typ-B-Programm dagegen die wichtigen von den unsinnigen Zügen trennen und sich auf die wesentlichen beschränken.

Leidvolle Erfahrungen der ersten Schachprogrammierer zeigten, daß diese Trennung, so schön sie auch wäre, nicht einwandfrei funktioniert. Immer wieder gab es Züge, die ignoriert wurden und unbedingt hätten untersucht werden müssen. Die erfolgreichen heutigen Programme sind daher im wesentlichen "Brute-Force"-Programme, wie man inzwischen Typ-A-Programme nennt.

Sie nutzen die wichtigste Eigenschaft eines Computers aus: seine Geschwindigkeit. Von Anfgang an wird das Programm so konzipiert, daß es möglichst schnell ist, möglichst viele Stellungen untersuchen kann. Die Bewertungsfunktion wird klein gehalten, damit sie nicht viel Zeit verbraucht. Es werden vor allem die allgemeinen, fast immer gÜltigen Stellungsmerkmale eingebaut. Dann kann das Programm viele Stellungen untersuchen, es kommt auch im "Brute-Force"-Stil tief genug. Und es kann nichts übersehen, es sei denn, es handelt sich um sehr tiefe Varianten.

Dies ist auch für uns attraktiv. Ein relativ kleines, schnelles und ganz passabel spielendes Programm sollte so zusammenkommen. Wir werden auch an diesem Programm viele Tricks sehen, die die Spielstärke vervielfachen, Tricks, die häufig mit nur 10 bis 20 BASIC-Zeilen eingebaut werden können.

Zwei Riesen-Schwächen werden uns aber auch begleiten. Die eine ist der Home-Computer, der mit großen Rechenanlagen natürlich nicht konkurrieren kann, jedenfalls nicht, wenn es um Geschwindigkeit geht. Und die andere ist: BASIC. BASIC benutzen wir, weil die Sprache allgemein bekannt ist. BASIC wird aber interpretiert, und das macht die Ausführung relativ langsam. Diesem Manko können wir mit einem BASIC-Compiler

beikommen - das Programm wird ca. vier Mal schneller. Wollten wir ein wirklich konkurrenzfähiges Programm schreiben, müßten wir Assembler benutzen. Das Programm wären dann noch 2 bis 10 Mal schneller, was einen entscheidenden Unterschied macht.

3.3 Der Zuggenerator - Eine zeitkritische Angelegenheit

Der Zuggenerator soll unser Einstiegspunkt zum Schachprogramm sein. Seine Arbeitsweise hängt natürlich eng mit der Stellungsdarstellung zusammen. Er muß besonders effizient programmiert werden, da er in jeder untersuchten Stellung aufgerufen wird und durchschnittlich 40 Züge berechnet. Das kostet Zeit.

Ziel soll es sein, eine Liste der legalen (nach den Spielregeln erlaubten) Züge zu erstellen. Die Züge sollen durch Standort und Zielfeld angegeben werden.

Wegen einer Sonderregel ist das Generieren der Züge eigentlich eine komplizierte Angelegenheit. Eine Figur darf nur dann ziehen, wenn der eigene König danach nicht im Schach steht. Dies kann nur umständlich (und zeitaufwendig) programiert werden. Doch diese Regel können wir einfach mißachten. Wenn tatsächlich ein Zug aufgrund dieser Regel unmöglich ist, so fällt dies spätestens dann auf, wenn wir ihn probeweise ausgeführt haben, und der Gegner dann den König schlagen kann.

Alle Probleme sind damit jedoch nicht gelöst - wir müssen uns um eine besonders effiziente, schnelle Routine bemühen. Betrachten wir hierzu die Damenzüge. Die Dame darf in jede Richtung gehen, bis sie auf den Brettrand oder eine andere Figur stößt. Wie lassen sich diese Züge bei verschiedenen Stellungsdarstellungen berechnen? Welche Methode ist die beste? Zur weiteren Vereinfachung wollen wir nur die Züge nach rechts oben berechnen.

3.3.1 Zweidimensionale Brettdarstellung

Hier wird das Brett durch ein zweidimensionales Array "dim br(8,8)" realisiert. Die einzelnen Felder von br geben an, welche Figur sich dort befindet. In unserem Programm werden die Figurenarten folgendermaßen codiert:

Figurenart	Wert
schwarzer König	1
schwarze Dame	2
schwarzer Turm	3
schwarzer Läufer	4
schwarzer Springer	5
schwarzer Bauer	6
freies Feld	7
weißer Bauer	8
weißer Springer	9
weißer Läufer	10
weißer Turm	11
weiße Dame	12
weißer König	13

Enthält also ein Feld von br den Wert 11, so steht dort ein weißer Turm.

8	1,8	2,8	3,8	4,8	5,8	6,8	7,8	8,8
7	1,7	2,7	3,7	4,7	5,7	6,7	7,7	8,7
6	1,6	2,6	3,6	4,6	5,6	6,6	7,6	8,6
5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5
4	1,4	2,4	3,4	4,4	5,4	6,4	7,4	8,4
3	1,3	2,3	3,3	4,3	5,3	6,3	7,3	8,3
2	1,2	2,2	3,2	4,2	5,2	6,2	7,2	8,2
1	1,1	2,1	3,1	4,1	5,1	6,1	7,1	8,1
	A	B	C	D	E	F	G	H

Diagramm der zweidimensionalen Stellungsdarstellung

Die Zahlen geben die Indices der Felder an; z.B. hat das Feld C2 die Indices br(3,2).

Wie lassen sich nun die Damenzüge nach rechts oben erzeugen? Dies würde folgendes Programm leisten:

```
100 REM DAMENZUEGE NACH RECHTS OBEN
110 REM STANDORT SIND DIE KOORDINATEN S1 UND S2
120 Z1=S1:Z2=S2: REM KOORDINATEN DER ZIELFELDER
130 IF Z1=8 OR Z2=8 THEN GOTO 170: REM RANDFELD?
140 Z1=Z1+1:Z2=Z2+1: REM NAECHSTES FELD
149 REM FREIES FELD?
150 IF BR(Z1,Z2)=7 THEN "ZUG EINTRAGEN": GOTO 130
160 IF BR(Z1,Z2) "IST GEGENRISCHE FIGUR" THEN "SCHLAGZUG EINTRAGEN"
170 REM FERTIG
```

Nachteil dieser Darstellungsweise ist, daß wir immer zwei Koordinaten verändern müssen. Dies kostet Schreibaufwand (für uns) und Rechenzeit (für den Rechner).

3.3.2 Eindimensionale kompakte Brettdarstellung

Wir stellen das Brett durch ein eindimensionales Array "dim br(64)" dar. Dem Feld A1 entspricht der Index 1, B1 der Index 2, usw.

8	57	58	59	60	61	62	63	64
7	49	50	51	52	53	54	55	56
6	41	42	43	44	45	46	47	48
5	33	34	35	36	37	38	39	40
4	25	26	27	28	29	30	31	32
3	17	18	19	20	21	22	23	24
2	9	10	11	12	13	14	15	16
1	1	2	3	4	5	6	7	8
	A	B	C	D	E	F	G	H

Diagramm der eindimensionalen kompakten Brettdarstellung

Die Zahlen geben die Indices der Felder an; z.B. hat das Feld H1 den Index 8.

Wie sieht jetzt der Zuggenerator aus?

```
100 REM LAEUFERZÜGE NACH RECHTS OBEN
110 REM S IST STANDORT
120 Z=S: REM ZIELFELD
130 IF "Z IST RANDFELD" THEN GOTO 170
140 Z=Z+9: REM 9 IST DIE SCHRITTWEITE NACH RECHTS OBEN
149 REM FREIES FELD?
150 IF BR(Z)=7 THEN "ZUG EINTRAGEN": GOTO 130
160 IF BR(Z) "IST GEGNERISCHE FIGUR" THEN "SCHLAGZUG EINTRAGEN"
170 REM FERTIG
```

Offensichtlich haben wir die Nachteile der zweidimensionalen Brettdarstellung überwunden. Doch nun haben wir Schwierigkeiten mit dem Erkennen des Rands. Versuchen Sie einmal, die Abfrage "if z ist Randfeld" zu programmieren! Dies ist fast unmöglich, und außerdem für jede Richtung anders. Noch schwieriger ist diese Abfrage bei Springerzügen.

3.3.3 Eindimensionale eingebettete Brettdarstellung

Hier benutzen wir die Technik der Einbettung. Wir codieren nicht das Brett, sondern ein erweitertes, um zwei Reihen nach jeder Seite vergrößertes Brett. Dem Feld A1 entspricht der Index 27, H1 der Index 34 und H8 der Index 118.

	133	134	135	136	137	138	139	140	141	142	143	144
	121	122	123	124	125	126	127	128	129	130	131	132
8	109	110	111	112	113	114	115	116	117	118	119	120
7	97	98	99	100	101	102	103	104	105	106	107	108
6	85	86	87	88	89	90	91	92	93	94	95	96
5	73	74	75	76	77	78	79	80	81	82	83	84
4	61	62	63	64	65	66	67	68	69	70	71	72
3	49	50	51	52	53	54	55	56	57	58	59	60
2	37	38	39	40	41	42	43	44	45	46	47	48
1	25	26	27	28	29	30	31	32	33	34	35	36
	13	14	15	16	17	18	19	20	21	22	23	24
	1	2	3	4	5	6	7	8	9	10	11	12

a b c d e f g h

Diagramm der eindimensionalen eingebetteten Brettdarstellung

Die Zahlen geben die Indices der Felder an;
z.B. hat das Feld H8 den Index 118

Zu Programmbeginn wird den Feldern, die nicht zum Brett gehören, ein eindeutiger Wert (14) zugewiesen, der nie verändert wird und sie als unbegehbar ausweist. Damit ist das Problem der Randerkennung gelöst. Die Damenzüge nach rechts oben können nun so generiert werden:

```
100 REM DAMENZUEGE NACH RECHTS OBEN
110 REM STANDORT IST S
120 Z=S: REM ZIELFELD
130 Z=Z+13: REM 13 IST NUN DIE SCHRITTWEITE
140 IF BR(Z)=14 THEN GOTO 170: REM UEBER RAND GELAUFEN ?
149 REM FREIES FELD?
150 IF BR(Z)=7 THEN "ZUG EINTRAGEN": GOTO 130
160 IF BR(Z) "IST GEGNERISCHE FIGUR" THEN "SCHLAGZUG EINTRAGEN"
170 REM FERTIG
```

Dies ist offensichtlich die geschickteste Art, Züge zu generieren. Auch bei Springerzügen landen wir immer auf dem erweiterten Brett, da wir es in jeder Richtung um 2 Felder vergrößert haben.

3.3.4 Der Zuggenerator im Ganzen

Überlegen wir zunächst, wie man alle Damenzüge, also nicht nur die nach rechts oben, generiert. Sicherlich ist es unnötig, die obige Schleife 8 mal mit veränderten Schrittweiten aufzuschreiben. Dies kann mit einer äußeren Schleife gelöst werden, die von 1 bis 8 läuft (für die 8 Bewegungsrichtungen) und die Schrittweiten aus einem vorbesetzten Array (im Programm: dim dw%(8) für Damenweite) erhält. So könnte dw%(1) den Wert 11 für die Schrittweite nach links oben enthalten, dw%(2) den Wert 13 für die Richtung nach rechts oben, usw. Die Turm- bzw. Läuferzüge können mit der Routine ebenfalls berechnet werden, wenn man die Laufgrenzen der äußeren Schleife verändert.

Auf ähnliche Art können Springer- und Königszüge eingebaut werden.

Bleiben noch die Bauernzüge und die Sonderfälle. Weiße und schwarze Bauern ziehen in verschiedene Richtungen, und so ist es naheliegend, für die beiden Farben verschiedene Routinen zu schreiben.

Die Sonderfälle sind das En-Passant-Schlagen und die Rochaden. Ersteres sollte nicht gemeinsam mit den übrigen Bauernzügen berechnet werden, da dies viele unnötige Abfragen kosten würde, auch wenn zuvor kein gegnerischer Bauer zwei Felder vorgerückt ist.

Die Rochade ist zwar ein Königszug, ob sie jedoch möglich ist, hängt von vielen Dingen ab: der König darf noch nie gezogen sein, er darf nicht im Schach stehen, nicht über ein angegriffenes Feld hinwegrochieren, und auch der beteiligte Turm darf noch nicht gezogen sein. Dies hat uns veranlaßt, hierfür eine eigene Routine zu schreiben.

Die Details des Zuggenerators werden zusammen mit dem Programm beschrieben.

3.4 Suchen und Entscheiden

Num kommen wir zu einem der kritischsten Bereiche des Schachprogramms, dem *Such- und Entscheidungsalgorithmus*. Hier fällt die Entscheidung, wie und was das Programm spielt.

Natürlich kann der Suchalgorithmus nur mit den anderen Teilen des Programms zusammen funktionieren. Wir müssen also davon ausgehen, daß wir schon über Bewertungsfunktion, Zuggenerator usw. verfügen. Doch dies muß uns nicht weiter stören - es genügt, wenn wir wissen, was die anderen Programmteile leisten.

Der Kern dieses Kapitels ist die Frage, welche Züge und Varianten untersucht werden müssen und welche nicht. Wir beginnen mit einem einfachen Ansatz und klären die Grundlagen (Minimax-Suche). Diesen ersten Versuch verfeinern wir Schritt für Schritt. Zunächst lernen wir den Alpha-Beta-Algorithmus kennen - bei geschickter Anwendung eine Technik mit

entscheidender Wirkung. Doch auch dies ist noch nicht gut genug. Damit das Programm die Folgen von Schlagzügen erkennen kann, führen wir die Ruhesuche ein. Diese werden wir auch auf Schachgebote ausdehnen. Weitere Verbesserungen erzielen wir durch die variable Suchtiefe, den iterativen Suchansatz und die Fenster-Technik. Das sind die bei heutigen Programmen üblichen Methoden.

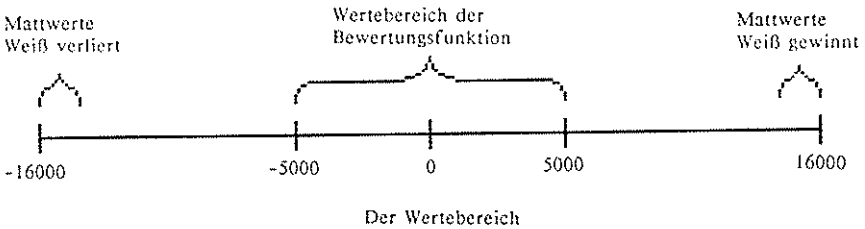
Bei allen Verfahren wird sich herausstellen, daß die Reihenfolge, in der die Züge untersucht werden, sehr wichtig ist. Daher widmen wir der Zugsortierung ein weiteres Kapitel.

Und noch eine Vorbemerkung: beim Schachspiel sind normalerweise ca. 40 Züge pro Stellung möglich. In den folgenden Betrachtungen werden wir aber so tun, als wären es nur zwei oder drei. So wird alles überschaubarer, und die wesentlichen Erkenntisse bleiben doch dieselben.

3.4.1 Die Bewertungsfunktion - eine Vorbetrachtung

Für die Suche ist die Bewertungsfunktion die wichtigste Verbindung zum Schachwissen. Wir kennen zwar deren Arbeitsweise noch nicht, wir müssen aber wissen, was sie leistet.

Stellungen werden vom Programm auf einer Zahlenskala bewertet, die von -16000 bis +16000 reicht. Hohe, positive Werte zeigen Vorteil für Weiß an, der Wert 0 eine ausgeglichene Stellung. Die Bewertungsfunktion liefert nur Werte von ca. -5000 bis +5000, extremere Zahlen zeigen ein Matt an (und das wird nicht von der Bewertung, sondern von der Suche festgestellt). So würde der Wert 15998 anzeigen, daß das Programm Weiß hat und jetzt mattsetzt - aber dazu später mehr.



Der Wertebereich

Normale Stellungen, in denen kein Matt in Sicht ist, werden dagegen von der Bewertungsfunktion beurteilt. Das Urteil setzt sich aus zwei Komponenten zusammen: aus der *Materialbalance* und der *positionellen Bewertung*.

Die Materialbalance gibt die Kräfte der Figuren beider Seiten an. So hat ein Bauer den Wert 100, Springer und Läufer haben den Wert 325, ein Turm zählt 500 Punkte und die Dame 900. Hat nun Schwarz einen Bauern mehr, so ergibt die Materialbalance insgesamt den Wert -100.

Die positionelle Bewertung ist komplizierter, aber auch bescheidener. Hier werden alle möglichen Stellungsmerkmale beurteilt: Doppelbauern, Freibauern, Figurenstandorte, Aktivität, Königssicherheit usw. Doch alle diese Faktoren zusammen ergeben (jedenfalls in 99% aller Fälle) in der Endsumme weniger als den Wert eines Bauern, also weniger als 100 Punkte. Warum und wieso - das wird später erläutert. Unter anderem soll so sichergestellt werden, daß das Programm nicht unsinnigerweise Figuren opfert.

Damit haben wir einen ersten Einblick in die Bewertungsfunktion. Im Zweifelsfall dominiert die Materialbalance gegenüber der positionellen Bewertung - ein Mehrbauer wiegt schwerer als alle strategischen Nachteile. Das Programm wird ziemlich habgierig spielen, was zu Gefahren führen kann. Dies muß dann die Suche herausfinden. Jedenfalls ist dafür gesorgt, daß das Programm auf seine Figuren aufpaßt.

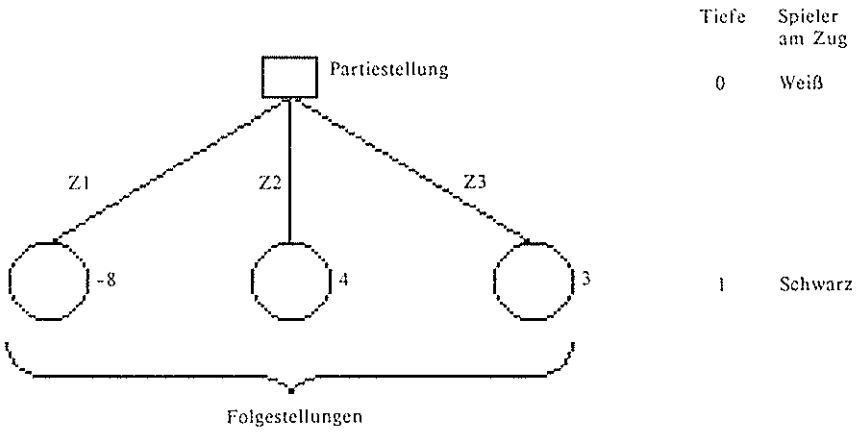
3.4.2 Die Minimax-Suche

a) Einfach, aber ein Anfang

Die einfachste Möglichkeit der Zugbestimmung ist folgende:

> Wir lassen das Programm in der Partiestellung nacheinander alle Züge ausprobieren - natürlich nur intern. Jedesmal wird die Bewertungsfunktion aufgerufen und am Ende wird der Zug gewählt, der den besten Wert erhielt. Ist also Weiß am Zug, so wird der Zug mit dem höchsten Wert, sonst der mit dem niedrigsten gemacht (vgl. vorigen Abschnitt).

Wir können das mit einer Zeichnung darstellen. In der Informatik nennt man solche Diagramme Graphen. Die Stellungen werden als *Knoten* (Punkte), die Züge als *Kanten* (Striche) gezeichnet. Enthält ein solcher Graph keine Kreise (bilden die Kanten keinen Kreis), so heißt er Baum. Dies ist hier immer der Fall.



Der obere Knoten stellt die Partiestellung dar. Wir zeichnen Knoten, in denen Weiß am Zug ist, eckig, die anderen rund. Von der Partiestellung, in der also Weiß am Zug ist, gehen 3 Züge aus, dargestellt durch die Kanten Z1 bis Z3. Hierauf folgen die entsprechenden Stellungen, in denen nun Schwarz am Zug ist. Folglich sind diese Knoten rund gezeichnet.

Den Knoten (Stellungen) wird eine *Tiefe* zugeordnet. Die Partiestellung hat die Tiefe 0, die Tiefe der anderen Stellungen entspricht der Zahl der Züge, über die sie erreicht wurden.

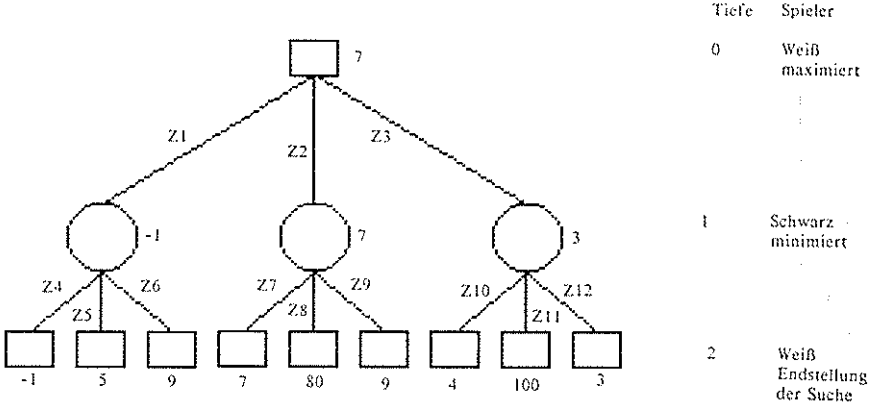
Das Programm hat alle Stellungen der Tiefe 1 erzeugt und bewertet. Die entsprechenden Werte schreiben wir zusätzlich an die Knoten.

Da Weiß am Zug ist, muß hier also der Zug Z2 gewählt werden.

b) Tiefere Suchen

Natürlich genügt dieses Verfahren nicht - so gut ist die Bewertungsfunktion auch wieder nicht. Häufig muß das Programm lange Zugfolgen untersuchen, um die richtige Entscheidung zu treffen. Es muß also tiefer analysieren.

Lassen wir daher das Programm nicht nur die Züge der Partiestellung ausprobieren, sondern auch die der Folgestellungen. Dies nennen wir eine *Suche der Tiefe 2*. Der *Suchbaum* (die Menge der untersuchten Stellungen) sieht dann so aus:



Wieder werden die Endstellungen von der Bewertungsfunktion beurteilt; die entsprechenden Werte sind unter die Knoten der Tiefe 2 geschrieben.

Das Programm untersucht die Züge in folgender Reihenfolge: Z1, Z4, Z5, Z6, Z2, Z7,Z8, Z9, Z3, Z10, Z11, Z12. (In der Informatik nennt man diese Reihenfolge eine "gerichtete Tiefensuche von links nach rechs"). Wenn es mit Z6 fertig ist, weiß es, daß über Z1 nur der Wert -1 erreichbar ist, denn Schwarz hat nach Z1 die Auswahl und wird den Zug wählen, der zur

Stellung mit dem niedrigsten Wert führt. Dann können wir der Stellung nach Z1 den Wert -1 zuordnen.

Generell gilt das *Minimax-Prinzip*:

> Der Wert von Stellungen, in denen Schwarz am Zug ist, ist das Minimum der Werte der Folgestellungen. Ist Weiß am Zug, so ist es das Maximum. Dies ist das *Minimax-Prinzip*. Es gilt natürlich nur für Stellungen innerhalb des Suchbaums, Endstellungen werden durch die Bewertungsfunktion beurteilt.

In unserem Beispiel sollte Weiß also den Zug Z2 wählen.

c) Implementierung in BASIC

Jetzt haben wir eine bestimmte Suchmethode definiert. Sie heißt *"Erschöpfende Minimax-Suche bis zu einer festen Tiefe"*. Das Prinzip ist:

> Alle Zugfolgen einer bestimmten Tiefe werden analysiert. Die Endstellungen werden durch die Bewertungsfunktion beurteilt. Stellungen innerhalb des Suchbaums erhalten Werte nach dem Minimax-Prinzip. Es wird der Zug gemacht, der zur Stellung mit dem besten Wert führt.

Wir werden jetzt eine entsprechende Routine in BASIC schreiben. Allerdings werden wir BASIC und Umgangssprache mischen, um das Wesentliche klarer herausstellen zu können. In deutsch geschriebene Anweisungen werden im richtigen Programm durch GOSUB-Aufrufe ersetzt.

Realisieren können wir die Suche durch folgende Routine:

```
100 REM STELLUNG UNTERSUCHEN, WEISS AM ZUG
110 REM STELLUNG IST INTERN DARGESTELLT
120 REM T% IST TIEFE DER AKTUELLEN STELLUNG
130 REM MT% IST DIE SUCHTIEFE
140 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
```

```
160 IF T%=MT% THEN GOSUB "BEWERTUNG": MW%=PW%: GOTO 400
170 MW%=-16000
180 GOSUB "ZUGGENERATOR"
190 IF "KEINE ZUEGE MEHR DA" THEN GOTO 310
200 "WAEHLE ZUG AUS"
210 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
220 "FUEHRE ZUG AUS": REM INTERNE STELLUNGSDARSTELLUNG AENDERN
230 GOSUB 500: REM "UNTERSUCHE STELLUNG FUER SCHWARZ"
240 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
250 "ERZEUGE MIT STACK ALTE STELLUNG"
260 IF AW% <= MW% THEN GOTO 300
270 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
300 GOTO 190
310 REM SUCHSCHLEIFE FERTIG
400 RETURN: REM ENDE DER STELLUNGSANALYSE FÜR WEISS
```

Kommentare:

Diese Routine wird aufgerufen, wenn eine Stellung, in der Weiß zieht, untersucht werden muß. Sie setzt voraus, daß diese Stellung in den dafür vorgesehenen Variablen intern dargestellt ist. t% muß die Tiefe der Stellung, mt% die (maximale) Suchtiefe sein. Die Routine untersucht dann die Stellung und am Ende beschreibt mw% den Minimax-Wert. Müssen noch Züge untersucht werden, ist also die Suchtiefe noch nicht erreicht, so beschreibt mw% zu jeder Zeit den Wert des bisher besten untersuchten Zuges.

Zunächst wird abgefragt, ob der Suchhorizont schon erreicht ist (Zeile 160). Falls ja, wird die Bewertungsfunktion aufgerufen. Diese setzt die Variable pw% auf den Stellungswert. Dieser wird zum Wert der Stellung (mw%=pw%) und die Routine ist fertig.

Ist die maximale Suchtiefe noch nicht erreicht, müssen alle Züge untersucht werden. Der Stellungswert wird der Wert des besten untersuchten Zuges sein. Daher wird mw% zunächst auf den kleinsten zulässigen Wert gesetzt. (Zeile 170)

Die Schleife, in der alle Züge untersucht werden, beginnt bei Zeile 190 und endet bei Zeile 300. Sind alle Züge bearbeitet, wird sie verlassen (190).

Nun wird ein Zug ausgewählt, Stellung und rekursive Variablen werden in den Stack gepackt, und dann wird der Zug probeweise ausgeführt. Die interne Stellungsdarstellung wird verändert. (Zeilen 200-220)

Nun muß die so erzeugte Stellung ebenfalls untersucht werden. Dies Übernimmt die Schwesterroutine, die Stellungen mit schwarzem Anzugsrecht bearbeitet (Zeile 230).

Der von dieser Routine ermittelte Wert ist nun der aktuelle Wert (aw%), der Zug wird zurückgenommen (Zeile 250) und falls es der bisher beste war, wird mw% erhöht (Zeilen 260 und 270).

In dieser Routine fehlt noch die Bestimmung des zu spielenden Zuges. Dies läßt sich aber leicht einbauen. Falls die Tiefe der analysierten Stellung 0 ist, falls also die Partiestellung untersucht wird, muß der Zug, der zum besten (hier: höchsten) Wert führt, gewählt werden. Hierauf werden wir später eingehen, da wir auch eine etwas andere Lösung benutzen können.

Bemerkung: Rekursion, Stack und rekursive Variablen wurden bereits in Kapitel 3.1.3 für sich behandelt. Bei unserem Schachprogramm tritt Rekursion bei der Untersuchung von Stellungen und Zügen auf. Rekursiv sind alle Variablen, die stellungsspezifische Informationen enthalten. Das sind einmal die für Brett und Figurenstandorte, dann jene, die die Züge der Stellung beschreiben, aber auch mw% (siehe oben), denn der bisherige Wert der Stellung muß ja auch erhalten bleiben.

Die rekursiven Variablen müssen vor der Untersuchung eines Zuges gerettet, also weggespeichert werden (im Stack), denn wenn der Zug analysiert wird, wird die Stellung verändert und damit auch die rekursiven Variablen. Deren Werte brauchen wir aber anschließend noch; nach der Untersuchung eines Zuges wollen wir ja in der alten Stellung weiterrechnen. Dann werden sie wieder aus dem Stack geholt.

Für Schwarz ist die Implementierung der Suche analog:

```
500 REM STELLUNG UNTERSUCHEN, SCHWARZ AM ZUG
510 REM STELLUNG IST INTERN DARGESTELLT
520 REM T% IST TIEFE DER AKTUELLEN STELLUNG
530 REM MT% IST DIE SUCHTIEFE
540 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
560 IF T%=MT% THEN GOSUB "BEWERTUNG": MW%=PW%: GOTO 800
570 MW%= 16000
580 GOSUB "ZUGGENERATOR"
590 IF "KEINE ZUEGE MEHR DA" THEN GOTO 710
600 "WAEHLE ZUG AUS"
610 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
620 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
630 GOSUB 100: REM "UNTERSUCHE STELLUNG FUER WEISS"
640 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
650 "ERZEUGE MIT STACK ALTE STELLUNG"
660 IF AW% GROESSER-GLEICH MW% THEN GOTO 700
670 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
700 GOTO 190
710 REM SUCHSCHLEIFE FERTIG
800 RETURN: REM ENDE DER STELLUNGANALYSE FUER SCHWARZ
```

Der einzige Unterschied zur vorigen Routine ist, daß hier minimiert wird. Folglich wird mw% mit 16000 , dem größten denkbaren Wert, initialisiert und später ggf. erniedrigt.

d) Suchtiefe und Aufwand

Betrachten wir nun den Aufwand einer solchen Suche. Nehmen wir an, in jeder Stellung sind genau 40 Züge möglich. (Diese Größenordnung ist beim Schach realistisch.) Wie oft müssen wir dann die Bewertungsfunktion aufrufen? Diese Zahl ist zugleich die Zahl der Endstellungen des Suchbaums.

Eine Suche der Tiefe 1 untersucht 40 Züge, wir haben also 40 Endstellungen. Bei Tiefe 2 müssen in jeder dieser Endstellungen wieder 40 Züge untersucht werden, das macht insgesamt 1600 - eine große Zahl. Allgemein ist die Zahl der Endstellungen 40^mt, wenn mt die Suchtiefe ist. Eine kleine Tabelle zeigt uns, daß wir so mit annehmbaren Rechenzeiten nicht sehr weit kommen:

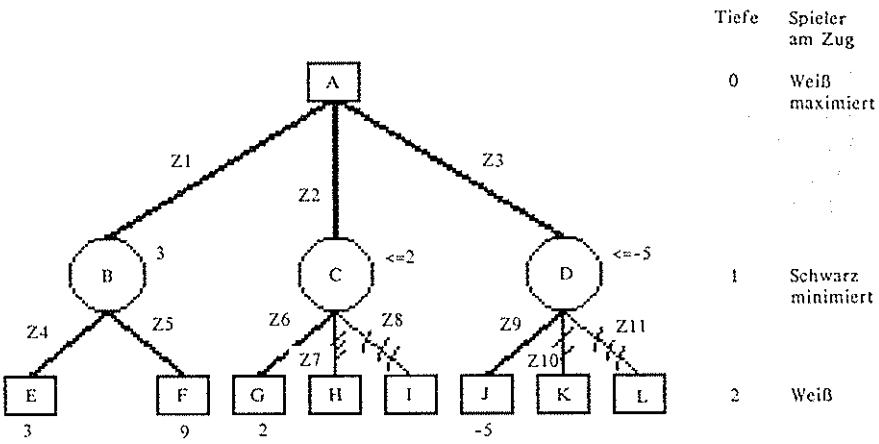
Tiefe	Zahl der Endstellungen
1	40
2	1600
3	64000
4	256000
5	10240000

Selbst ein Programm, das 100 Stellungen pro Sekunde bewerten kann, bräuchte also für Suchtiefe 4 ca. 40 Minuten, für Suchtiefe 5 mehr als einen Tag.

Man könnte nun meinen, das sei nicht weiter verwunderlich, denn diese Suchmethode ist ja auch selten dumm. Warum sollte das Programm auch alle Zugfolgen einer festen Länge untersuchen - natürlich müssen die sinnlosen, schlechten Züge ausgeschlossen werden. Das Programm sollte nur die interessanten Züge untersuchen. Bevor wir uns hierzu eine Meinung bilden, sollten wir aber noch die Kraft des Alpha-Beta-Algorithmus kennenlernen.

3.4.3 Der Alpha-Beta-Algorithmus - Mitdenken spart Zeit

Betrachten wir noch einmal eine Suche der Tiefe 2. Nehmen wir folgendes Beispiel:



Zuerst werden die Züge Z1, Z4 und Z5 untersucht. Dann steht fest, daß Stellung B den Wert 3 hat. Diesen Wert hat Weiß in der Grundstellung sicher - die Züge Z2 oder Z3 wählt er nur, wenn er einen besseren Wert erreichen kann. Nun werden Z2 und Z6 untersucht. Danach steht schon fest, daß der Wert von Stellung C höchstens 2 ist - Schwarz ist am Zug und wird den Zug mit dem kleinsten Wert wählen, einen höheren Wert als 2 muß er in Stellung C nicht zulassen. Somit interessieren die Züge Z7 und Z8 nicht mehr, Weiß darf Z2 sowieso nicht spielen.

Schachspieler merken so etwas und schließen die Untersuchung von Z2 an dieser Stelle ab. Sie würden sagen: "Z6 widerlegt Z2".

Ein Beispiel aus der Praxis ist eine Stellung, in der wir viele gute Züge haben, aber auch einen schlechten. Wenn wir diesen untersuchen und feststellen, daß der Gegner daraufhin unsere Dame gewinnen kann, überlegen wir nicht mehr, ob er uns auch noch mattsetzten kann - der Damenverlust ist abschreckend genug.

Im obigen Beispiel tritt die Situation noch einmal auf. Auch nach der Analyse von Z3 und Z9 müssen die Züge Z10 und Z11 nicht mehr untersucht werden. Einen solchen Ausschluß von Zügen von der Untersuchung nennt man "Cutoff".

Es müssen also nur die im Suchbaum dicker gezeichneten Züge untersucht werden.

Auch unser Programm sollte diese Situationen erkennen. Wir erhalten das gleiche Suchergebnis, können aber die Untersuchung vieler Züge sparen. Dies leistet der Alpha-Beta-Algorithmus.

a) Der Algorithmus

Der Alpha-Beta-Algorithmus kann relativ kurz implementiert werden, ist aber schwer zu verstehen. Die Grundidee ist folgende *(Alpha-Beta-Algorithmus)*:

> Zwei neue Variablen werden eingeführt: *Alpha* und *Beta*. Diese werden während einer Suche auf dem laufenden gehalten und geben immer die Werte an, die Weiß (Alpha) bzw. Schwarz (Beta) durch vorherige Abweichungen von der aktuellen Zugfolge schon "sicher" haben. Stellen wir nun fest, daß die aktuelle Stellung einen Wert hat, der größer als Beta bzw. kleiner als Alpha ist, so ist sie uninteressant - Schwarz bzw. Weiß hatten schon vorher einen besseren Zug.
> Alpha ist dabei immer kleiner als Beta. Man spricht auch vom *Alpha-Beta-Fenster*. Damit ist der Ausschnitt des Wertebereichs gemeint, der zwischen Alpha und Beta liegt. Nur für diese Werte interessieren wir uns wirklich.

Im vorigen Beispiel sieht das dann so aus: Zunächst sind Alpha und Beta auf -16000 und +16000 gesetzt. Nach der Untersuchung von Z1, Z4 und Z5 hat Weiß den Wert 3 sicher - durch Z1. Daher wird Alpha auf 3 erhöht. Nach der Analyse von Z6

weiß die Suche, daß in der Minimax-Wert von Stellung C höchstens 2 ist - und somit außerhalb des Alpha-Beta-Fensters, welches nur noch von 3 bis 16000 reicht.
Den Alpha-Beta-Algorithmus können wir wie folgt in die bisherige Suchroutine einbauen:

```
  100 REM STELLUNG UNTERSUCHEN, WEISS AM ZUG
  110 REM STELLUNG IST INTERN DARGESTELLT
  120 REM T% IST TIEFE DER AKTUELLEN STELLUNG
  130 REM MT% IST DIE SUCHTIEFE
  140 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
  150 REM AL% UND BE% SIND DIE WERTE FUER ALPHA UND BETA
  160 IF T%=MT% THEN GOSUB "BEWERTUNG": MW%=PW%: GOTO 400
  170 MW%=-16000
  180 GOSUB "ZUGGENERATOR"
  190 IF "KEINE ZUEGE MEHR DA" THEN GOTO 310
  200 "WAEHLE ZUG AUS"
  210 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
  220 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
  230 GOSUB 500: REM "UNTERSUCHE STELLUNG FUER SCHWARZ"
  240 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
  250 "ERZEUGE MIT STACK ALTE STELLUNG"
  260 IF AW% <= MW% THEN GOTO 300
  270 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
* 280 IF MW% >= BE% THEN GOTO 310: REM CUTOFF
* 290 IF MW% > AL% THEN AL%=MW%
  300 GOTO 190
  310 REM SUCHSCHLEIFE FERTIG
  400 RETURN: REM ENDE DER STELLUNGSANALYSE FÜR WEISS
```

Kommentare:

In Zeile 280 wird abgefragt, ob der Wert, den Schwarz sicher hat, bereits überschritten ist. Dann ist die Stellung zu gut für Weiß. (mw% kann nur noch wachsen.) Schwarz hatte schon vorher bessere Züge zur Verfügung. Die restlichen Züge müssen also nicht mehr untersucht werden. (Diesen *Cutoff* nennt man *Beta-Cutoff*, da der Vergleich mit Beta zum Abbruch führt.)

Wenn andererseits mw% kleiner als Alpha ist, können wir nicht abbrechen. mw% kann noch steigen und damit größer als Alpha werden.

Wir müssen aber Alpha erhöhen (Zeile 290), wenn mw% größer als der aktuelle Wert von Alpha ist. Damit teilen wir der Suche mit, daß Weiß durch den eben untersuchten Zug den Wert von mw% sicher hat, so daß dies berücksichtigt werden kann, wenn die die alternativen Züge analysiert werden.

Mit nur zwei zusätzlichen Zeilen sind wir also fertig - in der einen wird die Cutoff-Bedingung abgefragt und in der anderen der Wert, den Weiß sicher hat, neu berechnet.

Wäre Schwarz am Zug, so wäre alles umgekehrt: mw% könnte bei weiterer Suche fallen - die Rollen von Alpha und Beta wären vertauscht. Hier tritt dann der *Alpha-Cutoff* auf.

Vor dem ersten Aufruf der Suche in der Grundstellung müssen Alpha und Beta auf die Werte -16000 und 16000 gesetzt werden. Diese Werte sind außerhalb des Wertebereichs für Stellungen und zeigen an, daß noch kein Spieler etwas sicher hat.

Al% und be% sind rekursive Variablen. Wenn wir in einer Stellung einen Wert sicher haben, so noch lange nicht in der Vorgängerstellung. Wir müssen also die Veränderungen (Zeile 290) zurücksetzen.

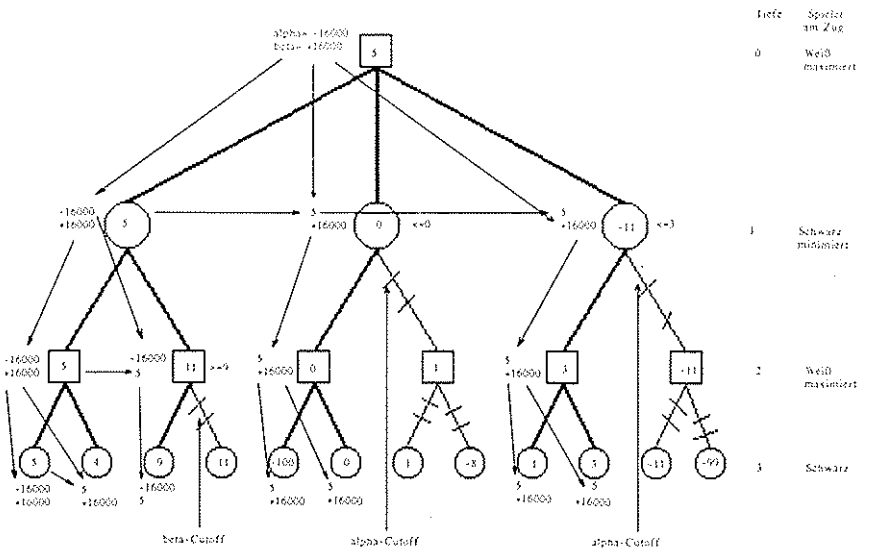
Für Schwarz ist die Implementierung des Alpha-Beta-Algorithmus analog:

```
500 REM STELLUNG UNTERSUCHEN, SCHWARZ AM ZUG
510 REM STELLUNG IST INTERN DARGESTELLT
520 REM T% IST TIEFE DER AKTUELLEN STELLUNG
530 MT% IST DIE SUCHTIEFE
540 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
550 REM AL% UND BE% SIND DIE WERTE FUER ALPHA UND BETA
560 IF T%=MT% THEN GOSUB "BEWERTUNG": MW%=PW%: GOTO 800
```

```
  570 MW%= 16000
  580 GOSUB "ZUGGENERATOR"
  590 IF "KEINE ZUEGE MEHR DA" THEN GOTO 710
  600 "WAEHLE ZUG AUS"
  610 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
  620 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
  630 GOSUB 100: REM "UNTERSUCHE STELLUNG FUER WEISS"
  640 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
  650 "ERZEUGE MIT STACK ALTE STELLUNG"
  660 IF AW% >= MW% THEN GOTO 700
  670 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
* 680 IF MW% <= AL% THEN GOTO 710: REM CUTOFF
* 690 IF MW% < BE% THEN BE%=MW%
  700 GOTO 190
  710 REM SUCHSCHLEIFE FERTIG
  800 RETURN: REM ENDE DER STRELLUNGSANALYSE FUER SCHWARZ
```

b) Ein Beispiel

Nachdem wir den Suchalgorithmus definiert haben, wollen wir noch an einem Beispiel betrachten, wie die Alpha- und Beta-Werte gesetzt werden. Der Suchbaum hat diesmal die Tiefe 3. Wir gehen wie immer davon aus, daß er von links nach rechts abgesucht wird.



Die dicker gezeichneten Züge sind wieder die, die wirklich untersucht werden müssen. Die weggeschnittenen sind durchgestrichen.

Links neben den Stellungen sind die Alpha- und Beta-Werte, mit denen sie untersucht werden, eingezeichnet. Die korrekten Minimax-Werte sind in die Knoten geschrieben. Rechts neben einige Stellungen ist die Abschätzung geschrieben, die zu einem Cutoff führt.

Die zusätzlichen Pfeile geben an, woher die Alpha- bzw. Beta-Werte kommen.

Wir sehen, daß zunächst die Alpha- und Beta-Werte einfach nach unten weitergegeben werden. Wird aber in einer Stellung ein zweiter oder dritter Zug untersucht, so können die Werte der Vorgängerzüge zum Alpha- oder Beta-Wert werden - und zwar zum Alpha-Wert, wenn Weiß am Zug ist, sonst zum Beta-Wert.

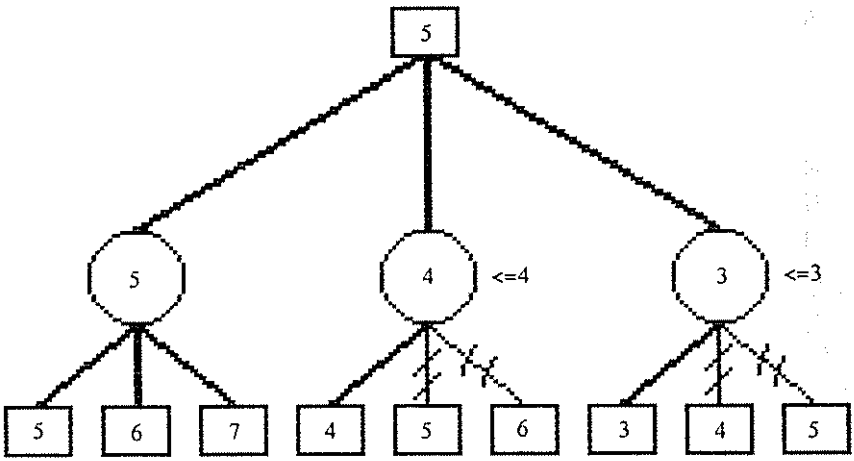
Es gibt zwei Arten von Cutoffs: den Alpha-Cutoff, wenn der Alpha-Wert unterschritten ist, und den Beta-Cutoff, wenn der

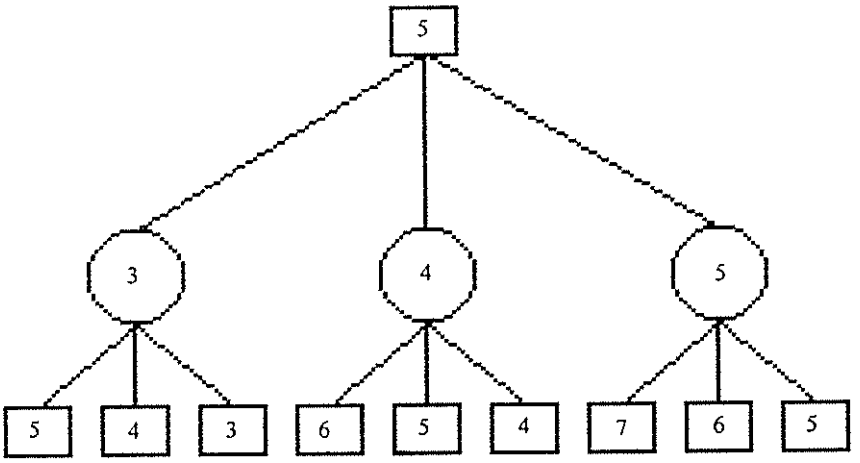
Beta-Wert überschritten ist. Der Erstere kann nur auftreten, wenn Weiß am Zug ist, der andere nur, wenn Schwarz am Zug ist.

Im Beispiel treten ein Beta-Cutoff (bei Tiefe 2, Weiß am Zug), und zwei Alpha-Cutoffs (bei Tiefe 1) auf.

c) Die Wirksamkeit das Alpha-Beta-Algorithmus

Qualitativ ist die Wirksamkeit schnell verdeutlicht. Betrachten wir zwei Suchbäume im Vergleich:





Die Bäume sind fast identisch, nur die Reihenfolge, in der die Werte an die Endknoten geschrieben sind, ist umgekehrt.

Werden beide Bäume von links nach rechts abgesucht, so treten beim ersten 2 Cutoffs auf, beim zweiten keine.

Das liegt daran, daß beim ersten Baum der beste Zug immer zuerst untersucht wird, beim zweiten nicht. Würden wir den zweiten von rechts nach links absuchen, also in jeder Stellung

zuerst den ganz rechts gezeichneten Zug analysieren, hätten wir auch hier die Cutoffs.

Generell gilt:

> Wenn zuerst immer der beste Zug untersucht wird, treten die meisten Cutoffs auf.
> Untersuchen wir die Züge in anderer Reihenfolge, haben wir eventuell gar keine Cutoffs.

Soweit die qualitative Abschätzung. Doch wieviele Stellungen könnnen wir wirklich einsparen?

Gehen wir wieder davon aus, daß in jeder Stellung 40 Züge möglich sind und daß wir bis zur Suchtiefe mt analysieren. Außerdem nehmen wir an, daß immer der beste Zug zuerst untersucht wird, daß wir also alle denkbaren Cutoffs erhalten. Berechnen wir die Zahl der Endstellungen.

Bei Suchtiefe 1 können keine Cutoffs auftreten. Es gibt also 40 Endstellungen.

Bei Suchtiefe 2 wird zunächst der beste Zug untersucht. Alle Gegenzüge müssen berechnet werden. Macht 40 Endstellungen (Vgl. auch vorstehende Abbildung, 1. Baum, Z1). Die schlechteren Züge in der Partiestellung kommen nun dran. Auf sie muß jedoch immer nur ein Gegenzug, der Widerlegungszug, analysiert werden, denn dieser führt sofort einen Cutoff herbei (vgl. vorstehende Abbildung, 1. Baum). Somit erhalten wir für die 39 Alternativzüge nur 39 Endstellungen, insgesamt also 79.
Allgemein gilt: Sind in jeder Stellung n Züge möglich, wird bis zur Tiefe t gesucht und wird immer der beste Zug zuerst analysiert, so müssen

$$n^{(-)t/2}+n^{(+)t/2}-1$$

Endstellugen untersucht werden. ('(-)t/2' bedeutet t/2 abgerundet, '(+)t/2' bedeutet t/2 aufgerundet.)

Betrachten wir die Zahlenwerte in einer Tabelle:

Tiefe	Zahl der Endstellungen bester Fall	schlechtester Fall
1	40	40
2	79	1600
3	1639	64000
4	3199	2560000
5	65569	102400000

Die Zahlen sprechen für sich: Können wir den Alpha-Beta-Algorithmus effizient anwenden, muß ein Programm nur einen Bruchteil der Arbeit machen. Es kann in der gleichen Zeit fast doppelt so tief suchen (vgl. Suchtiefe 2 / schlechtester Fall - Suchtiefe 4 / bester Fall).

d) Resumee

Eine Suche mit Alpha-Beta-Algorithmus berechnet das gleiche Ergebnis wie die normale Minimax-Suche. Der Aufwand verringert sich dramatisch, aber nur, wenn die Züge gut sortiert sind. Durch die Cutoffs geht keine Information verloren.

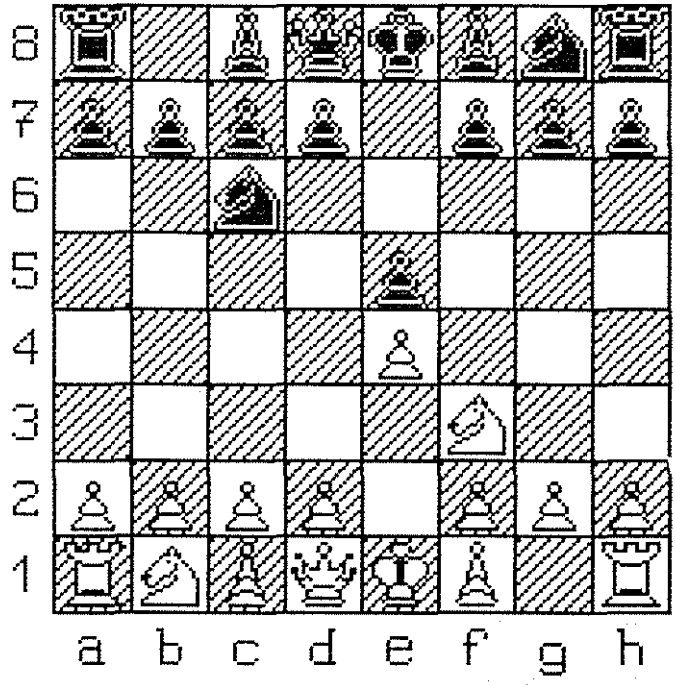
Slate und Atkin waren die ersten, die die Möglichkeiten dieser Methode im vollen Umfang erkannten. Logisch gesehen werden weniger die schlechten als vielmehr die unwichtigen Züge weggeschnitten. (Hat ein Zug eine Variante widerlegt, erfolgt der Cutoff. Möglicherweise gibt es noch stärkere Widerlegungen - aber eine genügt.) Versuchte man zuvor, schlechte Züge auszuschließen und riskierte schwere Fehlentscheidungen, so bemühten sich Slate und Atkin, die guten zu erraten. Ging es schief, würde das Programm länger rechnen - übersehen würde es nichts.

3.4.4 Die Ruhesuche

Die bisherige Suchmethode läßt noch manches zu wünschen übrig. Besonders kritisch ist, daß die Suche einfach bei einer

bestimmten Tiefe aufhört. Diese Tiefenschicht nennt man auch *Horizont*, da das Programm nicht darüber hinaus sehen kann.

Beschäftigen wir uns ganz praktisch damit, welchen Zug das Programm wählt, *wenn es bis zur Tiefe 1 sucht*. Nehmen wir die Stellung nach den Zügen 1. e4 e5 2. Sf3 Sc6.



Weiß ist am Zug, das Programm soll den Zug machen. Es erzeugt also alle Folgestellungen und läßt sie bewerten.

Von der Bewertungsfunktion wissen wir, daß der Materialterm gegenüber dem positionellen Term dominiert (vgl. 3.4.1). Das heißt: ein Mehrbauer wiegt stärker als alle strategischen Merkmale zusammen. Schlägt ein Spieler nur einen gegnerischen Bauern, so erhält er damit bessere Werte als mit dem besten strategischen Zug.

Dann wissen wir aber auch, welchen Zug das Programm in der obigen Stellung (bei Suchtiefe 1) macht: Es spielt Se5:! Diese Stellung hat eine höhere Bewertung als alle anderen, denn Weiß hat einen Bauern mehr. Daß Schwarz im nächsten Zug zurückschlagen kann, sieht das Programm wegen der Tiefenbeschränkung nicht!

Das darf nicht sein. Um Abhilfe zu schaffen, haben wir folgende Möglichkeiten:

A) Die Bewertungsfunktion muß erkennen, daß der Mehrbauer nicht zählt, weil Schwarz zurückschlagen kann und damit eine Figur gewinnt.

B) Nach einem Schlagzug wird immer eine Stufe tiefer gesucht, damit die Gegenzüge berücksichtigt werden können.

C) Ist eine Horizontstellung erreicht, werden zwar normale Züge nicht mehr untersucht, Schlagzüge aber doch.

Lösung A scheidet aus, da die Bewertungsfunktion damit überlastet ist. Nicht immer ist die Situation so einfach wie hier, häufig entscheiden lange Schlagfolgen darüber, ob ein Zug gut ist oder nicht.

Lösung B ist sehr teuer, da nach jedem Schlagzug durchnittlich 40 neue Züge untersucht werden müßten. Der Suchbaum würde stark wachsen, auch der Alpha-Beta-Algorithmus könnte das

nicht verhindern. Außerdem leistet der Ansatz nicht das gewünschte. Auch wenn der letzte Zug der Variante kein Schlagzug war, kann eine Figur velorengehen. In der Beispielstellung müßte z.B. erkannt werden, daß der Springer nach Sg5 verloren geht.

Bleibt nur noch Ansatz C. Dieser wird von praktisch allen Programmen gemacht und *Ruhesuche* genannt.

Der Name soll ausdrücken, daß die Bewertungsfuktion nur *"ruhige Stellungen"* beurteilen darf. Ist eine Stellung nicht ruhig (hier: sind Schlagzüge möglich), so darf die Stellung nicht für sich allein bewertet werden; die "Unruhefaktoren" (hier: die Schlagzüge) müssen untersucht werden.

Eine solche Ruhesuche wird nicht nur nach Schlagzügen gemacht, sondern in allen Stellungen, die nicht mehr zur normalen Suche gehören.

Wie sollen die Stellungen der Ruhesuche bewertet werden? Es werden ja nicht alle Züge untersucht, sondern nur die Schlagzüge. Die Methode ist folgende *(Ruhesuche)*:

> In allen Stellungen jenseits des Suchhorizonts wird zunächst die Bewertungsfunktion aufgerufen. Nun wird angenommen, daß der am Zug befindliche Spieler deren Wert sicher hat, aber es wird noch geprüft, ob er sich durch *Schlagzüge oder Bauernumwandlungen* verbessern kann. Diese Züge werden nacheinander untersucht. Liefern sie Werte, die besser sind, so werden diese zum Stellungswert, sonst die Bewertung.

In der Beispielstellung würde also nach Sf3xe5 der Schlagzug Sc6xe5 doch noch untersucht, die Folgestellung würde bewertet und es würde sich herausstellen, daß Weiß den Bauern nicht nehmen darf.

a) Erste Implementierung

Auch die Ruhesuche kann mit wenig Schreibaufwand eingebaut werden. Betrachten wir die Routine für Weiß. Zunächst wird wie bisher schon in Horizontstellungen die Bewertungsfunktion aufgerufen. Deren Wert hat der Spieler, der am Zug ist, sicher. Dies kann dadurch realisiert werden, daß mw% nicht mit -16000 (bzw. +16000) initialisiert wird, sondern mit der Bewertung. Anschließend muß nun die Suchschleife angesprungen werden. Hierzu muß lediglich Zeile 160 ersetzt werden durch:

```
160 IF T% >= MT% THEN GOSUB "BEWERTUNG": MW%=PW%: GOTO 180
```

In Stellungen der Ruhesuche dürfen nur die Schlagzüge ausprobiert werden. Dies erreichen wir, wenn wir Zeile 190 ersetzen durch:

```
190 IF "KEINE SCHLAGZÜGE MEHR DA" AND T% >= MT% THEN GOTO 310
191 IF "KEINE ZÜGE MEHR DA" THEN GOTO 310
```

Die Routine kann dann Ruhesuche und normale Suche zugleich berechnen.

b) Aber bitte mit forward-pruning!

Auch hier kann der Alpha-Beta-Algorithmus gewinnbringend eingesetzt werden. Schließlich hat der Spieler, der am Zug ist, ja den Wert der Bewertungsfunktion sicher!

Ist Weiß am Zug und dieser Wert größer als Beta, können wir auch hier aufhören. Die Schlagzüge können den Wert nur steigern, den Stellungswert also nur weiter aus dem Alpha-Beta-Fenster treiben.

Im Gegensatz zu bisherigen Cutoffs brechen wir die Suche ab, ohne auch nur einen Zug zu untersuchen. Derartige Verfahren fallen unter den Sammelbegriff *"forward-pruning"*.

Ist die Cutoff-Bedingung nicht erfüllt, so ist innerhalb der weiteren Suche zumindest die aktuelle Bewertung für Weiß sicher, Alpha kann also ggf. auf diesen Wert erhöht werden.

In unserer BASIC-Routine sieht das dann so aus:

```
  100 REM STELLUNG UNTERSUCHEN, WEISS AM ZUG
  110 REM STELLUNG IST INTERN DARGESTELLT
  120 REM T% IST TIEFE DER AKTUELLEN STELLUNG
  130 REM MT% IST DIE SUCHTIEFE
  140 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
  150 REM AL% UND BE% SIND DIE WERTE FUER ALPHA UND BETA
* 160 VS% = (T% < MT%)
* 161 IF VS% THEN MW%=-16000: GOTO 180
* 162 GOSUB "BEWERTUNG": MW%=PW%
* 163 IF MW% GRÖSSER/GLEICH BE% THEN GOTO 400: REM CUTOFF
* 164 IF MW% > AL% THEN AL%=MW%
  180 GOSUB "ZUGGENERATOR"
* 190 IF "KEINE SCHLAGZUEGE MEHR DA" AND NOT VS% THEN GOTO 310
* 191 IF "KEINE ZUEGE MEHR DA" THEN GOTO 310
  200 "WAEHLE ZUG AUS"
  210 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
  220 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
  230 GOSUB 500: REM "UNTERSUCHE STELLUNG FUER SCHWARZ"
  240 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
  250 "ERZEUGE MIT STACK ALTE STELLUNG"
  260 IF AW% <= MW% THEN GOTO 300
  270 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
  280 IF MW% >= BE% THEN GOTO 310: REM CUTOFF
  290 IF MW% > AL% THEN AL%=MW%
  300 GOTO 190
  310 REM SUCHSCHLEIFE FERTIG
  400 RETURN: REM ENDE DER STELLUNGSANALYSE FUER WEISS
```

Kommentare:

Zunächst wird (nur wegen der Übersichtlichkeit) die Variable vs% eingebaut (Zeile 160). Diese gibt an, ob das Programm sich noch in der vollen Suche (daher vs) oder

schon in der Ruhesuche befindet. (Wert -1 bedeutet ja, Wert 0 nein.)

Falls Ruhesuche vorliegt (Zeile 161), wird mw% entsprechend gesetzt (Zeile 162). Dann wird der Wert mit Beta (be%) verglichen (Zeile 163). Ist er schon größer, sind wird fertig - mehr wollten wir nicht wissen, ein Cutoff (forward-pruning) liegt vor. Sonst müssen wir den Alpha-Wert erhöhen (Zeile 164). Damit teilen wir der weiteren Suche mit, daß die Schlagzüge uns nur interessieren, wenn sie mw% - den Wert, den Weiß sicher hat - erhöhen.
Nun ist die Ruhesuche initialisiert und wir springen die Suchschleife an (Zeile 165).
An der Initialisierung in der vollen Suche hat sich nichts geändert.

Für Schwarz ist die Implementierung der Ruhesuche analog, wie immer tauschen Alpha und Beta die Rollen:

```
  500 REM STELLUNG UNTERSUCHEN, SCHWARZ AM ZUG
  510 REM STELLUNG IST INTERN DARGESTELLT
  520 REM T% IST TIEFE DER AKTUELLEN STELLUNG
  530 MT% IST DIE SUCHTIEFE
  540 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
  550 REM AL% UND BE% SIND DIE WERTE FUER ALPHA UND BETA
* 560 VS% = (T% < MT%)
* 561 IF VS% THEN MW%=16000: GOTO 580
* 562 GOSUB BEWERTUNG: MW%=PW%
* 563 IF MW% <= AL% THEN GOTO 800: REM CUTOFF
* 564 IF MW% < BE% THEN BE%=MW%
  580 GOSUB "ZUGGENERATOR"
* 590 IF "KEINE SCHLAGZUEGE MEHR DA" AND NOT VS% THEN GOTO 710
* 591 IF "KEINE ZUEGE MEHR DA" THEN GOTO 710
  600 "WAEHLE ZUG AUS"
  610 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
  620 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
  630 GOSUB 100: REM "UNTERSUCHE STELLUNG FUER WEISS"
  640 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
  650 "ERZEUGE MIT STACK ALTE STELLUNG"
```

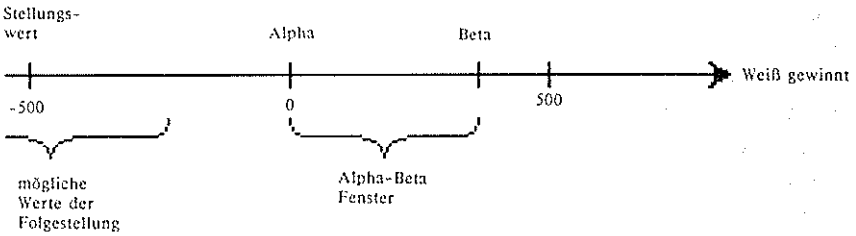
```
660 IF AW% >= MW% THEN GOTO 700
670 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
680 IF MW% <= AL% THEN GOTO 710: REM CUTOFF
690 IF MW% < BE% THEN BE%=MW%
700 GOTO 190
710 REM SUCHSCHLEIFE FERTIG
800 RETURN: REM ENDE DER STELLUNGANALYSE FUER SCHWARZ
```

c) Und noch einmal forward-pruning!

Es gibt eine weitere Cutoff-Möglichkeit, die wir uns nicht entgehen lassen wollen. Schlagzüge untersuchen wir in der Ruhesuche, weil die Folgestellungen erheblich bessere Bewertungen haben können - beliebig groß kann die Differenz aber nicht werden. Schlagen wir einen Bauern, wird der Wert um ca. 100 Punkte steigen - mehr nicht.

Machen wir ein Beispiel:

> Die Ruhesuche erreicht eine Stellung (Weiß am Zug) mit Bewertung -500; Alpha und Beta stehen auf 0 und 200. Sollen wir dann einen Zug untersuchen, der einen Bauern schlägt?
> Nein, der Wert der Folgestellung ist bestenfalls -300, ins Alpha-Beta-Fenster kommen wir nicht.



Auch dieses Verfahren gehört zum weiten Bereich des *forward-pruning*. Die Abschätzung, die zum Cutoff führt, basiert auf Vorausüberlegungen, nicht auf Suchwerten.

Bemerkung: Dieses Verfahren ist nicht völlig verträglich mit dem Alpha-Beta-Algorithmus. Die Vorausabschätzung muß nicht richtig sein - wenn z.B. ein Bauer mit Schach geschlagen wird, setzen wir vielleicht matt, obwohl wir nur einen Zugewinn von 100 Punkten erwarteten. Aber dies geschieht so selten, daß der Cutoff insgesamt das Programm verbessert.

Die Implementierung soll nur für Weiß vorgeführt werden:

```
  100 REM STELLUNG UNTERSUCHEN, WEISS AM ZUG
  110 REM STELLUNG IST INTERN DARGESTELLT
  120 REM T% IST TIEFE DER AKTUELLEN STELLUNG
  130 REM MT% IST DIE SUCHTIEFE
  140 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
  150 REM AL% UND BE% SIND DIE WERTE FUER ALPHA UND BETA
  160 VS% = (T% < MT%)
  161 IF VS% THEN MW%=-16000: GOTO 180
  162 GOSUB "BEWERTUNG": MW%=PW%
  163 IF MW% >= BE% THEN GOTO 400: REM CUTOFF
  164 IF MW% > AL% THEN AL%=MW%
  180 GOSUB "ZUGGENERATOR"
  190 IF "KEINE SCHLAGZUEGE MEHR DA" AND NOT VS% THEN GOTO 310
  191 IF "KEINE ZUEGE MEHR DA" THEN GOTO 310
  200 "WAEHLE ZUG AUS"
* 205 IF VS% OR PW%+"MAXIMALER ZUGEWINN" > AL% THEN GOTO 210
* 206 AW%= PW%+"MAX. ZUGEWINN"
* 207 GOTO 260: REM FORWARD-PRUNING, CUTOFF
  210 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
  220 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
  230 GOSUB 500: REM "UNTERSUCHE STELLUNG FUER SCHWARZ"
  240 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
  250 "ERZEUGE MIT STACK ALTE STELLUNG"
  260 IF AW% <= MW% THEN GOTO 300
  270 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
  280 IF MW% >= BE% THEN GOTO 310: REM CUTOFF
  290 IF MW% > AL% THEN AL%=MW%
```

```
300 GOTO 190
310 REM SUCHSCHLEIFE FERTIG
400 RETURN: REM ENDE DER STELLUNGSANALYSE FUER WEISS
```

Kommentare:

Falls ein Schlagzug der Ruhesuche nicht ins Alpha-Beta-Fenster führt (Zeile 205), wird der Wert der Folgestellung nach oben abgeschätzt (Zeile 206) und die Endauswertung wird angesprungen (Zeile 207).

d) Aufwand der Ruhesuche

Eigentlich sollte die Ruhesuche sehr billig sein. Es werden nur Schlagzüge untersucht. Wir haben neben dem Alpha-Beta-Algorithmus zwei neue Cutoffs aus dem Bereich des forward-pruning eingeführt. Bei einem müssen sogar überhaupt keine Züge mehr untersucht werden.

Dennoch wächst der Suchbaum erheblich - mal um den Faktor 2, manchmal sogar auf das Zehnfache.

Einmal liegt das daran, daß wir die Ruhesuche unbedingt brauchen. In jeder Horizontstellung muß das Programm prüfen, was die Schlazüge bringen. Gewinnen sie Material, muß die Suche unweigerlich die entsprechenden Varianten nachrechnen - da kann und darf kein Cutoff helfen. Schon dies kostet viel Zeit.

Zum anderen sind da folgende, nun allerdings ärgerliche Situationen: Das Programm untersucht einen relativ schlechten Zug von Weiß und erreicht über diesen eine Horizontstellung, in der wieder Weiß am Zug ist. Anstatt nun die Bewertung zu akzeptieren und die Variante fallen zu lassen, probiert es nacheinander alle Schlagzüge durch, obwohl alle angegriffenen Figuren zuverlässig gedeckt sind - theoretisch könnte sich Weiß ja verbessern. Auch hier helfen die Cutoffs nicht. Nützlich wäre an dieser Stelle eine Vorbewertung der Schlagzüge, die die völlig

unsinnigen ausschließt. Unser Programm geht hierauf nicht ein, im theoretischen Teil werden aber Lösungen vorgeschlagen.

Also: Die Ruhesuche ist notwendig, kostet aber viel Zeit. Mit mehr Schachwissen könnte das Programm hier einiges sparen.

3.4.5 Schach, Matt, Patt und die Ruhesuche

Mittlerweile verfügen wir über eine Suche, die die meisten Stellungen schon recht gut behandelt. Der wichtigste Aspekt des Spiels entgeht ihr aber noch: das *Erkennen von Mattsituationen.*

Wie zuvor erwähnt, ist die Bewertungsfunktion damit überfordert, dies muß die Suche übernehmen. Die Lösung betrachten wir im ersten Teil dieses Kapitels.

Im zweiten Teil überlegen wir dann, wie mit vertretbarem Aufwand Schachmatts auch in der Ruhesuche erkannt werden können.

a) Matt und Patt

Nach den Spielregeln ist ein Spieler schachmatt, wenn er am Zug ist, sein König im Schach steht und er dies mit keinem Zug ändern kann. Steht in einer solchen Situation sein König nicht im Schach, ist er Patt.

Dies muß die Suche erkennen. Folgende Probleme treten auf:

1) Wir müssen festlegen, welcher Wert einem Matt zugeordnet werden soll.

2) Wir müssen erkennen, ob der König des am Zug befindlichen Spielers im Schach steht.

3) Der Zuggenerator erzeugt nicht nur die legalen Züge (also die, die das Schach beseitigen), sondern alle denkbaren. Er generiert also auch unzulässige Züge. Diese müssen eliminiert werden.

Ist ein Spieler matt, so hat er verloren. Dies muß durch extreme Stellungswerte ausgedrückt werden. Bei unserem Programm sind dies +/-16000 (Schwarz ist matt/Weiß ist matt). Wichtig ist aber auch, daß wir zwischen einem direkten Matt und einem Matt in mehreren Zügen unterscheiden - im ersten Fall soll das Programm "Ich bin matt" bzw. "Ich setze matt" ausgeben, im zweiten dagegen einen Zug ausführen. Daher bewerten wir ein sofortiges Matt extremer als eines in mehreren Zügen. Kann der weiße (schwarze) König zwangsläufig nach n (Halb-)Zügen mattgesetzt werden, so weisen wir der Stellung den Wert -16000+n (16000-n) zu.

Der Wert für ein Patt ist selbstverständlich 0.

Auf das zweite Problem sollten wir hier nicht eingehen, da es ein rein technisches ist. Es soll von einer Hilfsroutine gelöst werden, die die Variable sh% entsprechend setzt.

Für den dritten Punkt kommen mehrere Lösungen in Betracht. Die effizienteste wäre, vor der Untersuchung eines Zuges zu prüfen, ob nach dessen Ausführung der eigene König im Schach steht. Leider ist dies extrem programmieraufwendig. Schon in normalen Stellungen muß geprüft werden, ob die ziehende Figur gefesselt ist. Liegt ein Schachgebot vor, muß zusätzlich getestet werden, ob die schachgebende Figur geschlagen wird oder ob die ziehende Figur sich der schachgebenden in den Weg stellt. Liegt ein Doppelschach vor, geben also zwei Figuren gleichzeitig Schach, ist die Lage wieder anders. Dann können nur noch Königszüge die Drohung beseitigen.

Um den Aufwand in Grenzen zu halten, entschieden wir uns für eine einfachere, aber laufzeitintensivere Lösung. Alle, also auch die illegalen Züge, werden untersucht. Ob sie unzulässig sind, wird erst in der Folgestellung getestet. Dies kann leicht geprüft werden - ein Zug ist illegal, wenn in der Folgestellung der König geschlagen werden kann. Wir müssen also nur die Schlagzüge durchgehen. Ist der Test positiv, wird der Stellung ein Extremwert zugeordnet. Bsp.: Kann in einer Stellung der

Tiefe t Schwarz den weißen König schlagen, so erhält sie den Wert -16000+t-1, bei vertauschten Farben 16000-t+1.

In der normalen, erschöpfenden Suche können wir nun Matt und Patt erkennen. Ist Weiß in einer Stellung der Tiefe t matt, so bleibt der Minimaxwert (mw%) auf -16000+t stehen - nach allen (illegalen) Zügen kann Schwarz in der (illegalen) Folgestellung der Tiefe t+1 den weißen König schlagen. Kann Weiß dagegen erst in mehreren Zügen mattgestzt werden, so liegt der Wert über -16000+t.

Auf die gleiche Art bemerken wir auch ein Patt. Der Minimaxwert mw% ist nach der Untersuchung der gleiche - auch hier ist kein Zug möglich. Daher muß nur geprüft werden, ob der König im Schach steht. Ist dies nicht der Fall, so ist Weiß patt - der Minimaxwert mw% wird auf 0 gesetzt.

Diese Überprüfung auf Matt und Patt funktioniert nur in der normalen, erschöpfenden Suche. In der Ruhesuche werden bisher jedoch nur die Schlagzüge analysiert. Dann kann nicht festgestellt werden, ob einer der vielen ruhigen Züge den König rettet.

Die erweiterte Suchroutine sieht nun so aus:

```
  100 REM STELLUNG UNTERSUCHEN, WEISS AM ZUG
  110 REM STELLUNG IST INTERN DARGESTELLT
  120 REM T% IST TIEFE DER AKTUELLEN STELLUNG
  130 REM MT% IST DIE SUCHTIEFE
  140 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
  150 REM AL% UND BE% SIND DIE WERTE FUER ALPHA UND BETA
* 151 GOSUB "ZUGGENERATOR"
* 155 GOSUB "ILLEGALE STELLUNG?": IF A1% THEN MW%=16001-T%: GOTO 400
* 156 GOSUB "LIEGT EIN SCHACHGEBOT VOR?"
  160 VS% = (T% < MT%)
* 161 IF VS% THEN MW%=-16000+T%: GOTO 180
  162 GOSUB "BEWERTUNG": MW%=PW%
  163 IF MW% >= BE% THEN GOTO 400: REM CUTOFF
  164 IF MW% > AL% THEN AL%=MW%
  180 REM SUCHSCHLEIFE
```

```
  190 IF "KEINE SCHLAGZUEGE MEHR DA" AND NOT VS% THEN GOTO 310
  191 IF "KEINE ZUEGE MEHR DA" THEN GOTO 310
  200 "WAEHLE ZUG AUS"
  205 IF VS% OR PW%+"MAXIMALER ZUGEWINN" > AL% THEN GOTO 210
  206 AW%=PW%+"MAXIMALER ZUGEWINN"
  207 GOTO 260: REM FORWARD-PRUNING, CUTOFF
  210 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
  220 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
  230 GOSUB 500: REM "UNTERSUCHE STELLUNG FUER SCHWARZ"
  240 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
  250 "ERZEUGE MIT STACK ALTE STELLUNG"
  260 IF AW% <= MW% THEN GOTO 300
  270 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
  280 IF MW% >= BE% THEN GOTO 310: REM CUTOFF
  290 IF MW% > AL% THEN AL%=MW%
  300 GOTO 190
  310 REM SUCHSCHLEIFE FERTIG
* 360 REM AUF PATT PRUEFEN
* 370 IF MW%=-16000+T% THEN IF NOT SH% THEN MW%=0
  400 RETURN: REM ENDE DER STELLUNGSANALYSE FUER WEISS
```

Kommentare:

In Zeile 155 wird neu geprüft, ob es sich um eine illegale Stellung handelt. Die angesprungene Routine testet, ob der gegnerische König geschlagen werden kann. Ist dies der Fall, setzt sie die Hilfsvariable a1% auf -1. Dann wird der Stellungswert entsprechend auf "Gewinn" gesetzt, die Untersuchung ist fertig.

Die eben aufgerufene Hilfsroutine benötigt die Zugliste. Deshalb wurde der Aufruf des Zuggeneators vorgezogen (Zeile 151).

In Zeile 156 prüft das Programm, ob der König im Schach steht. Die Variable sh% wird entsprechend gesetzt.

In Zeile 170 wird berücksichtigt, daß die Stellung über t% Halbzüge erreicht wurde und somit der Stellungswert auch nur ein Matt in t% Halbzügen ausdrücken darf.

In Zeile 370 wird schließlich das Patt erkannt. War kein Zug möglich, und steht der König nicht im Schach (not sh%), dann ist der Stellungswert 0.

Für Schwarz ist die Realisierung analog, sie wird im nächsten Abschnitt mit angegeben.

Die eben eingebaute Verbesserung des Suchalgorithmus kostet an zwei Stellen Zeit: wenn geprüft wird, ob die Stellung illegal ist und ob der König im Schach steht. Es müssen keine zusätzlichen Stellungen analysiert werden, da wir uns noch in der erschöpfenden Suche befinden.

b) Die Ruhesuche und Schachgebote

Selbstverständlich sollte auch die Ruhesuche ein Matt erkennen. Bisher wurden hier nur Schlagzüge zusätzlich untersucht. Die Überlegung war, daß diese eventuell den Stellungswert gegenüber der augenblicklichen Bewertung stark verbessern können.

Dies reicht nun nicht mehr aus. Wenn ein Schach vorliegt und wir dessen Konsequenzen in der Ruhesuche berechnen wollen, müssen wir folgendes bedenken:

1) Der Stellungswert kann sich gegenüber der aktuellen Bewertung (aus der Sicht des bedrohten Spielers, der am Zug ist) nicht nur verbessern (wenn z.B. die schachgebende Figur geschlagen wird), sondern auch verschlechtern (z.B.: Matt, Materialverlust wegen erzwungener Züge).

2) Eventuell müssen wie in der normalen Suche alle Züge ausprobiert werden um festzustellen, ob das Schach beseitigt werden kann. Dies bedeutet in der Ruhesuche einen erheblichen Zeitverbrauch.

3) Die vielen unzulässigen Züge, die vom Zuggenerator erzeugt werden, fallen besonders ins Gewicht.

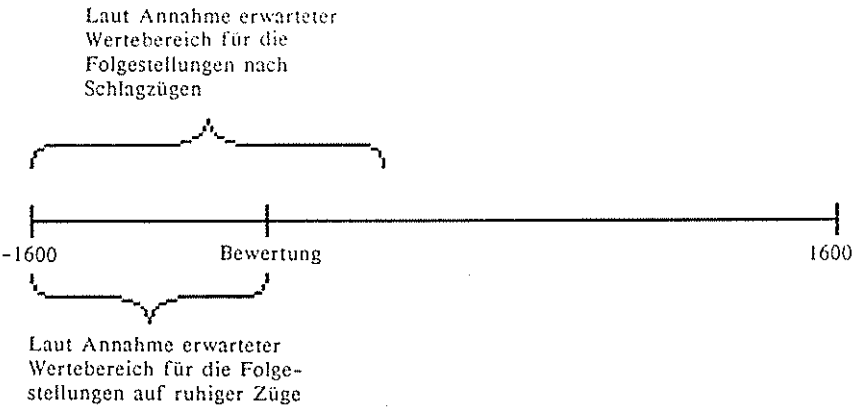
Die einfachste Lösung wäre, bei einem Schach wie in der normalen, erschöpfenden Suche vorzugehen, also alle Züge, bis auf Alpha-Beta-Cutoffs, zu untersuchen, so wie dies im vorigen Abschnitt besprochen wurde. Doch wie gesagt, in der Ruhesuche bedeutet dies einen erheblichen zusätzlichen Zeitaufwand, auch wenn nur selten ein Schachgebot vorliegt.

Mit einigen Überlegungen können wir aber neue, sinnvolle Cutoffs einführen.

Zunächst berechnen wir auch hier die Bewertungsfunktion. Dann machen wir zwei Annahmen:

1) Nur Schlagzüge können deren Wert verbessern.

2) Andere Züge können bestenfalls zu diesem Wert führen.



Wertebereiche laut Annahme, wenn Weiß am Zug ist.

Bemerkung: Diese Überlegungen stehen im Einklang mit der bisherigen Ruhesuche. Auch dort nahmen wir an, daß nur Schlagzüge den Stellungswert erheblich verbessern können (und somit untersucht werden müssen). Von den übrigen Zügen erwarteten wir, daß keiner zu besseren Werten als die Bewertung führt, einer aber diesen Wert erreichen kann (und somit die Bewertung den korrekten Wert der Minimaxbildung über diese Züge angibt). Nur die Hoffnung, daß einer der vielen ruhigen Züge den Bewertungswert erreicht, müssen wir wegen des Schachs fallenlassen - z.B. könnte ein Matt vorliegen.

Ist *Weiß am Zug*, ergibt sich folgendes Suchverfahren:

Auch wenn der König im Schach steht, wird die Bewertungsfunktion aufgerufen. Wir können aber nicht davon ausgehen, daß deren Wert auch erreicht wird.

Zunächst werden wie in der Ruhesuche die Schlagzüge untersucht.
Anschließend müssen wir in folgenden Fällen die ruhigen Züge nicht mehr (vollständig) untersuchen:

1) Ein Schlagzug hat einen besseren Wert als die Bewertungsfunktion ergeben. Aufgrund von Voraussetzung 2 müssen dann die ruhigen Züge nicht mehr untersucht werden.

2) Der Wert der Bewertungsfunktion liegt unterhalb von Alpha. Dann ist aufgrund von Annahme 2 nicht zu erwarten, daß die ruhigen Züge ins Alpha-Beta-Fenster führen. Die Stellung ist uninteressant, es muß kein Zug mehr untersucht werden.

3) Sonst werden die ruhigen Züge untersucht. Ergibt aber einer von ihnen einen besseren Wert als die Bewertung, können wir wieder abbrechen - erneut ist laut Annahme 2 nicht zu erwarten, daß dies sich wiederholt.

Diese Kriterien können die Suchzeit erheblich verkürzen. Schließlich sind die meisten Züge ruhige Züge, und die werden dann weggelassen. Die Cutoffs sind natürlich nur in vielen, nicht aber in allen Stellungen wirksam. Liegt z.B. ein Matt vor, so sind die Voraussetzungen nie erfüllt.

Die Suchroutine sieht nun so aus:

```
100 REM STELLUNG UNTERSUCHEN, WEISS AM ZUG
110 REM STELLUNG IST INTERN DARGESTELLT
120 REM T% IST TIEFE DER AKTUELLEN STELLUNG
130 REM MT% IST DIE SUCHTIEFE
140 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
150 REM AL% UND BE% SIND DIE WERTE FÜR ALPHA UND BETA
151 GOSUB "ZUGGENERATOR"
155 GOSUB "ILLEGALE STELLUNG?": IF A1% THEN MW%=16001-T%: GOTO 400
156 GOSUB "LIEGT EIN SCHACHGEBOT VOR?"
```

```
  160 VS% = (T% < MT%)
* 161 IF VS% THEN MW%=-16000+T%: GOTO 180
* 162 GOSUB "BEWERTUNG": MW%=PW%: IF SH% THEN MW%=-16000+T%
  163 IF MW% >= BE% THEN GOTO 400: REM CUTOFF
  164 IF MW% > AL% THEN AL%=MW%
  180 REM SUCHSCHLEIFE
  190 IF "KEINE SCHLAGZUEGE MEHR DA" AND NOT VS% THEN GOTO 310
  191 IF "KEINE ZUEGE MEHR DA" THEN GOTO 310
  200 "WAEHLE ZUG AUS"
  205 IF VS% OR PW%+"MAXIMALER ZUGEWINN" > AL% THEN GOTO 210
  206 AW%=PW%+"MAXIMALER ZUGEWINN"
  207 GOTO 260: REM FORWARD-PRUNING, CUTOFF
  210 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
  220 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
  230 GOSUB 500: REM "UNTERSUCHE STELLUNG FUER SCHWARZ"
  240 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
  250 "ERZEUGE MIT STACK ALTE STELLUNG"
  260 IF AW% <= MW% THEN GOTO 300
  270 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
  280 IF MW% >= BE% THEN GOTO 310: REM CUTOFF
  290 IF MW% > AL% THEN AL%=MW%
  300 GOTO 190
* 310 REM SUCHSCHLEIFE FERTIG ?
* 320 IF VS% OR NOT SH% THEN GOTO 360
* 330 IF MW% >= PW% THEN GOTO 360
* 340 IF PW% <= AL% THEN MW%=PW%: GOTO 360
* 350 IF PW% < BE% THEN BE%=PW%
* 351 VS%=-1: GOTO 190
  360 REM AUF PATT PRUEFEN
  370 IF MW%=-16000+T% THEN IF NOT SH% THEN MW%=0
  400 RETURN: REM ENDE DER STELLUNGSANALYSE FUER WEISS
```

Kommentare:

In Zeile 162 wird berücksichtigt, daß Weiß den Bewertungswert nicht sicher hat, wenn ein Schach vorliegt (sh%=-1). Dann wird mw% wie in der vollen Suche initialisiert.

Die Zeilen 320 bis 351 sind die wesentliche Neuerung. Falls in der Ruhesuche ein Schachgebot vorliegt (Zeile 320), muß die Stellung eventuell genauer untersucht werden.

Die Zeilen 330 und 340 überprüfen die Cutoff-Kriterien 1 und 2. Im Fall des zweiten Cutoffs wird der Stellungswert nach oben abgeschätzt (mw%=pw%).

Sind keine Cutoffs möglich, wird ggf. Beta erniedrigt (Zeile 350) - damit ist Kriterium 3 implementiert. Die volle Suche wird aufgenommen (Zeile 351).

Für Schwarz ist die Suchroutine analog:

```
  500 REM STELLUNG UNTERSUCHEN, SCHWARZ AM ZUG
  510 REM STELLUNG IST INTERN DARGESTELLT
  520 REM T% IST TIEFE DER AKTUELLEN STELLUNG
  530 MT% IST DIE SUCHTIEFE
  540 REM MW% IST DER ZU BERECHNENDE WERT DER STELLUNG
  550 REM AL% UND BE% SIND DIE WERTE FÜR ALPHA UND BETA
* 551 GOSUB "ZUGGENERATOR"
* 555 GOSUB "ILLEGALE STELLUNG?": IF A1% THEN MW%=-16001+T%: GOTO 800
* 556 GOSUB "LIEGT EIN SCHACHGEBOT VOR ?"
  560 VS% = (T% < MT%)
* 561 IF VS% THEN MW%=16000-T%: GOTO 580
* 562 GOSUB "BEWERTUNG": MW%=PW%: IF SH% THEN MW%=16000-T%
  563 IF MW% <= AL% THEN GOTO 800: REM CUTOFF
  564 IF MW% < BE% THEN BE%=MW%
  580 REM SUCHSCHLEIFE
  590 IF "KEINE SCHLAGZUEGE MEHR DA" AND NOT VS% THEN GOTO 710
  591 IF "KEINE ZUEGE MEHR DA" THEN GOTO 710
  600 "WAEHLE ZUG AUS"
  605 IF VS% OR PW%+"MAXIMALER ZUGEWINN" < BE% THEN GOTO 610
  606 AW%=PW%+"MAXIMALER ZUGEWINN"
  607 GOTO 660: REM FORWARD-PRUNING, CUTOFF
  610 "RETTE STELLUNG UND REKURSIVE VARIABLEN IM STACK"
  620 "FUEHRE ZUG AUS": REM INTERNE STELLUNGADARSTELLUNG AENDERN
  630 GOSUB 100: REM "UNTERSUCHE STELLUNG FUER WEISS"
```

```
  640 AW%=MW%: REM MW% IST DER ERMITTELTE WERT DES ZUGES
  650 "ERZEUGE MIT STACK ALTE STELLUNG"
  660 IF AW% >= MW% THEN GOTO 700
  670 MW%=AW%: REM DER ZUG HAT DEN STELLUNGSWERT VERBESSERT
  680 IF MW% <= AL% THEN GOTO 710: REM CUTOFF
  690 IF MW% < BE% THEN BE%=MW%
  700 GOTO 590
* 710 REM SUCHSCHLEIFE FERTIG ?
* 720 IF VS% OR NOT SH% THEN GOTO 760
* 730 IF MW% <= PW% THEN GOTO 760
* 740 IF PW% >= BE% THEN MW%=PW%: GOTO 760
* 750 IF PW% > AL% THEN AL%=PW%
* 751 VS%=-1: GOTO 590
* 760 REM AUF PATT PRUEFEN
* 770 IF MW%=16000-T% THEN IF NOT SH% THEN MW%=0
  800 RETURN: REM ENDE DER STELLUNGSANALYSE FUER SCHWARZ
```

Damit haben wir den normalen Brute-Force-Suchalgorithmus kennengelernt. Im nächsten Kapitel betrachten wir eine kleine Verbesserung, und anschließend beschäftigen wir uns damit, wie man Züge sortieren kann.

3.4.6 Variable Suchtiefe

Inzwischen haben wir den ersten Suchansatz in vielerlei Hinsicht verbessert. Wir führten den Alpha-Beta-Algorithmus, die Ruhesuche für Schlagzüge, die Ruhesuche bei Schachgeboten und 4 Arten des forward-pruning ein. Doch an einem haben wir kritiklos festgehalten: Die erschöpfende Suche endet bei einer festen Tiefenschicht. Was aber hindert uns daran, verschieden interessante Varianten verschieden tief zu analysieren?

Um die *variable Suchtiefe* für die erschöpfende Suche einzuführen, haben wir 2 Möglichkeiten:

1) Wir können weniger interessante Varianten weniger tief als normal untersuchen.

2) Wir können besonders interessante Varianten tiefer als normal untersuchen.

Die erste Methode verringert den Suchaufwand, beinhaltet aber die Gefahr, daß das Programm etwas übersieht - daher kommt sie für uns nicht in Frage (vgl. 3.4.3.d). Der zweite Ansatz ist in dieser Hinsicht unproblematisch, wird aber den Suchaufwand erhöhen. Betrachten wir ihn genauer.

Zunächst müssen wir festlegen, was eine besonders interessante Variante ist. Soweit ich weiß, gibt es hierfür noch gar keine erfolgreichen Verfahren. Versucht wurde dagegen, sehr gute Züge zu ermitteln und dies bei der Suche zu berücksichtigen.

Die Methode ist *(variable Suchtiefe)*:

> Wird ein herausragender Zug analysiert, also probeweise ausgeführt, so wird die Suchtiefe um 1 erhöht, die Folgestellung also einen Halbzug tiefer als sonst untersucht. Ist der Zug bearbeitet, muß selbstverständlich die Suchtiefe den alten Wert zurückerhalten.

Sicherlich dürfen nur wenige Züge das Prädikat "herausragend" erhalten. Bekäme es jeder Zug, würde die Suche nie enden - Suchtiefe und Stellungstiefe würden parallel erhöht. Doch diese Gefahr besteht schon, wenn wir nur 2,5% der Züge hervorheben!

Das bedeutet: nur ein minimaler Bruchteil der Züge darf als hervorragend gelten. Wir wollen trotzdem einen Ansatz in dieser Richtung wagen. Er basiert auf einem Vorschlag von Prof. Berliner anläßlich der Konferenz "Advances in Computer Chess IV" (1984). War der Vorgängerzug ein Schlagzug, so gelten in der jetzigen Stellung alle Züge als besonders interessant, die die eben gezogene Figur schlagen. Mit anderen Worten: wir zeichnen Schlagzüge, die zurückschlagen, als besonders interessant aus - die Folgestellung wird einen Halbzug tiefer als sonst untersucht. Dies ist nur deshalb durchführbar, weil solche Züge sehr selten

sind - in den meisten Stellungen gibt es sie gar nicht, weil der Vorgängerzug kein Schlazug war.

a) Implementierung

An der Suchroutine selbst müssen wir nichts ändern. Um die besonders interessanten Züge zu ermitteln, müssen wir in jeder Stellung wissen, ob der Vorgängerzug ein Schlagzug war und, wenn ja, wo die Figur nun steht. Hierzu benötigen wir eine zusätzliche Variable (im Programm: zv%(5)), die normalerweise den Wert 0 hat (Vorgängerzug war kein Schlagzug), sonst aber das Zielfeld des letzten Zuges angibt.

Beim Vertiefen eines Zuges prüfen wir mit dieser Variablen, ob er zurückschlägt. Wenn ja, wird die Suchtiefe (mt%) um 1 erhöht.

Wird die Variante später wieder verlassen, müssen wir mt% natürlich zurücksetzen. Daher wird mt% als rekursive Variable in den Stack aufgenommen.

Damit können die erforderlichen Änderungen mit zwei zusätzlichen Zeilen (in der Routine, die Stellungen vertieft) eingebaut werden - in der einen wird zv%(5) gesetzt, in der anderen geprüft, ob der Zug zurückschlägt.

b) Der Aufwand

Der zusätzliche Aufwand ist meist gering, da zurückschlagende Schlagzüge sehr selten sind. In manchen Stellungen kann es aber doch zu einer Vervielfachung der Suchzeit kommen. Daher ist diese Methode auch strittig.

Das eigentliche Ziel, die Varianten ihrer Bedeutung entsprechend verschieden tief zu untersuchen, können wir mit diesem Versuch sowieso nicht erreichen - dazu ist der Ansatz viel zu einfach. Dies ist aber auch das große ungelöste Problem der Schachprogrammierung, und sollte es einmal geklärt werden, dürfte die Spielstärke der Programme sprunghaft steigen.

3.4.7 Die iterative Suche - Drei auf einen Streich

Einige kleine Probleme haben wir bisher noch nicht betrachtet: die Bestimmung der Suchtiefe, die Zeitkontrolle, die Übergabe des besten Zuges an die Partieschleife. Dies alles löst die iterative Suche nebenbei.

a) Die Idee

Der bisherige Suchalgorithmus sieht vor, daß wir in einem Lauf die Stellung bis zu einer vorgegebenen Tiefe (+ Ruhesuche + variables Vertiefen) analysieren. Wir fangen also mit einer sehr umfangreichen Suche an. Sinnvoller wäre es, sich zunächst einen Überblick zu verschaffen und dann Schritt für Schritt weiterzumachen. So könnten wir bei der Voruntersuchung ein Bild vom Stellungswert und der besten Variante gewinnen, bevor wir intensiv analysieren.

Dies leistet die iterative Suche. Das Prinzip ist: Zunächst wird eine Suche der Tiefe 0, also eine Ruhesuche, durchgeführt, dann eine Suche der Tiefe 1, Tiefe 2, usw., solange wir Zeit haben. Jeder dieser Suchläufe wird Iteration genannt.

Bei jeder Iteration erhalten wir eine Hauptvariante (einen besten Zug, eine beste Zugfolge) und einen Wert für die Stellung (den Minimax-Wert). Beides können wir für die nächste Iteration verwerten.

Zum Einen sorgen wir dafür, daß die *Hauptvariante* beim folgenden Suchlauf *zuerst betrachtet wird*. Da sie oft die beste oder zumindest eine gute Variante bleibt, gewinnen wir viele Alpha-Beta-Cutoffs. Wir unterstützen also die Zugsortierung.

Auch mit dem Stellungswert (w) können wir etwas anfangen. In den meisten Fällen wird auch bei um eins erhöhter Suchtiefe in etwa der gleiche Wert herauskommen. Das nutzen wir, indem wir vor der nächsten Iteration Alpha und Beta nicht auf -16000/+16000 setzen, sondern auf die Werte w-d/w+d. Dieses Vorbesetzen von Alpha und Beta kann zu neuen Cutoffs führen

- je kleiner wir d wählen, desto eher. Ergibt sich allerdings bei der folgenden Suche ein Minimax-Wert, der außerhalb des Bereichs (w-d bis w+d) liegt, so müssen wir die Suche mit den normalen Alpha-Beta-Werten wiederholen - der Alpha-Beta-Algorithmus funktioniert dann nicht. Dies kostet dann doppelte Zeit. Daher sollte d so groß gewählt werden, daß dieser Fall selten eintritt. Diese Methode wird *Fenster-Technik* genannt.

Folgende Routine kann die iterative Suche steuern:

```
1000 REM STEUERUNG DER ITERATIVEN SUCHE
1005 REM INITIALISIEREN DER SUCHPARAMETER
1010 MT% = 0: REM DIE SUCHTIEFE
1015 AL% = -16000: BE% = 16000: REM ALPHA UND BETA
1020 REM DIE SCHLEIFE UEBER DIE ITERATIONEN
1025 T%=0: LT%=0: REM TIEFE DER PATIESTELLUNG
1030 SI%(1)=AL%: SI%(2)=BE%: REM WERTE SPEICHERN
1035 REM DER JEWEILIGE SUCHAUFRUF
1040 IF "WEISS AM ZUG" THEN GOSUB 100: GOTO 1050
1045 GOSUB 500
1050 REM SUCHE ERFOLGREICH? LIEGT DER MINIMAX-WERT IM ALPHA-BETA-
FENSTER?
1055 IF MW% > SI%(1) AND SI%(2) < BE% THEN GOTO 1070
1060 REM SUCHE WIEDERHOLEN
1065 AL%=-16000: BE%=16000: GOTO 1020
1070 IF "ZEIT ERSCHOEPFT" AND "ZUG GEFUNDEN" THEN GOTO 1095
1075 REM TIEFERE SUCHE VORBEREITEN
1080 AL%=MW%-100: BE%=MW%+100
1085 MT%=MT%+1
1090 GOTO 1020: REM NAECHSTE ITERATION
1095 RETURN
```

Kommentare:

Diese Routine ruft die verschiedenen Suchläufe auf. In den Zeilen 1010 und 1015 werden die Suchparameter vorbesetzt. Wir wollen mit einer Ruhesuche anfangen (mt%=0) und wissen nichts über den Stellungswert (Zeile 1015).

Danach beginnt die Schleife, in der wir die tiefer werdenden Suchläufe aufrufen. Die Anfangswerte von Alpha und Beta werden gespeichert, da sie beim Suchen verändert werden (Zeile 1030).
Dann wird ein Suchlauf aktiviert, je nach Farbe wird die entsprechende Routine angesprungen(Zeilen 1035 bis 1045)
Nun müssen wir kontrollieren, ob die Annahme, daß der Minimax-Wert auch bei dieser Tiefe zwischen den Startwerten von Alpha und Beta liegt, stimmt (Zeile 1055). Ist dies nicht der Fall, so müssen wir die Suche wiederholen - mit ungüstigeren Startwerten (Zeilen 1060/1065).
Zeile 1070 prüft, ob zuviel Zeit verbraucht wurde - dann soll das Programm mit dem Suchen aufhören und seine Zugwahl mitteilen.
Kann das Programm weitersuchen, wird die Tiefe erhöht, und Alpha und Beta werden auf Werte um den aktuellen Minimax-Wert herum gesetzt (Zeilen 1080 bis 1090).

Zusätzlich zu den bisherigen Überlegungen wurde hier auch eine Zeitsteuerung angedeutet. Wir kommen später darauf zurück.

b) Die Fenster-Technik

Nehmen wir die Fenster-Technik genauer unter die Lupe. Was wissen wir, wenn eine Suche nicht geklappt hat, wenn der Ergebnis-Wert nicht im Alpha-Beta-Fenster liegt?

Betrachten wir folgende Situation:

Das Programm spielt mit Weiß und untersucht die Partiestellung. Die Anfangswerte von Alpha und Beta seien A und B gewesen. Drei Fälle können nach der aktuellen Suchiteration auftreten (je nach Ergebnis-Wert mw%):

1) mw%>=B: Dann ist der beste Zug, den das Programm fand, mindestens so gut wie mw%, andere Züge können jedoch besser sein.

2) mw%<=A: Alle Züge sind schlechter als mw%, welcher Zug der relativ beste ist, steht nicht fest.
3) Sonst: Das Programm hat den besten Zug bei dieser Tiefe gefunden.

b1) Mattwerte

Wenn im ersten Fall mw% größer als 15000 ist, hat die Suche einen Weg zum Matt gefunden (so ist die Bewertungsfunktion aufgebaut, vgl. 3.4.1). Dann ist es unnötig, nochmals zu suchen, denn mehr als mattsetzen kann das Programm nicht.

Ist hingegen im zweiten Fall mw% kleiner als -15000, so wird das Programm zwangsläufig mattgesetzt und es ist egal, was es spielt.

In beiden Fällen muß die Suche nicht wiederholt werden, da sie extreme Werte lieferte. Dies können wir ausnutzen, wenn wir in den Zeilen 1015 und 1065 al%=-15000 und be%=15000 setzen und

```
1052 IF MW% < -15000 OR MW% > 15000 THEN GOTO 1095
```

ergänzen. Dann endet die Suche, sobald sie ein Matt gefunden hat - Wiederholungen sind unnötig.

b2) Leider nicht ohne Probleme

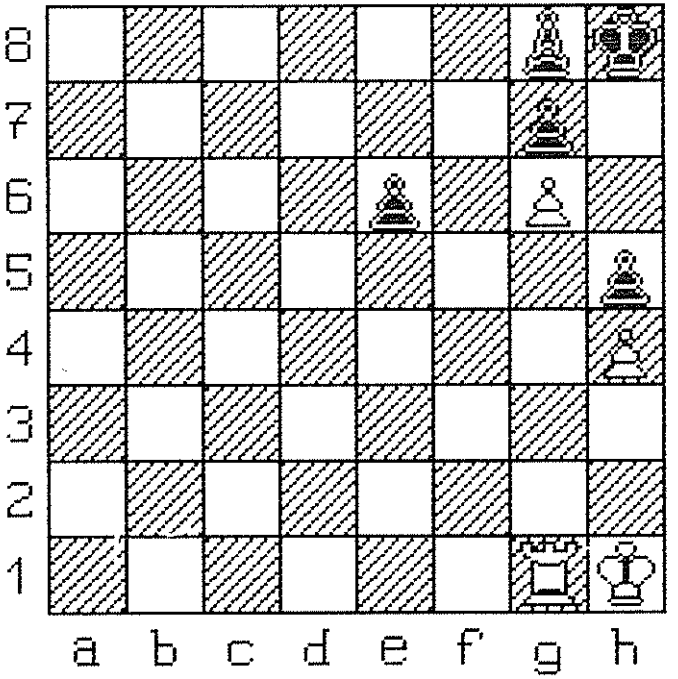
Eine zweite Möglichkeit nutzt unser Programm nicht aus. Wenn eine Suche wiederholt werden muß, können Alpha und Beta so neu gesetzt werden *(enges Wiederholungsfenster)*:

Fall 1: Alpha=mw%-1: Beta=15000
Fall 2: Alpha=-15000: Beta=mw%+1

Nach den obigen Feststellungen über den Alpha-Beta-Algorithmus muß bei der Wiederholung (gleiche Suchtiefe!) der Ergebniswert im neuen Fenster liegen.

Dies wäre richtig, wenn wir nur den Alpha-Beta-Algorithmus benutzen würden. Auch bei uns klappt es in 99% der Fälle, aber im hundersten macht uns das forward-pruning einen Strich durch die Rechung. In Kapitel 3.4.4.d wurde dies schon angedeutet.

Nehmen wir also an, wir setzen die Alpha- und Beta-Werte so (enges Wiederholungsfenster). Was geschieht in folgender Stellung? (Weiß am Zug)



Zur Vereinfachung gehen wir davon aus, daß die Bewertungsfunktion nur den Materialterm berechnet. Bei Suchtiefe 0 und 1 kommt das Programm auf den Ergebniswert 75, was dem Materialverhältnis in der Partiestellung entspricht.

Bei Tiefe 2 tut sich dann etwas. Die ursprünglichen Alpha- und Beta-Werte sind -25 und 175. Nach Tg5 werden alle schwarzen Gegenzüge druchprobiert, und in der Ruhesuche kann das

Programm immer Th5: spielen bzw. nach Lf7 auch gf:. In jedem Fall gewinnt es den Läufer, der Minimax-Wert mw% liegt also bei 400.

Die Suche wird wiederholt, nach dem neuen Verfahren erhalten Alpha/Beta die Werte 399/15000. Wieder wird Tg5 untersucht, und daraufhin auch e5 als Gegenzug. Nun wird aber Th5: ausgeschlossen - der Stellungswert vor diesem Zug ist 75, die Suche erwartet nur einen Bauerngewinn (macht 100 Punkte) und dann noch eventuelle positionelle Verbesserungen (wieder 100 Punkte). Damit kommt sie nicht ins jetzige Alpha-Beta-Fenster - der Zug wird durch forward-pruning (dummerweise) weggeschnitten. Daß der Bauer h5 mit Schach fällt, sieht das Programm nicht im Voraus. Als Folge ergibt sich bei dieser Iteration ein End-Ergebniswert von nur 275. Wir landen erneut nicht im Alpha-Beta-Fenster.

Nun müßte die Suche zum dritten Mal wiederaufgenommen werden. Wegen solcher - wenn auch seltener - Fälle wurde dieses Verfahren nicht im Programm aufgenommen.

c) Die Hauptvariante

Oben wurde gesagt, wir erhielten bei jeder Suchiteration eine Hauptvariante, eine Folge der besten Züge. Doch wie wird sie berechnet?

Dies ist eine relativ komplizierte Angelegenheit. Der beste Zug einer Stellung ist der Zug, der zum besten Wert führt, der also die Variable mw% auf ihren Endwert bringt. Dies gilt allerdings nur, wenn Alpha überschritten (Weiß am Zug) oder Beta unterschritten (Schwarz am Zug) wurde - sonst sagt die Suche nur aus, daß es keinen guten Zug in dieser Stellung gibt.

Welcher Zug der beste in einer Stellung ist, wissen wir erst am Ende der Stellungsanalyse. Davor erhalten wir nur "aktuelle beste Züge". Das sind diejenigen, die vorläufig als beste gelten, die in der Suchroutine mw% auf den momentanen Wert brachten. Zusätzlich muß ,wie schon gesagt, mw% größer als Alpha (Weiß am Zug) bzw. kleiner als Beta (Schwarz am Zug) sein.

Um die Hauptvariante zu ermitteln, müssen wir aber noch mehr überlegen. Nicht jeder aktuelle beste Zug bzw. beste Zug einer Stellung gehört auch zur Hauptvariante. Dies ist nur dann der Fall, wenn auch die Stellung zur Hauptvariante gehört, wenn sie also von der Partiestellung aus ausschließlich über beste Züge erreicht wird. Ob dies aber der Fall ist, wissen wir erst nach der Suche.

Das ist hier das große Problem: wir wissen alles erst hinterher. Aber lösbar ist es doch.

Mit folgendem Verfahren können wir die ersten 4 Züge der Hauptvariante bestimmen:

Wir definieren uns ein Array dim hv%(3,3,2) ("hv" für Hauptvariante). Die ersten beiden Indices benötigen wir, um die einzelnen Züge zu speichern, der letzte dient der Darstellung von Ursprungs- und Zielfeld eines Zuges. hv%(i,j,1) und hv%(i,j,2) stellen also zusammen einen Zug dar. i und j laufen von 0 bis 3 (i=0, 1, 2, 3; j=0, 1, 2, 3).

Haben wir an einer Stelle (i,j) keinen Zug dargestellt, so wollen wir hv%(i,j,1) hier den Wert 0 geben.

Von hv% benutzen wir nur die Komponenten (i,j,k), bei denen j>=i ist.

Hauptvariante von Stellung der Tiefe 0 (Partiestellung)
Hauptvariante von Stellung der Tiefe 1
Hauptvariante von Stellung der Tiefe 2
Hauptvariante von Stellung der Tiefe 3
3
2
1
0
0
1
2
3
j
i

Bei der Suche gehen wir nun so vor *(Hauptvarianten-Verfahren)*:

1) Erreichen wir eine Stellung der Tiefe t (t<3) so setzen wir hv%(t+1,t+1,1) auf 0. Wir löschen damit beste Züge aus zuvor untersuchten Stellungen dieser Tiefe.

2a) Erhalten wir in einer Stellung der Tiefe t (t<4) einen aktuellen besten Zug, also einen, der mw% verbessert, so tragen wir ihn in hv%(t,t,1) und hv%(t,t,2) ein. (Abkürzung: in hv%(t,t,.)). Er ist der Anfang der momentanen Hauptvariante von dieser Stellung aus.

2b) Weiterhin schauen wir nach, ob er das Ende der Variante ist. Wenn nicht, so enthält hv%(t+1,t+1,.) den richtigen Folgezug. Diesen kopieren wir - falls vorhanden - nach hv%(t,t+1,.). Ist auch dieser noch nicht das Ende der Variante, so finden wir den nächsten richtigen Zug in hv%(t+1,t+2,.) - dieser wird also auch kopiert, und zwar nach hv%(t,t+2,.). Dies machen wir solange, bis der zweite Index t+j den Wert 4 erreicht hat.

Invariante:

So stellen wir sicher, daß nach der Untersuchung einer Stellung der Tiefe t die von dieser Stellung ausgehende Hauptvariante in hv%(t,t,.), hv%(t,t+1,.), hv%(t,t+2,.) usw. bis hv%(t,4,.) steht.

Am Ende der Suche, also wenn die Partiestellung (Tiefe 0) fertig untersucht ist, finden wir folglich die von ihr ausgehende Hauptvariante in hv%(0,0,.), hv%(0,1,.), hv%(0,2,.) ,hv%(0,3,.).

Bemerkung: Dies ist ein kompliziertes, weil rekursives Verfahren. Am leichtesten wird es klar, wenn man zunächst überlegt, daß die Invariante für t=3 gilt. Dies folgt direkt aus Aktion 2a. (Aktion 2b kann bei t=3 nicht gemacht werden). Anschließend muß man sich

schrittweise überzeugen, daß dies dann auch für t=2, t=1 und t=0 gilt.
Dies folgt jeweils aus den drei Aktionen zusammen - steht in hv%(3,3,.) ein Zug (z3), so ist er der beste Zug der zuletzt untersuchten Stellung der Tiefe 3 (Züge aus vorherigen Untersuchungen wurden durch Schritt 1 gelöscht), und diese wurde als Folgestellung des aktuellen Zugs der Tiefe 2 analysiert. Ist nun der aktuelle Zug (z2) der beste in der aktuellen Stellung der Tiefe 2, so ist in der Folgestellung auch z3 der beste, also ist z2, z3 die Hauptvariante.
Analoge Überlegungen muß man dann für t=1 und t=0 anstellen.
Prinzipiell kann man dies auch durch Induktion zeigen.

Mit dem Hauptvarianten-Verfahren können wir also die ersten vier Züge der Hauptvariante bestimmen. Es funktioniert natürlich auch für längere Zugfolgen, wir müßten dann hv% größer definieren. Der Speicherbedarf ist relativ gering - für Varianten der Länge n benötigen wir ein Array, das n^2 Züge aufnehmen kann.

Die Implementierung kostet wenig Schreibaufwand. Den aktuellen besten Zug können wir in den Suchroutinen erfragen.

Bsp.: In die Suchroutine für Weiß (Kap. 3.4.5) nehmen wir folgende Zeile auf:

```
275 IF MW% > AL% THEN GOSUB "AKTUELLEN BESTEN ZUG BEARBEITEN"
```

Weiterhin können wir eine kurze Routine "aktuellen besten Zug bearbeiten" schreiben, die dann die Schritte 2a und 2b ausführt. (Im Programm: Zeilen 5400-5440).

Schritt 1 kann in der Routine, die Züge vertieft, also die interne Stellungsdarstellung ändert, in einer Zeile erledigt werden.

d) Zeitsteuerung und Zugwahl

In Abschnitt a wurde schon eine Zeitsteuerung angedeutet. Offensichtlich können wir die Suche frühestens dann abbrechen, wenn ein bester Zug gefunden wurde. Dies ist genau dann der Fall, wenn unsere Hauptvariante nicht leer ist, also wenigstens einen Zug enthält.

Sonst müssen wir überlegen, ob noch tiefer gesucht werden soll. Hierfür sind bisher kaum sinnvolle und sichere Verfahren veröffentlicht worden. Wir machen es uns ganz einfach: wenn eine bestimmte Zeitschranke überschritten wurde, beenden wir die Suche. Noch einfacher: wir brechen ab, wenn die Zahl der untersuchten Stellungen einen bestimmten Wert überschritten hat.

Diesen Wert erfragt das Programm bei Partiebeginn von seinem Gegner:

```
INPUT "Spielstufe";ZE%
```

In unserer Routine für die iterative Suche bauen wir dies mit

```
1070 IF "ZAHL UNTERSUCHTER STELLUNGEN" > ZE% AND HV%(0,0,0) <> 0 THEN
GOTO 1095
```

ein.

Nach einem solchen Abbruch steht der beste Zug in hv%(0,0,.). Hier kann ihn die Partieschleife ablesen.

e) Wirkung

Gegen die iterative Suche mag man einwenden, daß Arbeit doppelt gemacht wird - die Zugwahl des Programms hängt nur von der letzten Iteration ab, also sind die anderen überflüssig. Dies ist falsch. Erstens ist der letzte Suchlauf durchschnittlich 6 mal umfangreicher als alle vorherigen zusammen (die fallen also kaum ins Gewicht), und zweitens verhelfen die Informationen aus den vorherigen Iterationen zu so Vielen neuen Cutoffs, daß

der zusätzliche Zeitverbrauch um ein Vielfaches wettgemacht wird. Die Iterative Suche ist eines der wichtigsten Verfahren der Zugsortierung.

Daß die Zeitkontrolle und die Zugübergabe an die Partieschleife hier mitgelöst werden, ist nur ein angenehmer Nebeneffekt.

3.4.8 Zugselektion und Zugsortierung

a) Zugsortierung und Killerheuristik

Die vorherigen Kapitel haben gezeigt, daß der Alpha-Beta-Algorithmus nur dann seine ganze Kraft entfaltet, wenn wir in jeder Stellung den besten Zug zuerst analysieren. Dies ist natürlich nicht erreichbar - wüßten wir im voraus, welcher Zug das ist, müßten wir ja gar nicht mehr suchen. Immerhin können wir die Züge einigermaßen sinnvoll sortieren.

a1) Die Hauptvariante

Ein Sortierverfahren wurde bereits bei der iterativen Suche besprochen: Verfügen wir über eine Hauptvariante aus der letzten Iteration, so sollte diese zuerst analysiert werden. Anders ausgedrückt: sind wir noch innerhalb der Hauptvariante des vorigen Suchlaufs, sollten wir deren nächsten Zug zuerst untersuchen. Die Information, daß dieser Zug gut ist, beruht schließlich auf einer umfangreichen Suche.

Wie die Hauptvariante berechnet wird, wissen wir bereits. Vor einer neuen Iteration stehen deren Züge in hv%(0,0,...), hv%(0,1,...) bis hv%(0,3,...). Zu Klären ist nur noch, wie die Suche erkennt, ob sie noch in der Hauptvariante ist.

Hierzu führen wir die logische Variable gz% (für "gemerkter Zug") ein. Diese wird vor der Iteration auf "wahr" gesetzt (gz%=-1). Beim Vertiefen eines jeden Zuges wird geprüft, ob er der Hauptvariantenzug für diese Stellungstiefe ist und ob gz% noch auf "wahr" steht. Ist eine Bedingung nicht erfüllt, so hat

die Suche die Hauptvariante verlassen und gz% wird auf "falsch" gesetzt (gz%=0).

Da die Hauptvariante zuerst untersucht wird, funktioniert dies sogar, wenn gz% nicht als rekursive Variable behandelt wird.

Steht gz% auf "wahr", so befindet sich die Suche noch in der Hauptvariante und hv%(0,t%,...) enthält den entsprechenden Zug (t% ist die Stellungstiefe).

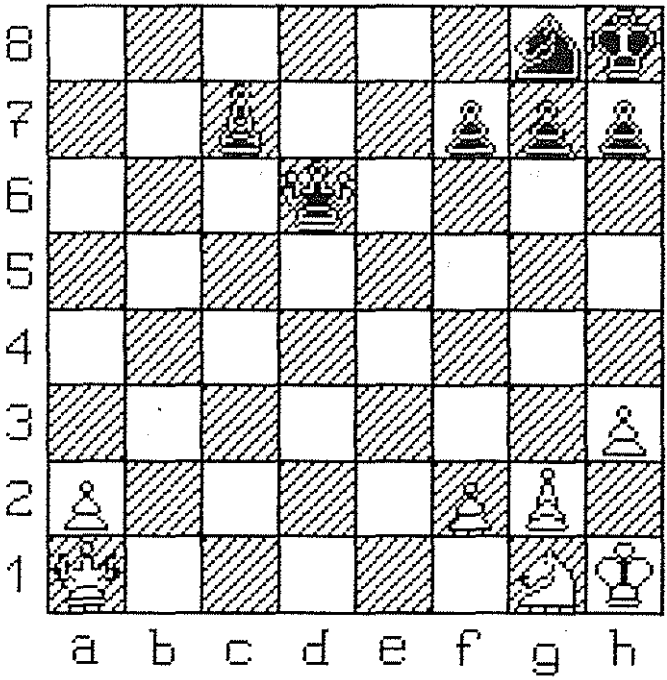
a2) Schlagzüge

In den meisten Stellungen hilft uns die Hauptvariante nicht weiter. Wie sollen wir dort die Züge sortieren? Da sind zunächst die Schlagzüge. Gerade bei Brute-Force-Programmen tauchen häufig Stellungen (in der Suche) auf, in denen Figuren ungedeckt sind - und dann sind Schlagzüge stark. Noch höhere Priorität sollte den Schlagzügen eingeräumt werden, die die zuletzt gezogene Figur schlagen - diese ist mit hoher Wahrscheinlichkeit ungedeckt.

Wir müssen also zwei Klassen von Schlagzügen unterscheiden: normale Schlagzüge und solche, die die zuletzt gezogene Figur beseitigen.

a3) Die Killerheuristik

Eine weitere, sehr wirksame Methode zum Sortieren der Züge ist die Killerheuristik. Betrachten wir folgende Stellung:



Weiß ist am Zug und muß etwas gegen Dh2++ unternehmen. Der Suchalgorithmus ist nicht so schlau, er probiert nacheinander alle Züge durch. Nehmen wir an, er fängt mit a3 an, untersucht die

schwarzen Gegenzüge der Reihe nach und stellt schließlich fest, daß Dh2 daraufhin mattsetzt. Nun testet er den nächsten Zug, sagen wir a4. Schön wäre es, wenn wenigstens jetzt Dh2 als erster Gegenzug geprüft wird - dann ist a4 sofort widerlegt. Genau dies leistet die Killerheuristik. Hat sich ein Zug in einer Stellung bewährt (wie hier Dh2+matt) und wird eine Nachbarstellung untersucht, wird der bewährte Zug vorgezogen.

Wir benötigen lediglich ein Array, das für jede Tiefenstufe den zuletzt besten Zug speichert:

- Nach der Analyse jeder Stellung wird der beste Zug dort eingetragen.

- Vor der Untersuchung einer Stellung wird der eingetragene beste Zug nach vorne sortiert.

Noch besser ist es, für jede Tiefenstufe zwei Züge zu speichern - dann geht ein bewährter Zug nicht verloren, nur weil er zufällig in einer Stellung schlecht war.

a4) Die Sortierung im Ganzen

Wir haben folgende Zugklassen betrachtet:

- Zug der Hauptvariante
- Schlagzüge, die die zuletzt gezogene Figur schlagen
- Killerzüge
- sonstige Schlagzüge
- sonstige (ruhige) Züge

In dieser Reihenfolge werden die Züge vom Programm untersucht. Das beste Sortierverfahren ist die Hauptvariante - deren Züge haben sich in einer Suche bewährt! Die anderen Kriterien sind mehr spekulativ. Schlagzüge, die die gezogene Figur schlagen, haben gegenüber Killerzügen zwei Vorteile: häufig sind sie genau das Richtige, denn die gezogene Figur ist ungedeckt, und außerdem verkürzen sie die Ruhesuche.

In der Ruhesuche benutzen wir die gleiche Reihenfolge. Die Killerzüge werden ausgelassen, wenn sie keine Schlagzüge sind, und die ruhigen Züge sowieso.

Für die Effizienz der Suche sind alle Verfahren sehr wichtig. Nimmt man ein beliebiges weg, wird sich der Suchbaum mindestens verdoppeln (im Durchschnitt natürlich). Räumt man z.B. Schlagzügen, die zurückschlagen, keine besonders hohe Priorität ein, wird die Ruhesuche in die Irre laufen.

Manche Programme ziehen auch Schachgebote (Züge, die Schach geben) vor (wie unseres die Schlagzüge). Bei unserem Programm scheitert dies daran, daß es gar nicht feststellen kann, ob ein Zug ein Schachgebot ist. (Würde man dies einbauen, so käme außerdem in Betracht, Schachgebote bis zu einer bestimmten Tiefe auch in die Ruhesuche aufzunehmen.)

Auch sonst bieten sich hier einige Möglichkeiten, kreativ zu werden und die Zugsortierung zu verbessern.

b) Sortieren oder Selektieren - wann und wo

Sortierverfahren kosten immer viel Zeit, auch die besten. Das kann sich ein Schachprogramm kaum leisten.

Wir sollten also möglichst darauf verzichten! (Auf das Sortieren, natürlich). Andererseits müssen wir die Reihenfolge, die wir eben festgelegt haben, auch einhalten - eine irregeleitete Suche ist noch aufwendiger!

Die Lösung ist einfach: wir sortieren die Züge nicht in eine neue Reihenfolge, sondern wir selektieren sie aus der ungeordneten Liste, die der Zuggenerator liefert.

Um es klarer auszudrücken: nach dem Aufruf des Zuggenerators wird nicht die Zugliste im ganzen sortiert. Erst in der Suchschleife, wenn ein Zug ausgewählt werden muß, wird die noch ungeordnete Zugliste betrachtet. Und auch jetzt wird nur der benötigte Zug gesucht. Dies erledigt eine Routine.

Sie merkt sich, in welcher Selektionsphase sie ist, in der Variablen zv%(4):

Wert	Selektionsphase
1	Hauptvariantenzug
2	zurückschlagende Schlagzüge
4	erster Killerzug
5	zweiter Killerzug
3	normale Schlagzüge
6	sonstige Züge

Ist sie beispielsweise in der Hauptvarianten-Phase, sieht sie nach, ob die Hauptvariante überhaupt einen Zug vorschlägt und wenn ja, ob dieser auch in der Zugliste ist. Ist auch dies der Fall, so streicht sie ihn dort und liefert ihn an die Suchschleife.

Der Vorteil bei diesem Vorgehen ist, daß nur die wirklich benötigten Züge selektiert werden. Führt der erste zum Cutoff, müssen die anderen nicht geordnet werden.

Aber auch dieses Verfahren kostet Zeit. Um die Arbeit der Selektions-Routine zu erleichtern, schreibt der Zuggenerator Schlagzüge und normale Züge in verschiedene Listen (was ohne Aufwand geht).

Die Selektion muß daher nur bei Hauptvarianten- und Killerzügen die ganze Zugliste durchmustern - und auch hier nur in der erschöpfenden Suche, denn in der Ruhesuche schaut sie nur nach entsprechenden Schlagzügen. In den Selektionsphasen 2 und 4 werden nur die realtiv wenigen Schlagzüge geprüft, und in Phase 6 werden die normalen Züge in der unsortierten Reihenfolge übernommen.

Die Züge werden also in drei Schritten behandelt:

- Der Zuggenerator schreibt sie in die Schlagzug- bzw. Normalzugliste.
- Die Selektionsroutine schlägt sie einzeln vor
- Die Suche benutzt sie

Wie Zuggenerator und Selektionroutine den Überblick über die Zugliste wahren, wird in der Programmdokumentation beschrieben.

3.5 Die Stellungsbewertung

3.5.1 Einleitung

Es sind im wesentlichen zwei Programmteile, durch die das spezielle Anwendungsproblem "Schachspiel" lösbar wird. Diese sind der Zuggenerator, der uns durch die Implementierung der Spielregeln alle Zugmöglichkeiten für eine vorgegebene Stellung liefert, sowie die Stellungsbewertung mit ihrem schachspezifischen Wissen. Nachdem wir inzwischen den Zuggenerator kennengelernt haben, wollen wir uns nun folgenden Fragen zuwenden:

1. Welche schachtheoretischen Erkenntnisse benötigt ein Rechner, um eine Stellung "bewerten" zu können?
2. Wie wird dieses Wissen im Rechner dargestellt?

Diese beiden Fragen werden in den nächsten Abschnitten im Mittelpunkt unserer Betrachtungen stehen. Dabei werden wir keineswegs nur graue (Schach-)Thoerie kennenlernen, sondern wir werden versuchen, anhand unterschiedlicher in Schachprogrammen angewandter Methoden und eigener Überlegunen einen Überblick über die Möglichkeiten zu gewinnen, eine Stellungsbewertung zu programmieren.

Doch bevor wir uns mit Einzelheiten auseinandersetzen, zuvor noch einige einleitende Bemerkungen: Wie wir bei der Beschreibung des Such- und Entscheidungsalgorithmus gesehen haben, muß das Ergebnis der Bewertung in einer Zahl zusammengefaßt werden, damit die Bewertungen unterschiedlicher Stellungen schnell miteinander verglichen werden können. Diese Zahl soll eine Aussage darüber treffen, zu welchem Spielausgang die zu bewertende Stellung tendiert, wenn man voraussetzt, daß beide Spieler die Partie mit den jeweils für sie besten Zügen fortset-

zen. Das kann man erreichen, wenn für Positionen, in denen Weiß Vorteil hat, positive Werte und bei für Schwarz vorteilhaften Stellungen negative Werte errechnet werden. Außerdem muß sich die "Größe des Vorteils", den Gewinnchancen des besser stehenden Spielers entsprechend, in dem Betrag der Bewertung widerspiegeln. Da hierbei weder Weiß noch Schwarz bevorzugt behandelt werden dürfen, muß ein jeweils gleichgroßer Vorteil auch zu betragsmäßig gleichen Werten und eine ausgeglichene Stellung zum Wert "0" führen. Dies nennt man ein *"symmetrisches Verhalten der Bewertungsfunktion"*.

In dieses Schema lassen sich auch die nach den Spielregeln entschiedenen Positionen einordnen: Stellungen, in denen Schwarz bzw. Weiß mattgesetzt wurde, bekommen die höchstmöglichen bzw. die niedrigstmöglichen Werte, die durch keine anderen Bewertungen erreicht werden dürfen. Unentschiedene Stellungen (patt, technisches Remis wie z.B. bei alleinstehenden Königen) werden durch den Null-Wert repräsentiert. Dazu ist allerdings zu sagen, daß die Bewertungsfunktion meistens nicht so programmiert wird, daß sie Matt- oder Pattstellungen erkennen kann. Es ist nämlich relativ aufwendig festzustellen, ob nach einem Schachgebot Züge vorhanden sind, die die Bedrohung des Königs beseitigen, ohne diese Züge (durch Verändern der Stellung) auszuprobieren. So ist es beispielsweise möglich, daß die einzige Figur, die den schachbietenden gegnerischen Stein schlagen könnte, durch eine Fesselung daran gehindert wird. Daher wird die korrekte Behandlung solcher Stellungen meist vom Suchverfahren miterledigt, da es im Rahmen seiner Untersuchengen mit nur wenigen zusätzlichen Aktionen solche Sachverhalte feststellen kann.

Welche Methoden müssen wir also benützen, damit sich die Stellungsbewertung in einem Wert ausdrücken läßt, der alle genannten Anforderungen erfüllt? Für die entschiedenen Stellungen ist die Sache klar: Hier handelt es sich um (durch die Spielregeln) festgelegte Fälle, die bei entspechender Programmierung exakt überprüft werden können. Wesentlich schwieriger ist es dagegen, bei noch offenem Spielausgang die beiderseitigen Chancen abzuschätzen. Wie wir noch sehen werden, ist im allgemeinen wirklich nicht mehr als eine Abschätzung für die noch

nicht entschiedenen Positionen möglich, da Schach ein äußerst vielfältiges Spiel ist, in dem selbst kleine Unterschiede in den Figurenaufstellungen große Bedeutung haben können. Dazu kommen noch unterschiedliche Anforderungen an die Strategie (den Plan für ein optimales Zusammenwirken der Figuren) in den aufeinanderfolgenden Partiestadien Eröffnung, Mittelspiel und Endspiel und bezüglich des "Charakters" der aktuellen Stellung. Hierauf werden wir noch näher zu sprechen kommen. Wir sehen also schon, daß wir uns bei der Konstruktion einer Bewertungsfunktion auf eine der heikelsten Unternehmungen einlassen, die die Schachprogrammierung zu bieten hat. Doch diese Arbeit ist gleichzeitig auch eine der interessantesten, bei der oft der Programmierer noch einiges über das Schachspiel lernen kann.

Wir werden versuchen, die Aufgabe dadurch zu lösen, daß wir in der vorgegebenen Stellung nach Indizien (im folgenden *Stellungsmerkmale* genannt) suchen, aus denen wir auf mögliche Vorteile für einen Spieler schließen können. Diese Merkmale lassen sich in zwei Klassen einteilen, die sich daraus ergeben, daß wir zum einen die auf dem Brett befindlichen Figuren unabhängig von ihren Standorten betrachten wollen *(materielle Bewertung)* und zum anderen das Zusammenwirken zwischen den Figuren gleicher Farbe aber auch die Wechselbeziehungen zu gegnerischen Steinen untersuchen werden *(positionelle Bewertung)*.

Eine Bewertung des Materials läßt sich recht einfach vornehmen, wenn man den Figurentypen König, Dame, Turm, Läufer, Springer und Bauer Grundwerte zuordnet. Wie man dabei vorgeht, werden wir demnächst sehen. Die Einschätzung positioneller Merkmale ist dagegen erheblich schwieriger. Es ist klar, daß wir in der Bewertungsfunktion nicht alle Aspekte berücksichtigen können, die in der Schachtheorie eine Rolle spielen. Dazu ist der Speicher in dem uns zur Verfügung stehenden Rechner (und nicht nur in diesem!) viel zu klein. Außerdem würde eine solche Bewertung für praktische Fälle, d.h. zum Spielen einer Partie, viel zu lange dauern.

Wir müssen uns in der Auswahl also beschränken. Eine Besprechung der wichtigeren und in den meisten heutigen Schachprogrammen berücksichtigten Stellungsmerkmale folgt im Anschluß an die Einführung der Materialbewertung. Einige Beispiele seien aber jetzt schon genannt: Schwachstellen in der Bauernaufstellung wie isolierte Bauern oder Doppelbauern, "starke" Bauern wie z.B. Freibauern, Bauernschutz des Königs vor gegnerischen Angriffen, Beherrschung der Brettmitte. Bei unseren Bewertungen werden wir Stellungsmerkmale, die unter schachtheoretischen Gesichtspunkten zusammengehören, unter einem Oberbegriff zusammenfassen (z.B. isolierte Bauern, Freibauern, usw. als Bewertung der Bauernstruktur).

Neben der Notwendigkeit, sich auf wichtige Stellungsmerkmale zu beschränken, haben wir das Problem, daß die Bedeutung, die den positionellen Bewertungen zukommt, von Stellung zu Stellung variieren kann, so daß eine bestimmte Schwäche in manchen Positionen spielentscheidend und in anderen ohne Belang sein kann. Dementsprechend lassen sich in diesem Bereich kaum allgemeingültige Einschätzungen programmieren, die für alle möglichen Positionen korrekte Bewertungen im schachtheoretischen Sinn liefern. Vielmehr können wir nur eine Reihe von "Faustregeln" benutzen, die in der Mehrzahl der Stellungen näherungsweise zutreffende Werte ermitteln. Ein solches Vorgehen wird in der Fachsprache der (Schach-) Programmierung als Anwendung *heuristischer Verfahren* bezeichnet.

Unter Berücksichtigung dieser Gesichtspunkte müssen wir uns entscheiden, wie stark die Einzelbewertungen der zu implementierenden Kriterien die Gesamtbewertung der Stellung beeinflussen und wie sie überhaupt miteinander verknüpft werden sollen. Ein möglicher Ansatz ist der, alle Kriterien oder Stellungsmerkmale einzeln und unabhängig voneinander zu bewerten und diese Werte dann aufzusummieren. Da dies aber der Komplexität der Bewertung nur recht ungenügend Rechnung trägt, und außerdem zwischen den zu bewertenden Kriterien Abhängigkeiten bestehen, werden zumeist sogenannte *Wichtungskoeffizienten* und/oder *Auswahloperatoren* eingeführt, mit deren Hilfe eine Manipulation der Einzelbewertungen vorgenommen

werden kann. Die Wichtungskoeffizienten sind im allgemeinen konstante Faktoren, mit denen der Wert für ein Kriterium multipliziert wird, und haben die Aufgabe, die einzelnen Werte zueinander in Beziehung zu setzen, d.h., sie entsprechend ihrer Bedeutung für das Einschätzen der Gewinnaussichten anzupassen. Durch die Benutzung von Auswahloperatoren können Kriterien innerhalb der Gesamtbewertung "ausgeschaltet" werden, wenn sie für die aktuelle Stellung "uninteressant" sind (z.B. keine Bewertung der Königssicherheit in Endspielpositionen).

Aus den bisher angesprochenen Schwierigkeiten resultiert eine "Philosophie", der sich die meisten Schachprogramme bedienen. Sie besagt, daß die positionellen Bewertungen durch die Wichtungskoeffizienten so stark in ihrer Bedeutung eingeschränkt werden, daß ihr Gesamtwert nur selten den Wert für einen Bauern übersteigt. (Allerdings werden wir auch Stellungsmerkmale kennenlernen, die da eine Ausnahme machen.) Diese hohe Einschätzung des Materials liegt in der Erfahrung begründet, daß ein materielles Übergewicht auch von Schachprogrammen meistens in einen Sieg umgesetzt werden kann, während dies für einen positionellen Vorteil zumindest als zweifelhaft angesehen werden muß. Auf diese Art soll das Programm daran gehindert werden, für eventuell nicht völlig korrekt bewertete positionelle Vorteile (zuviel) Material zu opfern.

Dies hat allerdings auch negative Konsequenzen für die Spielweise des Programmes: Langfristige Materialopfer, d.h., Züge, die Material mit der Absicht aufgeben, ein positionelles Übergewicht, z.B. Angriff gegen den gegnerischen König, zu erlangen, und nicht auf baldigen Materialgewinn abzielen, liegen außerhalb des Spielverständnisses der Programme, die sich dieser Einschätzung bedienen. Dies hat zur Folge, daß solche Programme erstens die Möglichkeiten eines langfristigen Opfers nicht nutzen und, schlimmer noch, die Gefahr eines drohenden Opfers nicht erkennen können. Insbesondere bedeutet dieses Verhalten einen Nachteil in den Eröffnungen, wenn ein "Gambit" gespielt wird, d.h., eine der beiden Seiten durch ein Bauernopfer im Eröffnungsstadium versucht, einen Entwicklungsvorsprung zu erreichen und damit die Initiative an sich zu reißen. In solchen Partien werden diese Programme zum einen

"bedenkenlos" ihre positionelle Bewertung verschlechtern, um einen vom Gegner geopferten Bauern zu behalten, und zum anderen sofort auf Remis spielen, falls ein Gambit aus der eventuell implementierten Eröffnungsbibliothek gewählt wurde.

Nach so vielen notwendigen Vorbemerkungen ist es soweit: Nun wollen wir uns ansehen, wie wir unserem Computer schachliches Wissen "beibringen" können. Dabei werden wir uns von dem jeweiligen Nutzen überzeugen, der sich durch den Einbau der Stellungsmerkmale in die Bewertungsfunktion ergibt. Dies wird u.a. durch die Betrachtung von Beispielstellungen geschehen. Ferner werden wir die Vor- und Nachteile unterschiedlicher Bewertungsmethoden diskutieren, um uns sodann für einen Lösungsansatz zu entscheiden, den wir in unserem Programm benutzen werden.

3.5.2 Die Materialbewertung

Beginnen wollen wir die Programmierung der Bewertungsfunktion bei dem Verfahren zur *Materialbewertung*, weil die Materialverhältnisse für die Einschätzung der beiderseitigen Gewinnchancen von so entscheidender Bedeutung sind.

a) Die Figurenwerte

Wie in der Einleitung angesprochen ist es dazu nötig, den verschiedenen Figurentypen Werte zuzuweisen. Durch sie wird eine Einstufung der Figuren entsprechend ihrer jeweiligen "Kampfkraft", die sie unabhängig von der aktuellen Stellung haben, vorgenommen. Als Beispiel für unterschiedliche Einsatzfähigkeiten kann eine Dame oder ein Turm in Zusammenarbeit mit dem König einen alleinstehenden gegnerischen König mattsetzen, während dies mit einem Läufer und dem König nicht möglich ist. Hier wird die Unterstützung durch einen weiteren Läufer oder einen Springer erforderlich.

In der Regel liegt die Stärke einer Figur in den Zugmöglichkeiten begründet, die ihr durch die Spielregeln zugestanden werden. So ist es klar, daß die Dame ohne Einschränkungen bezüglich

Bewegungsrichtungen oder Schrittweite eine allgegenwärtige und damit äußerst mächtige Figur ist, falls sie nicht durch eigene oder (gedeckte) gegnerische Steine behindert wird. Der Turm kann dagegen in den meisten Fällen nur etwa die Hälfte der Felder betreten. Für Läufer bzw. Springer ergeben sich Begrenzungen des Wirkungsbereiches daraus, daß sie nur auf die Felder einer Farbe ziehen können bzw. mehrere Züge benötigen, um von einem Brettrand zum gegenüberliegenden zu gelangen. Ein einzelner Bauer kann sogar schon "lahmgelegt" werden, wenn das vor ihm liegende Feld durch eine gegnerische Figur blockiert wird. (Bei Unterstützung durch weitere Bauern auf den links und rechts angrenzenden Feldern kann eine solche "Bauernwalze" allerdings zur Macht werden, der alle feindlichen, ansonsten höherwertigen Steine weichen müssen.)

Aus diesen Erfahrungen ergibt sich die Einschätzung für die Stärke der Figuren, die sich bei allen Schachspielern (Menschen und Programmen) in annähernd gleichen Werten ausdrückt. Für die Materialbewertung in unserem Programm wollen wir folgende *Figurenwerte* festlegen:

Figurentyp	Wert
König	-
Dame	900
Turm	500
Läufer	325
Springer	325
Bauer	100

Diese Werte lassen sich benutzen, um für beide Spieler ein Maß ihrer jeweiligen Materialstärke zu bestimmen; und zwar geschieht dies ganz einfach dadurch, daß wir (für Weiß und Schwarz) getrennt für alle auf dem Brett befindlichen Steine den zugehörigen Figurenwert aufsummieren. Derjenige Spieler, dessen Summe einen höheren Betrag aufweist, befindet sich materiell im Vorteil, wobei es gleichgültig ist, ob er aus einem Mehrbauern oder aus dem Besitz von höherwertigen Figuren, z.B. Turm gegen Läufer, resultiert.

Bei der Betrachtung obiger Tabelle fällt auf, daß dem König kein Wert zugewiesen wurde, da er nach den Spielregeln nicht geschlagen werden kann und daher zu beiden Summen jeweils einen konstanten Beitrag liefern würde. Wir begehen daher keinen Fehler, wenn wir die Könige bei der Materialbewertung unberücksichtigt lassen.

b) Die Materialbalance

Als nächstes stellt sich die Frage, auf welche Art wir die aus den Figurenwerten gebildeten Materialsummen miteinander vergleichen wollen, um zu unserer Materialbewertung zu kommen. Hierzu bieten sich grundsätzlich zwei Verfahren an:
Die erste und von fast allen Programmen angewandte Methode ist die Berechnung der sogenannten *Materialbalance*. Dies bedeutet, daß die *Differenz* zwischen den Materialsummen gebildet wird. Durch die Formel

Materialbewertung = Materialsumme (Weiß) - Materialsumme (Schwarz)

können wir gleichzeitig sicherstellen, daß unsere (materielle) Bewertung sich dem Schema aus der Einleitung anpaßt, wonach Vorteil für Weiß bzw. Schwarz durch positive bzw. negative Werte und Ausgleich durch den Wert "0" angezeigt wird.

Dieser Ansatz bewirkt außerdem, daß sich an der Gesamtbewertung lediglich etwas ändern kann, wenn entweder nur eine Seite Material verliert (durch einen Schlagzug) bzw. gewinnt (durch eine Bauernumwandlung), oder wenn ein ungleicher Materialabtausch stattfindet. Der Abtausch von gleichen oder gleichwertigen Figuren bleibt dagegen ohne Einfluß.

c) Das Materialverhältnis

Bei der zweiten Bewertungsmöglichkeit ist dies anders. In diesem Fall wird nämlich nicht die Differenz sondern der *Quotient* aus den Materialsummen gebildet. Folglich spricht man dann von dem *Materialverhältnis*

Materialbewertung = Materialsumme (Weiß) : Materialsumme (Schwarz) .

Diese Berechnugsvorschrift hat einige Eigenschaften, die wir näher betrachten wollen.

1. Die Könige müssen mit Figurenwerten in die Materialsummen aufgenommen werden, da ansonsten im Falle eines alleinstehenden schwarzen Königs durch "0" dividiert würde.

2. Es fällt auf, daß wir nun nur noch positive Werte für die Materialbewertung erhalten, da ausgeglichene Stellungen den Wert "1" und für Weiß (Schwarz) vorteilhafte Stellungen einen Wert >1 (<1) zugewiesen bekommen. Um diese Ergebnisse in Übereinstimmung mit unseren Anforderungen an ein symmetrisches Verhalten der Bewertungsfunktion zum Bezugspunkt "0" zu bringen, müßten wir jeweils den konstanten Wert "1" abziehen.

3. Die Materialbewertung wird bei ausgeglichenen Stellungen von der Höhe der Figurenwerte unabhängig, da der Quotient aus zwei gleichgroßen Zahlen immer gleich "1" ist, egal wie groß der Betrag dieser Zahlen ist. In diesem Fall haben die Materialsummen nur noch eine relative, aber keine absolute Bedeutung mehr.

4. Bei ungleichen Materialsummen ist dagegen die Bewertung nicht nur von den Figurenwerten, sondern auch von dem Umfang des auf dem Brett befindlichen Materials abhängig.
   Dazu ein Beispiel (mit den Figurenwerten aus der vorstehenden Abbildung sowie dem Wert "11111" für den König):

   a) Weiß gewinnt in der Eröffnung einen Bauern (ansonsten noch das Material aus der Grundstellung vorhanden).
      Daraus ergibt sich für die Bewertung:
      MB=15111:15011=1.0066

b) Weiß hat im Endspiel einen Bauern mehr (König und Bauer gegen König).
Auch hier das Materialverhältnis:
MB=11211:11111=1.0090

Es ergeben sich also unterschiedliche Bewertungen, obwohl der Materialvorteil in beiden Positionen der gleiche ist. Auf dieses interessante Verhalten werden wir gleich noch einmal zu sprechen kommen.

5. Ein vorhandener Materialvorteil läßt sich nicht mehr mit den Beträgen der Figurenwerte in Einklang bringen.
Ein weiterer Vergleich (mit den Figurenwerten wie bei 4.):

a) Material wie bei 4a), also MB=1.0066.
b) Weiß gewinnt in Fortführung von a) einen zweiten Bauern.
Daraus folgt MB=15111:14911=1.0134

Materiell betrachtet hat sich der Vorteil verdoppelt. An der Materialbewertung ist dies aber nicht abzulesen.

Aus den Punkten 3 bis 5 läßt sich schließen, daß nun auch keine Abstimmung zwischen der materiellen Bewertung und den positionellen Einschätzungen mehr möglich ist. Daraus folgt, daß dieses Verfahren nur benutzt werden kann, wenn die Materialstärken als vorrangig behandelte Entscheidungsgrundlage für die Auswahl des Zuges betrachtet werden sollen. Die positionellen Stellungsmerkmale werden dann nur noch zum Vergleich von materiell gleichwertigen Positionen herangezogen.

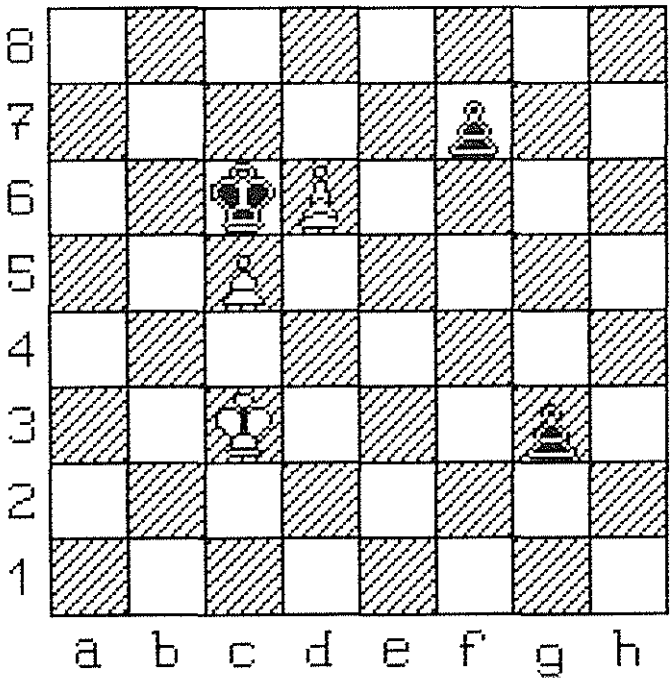
Diesen Ansatz finden wir in dem ersten voll funktionsfähigen Schachprogramm "The Bernstein Chess Program", das von einer Forschungsgruppe, bestehend aus A. Bernstein, M. DE V. Roberts, T. Artbuckle und M. A. Belsky, geschrieben und 1958 der Öffentlichkeit vorgestellt wurde (BER). Für die Durchführung der Partien kam eine IBM 704 zum Einsatz. Dies

war einer der letzten Computer, dessen Schaltungen noch mit Vakuumröhren bestückt waren (statt der moderneren Halbleiterbauelemente Diode und Transistor). Die Leistungsfähigkeit dieser Rechenanlagen war mit ca. 42000 Befehlen pro Sekunde entsprechend gering. Im Vergleich dazu ist ein Commodore 64 ungefähr 20mal schneller. Dies macht die Entscheidung von Bernstein & Co verständlich, für einen Teil der untersuchten Stellungen die zeitaufwendige positionelle Bewertung einzusparen; nur so waren beim Spiel einer Partie annehmbare Rechenzeiten für die Auswahl der Züge durch das Programm möglich. Trotzdem wurde die Geduld des menschlichen Spielpartners auf eine harte Probe gestellt: Um sich für einen Zug zu entscheiden, benötigte der Rechner im Durchschnitt ca. 8 Minuten, obwohl von der aktuellen Stellung ausgehend jeweils nur ein (kleiner) Teil der möglichen Zugfolgen und diese nur vier Halbzüge tief untersucht wurden.

d) Der Abtauschbonus

Bei der Beschreibung der Eigenschaften, die sich durch die Verwendung des Quotienten für die Materialbewertung ergeben, wurde u.a. die Abhängigkeit des Ergebnisses von dem Umfang des auf dem Brett befindlichen Materials bei ungleichen Materialstärken angesprochen (Punkt 4). Eine Schlußfolgerung aus diesem Verhalten ist, daß die im Vorteil befindliche Seite bestrebt sein sollte, gleichwertige Figuren abzutauschen, um so die Bewertung zu seinen Gunsten zu beeinflussen. In der Regel ist dies tatsächlich eine gute Strategie für einen Spieler mit mehr Material, da sich durch die Vereinfachung der Stellung ein Übergewicht häufig leichter ausnutzen läßt. Zum Beispiel spielt ein Mehrbauer im allgemeinen erst in Endspielpositionen eine Rolle, wenn das Material stark reduziert ist. Dann kann der Gegner unter Umständen nicht verhindern, daß dieser Bauer zum Freibauern gemacht und später in eine Dame umgewandelt wird.

Andererseits gibt es auch Beispiele, in denen dieses Abtauschkonzept gerade das Verkehrte ist. Betrachten wir folgende Position:



Weiß am Zug hat in dieser Stellung die Möglichkeit, den schwarzen Bauern auf G3 zu schlagen und damit einen Bauerntausch herbeizuführen, da Schwarz anschließend auf C5 nehmen kann. Damit würde Weiß zwar das Materialverhältnis zu seinen Gunsten verbessern, sich allerdings auch die letzte Gewinnchance rauben, weil er mit dem Läufer allein nicht mattsetzen

kann. Der richtige Zug in dieser Stellung ist Kc3-d4 mit der möglichen Zugfolge

Kc3-d4 g3-g2

Der Bauer könnte jetzt geschlagen werden, da der weiße König nun seinen Bauern deckt.

Ld6-h2 ...

Sonst wandelt sich der Bauer auf G1 um.

... Kc6-d7
Kd4-e4 Kd7-c6
Lh2-g1 ...

Der Läufer nimmt nun zwei Aufgaben wahr: Deckung des eigenen Freibauern und Blockierung des gegnerischen Freibauern.

... Kc6-d7
Ke4-f3 ...

Nun kann der weiße König darangehen, die gegnerischen Bauern zu fangen, da der schwarze König sich nicht zu weit von dem Bauern auf C5 entfernen darf, wenn er nicht seine Umwandlung zulassen will. Weiß hat nun eine leicht zu gewinnende Stellung erreicht.

Um solche, vor allem in Endspielen auftretende Sonderfälle richtig zu behandeln, ist es daher unumgänglich, spezielle Bewertungen vorzunehmen, um das Programm am unvorteilhaften Bauerntausch zu hindern. Das kann zum Beispiel dadurch geschehen, daß man in Endspielstellungen mit ungleichem Material einen Bonus für die stärkere Seite gibt, dessen Höhe von der Anzahl Bauern abhängt, über die sie noch verfügt.

Die Materialbalance kann Fehleinschätzungen wie in obiger Position nicht vornehmen, da sie von gleichwertigem Materialabtausch nicht beeinflußt wird. Es gibt aber in manchen

Schachprogrammen Ansätze, die trotz Berechnung der Materialbalance einen *Abtauschfaktor* berücksichtigen, um die positiven Aspekte der Eigenschaft 4 auszunutzen. Dabei kommen in der Regel ziemlich komplizierte Verfahren heraus; indsbesondere dann, wenn solche Spezialfälle wie oben richtig behandelt werden sollen. Stellvertretend sei hier die Methode aus dem Programm CHESS 4.5 der Amerikaner D. Slate und L. Atkin (SLA) beschrieben.

Die materielle Bewertung, in dem Programm mit "total material advantage" (engl.: gesamter Materialvorteil) bezeichnet, wird hier wie folgt berechnet:

total material advantage = min (3100,MS)

mit

MS = min (2400,MD) + MD * PA * (8000 - MT) / (6400 * (PA + 1))

Folgende Variablen werden in dieser Formel benutzt:

total material advantage (im folgenden mit "TMA" abgekürzt): Ergebnis der Materialbewertung

MD: Materialbalance aus der Sicht des Spielers mit Vorteil (d.h. MD=0 für ausgeglichene Stellungen; sonst positive Werte, auch bei Vorteil von Schwarz)

PA: Anzahl Bauern von der Seite mit Materialvorteil

MT: Summe des gesamten Materials auf dem Brett

Der Ausdruck "min" steht für die übliche Minimumbildung aus mehreren Zahlenwerten.

Die Eigenschaften dieser Bewertungsmethode lassen sich am besten an einigen Beispielen verdeutlichen:

1. Für materiell ausgeglichene Stellungen gilt MD=0. Daraus folgt MS=0 und schließlich TMA=0.

2. Die Veränderung von TMA durch Materialabtausch ist abhängig von den Materialsummen in der Ausgangsstellung. Die zur Berechnung der Summen benötigten Figurenwerte weisen in CHESS 4.5 folgende Beträge auf:

Figurentyp	Wert
König	-
Dame	900
Turm	500
Läufer	350
Springer	325
Bauer	100

Nun betrachten wir einige Materialverhältnisse und deren Veränderungen durch bestimmte Zugfolgen:

Material in der Ausgangsstellung:

I. Weiß hat einen Springer mehr:
Weiß: König, 1 Dame, 2 Türme, 1 Läufer, 2 Springer, 6 Bauern
Schwarz: König, 1 Dame, 2 Türme, 1 Läufer, 1 Springer, 6 Bauern

II. Weiß hat zwei Bauern mehr:
Weiß: König, 1 Läufer, 1 Springer, 3 Bauern
Schwarz: König, 1 Läufer, 1 Springer, 1 Bauer

III. Weiß hat einen Turm mehr:
Weiß: König, 1 Dame, 2 Türme, 1 Läufer, 2 Springer, 6 Bauern
Schwarz: König, 1 Dame, 1 Turm, 1 Läufer, 2 Springer, 6 Bauern

Materialveränderung durch Variante:

a) kein Abtausch

b) Abtausch von einem Bauern

c) Abtausch eines weißen Läufer gegen einen schwarzen Springer

d) Abtausch von einem Läufer

Die Tabelle aller möglichen Kombinationen von Materialverhältnissen und Varianten beinhaltet folgende Werte:

	MD	PA	MT	MS	tma	DW
I a	325	6	6675	382,67	382,67	0
I b	325	5	6475	389,53	389,53	6,86
I c	300	6	6000	380,36	380,36	-2,31
I d	325	6	5975	413,14	413,14	30,47
II a	200	3	1750	346,48	346,48	0
II b	200	2	1550	334,38	334,38	-12,10
II c	175	3	1075	317,02	317,02	-29,46
II d	200	3	1050	362,89	362,89	16,87
III a	500	6	6500	600,45	600,45	0
III b	500	5	6300	610,68	610,68	10,23
III c	475	6	5825	638,65	638,65	38,20
III d	500	6	5800	647,32	647,32	46,87

Erläuterung zur Tabelle:

Die erste Spalte bezeichnet die Kombination, für die die Materialbewertung vorgenommen wird. Die Spalten 2 bis 6 zeigen die aktuellen Werte für die in der Formel auftretenden Variablen. In der letzten Spalte steht der jeweilige Differenzwert DW (der sogenannte "Abtauschbonus"). Er gibt an, für wie günstig das Programm den Abtausch von Material hält. Diese Werte beziehen sich immer auf den zugehörigen Fall a) (Variante ohne Abtausch). An ihnen können wir folgendes ablesen:

(i) Der Spieler mit Materialvorteil wird bestrebt sein, gleichwertiges Material abzutauschen; und zwar umso mehr, je größer der Vorteil ist (siehe positive Werte für DW in den Kombinationen Id, IId, IIId).

(ii) Bei genügend großem Übergewicht wird auch ein Abtausch befürwortet, der den Vorteil geringfügig reduziert (siehe die Fälle Ic, IIc, IIIc: Die Materialbalance MD sinkt um den Betrag 350-325=25. Dies ist die Differenz zwischen den Figurenwerten von Läufer und Springer. Daraus resultiert eine Verringerung von TMA für die Kombinationen Ic und IIc (negative Werte für DW), führt aber zu einer Erhöhung von TMA bei einem Turm Vorsprung (positiver Wert für DW im Fall IIIc)).

(iii) Durch den Anteil von "PA/(PA+1)" im 2. Summanden zur Bestimmung von MS soll der Spieler mit Materialvorteil daran gehindert werden, Bauern zu tauschen, falls er nur noch über eine geringe Anzahl Bauern verfügt (siehe negativen Wert für DW in der Kombination IIb; dagegen positive Werte in den Fällen Ib, IIIb). Die Beispielstellung aus der letzten Abbildung wird übrigens dadurch richtig behandelt. Es ergibt sich nämlich für die Zugfolge Ld6xg3, Kc6xd5 eine Verschlechterung der materiellen Bewertung um DW=-203.71.

Zusammenfassend können wir feststellen, daß es den Programmautoren Slate und Atkin gelungen ist, die strategische Bedeutung des Materialabtausches gut in die Bewertung zu integrieren. Allerdings muß dafür ein ziemlich hoher Rechenaufwand in Kauf genommen werden, da die Ermittlung von TMA mit Hilfe der oben angegebenen Formel wesentlich länger als die Bestimmung der einfachen Materialbalance dauert.

Dazu kommt, daß z.T. relativ hohe Beträge als Abtauschbonus vergeben werden (z.B. DW=46.87 im Fall IIId, d.h. der halbe Wert eines Bauern!), so daß der ohnehin schwache Einfluß der

positionellen Stellungsmerkmale auf die Gesamtbewertung weiter reduziert wird.

e) Verfahren in unserem Programm

Nachdem wir bisher einige Verfahren zur Materialbewertung kennengelernt und ihre wesentlichen Vor- und Nachteile analysiert haben, besitzen wir nun eine ausreichende Entscheidungsgrundlage dafür, welchen Ansatz wir in unserem Programm benutzen wollen. Aufgrund der angesprochenen Schwierigkeiten, die sich durch die Verwendung des Quotienten aus den Materialsummen ergeben, werden wir die Materialbalance zur Bewertung heranziehen. Des weiteren müssen wir die besondere Rolle der Bauern in Endspielpositionen berücksichtigen. Das soll durch einen speziellen Bonus geschehen, den wir bei ungleichen Materialstärken und ausschließlich in Endspielstellungen dem im Vorteil befindlichen Spieler zukommen lassen. Der Betrag wird abhängig von der Bauernanzahl dieses Spielers durch die Formel

$$\text{Bonus} = \min(\text{Bauernanzahl},3) * 10$$

ermittelt. Damit erreichen wir, daß (durch die Minimumbildung in gewissen Grenzen) die Bauern der einen Seite praktisch einen höheren Figurenwert erhalten und daher nur noch gegen gegnerische Bauern getauscht werden, wenn die Verringerung des Bonus durch eine steigende positionelle Bewertung kompensiert wird. Auf die Berechnung eines generellen Abtauschbonus wie in CHESS 4.5 werden wir dagegen verzichten, um den Zeitaufwand für die Materialbewertung nicht zu sehr zu erhöhen.

Bevor wir im nächsten Abschnitt zu den positionellen Stellungsmerkmalen kommen, noch einige Anmerkungen zur Umsetzung des ausgewählten theoretischen Ansatzes in unserem Programm: Die Berechnung der Materialbalance läßt sich auf zwei Arten durchführen.

Die erste ist die am einfachsten zu programmierende, aber auch die rechenzeitaufwendigste Methode. Hier werden unabhängig voneinander für jede Endstellung einer Zugfolge im Suchbaum die beiden Materialsummen gebildet, indem die Figurenwerte

für die auf dem Brett befindlichen Steine aufaddiert werden. Etwas beschleunigen läßt sich dieses Verfahren, wenn man zur Informationsbeschaffung über die vorhandenen Figuren nicht die Einträge der Brettdarstellung sondern die Daten der Materiallisten benutzt, da dann anstelle von 64 Rechenoperationen (der Felderanzahl entsprechend) nur noch eine Anzahl ausgeführt werden muß, die sich aus der tatsächlichen Figurenanzahl für die vorliegende Stellung ergibt.

Trotzdem wird auf diese Weise eine Menge Rechenzeit verschenkt. Es gibt nämlich ein Verfahren, das den Aufwand auf das absolut notwendige Minimum beschränkt. Durch die sogenannte *inkrementelle Bearbeitung* wird erreicht, daß nur noch Berechnungen anfallen, wenn sich Veränderungen bezüglich des Materials ergeben.

Dies sieht detailliert so aus: Für die Ausgangsstellung des Spielbaumes muß (einmalig) die vollständige Bestimmung der Materialsummen vorgenommen werden. Von diesen Werten ausgehend werden bei der Untersuchung einer Zugfolge jeweils bei der Aktualisierung der Stellung durch Ausführen des entsprechenden Zuges die evtl. veränderten Materialstärken in Abhängigkeit von der Zugart (Schlagzug, Bauernumwandlung, sonstige Züge) "hochgerechnet", indem z.B. bei einem Schlagzug der Wert der geschlagenen Figur von der zugehörigen Materialsumme abgezogen wird. Im Falle der sogenannten "ruhigen" Züge haben wir so eine hundertprozentige Ersparnis an Rechenzeit, da für sie keine Berechnungen erforderlich sind. Führen wir diese Aktualisierung der Materialbewertung entlang der Züge aller zu untersuchender Varianten durch, so erhalten wir mit minimalem Aufwand (bzgl. der Summenbildung) die Werte der Materialbalance für alle Endspielstellungen im Suchbaum.

Der einzige Nachteil liegt darin, daß wir bei jedem Ausführen eines Zuges uns die alten Summenwerte "merken" müssen, damit diese Informationen auch bei Veränderungen nicht verloren gehen und wir auf sie zurückgreifen können, wenn der Zug im Verlauf der Suche zurückgenommen und der alte Zustand vor dem Vertiefen eines Alternativzuges wieder hergestellt werden soll.

Dazu bietet sich der *Stack* an, den wir bei der Beschreibung des Suchalgorithmus kennengelernt haben. In diesem werden Daten abgelegt, die durch die Ausführung von Zügen überschrieben werden können und die später bei der Rückkehr zur gleichen Stellung wieder benötigt werden. Auf dieselbe Art lassen sich die Materialsummen behandeln, so daß wir in diesem Keller auch unsere Materialwerte abspeichern werden. Selbst wenn dadurch vier zusätzliche Speicheroperationen pro Stellung notwendig werden (2 Summenwerte ablegen, 2 Summenwerte holen), so erzielen wir mit diesem Verfahren doch eine große Zeitersparnis im Vergleich zu den vollständigen Summenberechnungen, zumal für die mit Abstand größte Anzahl von Stellungen im Suchbaum, den Endpositionen, diese Kellereinträge nicht ausgeführt werden müssen, da von ihnen aus keine Züge mehr untersucht werden.

Außerdem gewinnen wir den Vorteil, daß wir jetzt in allen Positionen und nicht mehr nur in den Endstellungen die Materialbalance zur Verfügung haben. Dies ist für einige Beschleunigungsmaßnahmen, die wir noch kennenlernen werden, von großer Bedeutung. Wegen dieser geschilderten positiven Effekte werden wir das inkrementelle Konzept auch in unserem Programm benutzen.

Die Besprechung der Materialbewertung ist damit abgeschlossen. Die Programmierung der ausgewählten Verfahren wird in Kapitel 4 anhand des Listings in einer detaillierten Dokumentation erläutert. In unserem nächsten Abschnitt über die Theorie der Schachprogrammierung werden wir uns nun mit der Bewertung der positionellen Stellungsmerkmale beschäftigen.

3.5.3 Die positionelle Bewertung

Den einleitenden Worten aus Abschnitt 3.5.1. zufolge haben wir bei der Programmierung des positionellen Teils der Bewertungsfunktion zwei Schwerpunkte zu behandeln. Der erste ist die Auswahl der zu implementierenden Stellungsmerkmale, die für eine gute Positionsanalyse vonnöten sind. Sodann müssen wir uns

überlegen,wie stark diese Einzelbewertungen zu wichten sind, damit sie jeweils in Übereinstimmung mit ihren tatsächlichen Bedeutungen in der zu bewertenden Stellung stehen. Auf den folgenden Seiten wollen wir versuchen, diese Aufgaben, so gut es geht, zu bewältigen. Dabei werden wir uns einerseits von der Schachtheorie leiten lassen und uns andererseits ansehen, welche Lösungsansätze in anderen Schachprogrammen schon verwendet wurden.

a) Einteilung der Stellungen in Klassen

Da sowohl die Auswahl als auch die Wichtung der zu berücksichtigenden Stellungsmerkmale (unter Umständen stark) von dem jeweils vorliegenden Stellungstyp abhängen, müssen wir zunächst die Stellungen nach bestimmten Gesichtspunkten in unterschiedliche Klassen einteilen, innerhalb derer eine mehr oder weniger einheitliche Bewertungsmethode benutzt werden kann. Ein grobes Raster hierfür ergibt sich durch folgenden Ansatz:

1. entschiedene Stellungen (matt, patt, technisches Remis)

2. "fast" entschiedene Stellungen (Positionen während der Mattführung)

3. restliche Stellungen

In den nächsten Abschnitten werden wir auf die einzelnen Klassen eingehen. Dabei haben wir mehrere Fragen zu klären:
Durch welche Eigenschaften grenzen sich diese Klassen voneinander ab?
Welche Bewertungen werden auf die Stellungen innerhalb einer Klasse angewandt?
Kann die große Klasse der "restlichen Stellungen" weiter unterteilt werden?

b) Entschiedene Stellungen

Die Positionen, die unter diese Kategorie fallen, bereiten uns keinerlei Probleme. Das liegt daran, daß der entsprechende Sachverhalt durch die Spielregeln fest definiert ist und daher leicht festgestellt werden kann. Für *Matt- und Pattstellungen* erledigt diese Aufgabe der Suchalgorithmus (durch Überprüfung der Zugmöglichkeiten und Kenntnis einer eventuellen Bedrohung des Königs). Dies wurde im Abschnitt "Suchen und Entscheiden" beschrieben. Dabei wurde auch auf das "Bewertungsverfahren" eingegangen. Für solche Positionen wird erst gar nicht die Bewertungsfunktion aufgerufen, sondern direkt von der Suche werden die Werte den Stellungen zugewiesen ("16000-t" für das Matt des schwarzen Königs, "-16000+t" für das Matt des weißen Königs, wobei "t" die Suchtiefe der Mattstellung angibt, sowie der Wert "0" für Pattstellungen).

In die Klasse der entschiedenen Stellungen gehört aber auch das sogenannte *"technische Remis"*. Damit sind alle Positionen gemeint, in der beide Spieler materiell so stark geschwächt sind, daß sie selbst bei Fehlern des Gegners nicht mehr den Sieg (das Matt) erreichen können. Beispiele hierfür sind Materialverhältnisse mit alleinstehenden Königen, aber auch mit maximal einer Leichtfigur (Springer oder Läufer) auf jeder Seite. Selbst ein Endspiel mit König und zwei Springern gegen König ist "tot remis", wie man unter Schachspielern sagt.

In solchen Stellungen hat die Bewertungsfunktion leichtes Spiel: Sie braucht nur von beiden Parteien die Anzahl und eventuell die Typen der vorhandenen Figuren zu untersuchen. Auf diese Art lassen sich ohne weiteres die wenigen Sonderfälle herausfiltern, die ebenso wie Pattstellungen den Wert "0" bekommen. Falls (im Zusammenhang mit der Materialbewertung) nicht nur die Summen sondern auch die vorhandenen Figurentypen bestimmt werden, können die Abfragen auf technisches Remis sogar sehr schnell bearbeitet werden. Da wir aber in unserem Programm auf diese Zusatzinformation verzichten wollen, weil deren Bestimmung einiges an Rechenzeit erfordert, lassen wir den (äußerst seltenen) Fall "König + 2 Springer / König" unberücksichtigt. Für die anderen Materialverhältnisse können

wir dagegen auch ohne Kenntnis der Figurentypen die Überprüfung auf technisches Remis leicht vornehmen.

c) "Fast" entschiedene Stellungen

In der Regel, d.h. fast während der gesamten Partie, stehen Schachspieler - ob Menschen oder Computer - vor der schwierigen Aufgabe, aus einer ausgeglichenen Stellung einen Vorteil herauszuholen, einen erworbenen Vorteil festzuhalten und möglichst auszubauen oder bei gegnerischem Übergewicht um den Ausgleich zu kämpfen. Um diese Probleme zu lösen, sind relativ komplizierte Entscheidungen zu treffen, die in den Schachprogrammen unter anderem von seiten der Bewertungsfunktion entsprechend komplexe positionelle Abschätzungen erforderlich machen.

Ist jedoch schon eine Vorentscheidung gefallen (mit anderen Worten: Besitzt ein Spieler einen Vorteil, der zum baldigen Mattsetzen ausreicht), dann müssen in der Bewertung ganz andere positionelle Kriterien berücksichtigt werden. Diese Unterscheidung wird von allen Schachprogrammen vorgenommen.

Betrachten wir als Beispiel ein Endspiel mit König und Turm gegen einen alleinstehenden König. Das Ziel "Matt des gegnerischen Königs" läßt sich nur erreichen, wenn

1. der gegnerische König an den Brettrand gedrängt wird

und

2. der eigene König seinem Opponenten möglichst "dicht auf den Fersen" bleibt, um ihm zum einen zusammen mit dem Turm die Fluchtwege abzuschneiden und zum anderen den Turm vor dem Geschlagenwerden zu schützen.

Diese beiden Stellungsmerkmale reichen schon aus, um ein Endspiel mit König und Turm bzw. Dame gegen König zu

gewinnen. Müssen dagegen Springer an der Mattführung beteiligt werden (z.B. König und Läufer und Springer gegen König), dann muß wegen seines kleinen Wirkungsbereiches auch der Abstand zwischen Springer und gegnerischem König so gering wie möglich gehalten werden.

In der *"Matt-Routine"*, das ist das Unterprogramm innerhalb der Bewertungsfunktion, das die Bewertungen während des Mattsetzens ermittelt, sind also nur Abstandsbestimmungen durchzuführen. Um die Stellung bzgl. der Erfüllung von Teilaufgabe 1 einzuschätzen, wird am einfachsten die Distanz zwischen dem gegnerischen König und der (geometrischen) Brettmitte berechnet, da ansonsten die Abstände zu allen vier Bretträndern ermittelt und anschließend das Minimum für diese Ergebnisse gebildet werden müßte. (Dieser Wert muß dann allerdings aus der Sicht des mattsetzenden Spielers größer werden!)

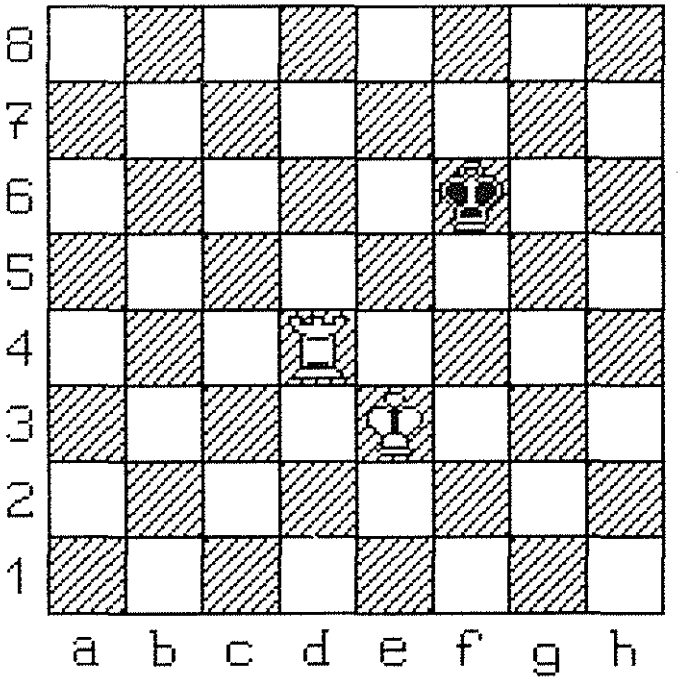
Sowohl der Königsstandort als auch der Brettmittelpunkt kann durch ein Zahlenpaar angegeben werden, wenn man die Reihen und Linien des Brettes durchnumeriert. Unter Berücksichtigung der von uns gewählten rechnerinternen Brettdarstellung, die im Zusammenhang mit dem Zuggenerator beschrieben wurde, benutzen wir für die Reihen 1-8 die Werte 2 bis 9 und für die Linien A-H die Werte 3 bis 10. Damit ist gewährleistet, daß wir für jedes Feld auf dem Brett durch den Ausdruck "12 * Reihenwert + Linienwert" die im Computer benutzte Zahl berechnen können. Anders herum läßt sich aus dem rechnerinternen Wert eines Feldes der Reihen- sowie der Linienwert bstimmen. Dies geschieht durch die ganzzahlige Division des Feldwertes durch die Konstante 12. Zum Beispiel erhalten wir für das Feld f5 (=80):

80 : 12 = 6 Rest 8

Der Reihenwert 6 ergibt die 5. Reihe und der Linienwert 8 die F-Linie. Der rechnerische Mittelpunkt des Brettes liegt zwischen der 4. und der 5. Reihe sowie der D- und der E-Linie und wird durch das Zahlenpaar (5.5,6.5) gekennzeichnet.

Auf diese Art lassen sich zwischen beliebigen Punkten des Brettes die Abstände berechnen, indem wir getrennt den Reihen- und den Linienabstand bestimmen und diese Werte dann aufsummieren. Mit der *Abstandsbetrachtung* haben wir übrigens ein elementares Bewertungsverfahren kennengelernt, das wir später auch noch bei anderen positionellen Stellungsmerkmalen gebrauchen können.

An der Bewertung einer Beispielstellung wollen wir das bisher Besprochene noch einmal verdeutlichen:



Die Standorte der beiden Könige lassen sich nach obiger Methode durch die Zahlenpaare (4,7) für Weiß und (7,8) angeben. Daraus ergibt sich für den Abstand

schwarzer König-Brettmitte: (7-5.5)+(8-6.5)=3

sowie für den Abstand

schwarzer König-weißer König: (7-4)+(8-7)=4.

Diese Berechnungen lassen sich aber nicht unmittelbar zur Stellungsbewertung heranziehen, da aus der Sicht des mattsetzenden Spielers der erste Abstandswert so groß, der zweite jedoch so klein wie möglich sein sollte. Daß wir beide Werte gleichermaßen benutzen können, erreichen wir durch steigende Bewertungen bei sich verringerndem Abstand für das zweite Stellungsmerkmal. Dies geschieht dadurch, daß wir den aktuellen Abstand (im Beispiel =4) von der maximal möglichen Distanz (=14, z.B. für Könige auf A1 und H8) abziehen. Die so konstruierte Differenz (im Beispiel =10) leistet das Gewünschte.

Nun müssen diese beiden Einzelbewertungen nur noch entsprechend ihrer Bedeutung für das Erreichen einer Mattposition gewichtet und damit auch gleichzeitig in Beziehung zur Materialbalance gebracht werden. Dies erreichen wir durch die in der Einleitung angesprochenen Wichtungskoeffizienten. In unserem Programm wollen wir die Beträge 9.4 für den Abstand des mattzusetzenden Königs zur Brettmitte bzw. 1.6 für den Abstand der Königsstandorte verwenden. Damit wird das Abdrängen des gegnerischen Königs an den Brettrand zum vordringlichen Ziel erklärt. Gleichhohe, aber negative Werte werden für Bewertungen beim Mattsetzen des weißen Königs benutzt, um das symmetrische Verhalten der Bewertungsfunktion sicherzustellen.

So erhalten wir für die letzte Stellung folgende Gesamtbewertung:

500		Materialbalance	
+ 9,4 * 3		Abstand SK - Brettmitte	positionelle
+ 1,6 * 10		Abstand WK - SK	Bewertung
= 544,2			

Nachdem wir uns auf ein Bewertungsverfahren für Positionen während der Mattführung festgelegt haben, bleibt die Frage zu

klären, wie groß der Vorteil eines Spielers eigentlich sein muß, damit wir eine Stellung aus dieser Klasse vorliegen haben. Es ist klar, daß uns die beiderseitigen Materialstärken darüber Aufschluß geben müssen. Folgende Bedingungen sollten erfüllt sein, wenn wir die Mattroutine für Weiß oder Schwarz aufrufen wollen:

1. Die Materialbalance liefert einen Wert, dessen Betrag mindestens dem Wert eines Turmes entspricht (Minimumforderung an den mattsetzenden Spieler, um die Mattführung realisieren zu können).

2. Die Materialsumme des mattzusetzenden Spielers übersteigt nicht den Wert von zwei Leichtfiguren (Maximumforderung zur Einschätzung der Verteidigungsmöglichkeiten).

3. Es sind in der Stellung keine Bauern mehr vorhanden (Forderung an beide Spieler: Zum einen soll der mattzusetzende Spieler sein Verteidigungspotential nicht mehr erhöhen können. Auf der anderen Seite ist es für den im Vorteil befindlichen Spieler im allgemeinen leichter, noch vorhandene Bauern erst umzuwandeln und so das Materialübergewicht zu vergrößern, als gleich das Matt anzustreben. Das gilt auch und insbesondere dann, wenn die Bedingungen 1 und 2 erfüllt sind.)

Bei der Überprüfung der *Voraussetzungen zum Mattsetzen* können wir die Materialverhältnisse zweckmäßigerweise auch gleich daraufhin testen, ob die Position "technisch remis" ist (s.o.). Im Rahmen der Abfragen 1-3 lassen sich solche Stellungen einfach herausfiltern, da für sie die Aussage 3 ebenfalls zutrifft und zudem die Bedingung 2 für beide Spieler sogar verschärft gilt. Es bietet sich also an, die aktuelle Stellung in einer der eigentlichen Bewertung vorgeschalteten Analyse gleichzeitig auf diese beiden Stellungstypen hin zu untersuchen. Diesen Weg werden wir auch in unserem Pogramm beschreiten.

d) Restliche Stellungen

Die bisher angesprochenen Stellungstypen werden wir in unserem Suchbaum sehr selten, und wenn, dann nur gegen Ende der Partie wiederfinden. Für den großen Rest der Stellungen haben wir noch keine Einteilung gefunden. Mögliche Ansätze für eine Unterscheidung der Positionen wurden schon in der Einleitung erwähnt: das *Partiestadium* und der *"Stellungscharakter"*.

Das Partiestadium

Betrachten wir zunächst das Partiestadium. Ein Spielverlauf läßt sich im wesentlichen in drei Phasen einteilen. Die erste ist die *Eröffnung*, die dadurch gekennzeichnet ist, daß beide Spieler versuchen, möglichst schnell ihre Kräfte zu mobilisieren, d.h., die Figuren von den Grundreihen (1. Reihe für Weiß, 8. Reihe für Schwarz) aus auf Felder zu bringen, auf denen sie einen größeren Wirkungsbereich haben. Des weiteren wird versucht, durch frühzeitiges Ausführen der Rochade den König auf einen Platz in der Nähe eines Eckfeldes zu schaffen, wo er vor gegnerischen Angriffen geschützt und seinen eigenen Figuren nicht im Weg steht.

Dieser Partieabschnitt ist in der Regel beendet, wenn beide Seiten ihre Entwicklung abgeschlossen, d.h. bis auf den König und die Türme alle Figuren die Grundreihen verlassen und die Könige rochiert haben. Doch der Übergang von der Eröffnung ins Mittelspiel ist häufig fließend und von der gewählten Eröffnungsvariante abhängig. So gibt es manche Zugfolgen, in denen einzelne Figuren erst nach relativ langer Zeit ins Geschehen eingreifen und dafür andere Figuren mehrmals bewegt werden.

Das *Mittelspiel* ist geprägt vom beiderseitigen Bemühen, einen materiellen Vorteil zu erringen oder, seltener, den gegnerischen König direkt mattzusetzen. Diese Bestrebungen setzen sich aus zwei Kampfelementen, dem *taktischen* und dem *strategischen*, zusammen. Durch taktische Züge wird der Versuch unternommen, zu kurzfristigem Material- oder Partiegewinn zu kommen, ohne daß es dem Gegner gelingt, dies zu verhindern. Eine Folge

von taktischen Zügen wird *Kombination* genannt. Häufig beginnen solche Kombinationen mit einem vom Gegner nicht erwarteten Opfer eigenen Materials, durch das man mehrere Drohungen auf einmal schafft, die nicht alle gleichzeitig pariert werden können, sodaß man später das Material "mit Zinsen" zurückbekommt. Diese taktische Seite des Schachspiels liegt den meisten Programmen am besten, da es gerade ihre Stärke ist, viele Zugfolgen mit großer Suchtiefe zu analysieren, zumal der Erfolg einer Kombination sich leicht an der Materialbalance ablesen läßt.

Doch die Erfolgsaussichten hängen in den meisten Stellungen getreu dem Motto "Ohne Fleiß kein Preis!" von dem strategischen Geschick ab, mit dem sich ein Spieler erst Angriffsmöglichkeiten erarbeiten muß. Chancen zu einer gewinnbringenden Abwicklung ergeben sich nämlich nicht von allein - von (taktischen) Fehlern des Gegners einmal abgesehen -, sondern sie müssen in einem oft lang andauernden zähen Ringen erkämpft werden, bei dem beide Spieler versuchen, günstige Aufstellungen für die eigenen Figuren zu finden und ernsthafte Schwächen in der gegnerischen Formation zu provozieren. Auf diesem Gebiet haben praktisch alle Schachprogramme ihre Schwachstellen, weil die zum Teil lückenhafte und zu grobe Bewertung der positionellen Stellungsmerkmale nur unzureichend durch große Suchtiefen beim Analysieren der Zugfolgen kompensiert werden kann.

Fällt im Mittelspiel noch keine Entscheidung durch einen unabwendbaren Mattangriff, so wird doch im Verlauf der Partie soviel Material getauscht, daß man ab einer bestimmten Grenze vom *Endspielstadium* sprechen kann. Häufig wird der Übergang ins Endspiel von demjenigen Spieler angestrebt, der einen materiellen Vorteil erringen konnte und nun durch die Verringerung der Verteidigungskräfte und damit auch durch eine Vereinfachung der Stellung sein Übergewicht bequemer ausnutzen will.

Das Erreichen einer Endspielstellung wird durch die auf dem Brett befindlichen Figuren bestimmt: Wenn jeweils das gegnerische Material so weit reduziert ist, daß der König seinen

geschützten Standort verlassen und aktiv am Spiel teinehmen kann, ohne einen direkten Mattangriff fürchten zu müssen, ist der Eintritt ins Endspielstadium vollzogen. Damit erlangen auch die Bauern eine große Bedeutung, da beide Seiten versuchen, Freibauern zu bilden - das sind Bauern, die nicht mehr durch gegnerische Bauern durch Blockierung oder Schlagen an der Umwandlung gehindert werden können - und durch deren Umwandlung zu einem spielentscheidenden Materialplus zu kommen. Bei der Ausführung dieses Planes kann im übrigen ein im Mittelspiel gewonnener Vorteil wichtige Unterstützung leisten.

Für unser Schachprogramm bedeutet die Einteilung des Partieverlaufes in drei Abschnitte, daß wir (durch Auswahloperatoren und Wichtungskoeffizienten) dafür sorgen müssen, daß die Bewertungen der positionellen Stellungsmerkmale möglichst gut an ihre jeweiligen Bedeutungen angepaßt werden. Dementsprechend muß es möglich sein, daß die Werte je nach Partiestadium unterschiedlich stark bzw. überhaupt nicht in die Gesamtbewertung einfließen. Wir werden versuchen, dieses Ziel zu erreichen, indem wir zum einen eröffnungsorientierte Kriterien in die Bewertungsfunktion aufnehmen, wobei wir, dem fließenden Übergang zwischen Eröffnung und Mittelspiel Rechnung tragend, bei den Bewertungen zwischen Positionen dieser beiden Stadien keinen Unterschied machen werden. Diese Stellungsmerkmale haben den Zweck, eine schnellstmögliche Entwicklung der Figuren zu fördern.

Zum anderen wollen wir eine sogenannte *"Endspielabfrage"* programmieren, die die Materialverhältnisse untersucht und uns sagt, welche Stellungen als Endspielositionen behandelt werden müssen. Dies geschieht insbesondere in Hinblick auf die Bewertungen, die für die Könige und die Bauern vorgenommen werden und die oben erläuterten unterschiedlichen Pläne im Mittelspiel bzw. im Endspiel berücksichtigen sollen.

Der Stellungscharakter

Eine weitere Möglichkeit, Stellungen von der Bewertungsfunktion unterschiedlich behandeln zu lassen, ist der *"Charakter"*

einer Stellung. Hiermit ist eine Eigenschaft gemeint, für die die Zugmöglichkeiten beider Spieler ausschlaggebend ist. Diese werden fast immer von den beiderseitigen Bauernformationen beeinflußt. So gibt es Positionen, in denen die Bauern so aufgestellt sind, daß sie insbesondere die sonst langschrittigen Damen, Türme und Läufer behindern und deren Aktionsradien stark einengen. Daher ist auch eine Umgruppierung der Kräfte nicht so leicht möglich, und die Figuren sind in der Regel auf die Wahrnehmung von ein oder zwei Aufgaben beschränkt. In solchen Fällen spricht man von *geschlossenen Stellungen*. Das Gegenteil dazu sind die *offenen Stellungen*. Hier existieren z.B. offene, d.h. durch keine eigenen oder gegnerische Bauern verstellte Linien, über die rasche Standortwechsel möglich sind. Dementsprechend kann sich in solchen Stellungen das Kampfgeschehen schnell verlagern, da die Figuren kurzfristig ihre Angriffsziele wechseln können.

Der Charakter einer Stellung sollte bei der strategischen Planung des weiteren Parieverlaufes berücksichtigt werden. Zum Beispiel kann der König in geschlossenen Stellungen manchmal früher aktiviert werden, weil dem Gegner nicht alle Figuren zum Ausnützen eines etwas unsichereren Königsstandortes zur Verfügung stehen. Der Übergang ins Endspiel findet also eher statt. Des weiteren bekommen Bauernzüge eine größere Bedeutung, da sie beispielsweise die Öffnung einer Linie vorbereiten können, über die dann eventuell die Türme ins gegnerische Lager eindringen.

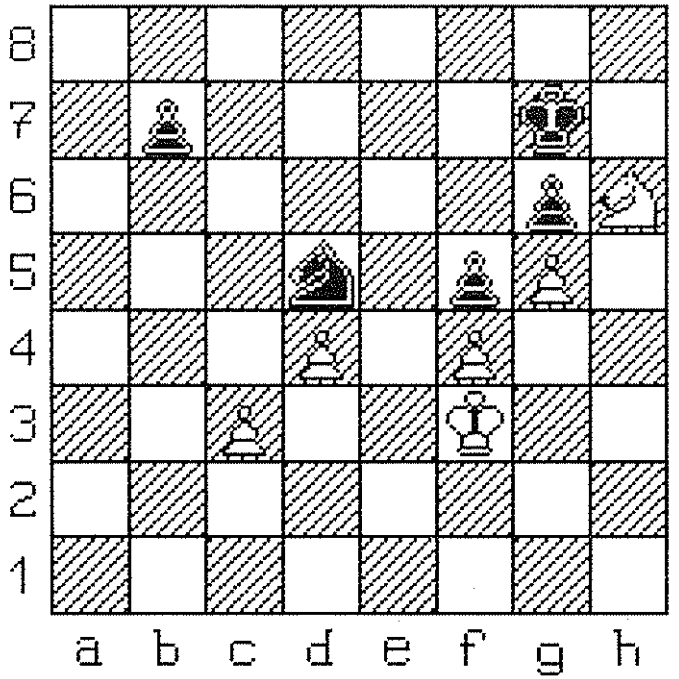
Schachprogramme tun sich allerdings recht schwer, solche Positionen korrekt einzuschätzen, weil die Pläne meist nur so langfristig realisiert werden können, daß der mögliche Erfolg nicht innerhalb der Suche festgestellt werden kann. Die Bewertung der positionellen Stellungsmerkmale liefert daher im besten Fall nur Hinweise für die Wahl der richtigen Spielweise. Entsprechend selten findet man in den Bewertungsfunktionen der Schachprogramme den Stellungscharakter berücksichtigt.

Grundlagen der positionellen Bewertung

Welche Kriterien können wir aber nun zur positionellen Bewertung heranziehen?

Die Berechnung der Materialbalance trifft eine Aussage über die "generelle" Schlagkraft für die Figuren, die den Spielern zur Verfügung stehen. Dieses Maß ist von den Figurenstandorten in der aktuellen Stellung unabhängig. Es liegt daher nahe, in der positionellen Bewertung Abschätzungen bzgl. der Wirkungsbereiche der Figuren vorzunehmen, die sich aus ihrer Aufstellung ergeben.

Drei Komponenten beeinflussen den Aktionsradius der Figuren. Am Beispiel der Bewertung von Freibauern läßt sich dies gut verdeutlichen. Betrachten wir folgende Position:



1. Komponente: Bedeutung des Standortes selbst

Je weiter der Bauer vorgerückt ist, desto intensiver wird für den Gegner die Bedrohung durch eine eventuelle Umwandlung. Stünde der Freibauer nicht auf d4 sondern schon auf d7, so wären die gegnerischen Kräfte (König und Springer) wesentlich stärker an die Bekämpfung des Bauern gebunden.

2. Komponente: Gegenseitige Beeinflussung von Figuren beider Farben

Eine wichtige Maßnahme im Kampf gegen einen Freibauern besteht darin, das vor dem Bauern liegende Feld zu blockieren und ihn so am Weiterziehen zu hindern. In der Beispielstellung wird diese Aufgabe von dem schwarzen Springer auf D5 wahrgenommen.

3. Komponente: Zusammenwirken der Figuren einer Farbe

Allein kann ein Freibauer nur in den seltensten Fällen die Umwandlung realisieren. Dazu bedarf es meistens der Unterstützung durch weitere Figuren. Ideal ist es z.B., wenn zwei Bauern "Seite an Seite" der gegnerischen Grundreihe zustreben. So schaffen sie sich gegenseitig freie Bahn, da eine Blockade dieser Bauern durch gegnerische höherwertige Steine wegen des dann drohenden Materialverlustes nicht so leicht möglich ist. In obiger Position kann durch den Bauernvorstoß c3-c4 die Verteidigungsfigur auf D5 vertrieben werden.

Mit der Bewertung solcher (statischer) Stellungsmerkmale bekommen wir allerdings nur eine "Momentaufnahme"; selbst kurzfristige Veränderungen werden auf diese Art nicht erkannt. Verlegen wir z.B. den Standort des schwarzen Springers von D5 nach D7, so gilt der Freibauer auf D4 als nicht blockiert und wird entsprechend hoch bewertet. Erst eine zwei Halbzüge tiefe Suche, während der unter anderem die Zugfolge d4-d5, Se7xd5 analysiert wird, stellt fest, daß der Freibauer z.Zt. nur unter Materialverlust weiterziehen kann. Daher sollte eine weitere, dynamische Komponente in die positionelle Bewertung aufgenommen werden, die zumindest die in der zu bewertenden Stellung gegebenen Zugmöglichkeiten berücksichtigt. Einen praktikablen Ansatz hierfür werden wir gleich kennenlernen.

Dieses kleine Beispiel gewährt uns einen ersten Einblick in die positionellen Zusammenhänge, die wir mit unserer Bewertungsfunktion erfassen wollen. Im folgenden werden wir uns eingehend mit den einzelnen Stellungsmerkmalen und den dazugehörigen Bewertungsmethoden beschäftigen. An den Anfang

stellen wir die eben erwähnten Bewertungen der Zugmöglichkeiten, weil wir Teilaspekte der dort benutzten Verfahren später bei der Beschreibung der Abschätzungen für die statischen Kriterien wiederfinden werden.

Die Bewertung der Zugmöglichkeiten

- Mobilität, Felderkontrollen

Im Jahr 1950 veröffentlichte C.E. Shannon zwei Artikel (SHA1, SHA2), in denen er sich mit den theoretischen Möglichkeiten beschäftigte, einen Computer für das Schachspiel zu programmieren. Schon damals schlug er vor, den dynamischen Charakter einer Stellung in die Bewertung einzubeziehen. Um dies zu erreichen, sollten für beide Spieler die Summen ihrer legalen, d.h. den eigenen König nicht ins Schach stellenden Zugmöglichkeiten berechnet und ihre Differenz gewichtet und in die Gesamtbewertung aufgenommen werden. Durch die Beschränkung auf die legalen Züge wurden gleichzeitig eventuell vorkommende gefesselte Figuren bestraft, da durch sie die sogenannte *"Mobilität"* verringert wird. Das Ziel dieser Bewertungsmethode war es, ein Maß für die Chancen zur freien Entfaltung der Kräfte zu erhalten, da im allgemeinen mit der wachsenden Anzahl von Zugmöglichkeiten auch die Wahrscheinlichkeit steigt, eine günstige Figurenaufstellung zu erlangen, die die Durchführung strategischer Pläne oder taktischer Kombinationen erlaubt.

Allerdings ist diese Art der Zusammenfassung in einem einzigen Wert für eine genaue Einschätzung doch sehr grob, weil alle Züge ohne Unterschied behandelt werden; weder der Typ der ziehenden Figur noch die Region des Zielfeldes werden berücksichtigt. Dabei spielen diese beiden Aspekte eine nicht unwesentliche Rolle. Zu den generell wichtigen Bereichen des Brettes, deren Kontrollierung durch Figuren erstrebenswert ist, gehören das Zentrum (die Felder D4, E4, D5 und E5) und die Königsumgebungen (die Nachbarfelder der beiden Königsstandorte), da einerseits die Beherrschung des Zentrums den Figuren einen großen Aktionsradius sichert und andererseits die Figuren in der Lage sein müssen, den eigenen König zu schützen und

Angriffe auf den gegnerischen König vorzubereiten. Beispiele für einzelne bedeutsame Felder, die sich aus der aktuellen Stellung ergeben können, sind die Felder vor Freibauern (s.o.) oder die Standorte von Figuren, die aus bestimmten Gründen ihren Platz nicht verlassen können oder dürfen (z.B. gefesselte Figuren, rückständige Bauern, an Verteidigungsaufgaben gebundene Figuren).

Der Einfluß des Figurentyps wird an folgendem kleinen Beispiel deutlich: Nehmen wir an, auf dem Feld E4 steht ein weißer Bauer. Dann sind für alle schwarzen Figuren (mit Ausnahme der Bauern) die Zugmöglichkeiten nach D5 und F5 wertlos, weil deren Ausführungen zum sofortigen Materialverlust durch einen Schlagzug mit dem weißen Bauern führen würden. Werden die Felder D5 und F5 dagegen nur von einer weißen Dame auf E4 beherrscht, können beliebige schwarze Steine dort postiert werden, vorausgesetzt, sie sind durch mindestens eine weitere schwarze Figur gedeckt. Dies gilt selbstverständlich nicht für den schwarzen König, der sich nicht dem Geschlagenwerden aussetzen darf. Es bietet sich also an, die Felderkontrollen, d.h., die Zielfelder der Zugmöglichkeiten in Abhängigkeit vom Typ der ziehenden Figur zu bewerten, wobei generell die Kontrollen umso höher einzuschätzen sind, je geringer der Wert der Figur (bezogen auf die Figurenwerte der Materialbalance) ist. Mit einem solchen Verfahren, das z.B. auch in CHESS 4.5 implementiert war, bekommt man ein recht gutes Maß für die wichtige Eigenschaft des von beiden Spielern jeweils *"beherrschten Raumes"*. Deshalb werden wir eine entsprechende Bewertung auch in unserem Programm vornehmen, obwohl gleich an dieser Stelle angemerkt werden muß, daß die Methode, die Zugmöglichkeiten feld- und figurabhängig zu bewerten, sehr zeitaufwendig ist. Auf die Art der Realisierung und einige praxisgerechte Modifikationen werden wir noch zu sprechen kommen.

Zuvor aber ein weiterer Aspekt der theoretischen Bedeutung der Felderkontrollen: Auch wenn wir eine differenzierte Bewertung der Zugmöglichkeiten eingeführt haben, so beschränkt sich die gewonnene Information durch die Summenbildung nach wie vor auf einen einzigen, allerdings im Vergleich zur Mobilität

wesentlich präziseren Wert. Es wäre zu wünschen, "Zwischenergebnisse" der Berechnung, die sich jeweils auf ein Feld beziehen, zur Verfügung zu haben, weil sich daraus noch genauere Aussagen über die realen Kraftverhältnisse ableiten ließen. Dies würde jedoch bedeuten, 2*64 Listen zu führen, die (getrennt für Weiß und Schwarz) alle Figuren enthalten, die das betreffende Feld kontrollieren. Dabei ist die besondere Gangart der Bauern zu beachten, wonach für das Vorrücken und das Schlagen unterschiedliche Richtungen festgelegt sind. Da wir uns bei den Felderkontrollen insbesondere für die gegenseitigen Beeinflussungen, die Vorherrschaft über bestimmte Felder oder das Schlagen von gegnerischem Material interessieren, sollen die Bauern nur hinsichtlich ihrer Schlagmöglichkeiten in den Listen berücksichtigt werden.

Der Nutzen solcher Listen ist unbestritten: Nicht nur die Bewertung der dynamischen Eigenschaften einer Stellung läßt sich durch sie verfeinern, sondern auch viele Untersuchungen, die im Ablauf des Such- und Entscheidungsprozesses anfallen, lassen sich so mit deutlich geringerem Aufwand durchführen. Beispielsweise gehört zu jeder Stellungsanalyse die Beantwortung der Frage "Steht der König im Schach?". Falls wir die Felderkontrollen nicht benutzen, bleibt uns nichts anderes übrig, als entweder alle gegnerischen Zugmöglichkeiten zu untersuchen oder ausgehend vom Königsstandort alle für ein Schachgebot in Frage kommenden Felder zu überprüfen. Beide Verfahren sind sehr zeitraubend. Im anderen Fall brauchen wir nur in der dem Standort des Königs und der gegnerischen Seite zugeordneten Liste nachsehen, ob dort ein Eintrag vorhanden ist. Auf die gleiche Art können wir feststellen, ob beliebige Figuren angegriffen sind oder ob sichere Fluchtfelder für eine angegriffene Dame existieren. Im letztgenannten Beispiel müssen allerdings mehrere Listen, nämlich die der Zielfelder für sämtliche Zugmöglichkeiten der Dame abgefragt werden. Diese Methode ist jedoch nicht uneingeschränkt verwendbar, da Kontrollen unter Umständen bedeutungslos sind. So kann die Dame durchaus auch auf ein von einer gegnerischen Figur kontrolliertes Feld ziehen, wenn jener Stein momentan gefesselt ist und damit am Schlagen gehindert wird. Insgesamt sehen wir also, daß die Suche durch die Benutzung der Kontrollen zwar keineswegs überflüssig wird

und wir die Züge im Normalfall immer ausprobieren müssen, daß wir aber häufig zumindest gute Vorinformationen (z.B. für die Zugsortierung) zur Verfügung haben.

- Aktivität

Unter diesem Begriff werden alle *taktischen Möglichkeiten* zusammengefaßt, die in der aktuellen Stellung gegeben sind. Zu diesen zählen sowohl Züge, die die Materialbalance verändern können (Schlagzüge, Bauernumwandlungen) als auch Angriffe auf den König (Schachgebote, insbesondere die besonders bedrohlichen "Abzugs-" und "Doppelschachs"). Diese aktiven Spielfortsetzungen geben zusammen mit dem beherrschten Raum, den Felderkontrollen also, ein Maß für die Initiative an und treffen damit eine Aussage darüber, wessen Pläne als vorrangig zu betrachten sind. Gute Abschätzungen für die Aktivität tragen daher wesentlich zu einer präzisen Bewertung der Stellung bei.

Leider ist eine entsprechende Bewertungsmethode gar nicht so einfach zu finden, da oft komplizierte Sachverhältnisse zu klären sind. Denn wie stark ist die Drohung des Materialverlustes einzuordnen, wenn eine Figur von mehreren gegnerischen Steinen angegriffen und gleichzeitig von mehreren eigenen verteidigt wird? Oder wie groß ist die Gefahr für einen König nach einem Schachgebot? Eigentlich müßten zur Beantwortung beider Fragen ganze Zugfolgen untersucht werden (Abtauschfolgen bzw. Schachgebote und deren Antwortzüge), um zu einem korrekten Urteil zu kommen. Dieses Vorgehen, die Züge tatsächlich in den Suchbaum aufzunehmen, entspricht dem Ruhesucheverfahren, das im Abschnitt "Suchen und Entscheiden" eingeführt wurde.

Es gibt aber auch einen Algorithmus, der die Erfolgsaussichten von Schlagzügen und - mit Modifikationen - sogar von Bauernumwandlungen mit gewissen Einschränkungen "vorhersagen" kann, ohne gleich den Suchbaum zu vergrößern. Er wurde 1961 in seiner ursprünglichen Form von M. Smith entwickelt, der ihn auch in seinem Schachprogramm namens SOMA benutzte. Da dieses Programm nur Suchen der Tiefe 1 ohne anschließende

Ruhesuche durchführte, war es besonders wichtig, die in den Endstellungen möglichen taktischen Fortsetzungen durch akzeptable Abschätzungen in der Bewertung zu berücksichtigen. Bevor wir jedoch die Arbeitsweise seines Verfahrens kennenlernen, müssen wir uns überlegen, was bei der Analyse einer Abtauschzugfolge zu beachten ist.

Nehmen wir an, Weiß hat die Möglichkeit, einen schwarzen Stein zu schlagen. Ferner setzen wir voraus, daß einerseits dieser Angriff durch weiße Figuren, die das Zielfeld des zu bewertenden Schlagzuges ebenfalls kontrollieren, unterstützt wird, und daß andererseits weitere schwarze Figuren das Angriffsobjekt verteidigen (oder "decken", wie man in der Schachsprache sagt). Ohne diese beiden Forderungen liefert der Algorithmus zwar auch das richtige Ergebnis; sein Einsatz ist dann allerdings kaum gerechtfertigt, weil die Bewertung dann sehr einfach vorzunehmen ist. Wird die angegriffene Figur nämlich nicht gedeckt, so ergibt sich der Erfolgswert für den Schlagzug aus dem Wert der geschlagenen Figur (*günstigster Fall für den Angreifer*; diesen Wert hat der Verteidiger sicher). Fehlt dagegen die Unterstützung durch weitere Steine des Angreifers und hat der Gegner Verteidigungszüge parat, so steht die Bewertung mit der Differenz der Figurenwerte von geschlagener und schlagender Figur fest, da Schwarz auf jeden Fall zurückschlagen wird (*ungünstigster Fall für den Angreifer*; diesen Wert hat der Angreifer sicher). Übertragen auf den Alpha-Beta-Algorithmus in der Minimaxsuche bedeutet dies eine Vorbesetzung von Alpha (günstigster Fall) und Beta (ungünstigster Fall), zwischen denen der Wert für die tatsächlichen Gewinnaussichten des Schlagzuges liegen muß, da beide Spieler zu jedem beliebigen Zeitpunkt der Abtauschzugfolge die Wahl haben, den momentanen Stand zu akzeptieren und auf weitere Schlagfortsetzungen zu verzichten, wenn dadurch (zusätzliches) Material verlorenginge, oder durch die Ausführung des nächsten Schlagzuges zu versuchen, den augenblicklichen Wert zu ihren Gunsten zu verschieben. Außerdem hängt die Bewertung des Schlagzuges sicherlich von der Anzahl der am Abtausch teilnehmenden Figuren beider Seiten sowie der Reihenfolge, in der sie gezogen werden, ab.

Die eben erwähnten Aspekte werden in Smith' Algorithmus berücksichtigt. Zu Beginn werden in diesem Verfahren alle Figuren von Weiß und Schwarz ermittelt, die sich an der Abtauschfolge beteiligen können; im übrigen eine relativ leicht zu erledigende Aufgabe, wenn man die Listen mit den Felderkontrollen zur Verfügung hat, da in ihnen zumindest ein Teil der benötigten Information gespeichert ist. Diese herausgesuchten Figuren werden anschließend *nach aufsteigenden Figurenwerten sortiert*, so daß jeweils die Figur mit dem niedrigsten Wert bei Schlagalternativen zunächst zum Einsatz kommt. Dadurch ist gewährleistet, daß der Materialverlust so gering wie möglich gehalten wird, falls der Gegner zurückschlagen kann. Die einzige Ausnahme von dieser Regel sind die sogenannten *"indirekten Kontrollen"*. Sie treten auf, wenn eine Figur durch eine andere hindurch Felder kontrolliert. Dies ist z.B. bei einer Dame auf D2 und einem Turm auf D1 der Fall. Alle Felder auf der D-Linie werden von Dame und Turm beherrscht, da sie auch von dem Turm betreten werden können, freilich erst, wenn die Dame ihren Platz geräumt hat. Bei solchen Figuraufstellungen muß immer zuerst die vordere Figur ins Gefecht geschickt werden, selbst wenn es die wertvollere ist. Die indirekt kontrollierenden Figuren stehen allerdings nicht in den Kontrollisten und müssen gesondert (und mit ziemlich hohem Aufwand) berechnet werden. Die Art, wie dies geschieht, soll uns hier nicht weiter interessieren.

Für die weitere Erläuterung des Verfahrens setzen wir voraus, daß m Figuren den gegnerischen Stein angreifen (direkt oder indirekt) und n Figuren diesen verteidigen. Dann führen wir folgende Bezeichnungen ein:

a(1),a(2),...,a(m) für die Werte der angreifenden Figuren,
v(1),v(2),...,v(n) für die Werte der verteidigenden Figuren

und

v(0) für den Wert der bedrohten Figur.

Wenn wir nun die Erfolgsaussichten eines Schlagzuges bewerten wollen, müssen wir alle denkbaren Ausgänge der Abtauschfolgen

(angegeben als resultierende Veränderung der Materialbalance) betrachten und unter Beachtung vor allem der Abbruchsmöglichkeiten das richtige Ergebnis aus einer Menge heraussuchen. Zu diesem Zweck bilden wir zwei Klassen. In der ersten stehen alle Ergebnisse für Abtauschfolgen, in denen der Angreifer zuletzt schlägt:

$$A(1)=v(0)$$
$$A(2)=v(0)-a(1)+v(1)$$
$$A(3)=v(0)-a(1)+v(1)-a(2)+v(2)$$
$$\vdots$$
$$A(x)=v(0)-a(1)+v(1)-+...-a(x-1)+v(x-1)$$

Die Anzahl x der Werte in dieser Klasse ist von der Anzahl der angreifenden und verteidigenden Figuren abhängig. Ist nämlich der Angreifer in der Überzahl (m größer n), so werden maximal der bedrohte Stein und die n verteidigenden Steine geschlagen (x=n+1). Kann dagegen der Verteidiger auf jeden Schlagzug seines Gegners reagieren (m<=n), so bricht die Abtauschfolge spätestens ab, wenn der letzte angreifende Stein bewegt wurde (x=m).

Die zweite Klasse enthält analog die Werte für alle Abtauschfolgen, in denen der Verteidiger zuletzt schlägt:

$$V(1)=v(0)-a(1)$$
$$V(2)=v(0)-a(1)+v(1)-a(2)$$
$$V(3)=v(0)-a(1)+v(1)-a(2)+v(2)-a(3)$$
$$\vdots$$
$$V(y)=v(0)-a(1)+v(1)-a(2)+-...+v(y-1)-a(y)$$

Für die Begrenzung y dieser Klasse gilt y=min(m,n). Die Anzahl der Werte richtet sich also nach dem "Durchhaltevermögen" beider Spieler.

Diese Resultate lassen sich zu einem Zwischenstand der Berechnung zusammenfassen:

$$A=\min(A(1),...,A(x))$$

und

$$V=\max(V(1),...,V(y)).$$

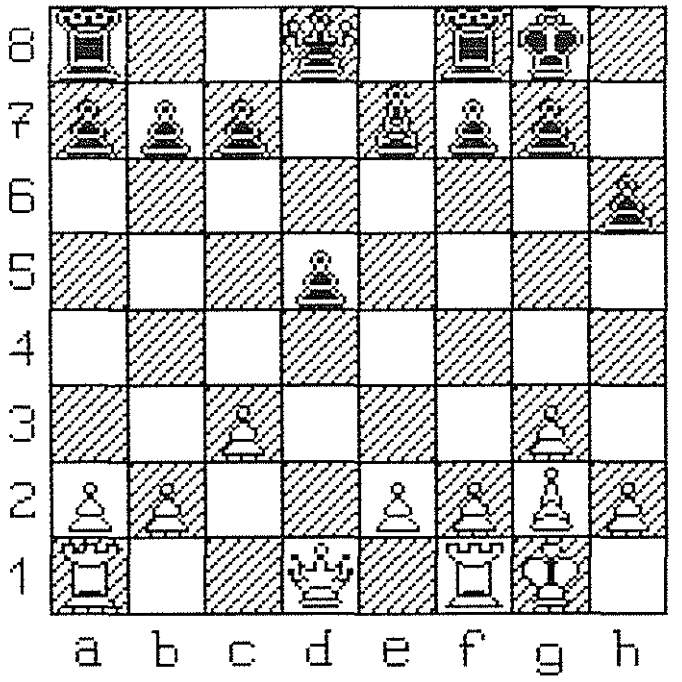
A ist die untere Schranke des Materialgewinns, wenn der Angreifer die Abtauschfolge so lange wie möglich fortsetzt. Entsprechend ist V die obere Schranke des Materialgewinns, wenn der Verteidiger den letzten Schlagzug ausführt.

Das Endergebnis kann jetzt ganz einfach errechnet werden:

$$S=\begin{cases}\min(A,V), & \text{falls } m<=n\\ \max(A,V), & \text{falls } m>n\end{cases}$$

Diese Formel gilt deswegen, weil sich der Spieler mit der größeren Anzahl einzusetzender Figuren aus A und V den für ihn günstigeren Wert aussuchen kann. Wenn z.B. der Angreifer in der Überzahl ist (m größer n), und es für ihn besser ist, einen Schlagzug nicht mehr auszuführen und den aktuellen Wert zu akzeptieren - dies ist für V größer A der Fall - so macht er durch Maximumbildung davon Gebrauch.

Diesen mathematischen Ansatz, der im Prinzip einer simulierten Minimaxsuche entspricht, wollen wir an einer Beispielstellung verdeutlichen:



Für diese Position ergeben sich folgende Berechnungen:

v(0)=100 (bedrohte Figur: schwarzer Bauer auf d5)
m=2 (Dame auf d1, Läufer auf g2)
n=1 (Dame auf d8)
a(1)=325,a(2)=900
v(1)=900

Die möglichen Schlagfolgen bewirken folgende Veränderungen der Materialbalance:

A(1)=v(0)=100
A(2)=v(0)-a(1)+v(1)=100-325+900=675

(Wegen m>n ist x=n+1=2.)

V(1)=v(0)-a(1)=100-325=-225

(Hier gilt y=min(m,n)=1.)

Daraus resultieren die Schrankenwerte

A=min(100,675)=100

und

V=max(-225)=-225.

Insgesamt ergibt sich wegen m>n:

S=max(100,-225)=100

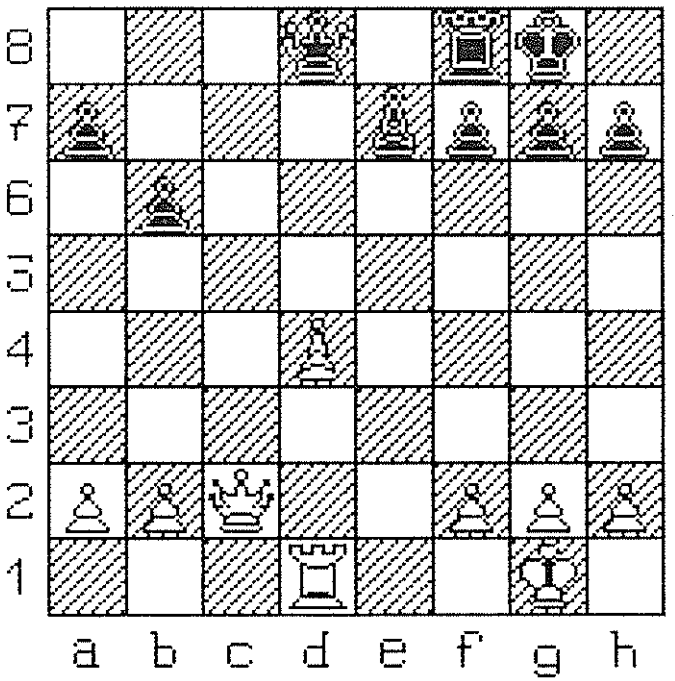
Weiß verbessert also durch das Schlagen auf d5 die Materialbalance zu seinen Gunsten um den Betrag "100". Dabei wird davon ausgegangen, daß Schwarz mit seiner Dame nicht zurückschlagen wird, um nicht weiteres Material zu verlieren (A(2)=675 fällt bei der Berechnung von A heraus).

Dieses Verfahren von Smith liefert (mit Einschränkungen, s.u.) korrekte Bewertungen für Schlagzüge, ist jedoch bei vielen angreifenden und verteidigenden Figuren ziemlich rechenzeitaufwendig, da alle Abtauschfolgen berücksichtigt werden müssen. Es ist daher in dieser Form in keinem Turnierschachprogramm realisiert worden.

Eine praxisnähere, schnellere Variante wird dagegen in dem Programm BOBBY, das von H.-J. Kraas und G. Schrüfer an der Universität in Braunschweig entwickelt wurde, benutzt. Der dort

implementierte Algorithmus zur Berechnung der sogenannten "Abtauschwerte" bricht die Analyse zum frühestmöglichen Zeitpunkt ab; hier handelt es sich um eine tatsächliche Simulation der Minimaxsuche auf statischem Wege. Dieses Ziel wird erreicht, indem jeder Spieler, der während der Abtauschfolge am Zug ist, sich entscheiden kann, ob er die Abtauschfolge fortsetzen will (falls er noch Schlagmöglichkeiten hat) oder ob vielleicht der *Wert vor seinem letzten Schlagzug günstiger war, als er jetzt durch seinen nächsten Schlagzug werden kann.* Neben dieser Benutzung des *"rückwirkenden Cutoffs"* wurde die ursprüngliche Methode auf die Behandlung von Bauernumwandlungen ausgedehnt. Zusätzlich konnte die Bewertung durch die Beachtung von Fesslungen präzisiert werden.

Trotzdem kann auch dieses Verfahren zu Fehlurteilen führen. Diese sind in der eventuell in einer Stellung vorhandenen gegenseitigen Beeinflussungen der Zugmöglichkeiten begründet. Was damit gemeint ist, läßt sich durch folgende Position veranschaulichen:



Betrachten wir den Schlagmöglichkeit Ld4xg7. Wegen der Deckung des Bauern durch den schwarzen König wird dieser Zug von dem Analyseverfahren mit "-225" bewertet. Dabei wird aber außer acht gelassen, daß durch eine Deblockierung jetzt der Turm auf d1 die schwarze Dame auf d8 angreift, so daß dort ein höherer Materialverlust für Schwarz droht, als ihm das Schlagen des Läufers einbringen würde. Der richtige Wert wäre hier also "+100". Auch an diesem Beispiel wird - wie schon bei

den Fluchtzügen von bedrohten Figuren - deutlich, daß die statische Bewertung der taktischen Möglichkeiten zwar sehr sinnvoll zur Steuerung der dynamischen Suche ist, diese aber nicht ersetzen kann.

Verfahren in unserem Programm

Auf den vorangegangenen Seiten wurden einige Verfahren (z.B. die Abtauschanalyse) und Hilfskonstruktionen (z.B. die Listen für die Felderkontrollen) vorgestellt, mittels derer die Schachprogrammierer versucht haben, die Probleme, die bei der Bewertung dynamischer Stellungsmerkmale vorhanden sind, in den Griff zu bekommen. Für uns stellt sich nun die Frage, was wir diesem umfangreichen Katalog entnehmen und in unser Programm einbauen können; denn wir werden uns bei dieser Auswahl beschränken müssen, weil wir im Vergleich zu einer großen Rechenanlage, wie sie beispielsweise für BOBBY zur Verfügung steht, wesentlich weniger Speicherplatz ausnutzen können und unser Rechner deutlich langsamer arbeitet.

Von diesen "Sparmaßnahmen" sind z.B. die Listen für die Felderkontrollen betroffen. Der erste Grund für diese Entscheidung ist der erhebliche Speicherbedarf, den sie verursachen; schließlich haben wir es mit 2x64=128 Listen zu tun, die jeweils maximal 16 Einträge enthalten können. Falls jeder Eintrag durch eine Integer-Variable repräsentiert würde (z.B. durch die Dimensionsangabe "dim fk%(2,64,16)"), hätten wir auf einen Schlag 4 KBytes weniger Hauptspeicher zur Verfügung. Es gibt zwar die Möglichkeit, die Einträge extrem dicht aneinanderzureihen - der dazugehörige Begriff heißt "Bitlisten" - , doch dann haben wir das Problem, daß wir gezielt einzelne Bits abfragen müssen, um die vorhandenen Felderkontrollen geliefert zu bekommen. Diese Zugriffsart wird in der Programmiersprache BASIC nicht direkt (durch einen entsprechenden Befehl) unterstützt, da über die Variablen immer mindestens 2 Bytes = 16 Bits adressiert werden. Der Umweg über das sogenannte "Maskieren" (mit Hilfe der logischen Operation "und") hat wiederum den Nachteil, zu rechenzeitaufwendig zu sein. Der Zeitaufwand ist sowieso schon wegen der notwendigen Aktualisierung bzw. Neuberechnung der Listen von Stellung zu Stellung

ein weiteres Handicap, das nur ausgeglichen werden kann, wenn viele Teile des Programmes von diesen Listen Gebrauch machen. Für unsrere Belange sind sie jedoch "eine Nummer zu groß".

Das gleiche gilt für das statisch arbeitende Verfahren zur Abschätzung der Gewinnaussichten von Schlagzügen. Auch dadurch wäre unser Programm aufgrund der relativ niedrigen Rechenleistung über Gebühr belastet. Darüber hinaus fehlt ihm eine wesentliche Voraussetzung, die in vielen Stellungen für exakte Bewertungen erforderlich ist: die Berücksichtigung von gefesselten Figuren.

Was bleibt, sind die Bewertungen für die Felderkontrollen, die wir, wie zuvor erläutert, figurtyp- und zielfeldabhängig wichten wollen. Zunächst ist zu klären, an welcher Stelle im Programm die erforderlichen Berechnungen zweckmäßigerweise durchgeführt werden sollten. Falls wir uns (aus Bequemlichkeit oder aus Gründen der Übersichtlichkeit) für ein Unterprogramm entscheiden, das von der Bewertungsfunktion aufgerufen wird, haben wir eine Menge Rechenzeit verschenkt. Weil die Felderkontrollen nämlich von den Zugmöglichkeiten abhängig sind, müßten wir dann im Prinzip bei deren Bewertung noch einmal die gleiche Arbeit wie der Zuggenerator leisten. Es bietet sich daher an, die Feststellung der Felderkontrollen und deren Bewertung parallel zur Bestimmung der Zugmöglichkeiten in den Unterprogrammen vorzunehmen, die vom Zuggenerator benutzt werden.

In diesem Fall ist allerdings die Steuerung des Informationsflusses etwas komplizierter; dieses (Neben-) Resultat des Zuggenerators muß ja irgendwie der Bewertungsfunktion zur Weiterverarbeitung zur Kenntnis gebracht werden. Dazu kommt das Problem, daß wir in jeder Stellung den Zuggenerator nur für eine Farbe aufrufen, nämlich für den am Zug befindlichen Spieler. Doch hier haben wir mit der Verwendung des Stacks, den wir eigentlich für die Rekursion des Suchverfahrens eingeführt haben, eine elegante Lösung parat; und die funktioniert so:

Jedesmal, wenn der Zuggenerator in Aktion tritt, um für eine Position die Zugmöglichkeiten zu ermitteln, wird auch für die

Felderkontrollen des betreffenden Spielers eine Bewertung vorgenommen, die als Ergebnis einen in Abhängigkeit von den Figurentypen und den Zielfeldern gewichteten Summenwert liefert, welcher in einer Integer-Variablen abgespeichert und auf diese Weise an die Bewertungsfunktion weitergegeben werden kann. Als "Übermittlungsträger" benutzen wir den *Stack*, in dem wir eine feste Eintragstelle (bezüglich des zweiten Index) für den Kontrollwert reservieren. Dies läßt sich für alle Stellungen im Suchbaum durchführen, weil selbst für die Endknoten noch der Zuggenerator aufgerufen werden muß, um illegale Stellungen herauszufinden (vgl. Beschreibung des Suchalgorithmus). Für die anderen Stellungen, die nicht auf der festgelegten maximalen Suchtiefe liegen und daher sowieso vertieft werden müssen, versteht sich der Aufruf des Zuggenerators von selbst.

Der Stack ist deswegen für diese Anwendung so nützlich, weil wir in ihm die Kontrollbewertungen aller Stellungen, die beim Vertiefen einer Zugfolge nacheinander erreicht werden, abspeichern können, ohne daß diese von jeweils neuen Werten überschrieben werden. Dazu brauchen wir nur den Eintrag an die der aktuellen Suchtiefe entsprechenden Stelle zu setzen. So hat die Bewertungsfunktion alle Abschätzungen für die Felderkontrollen - pro Stellung allerdings jeweils nur für eine Seite - von der Ausgangs- bis hin zur aktuellen Endposition der Variante zur Verfügung. Wenn wir nun die beiden letzten Eintragungen (je eine für Weiß und Schwarz) zur Bewertung heranziehen, erhalten wir ein hinreichend genaues Maß für den von beiden Seiten beherrschten Raum; und dies für einen Bruchteil der Rechenzeit, die wir benötigen würden, wenn wir für jeden Endknoten des Suchbaumes die Felderkontrollen extra durch ein dem Zuggenerator ähnliches Verfahren (mit ähnlich hohem Aufwand) bestimmen würden.

Soweit zur technischen Abwicklung, zur Informationsbeschaffung. Nun müssen wir noch die *Wichtung der Felderkontrollen* festlegen. Da wir bei der Bewertung zwei Kriterien zu berücksichtigen haben, muß für jedes kontrollierte Feld ein Produkt aus zwei Faktoren (einem für den Figurentyp und einem für das Zielfeld) in die Gesamtsumme aufgenommen werden. Für die

Höhe dieser Faktoren sind die im theoretischen Teil angestellten Überlegungen maßgebend.

Betrachten wir zunächst den *Einfluß des Figurentyps*. Wir hatten festgestellt, daß Kontrollen umso wertvoller sind, je geringer der Wert der kontrollierenden Figur ist. Dies ist jedoch für die nun durchzuführende Bewertung nicht die ganze Wahrheit. So hat z.B. der König einen gewissen Sonderstatus. Von ihm wird in der Eröffnung und während des Mittelspiels nicht verlangt, daß er möglichst viele Felder betreten kann, sondern er soll im Gegenteil in einer Ecke, abgeschirmt vor gegnerischen Attacken, auf seinen späteren Einsatz im Endspiel warten. Für die beiden ersten Partiestadien wäre also eine Bewertung seiner Zugmöglichkeiten im Sinne des beherrschten Raumes geradezu falsch! Nur im Endspiel, wenn ihm keine direkte Gefahr mehr droht, muß ein Maß seiner Aktivität berechnet werden. Doch hierfür gibt es auch andere Methoden (die wir in unserem Programm benutzen wollen); deshalb mehr zu diesem Thema erst im Abschnitt über die Königsbewertung.

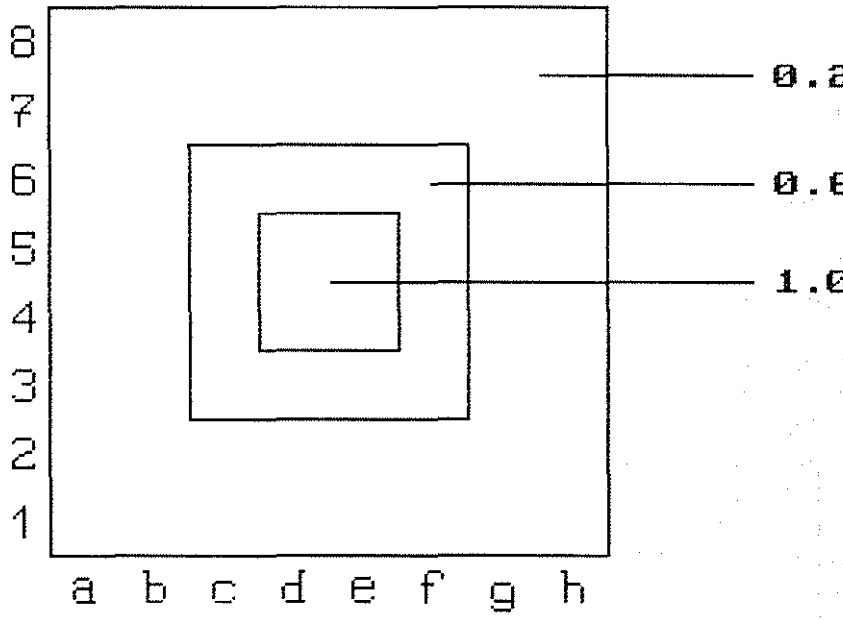
Die beiden weiteren Sonderfälle beziehen sich auf die Bauern und Springer. Aufgrund der für sie festgelegten Gangarten ist jeweils die Menge der von ihnen kontrollierten Felder ausschließlich von ihren aktuellen Standorten (und nicht von der Aufstellung der übrigen Figuren) abhängig. Deshalb kann für diese beiden Figurentypen die Bewertung ihrer Zugmöglichkeiten losgelöst vom Zuggenerator direkt in der Bewertungsfunktion durchgeführt werden. Die einzigen Informationen, die dabei benötigt werden, sind die augenblicklichen Positionen der betreffenden Figuren; das recht aufwendige Aufsummieren über die kontrollierten Felder entfällt. So wird eine Menge Rechenzeit gespart. Welche Bewertungsmethoden wir hierfür in unserem Programm anwenden werden, wird in den folgenden Abschnitten über die Bauern- bzw. die Figurenbewertung beschrieben.

Für die restlichen Typen Dame, Turm und Läufer, die im Englischen treffend als "sliding pieces" (zu deutsch: gleitende oder rutschende Figuren) bezeichnet werden, bleibt uns in der Tat nichts anderes übrig, als parallel zur Bestimmung der Zugmöglichkeiten die Bewertung der einzelnen kontrollierten

Felder vorzunehmen. Nur so lassen sich Behinderungen durch eigene und gegnerische Figuren erkennen, die die Aktionsradien von Figuren diesen Typs stark beeinflussen können. An dieser Stelle werden wir auch die grundlegende Abhängigkeit zwischen Figurenwert und Nutzen der Felderkontrollen berücksichtigen; und zwar tun wir dies durch die Verwendung von Wichtungskoeffizienten im Verhältnis 1:3:4 für Dame, Turm und Läufer. Dadurch erreichen wir gleichzeitig, daß der beherrschte Raum der Dame, die im allgemeinen ungefähr doppelt so viele Felder wie Turm oder Läufer kontrolliert, nicht überbewertet wird.

Nach der figurentypabhängigen Wichtung müssen wir nun noch die Bedeutung der verschiedenen Brettregionen abschätzen. Dabei wollen wir uns auf den besonderen *Einfluß des Zentrums* beschränken, weil einerseits eine Einbeziehung der Königsstandorte unsere relativ bescheidenen Rechenzeit- und Speicherplatzkapazitäten sprengen würde und andererseits die Bearbeitung einzelner wichtiger Felder im Rahmen des Zuggenerators praktisch unmöglich ist. Häufig werden diese Felder auch erst im Verlauf der Stellungsbewertung oder sogar zu einem späteren Zeitpunkt der Suche erkannt. Dies betrifft z.B. die Stopfelder von Freibauern, die von der Bauernbewertung analysiert werden, oder die Standorte von gefesselten Figuren, da Fesslungen erst nach dem Vertiefen eines illegalen Zuges in der Nachfolgerstellung ermittelt werden.

Die zentrumsbezogene Wichtung der Kontrollen läßt sich am einfachsten durch ein Feld (im Programm: dim ko%(118); die Obergrenze 118 orientiert sich an der rechnerinternen Brettdarstellung) realisieren, daß beim Programmstart mit Konstanten nach folgenden Schema belegt wird:



Der Zuggenerator greift bei der Produktbildung zur Bewertung der Kontrollen jeweils auf das Element in ko% zu, das über den Index dem entsprechenden Zielfeld der gerade berechneten Zugmöglichkeit zugeordnet ist. Für die Kontrolle des Feldes C4 durch einen Turm ergibt sich beispielsweise der Wert 3*ko%(65) = 3*0.6 = 1.8. Auf diese Art werden vom Zuggenerator alle Felderkontrollen behandelt. Die Einzelergebnisse werden aufsummiert und über den Stack der Bewertungsfunktion zur Verfügung gestellt.

Die Bewertung der dynamischen Eigenschaften wird sich im allgemeinen von Position zu Position relativ stark verändern. Dies ist z.B. bei der Ausführung eines Entwicklungszuges, durch den eine Figur von der Grundreihe auf ein zentral gelegenes Feld gebracht wird, der Fall, da hierdurch die Anzahl der Felderkontrollen sprunghaft steigt. Die im folgenden aufgeführten statischen Stellungsmerkmale sind dagegen kaum so kurzfristigen Bewertungsschwankungen unterworfen.

Die Bauernbewertung

Dies trifft insbesondere auf die als nächstes zu besprechende Bewertung der Bauern zu, die ihre Standorte nur in kleinen Schritten wechseln können und deren Formationen daher relativ dauerhafte Eigenschaften aufweisen. Aus diesem Grund sind die Stärken und Schwächen von Bauern ein wesentlicher Faktor bei der Beurteilung einer Position und beeinflussen häufig den weiteren Spielverlauf. So sind die Spieler meist darum bemüht, den Angriff gegen Schwachpunkte in der Bauernaufstellung, auch *Bauernstruktur* genannt, zu richten. Auf der anderen Seite bieten starke Bauernketten z.B. dem König Schtz vor feindlichen Attacken und sind imstande, durch energisches Vorrücken Lücken in die gegnerische Verteidigung zu reißen.

Shannons theoretischer Ansatz beinhaltete drei typische Schwächen, die in die Bauernbewertung aufgenommen werden sollten. Es sind dies:

- *Doppelbauern*

Damit werden zwei Bauern, die auf derselben Linie stehen, bezeichnet. Diese Konfiguration kann nur durch das Schlagen mit einem Bauern entstehen. Gewöhnlich ist ihr Vorrücken behindert; das Blockieren des vorderen Bauern reicht aus, um beide "lahmzulegen". Außerdem sind sie oft schwerer gegen Angriffe zu decken, als wenn sie nebeneinander stehen und sich gegenseitig Schutz bieten können.

- *isolierte Bauern*

Das sind Bauern, denen auf den benachbarten Linien (bei Randbauern auf der A- oder H-Linie nur auf der einen benachbarten Linie) Bauern der eigenen Farbe fehlen. Auch sie sind wegen der schlechten Deckungsmöglichkeiten häufig das Ziel von Figurenangriffen.

- *rückständige Bauern*

Der Begriff dieser Bauernschwäche wird oft voneinander abweichend definiert. Um für unsere Zwecke den Sachverhalt des rückständigen Bauern festzulegen, folgt hier ein Auszug aus dem Buch von H. Kmoch, der 1956 den Versuch unternahm, eine einheitliche Sprechweise für die Analyse von Bauernformationen einzuführen (KMO):

> "Ein Kandidat mit endgültig geschwächtem Stopp wird rückständiger Bauer genannt. Wir bevorzugen die Bezeichnung Hinker, um die Verbauerung der Ausdrucksweise einzuschränken. Die Schwächung des Stoppfeldes drückt sich einerseits in der endgültig verlorenen Bauerndeckung, andererseits in der Beherrschung durch einen Wächter aus."

und

> "Steht der rückständige Bauer auf einer geschlossenen Linie, so nennen wir ihn getarnter Hinker."

Etwas verständlicher ausgedrückt, besitzt der rückständige Bauer folgende Eigenschaften:

1. Er steht auf einer halboffenen Linie, d.h., auf dieser Linie befinden sich keine gegnerischen Bauern. Fehlt die Eigenschaft 1, so ist der Bauer ein getarnter oder unechter rückständiger Bauer.

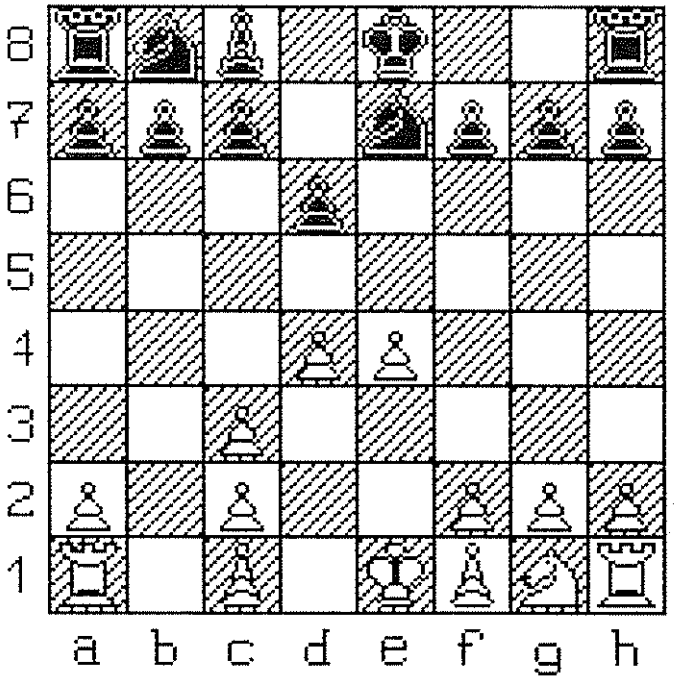
2. Er ist nicht durch eigene Bauern gedeckt und kann auch nicht mehr durch Vorrücken eigener Bauern gedeckt werden.

3. Das vor ihm liegende Feld wird nicht von eigenen Bauern kontrolliert.

4. Er kann nicht vorgerückt werden, ohne gegnerischen Bauernangriffen ausgesetzt zu sein.

Rückständige Bauern sind aus den gleichen Gründen wie isolierte Bauern Schwachstellen in der Bauernstruktur.

Die Bewertung dieser nachteiligen Bauernkonfigurationen stellte sich Shannon recht einfach vor. Für jede festgestellte Schwäche sollte ein konstanter Punktabzug in der Höhe eines halben Bauern vorgenommen werden. Der Begriff *"Punktabzug"* ist dabei jeweils aus der Sicht des betroffenen Spielers zu sehen, d.h., bei Schwachstellen in der Bauernstruktur von Schwarz Abzug negativer Werte und somit Erhöhung der Gesamtbewertung. Dies gilt im übrigen auch für alle folgenden Stellungsmerkmale, in denen ein Punktabzug vorgenommen werden soll. Diese Methode, die Bauernschwächen einzeln und unabhängig voneinander durch jeweils konstante Verringerung (Erhöhung) des Gesamtwertes einzuschätzen, ist später auch in den meisten Schachprogrammen benutzt worden. Allerdings sind die Punktabzüge dort betragsmäßig wesentlich geringer, um Fehlbewertungen der folgenden Art zu vermeiden:



Bei der Analyse dieser Stellung werden in der Bauernformation von Weiß zwei Schwächen entdeckt: ein isolierter Bauer auf A2 und Doppelbauern auf der C-Linie. Auf der anderen Seite hat Schwarz einen Bauern weniger. In der Summe von Material- und Bauernbewertung wird nach Shannons Ansatz diese Position als ausgeglichen angesehen. Das stimmt aber mit der (schachtheoretischen) Realität nicht überein. Nicht nur wegen des Mehrbauern sondern auch aufgrund der besseren Kontrolle

des Zentrums, die nicht zuletzt auf den Doppelbauern C2/C3 beruht und Weiß den freieren Aufbau seiner Figuren sichert, steht Weiß deutlich überlegen.

Obwohl die Arbeit Shannons nur theoretische Überlegungen waren, und er daher nie die Gelegenheit hatte, die Spielstärke anhand einiger vom Computer gespielter Partien zu überprüfen, so wußte er doch um die Schwachstellen in seiner Bewertung und gab im Anhang seiner Artikel weitere Stellungsmerkmale an, um die positionelle Abschätzung der Stellung zu präzisieren, ohne diese Eigenschaften allerdings numerisch einzuordnen.

Zu diesen gehört zum einen die gerade erwähnte *Zentrumskontrolle durch Bauern*. Die zweite Erweiterung betrifft die Bewertung von *Freibauern*. Wie schon erläutert, zeichnet sich ein Freibauer dadurch aus, daß er nicht mehr von gegnerischen Bauern durch Blockieren oder Geschlagenwerden an der Umwandlung gehindert werden kann. Aus diesem Grund sind höherwertige Figuren des Gegners an die Aufgabe gebunden, den Freibauern von der Grundlinie ferzuhalten, und können nur noch bedingt andere Aufgaben wahrnehmen. Das bedeutet natürlich eine wesentliche Schwächung der gesamten Kampfkraft des Gegners und sollte daher gebührend in der Bewertung berücksichtigt werden.

Die drei wichtigsten Faktoren für eine gute Abschätzung des aus einem Freibauern resultierenden positionellen Vorteils haben wir schon kennengelernt: den Standort des Freibauern (oder andersherum: den Abstand vom Umwandlungsfeld) und die Möglichkeiten, den weiteren Vormarsch des Bauern durch eigene Figuren zu unterstützen bzw. durch gegnerische Figuren aufzuhalten. Viele Schachprogramme vergeben allerdings nur abhängig vom Standort des Bauern einen Bonus und verzichten auf eine Bewertung der Umwandlungschancen, weil sie halbwegs präzise nur mit relativ großem Aufwand vorgenommen werden kann. Nebenbei bemerkt: Dieser Aufwand ließe sich in Grenzen halten, wenn die im vorigen Abschnitt beschriebenen Listen der Felderkontrollen benutzt werden könnten.) Trotzdem werden wir zumindest in einem vereinfachten Ansatz versuchen, den Einfluß der anderen Figuren abzuschätzen. Ein Anhaltspunkt für

diesen Einfluß ergibt sich außerdem aus den beiderseitigen Materialstärken. Damit läßt sich die Tatsache berücksichtigen, daß in Endspielstellungen die Freibauern eine größere Bedeutung als im Mittelspiel haben, weil dem Gegner weniger Figuren zu seiner Bekämpfung zur Verfügung stehen. Es ist daher ein praktikabler Weg, beim Eintritt ins Endspielstadium die Bewertung für Freibauern zu erhöhen. Darüber hinaus bieten sich auch für Positionen mit stark reduziertem Material, insbesondere wenn nur noch Könige und Bauern auf dem Brett sind, spezielle Verfahren an, die zu einer recht genauen Freibauernbewertung führen können (mehr hierzu im Abschnitt über die Königsbewertung).

Eine wichtige Eigenschaft, die in Shannons Artikeln fehlt, ist das *Vorrücken der Bauern*. Ungefähr zur gleichen Zeit wie Shannon, also Anfang der Fünfziger Jahre, entwickelte der englische Wissenschaftler A. Turing ein "Schachprogramm", das zwar nie auf eine Rechenanlage gebracht werden konnte, von dem aber eine Partie gegen einen Menschen bekannt ist, in der Turing die Berechnungen der Maschine (mühsam) per Hand simulierte. Der Partieverlauf ist in LEV bzw. NEW nachzulesen. In diesem Programm war eine Bewertung für das Vorrücken der Bauern enthalten. Allerdings wurde kein Unterschied zwischen den einzelnen Linien gemacht. Das wirkte sich dann in der Partie so aus, daß es häufig zu Zügen mit den Randbauern kam, weil der Gegner diese Züge eher als solche mit den Mittelbauern zuließ.

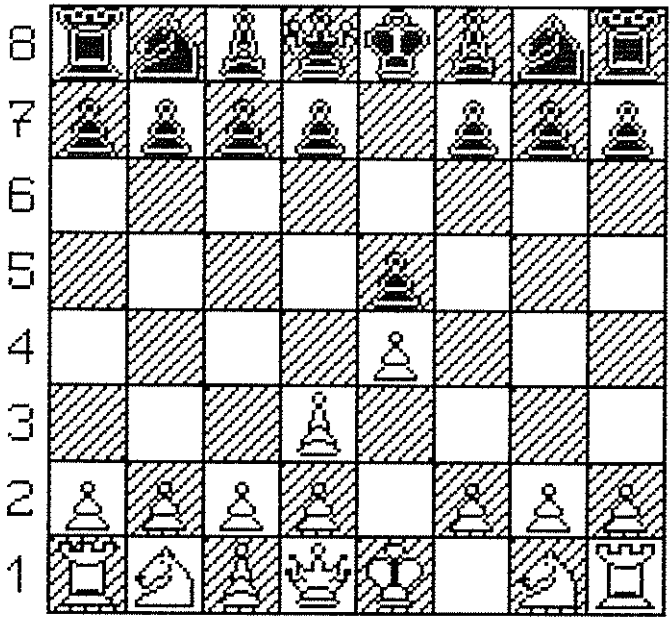
Aus einem solchen Spielverhalten resultieren meistens positionelle Nachteile: ein stärkerer Einfluß des Gegners im Zentrum und damit die besseren Entwicklungsmöglichkeiten für seine Figuren sowie nach der Ausführung der Rochade ein fehlender Bauernschutz vor feindlichen Angriffen auf den König.

Dem können wir entgegenwirken, indem wir während der Eröffnung und des Mittelspiels den Bonus für das Vorrücken in Abhängigkeit von den Linien vergeben. Aus der Sicht von Weiß könnte das z.B. so aussehen: hohe positive Werte für D- und D-Bauern, etwas niedrigere positive Werte für C- und F-Bauern sowie der Wert "0" oder sogar negative Werte für A-, B-, G-

und H-Bauern. Durch die Gleichbehandlung der Bauern auf beiden Seiten, dem Damenflügel (A-, B- und C-Linie) und dem Königsflügel (F-, G- und H-Linie), ist es möglich, die Schutzfunktion der Bauern sowohl im Fall einer langen als auch bei einer kurzen Rochade zu gewährleisten.

Dieser Ansatz findet sich z.B. in der Bewertungsfunktion von CHESS 4.5. Noch einen Schritt weiter geht das Schachprogramm BOBBY. Dort ist es möglich, nach Ausführung der Rochade die Werte für die Bauern, die nicht dem König zum Schutz dienen müssen, zu erhöhen. Da dies zudem in Abhängigkeit von dem Standort des gegnerischen Königs geschieht und im Fall entgegengesetzter Rochaden sehr hohe Werte in Anrechnung gebracht werden, werden diese Bauern praktisch zum Sturm auf die feindliche Königsstellung veranlaßt. Im Endspiel sind dagegen solche Überlegungen hinfällig; dann bietet es sich an, alle Linien gleich zu behandeln, weil andere Gesichtspunkte, z.B. die Bildung von Freibauern, in den Vordergrund treten. Zusammenfassend läßt sich sagen, daß die linienbezogene Bewertung für das Vorrücken der Bauern einerseits einen wichtigen Beitrag zur strategischen Planung liefert und andererseits als Ersatz für die fehlende Bewertung der Felderkontrollen der Bauern dienen kann (s.o.).

Zum Abschluß der Betrachtung der Stellungsmerkmale in der Bauernstruktur kommen wir zu einer eröffnungsspezifischen Bewertung. Insbesondere wenn ein Schachprogramm keine Eröffnungsbibliothek zur Verfügung hat, sind Positionen der folgenden Art nicht unüblich:



Da in vielen Bewertungsfunktionen die Entwicklung einer Figur zumindest kurzfristig höhere Zugewinne in der Bewertung verspricht als das Ziehen von Bauern (hier z.B. d2-d4), wird ein *Zentrumsbauer blockiert*. Das hat zur Folge, daß sich im weiteren Spielverlauf Weiß mit der Entwicklung seiner Figuren (im Beispiel vor allem des Läufers auf C1) schwer tun wird. Häufig wird auch ein nochmaliges Ziehen des Läufers auf D3 erforderlich. Um dies zu vermeiden, sollte in der Bauernbewertung für

solche Aufstellungen ein entsprechender Punktabzug vorgenommen werden.

Verfahren in unserem Programm

Wir haben inzwischen viel über die Vor-und Nachteile der unterschiedlichen Bewertungsmethoden für die Bauernstruktur erfahren und wollen nun dieses Wissen für unser Schachprogramm ausnutzen. Folgende Stellungsmerkmale werden wir berücksichtigen (Angabe der zugehörigen Bewertung aus der Sicht von Weiß):

1. Blockierung der Zentrumsbauern
   konstanter Punktabzug (-10)

2. isolierte Bauern
   konstanter Punktabzug (-15)

3. Mehrfachbauern
   Punktabzug abhängig von der Anzahl Bauern auf der betrachteten Linie (-4 für Doppelbauern; -40(!) für 3 und mehr Bauern, weil dann meistens mindestens ein Bauer verlorengeht)

4. Freibauern
   Pluspunkte nach der Formel

   Wert= Endspielfaktor * Umwandlungsfaktor * Standort * Standort

   Dazu im einzelnen:
   Der Endspielfaktor wird für Eröffnungs- und Mittelspielpositionen auf 1, im Endspiel auf 2 gesetzt.

Im Umwandlungsfaktor sind enthalten:

- Grundwert zur Wichtung (+1.15)
- Unterstützung durch Bauern auf Nachbarlinien (+0.35)
- Blockade durch gegnerische Figur (-0.35).

- Bezüglich des Standortes interessiert nur, wie weit der Freibauer schon vorgerückt ist, und nicht, auf welcher Linie er steht. Dementsprechend wird nur die Reihenangabe zur Bewertung herangezogen. Diese fließt wegen ihrer großen Bedeutung für die Umwandlungschancen quadratisch in die Berechnung ein.

Betrachten wir als Beispiel noch einmal die Stellung auf Seite 203. Die Anwendung der Formel ergibt für den Freibauern auf D4:

Wert=2*(1.15+0.35-0.35)*4*4=2*1.15*16=36.8.

Ausschlaggebend für diesen Betrag sind das Erreichen des Endspielstadiums, die Deckung durch den weißen Bauern auf C3, die Blockade durch den schwarzen Springer auf D5 und der Standort auf der 4. Reihe.

5. Vorrücken der Bauern
Pluspunkte für jedes Vorwärtsziehen um eine Reihe in Abhängigkeit vom Partiestadium und den Linien nach dem Schema

Eröffnung/Mittelspiel - (0.0, 0.0, 3.9, 5.4, 7.0, 2.3, 0.0, 0.0)
Endspiel - (2.0,...,2.0).

Dabei ist jeweils der 1. Wert der A-Linie, der 2. Wert der B-Linie usw. zugeordnet.

6. rückständige Bauern
Wir werden für unsere Bewertung einen etwas abweichenden Begriff des rückständigen Bauern verwenden, um die Feststellung dieses Sachverhaltes etwas zu vereinfachen und damit die Bewertung zu beschleunigen. Zum einen machen wir keinen Unterschied zwischen echten und getarnten rückständigen

Bauern. Desweiteren werden wir einen Bauern genau dann als rückständig ansehen, wenn er nicht durch eigene Bauern gedeckt ist und außerdem das vor ihm liegende Feld von mindestens einem gegnerischen und keinem eigenen Bauern kontrolliert wird. Die Anzahl der gegnerischen Bauernkontrollen bestimmt dann die Höhe des Punktabzuges (-2 für eine, -4 für zwei Kontrollen).

Die Figurenbewertung

Über die Figurentypen Springer, Läufer, Turm und Dame gibt es keine so ausgearbeitete Schachtheorie wie für die Bauern. Während es über die "Kunst der Bauernführung" - so auch der Titel des oben zitierten Buches von H. Kmoch - komplette Werke von unterschiedlichen Autoren gibt, beschränken sich die Ausführungen zu den jetzt zu behandelnden Figurentypen auf einige (heuristische) Grundsätze, deren Bedeutung zumeist an Beispielstellungen demonstriert wird. Das Fehlen eines relativ festen "Regelwerkes" ist durch die vielfältigen Möglichkeiten des Zusammenspiels dieser Figuren, auch in Relation zur Bauernstruktur, begründet und läßt ein umfangreiches starres Bewertungsschema nicht zu. Dementsprechend gering (im Vergleich zur Bauern- oder Königsbewertung) ist in den existierenden Schachprogrammen die Anzahl der für diese Figurentypen berücksichtigten Stellungsmerkmale.

Dafür hat sich bei der Implementierung dieser Kriterien quasi ein "Standard" gebildet; es sind häufig nahezu die gleichen Einzelbewertungen, die von den Programmen vorgenommen werden. An diesen Realisierungen wollen wir uns für unser Konzept zum großen Teil orientieren, so daß wir im folgenden die Beschreibung der theoretischen Ansätze und die Implementierung in unserem Programm zusammenfassen können. Dabei gehen wir so vor, daß wir die zu bewertenden Eigenschaften nach Figurentypen sortiert betrachten und die numerische Einordnung der Stellungsmerkmale wiederum aus der Sicht von Weiß angeben.

Springer

Bei unseren Überlegungen zum Thema "Felderkontrollen" haben wir festgestellt, daß der Aktionsradius des Springers allein von seinem Standort und nicht von anderen Figuren abhängt. Damit steht es im Gegensatz zu Läufern, Türmen und Damen. Gemeinsam ist allen diesen Typen (und auch den Bauern) dagegen, daß sie die Beherrschung des Zentrums anstreben sollen. Bei den "sliding pieces" wurde eine entsprechende Bewertung durch das Konstanten-Feld ko% erreicht. Für den Springer müssen wir für eine analoge Abschätzung den *Abstand zwischen Standort und Zentrum* messen. Dies geschieht genauso wie die Berechnung der Distanz Königsstandort - geometrischer Brettmittelpunkt (5.5,6.5) innerhalb der Matt-Routine. Da die Felder c3, f3, c6 und f6 (jeweils mit dem Zentrumsabstand 3) die nach der Eröffnung am häufigsten eingenommenen Standorte sind, wollen wir die Abstandsbewertung für diese Felder auf den Wert "0" normieren und für größere bzw. kleinere Abstände negative bzw. positive Werte vorsehen. Daraus ergibt sich unsere Berechnungsvorschrift:

Abstandswert=0.6*(6-2*Zentrumsabstand)

Die zweite Eigenschaft, die wir in der Springerbewertung berücksichtigen, ist seine *Entwicklung*. Demnach werden wir jeden auf der eigenen Grundreihe verharrenden Springer mit dem konstanten Punktabzug (-4.7) bestrafen, weil er nicht nur selbst passiv steht - dies wird schon von der Zentrumsbewertung erfaßt - , sondern weil er auch die schnelle Entwicklung der anderen Figuren (z.B. die Ausführung der Rochade) behindert.

Läufer

Der Wirkungsbereich der Läufer ist schon grundsätzlich beschränkt, da sie nur die Felder einer Farbe betreten können. Es ist daher (speziell bei nur einem vorhanden Läufer) eine gute Strategie, die restlichen Figuren auf die Felder der anderen Farbe zu stellen. Dies gilt insbesondere für die nur schwer beweglichen Bauern. Shannon hatte eine entsprechende Läufer/Bauern-Bewertung in seine Aufzählung der zu imple-

mentierenden Stellungsmerkmale aufgenommen. In unserem Programm werden wir es jedoch diesbezüglich bei der Bewertung der Felderkontrollen bewenden lassen, die diese Abschätzung des Aktionsradius implizit miterledigt. Berücksichtigen wollen wir dagegen den Besitz des *Läuferpaares*, weil zwei Läufer sich vor allem in offenen Stellungen gut in ihrer Wirkung ergänzen. Dies geschieht durch einen Bonus von 4 Punkten. Komplettiert wird die Läuferbewertung durch die Bestrafung eines *Entwicklungsrückstandes* (analog zu den Springern: -5.5 pro Läufer auf der Grundreihe).

Turm

Für diesen Figurentyp sollen einige vorteilhafte Aufstellungen bewertet werden:

1. *offene/halboffene Linien*

   Offene bzw. halboffene Linien sind dadurch gekennzeichnet, daß auf ihnen keine bzw. keine eigenen Bauern stehen. Über offene Linien ist häufig ein Eindringen der Türme in das gegnerische Lager, verbunden mit einer ernsthaften Bedrohung des Königs, möglich. Auf halboffenen Linien können gegnerische Bauern, insbesondere wenn diese rückständig sind, unter Beschuß genommen werden. Der Bonus für die Besetzung dieser Linien beträgt für offene Linien 4 und für halboffene Linien 1.5 Punkte.

2. *Verdopplung*

   Falls sich zwei Türme der gleichen Farbe auf derselben Linie befinden (z.B. auf d1 und d2), wirkt deren geballte Kraft so stark, daß sich auf dieser Linie kaum noch etwas ohne das "Einverständnis" der betreffenden Seite abspielen kann. Vor allem das Einbrechen in die gegnerischen Reihen wird auf diese Art optimal vorbereitet. Die Bewertung in unserem Programm trägt dem durch 8 Punkte Rechnung.

3. *Turm auf der 7. Reihe*

Hat ein Turm tatsächlich das Eindringen geschafft, so wird dies mit 5 Punkten belohnt. Wenn außerdem der gegnerische König noch auf seiner Grundreihe steht, ist dies zusätzliche 6 Punkte wert, da im Mittelspiel dadurch die Mattgefahr relativ groß ist und im Endspiel der König so gehindert wird, aktiv am Spiel teilzunehmen.

Diese drei bewerteten Eigenschaften können zusammengefaßt als Versuch betrachtet werden, dem Programm ein wenig langfristiges Planen zu ermöglichen. Wie man an der Reihenfolge der Stellungsmerkmale sieht, ist es hier, wie fast immer, erforderlich, zunächst das Erreichen von strategischen Teilzielen anzustreben, ehe nach und nach ein bestimmter Vorteil gewonnen werden kann.

Das letzte Kriterium der Turmbewertung ist vor allem in Endspielstellungen von großer Bedeutung. Es handelt sich um die *Beziehung zwischen Türmen und Freibauern.* Aufgrund ihrer Gangart sind die Türme bestens dazu geeignet, eigene Freibauern beim Vorrücken zu unterstützen bzw. gegnerische Freibauern zu behindern, sobald sie auf denselben Linien wie diese Bauern aufgestellt werden. Der große Nutzen ergibt sich sowohl durch die Deckung des eigenen bzw. den direkten Angriff des gegnerischen Bauern als auch durch die Kontrolle des Stopfeldes. Gerade in Turmendspielen, d.h., es sind nur noch die Könige sowie Türme und Bauern auf dem Brett, sind beide Spieler nach der Bildung von Freibauern um solche Aufstellungen bemüht, die häufig sogar spielentscheidend sein können. In unserem Programm werden wir diesen Vorteil mit 10 Punkten in Mittelspiel- und mit 15 Punkten in Endspielpositionen bewerten.

Dame

Die Dame ist eine Figur, die mit Vorsicht eingesetzt werden muß. Da sie von allen Figuren (mit Ausnahme des "unbezahlbaren" Königs) den höchsten Wert hat, muß sie feind-

lichen Attacken fast immer ausweichen. Sie sollte daher eigentlich nur auf solchen Feldern aufgestellt werden, die sicher kontrolliert werden. Diese Kontrollen müssen zu Beginn der Partie erst erkämpft werden; dies ist die vordringliche Aufgabe der Bauern und der Leichtfiguren in der Eröffnungsphase. Erst wenn die Territorien abgesteckt sind, kann sich auch die Dame am Spiel beteiligen.

Für Programme, die diese Forderung nicht beachten, besteht die Gefahr, daß die *zu früh aktivierte Dame* andauernd von gegnerischen Figuren angerempelt wird. Die Folge ist, daß die Dame ein um's andere Mal gezogen werden muß und der betreffende Spieler in der Entwicklung zurückbleibt. Aus diesem Grund sollten Damenzüge, die zu solchen Partieverläufen führen können, durch Punktabzüge bestraft werden.

Trotz dieser an sich einleuchtenden Überlegungen haben nur wenige Schachprogramme dieses Merkmal in ihrer Bewertungsfunktion realisiert. Zu diesen gehört z.B. OSTRICH, das Programm des amerikanischen Wissenschaftlers M. Newborn, in dem grundsätzlich Züge mit der Dame, die vor dem 9. Zug der Partie vorkommen, mit Minuspunkten belegt werden. Ein etwas genaueres Maß wird in dem Programm BOBBY benutzt und soll hier übernommen werden. Dabei geht man von der Anzahl noch nicht entwickelter Leichtfiguren und dem Standort der Dame aus: Falls nämlich die Dame die 2. Reihe überschreitet und sich noch Springer oder Läufer auf der Grundreihe befinden, wird ein Punktabzug vorgenommen, dessen Höhe durch den Ausdruck

-2.5 * Anzahl nicht entwickelter Leichtfiguren

berechnet wird.

Die Besprechung der Stellungsmerkmale für Springer, Läufer, Türme und Damen ist damit abgeschlossen, und wir können uns der Königsbewertung, einem der wichtigsten Gebiete bei der Programmierung der Bewertungsfunktion, zuwenden.

Die Königsbewertung

Die bedeutsame Rolle des Königs im Schachspiel ist unbestritten; ist doch das Mattsetzen des gegnerischen Königs das eigentliche Ziel der gesamten Partie. Doch der König ist nicht nur ein Angriffsobjekt. Speziell im Endspiel wird er zu einer aktiven Figur. In Bauernendspielen ist er gar die einzige Figur, die eigene Freibauern unterstützen und feindliche bekämpfen kann. Sein Wirkungsbereich kann dann genauso spielentscheidend sein wie in der Eröffnung und im Mittelspiel seine Verwundbarkeit. Es gilt daher, möglichst genaue Bewertungen seiner Sicherheit bzw. seiner Aktivität (je nach Partiestadium) vorzunehmen. Von großer Bedeutung ist in diesem Zusammenhang auch die Entscheidung, ob eine Position schon als Endspielstellung betrachtet werden kann, damit der König nicht bei falscher Einschätzung der Gegebenheiten "ins offene Messer rennt".

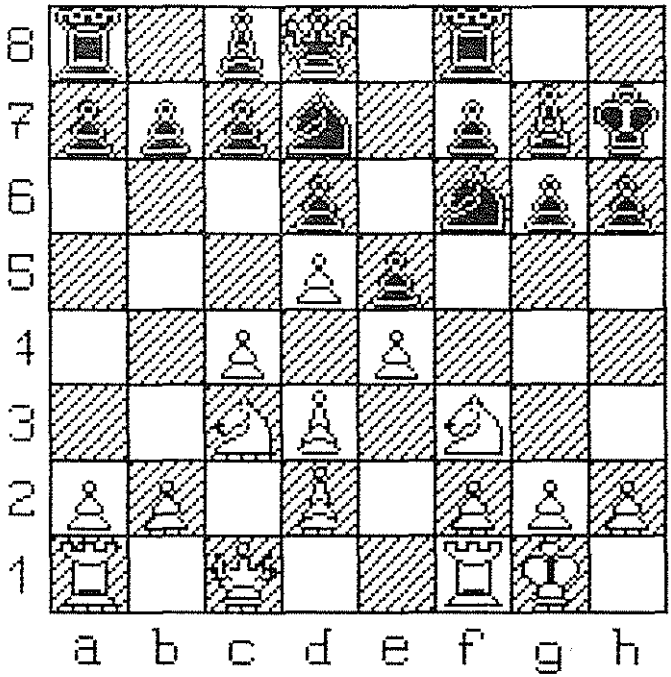
Mit der Unterscheidung nach dem vorliegenden Partieabschnitt, der sogenannten *"Endspielabfrage"*, wollen wir unsere Besprechung der bisher in Schachprogrammen verwendeten Verfahren beginnen. Eine sicherlich sehr unzuverlässige Methode ist es, die Partiedauer zur Entscheidung heranzuziehen und die Stellungen nach einer gewissen Anzahl gespielter Züge als Endspielpositionen zu behandeln. Dies kann zum schnellen Partieverlust führen, wenn im bisherigen Spielverlauf nur wenig Material abgetauscht wurde.

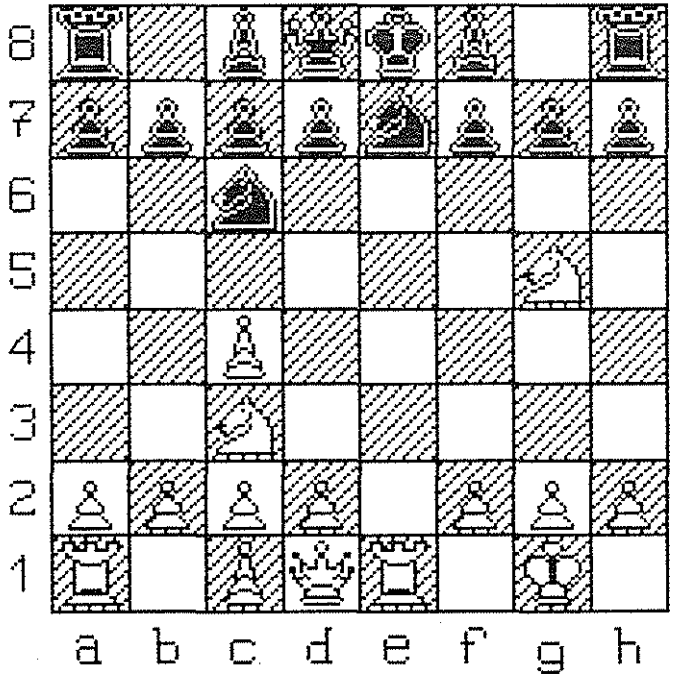
Stattdessen ist es die *gegnerische Materialstärke*, die in erster Linie darüber Auskunft gibt, ob sich der König aus seinem Versteck wagen darf. Das gegnerische Angriffspotential sollte möglichst stark reduziert sein, damit der König vor plötzlichen Mattdrohungen sicher ist. Dieser Aspekt wird von praktisch allen Schachprogrammen berücksichtigt, allerdings mit unterschiedlich hohen Werten für die obere Schranke der Materialsumme, bei dessen Unterschreiten der Übergang ins Endspiel vollzogen wird. In der Regel liegen diese Werte in der Größenordnung von 3 Figuren (Springern, Läufern und Türme, aber keine Damen) und einigen Bauern; das sind bei Verwendung der Figurenwerte aus unserer Materialbewertung ca. 1500 - 2000 Punkte. Dabei ist zu beachten, daß die Überprüfung getrennt

für die Materialsummen beider Spieler durchgeführt werden muß. So kann es bei Materialvorteil für eine Seite vorkommen, daß der König des betreffenden Spielers sich schon aktiv am Geschehen beteiligen kann, während sein Opponent noch auf seine Sicherheit bedacht sein muß.

Eine weitere Entscheidungshilfe für die Endspielabfrage, die jedoch der Materialstärke untergeordnet sein sollte, bietet sich mit dem Stellungscharakter an. So könnte man die Schranke für das Material in gewissen Grenzen betragsmäßig variieren, je nachdem, ob eine offene oder geschlossene Position vorliegt. Auf diesen Zusammenhang sind wir schon früher eingegangen. Wir haben aber auch die Probleme aufgezeigt, die sich bei dem Versuch ergeben, den Stellungscharakter in die Bewertungsfunktion einfließen zu lassen. Entsprechend selten findet man diesbezügliche Ansätze in den existierenden Schachprogrammen.

Wenn wir mit Hilfe der Endspielabfrage festgestellt haben, daß eine *Bewertung der Königssicherheit* vorgenommen werden muß, stellt sich die Frage, welches Verfahren geeignet ist, präzise Abschätzungen zu liefern. So wichtig es ist, den vorliegenden Sachverhalt exakt zu analysieren, damit der König keinen Gefahren ausgesetzt wird, so unterschiedlich sind die bisher benutzten Methoden. Eine sehr einfache und schnell zu berechnende, aber leider häufig irreführende Bewertung wurde in dem Programm MAC HACK VI von R. Greenblatt (GRE) implementiert. Dort wurde lediglich der Standort des Königs betrachtet, der als sicher angenommen wurde, solange er sich auf der Grundreihe befindet. Die Bewertungen zu den beiden folgenden Stellungen zeigen, zu welchen Fehlurteilen dieses Verfahren gelangen kann:


8
7
6
5
4
3
2
1
a b c d e f g h



Die von Greenblatt für die Bewertung der Königssicherheit benutzte Berechnungsvorschrift lautet:

$$K = 8 * ((9-R(SK)) - R(WK))$$

Dabei bedeuten R(WK) bzw. R(SK) die Reihenangaben für die Standorte des weißen bzw. des schwarzen Königs. Daraus ergibt sich für die erste Stellung K=8, während für die zweite K=0

gilt. Diese Ergebisse sind so zu interpretieren, daß in der ersten Position die Königsstellung von Weiß sicherer als die des schwarzen Königs erachtet wird, während die andere Position bzgl. der Königssicherheit als ausgeglichen beurteilt wird. Tatsächlich steht aber in I der schwarze König völlig sicher, während in II nur noch das Opfer von Material (d7-d5) gegen die Mattdrohung Lc4xf7++ schützt. Das Fehlen wichtiger Stellungsmerkmale wie *Kontrollen in der Nähe des Königs* - sowohl der Läufer auf C4 als auch der Springer auf G5 bedrohen F7 - oder *Entwicklungsrückstand* sind in erster Linie für die unrealistischen Werte verantwortlich.

Ein besser geeigneter Ansatz stammt von Shannon. Er schlug nämlich vor, den *Bauernschutz* sowie Kontrollen in der Nähe des gegnerischen Königs zu bewerten. In der Tat sind fehlende oder (zu weit) vorgerückte Bauern und Zugmöglichkeiten auf dem Königsstandort benachbarte Felder wichtige Indizien für eine mögliche Gefährdung des Königs. Im Grundsatz verschieden, aber trotzdem zu ähnlichen Ergebnissen führend ist die Bewertungsmethode von Turing. Dieser berechnet die imaginäre Mobilität einer auf dem Königsstandort aufgestellten Dame und bestimmt die Höhe eines eventuellen Punktabzuges proportional zur Anzahl der ermittelten Zugmöglichkeiten. Durch dieses Maß wird implizit ebenfalls der Bauernschutz bewertet, da eine vor dem König postierte Bauernkette die Mobilität stark einschränkt. Ein Vorteil dieses originellen Verfahrens liegt darin, daß es auch schon in der Eröffnung, also vor der Ausführung der Rochade, angewendet werden kann. In dieser Phase wird es möglichst frühzeitig die Rochade anstreben, um die durch die Anfangszüge entstandene große Mobilität des Königs auf E1 zu reduzieren.

Nach dieser eher exemplarischen Übersicht bzgl. der Bewertungsalgorithmen für die Königssicherheit kommen wir nun zu einem Verfahren, mit dem versucht wurde, die Erfordernisse des Schachspiels weitgehend zu berücksichtigen. Es stammt aus dem Schachprogramm BOBBY, und wir werden es hier ausführlich beschreiben, weil wir es (mit geringen Vereinfachungen) in unser Programm übernehmen wollen.

Die Entwicklung und damit auch die Eröffnung gilt als abgeschlossen, sobald die Türme verbunden sind. In der Regel hat damit auch der König eine Rochadestellung und den optimal geschützten Platz erreicht. Da BOBBY nur eine relativ kleine Eröffnungsbibliothek zur Verfügung steht, wurde versucht, bei der Konstruktion dieser Bewertung "zwei Fliegen mit einer Klappe zu schlagen". Dies wird deutlich, wenn man die verarbeiteten Stellungsmerkmale betrachtet:

1. *Anzahl der noch vorhandenen gegnerischen Figuren mit besonderer Berücksichtigung der Dame:*
   Damit wird die Wichtigkeit für die Beachtung der Königssicherheit bestimmt. Als Folge dieses Faktors wird ein Spieler mit geschwächter Königsstellung (z.B. fehlender Bauernschutz) versuchen, Figuren abzutauschen, bevor ihm diese gefährlich werden können.

2. *Relation der beiden Königsstandorte:*
   Es gibt einen Punktabzug für denjenigen Spieler, dessen König noch auf einer Zentrumslinie (D- oder E-Linie) steht, während sein Gegner seinen König schon in Sicherheit gebracht hat. Die Höhe dieses Abzuges ist davon abhängig, ob die Linie, auf der der ungeschützte König steht, schon geöffnet ist, da dann wegen des fehlenden Bauernschutzes der betreffende König besonders gefährdet ist.

3. *Anzahl der Felder, die der König momentan noch von einem sicheren Platz (A1, B1, C1 bzw. G1, H1 für den weißen König analog auf der 8. Reihe für Schwarz) entfernt ist:*
   Dies ist dann von Bedeutung, wenn der König seine Rochademöglichkeiten verloren hat, z.B., nachdem er am Damentausch auf D1 beteiligt war. Damit in solchen Fällen der König nicht eine "Reise über alle Grundreihenfelder" antritt, um auf G1 unterzuschlüpfen, wird der Abstand zu den sicheren Feldern (s.o.) als entscheidendes Kriterium für die anzustrebende Rochadestellung herangezogen.

4. *Reihe des Königsstandortes:*
Dieser Punkt ist von Interesse, wenn der Gegner den König durch ein Figurenopfer von der Grundreihe geholt hat und versucht, ihn durch den folgenden Angriff weiter ins Zentrum zu zwingen. Wegen der großen Wichtigkeit dieses Kriteriums werden für solche nachteiligen Aufstellungen Werte von der Größenordnung eines Bauern und mehr in Anrechnung gebracht.

5. *Anzahl der noch auszuführenden Züge bis zum Erreichen einer Rochadestellung:*
Dadurch fließt die Entwicklungsdauer in die Bewertung ein, da ein gegnerischer Angriff auf den König umso gefährlicher ist, je mehr Zeit (in Zügen gemessen) noch benötigt wird, um den König auf ein geschütztes Feld zu bringen. Gleichzeitig fördert dieser Aspekt eine rasche Entwicklung der Figuren in der Eröffnung.

6. *Gegnerische Figuren, die bis zur Grundreihe vorgedrungen sind*, werden berücksichtigt, da sie im allgemeinen das Manöver, den König in Sicherheit zu bringen, erschweren.

7. *Bauernschutz auf dem Rochadeflügel:*
Fehlende oder vorgerückte Bauern, die dem Gegner den Königsangriff erleichtern können, werden ebenfalls in der Bewertung der Königssicherheit berücksichtigt; und zwar schon vor der Ausführung der Rochade, so daß nicht zuerst die Flügelbauern vorgerückt werden und anschließend für den König kein Unterschlupf mehr gefunden wird.

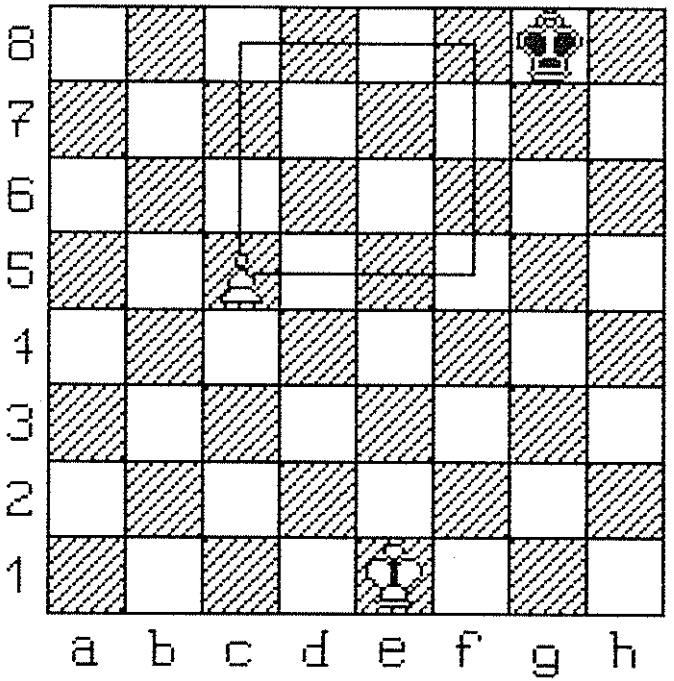
8. *Die lange Rochade wird durch einen (geringen) Punktabzug etwas gehandicapt*, da der König meist auf dem Königsflügel sicherer als auf dem Damenflügel steht. Dieses Kriterium wirkt sich erst aus,

wenn eine Stellung nach der Ausführung der Rochade bewertet werden soll.

Es sind also viele Stellungsmerkmale, von denen die Bewertung der Königssicherheit abhängt. Wie stark sie im einzelnen die numerische Einschätzung beeinflussen, soll uns an dieser Stelle nicht interessieren. Darauf gehen wir später bei der Beschreibung der in unserem Programm implementierten Version ein. Über die eben aufgeführten Kriterien hinaus tragen in BOBBY die Bewertung der Felderkontrollen mit Wichtung der Nachbarfelder der Könige sowie die Berechnung des Abstandes zum gegnerischen König für Springer, Türme und Damen ihren Anteil zu einer präzisen Königsbewertung in Eröffnungs- und Mittelspielpositionen bei.

Doch damit vorerst genug zum Thema "Königssicherheit" - kommen wir nun zum letzten Kapitel über Stellungsbewertungen, der speziellen *Endspielbehandlung*. Wie schon erläutert, hängt eine gute Beurteilung von Endspielpositionen eng mit genauen Abschätzungen der *(aktiven) Rolle des Königs* zusammen. Daß außerdem eine Reihe weiterer auf das Endspiel zugeschnittener Wichtungen bereits vorgestellter Stellungsmerkmale durchgeführt wird, haben wir auf den vorangegangenen Seiten erfahren. Als Beispiele seien nochmals die Verdopplung der Freibauernbewertung bzw. die erhöhte Berücksichtigung der Relation Turm-Freibauer genannt.

In der "Frühzeit" der Schachprogrammierung (Shannon, Turing, Greenblatt) wurde die Notwendigkeit, die Eigenarten der letzten Partiephase bei der Bewertung besonders zu beachten, noch nicht erkannt. Anfang der 70er Jahre implementierte M. Newborn (NEW) in seinem Programm OSTRICH ein Verfahren, daß den König dazu bewegen sollte, sich im Endspiel den gegnerischen Bauern zu nähern, um sie eventuell zu schlagen. Dies geschah durch einen Bonus, der vergeben wurde, falls sich der *König im Quadrat eines gegnerischen Bauern* befindet. Mit dem Ausdruck "Quadrat" eines Bauern werden diejenigen Felder bezeichnet, von denen aus der König einen gegnerischen Freibauern an der Umwandlung hindern kann.



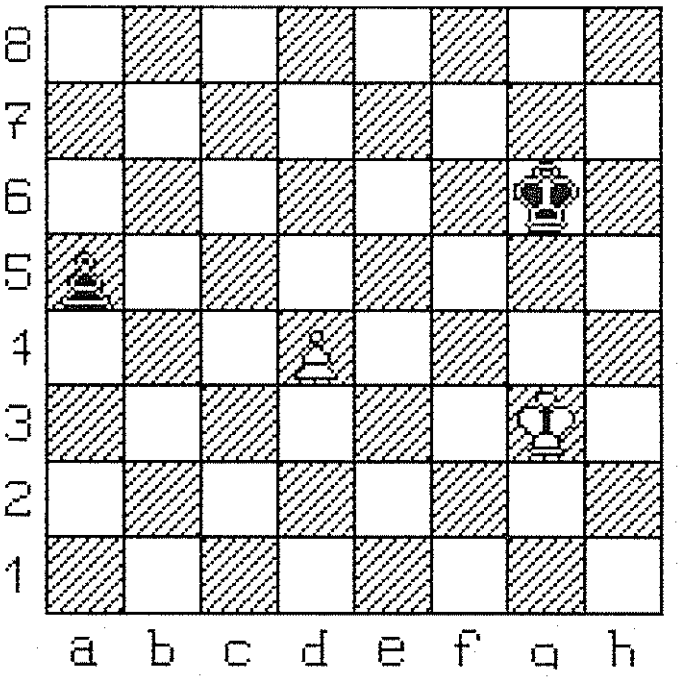
Ist in dieser Beispielstellung Weiß am Zug, kann Schwarz nicht mehr die Umwandlung des Bauern C5 verhindern. Mit getauschtem Anzugsrecht betritt Schwarz rechtzeitig das Quadrat durch Kg8-f7 oder Kg8-f8 und kann den weißen Freibauern stoppen.

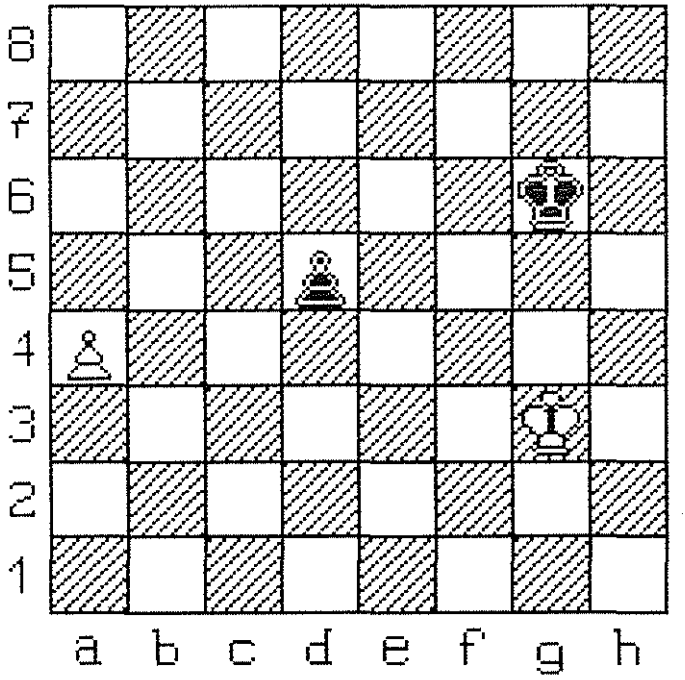
Etwas differenzierter als in OSTRICH verläuft die Endspielbewertung in CHESS 4.5. Hier fließen zwei Kriterien in die Beur-

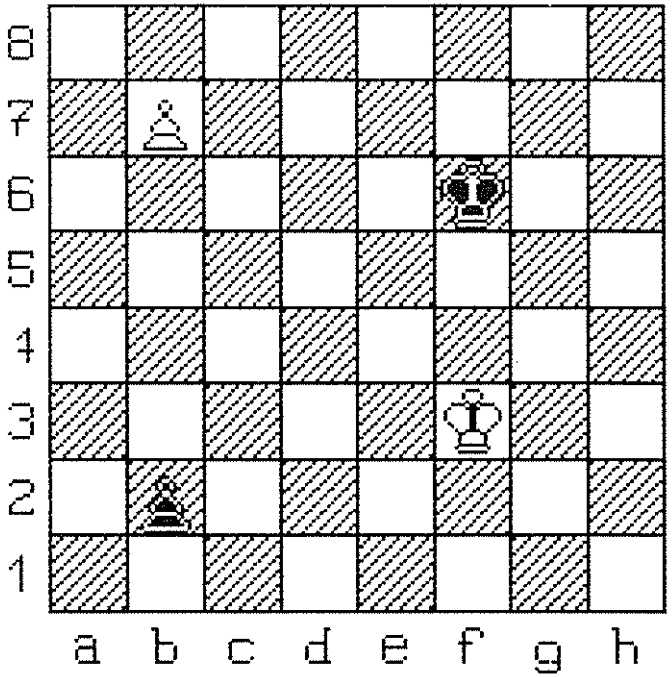
teilung der Königsaktivität ein. Das erste betrifft die *Zentralisierung des Königs*. Diese Bewertung ist durch die kurzschrittige Gangart der Könige motiviert, der vom Zentrum aus schneller in umkämpfte Regionen des Brettes eilen kann, sei es, um einen gegnerischen Freibauern an der Umwandlung zu hindern oder einen eigenen schwachen Bauern (rückständig oder isoliert) vor dem Geschlagenwerden zu schützen. Desweiteren kann dem gegnerischen König durch die Beherrschung der Brettmitte häufig das Eindringen ins eigene Lager verwehrt werden.

Die zweite Teilbewertung bezieht sich auf die *Abstände des Königs zu eigenen und gegnerischen Bauern*. Dadurch soll ebenfalls erreicht werden, daß sich der König so nahe wie möglich am Schauplatz des Kampfgeschehens aufhält. Dieses Maß wird bestimmt, indem alle Reihen- und Linienabstände zu den Bauernstandorten aufsummiert werden und dieses Ergebnis dann durch die Anzahl vohandener Bauern geteilt wird. Dieser mittlere Abstand fließt gewichtet in die Endspielbewertung ein.

Mit diesen beiden Stellungsmerkmalen ist in Zusammenarbeit mit der Bauernbewertung (und dort insbesondere durch die Einschätzung der Freibauern) eine recht gute Positionsanalyse von Endspielstellungen möglich. Daß es aber trotzdem noch genügend Probleme durch zu grobe oder falsche Bewertungen gibt, mögen die folgenden Beispiele zeigen:


8
7
6
5
4
3
2
1
a b c d e f g h


8
7
6
5
4
3
2
1
a b c d e f g h



Die Stellungen lassen sich schachtheoretisch wie folgt abschätzen:

zu I: Die Anwendung der Quadratregel ergibt, daß Schwarz unabhängig vom Anzugsrecht gewonnenes Spiel hat, da sein Bauer genügend weit vom weißen König entfernt steht, um ungehindert nach A1 durchzulaufen, während der Bauer auf D4 durch den schwarzen König gestoppt werden kann.

zu II: Analog zu I ist diese Stellung für Weiß gewonnen.

zu III: Für den Ausgang dieser Partie ist maßgebend, welcher Spieler am Zug ist, da dieser zuerst seinen Bauern umwandeln und mit der dadurch entstehenden Dame (bzw. dem Turm) das gegnerische Umwandlungsfeld sofort (indirekt) kontrolliert.

Mit dem Bewertungsverfahren von CHESS 4.5 würden alle drei Positionen als ausgeglichen eingeschätzt, weil jeweils beide Könige den gleichen Abstand zur Brettmitte sowie den gleichen mittleren Abstand zu den verbliebenen Bauern haben, obwohl, wie wir gerade gesehen haben, die Stellungen durch nicht zu verhindernde Bauernumwandlungen zwangsläufig entschieden werden können. Man könnte meinen, daß das Erkennen dieser Sachverhalte eigentlich nicht die Aufgabe der Bewertungsfunktion sei, sondern vielmehr von dem Suchprozeß erledigt werden sollte. Dem läßt sich entgegenhalten, daß z.B. für die Stellungen I und II die Suche bis zu 14(!) Halbzüge tief die Zugfolgen analysieren muß, um zum richtigen Resultat zu kommen. Bei noch komplizierteren Endspielstellungen kann die erforderliche Suchtiefe unter Umständen noch wesentlich größer sein und die Suche entsprechend aufwendiger werden lassen.

Aus diesen Gründen wäre es praktisch, einen Bewertungsalgorithmus zur Verfügung zu haben, der auf statischem Wege, d.h., ohne Zugfolgen vertiefen zu müssen, gute Näherungslösungen zumindest für manche Endspieltypen anbietet. In der Endspielbewertung von BOBBY wurde der Versuch einer entsprechenden Konstruktion unternommen. Im folgenden wird die Arbeitsweise dieses Verfahrens grob skizziert:

Von besonderer Bedeutung in solchen Positionen, wie wir sie eben betrachtet haben, sind die Freibauern, da sie oft das Spiel entscheiden. Dies gilt vor allem in Bauernendspielen, wenn nur noch die Könige ihren Vormarsch beeinflussen können.

In der nun zu besprechenden Endspielbewertung werden daher in erster Linie die Zusammenhänge zwischen Freibauern und den Königsstandorten untersucht. Dabei interessieren besonders Freibauern, die nicht mehr vom gegnerischen König an der

Umwandlung gehindert werden können. Zu diesem Zweck wurde in diesem Algorithmus die Anwendung der Quadratregel realisiert. Eine korrekte Bewertung der Stellungen I und II ist damit schon kein Problem mehr. Des weiteren werden Bauern berücksichtigt, in deren Quadrat zwar der König steht, die aber wegen Unterstützung durch den eigenen König bis zur Grundreihe durchlaufen können.

Sind solche unaufhaltsam vorrückenden Bauern auf beiden Seiten vorhanden, muß überprüft werden, welcher Spieler als erster umwandeln kann. Dies wird durch die Anzahl von Zügen, die die entsprechenden Bauern noch bis zum Erreichen der gegnerischen Grundreihe benötigen, sowie durch das Anzugsrecht entschieden. Wenn festgestellt wird, daß beide Bauern unmittelbar aufeinanderfolgend umgewandelt werden können, gibt es Ausnahmesituationen, in denen der zweite Spieler nicht mehr zur Umwandlung kommt. Dies ist der Fall, wenn die eingetauschte Figur gleichzeitig Schach bietet oder das gegnerische Umwandlungsfeld kontrolliert (wie in Stellung III). Solche Möglichkeiten werden von der Endspielbewertung ebenfalls erfaßt.

Wenn von diesem Verfahren festgestellt wird, daß ein Spieler durch eine unvermeidbare Umwandlung in (entscheidenden) Vorteil gelangen kann, werden z.T. sehr hohe Bewertungen vorgenommen. Dies kann in zweifelsfreien Stellungen sogar zur Vergabe eines Bonus in der Höhe einer Dame führen. Sicherlich sind solche Bewertungen eine extreme Verletzung der generell benutzten Einschätzung, ein materielles Plus sehr viel höher als positionelle Vorteile zu bewerten. Da aber diese spezielle Endspielbehandlung (mit wenigen Einschränkungen) einer Ersetzung des Suchprozesses gleichkommt und ihre Ergebnisse daher als gleichrangig angesehen werden können, ist diese Vorgehensweise gerechtfertigt.

In Stellungen, in denen zwar Freibauern vorhanden sind, diese aber vom gegnerischen König aufgehalten werden können, ist oft der Abstand zwischen ihnen und den Königen entscheidend. In diesem Zusammenhang spricht man von "entfernten Freibauern". Damit sind solche Positionen gemeint, in denen der eine

König einen weiteren Weg zum gegnerischen Freibauern als der Gegenspieler zurückzulegen hat, so daß er unter Umständen zu spät in den Kampf um die verbleibenden Bauern eingreifen kann, nachdem beide Seiten jeweils den feindlichen Freibauern beseitigt haben. Um solche Fälle richtig zu behandeln, wird in der Endspielbewertung die Entfernung Freibauer-König berücksichtigt.

Schließlich wird in Endspielstellungen ohne Freibauern überprüft, wie weit die beiden Könige jeweils von dem am nächsten liegenden schwachen Punkt in der gegnerischen Bauernformation entfernt sind. Mit den "schwachen Punkten" sind Bauern gemeint, die höchstens von dem eigenen König gedeckt sind; dies ist insbesondere bei rückständigen Bauern der Fall. Durch Beachtung dieses Kriteriums soll erreicht werden, daß sich Angriffe des Königs zweckmäßigerweise gegen solche (schwer zu verteidigende) Ziele richten und, aus der anderen Sicht, schwache Bauern in der Nähe des gegnerischen Königs nach Möglichkeit vermieden werden.

Damit sind sämtliche in BOBBYs statischer Endspielbewertung berücksichtigten Stellungsmerkmale beschrieben. Wir beenden hiermit unsere theoretischen Betrachtungen und wenden uns wieder der Praxis zu.

Verfahren in unserem Programm

Das gerade erworbene Wissen wollen wir nun bei der Programmierung der königsbewertung einsetzen. Dabei stehen drei Punkte auf der Tagesordnung. Beginnen wir mit der

Endspielabfrage:

Den theoretischen Überlegungen folgend werden wir die Materialsummen verwenden, um zwischen Eröffnungs- und Mittelspielpositionen einerseits und Endspielstellungen andererseits zu unterscheiden. Als Grenzwert setzen wir 2000 Punkte an. Bezogen auf unsere Figurenwerte ist das ungefähr die Materialstärke von zwei Türmen, einem Läufer und sieben Bauern. Diesen Betrag müssen wir mit beiden Materialsummen

vergleichen, um die Endspielabfrage für beide Seiten getrennt zu beantworten. Beim jeweiligen Unterschreiten dieser Grenze wird in der Königsbewertung für die betreffende Farbe die Königsaktivität berücksichtigt, ansonsten die Königssicherheit.

Der zweite Punkt ist die Konstruktion des

Verfahrens zur Bewertung der Königssicherheit:

Wie schon erwähnt, wollen wir im wesentlichen den Algorithmus aus dem Schachprogramm BOBBY übernehmen. Wenn wir im folgenden die zu implementierenden Stellungsmerkmale und deren numerische Einschätzungen beschreiben, reicht eine einfache Aufzählung wie im Falle der Bauern- oder der Figurenbewertung nicht aus. Dort war zum einen leicht festzustellen, ob in der aktuellen Stellung die betreffenden Kriterien erfüllt waren, und zum anderen konnten die Einzelbewertungen unabhängig voneinander vorgenommen werden. Das ist hier anders. Nicht nur, daß wir es jetzt mit einer Reihe von Stellungsmerkmalen zu tun haben, deren Bewertungen nur Zwischenergebnisse liefern, die am Schluß in einer "Gesamtformel" zusammengefaßt werden müssen, es ist beispielsweise auch nicht so einfach, den Bauernschutz schon vor der Ausführung der Rochade in der Bewertung zu berücksichtigen, weil wir uns dadurch vorab für die Bauern eines Flügels entscheiden müssen. Wir werden daher, ähnlich wie bei der Beschreibung der Felderkontrollenbewertung und der Verwendung des Stacks, an dieser Stelle auch auf den Informationsfluß in dem Verfahren zur Königsbewertung eingehen müssen. Dazu werden wir Variablennamen benutzen, die für die Einzelbewertungen stehen. Das werden nicht unbedingt die gleichen Bezeichnungen sein, die später im Programm auftauchen; ihr Zweck ist vielmehr der, die Bewertungsvorschrift(en) übersichtlicher gestalten zu können. Für das Endergebnis der Berechnung steht z.B. die Variable ks%.

Ein Kriterium, das in jede Bewertung der Königssicherheit einfließt, egal wo der König gerade steht, ist die Anzahl der gegnerischen Figuren. In dieser Summe werden Bauern gar nicht, Damen dagegen mit dem Wert 3 berücksichtigt. Damit

trotzdem für die Grundstellung der Betrag 8 benutzt werden kann, wird von der Summe immer die Konstante 2 abgezogen. Diese Beschreibung gilt für die weißen Figuren; die Berechnung für Schwarz verläuft analog mit umgekehrtem Vorzeichen, so daß wir für die Bewertung der Sicherheit des weißen Königs eine Wichtung in Abhängigkeit der schwarzen Figuren vornehmen, die in Eröffnungs- und Mittelspielpositionen zwischen den Werten -8 und -1 liegt. Für diese Figurensumme benutzen wir in der weiteren Beschreibung die Variable fs%.

Der zweite Anteil an der Bewertung ergibt sich aus den übrigen Stellungsmerkmalen, mit denen die noch benötigte Entwicklungsdauer, die Schutzfunktion der Bauern, u.a. überprüft werden. Dies ist eine Abschätzung der Risikofaktoren, die umso höhere Werte liefert, je mehr Schwachstellen in der Königsumgebung gefunden werden. Ist dagegen "alles o.k.", spiegelt sich dies im Wert 0 wider. Insgesamt erhalten wir dann durch die Multiplikation mit fs% das Ergebnis 0 für einen geschützten König und mehr oder weniger hohe negative Werte als Bestrafung für die Vernachlässigung der Königssicherheit.

Wir müssen uns nun überlegen, auf welche Art wir die restlichen Kriterien zu einer Bewertung zusammenfassen können. Der erste zu bewertende Aspekt ist die Relation der Königsstandorte. Dazu setzen wir die Variable r% nach folgendem Schema:

r%=0, falls Weiß schon rochiert hat oder noch beide Könige auf einer Zentrumslinie stehen,

r%=1, falls Weiß noch nicht, Schwarz aber schon rochiert hat und die Linie des Standortes des weißen Königs geschlossen oder halboffen ist,

r%=2, wie für r%=1, aber Linie offen.

r% ist der erste Summand für die Bewertung des Schutzes. Welche Einzelwerte noch hinzukommen, richtet sich nach dem augenblicklichen Königsstandort, für den vier Fälle unterschieden werden:

1. Exponierte Königsstellung

Steht der weiße König weder auf der 1. noch auf der 2. Reihe, so wird eine Sonderbewertung vorgenommen, die zu einer drastischen Bestrafung für die hohe Gefährdungswahrscheinlichkeit des Königs führt. Zu dem Wert von r% werden 20 Punkte addiert, so daß die Bewertung durch die Multiplikation mit der Figurensumme den Wert von 1.5 bis 2 Bauern erreichen kann. Für die Bewertung der Königssicherheit ergibt sich

(Formel KS1) ks% = fs% * (r% + 20)

2. Königsstellung vor Ausführung der Rochade

Wenn der König noch auf der E-Linie steht, ist der Reihenabstand zu den am nächsten gelegenen sicheren Feldern auf dem Damenflügel (C1) bzw. Königsflügel (G1) gleich groß (=2). Wir müssen daher vorsorglich die Königssicherheit für beide Seiten des Brettes berechnen, um uns anschließend für den günstigeren Flügel zu entscheiden. Dies geschieht, indem wir für beide Bretthälften nach obigem Schema (hohe Gefährdung = hohe Werte) einen Schutzwert bestimmen (in den Variablen sw%(1) für den Damenflügel und sw%(2) für den Königsflügel) und danach das Minimum dieser Werte bilden.

Folgende Kriterien fließen in die Abschätzung ein:

- Bauernschutz:
  Erhöhung des Schutzwertes um jeweils 0.5 Punkte für vorgerückte, d.h. nicht auf der 2. Reihe stehende Bauern bzw. Erhöhung des Schutzwertes um jeweils 1.0 Punkte für fehlende Bauern.

  Betrachtet werden auf dem Damenflügel A-, B- und C-Linie sowie auf dem Königsflügel G- und H-Linie.

- Entwicklungsdauer:
  Erhöhung des Schutzwertes um jeweils 0.5 Punkte für nicht entwickelte Figuren.

Betrachtet werden hier die Felder B1, C1 und D1 bzw. F1 und G1.

Zusätzlich wird auf den Schutzwert ein Betrag addiert, der von der Anzahl benötigter Züge des Königs bis zum Erreichen einer Rochadestellung abhängt. Ist die Rochade noch möglich, kommen 0.5 Punkte hinzu, ansonsten 2.0 Punkte.

Die Abschätzung der Königssicherheit berechnet sich dann insgesamt aus

(Formel KS2) ks% = fs% * (r% + min(sw%(1),sw%(2))).

3. Königsstellung nach Ausführung der langen Rochade

Hier kann die Bewertung im Vergleich zu Fall 2 vereinfacht werden, da wir nur den Damenflügel und dort nur den Bauernschutz berücksichtigen müssen. Außerdem gilt nach der Rochade grundsätzlich r%=0.
Die Berechnungsvorschrift lautet also

(Formel KS3) ks% = fs% * sw%(1).

4. Königsstellung nach Ausführung der kurzen Rochade

Dieser Fall wird analog zu 3 behandelt; daher

(Formel KS4) ks% = fs% * sw%(2).

Mit Hilfe dieses Verfahrens und insbesondere der Fallunterscheidung für den Königsstandort können wir den meisten Stellungen eine recht gut angepaßte Bewertung der Königssicherheit zuteil werden lassen. Im Kapitel 4 werden wir sehen, daß dieser relativ kompliziert anmutende Algorithmus durchaus mit vertretbarem Programmieraufwand implementiert werden kann.

Kommen wir nun zum letzten Punkt, dem

Verfahren zur Bewertung der Königsaktivität:

Die Bewertungsmethoden zur Endspielbehandlung beinhalten zum einen eher generelle Prinzipien wie Zentralisierung und mittlerer Abstand zu den Bauern und zum anderen speziell auf diesen Spielabschnitt zugeschnittene Stellungsmerkmale wie z.B. die statische Abschätzung der Umwandlungschancen von Freibauern in Bauernendspielen. Wenn wir in unserem Programm lediglich die allgemeinen Grundsätze berücksichtigen werden, so liegt der einzige Grund hierfür in dem üppigen Speicherplatzbedarf, der für solche wie in BOBBY implementierte Verfahren veranschlagt werden muß.

Es bleiben daher nur zwei zu bewertende Kriterien übrig:

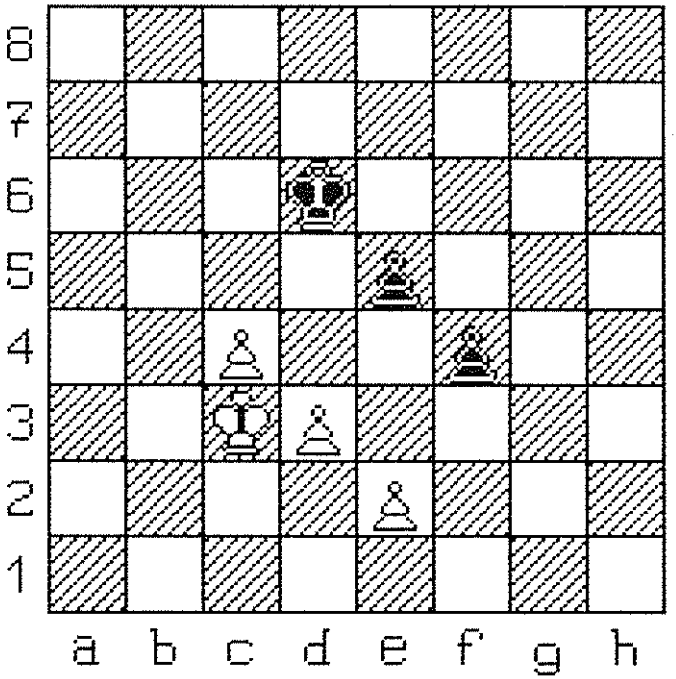
1. Zentralisierung

Dieses Maß ist uns aus den Beschreibungen der Matt-Routine bzw. der Springerbewertung wohlbekannt. Der Wert ergibt sich aus der Summe von Reihen- und Linienabstand zum Brettmittelpunkt (5.5,6.5), die mit dem Faktor -4 gewichtet wird. Dadurch wird erreicht, daß ein großer Abstand zum Zentrum durch entsprechend hohe Punktabzüge bestraft wird.

2. Mittlerer Abstand zwischen König und Bauern

Die Bewertung für dieses Stellungsmerkmal ist etwas komplexer als die Ermittlung des Zentralisierungsgrades, zumal wir uns nicht darauf beschränken werden, alle Bauern gleich zu behandeln, sondern vielmehr die Bauern ihrer strategischen Bedeutung entsprechend in der Bestimmung des mittleren Abstandes berücksichtigen werden. Dazu teilen wir die Bauern in drei Gruppen ein, wobei wir uns in etwa an einem Ansatz in BOBBYs Endspielbewertung orientieren werden: die Freibauern, die rückständigen Bauern und den Rest. Dieser Aufteilung folgend multiplizieren wir den Abstand des Königs zu einem Freibauern mit dem Wichtungskoeffizienten 6, zu einem rückständigen Bauern mit 3 und zu den übrigen Bauern mit 2. Diese Abstufung bewirkt, daß sich die Könige im Endspiel in erster Linie um die Freibauern kümmern, falls solche vorhanden sind,

und diese gegebenenfalls beim Vorrücken unterstützen bzw. behindern. Die rückständigen Bauern sind die schwächsten Stellen in der Bauernstruktur. Es liegt daher nahe, auch diesen eine erhöhte Aufmerksamkeit zu widmen, um eigene Angriffe gegen diese Punkte zu starten bzw. gegnerische Attacken abzuwehren. Um aber bei der Berechnung des mittleren Abstandes nicht zu einem Resultat zu gelangen, das durch die Multiplikation mit den Wichtungskoeffizienten über der maximal möglichen Distanz (=14) liegt, dürfen wir die Summe der Abstände zu allen Bauern nicht einfach durch die Bauernanzahl dividieren, sondern wir müssen die "Summe der Bedeutungen" benutzen. Dies machen wir uns an einem Beispiel klar:



Für die Bewertung der Aktivität des weißen Königs benötigen wir (neben dem Zentrumsabstand, den wir im Moment nicht betrachten wollen) die Bedeutung der einzelnen Bauern sowie die jeweiligen Abstände des Königs zu ihnen. In einer tabellarischen Übersicht sieht diese Information so aus:

Figur	Standort	Bedeutung	Abstd. z. w. König
weißer Bauer	C4	6 (Freibauer)	1
weißer Bauer	D3	2 (ohne Merkmal)	1
weißer Bauer	E2	3 (rückständig)	4
schwarzer Bauer	F4	2 (ohne Merkmal)	4

Der mittlere, bzgl. der Bedeutung der Bauern gewichtete Abstand berechnet sich demnach zu

$$(6*1 + 2*1 + 3*3 + 3*4 + 2*4) : (6+2+3+3+2) = 37 : 16 = 2.3125$$

Zum Vergleich: Würden wir die Abstandsbewertung ohne Berücksichtigung der unterschiedlichen Wichtigkeit der Bauern durchführen, bekämen wir als Ergebnis

$$(1+1+3+4+4) : 5 = 13 : 5 = 2.6$$

Da der Abstandswert so klein wie möglich sein sollte, wirkt es sich durch die gewichtete Mittelwertbildung positiv aus, daß der König so nahe bei seinem Freibauern auf C4 steht und diesen beim Vormarsch unterstützen kann.

Die endgültige Bewertung für dieses Stellungsmerkmal wird mittels folgender Formel vorgenommen:

$$\text{Abstandswert} = -6 * (\text{mittl. gewichteter Abstand} - 6)$$

Es ergibt sich also insgesamt eine positive Abschätzung, wenn es dem König gelingt, den mittleren Abstand auf einen Wert unter 6 zu reduzieren; für größere Distanzen werden Punktabzüge vorgenommen.

3.6 Hilfsroutinen

Hilfsroutinen sind Routinen, die technische Teilaufgaben lösen. Meist werden sie von verschiedenen Moduln oder Routinen benutzt.

Für diesen Abschnitt, in dem wir uns mit den Prinzipien der Schachprogrammierung beschäftigen, sind sie ohne große Bedeutung. Deshalb beschränken wir uns hier im wesentlichen auf eine Aufzählung.

a) König im Schach?

Nur eine Hilfsroutine verlangt eine konzeptionelle Entscheidung. Die Suche muß wissen, ob der König des Spielers, der am Zug ist, im Schach steht. Es gibt zwei Möglichkeiten, dies zu berechnen:

1) Wir berechnen die Züge des Spielers, der nicht am Zug ist, und sehen in der Schlagzugliste nach, ob er den gegnerischen König schlagen könnte.

2) Wir gehen vom Standort des Königs aus und prüfen alle Felder, von denen aus er angegriffen sein könnte.

Das Problem des ersten Ansatzes ist, daß wir normalerweise nur für den am Zug befindlichen Spieler die Zugliste berechnen. Wir müßten dies auch für den Gegner machen, und das kostet relativ viel Zeit. Daher haben wir im Programm die Alternative vorgezogen.

Sie ist wesentlich schneller, denn praktisch müssen nur die vom Königsstandort aus mit Damen- oder Springerzügen erreichbaren Felder betrachtet werden.

Der erste Ansatz hat aber auch Vorteile: Durch die Berechnung der Züge des Spielers, der nicht am Zug ist, kann die Bewertungsfunktion substanzielles Wissen gewinnen (vgl. Bewertung der Zugmöglichkeiten).

b) Die übrigen Hilfsroutinen

Die weiteren Hilfsroutinen helfen uns bei der Strukturierung des Programms. Sie lösen eine recht umfangreiche Teilaufgabe, die

dort, wo sie erledigt werden muß, durch eine einzige gosub-Anweisung realisiert werden kann.

Der Zuggenerator sollte möglichst wenig über die Züge berechnen, da die meisten (wegen Cutoffs) doch nicht untersucht werden. Für die übrigen Züge benötigen wir aber eine Vielzahl von Informationen. Diese berechnet eine Hilfsroutine ("Bestimmung des Zuges"), die alle Eigenschaften des aktuellen, nur durch Standort und Zielfeld angegebenen Zuges bestimmt.

Die Suche muß Varianten analysieren, also Züge probeweise ausführen. Auch hierfür haben wir eine Routine ("Zug vertiefen") geschrieben. Sie verändert die interne Brettdarstellung entsprechend dem Zug.

In diesem Zusammenhang tritt Rekursion auf - nach der Analyse einer Variante müssen wir die Ursprungsstellung wiedergewinnen. Wie dies mit einem Stack gelöst wird, wurde in 3.1.3 erläutert. Für den Umgang mit dem Stack haben wir zwei Hilfsroutinen geschrieben: die eine ("Stellung im Stack retten" speichert die rekursiven Variablenwerte im Stack, die andere ("Zug rückgängig machen") setzt sie wieder ein.

3.7 Ein-Ausgabe-Routinen

Jedes Programm benötigt Ein- und Ausgabe. Unser Schachprogramm muß Züge, Felder und Stellungen einlesen und ausdrucken können.

Aus zwei Gründen ist bei unserem Programm Ein- und Ausgabe etwas vernachlässigt worden: Das Programm soll kompatibel sein, also auf jedem Rechner mit BASIC laufen, und das eigentliche Thema dieses Buchs ist Schachprogrammierung, und nicht Graphik.

Daher benutzen wir für die Ausgabe nur den normalen Zeichensatz.

In Kapitel 4 werden die technischen Aspekte von Ein- und Ausgabe diskutiert; hier lassen wir es bei einer Aufzählung der Routinen.

Ausgaberoutinen:

Feld ausgeben
Figur ausgeben
Zug ausgeben
Brett ausgeben

Eingaberoutinen:

Feld einlesen
Figur einlesen
Stellung einlesen

3.8 Die Partieverwaltung

Mit dem Zuggenerator, dem Such- und Entscheidungsalgorithmus und der Bewertungsfunktion haben wir die wichtigsten Teile eines Schachprogrammes - zum Teil ausführlich - kennengelernt. Aber mit diesen Verfahren allein können wir wenig anfangen. So müssen wir zunächst die *Stellung einlesen*, mit der die Arbeit begonnen werden soll. Aber auch das reicht noch nicht aus. Ein solches Programm würde die Position einlesen, den Suchprozeß anstarten, der sich mit Hilfe des Zuggenerators und der Stellungsbewertung für einen Zug entscheiden würde, und schließlich diesen Zug bekanntgeben und den Programmlauf beenden. Um tatsächlich eine Partie Schach spielen zu können, benötigen wir noch einen Organisationsprozeß, der abwechselnd vom Programm einen Zug bestimmen läßt und vom (menschlichen) Gegner einen Zug einliest. Diese Züge müssen natürlich auch auf dem Brett ausgeführt, d.h., die vorliegende Stellung muß aktualisiert werden. Diese letztgenannten Aufgaben und einige weitere nimmt die sogenannte *Partieschleife* wahr. Infolgedessen beschäftigen wir uns in den beiden letzten Abschnitten des Kapitels 3 mit der Stellungseingabe und der Partieschleife.

3.8.1 Die Stellungseingabe

Der erste Schritt überhaupt im Ablauf des Programmes ist das Einlesen der Stellung, mit der die Partie begonnen werden soll. Diese und alle im partieverlauf folgenden Stellungen werden im Rechner durch entsprechende Datenstrukturen dargestellt. Zu diesen gehören die Informationen über die Figurenstandorte, das Anzugsrecht, momentane En-Passant-Schlagmöglichkeiten und noch vorhandene Rochaderechte. Die Stellungseingabe besteht also darin, den betreffenden einfachen Variablen und Feldern Anfangswerte zuzuweisen, die in eindeutiger Weise die Position kennzeichnen. Wegen der Beziehung zwischen der (rechnerexternen) Stellung und ihrer (rechnerinternen) Repräsentation spricht man auch von *Stellungsinitialisierung*.

Zwei Möglichkeiten der Initialisierung wollen wir in unserem Programm zulassen: den Start mit der *Grundstellung* des Schachspiels sowie mit *frei wählbaren Positionen*, die vom Benutzer Figur für Figur eingegeben werden müssen, um z.B. eine abgebrochene Partie fortzusetzen. Die Art der Eingabe ist in der Regel von den verwendeten Datenstrukturen vollkommen unabhängig und richtet sich nur nach dem Maß an Bedienungskomfort, den der Programmierer in Relation zum Programmieraufwand anbieten will. Da dieses Programm in erster Linie als Demonstrationsobjekt gedacht ist - auch wenn es durchaus möglich und sogar erwünscht ist, die Spielstärke des Programmes in vollständigen Partien zu testen -, wird die Handhabung natürlich nicht so ausgefeilt sein, wie man es von kommerziellen Produkten gewohnt ist. Ein weiterer Grund ist der zur Verfügung stehende Speicherplatz: Solche Eingabehilfsmittel können meistens nur durch recht umfangreiche Unterprogramme realisiert werden, die entweder den wichtigeren Programmteilen wie der Bewertungsfunktion oder der Zugsortierung Speicherplatz wegnehmen oder zwischendurch, während das Programm einen Zug ermittelt, ausgelagert werden müssen und damit das Programm langsamer machen.

Um zwischen einem Partiebeginn mit der Grundstellung und mit anderen Positionen zu unterscheiden, ist die erste Eingabe des

Benutzers erforderlich. Hat er sich für die Grundstellung entschieden, läuft die Initialisierung der Datenstrukturen automatisch ab, ohne daß der Benutzer noch einmal aktiv werden muß. Dies ist möglich, weil zur Vorbesetzung der Variablen und Felder ausschließlich konstante Werte verwendet werden können. Im anderen Fall fordert der Rechner die Standorte der einzelnen Figuren, das Anzugsrecht und eventuelle Rochademöglichkeiten vom Benutzer an. Dazu einige Bemerkungen:

Bei der Eingabe der Figurenstandorte gibt es viele Möglichkeiten, Fehler zu machen, die zu Stellungen führen, die in einer korrekten Partie niemals auftauchen können: Eingabe von mehreren Königen oder gar keinem König für eine Farbe, mehr als 16 Figuren für eine Seite, 9 gleichfarbige Bauern, Bauern auf der Grundreihe und vieles mehr. Im Prinzip ist es die Aufgabe der Einleseroutine, solche Fehler zu erkennen und den Benutzer darauf hinzuweisen. Dies würde aber das Programm ziemlich aufblähen, und so beschränkt man sich meistens darauf, einige der wichtigeren Fälle zu behandeln. In unserem Programm sind dies: Königsanzahl, Gesamtanzahl der Figuren pro Spieler sowie die Überprüfung der Feldangaben daraufhin, ob das Feld überhaupt zum Brett gehört (Zurückweisung von z.B. "B9") bzw. ob es noch nicht durch eine andere Figur belegt ist. Die Eingabe selbst werden wir so durchführen, daß jeweils ein Figurentyp vom Rechner angezeigt wird und der Benutzer nur noch den Standort für eine Figur dieses Typs eingeben muß. Befindet sich in der einzulesenden Stellung keine (weitere) Figur des angezeigten Typs auf dem Brett, kann durch die leere Eingabe zum nächsten Typ weitergeschaltet werden.

Die Rochademöglichkeiten werden nur abgefragt, wenn die jeweils beteiligten Könige und Türme noch auf ihren ursprünglichen Feldern stehen. Der Benutzer entscheidet dann darüber, ob diese Figuren im imaginären vorausgegangenen Partieverlauf schon gezogen wurden.

Auf eine Angabe von e.p.-Schlagmöglichkeiten wurde verzichtet, da diese Fälle beim Partieabbruch vermieden werden können.

3.8.2 Die Partieschleife

Die wichtigste Aufgabe dieses Programmteils ist es, abwechselnd vom Suchverfahren bzw. dem Gegner einen Zug anzufordern und diese auszuführen. Noch bevor dies geschehen kann, muß allerdings die Farbverteilung für die Partie vorgenommen werden. Dazu wird der Benutzer gefragt, mit welcher Farbe er spielen möchte. Des weiteren muß der Benutzer festlegen, wie tief die Suche die einzelnen Varianten analysieren soll. Sinngemäß gehören diese beiden Abfragen zur Steuerung des Partieverlaufes und werden daher an dieser Stelle aufgeführt. Da sie aber andererseits eng mit der Initialisierung der Datenstrukturen zusammenhängen, sind sie programmtechnisch dem Unterprogramm zur Stellungseingabe zugeordnet. Die Behandlung der Partiezüge geschieht zweckmäßigerweise in einer Schleife, die aus zwei Teilen (einem für die Züge des Programmes und einem für die gegnerischen Züge) besteht. Durch das Anzugsrecht und die Farbverteilung wird bestimmt, bei welchem Teil die Partie aufgenommen wird; anschließend werden beide Teile immer abwechselnd ausgeführt.

Der Teil für die Programmzüge hat folgendes Aussehen. Zunächst wird der Such- und Entscheidungsalgorithmus aufgerufen, der für die vorliegende Stellung den zu spielenden *Zug auswählen* soll. Hat dieser eine Fortsetzung bestimmt, so findet die Partiesteuerung das Ergebnis in der aktuellen Hauptvariante, dessen erster Zug dem Programm den nach dem Minimax-Prinzip bestmöglichen Wert sichert. Dieser Zug wird ausgeführt, indem die Stellung aktualisiert wird (in den Datenstrukturen und auf dem Bildschirm), und es geht mit dem anderen Teil der Partieschleife, dem für die Züge des Gegners, weiter. Einzige Ausnahme: Das Programm setzt matt. In diesem Fall wird nach der Ausführung des Zuges der Programmlauf beendet.

Es kann aber auch vorkommen, daß kein bester Zug geliefert wurde; nämlich dann, wenn wegen einer Matt- oder Pattsituation gar kein Zug spielbar ist. Der Sachverhalt von Mattstellungen ist an dem von der Suche berechneten Minimax-Wert abzulesen. Dies ist durch das Bewertungsschema (s. Abschnitt 3.4) gewährleistet. Das Patt muß dagegen auf anderem Wege erkannt

werden, da der Wert 0 für jede als ausgeglichen eingeschätzte Position steht. Für den Sonderfall des Patt wird daher eine extra eingeführte (logische) Variable benutzt, die während der Suche auf den Wert -1 (= wahr) gesetzt wird, wenn eine Pattsituation vorliegt. Diese Matt- und Pattstellungen führen genauso zum Programmende wie die Positionen, in denen das Programm mattsetzt; in diesen Fällen natürlich ohne Ausführung eines Zuges.

Ein Punkt, der in der Partieverwaltung unseres Programmes nicht realisiert wurde, der aber in Programmen, die in Schachturnieren eingesetzt werden sollen, unbedingt berücksichtigt werden muß, ist die *Behandlung von Remisangeboten* des Gegners. Die Entscheidung über die Annahme oder Ablehnung eines solchen Angebotes wird zumeist in Abhängigkeit von der mit dem besten Zug gelieferten Bewertung getroffen. Sie fällt dann z.B. positiv aus, wenn das Programm mit den weißen Steinen spielt und die Suche mit einem Wert kleiner 0 abschließt.

Der zweite Teil der Partieschleife besteht aus der *Eingabe der Züge* des Gegners und deren Überprüfung, ob sie in der gegebenen Stellung spielbar sind. Dazu gehört zum einen der Test auf eine korrekte Eingabeform (Angabe von Ursprungs- und Zielfeld sowie bei Bauernumwandlungen die Angabe der eingewechselten Figur). Zum anderen muß sichergestellt sein, daß der Zug tatsächlich in der Menge der legalen Zugmöglichkeiten enthalten ist. Dies bedeutet, daß zunächst mittels des Zuggenerators alle existierenden Zugmöglichkeiten berechnet werden müssen und der eingegebene Zug in ihnen enthalten sein muß. Ist dies nicht der Fall, wird der Benutzer nach einer Fehlermeldung erneut zur Zugeingabe aufgefordert. Anderenfalls wird der Zug ausgeführt und die resultierende Stellung daraufhin untersucht, ob der König auch nicht geschlagen werden kann. Bei einer vorhandenen Schlagmöglichkeit (z.B. nach einem Zug mit einer gefesselten Figur) muß der Zug rückgängig gemacht und die Zugeingabe wiederholt werden. Ansonsten wird die neue Stellung auf dem Bildschirm gezeigt und die Partieschleife wechselt zur oben beschriebenen Behandlung der Züge des Programmes.

4. Das Programm

In diesem Abschnitt wird das komplette Schachprogrammlisting angegeben und kommentiert. Beim Beschreiben richten wir uns nicht allein nach der Reihenfolge der BASIC-Zeilennummern, sondern auch nach der Struktur. Wir orientieren uns daran, wie es erstellt bzw. eingetippt werden könnte, wie es also entwickelt wurde.

Abschnitt 4.1 gibt zunächst einen groben Überblick zum Programm. Die wesentlichen Datenstrukturen und Moduln werden angegeben.

Dem schließt sich die Erläuterung der grundlegenden Datenstrukturen (für Stellung, Brett, Figuren, Züge und Zuglisten) an. Ohne diese bleibt das Programm unverständlich.

Die folgenden Kapitel beschreiben jeweils einen inhaltlich zusammenhängenden Bereich des Programmlistings, ein Modul. Zuerst werden immer die Datenstrukturen und ihre Bedeutung, dann die Routinen und Abläufe dargestellt.
Zum Schluß stellen wir eine alphabetisch geordnete Liste aller Variablen zur Verfügung. Diese enthält Verweise auf den Abschnitt, in dem die Variablen beschrieben werden.

Es wurde schon erwähnt, daß ein Schachprogramm in Assembler um ein Vielfaches schneller ist. Dies liegt unter anderem daran, daß die Ausdrucksmöglichkeiten in Maschinensprache mächtiger sind. So wird für bestimmte Werte eigentlich nur 1 Byte benötigt, BASIC bietet hierfür keinen Variablentyp an. Derartige Werte verbrauchen nicht nur weniger Speicherplatz, sie können auch schneller bearbeitet werden. Um eine Umstellung auf Assembler zu erleichtern, finden Sie in der Variablenliste Angaben über den wirklich benötigten Speicherplatz.

Bemerkung: Dieses Kapitel überschneidet sich teilweise mit dem vorherigen. Hier werden nicht mehr die Ideeen, sondern deren Umsetzung in ein Programmm besprochen.

Bermerkung zu BASIC: Dieses Programm wurde auf einem Commodore entwickelt. Wir benutzten dabei den Compiler "BASIC 64". Für diesen wurden einige Kommentarzeilen eingefügt, die er als Compiler-Direktive auffaßt. Sie beginnen alle mit "REM@". Weiterhin haben wir auf die STEP-Anweisung verzichtet.
Um das Programm kompatibel zu allen Home-Computern mit BASIC zu halten, benutzen wir bei Arrays nie den Index 0.

4.1 Datenstrukuren und Module im Überblick

a) Datenstrukturen

Das Programm enthält folgende Datenstrukturen:

- Stellungdarstellung
  - Brettdarstellung
    - Figurenarten
    - Brett
    - Figurenliste
    - Anzahl der Figuren
    - Verweislisten: Figuren - Brett
  - Zusatzinformationen zur Stellung
    - Anzugsrecht
    - Rochaderechte
    - En-Passant-Informationen
    - König-im-Schach-Information
- Zugliste
  - Arrays für Züge
  - Zuglistenverwaltung
- Zuginformationen
  - Standort der ziehenden Figur
  - Zielfeld des Zuges
  - Art der ziehenden Figur

ggf. Art der geschlagenen Figur
ggf. Art der bei Bauernumwandlung entstehenden Figur
Art des Zuges
Stack
Variablen für Suche
Variablen für Bewertung
Hilfsvariablen
Variablen für Partieschleife

Die Stellungsdarstellung ist von grundlegender Bedeutung, sie wird daher am Anfang besprochen. Das gleiche gilt für die Zuginformationen. Die übrigen Datenstrukturen werden in Zusammenhang mit der Benutzung erläutert.

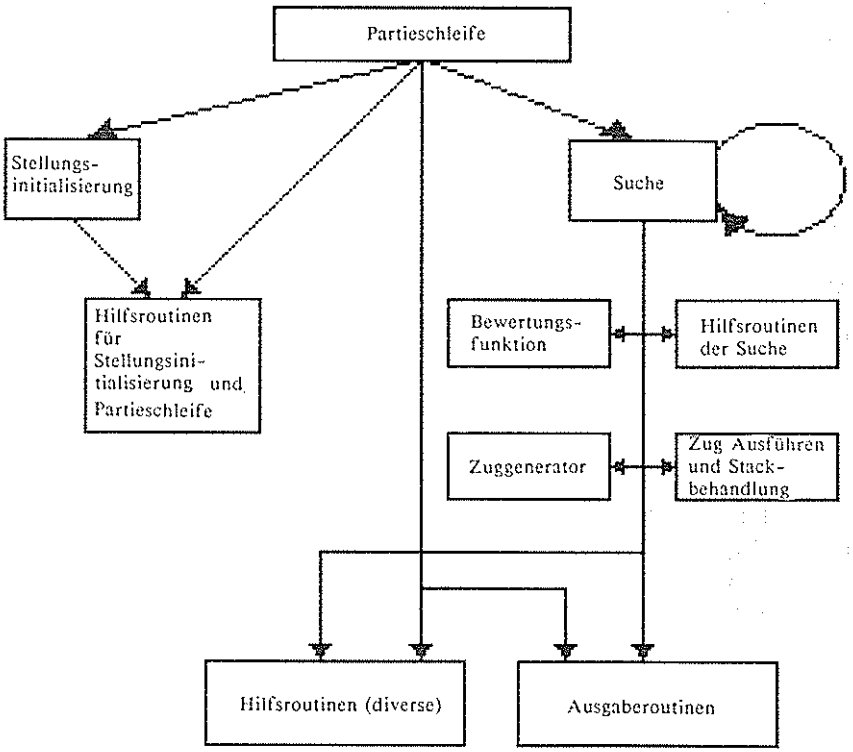
b) Die Moduln

Das Programm besteht aus folgenden Moduln:

Deklarationen (0-599)
Partieschleife (600-998)
Zuggenerator (999-1999)
Bewertungsfunktion (1999-3999)
Zug intern ausführen und Stackbehandlung (4000-4999)
Hilfsroutinen für Suche (5000-5999)
Suche (6000-6999)
Hilfsroutinen (diverse) 8000-8999)
Stellungsinitialisierung (9000-9299)
Hilfsroutinen für Stellungsinitialisierung und Partieschleife (9300-9999)
Ausgaberoutinen (11000-11999)

Hinter den jeweiligen Programmteilen stehen die ihnen zugeordneten Zeilennummerbereiche im Programm.

Die Aufrufstruktur ist:



4.2 Die grundlegenden Datenstrukturen

4.2.1 Stellungsdarstellung

Alle Variablen (nicht die Konstanten!) der Stellungsdarstellung sind rekursiv (vgl. 3.1.3). Wenn eine Variante fertig analysiert ist, muß die ursprüngliche Stellung wieder korrekt dargestellt sein - die entsprechenden Werte müssen also gespeichert werden. Daher sind diese Variablen in Bezug auf die Suche rekursiv.

a) Brettdarstellung

Das Programm muß das Brett und die Figuren intern darstellen. Die entsprechenden Variablen werden hier beschrieben.
Wichtig ist, daß wir zwischen Figur und Figurenart unterscheiden. So ist "weißer Läufer" zum einen eine Figurenart, andererseits können zwei derartige Steine auf dem Brett sein, eben zwei Figuren von der Art "weißer Läufer".
Den Figurenarten sind eindeutige Werte zugeordnet. Um das programm lesbarer zu machen, wurden auch Konstanten für die verschiedenen Arten eingeführt. Die entsprechenden Variablen und Werte sind:

Variable	Wert	Figurenart
sk%	1	schwarzer König
sd%	2	schwarze Dame
st%	3	schwarzer Turm
sl%	4	schwarzer Läufer
ss%	5	schwarzer Springer
sb%	6	schwarzer Bauer
ff%	7	freies, unbesetztes Feld
wb%	8	weißer Bauer
ws%	9	weißer Springer
wl%	10	weißer Läufer
wt%	11	weißer Turm
wd%	12	weißer Dame
wk%	13	weißer König
kf%	14	unbetretbares Feld (Randfeld, vgl.3.3.3)

(kf% steht für "kein Feld".) Diese Variablen werden im ganzen Programm nicht verändert, sie werden daher wie Konstanten behandelt.

Auf die Brettdarstellung wurde in Abschnitt 3.3 eingegangen. Wir benutzen also die eindimensionale eingebettete Darstellung. Die Variable ist (mit Zeilenangabe):

```
162 DIM BR%(144)
```

Sie stellt das erweiterte Brett dar, jedem Spielfeld entspricht ein Element von br%. Dessen Wert gibt an, was für eine Figur dort steht. Die durch die Erweiterung hinzugekommenen Felder (und nur diese) sind durch den Wert kf% (=14) markiert.
Zusätzlich führen wir Listen der Figuren (nicht der Figurenarten). Für jede der maximal 2 mal 16 auf dem Brett befindlichen Figuren enthalten sie Art und Standort:

```
166 DIM FI%(2,16)
```

(fi% für Figurenliste.) Der erste Index ist für die beiden Spieler (1=Weiß,2=Schwarz), der zweite für die einzelnen Figuren. Inhalt ist die Figurenart. für "kein Eintrag" steht der Wert 7 (=ff%).

fi%(1,1) enthält immer den weißen König (fi%(1,1)=wk%=13), fi%(2,1) den schwarzen.

Für den Standort haben wir analog:

```
166 DIM FS%(2,16)
```

Wie fi%, nur daß hier die Standorte der Figur gespeichert sind. Die 0 ist der Wert des leeren Eintrags. fs% steht für "Figuren-Standort".

Zusätzlich zu den Figurenlisten merken wir uns in

```
166 DIM FZ%(2),
```

(fz% für Figuren-Zahl), wieweit die Listen überheupt gefüllt sind. fz%(1) gibt den letzten Eintrag in den Figurenlisten für Weiß an, fz(2) für Schwarz.

Manchmal muß das Programm wissen, welche Figur auf einem vorgegebenen Feld steht. Hierzu dient

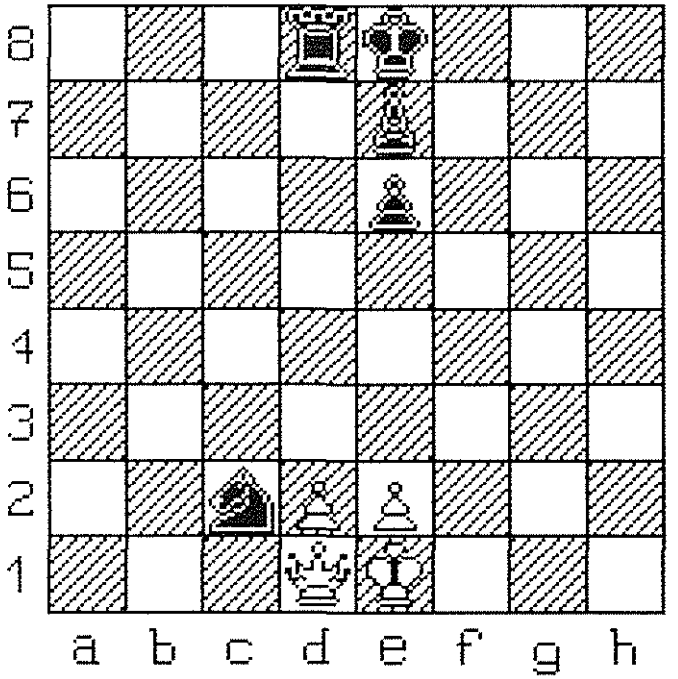
```
162 DIM FA%(118)
```

(fa% für Figuren-Adresse.) Bei einem vorgegebenen Feld f finden wir die dort stehende Figur entweder in fi%(1,fa%(f))

oder in fi%(2,fa%(f)), je nachdem, ob es eine weiße oder schwarze ist. Steht auf einem Feld f keine Figur, so hat fa%(f) den Wert 0. fa% benötigt im Gegensatz zu br% nur 118 Einträge, da es nur für Felder des Bretts benutzt wird. Die Indizes über 118 stehen ja für Felder jenseits des richtigen Schachbretts.

b) Ein Beispiel

Betrachten wir zur Illustration folgende Stellung:



Dann ist br% wie folgt belegt:

	14	14	14	14	14	14	14	14	14	14	14	14
	14	14	14	14	14	14	14	14	14	14	14	14
8	14	14	7	7	7	3	1	7	7	7	14	14
7	14	14	7	7	7	7	4	7	7	7	14	14
6	14	14	7	7	7	7	6	7	7	7	14	14
5	14	14	7	7	7	7	7	7	7	7	14	14
4	14	14	7	7	7	7	7	7	7	7	14	14
3	14	14	7	7	7	7	7	7	7	7	14	14
2	14	14	7	7	5	8	8	7	7	7	14	14
1	14	14	7	7	7	12	13	7	7	7	14	14
	14	14	14	14	14	14	14	14	14	14	14	14
	14	14	14	14	14	14	14	14	14	14	14	14

a b c d e f g h

Der Wert 7 steht für freies, leeres Feld; den Wert 14 enthalten alle Felder außerhalb des richtigen Bretts.

fi% und fs% haben folgende Belegung:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	13	12	8	8	7	7	7	7	7	7	7	7	7	7	7	7
2	1	3	5	4	6	7	7	7	7	7	7	7	7	7	7	7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	31	30	42	43	0	0	0	0	0	0	0	0	0	0	0	0
2	115	114	41	103	91	0	0	0	0	0	0	0	0	0	0	0

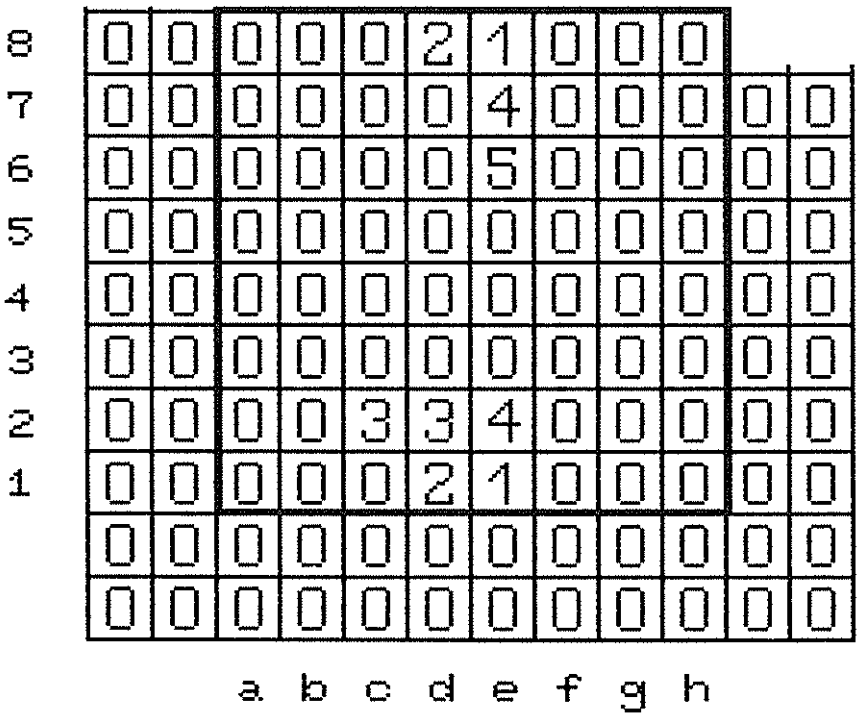
1	4
2	5

Die Einträge in fs% geben natürlich die interne Darstellung der Standorte an.

Dabei gibt fz% an, wieweit die Listen fi% und fs% überhaupt gefüllt sind.

Wenn wir wissen wollen, welche Art die dritte Figur von Weiß hat, können wir dies in fi%(1,3) sehen. Wo sie steht, teilt fs%(1,3) mit. Die schwarzen Figuren sind in fi%(2,1) bis fi%(2,5) bzw. in fs%(2,1) bis fs%(2,5) beschrieben.

fa% ist so gesetzt:



Würde in unserer Beispielstellung der Zug Dc2: ausgeführt, also der schwarze Springer geschlagen, so wird br%(41) auf 12 (=wd%) gesetzt (die 41 entspricht dem Feld C2), und fi%(2,3) erhielte den Wert 7 (=ff%). Dies drückt aus, daß der Springer nicht mehr vorhanden ist. Die Werte fa%(41) und fs%(2,3) sind fortan belanglos, können aber sicherheitshalber auf 0 gesetzt werden. Der neue Standort der weißen Dame würde in fs%(1,2) eingetragen. fz% bleibt unverändert, da das Komprimieren der Materiallisten zu umständlich wäre.

Des weiteren müßte in fs% der neue Standort der weißen Dame eingetragen werden.

Die Datenstrukturen enthalten einige Redundanz, alles ist doppelt gespeichert. Dies ist dennoch sinnvoll, da diese Darstellung häufig einen effizienteren Zugriff zu den gewünschten Informationen erlaubt.

c) Zusatzinformationen

Die Darstellung der Figurenstandorte reicht nicht aus, um eine Stellung eindeutig zu beschreiben. Zusätzlich benötigen wir:

w%: gibt an, ob Weiß am Zug ist (logische Variable).

dim ro%(2,2): gibt an, wer noch wohin rochieren darf:

ro%(1,1) = lange Rochade Weiß
ro%(1,2) = kurze Rochade Weiß
ro%(2,1) = lange Rochade Schwarz
ro%(2,2) = kurze Rochade Schwarz

Bekanntlich kann ein König nicht mehr rochieren, wenn er einmal gezogen hat oder wenn der entsprechende Turm bewegt wurde. Ob dies geschehen ist, gibt das logische Array ro% für die vier Rochademöglichkeiten an. Wie immer entspricht der Wert -1 dem "ja", 0 dem "nein".

ep%: hat normalerweise den Wert 0. Ist jedoch im letzten Zug ein Bauer zwei Felder vorgerückt, enthält ep% dessen Zielfeld.

Ein En-Passant-Schlagzug ist genau dann möglich, wenn im Vorgängerzug ein Bauer zwei Felder vorging und nun neben einem gegnerischen Bauern steht. Dies erkennt das Programm mit Hilfe von ep%.

sh%: Die Suche muß wissen, ob der König in der aktuellen Stellung im Schach steht. Dies gibt die logische Variable sh% an.

4.2.2 Zugdarstellung

Züge werden im Schachprogramm laufend benutzt. Daher haben wir einen Satz von Variablen eingeführt, der den jeweils aktuellen Zug beschreibt:

s%:	Standort der ziehenden Figur
z%:	Zielfeld der ziehenden Figur
zf%:	Art der ziehenden Figur
f%:	ggf. Art der geschlagenen Figur
nf%:	ggf. Art der neu entstehenden Figur bei Bauernumwandlungen
za%:	Art des Zuges
zg%:	Veränderung der Materialbalance durch den Zug

Diese Variablen beschreiben zusammen einen Zug. Häufig wird nur ein Teil benötigt, im Zuggenerator z.B. meist nur s% und z%.

f% und nf% werden nur bei Schlagzügen oder Bauernumwandlungen benutzt.

Wir unterscheiden 6 Arten von Zügen (za% für Zug-Art):

Art	Wert
normaler Zug	20
Rochade	21
En-Passant-Schlagen	22
Schlagzug	23
Bauernumwandlung	24
"illegaler Zug"	0

Ist ein Zug sowohl Bauernumwandlung als auch Schlagzug, so ist seine Art "Bauernumwandlung".

Bei Schlagzügen und Umwandlungen ändert sich die Materialbalance. Dies läßt sich leicht berechnen, zg% (für Zugewinn) gibt die Differenz an.

Die Variablen werden von mehreren Programmteilen zu verschiedenen Zwecken benutzt (Zuggenerator, Suche/Suchschleife, Partieschleife, Bewertungsfunktion). Es muß unbedingt darauf geachtet werden, daß die Benutzungsstellen sich zeitlich nicht überlappen, oder daß die Werte (wie bei rekursiven Variablen) im Stack gerettet und rechtzeitig wieder bereitgestellt werden.

4.2.3 Die Zuglisten

Wie in Abschnitt 3.4.8 schon angedeutet, enthält unser Programm getrennte Zuglisten für Schlagzüge und Bauernumwandlungen einerseits und normale Züge andererseits, da diese Züge verschieden behandelt werden.

Die Züge werden in Arrays gespeichert:

```
168 DIM Z1%(400,2), Z2%(100,3)
```

z1% ist die Liste für normale Züge. Sie kann 400 Züge aufnehmen; jeder Zug wird durch Standort (z1%(i,1)) und Zielfeld (z1%(i,2)) charakterisiert. Die kleinere Liste z2% benötigt je Zug eine zusätzliche Angabe, da Bauernumwandlungen sonst noch nicht eindeutig bestimmt sind. Hier muß in z2%(i,3) noch die entstehende neue Figur beschrieben werden. Bei normalen Schlagzügen ist dieser Wert ohne Belang.

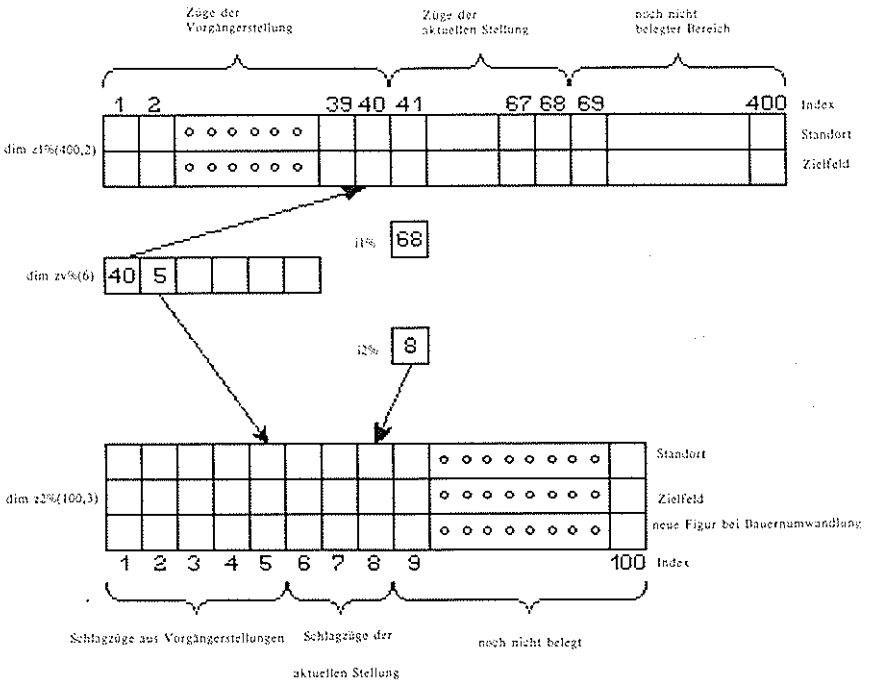
In den beiden Listen wird also nur ein Minimum an Information gespeichert, gerade soviel, daß der Zug eindeutig angegeben ist. Dies hat zwei Vorteile: beim Generieren der Züge müssen weniger Informationen berechnet und abgespeichert werden, und wir sparen Speicher. Später, wenn dies erforderlich ist, können die weiteren Eigenschaften des Zuges immer noch bestimmt werden - und für viele Züge ist dies nicht notwendig.

Die großen Zuglisten müssen verwaltet werden. Sie werden von links nach rechts, also von Index 1 an, gefüllt. Die Variablen i1% und i2% geben den Füllungsgrad an - i1% zeigt auf den höchsten benutzten Index in z1%, i2% auf den von z2%.

Die Arrays speichern gleichzeitig die Züge mehrerer Stellungen. Analysiert das Programm von der Grundstellung des Schachspiels an die Variante e2-e4, e7-e5, so enthalten sie die Zuglisten von drei Stellungen: der Grundstellung, der Stellung nach e2-e4 und der Stellung nach e2-e4, e7-e5. Um diese unterscheiden zu können, wurde die Zuglistenverwaltung

```
172 DIM ZV%(6)
```

eingeführt. Bevor der Zuggenerator den ersten Zug bestimmt, speichert er in zv%(1) und zv%(2) die aktuellen Werte von i1% und i2% ab. Damit geben diese beiden Komponenten von zv% zusammen mit i1% und i2% an, von wo bis wo die aktuellen Züge (nach dem Generieren) eingetragen sind: in z1% von z1%(zv%(1)+1,...) bis z1%(i1%,...) und in z2% von z2%(zv%(2)+1,...) bis z2%(i2%,...).



Die Bedeutung der anderen Komponenten von zv% wird später erläutert.

Auch zv%, i1% und i2% sind rekursive Variablen - nach der Untersuchung einer Stellung müssen sie wieder auf den Wert für die Vorgängerstellung zurückgesetzt werden.

4.3 Deklarationen (1-599)

Die Deklarationen befinden sich auf den Zeilen 100 bis 224. "Deklarationen" ist ein Begriff aus den höheren Programmiersprachen. Damit ist gemeint, daß Variablen, bevor sie benutzt werden, zunächst an einer Stelle beschrieben werden, so daß später klar ist, was sie bedeuten.

BASIC kennt dies nur für Arrays. Hier muß in einer dim-Anweisung die Größe angegeben werden.

In höheren Programmiersprachen kann man auch Konstanten deklarieren. Für bestimmte Werte werden also Namen vergeben. In BASIC sind Konstanten nicht vorgesehen, man kann normale Variablen aber auf diese Art benutzen: zu Programmbeginn wird ihnen ein Wert zugeordnet, und dieser wird nie verändert. Damit sind diese Variablen konstant.

In unserem Deklarationsteil werden nun alle Arrays und Konstanten definiert. Daher auch die data-Zeilen: mit denen wird den Konstanten der Wert zugewiesen.

```
10 PRINT"            DEMOSCHACH"
11 PRINT
12 PRINT"   (C)1985 G.SCHRUEFER,H.-J.KRAAS"
100 REM KONSTANTEN
102 SK%=1:SD%=2:ST%=3:SL%=4:SS%=5:SB%=6
104 WB%=8:WS%=9:WL%=10:WT%=11:WD%=12
106 WK%=13:FF%=7:KF%=14
108 DIM FW%(12)
110 FOR I=2 TO 12:READ FW%(I):NEXT I
118 DIM SW%(8),DW%(8)
120 FOR I=1 TO 8:READ SW%(I):NEXT I
124 FOR I=1 TO 8:READ DW%(I):NEXT I
128 DIM KO% (118)
130 FOR I=24 TO 108
132 FOR J=3 TO 10:KO%(I+J)=20:NEXT J
134 I=I+11:NEXT I
136 FOR I=48 TO 84
138 FOR J=5 TO 8:KO%(I+J)=KO%(I+J)+40:NEXT J
140 I=I+11:NEXT I
142 FOR I=60 TO 72
144 FOR J=6 TO 7:KO%(I+J)=KO%(I+J)+40:NEXT J
146 I=I+11:NEXT I
150 DIM KV%(10)
152 FOR I=3 TO 10:READ KV%(I):NEXT I
155 WS$="W=WEISS,S=SCHWARZ"
```

```
156 ZK$="WEISS:  SCHWARZ:KOENIG  DAME    TURM    LAEUFER SPRINGERBAUER
"
158 BL$="! . . . . "
160 REM VARIABLEN (FELDER)
162 DIM BR%(144),FA%(118)
164 DIM RO%(2,2)
166 DIM FI%(2,16),FS%(2,16),FZ%(2)
168 DIM Z1%(400,2),Z2%(100,3)
170 DIM ST%(8,44)
172 DIM ZV%(6)
174 DIM HV%(4,4,3)
176 DIM KI%(8,4)
178 DIM SI%(3)
180 DIM BF%(2,11,6),BZ%(2,11)
182 DIM BV%(2,10),FB%(2,10),RB%(2,10)
184 DIM ES%(2),RF%(2),MS%(2)
188 DIM A0%(2)
200 REM DATA-ZEILEN
202 DATA -900,-500,-325,-325,-100,0,100,325,325,500,900
204 DATA 10,23,25,14,-10,-23,-25,-14
206 DATA 11,13,-11,-13,1,12,-1,-12
208 DATA 0,0,390,540,700,230,0,0
210 DATA 13,31,12,30,11,27,11,34,10,29,10,32,9,28,9,33
212 DATA 8,39,8,40,8,41,8,42,8,43,8,44,8,45,8,46
214 DATA 1,115,2,114,3,111,3,118,4,113,4,116,5,112,5,117
216 DATA 6,99,6,100,6,101,6,102,6,103,6,104,6,105,6,106
218 DATA 11,3,9,7,10,5,12,2,13,1,10,6,9,8,11,4
220 DATA 8,9,8,10,8,11,8,12,8,13,8,14,8,15,8,16
222 DATA 6,9,6,10,6,11,6,12,6,13,6,14,6,15,6,16
224 DATA 3,3,5,7,4,5,2,2,1,1,4,6,5,8,3,4
```

Kommentare:

102-106:	Konstanten für Figurenarten
108-110:	Konstanten für Figurenwerte
118-124:	Konstanten für Zuggenerator
128-152:	Konstanten für Bewertungsfunktion
155-158:	Konstanten für Ausgaberoutinen

162-166:	Stellungsdarstellung
168:	Zuglisten
170:	Stack
172:	Zuglistenverwaltung
174-178:	Hauptvariante, Killerliste, Protokoll (Suche)
180-184:	Variablen der Bewertungsfunktion
188:	Zugausgabe
200-224:	DATA-Zeilen für Konstanten-Werte:
202:	Figurenwerte
204:	Springerweiten/Distanzen
206:	Damenweiten/Distanzen
208:	Vorrückbonus (für die Bauernbewertung)
210-224:	Werte der Stellungsdarstellung für die Grundstellung

4.4 Initialisieren und Einlesen der Partiestellung

```
9000 REM INITIALISIEREN DER ANFANGSSTELLUNG
9012 REM BRETT UND FIGURENADRESSEN LOESCHEN
9013 FOR I=1 TO 24:BR%(I)=KF%:NEXT I
9014 FOR I=24 TO 108
9015 FOR J= 1 TO  2:BR%(I+J)=KF%:NEXT J
9016 FOR J= 3 TO 10:BR%(I+J)=FF%:NEXT J
9017 FOR J=11 TO 12:BR%(I+J)=KF%:NEXT J
9018 I=I+11:NEXT I
9019 FOR I=121 TO 144:BR%(I)=KF%:NEXT I
9020 FOR I=1 TO 118:FA%(I)=0:NEXT I
9021 REM MATERIALLISTEN LOESCHEN
9022 FOR I=1 TO 2
9023 FOR J=1 TO 16:FI%(I,J)=FF%:FS%(I,J)=1:NEXT J
9026 NEXT I
9027 REM STELLUNGSEINGABE
9028 PRINT "WAEHLEN SIE DIE ANFANGSSTELLUNG"
9029 PRINT "G=GRUNDSTELLUNG, SONST BELIEBIG"
9031 ES$="":INPUT ES$
9032 IF ES$<>"G" THEN GOTO 9058
9033 REM ANFANGSSTELLUNG=GRUNDSTELLUNG
9034 FOR I=1 TO 2
```

```
9035 FOR J=1 TO 16:READ FI%(I,J),FS%(I,J):NEXT J
9038 FZ%(I)=16
9039 NEXT I
9040 FOR I= 27 TO  34:READ BR%(I),FA%(I):NEXT I
9041 FOR I= 39 TO  46:READ BR%(I),FA%(I):NEXT I
9042 FOR I= 99 TO 106:READ BR%(I),FA%(I):NEXT I
9043 FOR I=111 TO 118:READ BR%(I),FA%(I):NEXT I
9044 MS%(1)=4000:MS%(2)=-4000:MB%=0
9045 W%=-1:EP%=0
9046 FOR I=1 TO 2
9047 FOR J=1 TO 2:RO%(I,J)=-1:NEXT J
9050 NEXT I
9051 GOTO 9202
9056 REM STELLUNGSEINGABE UEBER TASTATUR
9057 REM DATEN WEGLESEN
9058 FOR J=1 TO 128:READ A1%:NEXT J
9062 PRINT "GEBEN SIE DIE FIGURENSTANDORTE EIN"
9065 PRINT "(FELDANGABEN IN BRETTNOTATION (A1-H8))."
9067 PRINT "MIT DER LEEREN EINGABE "
9068 PRINT "(<ENTER> BZW. <RETURN> DRUECKEN)"
9069 PRINT "WECHSELN SIE ZUM NAECHSTEN FIGURENTYP."
9070 REM EINGABE DER FIGUREN
9072 F%=13: A5%=-1
9074 FOR I=1 TO 2
9076 FZ%(I)=0:MS%(I)=0
9078 PRINT MID$(ZK$,(I-1)*8+1,8)
9080 FOR K=2 TO 7
9082 ES$="":PRINT MID$(ZK$,K*8+1,8);:INPUT ES$
9084 GOSUB 9900
9086 IF S%=0 THEN GOTO 9094
9088 IF S%=1 THEN K=K-1:GOTO 9096
9090 GOSUB 9920
9092 IF (K>2) OR A1% THEN K=K-1:GOTO 9096
9094 F%=F%+A5%
9096 NEXT K
9098 F%=1:A5%=1
9100 NEXT I
9102 MB%=MS%(1)+MS%(2)
9154 REM EINGABE DES ANZUGRECHTS
9155 PRINT"WELCHER SPIELER IST AM ZUG?"
```

```
9156 PRINT WS$;:ES$="":INPUT ES$
9159 W%=ES$="W"
9162 REM EINGABE DER ROCHADEMOEGLICHKEITEN
9163 PRINT "WELCHE ROCHADEMOEGLICHKEITEN GIBT ES?"
9164 PRINT "ANTWORTEN SIE AUF EVENTUELLE FRAGEN MIT"
9165 PRINT "M=MOEGLICH, U=UNMOEGLICH"
9167 IF (BR%(31)<>13) OR (BR%(27)<>11) THEN GOTO 9173
9168 INPUT"LANGE ROCHADE VON WEISS";ES$
9169 RO%(1,1)=ES$="M"
9173 IF (BR%(31)<>13) OR (BR%(34)<>11) THEN GOTO 9179
9174 INPUT"KURZE ROCHADE VON WEISS";ES$
9175 RO%(1,2)=ES$="M"
9179 IF (BR%(115)<>1) OR (BR%(111)<>3) THEN GOTO 9185
9180 INPUT "LANGE ROCHADE VON SCHWARZ";ES$
9181 RO%(2,1)=ES$="M"
9185 IF (BR%(115)<>1) OR (BR%(118)<>3) THEN GOTO 9191
9186 INPUT "KURZE ROCHADE VON SCHWARZ";ES$
9187 RO%(2,2)=ES$="M"
9191 EP%=0
9200 REM PARTIEPARAMETER
9202 PRINT "MIT WELCHER FARBE MOECHTEN SIE SPIELEN?"
9204 PRINT WS$;:ES$="":INPUT ES$
9206 GW%=ES$="W"
9214 INPUT "GEBEN SIE BITTE DIE SPIELSTUFE EIN      (1-12)";ZE%
9216 ZE%=3*(2^ZE%)
9298 RETURN:REM ENDE DER STELLUNGSEINGABE

9900 REM UMWANDLUNG BRETTNOTATION -> INTERNE DARSTELLUNG
9901 IF LEN(ES$)>0 THEN GOTO 9904
9902 IF (F%<>1) AND (F%<>13) THEN S%=0:RETURN
9903 S%=1:PRINT "FEHLENDE FELDANGABE FUER DEN KOENIG!":RETURN
9904 IF LEN(ES$)<>2 THEN GOTO 9912
9905 A1%=3+ASC(LEFT$(ES$,1))-ASC("A")
9906 A2%=2+ASC(MID$(ES$,2,1))-ASC("1")
9908 IF (A1%<3)OR(A1%>10)OR(A2%<2)OR(A2%>9) THEN GOTO 9912
9910 S%=12*A2%+A1%
9911 RETURN
9912 PRINT "FALSCHE FELDANGABE!":S%=1
9913 RETURN
9920 REM FUELLEN VON BRETT UND MATERIALLISTEN
```

```
9921 IF FZ%(I)=16 THEN GOTO 9930
9922 IF BR%(S%)<>FF% THEN GOTO 9931
9923 FZ%(I)=FZ%(I)+1
9924 FI%(I,FZ%(I))=F%
9925 FS%(I,FZ%(I))=S%
9926 BR%(S%)=F%
9927 FA%(S%)=FZ%(I)
9928 IF ABS(FF%-F%)<6 THEN MS%(I)=MS%(I)+FW%(F%)
9929 A1%=0:RETURN
9930 PRINT "MAXIMALE FIGURENANZAHL ERREICHT!":GOTO 9932
9931 PRINT "FELD SCHON BESETZT!"
9932 A1%=-1:RETURN
```

Kommentare:

In diesem Modul werden die im Abschnitt 4.2.1 erläuterten Variablen für die Stellungsdarstellung und für die Materialbewertung entsprechend der gewählten Anfangsposition und außerdem die Partieparameter "Farbverteilung" und "Spielstufe" gesetzt.

9013-9026: Nullsetzen der Datenstrukturen (z.B., um Inhalte einer vorangegangenen Partie zu löschen)

9028-9032: Verzweigung zur gewünschten Initialisierungsform (Grundstellung oder Stellungseingabe)

9034-9050: Belegung der Variablen für die Stellungsdarstellung und der Materialbewertung mit den Werten für die Grundstellung (Partiebeginn)

9058: Überlesen der DATA-Zeilen, die für die Initialisierung mit der Grundstellung vorgesehen sind

9062-9069: Bedienungsanleitung für die Stellungseingabe

9072-9100: Eingabe der Figurenstandorte:

- 9082: Anzeige eines Figurentyps und Eingabe des Standortes bzw. leere Eingabe
- 9084: Analyse der Eingabe und Umwandlung der Brettnotation in die interne Darstellung bzw. Fehlermeldung
- 9086: Weiterschalten zum nächsten Figurentyp nach leerer Eingabe (s%=0)
- 9088: Neueingabe zum gleichen Figurentyp im Fehlerfall (s%=1)
- 9090: Eintrag in die Variablen zur Stellungsdarstellung und Berechnung der Materialsummen
- 9092: Neueingabe zum gleichen Figurentyp nach erfolgreichem Eintrag (nicht für Könige: k>2) oder nach Fehlermeldung (a1%=-1)

9102: Berechnung der Materialbalance aus den Materialsummen

9155-9159: Eingabe des Anzugrechts (w%)

9163-9187: Eingabe der Rochaderechte (ro%)

9191: Initialisierung des e.p.-Feldes

9202-9206: Eingabe der Farbverteilung (gw%)

9214-9216: Eingabe der Spielstufe (ze%)

9900-9913: Hilfsroutine zur Bearbeitung einer Feldeingabe

9920-9932: Hilfsroutine zur Belegung der Variablen zur Stellungsdarstellung und Berechnung der Materialsummen

4.5 Zuggenerator (999-1999)

Das Zuggenerator-Modul besteht aus folgenden Routinen:

Dame-Läufer-Turm-Zuggenerator (1000-1099)
Springerzuggenerator (1100-1199)
Bauernzuggenerator Weiß (1200-1299)
Bauernzuggenerator Schwarz (1300-1399)
Königszuggenerator (1400-1499)
En-passant-Schlagzuggenerator (1600-1699)
Rochaden-Generator (1700-1800)
Zuggenerator-Steuerung (1900-1999)

Von außen wird nur die Steuerung angsprungen. Der Zuggenerator erwartet, daß die Stellung in den hierfür vorgesehenen Variablen korrekt dargestellt ist und schreibt die Züge in die Zugliste. Dabei setzt er auch die Zuglistenverwaltung (zv%).

a) Die Steuerung (1900-1999)

```
1900 REM ZUEGE GENERIEREN
1902 REM@M
1905 A5%=W%+2
1906 ZV%(1)=I1%:ZV%(2)=I2%:ZV%(3)=I2%:ZV%(4)=1
1907 IF T%<9 THEN ST%(T%,44)=0
1908 GOSUB 1610:REM EN-PASSANT-SCHLAGEN
1909 GOSUB 1700:REM ROCHADEN
1910 FOR J3=1 TO FZ%(A5%)
1913 ZF%=FI%(A5%,J3)
1915 IF ZF%=FF% THEN GOTO1940
1916 S%=FS%(A5%,J3)
1925 ONZF%GOSUB1400,1000,1000,1000,1100,1300,1950,1200,1100,1000,
1000,1000,1400
```

```
1940 NEXT J3
1949 REM@P
1950 RETURN
```

a5% dient zur Indizierung der Figurenliste (Wert 1 bei Weiß, 2 bei Schwarz am Zug).
zv% wird vor dem Generieren gesetzt (vgl. 4.2.3 und 4.9).
Zunächst werden die Sonderfälle bearbeitet (1908-1909)
In der Schleife (1910-1940) wird für alle Figuren des Spielers, der am Zug ist, Figurenart (zf%) und Standort (s%) der ziehenden Figur gesetzt und die jeweilige Generator-Routine mit diesen Informationen angestartet.

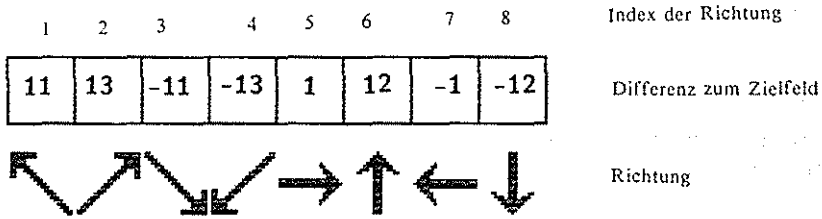
Alle einzelnen Generator-Routinen können also voraussetzen, daß s% und zf% schon den richtigen Wert haben. Alle benutzen die Variable z%, um das Zielfeld darzustellen.

b) Dame-Turm-Läufer-Zuggenerator (1000-1099)

```
999 REM@M
1000 REM DAMEN,TURM,LAEUFERZUEGE
1001 HW%=0:A4%=1
1002 IF (ZF%=WT%) OR (ZF%=ST%) THEN A1%=5:A4%=3:GOTO1004
1003 A1%=1
1004 IF (ZF%=WL%) OR (ZF%=SL%) THEN A2%=4:A4%=4:GOTO1009
1005 A2%=8
1009 FOR J=A1% TO A2%
1011 A3%=DW%(J)
1013 Z%=S%
1015 Z%=Z%+A3%
1020 F%=BR%(Z%)
1025 IF F%=KF% THEN GOTO 1085
1027 HW%=HW%+KO%(Z%)
1030 IF F%<>FF% THEN GOTO 1050
1035 REM RUHIGER ZUG
1040 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1045 GOTO 1015
1050 IF (F%>FF%)=W% THEN GOTO 1085
1055 REM SCHLAGZUG
1060 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
```

```
1085 NEXT J
1086 REM FELDERKONTROLLEN
1087 IF T%<9 THEN ST%(T%,44)=ST%(T%,44)+HW%*A4%
1090 RETURN
```

Den Bewegungsrichtungen der Dame (links-oben, rechts-oben, usw.) sind Zahlen zugeordnet (1-8). Das konstante Array dw% (für Damenweite) enthält die Felderdifferenzen für Nachbarfelder der jeweiligen Richtung (vgl. 3.3):



Belegung und Bedeutung von dim dw%(8)

Zunächst werden a1% und a2% für die Bewegungsrichtungen der ziehenden Figur gesetzt (1001-1005). Die anschließende Schleife (1009-1085) klappert diese Richtungen ab. Jeweils wird die Distanz zum nächsten Feld aus dw% geholt (1011).
Für diese Richtung werden Züge erzeugt. Zunächst werden in der inneren Schleife (1015-1045) die ruhigen Züge berechnet
Die Schleife wird verlassen, wenn das Zielfeld außerhalb des Brettes ist (1025) oder dort eine Figur steht (1030).
Im zweiten Fall wird ein Schlagzug erzeugt (1060), falls es sich um eine gegnerische Figur handelt.

Die Variablen a4% und hw% sowie Zeile 1087 gehören thematisch zur Bewertungsfunktion und werden dort beschrieben. Sie betreffen nicht das Generieren der Züge, sondern dienen der Berechnung der Felderkontrollen.

c) Springerzuggenerator (1100-1199)

```
1100 REM SPRINGERZUEGE
1101 FOR J=1 TO 8
1105 Z%=S%+SW%(J)
1110 F%=BR%(Z%)
1115 IF F%=KF% THEN GOTO1170
1120 IF F%=FF% THEN GOTO1155
1125 IF (F%>FF%)=W% THEN GOTO1170
1130 REM SCHLAGZUG
1135 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1145 GOTO 1170
1150 REM RUHIGER ZUG
1155 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1170 NEXT J
1175 RETURN
```

Die Felderdistanzen für die 8 Sprungmöglichkeiten des Springers werden aus dem konsten Array sw% (für Springerweiten) geholt, das analog zu dw% (Damenzuggenerator) aufgebaut ist.

1	2	3	4	5	6	7	8	Index der Richtung
10	23	25	14	-10	-23	-25	-14	Differenz zum Zielfeld

Richtung

Belegung und Bedeutung von dim sw%(8)

In der Schleife (1101-1170) wird das Zielfeld also mit Hilfe von sw% berechnet (1105). Analog zu den Damenzügen wird auch hier das Zielfeld überprüft:

Außerhalb des Bretts:	kein Zug (1115)
freies Feld:	ruhiger Zug (1120,1155)
eigene Figur:	kein Zug (1125)
gegnerische Figur:	Schlagzug (1135)

Dies geschieht analog auch in den anderen Generator-Routinen

d) Bauernzuggenerator für Weiß (1200-1299)

```
1200 REM BAUERNZUEGE
1201 REM WEISSER BAUER
1205 Z%=S%+12:REM EIN FELD VOR
1206 IF BR%(Z%)<>FF% THEN GOTO 1225
1207 IF Z%>106 THEN GOSUB 1260:GOTO1225
1208 REM RUHIGER ZUG
1209 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1215 Z%=S%+24:REM ZWEI FELDER VOR
1216 IF BR%(Z%)<>FF% THEN GOTO 1225
1217 IF S%>46 THEN GOTO 1225
1218 REM RUHIGER ZUG
1219 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1225 Z%=S%+11:REM SCHLAGEN LINKS OBEN
1226 F%=BR%(Z%)
1227 IF F%=KF% THEN GOTO 1235
1228 IF F%>SB% THEN GOTO 1235
1229 IF Z%>106 THEN GOSUB1260:GOTO1235
1230 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1235 Z%=S%+13:REM SCHLAGEN RECHTS OBEN
1236 F%=BR%(Z%)
1237 IF F%=KF% THEN GOTO 1245
1238 IF F%>SB% THEN GOTO 1245
1239 IF Z%>106 THEN GOSUB1260:GOTO1245
1240 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1245 RETURN
1260 REM BAUERNUMWANDLUNG WEISS
1265 F%=WD%
```

```
1266 FOR J1=I2%+1 TO I2%+4
1267 Z2%(J1,1)=S%:Z2%(J1,2)=Z%:Z2%(J1,3)=F%
1270 F%=F%-1
1271 NEXT J1
1272 I2%=I2%+4
1273 RETURN
```

Bauernzüge sind etwas umständlich. Bauern dürfen nur vorwärts ziehen, wenn dort keine Figur steht, und nur schräg schlagen, wenn dort eine gegnerische Figur ist. In der Ausgangsposition dürfen sie auch zwei Felder vorrücken, und erreichen sie die gegnerische Grundlinie, können sie sich in eine beliebige Figur umwandeln.

Dies macht das Programmieren zu einer Strafarbeit.

In Zeile 1217 wird abgefragt, ob der Bauer noch in der Ausgangsposition ist - nur dann darf er zwei Felder vor.
In den Zeilen 1217, 1229 und 1233 wird getestet, ob der Bauer die gegnerische Grundlinie erreicht und sich dort umwandelt. Ist dies der Fall, wird die Hilfsroutine für Bauernumwandlungen (1260-1273) angesprungen. Diese trägt dann die 4 möglichen Umwandlungen ein.

e) Bauernzuggenerator für Schwarz (1300-1399)

```
1300 REM SCHWARZER BAUER (ANALOG ZU 1201-1273)
1305 Z%=S%-12
1306 IF BR%(Z%)<>FF% THEN GOTO 1325
1307 IF Z%<39 THEN GOSUB 1360:GOTO1325
1309 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1315 Z%=S%-24
1316 IF BR%(Z%)<>FF% THEN GOTO 1325
1317 IF S%<99 THEN GOTO 1325
1319 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1325 Z%=S%-13
1326 F%=BR%(Z%)
1327 IF F%=KF% THEN GOTO 1335
1328 IF F%<WB% THEN GOTO 1335
1329 IF Z%<39 THEN GOSUB1360:GOTO1335
```

```
1330 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1335 Z%=S%-11
1336 F%=BR%(Z%)
1337 IF F%=KF% THEN GOTO 1345
1338 IF F%<WB% THEN GOTO 1345
1339 IF Z%<39 THEN GOSUB1360:GOTO1345
1340 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1345 RETURN
1360 REM BAUERNUMWANDLUNG SCHWARZ
1365 F%=SD%
1366 FOR J1=I2%+1 TO I2%+4
1367 Z2%(J1,1)=S%:Z2%(J1,2)=Z%:Z2%(J1,3)=F%
1370 F%=F%+1
1371 NEXT J1
1372 I2%=I2%+4
1373 RETURN
```

Diese Routine funktioniert analog zu der für Weiß, nur daß die Zeilennummern um 100 erhöht sind.

e) Königszuggenerator (1400-1499)

```
1400 REM KOENIGSZUEGE
1405 FOR J=1 TO 8
1415 Z%=S%+DW%(J)
1420 F%=BR%(Z%)
1425 IF F%=KF% THEN GOTO 1485
1430 IF F%=FF% THEN GOTO 1465
1435 IF (F%>FF%)=W% THEN GOTO 1485
1440 REM SCHLAGZUG
1445 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1456 GOTO1485
1460 REM RUHIGER ZUG
1465 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1485 NEXT J
1490 RETURN
```

Dieser Generator arbeitet analog zu dem für Springerzüge, nur daß die Distanzen zu den Nachbarfeldern hier wieder aus dw% geholt werden.

f) En-Passant-Schlagen (1600-1699)

```
1600 REM SONDERFAELLE
1610 REM EN-PASSANT-SCHLAGEN
1615 IF EP%=0 THEN GOTO 1645
1620 IF W% THEN A1%=12:A2%=WB%:A3%=SB%:GOTO1625
1621 A1%=-12:A2%=SB%:A3%=WB%
1625 S%=EP%+1
1626 IF BR%(S%)<>A2% THEN GOTO 1635
1627 Z%=EP%+A1%
1628 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%:Z2%(I2%,3)=A3%
1635 S%=EP%-1
1636 IF BR%(S%)<>A2% THEN GOTO 1645
1637 Z%=EP%+A1%
1638 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%:Z2%(I2%,3)=A3%
1645 RETURN
```

Ein En-Passant-Schlagzug ist nur möglich, wenn zuvor ein Bauer 2 Felder vorgegangen ist (1615).

Je nach Farbe werden die Hilfsvariablen a1%, a2% und a3% gesetzt: a1% gibt die Differenz eines eventuellen Zielfelds zu ep% an, und die beiden anderen Variablen sind Benennungen für "eigener Bauer" und "gegnerischer Bauer".
Geprüft wird, ob rechts (1625-1628) oder links (1635-1638) neben ep% ein eigener Bauer steht. Wenn dies der Fall ist, wird der Schlagzug eingetragen.

g) Rochadezuggenerator (1700-1800)

```
1700 REM ROCHADEN
1705 A1%=W%+2
1710 IF NOT RO%(A1%,2) THEN GOTO 1760
1715 REM KURZE ROCHADE
1720 IF W% THEN A2%=32:GOTO1730
1725 A2%=116
1730 A3%=(BR%(A2%)=FF%) AND (BR%(A2%+1)=FF%)
1735 IF NOT A3% THEN GOTO 1760
1740 I1%=I1%+1:Z1%(I1%,1)=A2%-1:Z1%(I1%,2)=A2%+1
```

```
1755 REM LANGE ROCHADE
1760 IF NOT RO%(A1%,1) THEN GOTO 1800
1765 IF W% THEN A2%=28:GOTO1775
1770 A2%=112
1775 A3%=(BR%(A2%)=FF%) AND (BR%(A2%+1)=FF%) AND (BR%(A2%+2)=FF%)
1780 IF NOT A3% THEN GOTO 1800
1785 I1%=I1%+1:Z1%(I1%,1)=A2%+3:Z1%(I1%,2)=A2%+1
1800 RETURN
```

Rochaden sind nur möglich, wenn der König und der beteiligte Turm noch (bisher unbewegt) auf ihren Ursprungsfeldern stehen. Dies kann des Programm durch ro% abfragen (1710, 1760).

Zusätzlich muß geprüft werden, ob die Zwischenfelder frei sind (1730, 1775).

a1% ist der Index des ziehenden Spielers, a2% der Standort des Königs.

Ein König darf nur rochieren, wenn er nicht im Schach steht und er kein vom Gegner angegriffenes Feld überquert. Dies zu prüfen ist sehr zeitaufwendig und geschieht deshalb noch nicht hier. Nur wenn die Rochade wirklich ausgeführt werden soll, muß das Programm diese Bedingung testen.

4.6 Die Bewertungsfunktion

In Kapitel 3 haben wir die (schach-) theoretischen Grundlagen der Stellungsbewertung kennengelernt. Nun geht es um deren Realisierung in unserem Schachprogramm. Dabei werden wir feststellen, daß die im Abschnitt 3.5 benutzte Gliederung nicht nur den Zweck hatte, den dargebotenen Stoff übersichtlicher zu gestalten, sondern daß wir damit auch schon in etwa die Aufteilung des Programmcodes in Module vorgenommen haben. Wir haben es daher mit den folgenden Bestandteilen der Bewertungsfunktion zu tun:

1. Unterprogramm zur Steuerung des Ablaufes (Aufruf der Stellungsanalyse, Aufrufe der Einzelbewertungen, u.a.)

2. Unterprogramm zur Analyse des Stellungstyps

3. Unterprogramm zur Bewertung der Bauernstruktur

4. Unterprogramm zur Bewertung der Figurenstandorte

5. Unterprogramm zur Bewertung der Königssicherheit / Königsaktivität

6. Unterprogramme für die Matt-Routinen

Wie diese Programmteile miteinander verknüpft sind, welche Variablen sie benutzen und was sie im einzelnen berechnen, wird in den nächsten Abschnitten beschrieben. Dabei wird jedem Modul ein eigener Abschnitt gewidmet, der jeweils so aufgebaut ist, daß zunächst das Programmlisting (incl. Kommentarzeilen) des entsprechenden Moduls und eine Liste der benutzten Variablen (mit deren Bedeutungen) aufgeführt werden, denen sich Anmerkungen zum Programmtext anschließen.

4.6.1 Die Steuerung des Ablaufes

```
2000 REM BEWERTUNGSFUNKTION
2001 REM@M
2002 GOSUB 3014:REM ANALYSE DES STELLUNGSTYPS
2003 ON PT% GOTO 2030,2034,2038
2004 REM NORMALE BEWERTUNGSFUNKTION
2005 REM BEWERTUNGSABBRUCH BEI ZU ERWARTENDEM ALPHA- BZW. BETA-CUTOFF
2006 IF MB%+120<=AL% THEN PW%=MB%+120:GOTO 2088
2007 IF MB%-120>=BE% THEN PW%=MB%-120:GOTO 2088
2008 REM ABSCHAETZUNG FUER RUHESUCHE
2009 IF T%>MT% THEN PW%=MB%+RS%:GOTO 2088
2010 PW%=MB%:REM MATERIALBEWERTUNG
2011 IF NOT(ES%(1) AND ES%(2)) THEN GOTO 2014
```

```
2012 IF MB%>0 THEN PW%=PW%+BZ%(1,1)*10:GOTO 2014
2013 IF MB%<0 THEN PW%=PW%-BZ%(2,1)*10
2014 HW%=0
2015 GOSUB 2104:REM BAUERNBEWERTUNG
2016 GOSUB 2302:REM FIGURENBEWERTUNG
2018 GOSUB 2504:REM KOENIGSBEWERTUNG
2021 REM BEWERTUNG DER KONTROLLEN
2022 IF T%=1 OR T%>8 THEN A1%=0:GOTO 2026
2023 A1%=ST%(T%,44)-ST%(T%-1,44)
2024 IF NOT W% THEN A1%=-A1%
2025 REM GESAMTBEWERTUNG
2026 PW%=PW%+INT((HW%+A1%)/100)
2027 RS%=PW%-MB%
2028 GOTO 2088
2029 REM MATTFUEHRUNG GEGEN WEISS
2030 GOSUB 3202:GOTO 2088
2032 REM REMISSTELLUNG
2034 PW%=0:GOTO 2088
2036 REM MATTFUEHRUNG GEGEN SCHWARZ
2038 GOSUB 3102
2088 REM@P
2089 RETURN
2099 REM@M
```

Variablenliste:

pt%:	Positionstyp
mb%:	Materialbalance
al%:	Alpha
be%:	Beta
pw%:	Wert der Stellung
t%:	Tiefe der aktuellen Stellung
mt%:	Suchtiefe
rs%:	positionelle Abschätzung während der Ruhesuche
es%(2):	Ergebnis der Endspielabfrage
bz%(2,11):	Bauernzähler
st%(8,44):	Der Stack
w%:	Anzugsrecht

diverse Hilfsvariablen

Kommentare:

Dieses Modul schafft zum einen die Verbindung zur Suche und organisiert zum anderen den internen Ablauf der Stellungsbewertung. Dies ist daran ersichtlich, daß dieser Programmteil von dem Suchalgorithmus mit "GOSUB 2000" angesprungen wird und daß nur von hier die anderen Unterprogramme der Bewertungsfunktion aufgerufen werden. Deren Zwischenergebnisse werden in diesem Modul weiterverarbeitet und als Endresultat der Suche übergeben.

2002: Aufruf der Stellungsanalyse, die nach den Kriterien aus 3.5.3 den Positionstyp bestimmt und einige weitere Arbeiten erledigt (siehe Abschnitt 4.6.2). Der Positionstyp wird codiert in pt% von der Analyse geliefert.

2003: Verzweigung zur dem Positionstyp entsprechenden Bewertung

2006-2007: Abschätzung des Stellungswertes (anstelle einer exakten positionellen Bewertung), falls die zu erwartende Gesamtbewertung so gut wie sicher nicht ins Alpha-Beta-Intervall fallen wird und es daher wahrscheinlich in der Suche zum Cutoff kommen wird. Dies ist eine Konsequenz dessen, daß die positionelle Bewertung im allgemeinen nur betragsmäßig kleine Werte (> Wert eines Bauern) liefert.

2009: Für Stellungen in der Ruhesuche interessiert im allgemeinen nur, ob sich Veränderungen bezüglich der Materialbalance ergeben können. Die positionelle Bewertung kann daher für diese Stellungen "abgeschaltet" werden. Dies geschieht dadurch, daß in den Stellungen bis zur maximalen Tiefe mt% die ermittelten positionellen Werte in rs% abgespeichert werden, wobei ein Ergebnis aus der Suchtiefe "t%" einen vorangegangenen Wert (aus Tiefe "t%-1") überschreibt und der Wert aus der Tiefe mt%

während der Ruhesuche zur Abschätzung benutzt wird.
Diese Maßnahme und die positionellen Abschätzungen bei zu erwartendem Alpha- bzw. Beta-Cutoff wurden im Kapitel 3 nicht beschrieben, tragen aber wesentlich zu einer Beschleunigung des Schachprogrammes bei, da in diesen Fällen die zeitaufwendige positionelle Bewertung eingespart werden kann, ohne daß ein großer Informationsverlust in Kauf genommen werden muß.

2010: Für die Materialbewertung wird kein eigenes Modul benötigt, da die beiden Materialsummen und damit die Materialbalance inkrementell außerhalb der Bewertungsfunktion berechnet werden. Dies geschieht für die Initialisierung der Materialwerte entweder in der Zeile 9044 (Stellungsinitialisierung mit der Grundstellung) oder in den Zeilen 9076, 9102 und 9928 (Stellungsbewertung mit einer eingelesenen Stellung). Die inkrementelle Behandlung dieser Variablen mit Hilfe des Stacks findet in folgenden Zeilen statt: 4144-4148 (Retten im Stack vor dem Vertiefen eines Zuges), 4311, 4323, 4336 und 4338 (Aktualisierung beim Vertiefen, falls erforderlich) sowie 4644-4648 (Zurückholen der alten Werte aus dem Stack nach Zugrücknahme). Diese angesprochenen Aktionen werden im Zusammenhang mit den Moduln, in denen sie stehen, aufgelistet. Die Berechnung der Materialsummen stützt sich auf die Benutzung der Figurenwerte aus dem Feld fw%(12) (siehe Zeilen 108-110, 202). Zur Beachtung: Für die schwarzen Steine werden negative Werte verwendet. Die Matterialbalance ist daher die Summe der Materialsummen.

2011-2013: Modifikation der Materialbewertung in Endspielpositionen mit ungleichem Materialverhältnis zur Behandlung des Bauerntausches (Die Bestimmung der Werte für das zweidimensionale Feld bz%

sowie deren Verwendung wird in den Abschnitten 4.6.2 und 4.6.3 erläutert.)

2014-2018: Aufrufe der Unterprogramme für die Bauern-, die Figuren- und die Königsbewertung. Deren Ergebnisse werden in der Hilfsvariablen hw% aufsummiert (innerhalb der Unterprogramme)

2022-2024: Die Werte für die Felderkontrollen werden, wie in 3.5.3 beschrieben, aus dem Stack bezogen, in dem sie vom Zuggenerator abgelegt wurden; und zwar interessiert die Differenz der Werte für die aktuelle Stellung (Tiefe=t%) und der unmittelbaren Vorgängerstellung (Tiefe=t%-1). Je nach der Farbe des am Zug befindlichen Spielers wird dieses Resultat mit unterschiedlichem Vorzeichen in der Gesamtbewertung berücksichtigt (Zeile 2024). Ferner ist zu beachten, daß die Kontrollen nur bewertet werden können, wenn mindestens zwei Werte im Stack vorliegen (also nicht in der Ausgangsstellung) und die aktuelle Suchtiefe nicht die Grenze der 1. Dimension des Stacks übersteigt. Dies ist für 2<=t%<=8 der Fall.

2026: In den Unterprogrammen der Bewertung werden Wichtungskoeffizienten benutzt, die den 100fachen Wert gegenüber den in Kapitel 3 angegebenen Werten aufweisen. Dies geschah, um in der Bewertungsfunktion ausschließlich mit Integer-Variablen rechnen zu können. Dies wird hier wieder rückgängig gemacht.

2027: siehe Anmerkungen zur Zeile 2009

2029-2038: Behandlung der Sonderfälle (Mattführung oder Remisstellung)

4.6.2 Die Stellungsanalyse

```
3000 REM STELLUNGSANALYSE
3012 REM ENDSPIELABFRAGEN
3014 ES%(1)=(MS%(2)>-2000)
3016 ES%(2)=(MS%(1)< 2000)
3018 REM BESTIMMUNG DER BAUERNSTANDORTE
3020 FOR I=1 TO 2
3022 FOR J=1 TO 11
3024 BZ%(I,J)=0
3026 NEXT J
3028 NEXT I
3030 FOR I=36 TO 96
3032 FOR J=3 TO 10
3034 IF BR%(I+J)<>WB% THEN GOTO 3040
3036 BZ%(1,1)=BZ%(1,1)+1:BZ%(1,J)=BZ%(1,J)+1:BF%(1,J,BZ%(1,J))=I+J
3038 GOTO 3044
3040 IF BR%(I+J)<>SB% THEN GOTO 3044
3042 BZ%(2,1)=BZ%(2,1)+1:BZ%(2,J)=BZ%(2,J)+1:BF%(2,J,BZ%(2,J))=I+J
3044 NEXT J
3046 I=I+11:NEXT I
3048 REM BESTIMMUNG DES STELLUNGSTYPS
3050 IF NOT(ES%(1) OR ES%(2)) THEN GOTO 3068
3052 IF (BZ%(1,1)+BZ%(2,1))>0 THEN GOTO 3068
3054 IF MB%<500 THEN GOTO 3060
3056 IF MS%(2)<-650 THEN GOTO 3068
3058 PT%=3:GOTO 3082:REM MATTFUEHRUNG GEGEN SCHWARZ
3060 IF MB%>-500 THEN GOTO 3066
3062 IF MS%(1)>650 THEN GOTO 3068
3064 PT%=1:GOTO 3082:REM MATTFUEHRUNG GEGEN WEISS
3066 IF (MS%(1)<=325)AND(MS%(2)>=-325) THEN PT%=2:GOTO 3082:REM REMIS
3068 PT%=4:REM NORMALE BEWERTUNG
3082 RETURN
```

Variablenliste:

es%(2):	Ergebnis der Endspielabfrage
ms%(2):	Materialsummen der Spieler
bz%(2,11):	Bauernzähler
br%(144):	Das Brett

bf%(2,11,6): Bauernstandorte
mb%: Materialbalance
pt%: Positionstyp

Anmerkungen:

Die Hauptaufgabe dieses Moduls ist das Setzen der Variablen pt% nach folgendem Schema:

pt%=1: Mattführung gegen den weißen König
pt%=2: Remisstellung
pt%=3: Mattführung gegen den schwarzen König
pt%=4: restliche Stellungen

Des weiteren werden erledigt: Endspielabfrage und Bestimmung der Bauernanzahl und der Bauernstandorte

3014-3016: nach Farben getrennte Endspielabfrage (Vergleich von gegnerischer Materialsumme und Schrankenwert)

3020-3046: Die Bauernstandorte sollen linienweise abgespeichert werden, um in der Bauernbewertung die Überprüfung der Stellungsmerkmale effizient vornehmen zu können. Daraus ergeben sich die Dimensionen und ihre Obergrenzen für die Datenstrukturen bf% und bz%. Der 1. Index bezeichnet den Spieler (1=Weiß, 2=Schwarz). Dies gilt im übrigen auch für alle weiteren Variablen in der Bewertungsfunktion, die für beide Farben geführt werden müssen. Der 2. Index steht für die Linie (3 bis 10 = A- bis H-Linie; zusätzlich 2 und 11 als imaginäre Nachbarlinien für die Randlinien). Die Begrenzung 6 für die 3. Dimension entspricht der maximal möglichen Bauernanzahl auf einer Linie. In bz% wird eingetragen, wieviele Bauern auf den einzelnen Linien pro Spieler vorhanden sind. Außerdem wird in bz%(1,1) bzw. bz%(2,1) die Gesamtzahl aller weißen bzw. schwarzen Bauern abgespeichert. Die Bestimmung der Bauernstand-

orte und Bauernanzahlen geschieht, indem das gesamte Brett abgesucht wird (in einem Durchgang für Weiß und Schwarz). Für die aufgefundenen Bauern werden die entsprechenden Eintragungen in bf% und bz% vorgenommen.

3050: Eine Remisstellung oder Mattführung ist nur im Endspielstadium möglich.

3052-3064: Überprüfung der Voraussetzungen für den Aufruf einer Matt-Routine

3066: Test auf Remisstellung anhand der Materialsummen: Die Abfrage ist erfüllt, falls beide Spieler noch maximal eine Leichtfigur besitzen.

4.6.3 Die Bauernbewertung

```
2100 REM BEWERTUNG FUER DIE WEISSEN BAUERN
2102 REM INITIALISIERUNGEN
2104 FOR I=3 TO 10:FB%(1,I)=0:RB%(1,I)=0:NEXT I
2106 IF ES%(1) THEN GOTO 2112
2108 FOR I=3 TO 10:BV%(1,I)=KV%(I):NEXT I
2110 GOTO 2118
2112 FOR I=3 TO 10:BV%(1,I)=200:NEXT I
2116 REM BLOCKIERUNG DER ZENTRUMSBAUERN
2118 IF (BR%(42)=WB%) AND (BR%(54)<>FF%) THEN HW%=-1000
2120 IF (BR%(43)=WB%) AND (BR%(55)<>FF%) THEN HW%=HW%-1000
2122 FOR I=3 TO 10
2124 IF BZ%(1,I)=0 THEN GOTO 2190
2126 REM ISOLIERTE BAUERN
2128 IF (BZ%(1,I-1)=0)AND(BZ%(1,I+1)=0) THEN HW%=HW%-1500
2130 REM MEHRFACHBAUERN
2132 IF BZ%(1,I)<2 THEN GOTO 2137
2133 IF BZ%(1,I)=2 THEN HW%=HW%-400:GOTO 2137
2134 HW%=HW%-4000
2136 REM FREIBAUERN
2137 S%=BF%(1,I,BZ%(1,I))
2142 IF (BZ%(2,I-1)<>0)AND(BF%(2,I-1,BZ%(2,I-1))>S%  ) THEN GOTO 2160
```

```
2144 IF (BZ%(2,I  )<>0)AND(BF%(2,I  ,BZ%(2,I  ))>S%  ) THEN GOTO 2160
2146 IF (BZ%(2,I+1)<>0)AND(BF%(2,I+1,BZ%(2,I+1))>S%+1) THEN GOTO 2160
2148 FB%(1,I)=-1
2150 A1%=115
2152 IF BR%(S%+12)<FF% THEN A1%=80
2154 IF (BR%(S%-13)=WB%)OR(BR%(S%-11)=WB%) THEN A1%=A1%+35
2156 IF ES%(1) OR ES%(2) THEN A1%=A1%*2
2157 A2%=INT(S%/12)-1
2158 HW%=HW%+A1%*A2%*A2%
2160 FOR J=1 TO BZ%(1,I)
2162 S%=BF%(1,I,J)
2164 REM VORRUECKEN DER BAUERN
2166 HW%=HW%+(INT(S%/12)-3)*BV%(1,I)
2168 REM RUECKSTAENDIGE BAUERN
2178 IF (BR%(S%-1)=WB%)OR(BR%(S%-13)=WB%) THEN GOTO 2186
2180 IF (BR%(S%+1)=WB%)OR(BR%(S%-11)=WB%) THEN GOTO 2186
2182 IF BR%(S%+23)=SB% THEN RB%(1,I)=RB%(1,I)-200
2184 IF BR%(S%+25)=SB% THEN RB%(1,I)=RB%(1,I)-200
2186 NEXT J
2188 HW%=HW%+RB%(1,I)
2190 NEXT I
2200 REM BEWERTUNG FUER DIE SCHWARZEN BAUERN (ANALOG ZU 2100-2190)
2204 FOR I=3 TO 10:FB%(2,I)=0:RB%(2,I)=0:NEXT I
2206 IF ES%(2) THEN GOTO 2212
2208 FOR I=3 TO 10:BV%(2,I)=KV%(I):NEXT I
2210 GOTO 2218
2212 FOR I=3 TO 10:BV%(2,I)=200:NEXT I
2218 IF (BR%(102)=SB%) AND (BR%(90)<>FF%) THEN HW%=HW%+1000
2220 IF (BR%(103)=SB%) AND (BR%(91)<>FF%) THEN HW%=HW%+1000
2222 FOR I=3 TO 10
2224 IF BZ%(2,I)=0 THEN GOTO 2290
2228 IF (BZ%(2,I-1)=0)AND(BZ%(2,I+1)=0) THEN HW%=HW%+1500
2232 IF BZ%(2,I)<2 THEN GOTO 2237
2233 IF BZ%(2,I)=2 THEN HW%=HW%+400:GOTO 2237
2234 HW%=HW%+4000
2237 S%=BF%(2,I,1)
2242 IF (BZ%(1,I-1)<>0)AND(BF%(1,I-1,1)<S%-1) THEN GOTO 2260
2244 IF (BZ%(1,I  )<>0)AND(BF%(1,I  ,1)<S%  ) THEN GOTO 2260
2246 IF (BZ%(1,I+1)<>0)AND(BF%(1,I+1,1)<S%  ) THEN GOTO 2260
2248 FB%(2,I)=-1
```

```
2250 A1%=-115
2252 IF BR%(S%-12)>FF% THEN A1%=-80
2254 IF (BR%(S%+11)=SB%)OR(BR%(S%+13)=SB%) THEN A1%=A1%-35
2256 IF ES%(1) OR ES%(2) THEN A1%=A1%*2
2257 A2%=10-INT(S%/12)
2258 HW%=HW%+A1%*A2%*A2%
2260 FOR J=1 TO BZ%(2,I)
2262 S%=BF%(2,I,J)
2266 HW%=HW%-(8-INT(S%/12))*BV%(2,I)
2278 IF (BR%(S%-1)=SB%)OR(BR%(S%+11)=SB%) THEN GOTO 2286
2280 IF (BR%(S%+1)=SB%)OR(BR%(S%+13)=SB%) THEN GOTO 2286
2282 IF BR%(S%-25)=WB% THEN RB%(2,I)=RB%(2,I)+200
2284 IF BR%(S%-23)=WB% THEN RB%(2,I)=RB%(2,I)+200
2286 NEXT J
2288 HW%=HW%+RB%(2,I)
2290 NEXT I
2294 RETURN
```

Variablenliste:

fb%(2,10): Information über Freibauern
rb%(2,10): Information über rückständige Bauern
es%(2): Ergebnis der Endspielabfrage
bv%(2,10): Bonus für das Vorrücken der Bauern
kv%(2,10): Werte für bv% in Eröffnung und Mittelspiel
br%(144): Das Brett
bz%(2,11): Bauernzähler
bf%(2,11,6): Bauernstandorte
diverse Hilfsvariablen

Anmerkungen:

Die Bewertung der Bauernstruktur wird für Weiß und Schwarz getrennt vorgenommen (2100-2190 bzw. 2200-2290). Mit Ausnahme der ersten Aktion (Blockierung der Zentrumsbauern prüfen) werden die Stellungsmerkmale linienorientiert untersucht. Dies geschieht in der Schleife mit der Variablen i im Zeilenbereich 2122-2190 bzw. 2222-2290. Die eingeschachtelte Schleife mit der Variablen j (Zeilen 2160-2186 bzw. 2260-2286) sorgt dafür, daß das Vorrücken der Bauern und die rückständi-

gen Bauern nicht nur einmal pro Linie (wie die isolierten Bauern, Mehrfachbauern und Freibauern) sondern genauso oft bewertet werden, wie Bauern auf der Linie postiert sind.

Sämtliche Einzelbewertungen werden in der Hilfsvariablen hw% aufsummiert und an das Steuerungsmodul (siehe 4.6.1) übergeben.

2106-2112: Setzen des linienbezogenen Bonus für das Vorrücken der Bauern in Abhängigkeit vom Partiestadium:
Eröffnung, Mittelspiel - Werte aus kv% (werden in den Zeilen 152 und 208 per DATA-Anweisung eingelesen)
Endspiel - konstante Werte (=200)
Da die Endspielabfrage nach Farben getrennt durchgeführt wird, muß auch bv% für Weiß und Schwarz vorhanden sein und getrennt gesetzt werden.

2118-2120: Feststellung von Blockierungen durch Untersuchung des Brettes br%

2124: Schleifendurchlauf auslassen, falls kein Bauer auf der entsprechenden Linie vorhanden

2128: Feststellung isolierter Bauern durch Überprüfung der Nachbarlinien (Hier machen sich die imaginären Linien 2 und 11 bezahlt, da durch sie die Randlinien a (=3) und h (=10) genauso wie die anderen Linien behandelt werden können.)

2132-2134: Feststellung der Mehrfachbauern (Information direkt aus bz%)

2137-2148: Feststellung der Freibauern durch Betrachtung der gegnerischen Bauern auf der aktuellen und den beiden Nachbarlinien; Linien mit Freibauern werden in fb% gekennzeichnet (Damit können in der Abschätzung der Königsaktivität die Abstände

zwischen den Königen und den Bauern entsprechend der Bedeutung der Bauern gewichtet bewertet werden (siehe 4.6.5).)

2150-2157: Untersuchung der Umwandlungschancen

2158: Bewertung der Freibauern

2166: Bonus in Abhängigkeit von der Anzahl vorgerückter Felder (=0 für die 2. Reihe!)

2178-2180: Überprüfung der Nachbarlinien auf Bauerndeckung bzw. eigene Bauernkontrollen des Stopfeldes bezüglich des aktuell betrachteten Bauern

2182-2184: Überprüfung der Nachbarlinien auf gegnerische Kontrollen des Stopfeldes; Einträge für rückständige Bauern in rb% analog zur Benutzung von fb% für Freibauern (Bewertung der Königsaktivität)

2200-2290: analog zu 2100-2190 für die schwarzen Bauern

4.6.4 Die Figurenbewertung

```
2300 REM BEWERTUNG FUER DIE WEISSEN FIGUREN
2302 A1%=0:A2%=1:A3%=-1:A4%=-1:A9%=-1
2304 A5%=1000:IF ES%(1) OR ES%(2) THEN A5%=1500
2308 FOR I=2 TO FZ%(1)
2310 S%=FS%(1,I)
2312 ON (FI%(1,I)-FF%) GOTO 2360,2317,2329,2337,2357
2314 GOTO 2360
2316 REM SPRINGER
2317 A9%=A9%+1
2318 IF S%<39 THEN A1%=A1%+1:HW%=HW%-470
2320 A6%=INT(S%/12)
2322 A6%=ABS(6.5-(S%-12*A6%))+ABS(5.5-A6%)
2324 HW%=HW%+60*(6-2*A6%)
2326 GOTO 2360
2328 REM LAEUFER
```

```
2329 A9%=A9%+1
2330 IF S%<39 THEN A1%=A1%+1:HW%=HW%-550
2332 IF A4% THEN A4%=0:GOTO 2360
2334 HW%=HW%+400:GOTO 2360
2336 REM TUERME
2337 A9%=A9%+1
2338 IF (S%<99)OR(S%>106) THEN GOTO 2342
2340 HW%=HW%+500:IF FS%(2,1)>106 THEN HW%=HW%+600
2342 A6%=S%-12*INT(S%/12)
2344 IF BZ%(1,A6%)>0 THEN GOTO 2348
2346 HW%=HW%+150:IF BZ%(2,A6%)=0 THEN HW%=HW%+250
2348 IF A3%<0 THEN A3%=A6%:GOTO 2352
2350 IF A6%=A3% THEN HW%=HW%+800
2352 IF FB%(1,A6%) OR FB%(2,A6%) THEN HW%=HW%+A5%
2354 GOTO 2360
2356 REM DAMEN
2357 A9%=A9%+3
2358 A2%=S%
2360 NEXT I
2362 IF (A1%>0)AND(A2%>46) THEN HW%=HW%-250*A1%
2364 RF%(1)=A9%
2400 REM BEWERTUNG FUER DIE SCHWARZEN FIGUREN (ANALOG ZU 2302-2364)
2402 A1%=0:A2%=144:A3%=-1:A4%=-1:A9%=1
2404 A5%=-1000:IF ES%(1) OR ES%(2) THEN A5%=-1500
2408 FOR I=2 TO FZ%(2)
2410 S%=FS%(2,I)
2412 ON (FF%-FI%(2,I)) GOTO 2460,2417,2429,2437,2457
2414 GOTO 2460
2417 A9%=A9%-1
2418 IF S%>106 THEN A1%=A1%+1:HW%=HW%+470
2420 A6%=INT(S%/12)
2422 A6%=ABS(6.5-(S%-12*A6%))+ABS(5.5-A6%)
2424 HW%=HW%-60*(6-2*A6%)
2426 GOTO 2460
2429 A9%=A9%-1
2430 IF S%>106 THEN A1%=A1%+1:HW%=HW%+550
2432 IF A4% THEN A4%=0:GOTO 2460
2434 HW%=HW%-400:GOTO 2460
2437 A9%=A9%-1
2438 IF (S%>46)OR(S%<39) THEN GOTO 2442
```

```
2440 HW%=HW%-500:IF FS%(1,1)<39 THEN HW%=HW%-600
2442 A6%=S%-12*INT(S%/12)
2444 IF BZ%(2,A6%)>0 THEN GOTO 2448
2446 HW%=HW%-150:IF BZ%(1,A6%)=0 THEN HW%=HW%-250
2448 IF A3%<0 THEN A3%=A6%:GOTO 2452
2450 IF A6%=A3% THEN HW%=HW%-800
2452 IF FB%(1,A6%) OR FB%(2,A6%) THEN HW%=HW%+A5%
2454 GOTO 2460
2457 A9%=A9%-3
2458 A2%=S%
2460 NEXT I
2462 IF (A1%>0)AND(A2%<99) THEN HW%=HW%+250*A1%
2464 RF%(2)=A9%
2466 RETURN
```

Variablenliste:

es%(2):	Ergebnis der Endspielabfrage
fz%(2):	Figurenzähler
fs%(2,16):	Figurenstandorte
fi%(2,16):	Figurenlisten
bz%(2,11):	Bauernzähler
fb%(2,10):	Information über Freibauern
rf%(2)	Figurenzähler (ohne Bauern)

diverse Hilfsvariablen

Anmerkungen:

Analog zur Bauernbewertung wird auch hier die Bewertung für Weiß und Schwarz getrennt durchgeführt. Ebenfalls analog werden die Einzelbewertungen in hw% aufsummiert. Um auf schnellem Wege an alle Figuren (mit Ausnahme der Könige und der Bauern) heranzukommen, ohne das gesamte Brett absuchen zu müssen, ist die Verwendung der Figurenlisten sehr hilfreich. Dementsprechend besteht die Bewertung für eine Farbe im wesentlichen aus einer Schleife (mit der Variablen i; Zeilenbereich 2308-2360 für Weiß bzw. 2408-2460 für Schwarz), die nacheinander die Figuren aus der Liste bewertet. Das Innere der Schleife teilt sich in einzelne Abschnitte, die jeweils die Stellungsmerkmale speziell für einen Figurentyp untersuchen.

Um zu diesen Programmteilen zu gelangen, findet zu Beginn der Schleife eine Fallunterscheidung nach dem Typ der aktuell behandelten Figur mit anschließender Verzweigung in den zugeordneten Abschnitt statt.

2302-2304: Initialisierung der Hilfsvariablen, die folgende Bedeutung haben:

- a1%: Anzahl der Leichtfiguren auf der Grundreihe
- a2%: Damenstandort
- a3%: für die Bewertung der Turmverdopplung (=-1: noch keinen Turm gefunden, sonst Angabe der Linie des Turmstandortes)
- a4%: für die Bewertung des Läuferpaares (=-1: noch keinen Läufer gefunden, =0: mindestens einen Läufer gefunden)
- a5%: für die Bewertung der Turm-Freibauern-Beziehung
- a9%: Aufsummierung der Figuren für rf%

2308: Schleifenbeginn bei 2, da für i=1 der König ausgewählt würde (siehe 4.2.1)

2310: Abspeicherung des Standortes der aktuell behandelten Figur vor der Verzweigung, da dieser für alle Einzelbewertungen benötigt wird

2318: Bewertung bei Entwicklungsrückstand eines Springers (mit Erhöhung von a1%)

2320-2324: Bewertung der Springerzentralisierung

2330: Bewertung bei Entwicklungsrückstand eines Läufers (mit Erhöhung von a1%)

2332-2334: Bewertung für vorhandenes Läuferpaar

2338-2340: Bewertung für einen Turm auf der 7. Reihe (mit Einfluß des Standortes des schwarzen Königs (=fs%(2,1))

2342: Linienangabe des Turmstandortes

2344-2346: Bewertung für einen Turm auf halboffener / offener Linie

2348-2350: Bewertung für Turmverdopplung

2352: Bewertung für eine vorhandene Turm-Freibauer-Beziehung (Turm und Freibauer auf der gleichen Linie)

2358: Abspeicherung des Standortes der Dame (wird ebenso wie a1% außerhalb der Schleife benötigt)

2362: Bewertung bei zu früh aktivierter Dame (abhängig von der Anzahl nicht entwickelter Leichtfiguren (a1%) und dem Damenstandort (a2%))

2464: Zuweisung der in a9% aufsummierten Figurenanzahl (Summenbildung in den Zeilen 2317, 2329, 2337 und 2357) an rf% (wird in der Bewertung der Königssicherheit benötigt; siehe 4.6.5)

2400-2464: analog zu 2300-2464 für die schwarzen Figuren

4.6.5 Die Königsbewertung

```
2500 REM BEWERTUNG DER KOENIGSSTANDORTE
2504 A6%=24:A7%=36:SF%=WB%
2510 REM FUER WEISS UND SCHWARZ GEMEINSAMER BEWERTUNGSTEIL
2512 FOR I=1 TO 2:REM ORGANISIERT ALS SCHLEIFE
2514 S%=FS%(I,1):A2%=INT(S%/12):A1%=S%-12*A2%:A3%=7-A1%
2516 IF ES%(I) THEN GOTO 2598
2518 REM EROEFFNUNGS- UND MITTELSPIELPOSITIONEN (-> KOENIGSSICHERHEIT)
2522 A4%=FS%(3-I,1):A5%=A4%-12*INT(A4%/12)
```

```
2524 IF (ABS(A3%)>1)OR(ABS(7-A5%)<=1) THEN KW%=0:GOTO 2532
2526 IF (BZ%(1,A1%)>0)OR(BZ%(2,A1%)>0) THEN KW%=100:GOTO 2532
2528 KW%=200
2530 REM EXPONIERTE KOENIGSSTELLUNG
2532 IF ABS(A2%-INT(A6%/12))>1 THEN HW%=HW%+RF%(3-I)*(KW%+2000):GOTO
2632
2534 A8%=1000:A9%=1000
2536 IF A3%<0 THEN GOTO 2558
2538 REM UNTERSUCHUNG DES DAMENFLUEGELS
2540 A8%=0
2542 FOR J=3 TO 5:REM BAUERNSCHUTZ
2544 IF BR%(A7%+J)=SF% THEN GOTO 2550
2546 IF BZ%(I,J)>0 THEN A8%=A8%+50:GOTO 2550
2548 A8%=A8%+100
2550 NEXT J
2552 IF A1%<=5 THEN HW%=HW%+RF%(3-I)*A8%:GOTO 2632
2553 IF RO%(I,1) THEN A8%=A8%+50:GOTO 2555
2554 A8%=A8%+200
2555 FOR J=4 TO 6:REM ENTWICKLUNGSDAUER
2556 IF BR%(A6%+J)<>FF% THEN A8%=A8%+50
2557 NEXT J
2558 IF A3%>0 THEN GOTO 2584
2560 REM UNTERSUCHUNG DES KOENIGSFLUEGELS
2562 A9%=0
2564 FOR J=8 TO 10
2566 IF BR%(A7%+J)=SF% THEN GOTO 2572
2568 IF BZ%(I,J)>0 THEN A9%=A9%+50:GOTO 2572
2570 A9%=A9%+100
2572 NEXT J
2574 IF A1%>=9 THEN HW%=HW%+RF%(3-I)*A9%:GOTO 2632
2575 IF RO%(I,2) THEN A9%=A9%+50:GOTO 2577
2576 A9%=A9%+200
2577 FOR J=8 TO 9
2578 IF BR%(A6%+J)<>FF% THEN A9%=A9%+50
2579 NEXT J
2580 REM AUSWAHL EINER BRETTHAELFTE ZUR BEWERTUNG
2584 IF A8%<A9% THEN A9%=A8%
2585 HW%=HW%+RF%(3-I)*(KW%+A9%)
2590 GOTO 2632
2592 REM ENDSPIELPOSITIONEN (-> KOENIGSAKTIVITAET)
```

```
2596 REM ZENTRALISIERUNG
2598 KW%=400*(ABS(6.5-A1%)+ABS(5.5-A2%))
2601 REM ABSTAND ZU EIGENEN UND GEGNERISCHEN BAUERN
2602 A6%=0:A7%=0:A9%=WB%
2604 FOR J=1 TO 2
2606 FOR K=2 TO FZ%(J)
2608 IF FI%(J,K)<>A9% THEN GOTO 2624
2610 ZF%=FS%(J,K):A5%=INT(ZF%/12):A4%=ZF%-12*A5%
2612 IF FB%(J,A4%) THEN A8%=6:GOTO 2618
2614 IF RB%(J,A4%)<>0 THEN A8%=3:GOTO 2618
2616 A8%=2
2618 A4%=ABS(A1%-A4%):A5%=ABS(A2%-A5%)
2620 A6%=A6%+A8%
2622 A8%=A8%*(A4%+A5%)
2623 A7%=A7%+A8%
2624 NEXT K
2626 A9%=SB%
2628 NEXT J
2630 IF A6%>0 THEN KW%=KW%+600*(A7%/A6%-6)
2631 HW%=HW%+(FF%-SF%)*KW%
2632 A6%=108:A7%=96:SF%=SB%
2634 NEXT I:REM ENDE DER BEWERTUNG FUER EINE FARBE
2636 RETURN
```

Variablenliste:

fs%(2,16):	Figurenstandorte
es%(2):	Ergebnis der Endspielabfrage
bz%(2,11):	Bauernzähler
rf%(2):	Figurenanzahl (ohne Bauern) (=fs% in Kapitel 3)
br%(144):	Das Brett
ro%(2,2):	Rochaderechte
fz%(2,16):	Figurenzähler
fi%(2,16):	Figurenlisten
fb%(2,10):	Information über Freibauern
rb%(2,10):	Information über rückständige Bauern

diverse Hilsvariablen

Anmerkungen:

In diesem Programmteil wird im Gegensatz zur Bauern- und Figurenbewertung der gleiche Programmcode für die Bewertung von Weiß und Schwarz verwendet. Dadurch wird einiges an Speicherplatz eingespart - zumal die Königsbewertung relativ umfangreich ist. Der Aufbau dieses Unterprogrammes gestaltet sich wie folgt: Um die Königsbewertung zweimal (für Weiß und für Schwarz) ausführen zu können, wird sie in eine Schleife eingebaut (mit der Variablen i; Zeilenbereich 2512-2634). Diese Schleife wird zweimal durchlaufen; dabei findet jeweils die Bewertung für einen Königsstandort statt. Damit dies mit den gleichen Abfragen geschehen kann, werden Hilfsvariablen benutzt, die vor der Schleife (Zeile 2504) auf die für Weiß geeigneten Werte und am Ende des 1. Durchlaufes (Zeile 2632 innerhalb der Schleife) auf die für Schwarz geeigneten Werte gesetzt werden. Der Schleifeninhalt besteht im wesentlichen aus zwei Teilen, je einem für die Bewertung der Königssicherheit bzw. der Königsaktivität. Durch die Abfrage von es% (Zeile 2516) wird zur dem Partiestadium entsprechenden Abschätzung verzweigt. Dort werden die in Kapitel 3 beschriebenen Stellungsmerkmale bewertet.

2504: Initialisierung der Hilfsvariablen (für die Bewertung des weißen Königs), die folgende Bedeutung haben:

- a6%: Berechnung der Grundreihenfelder
- a7%: Berechnung der Felder der 2. bzw. 7. Reihe
- sf%: Schutzfigur (wb% bzw. sb%)

2514: Abspeicherung der Informationen über den Königsstandort: Reihenangabe (a2%), Linienangabe (a1%), Abstand zur E-Linie (a3%)

2522: Linienangabe des gegnerischen Königs (a5%)

2524-2528: Überprüfung der Relation der Königsstandorte und Setzen der Hilfsvariablen kw% (=r% in Kapitel 3)

2532: Bewertung bei exponierter Königsstellung (Formel KS1 aus Kapitel 3)

2534: Initialisierung der Hilfsvariablen für die Berechnung der Schutzwerte:

a8%: für den Damenflügel (=sw%(1) in Kapitel 3)
a9%: für den Königsflügel (=sw%(2) in Kapitel 3)

2536: Falls der König auf der F-, G- oder H-Linie steht, wird keine Abschätzung der Königssicherheit für den Damenflügel vorgenommen. Dadurch bleibt a8% auf seinem hohen Initialisierungswert und fällt bei der späteren Minimumbildung heraus.

2540: eigentliche Initialisierung für das Aufsummieren über die Schwachstellen in der Königsumgebung

2542-2550: Bewertung des Bauernschutzes (Berechnung der Bauernfelder A2, B2 und C2 für Weiß bzw. A7, B7 und C7 für Schwarz mit Hilfe der Variablen a7%)

2552: Bewertung nach Ausführung der langen Rochade (Formel KS3 aus Kapitel 3); Eine Rochadestellung wird anhand der Linienangabe des Königsstandortes festgestellt. Dadurch sind falsche Abschätzungen in Positionen mit auf A1 bzw. A8 "eingeklemmtem Turm zwar theoretisch möglich, kommen aber in der Praxis kaum vor.

2553-2554: Abschätzung der benötigten Anzahl von Königszügen zum Erreichen der Rochadestellung

2555-2557: Abschätzung der benötigten Anzahl von Figurenzügen zum Beenden der Entwicklung (Berechnung der Figurenfelder auf der Grundreihe (B1, C1, D1 bzw. B8, C8, D8) mit Hilfe der Variablen a6%)

2558-2579: Bewertung der Königssicherheit auf dem Königsflügel (analog zu 2536-2557)

2584-2585: Bewertung vor Ausführung der Rochade (Formel KS2 aus Kapitel 3)

2598: Bewertung der Zentralisierung des Königs (1. Stellungsmerkmal der Königsaktivität)

2602: neue Verwendung der Hilfsvariablen in Endspielpositionen:

- a6%: Aufsummierung der "Bedeutungswerte" aller Bauern (Zeile 2620)
- a7%: Aufsummierung der nach Bedeutung gewichteten Abstände aller Bauern zum König (Zeile 2623)
- a9%: aktuell betrachteter Figurentyp (wb%/sb%)

2604-2606: zweifache Schleife, um die Bauern beider Seiten zu erfassen

2610: Abspeicherung der Reihenangabe (a5%) und der Linienangabe (a4%) des betrachteten Bauern

2612-2616: Einschätzung der Bedeutung des Bauern

2618: Berechnung des Linienabstandes (a4%) und des Reihenabstandes (a5%) zwischen dem König und dem betrachteten Bauern

2630: Bewertung des mittleren König-Bauern-Abstandes (2. Stellungsmerkmal der Königsaktivität)

2631: Gesamtbewertung der Königsaktivität; Durch den Faktor "ff%-sf%" wird das der Farbe zugehörige Vorzeichen ("-" für Weiß bzw. "+" für Schwarz) verwendet.

2632: Umsetzen der Hilfsvariablen für die Bewertung des schwarzen Königs

4.6.6 Die Matt-Routinen

```
3100 REM MATTSETZEN DES SCHWARZEN KOENIGS
3102 A2%=INT(FS%(1,1)/12):A1%=FS%(1,1)-12*A2%
3104 A9%=INT(FS%(2,1)/12):A8%=FS%(2,1)-12*A9%
3106 REM ZENTRALISIERUNG DES SCHWARZEN KOENIGS
3108 HW%=94*(ABS(6.5-A8%)+ABS(5.5-A9%))
3110 REM ABSTAND VON KOENIG UND SPRINGERN (WEISS) ZUM SCHWARZEN KOENIG
3112 A3%=2
3114 A5%=ABS(A1%-A8%):A6%=ABS(A2%-A9%)
3116 HW%=HW%+16*(14-(A5%+A6%))
3118 FOR I=A3% TO FZ%(1)
3120 IF FI%(1,I)<>WS% THEN GOTO 3128
3122 A3%=I+1
3124 A2%=INT(FS%(1,I)/12):A1%=FS%(1,I)-12*A2%
3126 GOTO 3114
3128 NEXT I
3130 REM GESAMTBEWERTUNG DER STELLUNG
3132 PW%=MB%+INT(HW%/10)
3134 RETURN
3200 REM MATTSETZEN DES WEISSEN KOENIGS (ANALOG ZU 3100-3134)
3202 A2%=INT(FS%(2,1)/12):A1%=FS%(2,1)-12*A2%
3204 A9%=INT(FS%(1,1)/12):A8%=FS%(1,1)-12*A9%
3208 HW%=94*(ABS(6.5-A8%)+ABS(5.5-A9%))
3212 A3%=2
3214 A5%=ABS(A1%-A8%):A6%=ABS(A2%-A9%)
3216 HW%=HW%+16*(14-(A5%+A6%))
3218 FOR I=A3% TO FZ%(2)
3220 IF FI%(2,I)<>SS% THEN GOTO 3228
3222 A3%=I+1
3224 A2%=INT(FS%(2,I)/12):A1%=FS%(2,I)-12*A2%
3226 GOTO 3214
3228 NEXT I
```

```
3232 PW%=MB%-INT(HW%/10)
3234 RETURN
3999 REM@P
```

Variablenliste:

fs%(2,16): Figurenstandorte
fz%(2): Figurenzähler
fi%(2,16): Figurenlisten
mb%: Materialbalance
diverse Hilfsvariablen

Anmerkungen

3102: Abspeichern der Reihenangabe (a2%) und der Linienangabe (a1%) des mattsetzenden Königs

3104: Abspeichern der Reihenangabe (a9%) und der Linienangabe (a8%) des mattzusetzenden Königs

3108: Bewertung der Zentralisierung des mattzusetzenden Königs

3114-3116: Bewertung des Abstandes zwischen den beiden Königen

3118-3128: Heraussuchen eventuell vorhandener Springer aus der Figurenliste der mattsetzenden Seite zur Abstandsbewertung

3132: Gesamtbewertung für Stellungen während der Mattführung (Materialbalance + positioneller Anteil)

3200-3234: Matt-Routine für Schwarz (analog zu 3100-3134)

4.7 Das Vertiefen von Zügen und der Stack (4000-4999)

Im Verlauf von Spiel und Suche muß das Programm Züge ausführen, also deren Folgestellung berechnen. Dies erledigt die Routine "Vertiefen" (4300-4499). Bei der Suche tritt dabei Rekursion auf. Was dies bedeutet und wie sie mit dem Stack gelöst wird, wurde in Abschnitt 3.1.3 beschrieben. Im Programm verwenden wir den Speicher-Stack bzw. den inkrementellen Speicher-Stack. Anschaulich gesprochen: der Stack muß es ermöglichen, die alte Stellung und diverse Variablen-Werte zu rekonstruieren.

a) Das Vertiefen (4300-4499)

```
4300 REM VERTIEFEN
4302 A1%=2+W%
4304 A2%=FA%(S%)
4306 BR%(S%)=FF%:FA%(S%)=0
4308 IF BR%(Z%)=FF% THEN GOTO 4316
4310 A3%=FA%(Z%)
4311 MS%(3-A1%)=MS%(3-A1%)-FW%(BR%(Z%))
4312 FI%(3-A1%,A3%)=FF%
4314 FS%(3-A1%,A3%)=0
4316 IF ZA%<>22 THEN GOTO 4330
4322 A5%=FA%(EP%)
4323 MS%(3-A1%)=MS%(3-A1%)-FW%(BR%(EP%))
4324 BR%(EP%)=FF%:FA%(EP%)=0
4326 FI%(3-A1%,A5%)=FF%
4328 FS%(3-A1%,A5%)=0
4330 BR%(Z%)=ZF%
4332 FA%(Z%)=A2%
4334 FS%(A1%,A2%)=Z%
4335 IF ZA%<>24 THEN GOTO 4338
4336 MS%(A1%)=MS%(A1%)+FW%(NF%)-FW%(BR%(Z%)):BR%(Z%)=NF%:FI%(A1%,A2%)
=NF%
4338 MB%=MB%+ZG%
4339 IF ZA%<>21 THEN GOTO 4360
4340 A3%=Z%-1:A4%=Z%+1
4341 IF Z%<S% THEN A3%=Z%+1: A4%=Z%-2
4342 BR%(A3%) = BR%(A4%): BR%(A4%)=FF%
```

```
4344 FA%(A3%) = FA%(A4%): FA%(A4%)=0
4346 FS%(A1%,FA%(A3%))=A3%
4348 RO%(A1%,1)=0: RO%(A1%,2)=0
4360 IF T%>3 OR NOT GZ% THEN GZ%=0:GOTO4362
4361 GZ%=(HV%(1,T%,1)=S%)AND(HV%(1,T%,2)=Z%)AND(HV%(1,T%+1,1)<>0)
4362 IF T%<4 THEN HV%(T%+1,T%+1,1)=0
4364 IF (ZV%(5)=Z%)AND(ZV%(6)>=22) THEN MT%=MT%+1
4366 ZV%(5)=Z%: ZV%(6)=ZA%
4370 EP%=0
4372 IF ((ZF%=WB%)OR(ZF%=SB%))AND(ABS(Z%-S%)=24) THEN EP%=Z%
4380 IF NOT W% THEN GOTO 4390:REM ROCHADEN
4382 IF ZF%=WK% THEN RO%(1,1)=0:RO%(1,2)=0:GOTO 4387
4384 IF ZF%<>WT% THEN GOTO 4387
4385 IF S%=27 THEN RO%(1,1)=0
4386 IF S%=34 THEN RO%(1,2)=0
4387 IFF%=ST%ANDZ%=111THENRO%(2,1)=0
4388 IFF%=ST%ANDZ%=118THENRO%(2,2)=0
4389 GOTO 4400
4390 IF ZF%=SK% THEN RO%(2,1)=0:RO%(2,2)=0:GOTO 4397
4394 IF ZF%<>ST% THEN GOTO 4397
4395 IF S%=111THEN RO%(2,1)=0
4396 IF S%=118THEN RO%(2,2)=0
4397 IFF%=WT%ANDZ%=27 THENRO%(1,1)=0
4398 IFF%=WT%ANDZ%=34 THENRO%(1,2)=0
4400 REM ROCHADERECHTE FERTIG
4460 T%=T%+1
4462 W%=NOT W%
4480 IF NOT DR% THEN GOTO 4496
4481 PRINT LEFT$(BL$,T%);
4482 A0%(1)=S%:A0%(2)=Z%
4484 GOSUB 11600
4494 PRINT "(";AL%;BE%;")"
4496 SI%(3)=SI%(3)+1
4498 RETURN
```

Diese Routine führt intern einen Zug aus. Sie erwartet, daß der Zug in allen dafür zur Verfügung stehenden Variablen beschrieben ist und verändert die Stellungsdarstellung.

In der Routine bezeichnen a1% den Index des ziehenden Spielers und a2% die Nummer der ziehenden Figur (in fs% und fi%). Im ersten Teil wird die Brettdarstellung verändert (4306-4348). Je nach Zugart (za%) müssen die verschiedenen Fälle getrennt behandelt werden.

Die Aktionen sind im einzelnen:

4306:	Standort behandeln
4308-4314:	Schlagzug: geschlagene Figur streichen
4316-4328:	E.p.-Schlagzug: geschlagene Figur streichen
4330-4334:	Figur auf Zielfeld stellen
4335-4338:	Bauernumwandlung behandeln
4339-4348:	Rochade: Turm versetzen
4370-4372:	ep% setzen
4380-4400:	Rochaderechte behandeln
4460-4462:	Tiefe und Anzugsrecht behandeln

Die Zeilen 4360-4366 gehören zur Suche und werden dort erläutert. Die Anweisungen, in denen mb% und mw%() berechnet werden, werden bei der Bewertungsfunktion erklärt. Das Ende (4480-4496) dient der Protokollierung (Testausdrucke, Zählen der Züge in si%(3)).

b) Der Stack (4000-4199 und 4500-4699)

Der Stack wird durch das Array

```
170 DIM ST%(8,44)
```

realisiert. Der erste Index wird für die verschiedenen Tiefen und der zweite für die verschiedenen Informationen je Stellung bereitgestellt. Der Stack muß folgendes rekonstruierbar machen:

Stellungsdarstellung:	st%(t%,1)-st%(t%,8)
Den aktuellen Zug:	st%(t%,1)-st%(t%,8)
Die Zuglistenverwaltung:	st%(t%,11)-st%(t%,16)
Rochaderechte und ep%:	st%(t%,20)-st%(t%,24)
rekursive Variablen der Suche:	st%(t%,30)-st%(t%,36)
rekursive Variablen der Bewertung:	st%(t%,40)-st%(t%,43)

st%(t%,44) wird von der Bewertungsfunktion benutzt und dort erläutert.

b1) Stellung im Stack retten (4000-4199)

```
4000 REM STELLUNG IM STACK RETTEN
4004 ST%(T%,1)=ZA%: ST%(T%,8)=NF%
4006 ST%(T%,2)=S%
4008 ST%(T%,3)=Z%
4010 ST%(T%,4)=ZF%
4012 ST%(T%,5)=F%
4014 ST%(T%,6)=FA%(S%)
4016 ON ZA%-19GOTO 4050,4022,4030,4018,4040
4018 REM SCHLAGZUG
4020 ST%(T%,7)=FA%(Z%): GOTO 4050
4022 REM ROCHADE
4024 IF Z%>S% THEN A1%=Z%+1:GOTO4028
4026 A1%=Z%-2
4028 ST%(T%,7) = A1%:GOTO 4050
4030 REM EN-PASSANT
4032 IF ZF%>FF% THEN A1%=Z%-12:GOTO4036
4034 A1%=Z%+12
4036 ST%(T%,7)=FA%(A1%)
4038 GOTO 4050
4040 REM UMWANDLUNG
4042 ST%(T%,7)=FA%(Z%)
4044 REM ENDE BRETT RETTEN
4050 ST%(T%,11)=I1%: ST%(T%,12)=I2%
4052 ST%(T%,13)=ZV%(1):ST%(T%,14)=ZV%(2)
4054 ST%(T%,15)=ZV%(3):ST%(T%,16)=ZV%(4)
4056 ST%(T%,17)=ZV%(5): ST%(T%,18)=ZV%(6)
4100 ST%(T%,20)=RO%(1,1):ST%(T%,21)=RO%(1,2)
4102 ST%(T%,22)=RO%(2,1):ST%(T%,23)=RO%(2,2)
```

```
4104 ST%(T%,24)=EP%
4120 REM FUER DIE SUCHE
4122 ST%(T%,30)=VS%
4124 ST%(T%,31)=MW%
4126 ST%(T%,32)=SH%
4128 ST%(T%,33)=GZ%
4130 ST%(T%,34)=AL%
4132 ST%(T%,35)=BE%
4134 ST%(T%,36)=MT%
4140 REM FUER DIE BEWERTUNG
4142 ST%(T%,40)=PW%
4144 ST%(T%,41)=MS%(1)
4146 ST%(T%,42)=MS%(2)
4148 ST%(T%,43)=MB%
4198 RETURN
```

Die Routine "Stellung im Stack retten" trägt die entsprechenden Informationen ein. Sie weiß dabei schon, welcher Zug vertieft werden soll. Dies muß aus den Variablen zur Zugbeschreibung hervorgehen.

Der Zug wird in allen Einzelheiten im Stack eingetragen (4004-4044). Mit einer kleinen Zusatzinformation (st%(t%,7)) ist dann die aktuelle Stellung später rekonstruierbar - wir müssen nicht br%(), fi%(), fa%() und fs%() speichern. Hier wird der Speicher-Stack inkrementell genutzt.

Im folgenden werden die Zusatzinformationen zur Stellung (4100-4104) sowie die Zuglistenverwaltung (4050-4056) gerettet. Dem schließen sich die rekursiven Variablen von Suche (4120-4134) und Bewertung (4140-4148) an.

b2) Zug rückgängig machen (4500-4699)

```
4500 REM ZUG RUECKGAENGIG MACHEN
4502 W%= NOT W%
4504 T%=T%-1
4506 EP%=ST%(T%,24)
4510 ZA%=ST%(T%,1): NF%=ST%(T%,8)
4512 S%=ST%(T%,2)
```

```
4514 Z%=ST%(T%,3)
4516 ZF%=ST%(T%,4)
4518 F%=ST%(T%,5)
4520 FA%(S%)=ST%(T%,6)
4522 A1%=ST%(T%,7)
4530 BR%(S%)=ZF%
4532 FS%(W%+2,FA%(S%))=S%
4534 IF ZA%=24 THEN FI%(W%+2,FA%(S%))=ZF%
4536 BR%(Z%)=F%
4538 IF F%=FF% THEN GOTO 4546
4540 FA%(Z%)=A1%
4542 FI%(1-W%,A1%)=F%
4544 FS%(1-W%,A1%)=Z%
4546 IF ZA%<>22 THEN GOTO 4560
4548 A2% = WB%
4550 IF W% THEN A2%=SB%
4552 FA%(EP%)=A1%
4554 FI%(1-W%,A1%)=A2%
4556 FS%(1-W%,A1%)=EP%
4558 BR%(EP%)=A2%
4560 IF ZA%<>21 THEN GOTO 4598
4562 A3%=Z%+1: A4%=Z%-1
4564 IF Z%<S% THEN A3%=Z%-2: A4%=Z%+1
4566 BR%(A3%)=BR%(A4%): BR%(A4%)=FF%
4568 FA%(A3%)=FA%(A4%): FA%(A4%)=0
4570 FS%(W%+2,FA%(A3%))=A3%
4598 REM ENDE BRETT UND MATERIAL
4600 I1%=ST%(T%,11): I2%=ST%(T%,12)
4602 ZV%(1)=ST%(T%,13):ZV%(2)=ST%(T%,14)
4603 ZV%(3)=ST%(T%,15):ZV%(4)=ST%(T%,16)
4604 ZV%(5)=ST%(T%,17): ZV%(6)=ST%(T%,18)
4610 RO%(1,1)=ST%(T%,20):RO%(1,2)=ST%(T%,21)
4612 RO%(2,1)=ST%(T%,22):RO%(2,2)=ST%(T%,23)
4620 REM FUER DIE SUCHE
4622 VS%=ST%(T%,30)
4624 MW%=ST%(T%,31)
4626 SH%=ST%(T%,32)
4628 GZ%=ST%(T%,33)
4630 AL%=ST%(T%,34)
4632 BE%=ST%(T%,35)
```

```
4634 MT%=ST%(T%,36)
4640 REM FUER DIE BEWERTUNG
4642 PW%=ST%(T%,40)
4644 MS%(1)=ST%(T%,41)
4646 MS%(2)=ST%(T%,42)
4648 MB%=ST%(T%,43)
4698 RETURN
```

Die Routine "Zug rückgängig machen" erzeugt wieder die alte Stellung - sie arbeitet spiegelbildlich zur Routine "Zug vertiefen" - mit Hilfe des Stacks.

Nachdem Tiefe und Anzugsrecht zurückgesetzt wurden (4502, 4504), wird der ausgeführte Zug zurückgewonnen, also in die entsprechenden Variablen eingetragen (4506-4522). Anschließend werden br%(), fi%(), fs%() und fa%() zurückgewonnen (4530-4598) - auf inkrementelle Art; es müssen nur die wirklich von Zug betroffenen Felder behandelt werden. Dann werden die Werte der Zuglistenverwaltung (4600-4612), der rekursiven Suchvariablen (4620-4634) und der rekursiven Bewertungsvariablen (4640-4648) aus dem Stack geholt.

4.8 Allgemeine Hilfsroutinen (8000-8999)

Das Programm verfügt nur über wenige allgemeine Hilfsroutinen, weil die meisten einem bestimmten Modul zugeordnet werden können.

a) Bestimmung des Zuges (8000-8099)

```
8000 REM BESTIMMUNG DES ZUGES
8005 REM ZUG VON WEISS
8010 ZF%=BR%(S%):F%=BR%(Z%):ZG%=-FW%(F%)
8012 IF ZF%<>WB% THEN GOTO 8022
8014 IF Z%>110 THEN ZA%=24:ZG%=ZG%+FW%(NF%)-FW%(WB%):GOTO 8036
8016 IF F%<>FF% THEN GOTO 8034
8018 IF Z%-S%=11 OR Z%-S%=13 THEN ZA%=22:NF%=FA%(Z%-12):ZG%=-FW%(SB%)
:GOTO 8036
8020 GOTO 8032
```

```
8022 IF ZF%<>WK% THEN GOTO 8032
8023 IF ABS(Z%-S%)<>2 THEN GOTO 8032
8024 ZA%=21:A2%=0
8025 A1%=31: GOSUB8400: IF A1% THEN ZA%=0: GOTO 8036
8026 IF Z%<S% THEN GOTO 8030
8027 NF%=FA%(34)
8028 A1%=32: GOSUB 8400: IF A1% THEN ZA%=0
8029 GOTO 8036
8030 A1%=30:GOSUB8400:IFA1%THENZA%=0:GOTO8036
8031 NF%=FA%(27): GOTO8036
8032 IF F%=FF% THEN ZA%=20:GOTO 8036
8034 ZA%=23
8036 RETURN
8050 REM ZUG VON SCHWARZ
8060 ZF%=BR%(S%):F%=BR%(Z%):ZG%=-FW%(F%)
8062 IF ZF%<>SB% THEN GOTO 8072
8064 IF Z%< 35 THEN ZA%=24:ZG%=ZG%+FW%(NF%)-FW%(SB%):GOTO 8086
8066 IF F%<>FF% THEN GOTO 8084
8068 IF S%-Z%=11 OR S%-Z%=13 THEN ZA%=22:NF%=FA%(Z%+12):ZG%=-FW%(WB%)
:GOTO 8086
8070 GOTO 8082
8072 IF ZF%<>SK% THEN GOTO 8082
8073 IF ABS(Z%-S%)<>2 THEN GOTO 8082
8074 ZA%=21:A2%=-1
8075 A1%=115: GOSUB8400: IF A1% THEN ZA%=0: GOTO 8086
8076 IF Z%<S% THEN GOTO 8080
8077 NF%=FA%(118)
8078 A1%=116: GOSUB 8400: IF A1% THEN ZA%=0
8079 GOTO 8086
8080 A1%=114:GOSUB8400:IFA1%THENZA%=0:GOTO8086
8081 NF%=FA%(111): GOTO8086
8082 IF F%=FF% THEN ZA%=20:GOTO 8086
8084 ZA%=23
8086 RETURN
```

Die beiden Routinen "Zug von Weiß" und "Zug von Schwarz" gehen davon aus, daß der aktuelle Zug in den Variablen s%, z% und ggf. nf% beschrieben ist und haben die Aufgabe, die restlichen Variablen zur Zugbeschreibung zu setzen (vgl. Kap.

4.2.2). Der Einfachheit halber gibt es eine Routine für weiße Züge (8005-8036) und eine für schwarze (8050-8086).

Bauernumwandlungen werden in den Zeilen 8014/8064 und E.p.-Schlagzüge in den Zeilen 8018/8068 behandelt.

Handelt es sich bei dem Zug um eine Rochade (za%=21), so wird erst hier geprüft, ob sie unmöglich ist, weil der König im Schach steht oder eines der Zugfelder bedroht ist (8025-8031 und 8075-8081). Die Hilfsroutine 8400 wird gleich besprochen. Die letzten beiden Zeilen klären, ob es sich sonst um einen ruhigen oder um einen Schlagzug handelt.

b) König im Schach (8400-8499)

```
8400 REM KOENIG IM SCHACH?
8402 REM@M
8404 IF NOT A2% THEN GOTO 8408
8406 IF BR%(A1%-11)=WB% OR BR%(A1%-13)=WB% THEN GOTO 8440
8407 GOTO 8410
8408 IF BR%(A1%+11)=SB% OR BR%(A1%+13)=SB% THEN GOTO 8440
8410 A3%=SS%: IF A2% THEN A3%=WS%
8412 FOR J=1 TO 8
8414 IFBR%(A1%+SW%(J))=A3%THENGOTO8440
8416 NEXT J
8418 IF A2%THEN A4%=WD%:A5%=WL%:A6%=WT%:GOTO 8422
8420 A4%=SD%:A5%=SL%:A6%=ST%
8422 FOR J=1 TO 8
8424 A3%=A1%
8426 A3%=A3%+DW%(J): A7%=BR%(A3%)
8428 IF A7%=FF% THEN GOTO 8426
8430 IF A7%=A4% THEN GOTO 8440
8432 IF J<5 AND A7%=A5% THEN GOTO 8440
8434 IF J>4 AND A7%=A6% THEN GOTO 8440
8436 NEXT J
8438 A1%=0: GOTO 8442
8440 A1%=-1
8442 REM@P
8444 RETURN
```

Diese Routine ermittelt, ob ein Feld von einem Spieler angegriffen ist. Sie wird nur dann benötigt, wenn festgestellt werden soll, ob ein König im Schach steht.

Dabei ist a1% das Feld, das überprüft werden soll, und die (hier) logische Hilfsvariable a2% gibt an, ob ein Angriff von Weiß auf den schwarzen König gesucht werden soll (ja) oder einer von Schwarz auf den weißen König (nein). Das Ergebnis soll in der Variablen a1% stehen (,die dann als logische Variable aufzufassen ist).

8404-8408: Wird der König von einem Bauern angegriffen?
8410-8416: Wird der König von einem Springer angegriffen?
8418-8436: Wird der König von Dame / Läufer / Turm angegriffen?

Auf Angriffe durch den gegnerischen König muß nicht geprüft werden, da dies nach den Spielregeln unmöglich ist.

c) Andere Hilfsroutinen

Weitere Hilfsroutinen sind:

Zug vertiefen
Stack-Behandlung
Zug in Zugliste suchen und ggf. streichen

Die ersten beiden wurden in Abschnitt 4.7 erläutert, die dritte wird in Kapitel 4.9 beschrieben

4.9 Die Suche und spezifische Hilfsroutinen (500-6999)

Die Suche wurde in Abschnitt 3.4 ausführlich erläutert. Dort wurden die Ideen und Methoden vorgestellt.

Das Suchmodul besteht aus zwei Teilen - aus den Hilfsroutinen (5000-5999) und den eigentlichen Suchroutinen (6000-6999)

Zum Teil realisieren die Hilfsroutinen Ansätze, die wir in Kaptiel 3.4 kennengelernt haben, zum Teil sind sie nur technische Hilfsmittel:

5100-5299: Der nächste Zug (Zugselektion, vgl. 3.4.8)
5300-5399: Zug suchen und streichen (technische Hilfe)
5400-5499: Neuen besten Zug merken (Berechnung der Hauptvariante, vgl. 3.4.7)
5500-5599: Initialisierungen (technische Hilfen)
5600-5699: König illegal im Schach? (technische Hilfe)
5700-5799: Killerzug eintragen (Killerheuristik, vgl. 3.4.8)

Die eigentlichen Suchroutinen sind:

6100-6299: Suche für Weiß (vgl. 3.4.2-3.4.5)
6300-6499: Suche für Schwarz (vgl. 3.4.2-3.4.5)
6800-6999: Aufruf der Suche (iterative Suche, vgl. 3.4.7)

4.9.1 Die Hilfsroutinen (5000-5999)

a) Der nächste Zug (5100-5299)

```
5000 REM HILFSROUTINEN DER SUCHE
5100 REM DER NAECHSTE ZUG
5104 IF NOT ((ZV%(4)<6) OR VS%) THEN S%=0: GOTO 5290
5106 ON ZV%(4) GOTO 5110,5130,5190,5150,5150,5170
5110 REM GEMERKTER BESTER ZUG
5112 IF NOT GZ% THEN ZV%(4)=2: GOTO5104
5114 S%=HV%(1,T%,1)
5116 Z%=HV%(1,T%,2)
5117 NF%=HV%(1,T%,3)
5118 GOSUB 5300
5120 ZV%(4)=2
5122 IF NOT A1% THEN GOTO 5104
5124 GOTO 5290
5130 REM SCHLAGZUEGE, BESONDERE
5132 A1%=ZV%(2)+1
5134 IF A1%>I2% THEN ZV%(4)=4:GOTO 5104
5136 S%=Z2%(A1%,1):Z%=Z2%(A1%,2):NF%=Z2%(A1%,3)
```

```
5138 ZV%(2)=A1%
5140 IF S%=0ORZ%<>ZV%(5)THEN GOTO 5132
5142 Z2%(A1%,1)=0: GOTO 5290
5150 REM KILLERZUG
5152 IF ZV%(4)=4 THEN A1%=1:ZV%(4)=5:GOTO 5154
5153 A1%=3:ZV%(4)=3
5154 S%=KI%(T%,A1%)
5155 IF S%=0 THEN GOTO 5104
5156 Z%=KI%(T%,A1%+1)
5160 GOSUB 5300
5162 IF NOT A1% THEN GOTO 5104
5166 GOTO 5290
5170 REM RUHIGE ZUEGE
5172 A1%=ZV%(1)+1
5174 IF A1%>I1% THEN S%=0: GOTO 5290
5176 S%=Z1%(A1%,1):Z%=Z1%(A1%,2)
5178 ZV%(1)=A1%
5180 IF S%=0 THEN GOTO 5172
5182 GOTO 5290
5190 REM SCHLAGZUEGE, NORMALE
5192 A1%=ZV%(3)+1
5194 IF A1%>I2% THEN ZV%(4)=6:GOTO 5104
5196 S%=Z2%(A1%,1):Z%=Z2%(A1%,2):NF%=Z2%(A1%,3)
5198 ZV%(3)=A1%
5200 IF S%=0ORZ%=ZV%(5)THEN GOTO 5192
5202 GOTO 5290
5290 RETURN
```

In Abschnitt 3.4.8 wurde die Reihenfolge, in der Züge untersucht werden sollen, festgelegt. Die Routine "der nächste Zug" übernimmt diese Selektion. Bei jedem Aufruf übergibt sie mit den Variablen s%, z% und eventuell nf% den richtigen Zug an die Suche. Findet sie keinen Zug mehr, setzt sie s% auf 0.

Wie weit sie bisher selektiert hat, merkt sie sich in dem Array zv%(). zv%(1), zv%(2), zv%(3) und zv%(4) werden vom Zuggenerator initialisiert und zeigen an, wo die Zuglisten für diese

Stellung sind (vgl. 4.2.3) und in welcher Selektionsphase die Suche ist (zv%(4)):

zv%(1): Beginn der Liste der ruhigen Züge
zv%(2): Beginn der Schlagzugliste
zv%(3): Beginn der Schlagzugliste

Die Belegungen von zv%(4) entsprechen den Selektionsphasen:

zv%(4)	Selektionsphase (mit Zeilenbereich)
1	Zug der Hauptvariante liefern (5110-5124)
2	Schlagzüge, die zurückschlagen (5130-5142)
3	übrige Schlagzüge (5190-5202)
4	erster Killerzug (5150-5156)
5	zweiter Killerzug (5150-5156)
6	ruhige Züge (5170-5178)

zv%(4) durchläuft die Werte in der Reihenfolge

1,2,4,5,3,6.

Die Selektionsroutine bleibt für mehrere Aufrufe in den Phasen 2, 3 und 6 - sie kann ja in einer Stellung mehrere Schlagzüge oder ruhige Züge liefern. In jeder Phase sieht sie die betreffende Zugliste von links nach rechts durch. Wenn sie einen Zug betrachtet hat, erhöht sie zv%(1) (in Phase 6), zv%(2) (in Phase 2) oder auch zv%(3) (in Phase 3). So findet sie beim nächsten Aufruf den richtigen Aufsetzpunkt.

In den Selektionsphasen 2 und 3 muß bekannt sein, welches das Zielfeld des vorigen Zuges war - davon hängt es ab, ob der Schlagzug zurückschlägt. Dies steht in zv%(5). zv%(5) wird beim Vertiefen (4366) gesetzt.

In den Phasen 1, 4 und 5 soll die Routine einen vorgeschlagenen Zug der Hauptvariante oder Killerheuristik liefern. Sie muß kontrollieren, ob dieser Zug überhaupt in der Zugliste steht und ihn dann dort streichen, damit er nicht noch einmal untersucht wird. Dies übernimmt eine Hilfsroutine (gosub 5300).

gz% gibt an, ob die Suche sich noch in der Hauptvariante der vorigen Iteration befindet (vgl. 3.4.8.b). gz% muß an verschiedenen Stellen im Programm berechnet werden: 4360, 4361 und 6835.

Die Selektionsroutine prüft an verschiedenen Stellen, ob ein Zug in der aktuellen Suchphase übergeben werden darf. (In der Ruhesuche sollen keine ruhigen Züge untersucht werden.) Dies geschieht jeweils mit der Variablen vs%.

b) Spezialzug suchen und streichen (5300-5399)

```
5300 REM SPEZIALZUG STREICHEN
5310 REM A) SCHLAGZUG
5312 A1%=0
5314 IF ZV%(3)>=I2% THEN GOTO 5340
5316 FOR J=ZV%(3)+1 TO I2%
5318 IF Z2%(J,1)<>S% THEN GOTO 5338
5320 IF Z2%(J,2)<>Z% THEN GOTO 5338
5322 IF (BR%(S%)<>WB%)OR(Z%<111) THEN GOTO 5328
5324 IF NF%=Z2%(J,3) THEN GOTO 5332
5326 GOTO 5338
5328 IF (BR%(S%)=SB%)AND(Z%<39) THEN GOTO 5324
5330 NF%=Z2%(J,3)
5332 A1%=-1
5334 Z2%(J,1)=0
5336 GOTO 5370
5338 NEXT J
5340 REM B) RUHIGER ZUG
5342 A1%=0: IF NOT VS% THEN GOTO 5370
5344 IF ZV%(1)>=I1% THEN GOTO 5370
5346 FOR J=ZV%(1)+1 TO I1%
5348 IF Z1%(J,1)<>S% THEN GOTO 5358
5350 IF Z1%(J,2)<>Z% THEN GOTO 5358
5352 A1%=-1
5354 Z1%(J,1)=0
5356 GOTO 5370
5358 NEXT J
5370 RETURN
```

Dies ist die eben angesprochene Hilfsroutine. Sie setzt voraus, daß s% und z% einen gesuchten Zug beschreiben. Diesen sucht die Routine in der Zugliste. Findet sie ihn, wird a1% auf "ja" gesetzt und der Zug aus der Liste gestrichen.

5310-5338: Durchmustern der Schlagzüge. Im Fall einer Bauernumwandlung (5322) wird auch nf% auf Übereinstimmung geprüft.

5340-5358: Durchmustern der ruhigen Züge (in der Ruhesuche unnötig (5342))

c) Hauptvariante berechnen (5400-5512)

```
5400 REM NEUEN BESTEN ZUG MERKEN
5410 IF T%>=4 THEN GOTO 5424
5412 FOR J=T%+1 TO 4
5414 HV%(T%,J,1)=HV%(T%+1,J,1)
5416 HV%(T%,J,2)=HV%(T%+1,J,2)
5418 HV%(T%,J,3)=HV%(T%+1,J,3)
5420 HV%(T%+1,J,1)=0
5422 NEXT J
5424 IF T%>4 THEN GOTO 5440
5426 HV%(T%,T%,1)=S%
5428 HV%(T%,T%,2)=Z%
5430 HV%(T%,T%,3)=NF%
5440 RETURN
5500 REM HV% LOESCHEN
5502 FOR J=1 TO 4
5504 FOR J1=1 TO 4
5506 HV%(J,J1,1)=0
5508 NEXT J1
5510 NEXT J
5512 RETURN
```

In Abschnitt 3.4.7.c wurde ein Verfahren angegeben, wie die Hauptvariante einer Suche berechnet werden kann. Hierzu wird ein Array benötigt, in unserem Programm

174 DIM HV%(4,4,3).

Die eigentliche Berechnung übernimmt die Routine "neuen besten Zug merken" (5400-5499). Sie trägt den aktuellen besten Zug (der durch s%, z% und nf% dargestellt sein muß) in hv% ein (5424-5430) und überträgt ggf. die Folgezüge (5410-5422).

Das Verfahren verlangt auch, daß beim Vertiefen eines Zuges durch die Suche ein Feld von hv% gelöscht wird. Dies geschieht in Zeile 4362 beim Vertiefen.

Die Routine "Hauptvariante löschen" steht in den Zeilen 5500-5512.

d) Die Killerheuristik (5700-5725 und 5520-5528)

```
5700 REM KILLERZUG EINTRAGEN
5705 IF Z%=ZV%(5) THEN GOTO 5725
5710 IF KI%(T%,1)=S% AND KI%(T%,2)=Z% THEN GOTO 5725
5715 KI%(T%,3)=KI%(T%,1):KI%(T%,4)=KI%(T%,2)
5720 KI%(T%,1)=S%:KI%(T%,2)=Z%
5725 RETURN

5520 REM KI% INITIALISIEREN
5522 FOR J=1 TO 4
5524 KI%(J,1)=0
5526 NEXT J
5528 RETURN
```

Die Killerheuristik wurde in Abschnitt 3.4.8.a motiviert und erläutert. Die Liste im Programm ist:

```
176 DIM KI%(8,4).
```

Sie kann je Tiefenstufe (erster Index) zwei Züge durch Standort und Zielfeld beschreiben: den ersten Killerzug durch ki%(t%,1) und ki%(t%,2) sowie den zweiten Killerzug durch ki%(t%,3) und ki%(t%,4).

Die Routine "Killerzug eintragen" (5700-5725) trägt den aktuellen besten Zug, der beim Aufruf durch s% und z% beschrieben sein muß, in die Liste ein. Ggf. wird dabei der bisherige erste

Killerzug zum Zweiten (5715). Züge, die die zuletzt gezogene Figur schlagen, werden nicht eingetragen, da dies wahrscheinlich nur in dieser einen Stellung gut ist.

Die Routine "Killerliste initialisieren" (5520-5528) wird zu Beginn eines Suchlaufs aufgerufen.

e) König illegal im Schach? (5600-5699)

```
5600 REM IST DER GEGNERISCHE KOENIG ILLEGAL IM SCHACH?
5602 A1%=0
5604 IF ZV%(2)>=I2% THEN GOTO 5616
5606 A2%=FS%(1-W%,1)
5610 FOR J=ZV%(2)+1 TO I2%
5612 IF Z2%(J,2)=A2% THEN A1%=-1:GOTO 5616
5614 NEXT J
5616 RETURN
```

Diese Routine durchforstet die Schlagzugliste und schaut nach, ob der König des Spielers, der nicht am Zug ist, geschlagen werden kann. Das Ergebnis liefert sie in a1%.

4.9.2 Die Suche (6000-6999)

Das Suchmodul besteht aus drei Routinen:

Stellung analysieren, Weiß am Zug (6100-6299)
Stellung analysieren, Schwarz am Zug (6300-6499)
Iterative Suche/Steuerung (6800-6999)

Folgende Variablen werden verwendet:

mt%:	Suchtiefe
t%:	Tiefe der aktuellen Stellung
mw%:	der gesuchte Minimax-Wert einer Stellung
al%:	Alpha
be%:	Beta
vs%:	(logische Variable) noch erschöpfende, volle Suche?
aw%:	Wert des aktuellen Zugs

p%:	(logische Variable) liegt Patt vor?
dr%:	(logische Variable) Testausdrucke?
sh%:	(logische Variable) König im Schach?
pw%:	Wert der Bewertungsfunktion
a1%:	Hilfsvariable für GOSUB-Aufruf
a2%:	Hilfsvariable für GOSUB-Aufruf
si%():	Array
zv%():	Array

si% (1) speichert den Startwert von Alpha bei einer Iteration, si%(2) den Startwert von Beta. Mit si%(3) werden die untersuchten Stellungen gezählt.

In zv%(5) wird das Zielfeld des Vorgängerzugs, mit zv%(6) die Zugart des Vorgängerzugs gespeichert. Dies wird für die variable Suchtiefe benötigt.

Die Suche wurde in Kapitel 3.4 ausführlich, und mit Listing, beschrieben. Am Listing haben sich nur die Zeilennummern geändert, und zusätzlich mußten wenige technische Vorkehrungen getroffen werden (Testausdrucke, Hilfsvariablen für GOSUB-Aufrufe). Ein Vergleich des Programmlistings mit den Listings in Abschnitt 3.4 zeigt dies.

Ein Verständnis der Suchroutinen kann hier nicht vermittelt werden - die Folgen der einzelnen Anweisungen lassen sich nicht mit wenigen Anmerkungen erklären. Hierzu muß man Kapitel 3.4 durcharbeiten.

Daher werden hier nur die (technischen) Änderungen (gegenüber 3.4) kommentiert.

a) Stellung analysieren, Weiß am Zug (6100-6299)

```
6000 REM DIE SUCHE
6100 REM SUCHE FUER WEISS
6105 GOSUB 1900: REM ZUEGE GENERIEREN
6107 GOSUB 5600: REM ILLEGALE STELLUNG?
6108 IF A1% THEN MW%= 16001-T%:GOTO6280
6110 A1%=FS%(1,1):A2%=0:GOSUB 8400
```

```
6111 SH%=A1%:REM SCHACH?
6112 VS%= T%<MT%
6115 IF VS% THEN MW%=-16000+T%:GOTO6135
6120 GOSUB 2000: REM BEWERTUNG
6121 IF DR%THEN PRINTLEFT$(BL$,T%);"BEWERTUNG";PW%
6126 IF SH% THEN MW%=-16000+T%:GOTO6135
6130 MW%=PW%
6135 IF T%>8 THEN GOTO6280
6136 IF MW%<BE% GOTO 6139
6137 IF DR%THEN PRINT LEFT$(BL$,T%);"FPW1"
6138 GOTO 6280
6139 IF MW%>AL% THEN AL%=MW%
6140 GOSUB 5100: REM NAECHSTER ZUG
6145 IF S%=0 THEN GOTO 6230
6150 GOSUB 8005: REM ZUG ANALYSIEREN
6151 IF ZA%=0 THEN GOTO 6220
6152 IF T%<=MT% THEN GOTO 6156
6153 IF PW%+ZG%+80>AL% THEN GOTO 6156
6154 AW%=PW%+ZG%+80: IF DR% THEN PRINT LEFT$(BL$,T%+1);"FPW2"
6155 GOTO 6185
6156 GOSUB 4000: REM STELLUNG RETTEN
6160 GOSUB 4300: REM ZUG VERTIEFEN
6170 GOSUB 6300: REM REKURSIVER AUFRUF
6171 AW%=MW%
6175 GOSUB 4500: REM ZUG ZURUECKNEHMEN
6185 IF AW%<=MW% THEN GOTO 6220
6190 MW%=AW%
6195 IF MW%<= AL% THEN GOTO 6220
6200 GOSUB 5400: REM HAUPTVARIANTE
6205 GOSUB 5700: REM KILLERZUG
6210 IF MW%<BE% THEN GOTO 6215
6211 IF DR% THEN PRINTLEFT$(BL$,T%+1);"CUTOFF -------"
6212 GOTO6260
6215 AL%=MW%
6219 REM EXTRA ZEITABBRUCH
6220 IF T%=1 AND MT%>2 AND SI%(1)<MW% AND SI%(3)>(ZE%+ZE%) THEN GOTO
6260
6225 GOTO 6140
6230 REM SUCHE FERTIG?
6235 IF VS% OR (NOT SH%) THEN GOTO 6260
```

```
6236 IF MW%>=PW% THEN GOTO 6260
6237 IF PW%<=AL% THEN MW%=PW%:GOTO 6260
6238 IF PW%<BE% THEN BE%=PW%
6239 VS%=-1:GOTO 6140:REM SCHACHMATT?
6260 P%=0: REM PATT?
6264 IF MW%=-16000+T% THEN IF NOT SH% THEN MW%=0:P%=-1
6280 IF DR% THEN PRINT LEFT$(BL$,T%);"WERT=";MW%
6290 RETURN
```

Diese Routine ist fast vollständig in Abschnitt 3.4.5.b abgelistet und erklärt, nur mit anderen Zeilennummern. Die Änderungen beschränken sich auf technische Details.

So sind überall, wo dr% auftaucht, zusätzlich Ausgaben vorgesehen, die manchmal etwas umständlich programmiert werden mußten. (Manche Cutoff-Abfrage mußte negiert werden, um im Cutoff-Fall einen Testausdruck zu ermöglichen).

Unser Stack reicht nur bis zur Tiefe 8. Zeile 6135 sorgt dafür, daß nicht tiefer gesucht wird.

In Zeile 6140 wird auf vereinfachte Weise der nächste Zug ausgewählt. Die Probleme nimmt uns die Hilfsroutine ab. Sind die Züge erschöpft, so setzt sie s% auf 0. Um alle Variablen der Zugbeschreibung zu setzen, wird in Zeile 6150 eine Routine aufgerufen. Bei Rochaden wird erst hier geprüft, ob sie wirklich möglich sind. Wenn nicht, erhält za% den Wert 0 und der Zug ist unmöglich (6151).

In den Zeilen 6200 und 6205 werden Killerliste und Hauptvariante berechnet.

In Zeile 6220 wurde ein zusätzlicher Zeitabbruch eingeführt. Befindet sich die Suche in der Grundstellung (t%=1), ist die Suchtiefe mindestens 2 (mt% größer als 2), wurde ein guter Zug gefunden (liegt der Minimaxwert schon über dem Startwert von Alpha, mw% größer als si%(1)) und wurde die vorgesehene Zahl der Stellungen von der Suche deutlich überschritten (si%(3) größer als ze%+ze%), so wird die Suche beendet (goto 6260).

Zusätzlich wurde die logische Variable p% aufgenommen. Sie gibt an, ob eine Pattstellung vorliegt (6260,6264).

b) Stellung untersuchen, Schwarz am Zug (6300-6499)

```
6300 REM SUCHE FUER SCHWARZ (ANALOG ZU 6100-6290)
6305 GOSUB 1900
6307 GOSUB 5600
6308 IF A1% THEN MW%=-16001+T%:GOTO6480
6310 A1%=FS%(2,1):A2%=-1:GOSUB 8400
6311 SH%=A1%
6312 VS%= T%<MT%
6315 IF VS% THEN MW%= 16000-T%:GOTO6335
6320 GOSUB 2000
6321 IF DR% THEN PRINTLEFT$(BL$,T%);"BEWERTUNG";PW%
6326 IF SH% THEN MW%= 16000-T%:GOTO6335
6330 MW%=PW%
6335 IF T%>8 THEN GOTO6480
6336 IF MW%>AL% THEN GOTO 6339
6337 IF DR% THEN PRINT LEFT$(BL$,T%);"FPS1"
6338 GOTO 6480
6339 IF MW%<BE% THEN BE%=MW%
6340 GOSUB 5100
6345 IF S%=0 THEN GOTO 6430
6350 GOSUB 8050
6351 IF ZA%=0 THEN GOTO 6420
6352 IF T%<=MT% THEN GOTO 6356
6353 IF PW%+ZG%-80<BE% THEN GOTO 6356
6354 AW%=PW%+ZG%-80:IF DR% THEN PRINT LEFT$(BL$,T%+1);"FPS2"
6355 GOTO 6385
6356 GOSUB 4000
6360 GOSUB 4300
6370 GOSUB 6100
6371 AW%=MW%
6375 GOSUB 4500
6385 IF AW%>=MW% THEN GOTO 6420
6390 MW%=AW%
6395 IF MW%>= BE% THEN GOTO 6420
6400 GOSUB 5400
6405 GOSUB 5700
```

```
6410 IF MW%>AL% THEN GOTO 6415
6411 IF DR% THEN PRINT LEFT$(BL$,T%+1);"CUTOFF ------"
6412 GOTO 6460
6415 IF MW%<BE% THEN BE%=MW%
6420 IF T%=1 AND MT%>2 AND SI%(2)>MW% AND SI%(3)>(ZE%+ZE%) THEN GOTO
6460
6425 GOTO 6340
6430 REM SUCHE FERTIG?
6435 IF VS% OR (NOT SH%) THEN GOTO 6460
6436 IF MW%<=PW% THEN GOTO 6460
6437 IF PW%>=BE% THEN MW%=PW%:GOTO 6460
6438 IF PW%>AL% THEN AL%=PW%
6439 VS%=-1:GOTO 6340
6460 P%=0
6464 IF MW%= 16000-T% THEN IF NOT SH% THEN MW%=0:P%=-1
6480 IF DR% THEN PRINT LEFT$(BL$,T%);"WERT=";MW%
6490 RETURN
```

In der Schwesterroutine (6300-6499) für Schwarz (am Zug) sind die gleichen Änderungen (gegenüber Kapitel 3.4), nur mit um 200 erhöhten Zeilennummern, vorgenommen worden.

c) Aufruf der Suche (6800-6999)

```
6800 REM AUFRUF DER SUCHE
6810 GOSUB 5500: REM HV% INITIALISIEREN
6812 GOSUB 5520: REM KI% INITIALISIEREN
6816 MT%=1
6817 AL%=-15000: BE%=15000
6818 INPUT "TESTAUSDRUCKE";ES$: DR%=ES$="J"
6820 IF SI%(3)>ZE% AND HV%(1,1,1)<>0 THEN GOTO 6960
6830 TI$="000000"
6835 T%=1: GZ%=HV%(1,1,1)<>0
6836 SI%(1)=AL%: SI%(2)=BE%: SI%(3)=0
6837 I1%=0: I2%=0
6840 IF W% THEN GOTO 6850
6845 GOSUB 6300: GOTO 6855
6850 GOSUB 6100
6855 GOSUB 11800
6856 PRINT SI%(3);"STELLUNGEN UNTERSUCHT,ZEIT=";TI$
```

```
6859 IF MW%>15000 OR MW%<-15000 OR P% THEN GOTO 6960
6860 IF (MW%>SI%(1)) AND (MW%<SI%(2)) THEN GOTO 6900
6865 PRINT "SUCHE WIEDERHOLT"
6870 AL%=-15000: BE%=15000
6875 GOTO 6835
```

Die dritte Routine "Aufruf der Suche" erledigt die Suchsteuerung bzw. die iterative Suche. Sie ist analog zum Listing in Kapitel 3.4.7, nur daß auch hier einige technische Details aufgenommen werden mußten und die Zeilen anders numeriert sind.

Die wichtigste Änderung ist:

> *Im Vergleich zur in Kapitel 3 beschriebenen Suche sind hier (und im ganzen Programm) die Werte für die Tiefe (t%, mt%) um 1 erhöht, weil wir den Index 0 nicht benutzen wollen. Die Partiestellung hat also die Tiefe 1 und nicht mehr 0.*

Zusätzlich werden hier einige Variablen initialisiert (6810-6817). Dann beginnt die Schleife (6820-6955) über die Iterationen, und auch hier werden zunächst einige Dinge initialisiert (6820-6837).

d) Variable Suchtiefe

Die variable Suchtiefe wurde entsprechend Abschnitt 3.4.6 aufgenommen. Sie wird beim Vertiefen berechnet, in den Zeilen 4364 und 4366. Weiterhin mußte deshalb die Suchtiefe mt% in den Stack aufgenommen werden (4134, 4634).

4.10 Partieschleife und Zugeingabe

```
600 GOSUB 9000:REM INITIALISIERUNG
602 GOSUB 11200:REM STELLUNGSAUSGABE
606 IF GW%=W% THEN GOTO 636
608 REM PROGRAMM AM ZUG
610 GOSUB 6800: REM ZUGBESTIMMUNG DURCH SUCHE
612 REM SONDERFAELLE
614 IF P% THEN PRINT "ICH BIN PATT!": END
```

```
616 IF W% AND (MW%>-15000) THEN GOTO 622
618 IF (NOT W%) AND (MW%<15000) THEN GOTO 622
620 PRINT "ICH GEBE AUF!": END
621 REM AUSGABE DES AUSGEWAEHLTEN ZUGES
622 PRINT "MEIN ZUG IST:  ";
623 FOR I=1 TO 2:A0%(I)=HV%(1,1,I):NEXT I
624 GOSUB 11600
625 A1%=BR%(A0%(1))
626 IF (A1%=WB%)AND(A0%(2)>106) THEN A1%=HV%(1,1,3):GOSUB 11400:GOTO 628
627 IF (A1%=SB%)AND(A0%(2)<39) THEN A1%=HV%(1,1,3):GOSUB 11400
628 PRINT
629 PRINT "DER WERT FUER DIESEN ZUG IST",MW%
630 REM AKTUALISIEREN DER PARTIESTELLUNG
631 S%=HV%(1,1,1):Z%=HV%(1,1,2):NF%=HV%(1,1,3)
632 ON (W%+2) GOSUB 8005,8050:GOSUB 4300
634 GOSUB 11200: REM STELLUNGSAUSGABE
635 IF ABS(MW%)>=15998 THEN PRINT "SIE SIND MATT!": END
636 REM GEGNER AM ZUG
638 GOSUB 9300: REM BEHANDLUNG DES GEGNERISCHEN ZUGES
640 IF S%>1 THEN GOSUB 11200: GOTO 610
998 END
999 REM@M

9300 REM EINLESEN EINES ZUGES VOM GEGNER
9301 I1%=0:I2%=0:T%=1
9302 Z$="":INPUT "GEBEN SIE BITTE IHREN ZUG EIN";Z$
9304 REM TEST AUF KORREKTE EINGABE
9305 IF LEN(Z$)<5 THEN PRINT "FALSCHE EINGABE!":S%=1:GOTO 9362
9306 ES$=MID$(Z$,4,2):GOSUB 9905:IF S%=1 THEN GOTO 9362
9308 A0%(2)=S%
9309 ES$=LEFT$(Z$,2):GOSUB 9905:IF S%=1 THEN GOTO 9362
9310 A0%(1)=S%
9311 IF MID$(Z$,7,1)="" THEN NF%=FF%:GOTO 9336
9312 ES$="DTLS"
9313 IF NOT W% THEN GOTO 9324
9314 IF MID$(Z$,7,1)<>"W" THEN GOTO 9332
9316 FOR I=1 TO 4
9318 IF MID$(Z$,8,1)=MID$(ES$,I,1) THEN NF%=13-I:GOTO 9336
9320 NEXT I
9322 GOTO 9332
```

```
9324 IF MID$(Z$,7,1)<>"S" THEN GOTO 9332
9326 FOR I=1 TO 4
9328 IF MID$(Z$,8,1)=MID$(ES$,I,1) THEN NF%=1+I:GOTO 9334
9330 NEXT I
9332 PRINT "FALSCHE FIGURANGABE!":S%=1:GOTO 9362
9334 REM TEST AUF AUSFUEHRBARKEIT (SPIELREGELN)
9336 IF (I1%+I2%)=0 THEN GOSUB 1900
9337 S%=A0%(1):Z%=A0%(2):VS%=-1
9338 GOSUB 5300
9340 IF NOT A1% THEN GOTO 9358
9342 ON (W%+2) GOSUB 8005,8050
9344 IF ZA%=0 THEN GOTO 9358
9346 GOSUB 4000
9348 GOSUB 4300
9350 GOSUB 1900
9352 GOSUB 5600
9354 IF NOT A1% THEN RETURN:REM KORREKTE ZUGEINGABE
9355 REM BEHANDLUNG EINES EINGABEFEHLERS
9356 GOSUB 4500
9358 S%=1:PRINT "ZUG NICHT AUSFUEHRBAR!"
9362 PRINT "MOECHTEN SIE DIE ZUGEINGABE WIEDERHOLEN?"
9364 ES$="":INPUT "(J=JA, SONST BELIEBIG)";ES$
9366 IF LEFT$(ES$,1)<>"J" THEN RETURN
9368 PRINT "BENOETIGEN SIE INFORMATION ZUR EINGABE?"
9370 ES$="":INPUT "(J=JA, SONST BELIEBIG)";ES$
9372 IF LEFT$(ES$,1)<>"J" THEN GOTO 9302
9373 PRINT
9374 PRINT "SIE MUESSEN URSPRUNGS- UND ZIELFELD"
9376 PRINT "IN BRETTNOTATION (A1-H8) SOWIE"
9378 PRINT "BEI BAUERNUMWANDLUNGEN ZUSAETZLICH"
9380 PRINT "DIE NEUE FIGUR DURCH ENTSPR. ABKUERZUNG"
9382 PRINT "(Z.B. WD=WEISSE DAME) EINGEBEN."
9384 PRINT "DIE FELD- UND FIGURANGABEN SIND JEWEILS"
9386 PRINT "DURCH EIN LEERZEICHEN ZU TRENNEN."
9387 PRINT
9388 GOTO 9302
```

Anmerkungen zur Partieschleife:

600: Aufruf der Stellungseingabe

602: Ausgabe der eingelesenen Stellung

606: Verzweigung zur Behandlung des Programm- bzw. des Gegnerzuges in Abhängigkeit von der Farbverteilung und dem Anzugsrecht

610: Aufruf des Such- und Entscheidungsalgorithmus

614-620: Behandlung von Positionen, in denen die Suche wegen einer Matt- oder Pattsituation keinen Zug liefern konnte, und Partieende

622-629: Ausgabe der Suchergebnisse (bester Zug, zugehöriger Minimax-Wert)

631-632: Ausführung des Zuges (Aktualisierung der Variablen zur Stellungsdarstellung)

634: Ausgabe der resultierenden Stellung

635: Partieende bei Sieg des Programmes

638: Aufruf der Zugeingabe des Gegners

640: Zugeingabe und -ausführung erfolgreich (s%>1): Ausgabe der resultierenden Stellung und Sprung zur Bestimmung des nächsten Programmzuges (Schleifenorganisation)
Zugeingabe und -ausführung fehlerhaft (s%=1): Partieende

Anmerkungen zur Zugeingabe:

9301: Vorbereitung der Analyse des einzugebenden Zuges

9302: Zugeingabe

9305-9332: Test auf korrekte Eingabeform; im einzelnen wird überprüft:

- Angabe des Ursprungsfeldes auf richtige Brettnotation (Zeilen 9309-9310; mit Hilfe des Unterprogrammes 9900-9913 (Listing siehe Abschnitt 4.4))
- Angabe des Zielfeldes auf richtige Brettnotation (Zeilen 9306-9308; wie Ursprungsfeld)
- bei Bauernumwandlungen: Angabe der eingetauschten Figur (Zeilen 9311-9332)

Fehlerfälle führen zum Setzen der Variablen s% auf den Wert "1" und zur Fehlerbehandlung (Zeilen 9362-9388)

9336: Aufruf des Zuggenerators

9337-9340: Überprüfung, ob der eingegebene Zug in den von dem Zuggenerator ermittelten Zugmöglichkeiten enthalten ist (falls nein (a1%=0): Fehlerbehandlung)

9342-9344: Bestimmung der Zusatzinformation für diesen Zug (Typ der ziehenden Figur, Zugart, u.a.)

9346: Partiestellung im Stack retten (wird wieder benötigt, falls der Zug nicht ausführbar ist)

9348: (probeweise) Ausführung des Zuges durch Aktualisierung der Variablen zur Stellungsdarstellung

9350: Aufruf des Zuggenerators in der resultierenden Stellung

9352-9354: Überprüfung, ob der gegnerische König geschlagen werden kann (falls nein (a1%=0): eingegebener Zug korrekt und daher Rücksprung in die Partieschleife mit Vollzugsmeldung (s% größer 1); sonst Fehlerbehandlung)

9356: Ausführung des illegalen Zuges rückgängig machen (Rücksetzen der Variablen zur Stellungsdarstellung auf die Werte aus dem Stack)

9358: Fehlerkennzeichnung setzen und Fehlermeldung

9362-9366: Abfrage auf Wiederholung der Zugeingabe (falls nicht erwünscht: Rücksprung in die Partieschleife mit Fehleranzeige (s%=1) zum Partieabbruch)

9368-9388: Bedienungsanleitung für die Zugeingabe auf Wunsch

4.11 Ausgaberoutinen (11000-11999)

Das Programm verfügt über 5 Ausgaberoutinen.

a) Figur drucken (11400-11499)

Diese Routine druckt die in a1% dargestellte Figurenart aus (Bsp.: WK für weißer König).

b) Feld drucken (11500-11599)

Diese Routine druckt das in a1% dargestellte Feld aus.

c) Brett ausgeben (11200-11299)

Diese Routine druckt, mit Hilfe der Routine "Figur drucken", das Brett br% aus. Mit wenigen Änderungen kann die Ausgabe (je nach Rechner-Typ) verschönert werden.

d) Zugausgabe (11600-11699)

Diese Routine gibt, mit Hilfe der Routine "Feld drucken", den in a0%(1) und a0%(2) (für Standort und Zielfeld) dargestellten Zug aus.

e) Hauptvariante drucken (11800-11899)

Diese Routine gibt die in hv%() dargestellte Hauptvariante aus.

4.12 Variablen-Referenz-Liste

a) Ganzzahlige und logische Variablen

Name	Art	rek	Bytes	Kap.	Bedeutung
a1%	-	n	-	-	Hilfsvariable
a2%	-	n	-	-	Hilfsvariable
a3%	-	n	-	-	Hilfsvariable
a4%	-	n	-	-	Hilfsvariable
a5%	-	n	-	-	Hilfsvariable
a6%	-	n	-	-	Hilfsvariable
a7%	-	n	-	-	Hilfsvariable
a8%	-	n	-	-	Hilfsvariable
a9%	-	n	-	-	Hilfsvariable
al%	g	r	2	4.9	Alpha
aw%	g	n	2	4.9	Aktueller Wert des untersuchten Zugs
be%	g	r	2	4.9	Beta
dr%	l	n	1	4.9	Testausdrucke
ep%	g	r	1	4.2.1	En-passant-Feld

f%	g	r	1	4.2.2	Art der geschlagenen Figur
ff%	g	k	1	4.2.1	Freies Feld
gw%	l	n	1	4.4	Der Gegner des Programms hat Weiß
gz%	l	r	1	4.9	Noch in Hauptvariante
hw%	-	n	-	-	Hilfsvariable
i1%	g	r	2	4.2.3	Index/Füllungsgrad der Liste der ruhigen Züge
i2%	g	r	2	4.2.3	Index/Füllungsgrad der Schlagzugliste
kf%	g	k	1	4.2.1	kein Feld (des Bretts)
kw%	-	n	-	-	Hilfsvariable
mb%	g	r	2	4.6.1	Materialbalance
mt%	g	r	1	4.9	Suchtiefe
mw%	g	r	2	4.9	Minimax-Wert
nf%	g	r	1	4.2.2	Art der neuen Figur bei Bauernumwandlung
p%	l	n	1	4.9	Patt
pt%	g	n	1	4.6.2	Positionstyp
pw%	g	r	2	4.6.1	Wert der Stellung/Bewertung
rs%	g	n	2	4.6.1	positionelle Abschätzung während der Ruhesuche
s%	g	r	1	4.2.2	Standort der ziehenden Figur
sb%	g	k	1	4.2.1	Schwarzer Bauer
sd%	g	k	1	4.2.1	Schwarze Dame
sf%	-	n	-	-	Hilfsvariable
sh%	l	r	1	4.2.1	König im Schach
sk%	g	k	1	4.2.1	Schwarzer König
sl%	g	k	1	4.2.1	Schwarzer Läufer
ss%	g	k	1	4.2.1	Schwarzer Springer
st%	g	k	1	4.2.1	Schwarzer Turm
t%	g	n	1	4.9	Tiefe der aktuellen Stellung
vs%	l	r	1	4.9	Volle Suche

w%	l	n	1	4.2.1	Weiß am Zug
wb%	g	k	1	4.2.1	Weißer Bauer
wd%	g	k	1	4.2.1	Weiße Dame
wk%	g	k	1	4.2.1	Weißer König
wl%	g	k	1	4.2.1	Weißer Läufer
ws%	g	k	1	4.2.1	Weißer Springer
wt%	g	k	1	4.2.1	Weißer Turm
z%	g	r	1	4.2.2	Zielfeld der ziehenden Figur
za%	g	r	1	4.2.2	Art des Zugs
ze%	g	n	2	4.9	Zahl der Stellungen bis Zeitabbruch
zf%	g	r	1	4.2.2	Art der ziehenden Figur
zg%	g	r	2	4.2.2	Zugewinn an Material durch Zug

Die Variablen sind alphabetisch geordnet.

Bedeutung der Spalten:

1. Spalte: l = logische, g = ganzzsahlige Variable
2. Spalte: k = konstant, r = rekursiv, n = nicht rekursive, i = inkrementell rekursiv
3. Spalte: Zahl benötigter Bytes für Wertebereich
4. Spalte: Erklärt in Kapitel
5. Spalte: Schlagwort zur Bedeutung

b) Arrays

Name	dim	Art	rek	Bytes	Kap.	Bedeutung
a0%	2	g	n	1	-	Hilfsvariable für die Zugausgabe
bf%	2,11,6	g	n	1	4.6.2	Bauern-standorte
bv%	2,10	g	n	2	4.6.3	Bonus für das Vor-rücken der Bauern
bz%	2,11	g	n	1	4.6.2	Bauernzähler
br%	144	g	i	1	4.2.1	Das Brett
dw%	8	g	k	1	4.5	Damenweite / Distanz
es%	2	1	n	1	4.6.2	Ergebnis der Endspiel-abfrage
fa%	118	g	i	1	4.2.1	Figuren-adressen / Verweise in Figurenlisten
fb%	2,10	1	n	1	4.6.3	Information über Frei-bauern
fi%	2,16	g	i	1	4.2.1	Figurenliste: Figurenarten
fs%	2,16	g	i	1	4.2.1	Figurenliste: Standorte
fw%	12	g	k	2	4.6.1	Figurenwerte für Material-bewertung
fz%	2	g	n	1	4.2.1	Figurenzähler
hv%	4,4,3	g	n	1	4.9	Haupt-variante
ki%	4,4	g	n	1	4.9	Killerliste

ko%	118	g	k	1	4.6.1	Felder-kontrollen (Zentrums-wichtung)
kv%	10	g	k	2	4.6.3	Werte für bv% in Eröffnung und Mit-telspiel
ms%	2	g	r	2	4.6.1	Materialsum-men der Spieler
rb%	2,10	g	n	2	4.6.3	Information über rück-ständige Bauern
rf%	2	g	n	1	4.6.4	Figurenzähler (ohne Bau-ern)
ro%	2,2	l	r	1	4.2.1	Rochade-rechte
si%	3	g	n	2	4.9	Suchinfor-mationen
st%	8,44	g	-	2	4.7	Der Stack
sw%	8	g	k	1	4.5	Springerweite / Distanz
z1%	400,2	g	n	1	4.2.3	Array für ruhige Züge
z2%	100,3	g	n	1	4.2.3	Array für Schlagzüge und Bauern-umwandlun-gen
zv%	6	g	r	2	4.2.3	Zuglistenver-waltung, vgl. auch 4.9

Bedeutung der Spalten:

1. Spalte: Dimension des Arrays
2. Spalte: Art der einzelnen Elemente (l = logische, g = ganzzahlige Variable)
3. Spalte: r=rekursiv, i = inkrementell rekursiv, k = konstant, n = nicht rekursiv
4. Spalte: Mindestspeicherbedarf je Element des Arrays
5. Spalte: Erläutert in Kapitel
6. Spalte: Schlagwort zur Bedeutung

c) Stringvariablen

Name	rek	Bedeutung
bl$	k	Hilfskonstante für Ausgabe
es$	n	Hilfsvariable für Eingabe
ws$	k	Hilfskonstante für Ausgabe
z$	n	Hilfsvariable für Eingabe
zk$	k	Hilfskonstante für Ausgabe

Bedeutung der Spalten:

1. Spalte: k = konstant, n = nicht rekursiv
2. Spalte: Schlagwort zur Bedeutung

5. So spielt man Schach gegen Computer

Spätestens, wenn Sie unser Programm abgetippt haben oder Sie sich die zu Ihrem Rechner passende Diskette zum Buch beschafft haben, verfügen Sie auch über einen elektronischen Schachpartner. Vielleicht fragen Sie sich schon die ganze Zeit, was Sie denn mit einem Schachprogramm (bzw. einem Schachcomputer) eigentlich anfangen können.

Klar, Sie können zu jeder Tages- und Nachtzeit das Gerät einschalten und eine gemütliche Partie spielen. Aber dafür brauchen Sie keine besonderen Kenntnisse über Computerschach. Sie werden auf der Ihnen genehmen Spielstufe gegen ein Programm antreten, mal verlieren, mal gewinnen und irgendwann das Interesse am Schachprogramm (oder dem Schachcomputer) verlieren.

Sie können aber auch etwas vom Computerschach lernen. Nach der bisherigen Lektüre haben Sie sicher eine Ahnung davon bekommen, nach welchen Algorithmen ein Schachprogramm funktioniert. Jetzt werden Sie lernen, welche Unterschiede es zwischen den verschiedenen Programmen gibt und wie sich diese bemerkbar machen. Mit jeder Partie, die Sie mit dem neuerworbenen Wissen spielen, werden Sie aber auch Ihre Spielstärke verbessern - das Schachprogramm wird zum Trainingspartner.

Für diejenigen, die bisher zu ihrem Leidwesen gegen Schachcomputer meistens verloren haben, bieten wir eine Menge Tricks, wie Sie so ein Programm (fast) nach Belieben schlagen können; das stärkt das Selbstvertrauen und die Gewißheit, daß der Mensch immer noch Chef am Brett ist.

Zunächst werden wir noch einmal detailliert erklären, welche verschiedenen Typen von Programmen es gibt und wie man bei einem unbekannten Programm feststellen kann, welche Strategie es anwendet. Dann zeigen wir, wie das Gedächtnis eines Programmes, die Eröffnungsbibliothek, funktioniert, wie man feststellt, welche Varianten gespeichert sind und wie man ein Programm schon in der Eröffnungsphase irritieren kann.

Das Mittelspiel ist die Domäne der Schachprogramme, wir präsentieren Fallen, die Sie dem Computer stellen können und erklären, warum die meisten Programme darauf hereinfallen. Im Endspiel ist der menschliche Spieler immer noch besser, also müssen Sie 'nur' lernen, Ihren Vorteil aus dem Mittelspiel in einen Sieg umzusetzen.

5.1 Testen Sie Ihr Schachprogramm

Wenn Sie mit Ihrem Schachprogramm oder Ihrem Schachcomputer nutzbringend umgehen wollen, müssen Sie es (bzw. ihn) kennenlernen. Folgende Kriterien sind charakteristisch für ein Programm:

1) Welche Strategie wird angewandt:
   - Shannon-A
   - Shannon-B?

2) Spielt das Programm
   - agressiv,
   - aktiv oder
   - passiv?

3) Verfügt das Programm über eine Eröffnungsbibliothek?

4) Wenn ja,
   - wie umfangreich ist diese?
   - welche Varianten sind enthalten?
   - werden Zugumstellungen erkannt?

5) Wie stark werden positionelle Kriterien gewertet?

6) Wieviele Halbzüge tief rechnet das Programm?

7) Rechnet daß Programm weiter, wenn der Gegner am Zug ist?

8) Beherrscht das Programm Endspiele mit König und Turm gegen den König?

9) Benutzt das Programm alle Bauernumwandlungen?

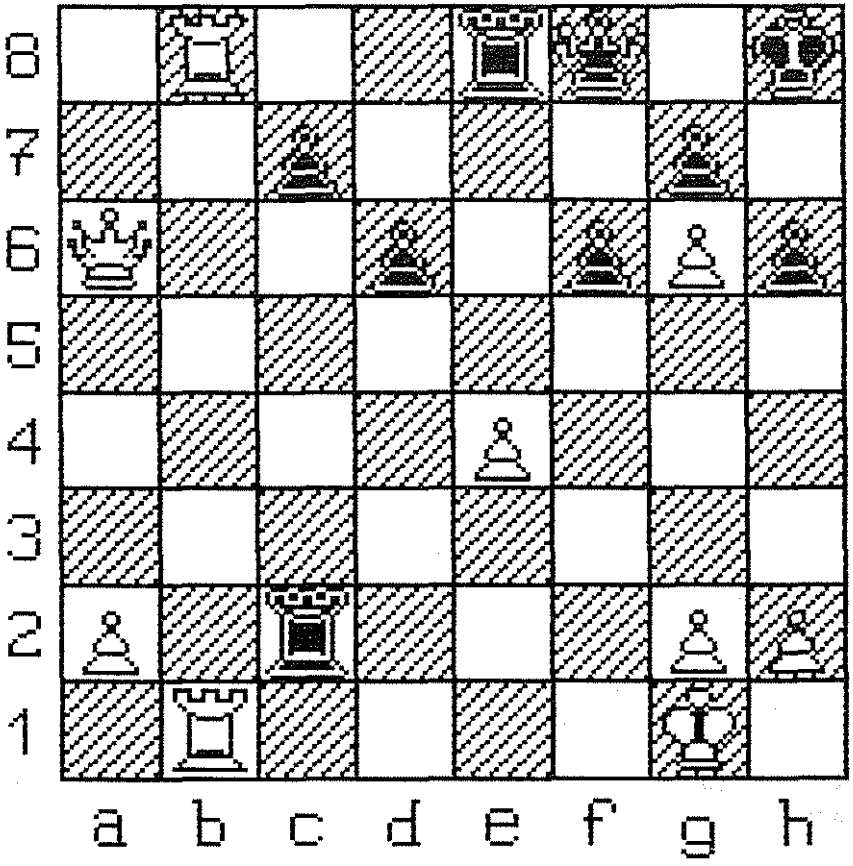
Beginnen wir beim Grundsätzlichen.

5.1.1 Welche Strategie?

Wie Sie bereits gehört haben, gibt es zwei grundsätzlich verschiedene Strategien bei Schachprogrammen; einmal die Brute Force-Strategie, bei der theoretisch jeder Zug jeder Variante berechnet und bewertet wird und die selektive Strategie, bei der von vorneherein nur die 'plausiblen' Züge untersucht werden.

Als erstes sollten Sie also testen, welchen Strategietyp Ihr Programm anwendet. Zwar ist die Wahrscheinlichkeit, an ein Shannon-B-Programm zu geraten, relativ gering, trotzdem sollten Sie den folgenden kleinen Test einmal durchführen.

Die Stellung stammt aus einer Partie des berühmten Turniers von Hastings 1919. Jose Raoul Capablanca, der spätere Weltmeister, spielt gegen einen ziemlich unbekannten Briten. Capablanca ist mit den weißen Steinen in der folgenden Stellung am Zug.



Capablanca zieht hier - scheinbar plausibel - die Dame nach A8. Geben Sie die Partie Ihrem Programm einmal vor. Was passiert?

Ein Shannon-B-Programm wird den wirklich gespielten Zug 1. Da8 - gleichgültig, wie lange es rechnen darf - nicht wählen. Zwar wird diese Fortsetzung erwogen, aber, weil selektiv

suchende Programm in der Regel tiefer rechnen, als schlecht erkannt. Nach

1. Da6-a8 ...

kann Schwarz mit

1. ... Tc2xa2

alle Türme abtauschen und dürfte das Endspiel mit einem Mehrbauern sogar gewinnen.

Brute Force-Programme sehen diese Fortsetzung je nach Rechentiefe nicht. Sie können also diese Stellung eingeben und das Programm starten. Wählen Sie die niedrigste Spielstufe und lassen Sie dann das Programm rechnen. Ein Programm mit selektiver Suche wird nur folgende Züge betrachten:

1. Da6-b7, Da6-a7 und höchstens noch Da6-b5, während ein Brute Force-Programm den Damenzug nach A8 auch berechnet.

Noch interessanter wird es, wenn man den Zug 1. Da6-a8 ausführt und das Programm die Entgegenung berechnen läßt: selektiv suchende Programme finden die richtige Entgegenung 1. ... Txa2 auf Anhieb!

Leider ist der Zug 1. ... Txa2 kein Beweis dafür, daß Sie ein Programm von Shannon-B-Typ haben; auch Brute Force-Programme finden diese Fortsetzung; und zwar dann, wenn Sie die Schlagzüge besonders intensiv und tief berechnen.

Letzendlich kann nur der Programmierer bzw. die Herstellerfirma definitiv mitteilen, welcher Art das Programm ist. Tests, wie der hier gezeigte, können nur ein Anhaltspunkt sein.

5.1.2 Initiativtest

Weit genauer läßt sich feststellen, wie aktiv ein Programm spielt. Frieder Schwenkel und Hans-Peter Ketterling, zwei renommierte Computerschach-Experten, haben einen sogenannten 'Initiativtest' entwickelt, der quantifizierbare Aussagen über den Aktivitätsgrad von Schachprogrammen zuläßt.

Lassen Sie das Programm mit den schwarzen Figuren spielen, Sie selbst ziehen Weiß. Aus der Grundstellung heraus ziehen Sie

1. Sb1-a3

und im zweiten Zug wieder zurück

2. Sa3-b1.

Diese sogenannten 'Nullzüge' zwingen das Programm dazu, die Initiative zu ergreifen. In der Folge spielen Sie nach Möglichkeit immer den gleichen Zug Sb1-a3, Sa3-b1; nur wenn Ihr König im Schach steht oder der Springer geschlagen wurde, wählen Sie andere Züge.

Ist der Springer vom Brett, ziehen Sie solange es geht Ta1-b1 bzw. Tb1-a1. Schachgeboten weichen Sie nach Möglichkeit aus, wobei der König möglichst nahe beim Ursprungsfeld E1 bleiben sollte. Kann der König nicht ausweichen, setzen Sie eine Figur zwischen die schachgebende Figur und den König; und zwar immer die wertvollste. Im Notfall schlagen Sie die schachgebende schwarze Figur.

In diesen Fällen sollten Sie jeweils den für Weiß schlechtesten Zug aussuchen.

Stellen Sie die niedrigste oder zweitniedrigste Stufe ein, unterbinden Sie - falls vorhanden - das Permanent Brain und den Zufallsgenerator. Sie werden staunen, wie unterschiedlich sich Programme verhalten.

Und so können Sie aus den Ergebnissen Rückschlüsse ziehen:

Es zählen die Züge bis zum Matt durch Schwarz. Gewinnt Schwarz unterwegs die weiße Dame, dürfen 2 bis 4 Züge subtrahiert werden. Daraus ergibt sich folgende Tabelle (nach: Computer-Schach & Spiele, Heft 5, Oktober/November 1985):

Züge bis Matt	Bewertung
weniger als 6	sehr agressiv
6 bis 7	agressiv
8 bis 12	aktiv
13 bis 20	zurückhaltend
21 bis 30	sehr zurückhaltend
mehr als 30	passiv

Optimal ist die Stufe "aktiv". Die wird nämlich dadurch erreicht, daß das Programm erst schnell die Figuren entwickelt (und eventuell noch rochiert), dann die Zentrumskontrolle übernimmt und anschließend den Mattangriff vorträgt. Eine klassisches Beispiel liefert der CC Sensory 9 (dessen implementiertes Programm übrigens aus der SARGON-Familie stammt):

Weiß: Tester
Schwarz: Sensory 9 auf Stufe 2 ohne Permanent Brain

01	Sb1-a3	e7-e6 07	Tb1-a1	d7-d5
02	Sa3-b1	Sg8-f6 08	Ta1-b1	e6-e5
03	Sb1-a3	Lf8xa3 09	Tb1-a1	Lc8-g4
04	Ta1-b1	La3-b4 10	Ta1-b1	Sf5-e4
05	Tb1-a1	0-0 11	Tb1-a1	Lb4-c5
06	Ta1-b1	Sb8-c6 12	Ta1-b1	Lc5-f2++

Diese Partie kann jederzeit mit völlig identischem Verlauf nachgespielt werden. Interessant können Versuche mit höheren Spielstufen sein; besonders Programme mit sehr umfangreichen Bewertungsfunktionen spielen bei mehr Rechenzeit anders - aus

den Abweichungen lassen sich Schlüße auf die Bewertung positioneller Faktoren ziehen. Aber dazu später mehr.

Auch unser Demonstrations-Programm schlägt sich im Initiativ-Test wacker!

Weiß: Tester
Schwarz: Demoschach (Stufe: 4)

01	Sb1-a3	e7-e5	08	Ta1-b1	e5-e4
02	Sa3-b1	Sg8-f6	09	Tb1-a1	d5-d4
03	Sb1-a3	Lf8xa3	10	Ta1-b1	e4-e3
04	Ta1-b1	La3-b4	11	Tb1-a1	Sc6-e5
05	Tb1-a1	Sb8-c6	12	Ta1-b1	c7-c5
06	Ta1-b1	d7-d5	13	Tb1-a1	Sf6-e4
07	Tb1-a1	Lc8-d7	14	Ta1-b1	e3xf2++

Spielt man mit umgekehrten Farben - also Computer mit Weiß, Tester mit Schwarz - tun sich alle Programme schwerer. SARGON III zeigt trotzdem, wie es geht:

Weiß: SARGON III auf IBM PC, Stufe 2E
Schwarz: Tester

01	b2-b3	Sb8-a6	11	Tf1-e1	Sb8-a6
02	Lc1-b2	Sa6-b8	12	Ta1-c1	Sa6-b8
03	g2-g3	Sb8-a6	13	h2-h3	Sb8-a6
04	Lf1-g2	Sa6-b8	14	a2-a3	Sa6-b8
05	Sg1-f3	Sb8-a6	15	c4-c5	Sb8-a6
06	0-0	Sa6-b8	16	Lg2-f1	Sa6-b8
07	c2-c4	Sb8-a6	17	Lf1-b5	Sb8-a6
08	Sb1-c3	Sa6-b8	18	Sf3-e5	Sa6-b8
09	d2-d4	Sb8-a6	19	Dd1-h5	Sb8-a6
10	e2-e4	Sa6-b8	20	Dh5-f7++	

Wenn Sie mehrere Programme mit diesem Kriterium vergleichen wollen, sollten Sie in jedem Fall mit jedem Programm mehrere

Partien spielen, wobei Sie unterschiedliche Spielstufen testen können. Der Mittelwert aus etwa 10 bis 12 Initiativtests dürfte dann repräsentativ sein.

5.1.3 Eröffnungsbibliothek

Nach den grundsätzlichen Eigenschaften eines Programmes wollen wir nun das Verhalten in der Eröffnung betrachten. Die erste Frage, die sich stellt, ist die, ob das Programm eine Eröffnungsbibliothek enthält.

Nahezu alle kommerziellen Programme verfügen über eine solche Sammlung von Eröffnungsvarianten. Ob überhaupt Eröffnungen gespeichert sind, läßt sich ganz einfach feststellen. Spielen Sie als Führer der weißen Figuren einfach

1. e2-e4

Wenn Schwarz nicht sofort, d.h. ohne spürbare Bedenkzeit antwortet, dann ist im Programm keine Bibliothek enthalten. Testen Sie diesen Effekt am Demonstrations-Programm aus dem Buch; es wird Ihren Eröffnungszug erst nach der üblichen Rechendauer mit

1. ... e7-e5

beantworten.

Programme ohne Bibliothek benutzen stellenweise einen Zufallsgenerator. Der sorgt dafür, daß ein Eröffnungszug des Spielers nicht jedesmal mit dem gleichen Gegenzug beantwortet wird.

Ob ein Zufallsgenerator im Spiel ist, können Sie ebenfalls leicht feststellen: eröffnen Sie in 10 bis 12 Partien mit dem gleichen Zug; wenn wenigstens einmal ein neuer Gegenzug vom Computer erfolgt, ist ein Zufallsgenerator implementiert. In der

Regel entscheidet der Zufallsgenerator zwischen folgenden Alternativen:

1. e2-e4 e7-e5 oder c7-c5 oder Sb8-c6;
1. d2-d4 d7-d5 oder Sg8-f6.

Beginnt der Computer, bietet ein Zufallsgenerator meistens

1. e2-e4 oder d2-d4

an.

Die schiere Anzahl an implementierten Eröffnungsvarianten läßt sich testweise praktisch nicht erkunden; selbst kleinere Bibliotheken (z.B. bei MEPHISTO II, Sensory 9, SARGON II für Commodore 64 etc.) lernt man mit der Zeit kennen. Um dieses Kennenlernen zu beschleunigen, kann man folgende Methode anwenden:

Eröffnen Sie mit Weiß

1. e2-e4.

Ganz gleich welchen Zug Schwarz ausführt, geben Sie den Befehl für Zugrücknahme ein und anschließend den für einen Computerzug. Je nachdem, welche Rolle ein Zufallsgenerator spielt, müssen Sie das Verfahren mehrfach durchführen, um das Programm dazu zu zwingen, einen neuen Gegenzug auszuführen. Dieses Verfahren funktioniert bei den meisten Programmen nur für den ersten Zug des Computers; die darauffolgenden Varianten können bei den meisten Programmen nicht zwangsweise entlockt werden.

Haben Sie bevorzugte Eröffnungsvarianten, können Sie auch so vorgehen - vorausgesetzt, Ihr Programm erlaubt die freie Figureneingabe:

Bauen Sie die Stellung auf, die sich nach einer bestimmten Anzahl von Zügen in Ihrer Lieblingseröffnung ergibt und zwar bis zu einer Stelle, wo der Computer am Zug wäre. Dann starten

Sie das Programm; antwortet es ohne größere Rechenzeit, können Sie davonausgehen, daß Ihre favorisierte Variante in der Eröffnungsbibliothek gespeichert ist.

Bei älteren Programmen gibt es eine simple Methode, das Programm zum Verlassen der Bibliothek zu zwingen: man spielt eine Variante so, daß erst durch Zugumstellung eine bekannte Position erreicht wird. Ein Beispiel aus der Sizilianischen Eröffnung:

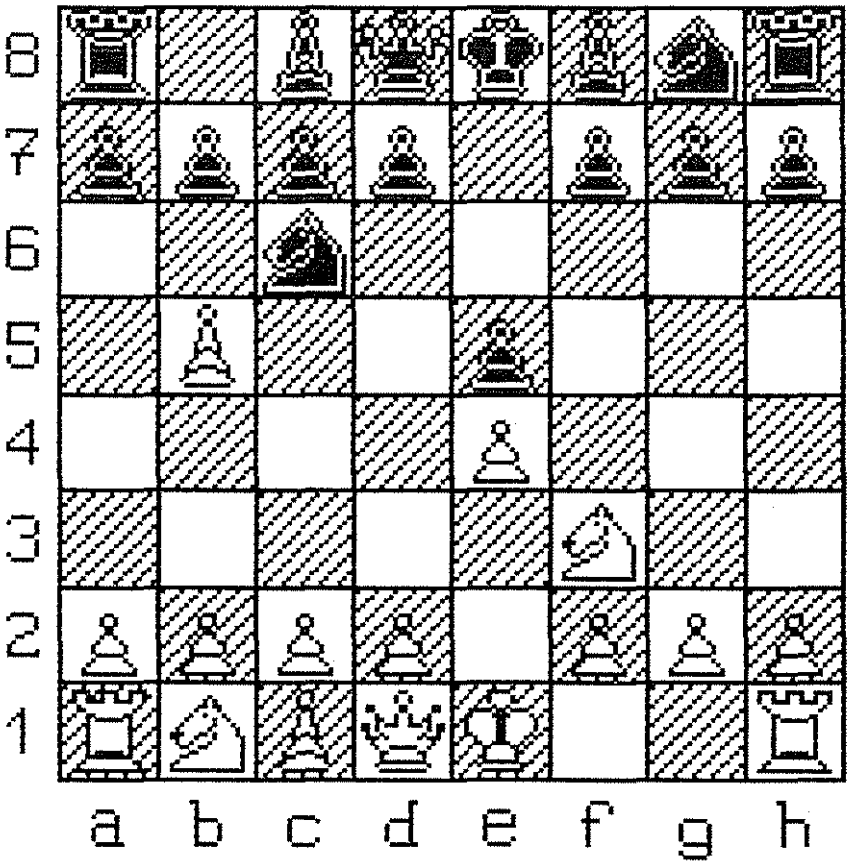
Najdorf-Variante: 1. e4 c5 2. Sf3 d6 3. d4 cd4: 4. Sd4: Sf6 5. Sc3 a6 6. Lg5 e6 7. f4 Le7 8. Df3 Dc7 9. 0-0-0 Sbd7 10. g4 b5 11. Lf6: Sf6: 12. g5 Sd7 13. a3 Tb8 14. h4 b4

Diese Variante ist in beinahe allen ernstzunehmenden Programmen enthalten. Sie ist eine der kompliziertesten in der Sizilianischen Verteidigung und führt zu verwickelten Mittelspielen - oft mit leichten Vorteilen für Schwarz. Schafft man es, das Programm zum Verlassen der Bibliothek zu zwingen, muß es schon in der Eröffnung rechnen, verbraucht dafür Zeit, die ihm später im Mittelspiel fehlt - die Gewinnchancen des Spielers steigen. Eine ganz einfache Zugumstellung bringt die meisten Programme aus der Eröffnungsbibliothek; man spielt statt der oben beschriebenen Zugfolge im 9. Zug schon g2-g4.

SENSORY 9 z.B. verläßt daraufhin die Bibliothek und spielt statt Sb8-d7 schon den Bauernvorstoß e6-e5 und verschlechtert seine Stellung.

Dieses Verhalten kann nicht verallgemeinert werden, in der Regel gilt jedoch, daß, wenn das Programm frühzeitig aus der Eröffnungsbibliothek gebracht wird, es mehr Zeit verbraucht und, falls der Spieler die Variante beherrscht, Schwierigkeiten haben wird.

Ob eine Stellungswiedererkennung in der Eröffnungsbibliothek eingebaut ist, erweist sich bei folgendem Test:
Bauen Sie eine Stellung auf, die sich nach wenigen Zügen einer häufig gespielten Variante ergibt; z.B.:



Lassen Sie das Programm nun rechnen; findet es auf Anhieb den Gegenzug aus der Eröffnungsbibliothek (4. ... Sf6), dürfen Sie sicher sein, daß eine Stellungserkennung eingebaut ist.

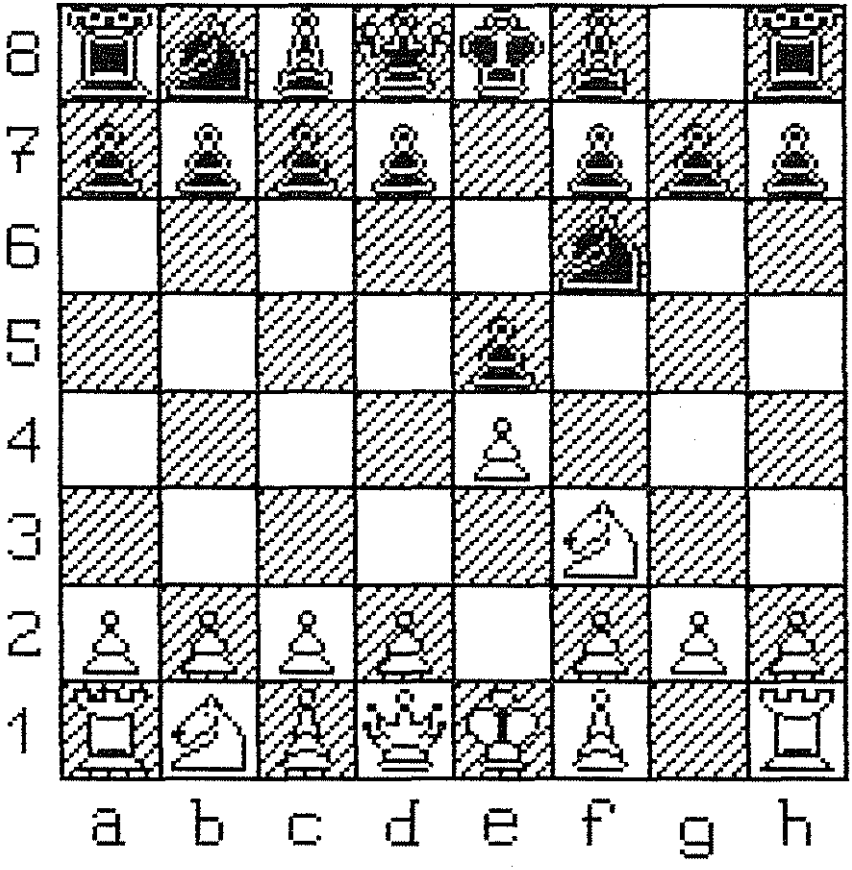
Übrigens können Sie mit der hier gezeigten Variante auch die Fähigkeit des Programmes erkunden, Zugumstellungen zu erkennen. Nach

1. Sg8-f6 Sb8-c6
2. e2-e4 e7-e5
3. Lf1-b5 ...

sollte das Programm mit den Zügen aus der Bibliothek fortsetzen.

Ganz findige Programme erkennen sogar Eröffnungsvarianten mit vertauschten Farben; leider sind solche Programme noch nicht in Schachcomputern oder für Mikrocomputer zu haben. Ein Test zeigt, daß der SENSORY 9 z.B. diese Fähigkeit nicht hat:

Bauen Sie folgende Anfangsstellung aus der Russischen Verteidigung auf:



Der Computer spielt Weiß und zieht - laut Bibliothek

3. Lf1-b5 ...

Sie ziehen folgerichtig mit Schwarz

3. ... Sb8-c6

Nun sollte der Computer bei vertauschten Farben auf den eigentlich 'schwarzen' Zug

3. Sb1-c3 ...

mit

3. ... Lf8-b4

antworten; das tut er aber nicht.
Diesen Test können Sie an allen Schachcomputern und bei allen Schachprogrammen nachvollziehen. Die Konsequenz dieser Einschränkung ist, daß Sie versuchen können, Eröffnungen so zu spielen, daß bewährte Varianten mit vertauschten Farben entstehen.

Spielen Sie doch einfach ganz exotische Eröffnungen! Wir haben z.B. kein Programm (auch kein spezielles Eröffnungsmodul für Schachcomputer!) gefunden, daß Entgegnungen auf Grob's Angriff 1. g2-g4 im Repertoire hatte.

5.1.4 Positionelle Stärken und Schwächen

Bei den Schachcomputern, die man vor ein paar Jahren kaufen konnte (etwa in den Jahren 1977 bis 1979) machten sich Schachspieler oft über die "Freßlust" der Programme lustig: wo ein Bäuerchen zu schlagen war, da schlugen diese Dinger zu - manchmal war der Bauer vergiftet und das Programm verlor. Seitdem hält sich hartnäckig das Gerücht, Schachprogramme wären außerstande, positionell zu spielen, geschweige denn für einen positionelle Vorteil Opfer zu bringen.

Was hat es nun mit dem Übergewicht materieller Faktoren bei der Zugauswahl auf sich. Julio Kaplan, Internationaler Schachmeister und Schachprogrammierer (u.a. für SciSys' TURBOSTAR) meint dazu (KAP):

> "Ein weiterer Punkt, an dem sich die Programme in zwei Gruppen einteilen lassen, ist, wenn sie Züge aufgrund positioneller Bewertung wählen. Alle Programme untersuchen auf jeder Ebene des Spielbaums immer die Materialbilanz. Es scheint auf der Hand zu liegen, zu deren Bewertung Punkte für positionelle Vorteile zu addieren und dann mit der Minimax-Prozedur eine Gesamtbewertung für einen Halbzug zu errechnen."

Dieser Ansatz beinhaltet jedoch ein Problem: er ist sehr zeitintensiv. Die Materialbilanz zu berechnen, ist sehr einfach - man muß nur die Figuren und ihre Werte zählen. Positionelle Terme sind viel komplexer und in der Berechnung zeitverbrauchend. Man muß einen ganzen Haufen von Eigenschaften betrachten, jede für sich ist so komplex wie die ganze Materialbewertung, deshalb ist es nicht ungewöhnlich, wenn ein Programm mindestens zehn- bis elfmal oder sogar mehr als fünfzigmal länger an der positionellen Bewertung rechnet als an der Materialbewertung.

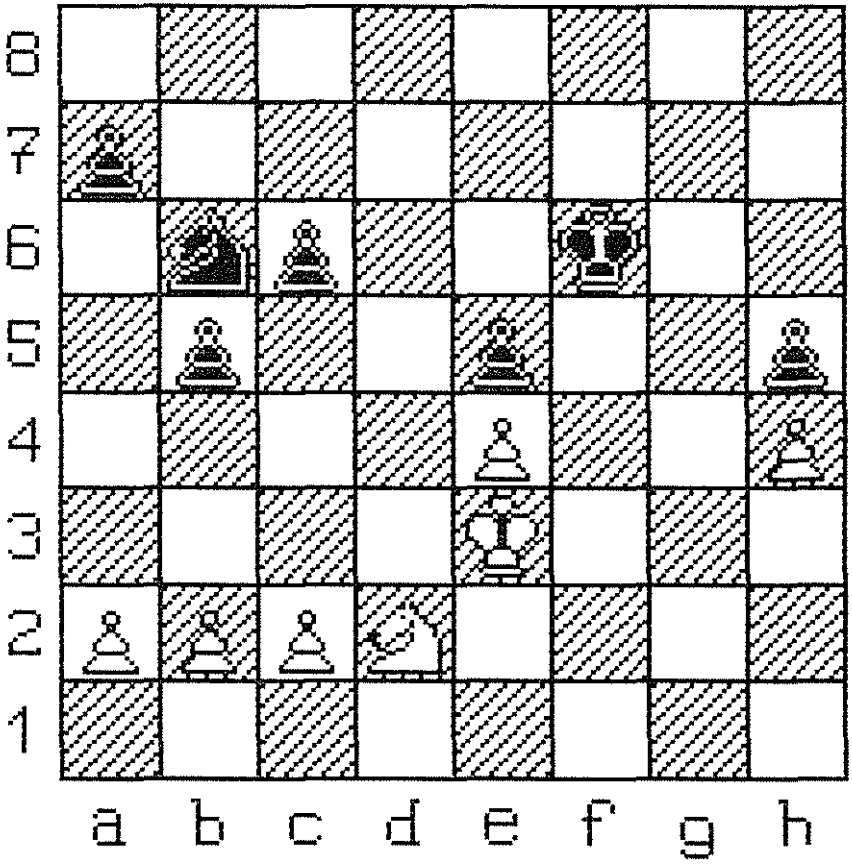
Schachprogrammierer haben deswegen folgende Alternative entwickelt: stellen Sie sich vor, nur die Halbzüge auf der ersten Ebene werden positionell bewertet und die folgenden rein materiell. Bei diesem Verfahren wird das Programm in die Lage versetzt, schlechte Züge aus positionellen Gründen zu streichen und so nur einen kleinen Teil der Rechenzeit für positionelle Bewertungen zu verbrauchen.

Diese Methode beschränkt jedoch das positionelle Verständnis des Computers ganz erheblich. Genau wie ein Programm mit festeingestellter Tiefe von drei Halbzügen für Ereignisse im vierten Halbzug völlig blind ist, so ist ein Programm, das

positionelle Faktoren nur im ersten Halbzug bewertet, für positionelle Veränderungen auf tieferen Ebenen ebenfalls völlig blind - selbst für sehr drastische, die im zweiten, dritten oder vierten Halbzug geschehen."

Julio Kaplan zeigt ein Beispiel, das Sie an Ihrem Programm - übrigens auch am DEMOSCHACH aus diesem Buch - ausprobieren können.

Stellen Sie folgende Position auf:



Schwarz am Zug sollte diese Partie leicht Remis halten. Gibt man diese Aufgabe jedoch einem Programm, das tatsächlich nur auf der ersten Ebene positionelle Berechnungen anstellt, darf man sich nicht wundern, wenn es

1. ... Sb6-c4+

spielt, was die schwarze Stellung völlig ruiniert und zum Verlust des scharzen C-Bauern führt (2. Sc4: bc: 3. Kd2 beliebig 4. Kc3 beliebig 5. Kc4:).

Anders z.B. SARGON (alle Versionen ab SARGON II auf allen Rechnern bei höheren Stufen):

1. ... Sb6-c4+

wird nicht einmal in Erwägung gezogen, lediglich Sa4 wird berechnet - gespielt wird

1. ... c6-c5

womit das Remis schon fast gesichert ist.

Wie 'denkt' aber ein Programm, das den ominösen Zug spielt? Kaplan beschreibt das so:

> "1. ... Sc4+ bringt den Springer auf ein starkes Feld, von wo aus er mehrere weiße Figuren angreift. In positioneller Hinsicht ist der Zug also sehr stark. Mal sehen, wie es materialmäßig aussieht."

Ab da rechnet das Programm den Spielbaum ausgehend von Sc4+ durch und findet den weißen Zug 2. Sc4:, die Antwort bc: und stellt fest, daß die Materialbilanz ausgeglichen ist. Auch wenn der Suchbaum auf sechs oder sieben Halbzüge ausgedehnt wird, bleibt dieses Ergebnis bestehen; erst im achten Halbzug würde das Programm den Verlust des C-Bauern 'sehen'. Aber so tief rechnen die kommerziellen Programme meist nicht.

Ein Programm, das auch im zweiten und dritten Halbzug positionelle Kriterien bewertet, verwirft den schlechten Springerzug schon dort, weil es den schwarzen Doppelbauern auf der c-Linie 'sieht'. Alle Programme in käuflichen Schachcomputern (mit wenigen Ausnahmen) und ein großer Teil der Schachprogramme für Mikrocomputer berechnen positionelle Terme auch auf tieferen Ebenen als Halbzug Nummer 1. Wie tief positionelle Kriterien mit bewertet werden, läßt sich schwer herausfinden - teilweise benutzen Programme diese Bewertung dynamisch, d.h. je tiefer im Spielbaum bewertet wird, desto weniger positionelle Terme gehen in die Gesamtbewertung ein.

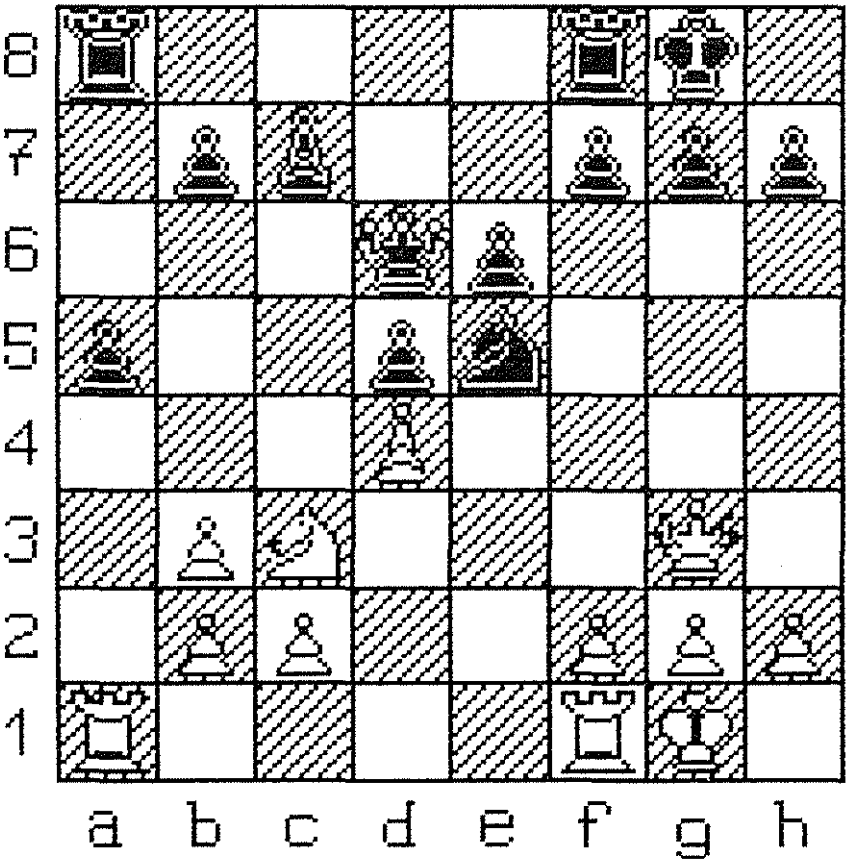
Die Erkenntnis, wie ein Programm positionelle Kriterien einbezieht, kann man nur schwer in Vorteile beim Spiel ausnutzen, zumal die meisten Programme sich ganz ähnlich verhalten - findet man jedoch heraus, daß das Programm wirklich nur in Tiefe 1 die Position bewertet, kann man versuchen, durch eigene positionelle Opfer (also Opferzüge, die materiellen Nachteil bringen, aber die Stellung verbessern) den Computer in Nachteil zu bringen. Gelegenheit dazu bietet sich im Mittelspiel, wenn die Figuren entwickelt sind, die Könige rochiert haben und der Kampf ums Zentrum entbrennt. Durch Figurenopfer auf Felder der Königsstellung kann man solche Programme in Verlegenheit bringen. Wir werden diesen Effekt später näher beleuchten.

Altertümliche Programme ohne Eröffnungsbibliothek und mit Schwächen in der positionellen Bewertung können nahezu nach Belieben geknackt werden: wählen Sie in der Eröffnung Gambit-Varianten und spielen Sie konsequent gegen die gegnerische Königsstellung - das wirkt immer!

5.1.5 Rechentiefe

Wie tief rechnet mein Programm? Diese Frage stellt sich jeder engagierte Computerschach-Spieler. Die Anzahl der berechneten Halbzüge pro Stellung können Sie mit 90%iger Sicherheit durch ein paar Tests feststellen.

Um wirklich brauchbare Ergebnisse zu erhalten, sollten Sie die Teststellungen mit allen verfügbaren Spielstufen (außer einer eventuell vorhandenen Problem- bzw. Analysestufe) durchführen. Und hier eine der zahlreichen denkbaren Teststellungen:



Die verwickelten Möglichkeiten sind deutlich; Weiß am Zug muß allerhand Fesselungen und Drohungen gegen Schwarz

berechnen und darf auch die Drohungen von Schwarz nicht übersehen.

Im einzelnen hat Weiß folgende Drohungen gegen Schwarz:

- Df3, gedeckt durch Ld4, droht Matt auf G7;
- Sb5 bedroht Dd6 und Lc7;
- Le5: führt zu vorteilhaftem Abtausch für Weiß.

Ein Programm, das zwei Halbzüge tief rechnet, wird folgende Überlegung anstellen:

"Also, etwas Aufregendes kann ich nicht entdecken. Wie sieht's denn mit dem Abtausch auf E5 aus?

1. Ld4xe5 Dd6xe5

Na ja, dann tauschen beide Seiten erst ihre gut postierten Leichtfiguren und die Bewertung bleibt konstant - nicht besonders verlockend.

1. Dg3xe5 ...

ist ja wohl ganz verkehrt, denn dann folgt

1. ... Dd6xe5

und ich habe die Dame gegen einen Läufer getauscht."

Wohlgemerkt, diese Berechnungen spielen sich ab, wenn das Programm nur zwei Halbzüge tief blickt - also den eigenen und den Gegenzug betrachtet.

In Wirklichkeit sind beide Züge, Le5: und De5: gleichwertig, weil der weitere Tausch zum materiellen Ausgleich führt. Programme, die weiter rechnen, sehen diesen Ablauf. Weil aber jeder Halbzug mehr, den das Programm berechnen soll, exponential mehr Rechenzeit erfordert, wenden die meisten Programmierer folgendes Rezept an:

Es wird eine Rechentiefe von z.B. zwei Halbzügen fest vorgegeben. Das Programm betrachtet alle Züge und widmet sich anschließend nur noch den Schlagzügen. Diese werden solange verfolgt, wie weitere Schlagzüge folgen. Züge, die kein weiteres Schlagen nach sich ziehen, nennt man ruhige Züge (siehe DEMOSCHACH). In der Ruhesuche - also der Betrachtung der ruhigen Züge - wird dann positionell bewertet. In unserem Test wird Weiß feststellen, daß der Abtausch auf e5 nichts bringt, egal welche Figur mit dem Tausch beginnt.

Einer der möglichen Züge von Weiß ist

1. Sc3-b5

Bei einer Suche über zwei Halbzüge wird das Programm diesen Zug verwerfen, weil es die Fortsetzung

1. ... Dd6-c6

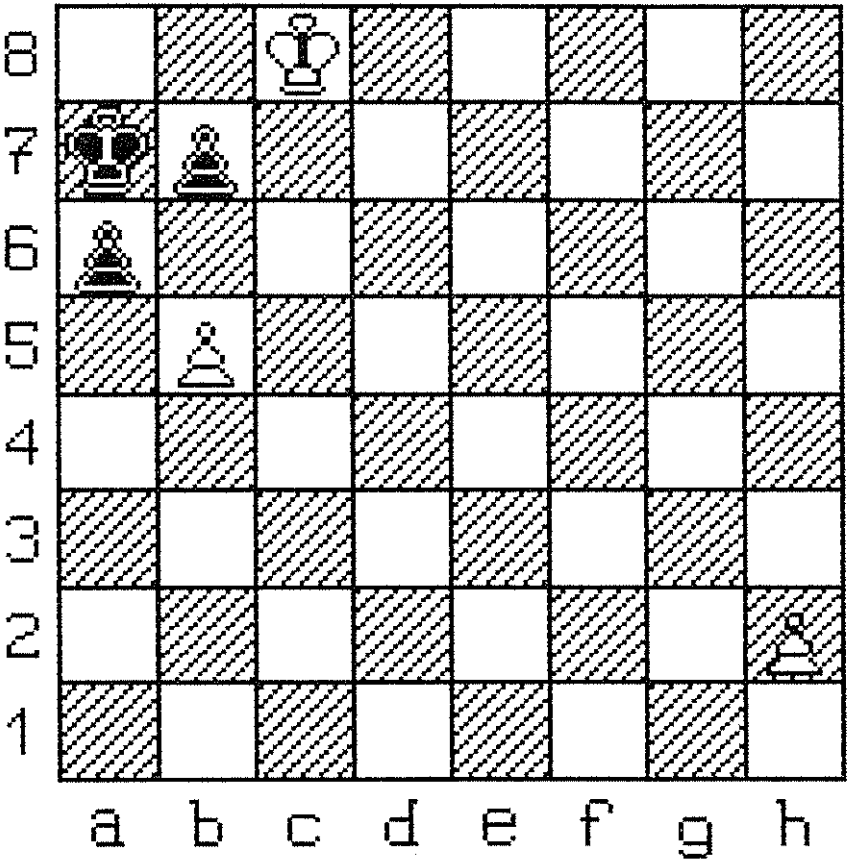
sieht und für uninteressant hält (kein Vorteil, weder für Weiß, noch für Schwarz).

In Wirklichkeit ist dieser Springerzug der Beginn eines Gewinnweges:

1. Sc3-b5 Dd6-c6
2. Sb5xc7 Dc6xc7
3. Ld4xe5 ...

und Weiß hat eine Figur gewonnen!

Eine extreme Teststellung, mit der Sie probieren können, wie tief Ihr Programm maximal rechnen kann, fanden wir in der Zeitschrift COMPUTERSCHACH INTERNATIONAL (Ausgabe März 1983):



Na, was würden Sie ziehen? Doch wohl nicht

1. b5xa6 ...

denn damit verliert Weiß:

1. b5xa6 b7-b5
2. h2-h4 b5-b4
3. h4-h5 b4-b3
4. h5-h6 b3-b2
5. h6-h7 b2-b1D
6. h7-h8D Db1-b8+
7. Kc8-d7 Db8xh8

Weiß kann aber auch gewinnen. Lassen Sie Ihr Schachprogramm diese Nuß knacken - Sie müssen ihm allerdings reichlich Rechenzeit geben, stellen Sie also die höchste Spielstufe ein. TURBOSTAR 432 (von SciSys) findet die Lösung auch auf niedrigeren Stufen:

1. b5-b6+ Ka7xb6
2. h2-h4 a6-a5
3. h4-h5 a5-a4
4. h5-h6 a4-a3
5. h6-h7 a3-a2
6. h7-h8D a2-a1D
7. Dh8xa1

Gute Programme verschwenden eigentlich nur ganz wenige Augenblicke an der Betrachtung des Zuges a6:; eigentlich setzen sie sich nur mit b6+ und h4, wobei der letztere Zug nach längerer Analyse verworfen wird.

Sie können sich leicht ausrechnen, daß bei der Betrachtung von mehr als 8 Halbzügen der Patzer-Zug a6: nicht mehr berechnet wird und nach 10 Halbzügen auch h4 aus dem Rennen ist.

Wenn Ihr Programm auf höchster Spielstufe (also nicht im Analyse- oder Problemmodus!) den Gewinnweg findet, wissen

Sie sicher, daß es bei geringem Material auf dem Brett 14 Halbzüge tief rechnen kann.

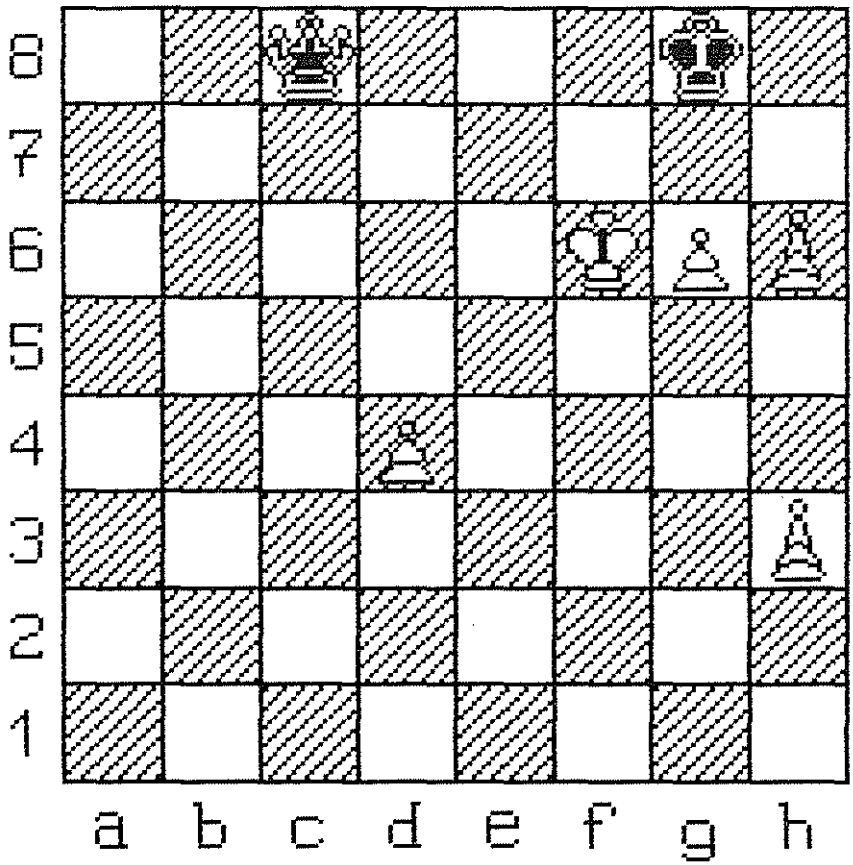
Probieren Sie das gleiche Problem auf der nächstniedrigen Stufe, wenn immer noch b6+ vorgeschlagen wird, rechnet das Programm auch hier 14 Halbzüge tief - gehen Sie alle einstellbaren Spielstufen abwärts durch, wenn sich das Programm für h4 entscheidet, wissen Sie, daß es auf dieser Spielstufe mit dem vorhandenen Material 10 Halbzüge blickt usw.

Spielt Ihr Programm unabhängig von der Spielstufe immer a6:, dann schaut Ihr Programm nie tiefer als 4 bis 6 Halbzüge, egal, wieviel Figuren noch auf dem Brett sind - eine eklatante Endspielschwäche dürfte dann die hervorragendste Eigenschaft des Programmes sein.

5.1.6 Permanent Brain

Diese Eigenschaft (zu deutsch: ständiges Gehirn) bewirkt, daß das Programm weiter rechnet, während Sie an Ihrem Zug arbeiten. Bevor Sie nun entrüstend aufspringen und mit dem Satz "Wie unfair" das Schachprogramm beschimpfen, lassen Sie sich gesagt sein, daß dieses Element ganz dem menschlichen Spielverhalten entspricht - wenn Sie gegen einen Menschen spielen, hören Sie ja auch nicht auf nachzudenken, wenn Ihr Gegener der Reihe ist. Unangenehm ist das Permanent Brain nur dann, wenn Sie gar nicht wissen, daß Ihr Programm damit arbeitet.

Manche Programme in Schachcomputern haben ein Permanent Brain, in den Bedienungsanleitungen ist dieses Merkmal jedoch nicht erwähnt. Wenn Ihr Schachcomputer in der Lage ist, ein Matt anzukündigen, können Sie leicht herausfinden, ob er mit Permanent Brain arbeitet:



Der Computer spielt Weiß und ist am Zug. Wählen Sie die niedrigste Spielstufe - Ihr Programm findet den Anfangszug

1. Lh3xc8

auf jeden Fall (falls nicht: ab in den Mülleimer mit dem Computer!). So, jetzt sind Sie am Zug. Lassen Sie sich Zeit, eine halbe Stunde, eine ganze Stunde - egal. Hat Ihr Computer

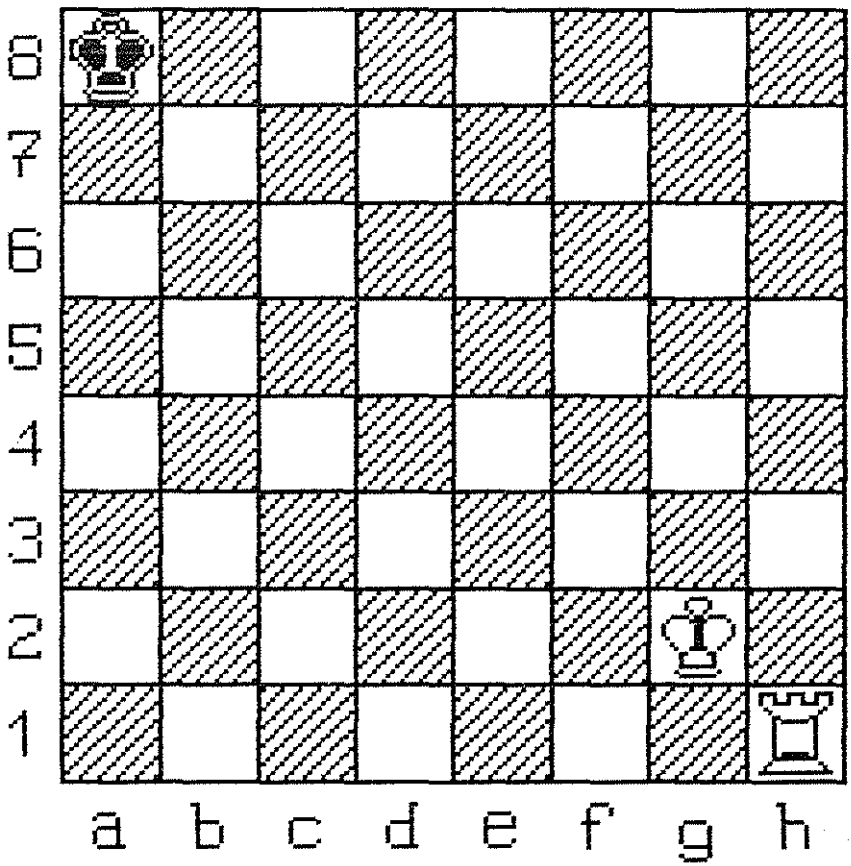
nämlich ein Permanent Brain, wird er in der Zwischenzeit weiterrechnen und alle Ihre Entgegnungen durchchecken; dabei findet er heraus, daß er Sie in spätestens 3 Zügen mattsetzen wird und zeigt dies an.

Antworten Sie sehr schnell, wird das Programm entweder sehr lange brauchen, bis es Matt in 3 ankündigt oder dies gar nicht tun.

5.1.7 König-Turm-Endspiele

Endspiele sind nun mal die Schwäche von Schachprogrammen, damit muß man sich für die nächsten Jahre wohl abfinden. Was man aber als Käufer eines Schachprogrammes oder eines Schachcomputers verlangen kann, ist, daß die grundlegenden Endspiele vom Computer siegreich beendet werden können.

Das einfachste aller Endspiele ist das König-Turm-Endspiel gegen den einsamen gegnerischen König.



Weiß ist am Zug. Geben Sie diese Stellung Ihrem Computer als Aufgabe. Beginnen Sie mit der niedrigsten Spielstufe und steigern Sie langsam. Je nachdem, welche Rechenzeit bzw. -tiefe das Programm in der niedrigsten Stufe zur Verfügung hat, können folgende kuriosen Spiele zustande kommen:

1. Kf3 Kb7 2. Ke4 Kc6 3. Th5 Kd6 4. Te5 Kc7 5. Kd5 Kd7 6. Te6 Kc7 7. Te7 Kc8 8. Kd6 Kd8

So weit, so gut - jetzt wird's komisch:

9. Ke6 Kc8 10. Kf7 Kd8 11. Kf8 Kc8 12. Ke8 Kb8
13. Td7 Kc8 14. Ke7 Kb8 15. Kd8 Ka8 16. Tc7 Kb8
17. Kd7 Ka8

Halt! Keinen Schritt weiter, sonst wird es ein Patt! Also muß der Turm wieder in Bewegung gesetzt werden:

18. Tc6 Kb8 19. Tc7 Ka8 und so weiter, und so weiter ...

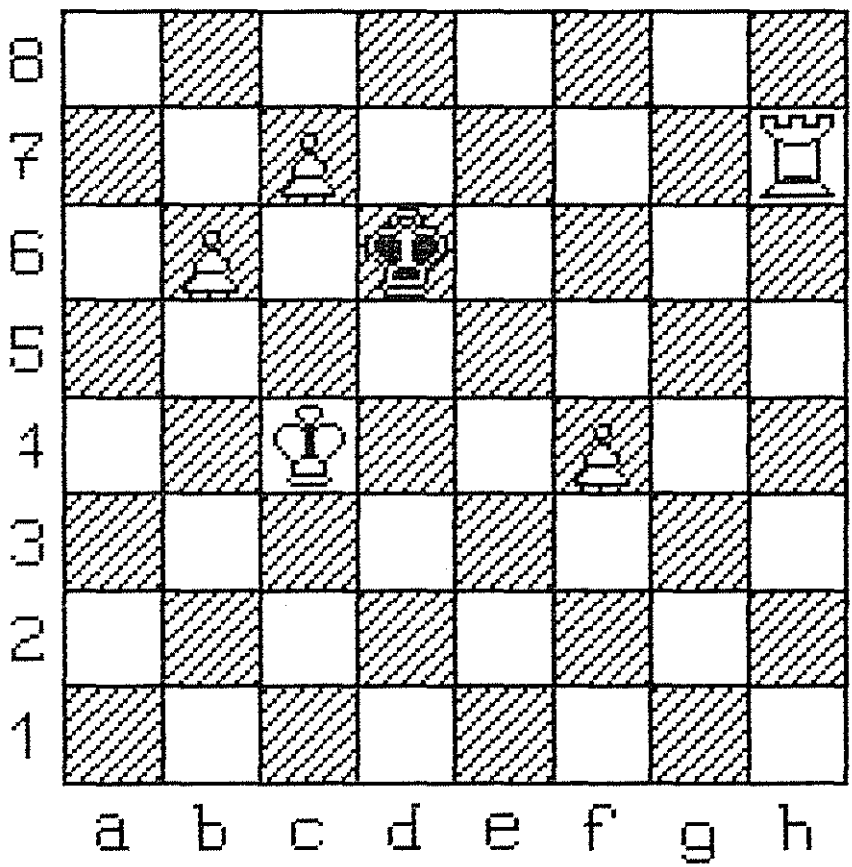
Wenn Ihr Programm so spielt, hat es entweder einen schweren Fehler oder die Suchtiefe reicht nicht aus - im Beispiel wird deutlich, daß der Computer nie tiefer als 4 Halbzüge blickt, sonst wären die Züge ab der Nummer 9 nicht passiert.

Natürlich ist es wichtig, die Endspielstärken und -schwächen eines Programmes gut zu kennen, sie herauszufinden ist allerdings sehr zeitaufwendig. Am besten arbeiten Sie mit einem der Standardwerke zu dem Thema oder einem guten Schachlehrbuch (z.B. Siegbert Tarrasch: Das Schachspiel, rororo Sachbuch 6816). Die Aufgaben, die eigentlich für den menschlichen Schachspieler gedacht sind, legen Sie Ihrem Programm vor und erfassen Sie tabellarisch die Häufigkeit, mit der das Programm pro Spielstufe die richtige Lösung findet.

Bei einigen Schachcomputer werden Sie ins Staunen kommen; wenn Sie nämlich nachrechnen, werden Sie finden, daß sie im Endspiel teilweise bis zu 20 Halbzüge tief rechnen können.

5.1.8 Unterverwandlungen

Als erfahrener Schachspieler wissen Sie sicher, daß es Spiele gibt, die man nur gewinnen kann, wenn man einen Bauer auf der achten Reihe nicht zur Dame, sondern zu einer anderen Figur werden läßt. Ob Ihr Programm das auch weiß, können Sie mit folgender Stellung testen:



Weiß ist am Zug.

1. c7-c8D

ergibt ein Patt. Nur die Verwandlung des Bauern auf C7 in einen Läufer führt zum Sieg.

1. c7-c8L Kd6-c6
2. Th7-h6++

Ihr Programm findet - wenn es die Unterverwandlung beherrscht - die Folge unabhängig von der Spielstufe, da die Pointe spätestens im dritten Halbzug folgt.

5.2 Schlagen Sie Ihr Schachprogramm! - 7 goldene Regeln zum Gewinnen

Jetzt haben Sie Ihr Schachprogramm schon ein wenig kennengelernt und allein diese Kenntnis wird Ihnen sicher zu manchem glänzenden Sieg verhelfen. Vielleicht wollen Sie aber auch Ihnen unbekannte Programme schlagen. Es kann ja sein, daß Sie als regelmäßiger Schachspieler auch an kleineren Turnieren teilnehmen und da kann es Ihnen heute passieren, daß Sie gegen ein Computerprogramm antreten müssen (z.B. bei den sogenannten 'offenen' Turnieren, bei denen Spieler aller Klassen und auch Schachcomputer zugelassen sind). Möglicherweise sind Sie auch aus niederen Beweggründen daran interessiert, den Computer nach Strich und Faden zu besiegen; vielleicht, weil Sie so Ihren Frust abreagieren können, vielleicht, weil Sie mit schnellen Siegen Ihren Freunden und Bekannten imponieren wollen - sei's drum: Der Computer wird es Ihnen nicht übel nehmen und Sie haben Ihren Spaß.

Wer einen Gegner besiegen will, muß dessen Stärken und Schwächen analysieren. Das haben wir zum Teil erledigt. Fassen wir zusammen:

+) Das Programm spart Zeit, indem es in der Eröffnung Züge aus der Bibliothek spielt.

+) Das Programm spielt in taktischen Situationen mit langen Abtauschvarianten fast immer 100%ig korrekt.

-) Das Programm bewertet materielle Vorteile in 99% der Fälle höher als einen positionellen Vorteil.

-) Das Programm ist nicht in der Lage, Pläne auszurechnen und in die Tat umzusetzen.

-) Das Programm ist in ruhigen Stellungen nicht fähig, über einen gewissen Horizont hinauszublicken.

-) Das Programm findet im Endspiel Gewinnwege nur unter großen Schwierigkeiten.

Aus diesen Tatsachen haben wir 7 goldene Regeln für das erfolgreiche Spiel gegen Schachprogramme entwickelt:

1. Spielen Sie ungewöhnliche Eröffnungen.

Je früher das Programm die Bibliothek verlassen muß, desto mehr muß es rechnen. Je mehr es in der Anfangsphase rechnen muß, desto weniger Zeit hat es später. Je weniger Zeit es im Mittelspiel hat, desto weniger tief kann es suchen.

2. Spielen Sie in der Eröffnung Gambit.

In Gambit-Varianten opfern Sie einen (oder mehrere) Bauern für positionelle Vorteile wie raschere Entwicklung oder Beherr-

schung des Zentrums. Sofern das Programm Ihr Gambit nicht in der Bibliothek gespeichert hat, wird es das Opfer annehmen und so in der Regel seine Stellung schwächen. In ganz schweren Fällen wird das Programm um jeden Preis versuchen, den Mehrbauern zu verteidigen - der Ruin der Stellung ist die Folge.

3. Vermeiden Sie im Mittelspiel taktische Verwicklungen.

Das Programm berechnet lange Abtäusche sowieso besser. Empfehlenswerter ist es, im Mittelspiel sehr ruhig aufzubauen und langfristige Pläne zu schmieden.

4. Provozieren Sie in verwickelten Situationen Abtäusche.

Ganz im Gegensatz zu dem, was man beim Spiel gegen menschliche Gegner empfiehlt, sollten Sie gegen den Computer im versuchen, klare Stellungen zu schaffen. Dann kann das Programm seinen Rechenvorteil nicht ausspielen.

5. Stellen Sie dem Programm Fallen.

Natürlich ist es Unsinn, dem Programm mit irgendwelchen Tricks beikommen zu wollen. Die hier gemeinten Fallen bestehen darin, langsam Positionen zu erarbeiten, in denen das Programm ratlos wird und schlechte Züge macht.

6. Streben Sie schon bei kleinen Vorteilen das Endspiel an.

Schachprogramme spielen im Endspiel relativ schwach. Wenn Sie das Spiel mit wenigen Figuren korrekt durchführen, hat ein Schachprogramm kaum eine Chance.

7. Spielen Sie das Endspiel konsequent nach Plan.

Was sich nach einer Binsenweisheit anhört, gewinnt beim Spiel gegen Computer besondere Bedeutung. Gerade im Endspiel sollten Sie nicht nach dem Motto "Kleinvieh macht auch Mist" die Stellung einfach nur verbessern, Sie sollten sich vielmehr einen Plan ausdenken und konsequent durchführen.

Wie Sie diese Regeln in der Praxis anwenden können, zeigen Ihnen die nächsten 7 Kapitel.

5.2.1 1.Regel: Ungewöhnliche Eröffnungen

Wir bieten Ihnen zwei Varianten (je eine mit Schwarz bzw. Weiß spielbar), die folgende Bedingungen erfüllen:

- Wir haben Sie in keiner Eröffnungsbibliothek gängiger Schachprogramm bzw. -computer gefunden.
- Die Varianten sind nicht widerlegt. D.h., daß derjenige, der sie anwendet nicht automatisch in eine Verluststellung gerät.

a) Grob's Angriff

Der Schweizer Fernschachmeister und Schachtheoretiker Henry Grob hat diese merkwürdig anmutende Eröffnung (wieder)entdeckt. Weiß opfert hier nicht unbedingt einen Bauern, aber eine Menge Sicherheit für einen scharfen Angriff auf den schwarzen Königsflügel. Es gibt zwar zahlreiche

Varianten, Spiele gegen Schachprogramme mit dieser Eröffnung (die ebenfalls in keiner Bibliothek zu finden ist) ergaben immer folgenden Ablauf:

01	g2-g4	d7-d5
02	Lf1-g2	Lc8xg4
03	c2-c4	c7-c6

Für diesen letzten Zug verbrauchen alle Programme extrem viel Zeit - Programme, die anzeigen, wieviele Halbzüge sie betrachten, rechnen diesen Zug mit Maximaltiefe und finden derart viele Schlagvarianten, daß sie bis zu sieben Halbzüge tiefer rechnen müssen.

Wir wollen hier keine vollständige Partie zeigen, sondern nur erwähnen, daß Weiß meistens sehr gutes Angriffsspiel erreicht, wobei die auf b3 postierte Dame lange Zeit in Richtung f7 droht und bei manchen Programmen Schwarz zum Verzicht auf die Rochade zwingt.

Besorgen Sie sich Literatur zu dieser Eröffnung und probieren Sie die Varianten aus (GRO).

b) Altindisch

Diese Eröffnung ist bei einigen Programmen in der Bibliothek gespeichert, jedoch meistens nur bis zum 3. Zug von Schwarz. Sie können also als Führer der schwarzen Steine das Programm aus den festen Bahnen bringen.

Altindisch ist nicht besonders spektakulär, erlaubt aber dem Spieler so zu spielen, wie es unseren goldenen Regeln 1, 3 und 4 vorschreiben. Schauen wir uns eine typische Partie an.

Gespielt am 17. November 1985
Weiß: SciSys TURBOSTAR 432
Schwarz: Spieler

01	d2-d4	Sg8-f6	06	Lc1-g5	c7-c6
02	c2-c4	d7-d6	07	Lf1-e2	Dd8-a5
03	Sb1-c3	Sb8-d7	08	0-0	0-0
04	e2-e4	e7-e5	09	Ta1-c1	h7-h6
05	Sg1-f3	Lf8-e7			

Hier merkt man schon, daß es keine dramatischen Abtauschserien gibt und das Spiel in ruhigen Bahnen abläuft. Der Spieler hat den Vorteil, kleine und feine positionelle Änderungen zu erreichen, die das Programm gar nicht als Bedrohung sehen kann.

10	Lg5-d2	Da5-c7	13	Tf1-e1	Lc8-g4
11	Le2-d3	Sd7-b8	14	Ld2-e3	Sb8-d7
12	d4xe5	d6xe5	15	Ld3-c2	Ta8-d8

Jetzt reißt ihm der Geduldsfaden:

16	Le3xa7	b7-b6
17	Lc2-a4	Sd7-c5
18	La7xb6	Dc7xb6
19	Dd1-c2	...

Und Weiß gibt den Läufer für zwei Bauern.

19	...	Lg4xf3

Abtausch ist angesagt!

20	g2xf3	Sc5-d3
21	Sc3-d5	c6xd5
22	Dc2xd3	Sf6-h5

Der weiße Königsflügel ist offen, die weißen Untertanen treiben sich auf dem anderen Flügel herum und der Doppelbauer auf der F-Linie behindert die Verteidigung.

23	Dd3-b3	Db6-g6+
24	Kg1-h1	Sh5-f4
25	Te1-g1	Dg6-h5
26	c4xd5	Td8-b8
27	Db3-c3	Tb8xb2

Die Dame darf den Turm nicht nehmen: 28. Db2: Df4:+ 29. Tg2 Dg2:++

28	d5-d6	Le7xd6
29	Tc1-c2	Tf8-b8
30	Tc2xb2	Tb8xb2
31	La4-b3	Tb2xf2
32	Dc3-c8+	...

Racheschach? Dazu ist ein Computer nicht in der Lage. Hier sieht man die Auswirkung des Horizonteffektes: Mit dem Schach schiebt das Programm den unvermeidlichen Partieverlust um ein paar Züge weiter, so daß er innerhalb der Suchtiefe nicht mehr auftaucht.

32	...	Ld6-f8	38	Df2-c2	Kg7-h7
33	Lb3xf7+	Dh5xf7	39	a2-a4	Db7-b4
34	Dc8-b8	Df7-h5	40	Dc2-d1	Db4-b2
35	Tg1xg7+	Kg8xg7	41	Dd1-g1	Db2-e2
36	Db8-a7+	Dh5-f7	42	Dg1-a7+	Lf8-g7
37	Da7xf2	Df7-b7	43	Da7xg7+	Kh7xg7

Die schwarze Dame begeht Selbstmord und damit ist die Partie auch gelaufen:

44	Kh1-g1	De2-g2++

Mit solchen Partien kann natürlich nicht bewiesen werden, daß man als Spieler mit der Altindischen Verteidigung immer gewinnt; es zeigt sich allerdings, daß ein frühzeitiges Verlassen der Eröffnungsbibliothek das Programm zu längeren Suchzeiten veranlaßt und insgesamt die Spielstärke vermindern.

5.2.2 2.Regel: Gambits

Der spanische Mönch Ruy Lopez (siehe Kapitel 1.1) benutzte diesen Begriff als erster für Eröffnungen, in denen ein (oder mehrere) Bauer geopfert wird. Eigentlich heißt im Italienischen "dare il gambetto" soviel wie "jemandem ein Bein stellen" und wird heutzutage in der Boxersprache verwendet. Im übertragenen Sinne trifft dieses Beinstellen ja auch auf Eröffnungen zu, bei denen man Material für positionelle Vorteile gibt.

Schachprogramme können meistens mit einem Gambit nicht umgehen - es sei denn, sie haben entsprechende Eröffnungen gespeichert. Ansonsten nehmen sie das Opfer gewöhnlich an und verteidigen das gewonnene Material auf Kosten der Stellung.

Beim folgenden Gambit ist dies sogar fast zwangsläufig der Fall.

a) Blackmar-Diemer-Gambit

Die Kenner werden aufstöhnen. Das BDG ist eine Variante, die im Turnierschach so gut wie nie auftaucht, von einer kleinen Gemeinde überzeugter Anhänger aber gepflegt wird.

01	d2-d4	d7-d5
02	e2-e4	...

Das sind die Anfangszüge dieses riskanten Gambits, das sich auf geradezu ketzerische Weise den üblichen Eröffnungsdogmen widersetzt. Es folgt üblicherweise

02	...	d5xe4
03	Sb1-c3	...

Die Verbesserung des Herrn Diemer zum widerlegten Gambit des Herrn Blackmar. Alle Programme, mit denen wir diese Eröffnung getestet haben, verließen nach dem zweiten Zug von Weiß die Bibliothek und begannen zu rechnen. Wir wollen an

einer Partie demonstrieren, daß der Spieler mit dieser exotischen Eröffnung dem Computer Paroli bieten kann.

Gespielt am 10. November 1985
Weiß: Spieler
Schwarz: CC Sensory 9 (Stufe 3)

03	...	Sb8-c6
04	Lf1-b5	Sg8-f6
05	f2-f3	...

Dieser Zug macht den Computer stutzig. Er muß schlagen, sonst hat Weiß auf einmal das Zentrum in Besitz.

05	...	e4xf3
06	Sg1xf3	Lc8-d7
07	0-0	Sf6-g4

Was will er? Anstatt seine Figuren zu entwickeln, macht er sinnlose Vorstöße. Eine Zwischenbilanz zeigt, daß Weiß wegen der geöffneten E- und F-Linie besser, weil aktiver steht. Und genau das ist der Sinn eines Gambits. Also ruhig weiter.

08	Lc1-f4	e7-e6
09	Sf3-g5	Sg4-f6

Siehste, war wohl nichts mit dem Springer. Weiß hat jetzt einen Plan: Massiver Angriff auf den schwarzen König über das Feld F7 mit Turm, Dame und Springer.

10	Dd1-f3	Lf8-d6
11	Lf4xd6	c7xd6
12	d4-d5	Dd8-b6+
13	Kg1-h1	...

Die Dame will doch wohl nicht den Läufer rauben? Auf 13. ... ed: 14. Sd5: Db5: folgt mit 15. Sc7+ der Dameverlust.

13	...	e6xd5
14	Sc3xd5	Sf6xd5

Das ist mindestens genauso schlimm!

15	Df3xf7	Ke8-d8
16	Df7xd5	Sc6-e5
17	Lb5xd7	Kd8xd7
18	Sg5-f7	Th8-e8
19	Ta1-e1	...

Das Mattnetz zieht sich zusammen; Weiß kontrolliert indirekt die 7. Reihe und Schwarz hat keine Gegendrohungen.

19	...	Kd7-c7
20	Sf7xe5	d6xe5
21	Tf1-f7+	Kc7-b8

Schwarz spielt jetzt praktisch mit einem Turm weniger.

22	b2-b3	...

Das ist zu schlapp! Hier verliert Weiß ein wichtiges Tempo, das es Schwarz erlaubt sich wieder etwas Luft zu verschaffen. Besser wäre Te5: - nach 23. De5:+ Kc8 folgt Matt in 4 Zügen: z.B. 24. Df5+ Kb8 (24. ... Kd8 25. Dd7++) 25. Dd7 Dc7 26. Tf8+ Dc8 27. Dc8:++.

22	...	e5-e4
23	Te1xe4	Te8xe4
24	Dd5xe4	a7-a6

Das ist die Folge des schwachen Zuges b3! Schwarz hat jetzt ein Luftloch auf A7. Weiß muß sich vorerst damit begnügen, die schwarzen Bauern abzuräumen.

24	Tf7xg7	h7-h6
25	Tg7-g8+	Kb8-a7
26	Tg8xa8	Ka7xa8

Weiß hat zwei Bauern mehr! Nun sollte irgendwie der Damentausch vollbracht werden.

27	h2-h3	Db6-a5
28	a2-a4	Da5-c3
29	De4-d3	Dc3-e1+
30	Kh1-h2	De1-b1
31	a4-a5	h6-h5
32	Dd3-c4	Ka8-b8
33	b3-b4	Db1-b2
34	Dc4-b3	Db2xb3

Er macht's. Der Doppelbauer auf der B-Linie scheint es wert zu sein. Doch nun kommen die Mehrbauern zu Geltung.

35	c2xb3	h5-h4	45	Ke6-d6	Kg5-f4
36	g2-g3	h4xg3	46	Kd6-c7	Kf4-e4
37	Kh2xg3	Kb8-c7	47	Kc7xb7	Ke4-d5
38	h3-h4	Kc7-d6	48	Kb7xa6	Kd5-c6
39	Kg3-f4	Kd6-e6	49	b4-b5+	Kc6-c7
40	Kf4-g5	Ke6-f7	50	Ka6-a7	Kc7-d6
41	h4-h5	Kf7-g7	51	b5-b6	Kd6-c6
42	h5-h6+	Kg7-h7	52	b6-b7	Kc6-e5
43	Kg5-f6	Kh7xh6	53	b7-b8D	
44	Kf6-e6	Kh6-g5			

Die letzten 19 Züge sind dann Routine.

Man sieht, auch mit einem angeblich unsinnigen Gambit kann man gewinnen.

b) Lettisches Gambit

Nicht ganz so selten wird das Lettische Gambit (auch Gambit in der Rückhand genannt) gespielt. Es ist eine Art Königsgambit mit vertauschten Farben, mit leider oft katastrophalem Ausgang für Schwarz. Allerdings treten solche Dramen eigentlich nur auf, wenn zwei Menschen gegeneinander spielen, Schachprogramme

tun sich schwerer und so führt das Lettische Gambit gegen einen Computer gespielt häufig zum Remis.

Verschiedene Programme bzw. Schachcomputer haben Lettisch im Repertoire, jedoch meistens nur eine ganz bestimmte Variante, nämlich:

01	e2-e4	e7-e5
02	Sg1-f3	f7-f5
03	Lf1-c4	f5xe4
04	Sf3xe5	...

Diese Fortsetzung gilt es also zu vermeiden - sie ist sowieso nicht besonders ersprießlich für Schwarz. Besser geht es so:

03	...	Sg8-f6
04	e4xf5	d7-d5

Danach verlassen die Programme, die Lettisch kennen, ihre Bibliothek.

05	Lc4-b5+	c7-c6

Verstellt zwar dem Springer das Feld, aber der ist noch nicht dran.

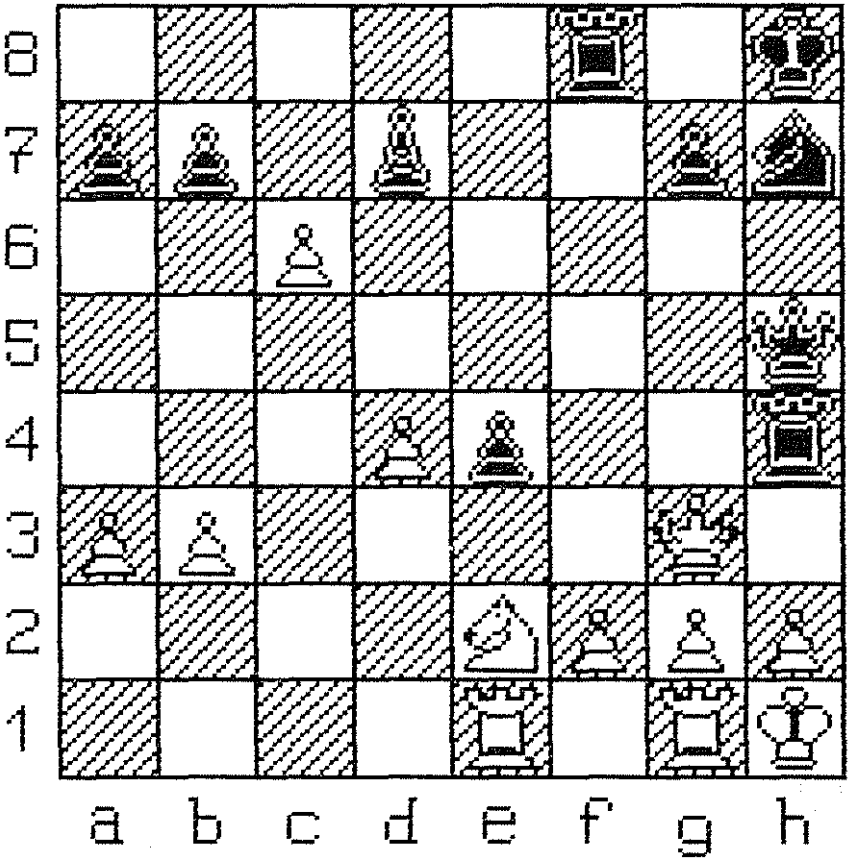
06	Lb5-e2	Lf8-d6
07	d2-d4	e5-e4
08	Sf3-h4	0-0
09	Lc1-g5	Sb8-d7
10	0-0	Dd8-e7

Es geht immer um den weißen Bauern auf F5; wenn er fällt, hat Schwarz starken Angriff.

11	Sb1-c3	De7-f7
12	Ta1-c1	Tf8-e8
13	Tf1-e1	Sd7-f8
14	Kg1-h1	...

Diese Ratlosigkeit zeigen in dieser Stellung fast alle Programme.

Nach weiteren zehn bis zwölf Zügen entstehen häufig Stellungen wie diese:

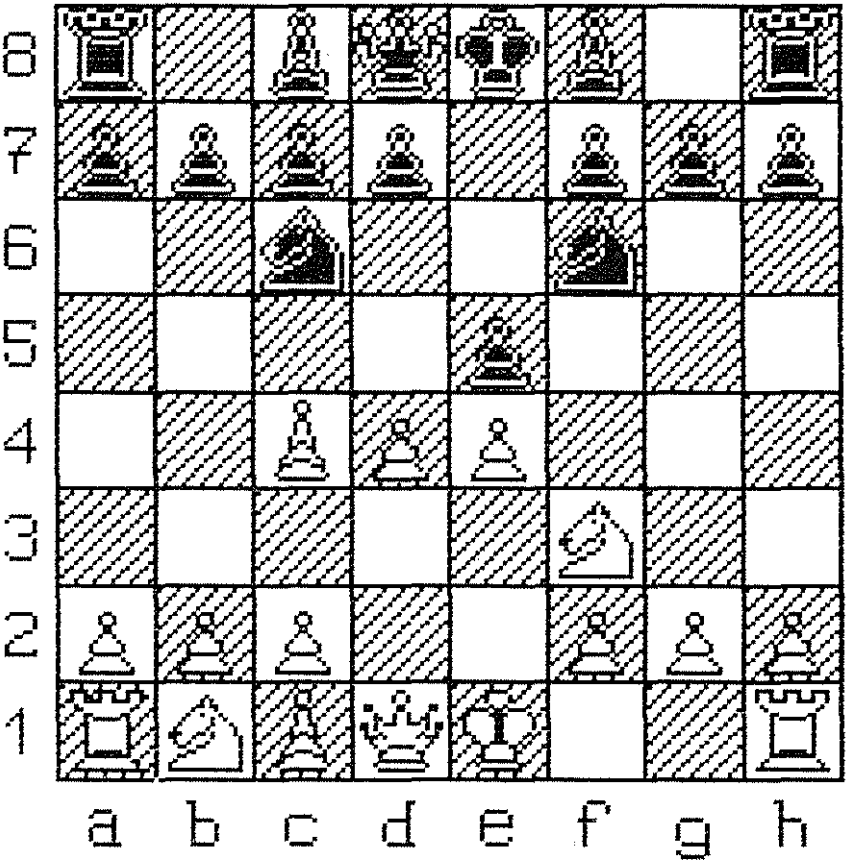


Schwarz hat dann oft eine Figur für drei Bauern getauscht und bedroht den weißen König massiv auf der H-Linie. Der Ausgang ist dann ungewiß - manchmal gibt es eine heftige

Tauscherei, bei der Weiß mit König und drei Bauern, Schwarz mit König und Turm übrigbleibt; dies Endspiel sollte dann Remis gehalten werden, jedenfalls bei den Konstellationen, die sich aus der oben gezeigten Stellung ergeben.
Mit dem Lettischen Gambit können Sie in jedem Fall experimentieren.

5.2.3 Regel 3: Keine taktischen Verwicklungen
Regel 4: Abtausch provozieren

Diese Regel in der Praxis vorzuführen, ist nicht ganz leicht. Betrachten wir einmal folgende Stellung, die aus dem berüchtigten Max-Lange-Angriff im Zweispringerspiel entsteht.



01	e2-e4	e7-e5
02	Sg1-f3	Sb8-c6
03	Lf1-c4	Sg8-f6
04	d2-d4	...

Dieser scharfe Zug löst zwangsläufig Tumulte aus. Der Übergang von der Eröffnung - wenn man die ersten drei Züge überhaupt so nennen kann - zum Mittelspiel ist fließend. Nehmen Sie einmal an, Sie spielten Weiß. Welche Fortsetzung wäre Ihnen lieber

1. Variante

04	...	e5xd4
05	Lc1-g5	Lf8-b4+
06	c2-c3	d4xc3
07	Sb1-c3	Lb4xc3+
08	b2xc3	Sc6-a5
09	Lc4-b3	Sa5xb3
10	a2xb3	h7-h6
11	Lg5xf6	Dd8xf6

usw.

2. Variante

04	...	e5xd4
05	0-0	Sf6xe4
06	Tf1-e1	d7-d5
07	Lc4xd5	Dd8xd5
08	Sb1-c3	Dd5-e6
09	Sf3xd4	Te1xe4

usw.

In der ersten Variante verliert Weiß durch zu frühes Abtauschen seine Angriffschancen, die es sich eigentlich durch den Gambit-Bauern erobern wollte - Der Sinn der Eröffnung ist damit verfehlt.
In der zweiten Variante startet Weiß mit Te1 eine Abtauschfolge, die letzlich zum Damengewinn führt (jedenfalls in dieser Variante).

Diese Aussagen lassen darauf schließen, daß es doch gut ist, die Stellung verwickelt zu gestalten, doch das ist nur in solchen Positionen richtig, wo der Spieler die Fortsetzung bis zum Ende der Tauschzüge absolut richtig vorherberechnet. Normalerweise ist das aber die Stärke der Schachprogramme, deshalb ist es besonders im Mittelspiel - wenn das Gambit für den Spieler schon einen Vorteil gebracht hat - empfehlenswert, die Stellung so zu bereinigen, daß sich der Eröffnungsvorteil in einen materiellen oder starken positionellen Vorteil ummünzen läßt.

Beim Spiel gegen einen menschlichen Gegner ist es oft besser, taktisch bestimmte Situationen noch weiter zu verkomplizieren, um eventuell einen direkten Mattangriff zu starten.

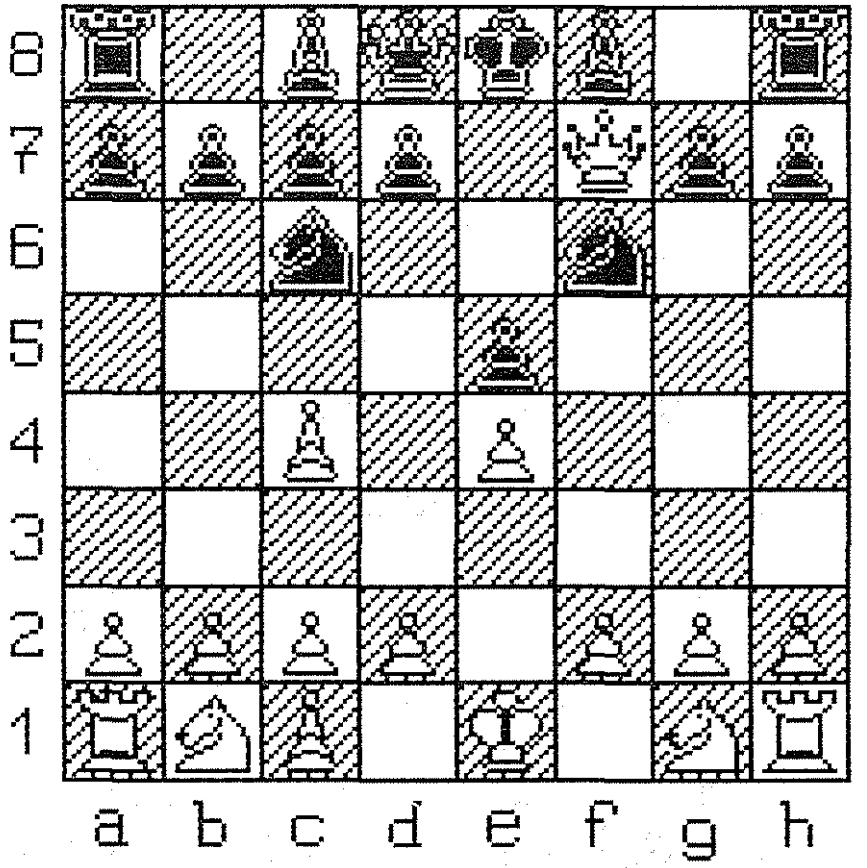
Beim Spiel gegen den Computer ist es besser, bei eigenem Vorteil klare Stellungen anzustreben. Vielleicht sind die Partien dann nicht so spektakulär, aber Ihre Chancen steigen.

5.2.4 Regel 5: Fallen stellen

Sicher kann man Schachprogrammen keine Fallen stellen, die so primitiv sind wie Schäfer-Matt und Seekadetten-Matt.

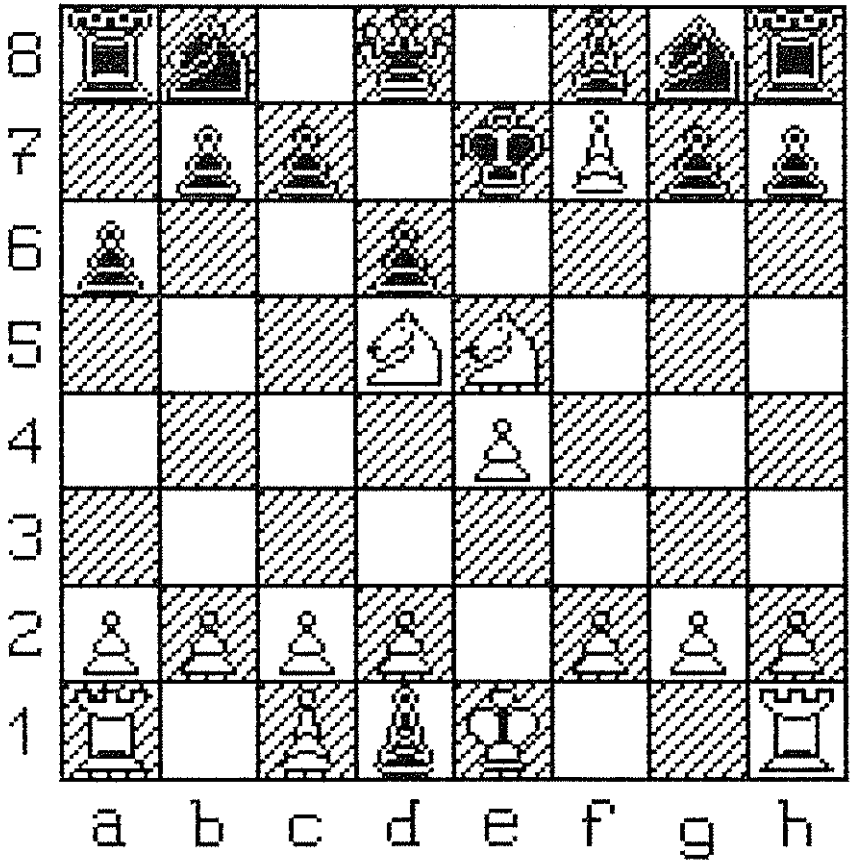
Schäfer-Matt:

1. e4 e5 2. Lc4 Sc6 3. Dh5 Sf6 4. Df7++


8
7
6
5
4
3
2
1
a b c d e f g h

Seekadetten-Matt:

1. e4 e5 2. Sf3 d6 3. Lc4 Lg4 4. Sc3 a6 5. Se5: Ld1: 6. Lf7:+ Ke7 7. Sd5++



Es gibt jedoch zwei Schwächen, die ein Schachprogramm auch im Mittelspiel hat:

- der Horizonteffekt und
- die Beseitigung von Fesselungen.

a) Horizonteffekt

Der Horizont, innerhalb dessen das Programm Folgezüge einer Stellung 'sehen' kann, ist durch die maximale Tiefe des Suchbaums begrenzt. Wie wir schon gesehen haben (Kapitel 1.4), kann ein Programm selbst Schlagabtäusche nicht bis zum Ende berechnen, wenn die Fähigkeit, dies zu tun, nicht implementiert ist. Die modernen Schachprogramme arbeiten beinahe alle mit einer solche Suche, die normal mit einer festen Tiefe arbeitet, aber tiefer sucht, wenn in der Endstellung der Maximaltiefe Schlagzüge gefunden werden.

Programme älterer Machart - bekannt für solches Fehlverhalten sind die käuflichen Schachcomputer aus den Jahren 1977, 1978 - fallen auf lange Schlagkombinationen herein. Das kann ein Spieler so leicht ausnutzen, daß wir diesen Effekt ignorieren können.

Lange Kombinationen ohne Schlagzüge können immer noch den begrenzten Horizont des Programmes verdeutlichen. Sehen Sie sich folgende Partie an; danach werden wir versuchen aus diesem speziellen Fall allgemeine Schlüsse zu ziehen.
Die Partie fanden wir in (KAP):

Weiß: Spieler
Schwarz: SARGON 2.5

01	d2-d4	Sg8-f6
02	c2-c4	e7-e6
03	Sg1-f3	b7-b6
04	Sb1-c3	Lc8-a6

Nachdem die Bibliothek keine Züge dieser Variante enthält, macht SARGON einen sehr seltsamen Zug - normal ist Lb7.

05	e2-e4	Lf8-b4
06	Sf3-e5	...

Der Versuch, SARGON hereinzulegen; nicht besonders empfehlenswert. Ld3 wäre solider.

06	...	Lb4xc3+
07	b2xc3	Sf6xe4
08	Lf1-d3	...

Soll wohl wieder eine Falle werden (8. ... Sc3: 9. Dc2 fängt den Springer). Aber warum spielt Weiß nicht kräftig: 8. Df3?

08	...	Se4-f6
09	Lc1-g5	...

Zwischenbilanz: Weiß hat nicht viel Vorteil durch seinen geopferten Bauern, um ehrlich zu sein: Gar keinen Vorteil. Die Frage ist, ob SARGON sein materielles Übergewicht in einen Sieg umsetzen kann.

09	...	0-0

Hat SARGON die Fesselung nicht erkannt? Oder unterschätzt? Jedenfalls ist es ganz schön riskant in den angegriffenen Königsflügel mit dem gefesselten Springer auf F6 zu rochieren.

10	Se5-g4	La6-b7
11	Sg4xf6+	g7xf6
12	Dd1-h5	...

Das Matt kann nicht mehr lange auf sich warten lassen.

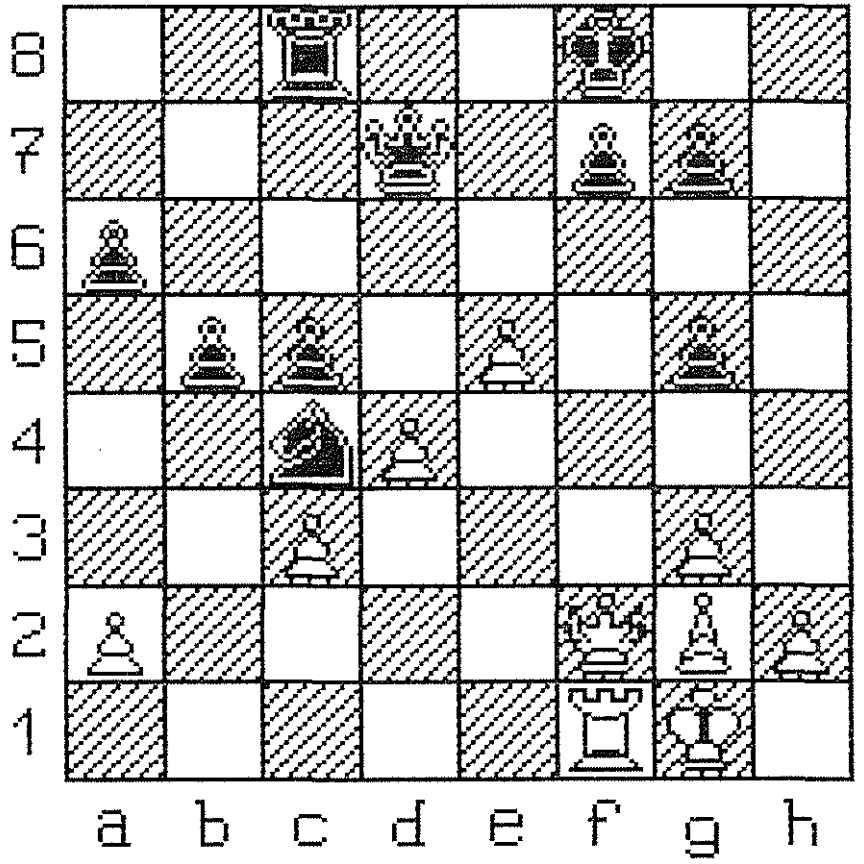
12	...	Tf8-e8
13	Dh5xh7+	Kg8-f8
14	Dh7-h6+	Kf8-g8
15	Ld3-h7+	Kg8-h8
16	Lh7-g6+	Kh8-g8
17	Dh6-h7+	Kg8-f8
18	Dh7xf7++	

SARGON hat die Fesselung seines Springers zu leicht genommen. Eine solche Fesselung bedeutet ja nicht nur, daß sich die gebundene Figur nicht bewegen kann, meist ist auch noch die deckende Figur mit gefesselt. So auch in dieser Partie: Die Königsstellung geht daran entzwei, daß der schlagende weiße Springer auf F6 mit dem Bauer genommen werden muß, der dadurch die G-Linie entblößt.

b) Fesselungen

Situationen dieser Art brauchen Sie gar nicht besonders zu provozieren; eine ganze Menge Eröffnungsvarianten führen zu Fesselungen. Wenn sicher ist, daß Schachprogramme Schwierigkeiten mit Fesselungen haben, so gilt das natürlich auch umgekehrt. Der berühmte schwarze Läufer auf G4, der den weißen Springer auf F3 fesselt, ist also für den Menschen, der die weißen Figuren führt, nicht so tragisch.

Als Beispiel hier eine Stellung, die zeigt, wie man mit Fesselungen die Figuren eines Programmes lahmgelegen kann.



Weiß am Zug

01 Lg2-h3 ...

Die Dame darf natürlich nicht nehmen, denn dann gibt die weiße Dame auf F7 Matt. Solche Situationen machen jedem Programm schwer zu schaffen - eine gegnerische Figur 'hängt' (ist angegriffen, aber ungedeckt), kann aber nicht geschlagen werden.

c) Trojanisches Pferd

Es gibt eine Falle, auf die der größte Teil aller kommerziellen Schachprogramme hereinfällt: Das trojanische Pferd. Nachdem der Spieler mit Weiß eine scheinbar völlig unsinnige Eröffnung spielt, gewinnt er plötzlich innerhalb von 5 Zügen.

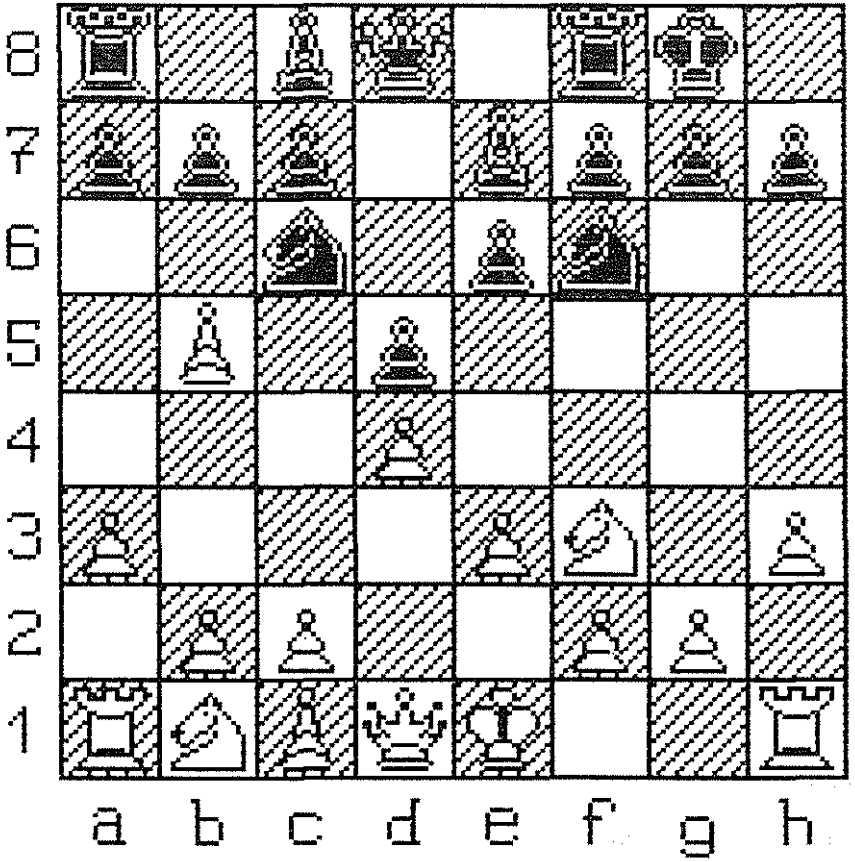
Ein Beispiel liefert unser Demonstrationsprogramm (DEMOSCHACH):

Gespielt am 13. November 1985
Weiß: Spieler
Schwarz: DEMOSCHACH (Stufe 3)

01	d2-d4	Sg8-f6
02	e2-e3	d7-d5
03	h2-h3	Sb8-c6
04	Lf1-b5	e7-e6
05	a2-a3	Lf8-e7
06	Sg1-f3	0-0

Nach diesen weniger aufregenden Anfangszügen ergibt sich eine völlig harmlose Stellung. Vermutlich wird Weiß gleich rochieren, Schwarz den Springer auf C6 entfesseln usw.

Aber erstens kommt es anders und zweitens als man denkt; in sechs Zügen ist Schwarz matt!



07	Sf3-g5	h7-h6
08	h3-h4	h6xg5
09	h4xg5	Sf6-e4
10	Dd1-h5	f7-f5
11	g5-g6	Le7-b4+
12	a3xb4	Sc6xb4
13	Dh5-h7++	

Was denkt sich das Programm denn dabei? Fangen wir anders an: Der menschliche Spieler verfolgt einen klaren Plan, er will den schwarzen König mattsetzen und zwar unter Verwendung der offenen H-Linie. Damit das klappen kann, definiert er sich Teilziele. Erstens muß Schwarz kurz rochieren, zweitens muß der schwarze Springer auf F6 vorhanden sein, drittens soll kein schwarzer Läufer sich innerhalb des Felderrechtecks F8-H8-H3-F3 befinden, viertens muß ein weißer Springer auf F3 plaziert sein und fünftens darf Weiß nicht kurz rochieren.

Nur um diese Teilziele zu erreichen, macht Weiß derart komische Züge in der Entwicklung; z.B. bewirkt im dritten Zug h2-h3, daß Lc8-g4 nicht geschehen kann.

Der Zug Nummer 5, a2-a3 gaukelt dem Programm einen Entwicklungsvorsprung vor, den es mit der Rochade ausnutzen will.

Wenn jetzt der Springer auf G5 sitzt, will ihn Schwarz mit H6 vertreiben und so ein weiteres Tempo gewinnen. Dadurch, daß Weiß das Pferd mit H4 deckt, ergibt sich für Schwarz die Gelegenheit, Material zu gewinnen. Der Abtausch auf G5 führt zur offenen H-Linie.

Die Folgen des Springerklaus kann das Programm innerhalb seines Horizonts nicht sehen; nachdem Weiß auf G5 zurückgenommen hat, folgen keine weiteren Schlagzüge auf diesem Feld.

Nichts ahnend rennt das Programm ins Verderben; denn wenn die weiße Dame auf H5 erscheint, ist es zu spät. Bei Programmen, die den Wert der Stellung anzeigen, sieht man nach diesem Damenzug, daß sich die Bewertung drastisch zu Ungunsten von Schwarz verändert. Zwar versucht das Programm mit Läuferschach auf B4 oder ähnlichem die Situation zu bereinigen, diese Versuche sind vergeblich.

Das Experiment "Trojanisches Pferd" läßt sich auf jedem Schachcomputer bei jeder Spielstufe nachvollziehen, wenn die obengenannten Voraussetzungen gegeben sind. Deshalb ist diese Falle bei gewieften Computerschachspielern auch bekannt.

5.2.5 6. Regel: Bei Vorteil Endspiel anstreben
7. Regel: Im Endspiel nach Plan vorgehen

Es ist wirklich eine Binsenweisheit, daß Computer ein schlechtes Endspiel spielen. Es stellen sich nur zwei Fragen.

Erstens: Wie kriege ich den Computer dazu, ins Endspiel einzusteigen?

Zweitens: Wie gewinne ich dann das Endspiel?

a) Vorteil erkennen

Die erste Frage läßt sich leicht beantworten: Indem man die Figuren abtauscht. Je weniger Figuren auf dem Brett sind, desto näher ist das Endspiel.

Wie muß ein Vorteil beschaffen sein, der es erlaubt, das Endspiel anzustreben? Es kann sich um einen materiellen oder um einen positionellen Vorteil handeln. Der rein materielle Vorteil ist leicht zu erkennen; haben Sie einen Mehrbauern aus dem Mittelspiel gewonnen, sollten Sie so oft abtauschen, daß im Idealfall nur noch der Mehrbauer und Ihr König, sowie der gegnerische König auf dem Brett sind. Gleiches gilt natürlich, wenn Sie eine ganze Figur mehr haben, wobei eine Leichtfigur (Läufer oder Springer) unter Umständen später gegen einen Bauern getauscht werden muß, um einen Mehrbauern zu erhalten, der dann zur Dame geht.

Ein positioneller Vorteil ist schwerer zu erkennen. Ein Vorteil kann zum Beipiel bei Bauerngleichheit in stark blockierten Stellungen der Besitz eines Springers sein. Auch wenn Sie im Gegensatz zum Computer über eine offene Linie für Ihren letzten Turm verfügen, ist das ein Pluspunkt. Ein zentralisierter König, der eventuell gegnerische Bauern rauben kann, ist möglicherweise ein Vorteil.

Eigentlich hilft nur viel Üben und ein bißchen Schachliteratur dabei, ein Gefühl für Vor- und Nachteile in einer Stellung zu bekommen. Nebenbei führt besseres Schachgefühl auch dazu, daß Sie die Bewertungsfunktion eines Programmes besser verstehen, denn dieses Modul ist ja nichts anderes als das dem Programm eingepflanzte Schachwissen.

b) Endspielbehandlung

In normalen Endspielsituationen versagen Programme eigentlich selten. Es gilt also, in jedem Fall sehr präzise zu spielen. Dazu muß man die "Technik des Endspiels" beherrschen. Technik meint hier die Fähigkeit, einen Plan fehlerfrei auszuführen.

Ein Schachmeister verfolgt nicht einen Plan durch die ganze Partie, sondern operiert mit überschaubaren Teilplänen; in der Eröffnung kann der Plan z.B. heißen:

"Ich will das Zentrum beherrschen, meinen König kurz rochieren und einen scharfen Angriff auf dem Damenflügel anzetteln, mit dem Ziel, eine Bauernüberlegenheit auf diesem Flügel zu erreichen."

Ein Plan für das Mittelspiel könnte sein:

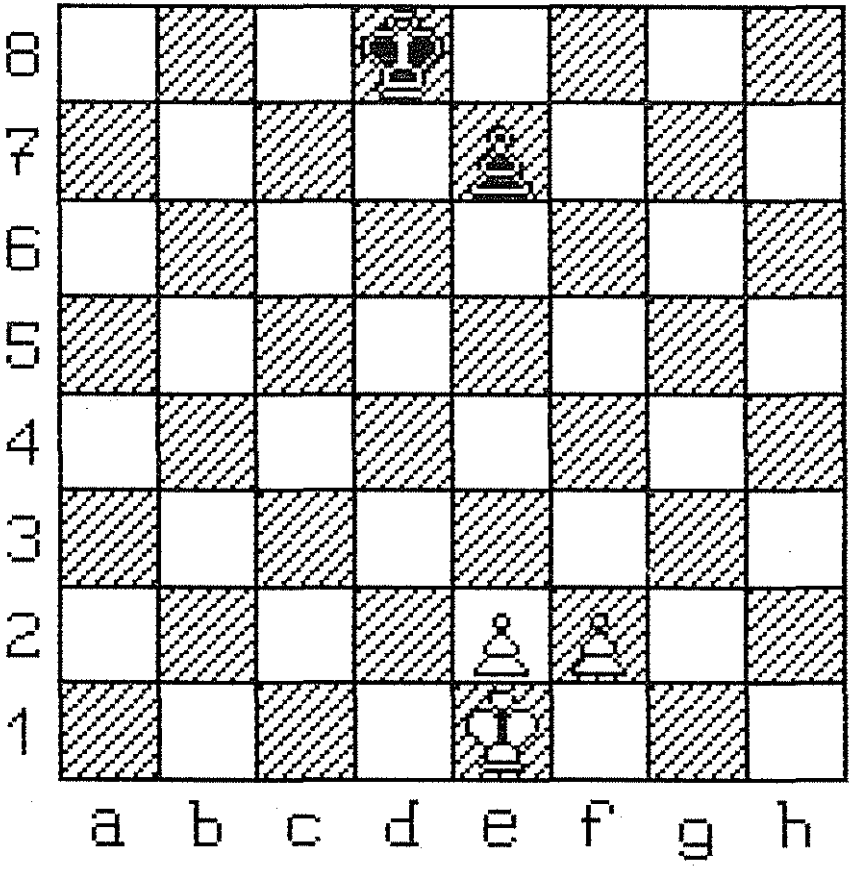
> "Ich will eine feste Bauernstruktur errichten, die den Gegner an Vorstößen hindert, seinen Figuren die wertvollen Felder raubt, mit dem Ziel, durch Abtäusche einen Freibauern zu erreichen."

Im Endspiel plant der Meister:

> "Ich will meinen Freibauern zur achten Reihe bringen und gleichzeitig den gegnerischen König daran hindern, diesen Bauern zu schlagen."

Solche Pläne sind natürlich situationsbezogen. Ein Beispiel für ein Endspiel:

Weiß am Zug



Wie könnte hier der Plan lauten? Etwa so:

"Der weiße König muß vor seinen Bauern gehen, um den schwarzen König zu verdrängen. Da der schwarze König dies ebenfalls versucht, muß Weiß schneller sein. Erreicht der weiße König die

schwarzen Bauern, bevor der schwarze König hinter die weißen Bauern kommt, gewinnt er."

Dieser Plan muß in die Tat umgesetzt werden.

01	Ke1-d2	Kd8-d7
02	Kd2-d3	Kd7-d6
03	f2-f4	...

Widerspricht dem Plan, ist also nicht ganz korrekt.

03	...	Kd6-d5
04	e2-e4+	...

Auch dieser Zug fördert nicht den Plan, sondern behindert ihn - der weiße König kommt auf diese Weise nicht vor seine Bauern.

04	...	Kd5-d6
05	Kd3-d4	...

Mit diesem Zug verschenkt Weiß den Gewinn. Der Plan ist gescheitert. Es folgt nämlich

05	...	e7-e5+
06	Kd4-e3	e5xf4+
07	Ke3xf4	Kd6-e6
08	e4-e5	...

Spieler mit Erfahrung erkennen sofort, daß sich der schwarze König zurückziehen muß, will er das Remis halten.

09	...	Ke6-e7

Dieser Zug verhindert, daß der weiße König den Bauern bis zum Umwandlungsfeld begleiten und schützen kann. Die Partie wird Remis.

Wie hätte Weiß spielen müssen, um zu gewinnen?

01	Ke1-d2	Kd8-d7
02	Kd2-d3	Kd7-d6
03	Kd3-e4	Kd6-e6
04	f2-f4	Ke6-f6
05	f4-f5	Kf6-f7
06	Ke4-e5	Kf7-e8
07	e2-e4	Ke8-d7
08	Ke5-d5	Kd7-d8
09	Kd4-c5	Kd8-e8
10	e4-e5	Ke8-f7
11	Kc5-d6	Kf7-f8
12	e5-e6	Kf8-g7
13	Kd7xe7	...

Das war's. Schwarz hat in dieser Stellung nicht optimal gespielt, das Ergebnis hätte aber auch nicht wesentlich verbessert werden können.

Ach so, Weiß wurde gespielt von unserem DEMOSCHACH (auf dem PC), Schwarz spielte einer der Autoren (der schlechteste Schachspieler des Teams).

Man sieht, daß auch Computer das Endspiel korrekt bewältigen können. Was läßt sich denn nun raten in Sachen Endspiel? Verbessern Sie Ihre Endspieltechnik, indem Sie oft Endspielpositionen aufbauen und gegen den Computer durchspielen. Informieren Sie sich darüber, wie Schachmeister diese Spielphase beherrschen und machen Sie sich mit den immer wiederkehrenden Bildern im Endspiel vertraut.

Das hilft Ihnen nicht nur den Computer zu besiegen, sondern wird auch Ihre Spielstärke gegen menschliche Gegner ganz erheblich verbessern.

5.2.6 Was haben Sie davon?

Die Beachtung der 7 goldenen Regeln führt nur oberflächlich betrachtet dazu, daß Sie ein Schachprogramm besiegen können; der viel wichtigere Effekt ist, daß Sie Ihre Spielstärke verbessern.

Größere Fähigkeiten im Schach wiederum steigern den Spaß am Spiel ganz erheblich. Und wenn Sie erstmal das Schachprogramm nach Belieben schlagen, werden Sie auch die Schwierigkeiten der Schachprogrammierung besser verstehen können. Und mehr wollte Ihnen unser Großes Computerschach-Buch eigentlich nicht vermitteln.

6.1 Verzeichnis der erwähnten Programme und Schachcomputer

Name:	Autor(en)/Hersteller:
BELLE	Joe Condon & Ken Thompson (USA)
BOBBY	H.-J. Kraas & G. Schrüfer (BRD)
CHESS 2.0 bis 4.7	David Slate & Larry Atkin (USA)
CRAY BLITZ	Robert Hyatt, Albert Gower & Harry Nelson (USA)
ITEP	Michail Domskoi, Wladimir Arlasarow und andere (UdSSR)
KAISSA	Michail Domskoi, Wladimir Arlasarow und andere (UdSSR)
MAC HACK VI	Richard Greenblatt (USA)
MEPHISTO	Hegener + Glaser (BRD)
OSTRICH	Monroe Newborn (USA)
PIONIER	Michail Botwinnik und andere (UdSSR)
SARGON I bis III	Dan & Kathe Spracklen (USA)
SENSORY 9	Fidelity Electronics (USA)
TURBOSTAR 432	SciSys Inc. (Hongkong)

6.2 Literaturverzeichnis

Zum Thema Computerschach existiert nur sehr wenig deutschsprachige Literatur, deshalb sind die meisten aufgeführten Titel englischsprachig. Das gilt besonders für die Bücher und Artikel, auf die im Text Bezug genommen wird. Diese Werke sind zum Teil schwer zu beschaffen und nicht ganz billig, für den ambitionierten Computerschach-Freund lohnt sich die Anschaffung der wichtigsten Titel, die mit einem "*" gekennzeichnet sind, trotzdem.

6.2.1 Erwähnte Titel

* BER: A.Bernstein et al.: A Chessplaying Program For IBM 704 Computer Proc. Western Joint Computer Conference 1958 S. 157 - 159

* FRE: P.Frey (Hrsg.): Chess Skill in Manand Machine Springer-Verlag New York 1978

* GRE: R.D.Greenblatt et al.: The Greenblatt Chess Program Proc. AFIPS Fall Joint Computer Conference 1967, 31 S. 801 - 810

GRO: H. Grob: Die Eröffnungen in der Schachpartie Schachverlag Grob Zürich 1974

KAP: J. Kaplan: How To Get The Most From Your Chess Computer R.H.M. Press Great Neck 1980

KMO: H. Kmoch: Die Kunst der Bauernführung
Siegfried Engelhardt Verlag
Berlin 1956

* LEV: D. Levy: Chess And Computers
Batsford Chess Books
London 1976

* NEW: M. Newborn: Computer Chess
Academic Press New York
1975

* SHA1: C. E. Shannon: Programming A Computer For Playing Chess
Philosophical Magazine 1950,41
S. 256 - 275

* SHA2: C. E. Shannon: Programming A Computer To Play Chess
Scientific American 2/1950
S. 48 - 51

* SLA: D. Slate, L. Atkin: CHESS 4.5 - The Northwestern University Chess Program
in (FRE) - siehe dort

WEL: D. E. Welsh: Computer Chess
Wm. C. Brown Publishers
Dubuque 1984

6.2.2 Verschiedene Titel zum Thema Computerschach

M. Gittel:	SARGON - Porträt eines Schachprogramms Eigenverlag Salzgitter 1983
Ketterling, Schwenkel, Weiner:	Schach dem Computer Mosaik Verlag München 1980
H. C. Opfermann:	Spielen mit dem Schachcomputer ECON Verlag Düsseldorf 1980
Pachmann, Kühnmund:	Computerschach Heyne Verlag München 1980
B.Schwarz:	Heim-Schachcomputer Kiwi-Verlag München 1981

6.2.3 Titel zum Thema Strategiespiele

R.Baumann:	Computerspiele und Knobeleien programmiert in BASIC Vogel Verlag Würzburg 1983
v. Neumann, Morgenstern:	Theory Of Games And Economic Behaviour Princeton University Press 1944
W. Schneider:	Strategiespiele und wie man sie auf dem ATARI 600XL / 800XL programmiert DATA BECKER Verlag 1985

6.3 Komplettes Listing: DEMOSCHACH

```
10 PRINT"            DEMOSCHACH"
11 PRINT
12 PRINT"   (C)1985 G.SCHRUEFER,H.-J.KRAAS"
100 REM KONSTANTEN
102 SK%=1:SD%=2:ST%=3:SL%=4:SS%=5:SB%=6
104 W8%=8:WS%=9:WL%=10:WT%=11:WD%=12
106 WK%=13:FF%=7:KF%=14
108 DIM FW%(12)
110 FOR I=2 TO 12:READ FW%(I):NEXT I
118 DIM SW%(8),DW%(8)
120 FOR I=1 TO 8:READ SW%(I):NEXT I
124 FOR I=1 TO 8:READ DW%(I):NEXT I
128 DIM KO% (118)
130 FOR I=24 TO 108
132 FOR J=3 TO 10:KO%(I+J)=20:NEXT J
134 I=I+11:NEXT I
136 FOR I=48 TO 84
138 FOR J=5 TO 8:KO%(I+J)=KO%(I+J)+40:NEXT J
140 I=I+11:NEXT I
142 FOR I=60 TO 72
144 FOR J=6 TO 7:KO%(I+J)=KO%(I+J)+40:NEXT J
146 I=I+11:NEXT I
150 DIM KV%(10)
152 FOR I=3 TO 10:READ KV%(I):NEXT I
155 WS$="W=WEISS,S=SCHWARZ"
156 ZK$="WEISS:  SCHWARZ:KOENIG  DAME    TURM    LAEUFER SPRINGERBAUER
"
158 BL$="! . . . . "
160 REM VARIABLEN (FELDER)
162 DIM BR%(144),FA%(118)
164 DIM RO%(2,2)
166 DIM FI%(2,16),FS%(2,16),FZ%(2)
168 DIM Z1%(400,2),Z2%(100,3)
170 DIM ST%(8,44)
172 DIM ZV%(6)
174 DIM HV%(4,4,3)
176 DIM KI%(8,4)
178 DIM SI%(3)
```

```
180 DIM BF%(2,11,6),BZ%(2,11)
182 DIM BV%(2,10),FB%(2,10),RB%(2,10)
184 DIM ES%(2),RF%(2),MS%(2)
188 DIM A0%(2)
200 REM DATA-ZEILEN
202 DATA -900,-500,-325,-325,-100,0,100,325,325,500,900
204 DATA 10,23,25,14,-10,-23,-25,-14
206 DATA 11,13,-11,-13,1,12,-1,-12
208 DATA 0,0,390,540,700,230,0,0
210 DATA 13,31,12,30,11,27,11,34,10,29,10,32,9,28,9,33
212 DATA 8,39,8,40,8,41,8,42,8,43,8,44,8,45,8,46
214 DATA 1,115,2,114,3,111,3,118,4,113,4,116,5,112,5,117
216 DATA 6,99,6,100,6,101,6,102,6,103,6,104,6,105,6,106
218 DATA 11,3,9,7,10,5,12,2,13,1,10,6,9,8,11,4
220 DATA 8,9,8,10,8,11,8,12,8,13,8,14,8,15,8,16
222 DATA 6,9,6,10,6,11,6,12,6,13,6,14,6,15,6,16
224 DATA 3,3,5,7,4,5,2,2,1,1,4,6,5,8,3,4
600 GOSUB 9000:REM INITIALISIERUNG
602 GOSUB 11200:REM STELLUNGSAUSGABE
606 IF GW%=W% THEN GOTO 636
608 REM PROGRAMM AM ZUG
610 GOSUB 6800: REM ZUGBESTIMMUNG DURCH SUCHE
612 REM SONDERFAELLE
614 IF P% THEN PRINT "ICH BIN PATT!": END
616 IF W% AND (MW%>-15000) THEN GOTO 622
618 IF (NOT W%) AND (MW%<15000) THEN GOTO 622
620 PRINT "ICH GEBE AUF!": END
621 REM AUSGABE DES AUSGEWAEHLTEN ZUGES
622 PRINT "MEIN ZUG IST:  ";
623 FOR I=1 TO 2:A0%(I)=HV%(1,1,I):NEXT I
624 GOSUB 11600
625 A1%=BR%(A0%(1))
626 IF (A1%=WB%)AND(A0%(2)>106) THEN A1%=HV%(1,1,3):GOSUB 11400:GOTO 628
627 IF (A1%=SB%)AND(A0%(2)<39) THEN A1%=HV%(1,1,3):GOSUB 11400
628 PRINT
629 PRINT "DER WERT FUER DIESEN ZUG IST",MW%
630 REM AKTUALISIEREN DER PARTIESTELLUNG
631 S%=HV%(1,1,1):Z%=HV%(1,1,2):NF%=HV%(1,1,3)
632 ON (W%+2) GOSUB 8005,8050:GOSUB 4300
634 GOSUB 11200: REM STELLUNGSAUSGABE
```

```
635 IF ABS(MW%)>=15998 THEN PRINT "SIE SIND MATT!": END
636 REM GEGNER AM ZUG
638 GOSUB 9300: REM BEHANDLUNG DES GEGNERISCHEN ZUGES
640 IF S%>1 THEN GOSUB 11200: GOTO 610
998 END
999 REM@M
1000 REM DAMEN,TURM,LAEUFERZUEGE
1001 HW%=0:A4%=1
1002 IF (ZF%=WT%) OR (ZF%=ST%) THEN A1%=5:A4%=3:GOTO1004
1003 A1%=1
1004 IF (ZF%=WL%) OR (ZF%=SL%) THEN A2%=4:A4%=4:GOTO1009
1005 A2%=8
1009 FOR J=A1% TO A2%
1011 A3%=DW%(J)
1013 Z%=S%
1015 Z%=Z%+A3%
1020 F%=BR%(Z%)
1025 IF F%=KF% THEN GOTO 1085
1027 HW%=HW%+KO%(Z%)
1030 IF F%<>FF% THEN GOTO 1050
1035 REM RUHIGER ZUG
1040 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1045 GOTO 1015
1050 IF (F%>FF%)=W% THEN GOTO 1085
1055 REM SCHLAGZUG
1060 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1085 NEXT J
1086 REM FELDERKONTROLLEN
1087 IF T%<9 THEN ST%(T%,44)=ST%(T%,44)+HW%*A4%
1090 RETURN
1100 REM SPRINGERZUEGE
1101 FOR J=1 TO 8
1105 Z%=S%+SW%(J)
1110 F%=BR%(Z%)
1115 IF F%=KF% THEN GOTO1170
1120 IF F%=FF% THEN GOTO1155
1125 IF (F%>FF%)=W% THEN GOTO1170
1130 REM SCHLAGZUG
1135 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1145 GOTO 1170
```

```
1150 REM RUHIGER ZUG
1155 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1170 NEXT J
1175 RETURN
1200 REM BAUERNZUEGE
1201 REM WEISSER BAUER
1205 Z%=S%+12:REM EIN FELD VOR
1206 IF BR%(Z%)<>FF% THEN GOTO 1225
1207 IF Z%>106 THEN GOSUB 1260:GOTO1225
1208 REM RUHIGER ZUG
1209 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1215 Z%=S%+24:REM ZWEI FELDER VOR
1216 IF BR%(Z%)<>FF% THEN GOTO 1225
1217 IF S%>46 THEN GOTO 1225
1218 REM RUHIGER ZUG
1219 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1225 Z%=S%+11:REM SCHLAGEN LINKS OBEN
1226 F%=BR%(Z%)
1227 IF F%=KF% THEN GOTO 1235
1228 IF F%>SB% THEN GOTO 1235
1229 IF Z%>106 THEN GOSUB1260:GOTO1235
1230 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1235 Z%=S%+13:REM SCHLAGEN RECHTS OBEN
1236 F%=BR%(Z%)
1237 IF F%=KF% THEN GOTO 1245
1238 IF F%>SB% THEN GOTO 1245
1239 IF Z%>106 THEN GOSUB1260:GOTO1245
1240 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1245 RETURN
1260 REM BAUERNUMWANDLUNG WEISS
1265 F%=WD%
1266 FOR J1=I2%+1 TO I2%+4
1267 Z2%(J1,1)=S%:Z2%(J1,2)=Z%:Z2%(J1,3)=F%
1270 F%=F%-1
1271 NEXT J1
1272 I2%=I2%+4
1273 RETURN
1300 REM SCHWARZER BAUER (ANALOG ZU 1201-1273)
1305 Z%=S%-12
1306 IF BR%(Z%)<>FF% THEN GOTO 1325
```

```
1307 IF Z%<39 THEN GOSUB 1360:GOTO1325
1309 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1315 Z%=S%-24
1316 IF BR%(Z%)<>FF% THEN GOTO 1325
1317 IF S%<99 THEN GOTO 1325
1319 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1325 Z%=S%-13
1326 F%=BR%(Z%)
1327 IF F%=KF% THEN GOTO 1335
1328 IF F%<WB% THEN GOTO 1335
1329 IF Z%<39 THEN GOSUB1360:GOTO1335
1330 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1335 Z%=S%-11
1336 F%=BR%(Z%)
1337 IF F%=KF% THEN GOTO 1345
1338 IF F%<WB% THEN GOTO 1345
1339 IF Z%<39 THEN GOSUB1360:GOTO1345
1340 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1345 RETURN
1360 REM BAUERNUMWANDLUNG SCHWARZ
1365 F%=SD%
1366 FOR J1=I2%+1 TO I2%+4
1367 Z2%(J1,1)=S%:Z2%(J1,2)=Z%:Z2%(J1,3)=F%
1370 F%=F%+1
1371 NEXT J1
1372 I2%=I2%+4
1373 RETURN
1400 REM KOENIGSZUEGE
1405 FOR J=1 TO 8
1415 Z%=S%+DW%(J)
1420 F%=BR%(Z%)
1425 IF F%=KF% THEN GOTO 1485
1430 IF F%=FF% THEN GOTO 1465
1435 IF (F%>FF%)=W% THEN GOTO 1485
1440 REM SCHLAGZUG
1445 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%
1456 GOTO1485
1460 REM RUHIGER ZUG
1465 I1%=I1%+1:Z1%(I1%,1)=S%:Z1%(I1%,2)=Z%
1485 NEXT J
```

```
1490 RETURN
1600 REM SONDERFAELLE
1610 REM EN-PASSANT-SCHLAGEN
1615 IF EP%=0 THEN GOTO 1645
1620 IF W% THEN A1%=12:A2%=WB%:A3%=SB%:GOTO1625
1621 A1%=-12:A2%=SB%:A3%=WB%
1625 S%=EP%+1
1626 IF BR%(S%)<>A2% THEN GOTO 1635
1627 Z%=EP%+A1%
1628 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%:Z2%(I2%,3)=A3%
1635 S%=EP%-1
1636 IF BR%(S%)<>A2% THEN GOTO 1645
1637 Z%=EP%+A1%
1638 I2%=I2%+1:Z2%(I2%,1)=S%:Z2%(I2%,2)=Z%:Z2%(I2%,3)=A3%
1645 RETURN
1700 REM ROCHADEN
1705 A1%=W%+2
1710 IF NOT RO%(A1%,2) THEN GOTO 1760
1715 REM KURZE ROCHADE
1720 IF W% THEN A2%=32:GOTO1730
1725 A2%=116
1730 A3%=(BR%(A2%)=FF%) AND (BR%(A2%+1)=FF%)
1735 IF NOT A3% THEN GOTO 1760
1740 I1%=I1%+1:Z1%(I1%,1)=A2%-1:Z1%(I1%,2)=A2%+1
1755 REM LANGE ROCHADE
1760 IF NOT RO%(A1%,1) THEN GOTO 1800
1765 IF W% THEN A2%=28:GOTO1775
1770 A2%=112
1775 A3%=(BR%(A2%)=FF%) AND (BR%(A2%+1)=FF%) AND (BR%(A2%+2)=FF%)
1780 IF NOT A3% THEN GOTO 1800
1785 I1%=I1%+1:Z1%(I1%,1)=A2%+3:Z1%(I1%,2)=A2%+1
1800 RETURN
1899 REM@P
1900 REM ZUEGE GENERIEREN
1902 REM@M
1905 A5%=W%+2
1906 ZV%(1)=I1%:ZV%(2)=I2%:ZV%(3)=I2%:ZV%(4)=1
1907 IF T%<9 THEN ST%(T%,44)=0
1908 GOSUB 1610:REM EN-PASSANT-SCHLAGEN
1909 GOSUB 1700:REM ROCHADEN
```

```
1910 FOR J3=1 TO FZ%(A5%)
1913 ZF%=FI%(A5%,J3)
1915 IF ZF%=FF% THEN GOTO1940
1916 S%=FS%(A5%,J3)
1925 ONZF%GOSUB1400,1000,1000,1000,1100,1300,1950,1200,1100,1000,1000,
1000,1400
1940 NEXT J3
1949 REM@P
1950 RETURN
2000 REM BEWERTUNGSFUNKTION
2001 REM@M
2002 GOSUB 3014:REM ANALYSE DES STELLUNGSTYPS
2003 ON PT% GOTO 2030,2034,2038
2004 REM NORMALE BEWERTUNGSFUNKTION
2005 REM BEWERTUNGSABBRUCH BEI ZU ERWARTENDEM ALPHA- BZW. BETA-CUTOFF
2006 IF MB%+120<=AL% THEN PW%=MB%+120:GOTO 2088
2007 IF MB%-120>=BE% THEN PW%=MB%-120:GOTO 2088
2008 REM ABSCHAETZUNG FUER RUHESUCHE
2009 IF T%>MT% THEN PW%=MB%+RS%:GOTO 2088
2010 PW%=MB%:REM MATERIALBEWERTUNG
2011 IF NOT(ES%(1) AND ES%(2)) THEN GOTO 2014
2012 IF MB%>0 THEN PW%=PW%+BZ%(1,1)*10:GOTO 2014
2013 IF MB%<0 THEN PW%=PW%-BZ%(2,1)*10
2014 HW%=0
2015 GOSUB 2104:REM BAUERNBEWERTUNG
2016 GOSUB 2302:REM FIGURENBEWERTUNG
2018 GOSUB 2504:REM KOENIGSBEWERTUNG
2021 REM BEWERTUNG DER KONTROLLEN
2022 IF T%=1 OR T%>8 THEN A1%=0:GOTO 2026
2023 A1%=ST%(T%,44)-ST%(T%-1,44)
2024 IF NOT W% THEN A1%=-A1%
2025 REM GESAMTBEWERTUNG
2026 PW%=PW%+INT((HW%+A1%)/100)
2027 RS%=PW%-MB%
2028 GOTO 2088
2029 REM MATTFUEHRUNG GEGEN WEISS
2030 GOSUB 3202:GOTO 2088
2032 REM REMISSTELLUNG
2034 PW%=0:GOTO 2088
2036 REM MATTFUEHRUNG GEGEN SCHWARZ
```

```
2038 GOSUB 3102
2088 REM@P
2089 RETURN
2099 REM@M
2100 REM BEWERTUNG FUER DIE WEISSEN BAUERN
2102 REM INITIALISIERUNGEN
2104 FOR I=3 TO 10:FB%(1,I)=0:RB%(1,I)=0:NEXT I
2106 IF ES%(1) THEN GOTO 2112
2108 FOR I=3 TO 10:BV%(1,I)=KV%(I):NEXT I
2110 GOTO 2118
2112 FOR I=3 TO 10:BV%(1,I)=200:NEXT I
2116 REM BLOCKIERUNG DER ZENTRUMSBAUERN
2118 IF (BR%(42)=WB%) AND (BR%(54)<>FF%) THEN HW%=-1000
2120 IF (BR%(43)=WB%) AND (BR%(55)<>FF%) THEN HW%=HW%-1000
2122 FOR I=3 TO 10
2124 IF BZ%(1,I)=0 THEN GOTO 2190
2126 REM ISOLIERTE BAUERN
2128 IF (BZ%(1,I-1)=0)AND(BZ%(1,I+1)=0) THEN HW%=HW%-1500
2130 REM MEHRFACHBAUERN
2132 IF BZ%(1,I)<2 THEN GOTO 2137
2133 IF BZ%(1,I)=2 THEN HW%=HW%-400:GOTO 2137
2134 HW%=HW%-4000
2136 REM FREIBAUERN
2137 S%=BF%(1,I,BZ%(1,I))
2142 IF (BZ%(2,I-1)<>0)AND(BF%(2,I-1,BZ%(2,I-1))>S%  ) THEN GOTO 2160
2144 IF (BZ%(2,I  )<>0)AND(BF%(2,I  ,BZ%(2,I  ))>S%  ) THEN GOTO 2160
2146 IF (BZ%(2,I+1)<>0)AND(BF%(2,I+1,BZ%(2,I+1))>S%+1) THEN GOTO 2160
2148 FB%(1,I)=-1
2150 A1%=115
2152 IF BR%(S%+12)<FF% THEN A1%=80
2154 IF (BR%(S%-13)=WB%)OR(BR%(S%-11)=WB%) THEN A1%=A1%+35
2156 IF ES%(1) OR ES%(2) THEN A1%=A1%*2
2157 A2%=INT(S%/12)-1
2158 HW%=HW%+A1%*A2%*A2%
2160 FOR J=1 TO BZ%(1,I)
2162 S%=BF%(1,I,J)
2164 REM VORRUECKEN DER BAUERN
2166 HW%=HW%+(INT(S%/12)-3)*BV%(1,I)
2168 REM RUECKSTAENDIGE BAUERN
2178 IF (BR%(S%-1)=WB%)OR(BR%(S%-13)=WB%) THEN GOTO 2186
```

```
2180 IF (BR%(S%+1)=WB%)OR(BR%(S%-11)=WB%) THEN GOTO 2186
2182 IF BR%(S%+23)=SB% THEN RB%(1,I)=RB%(1,I)-200
2184 IF BR%(S%+25)=SB% THEN RB%(1,I)=RB%(1,I)-200
2186 NEXT J
2188 HW%=HW%+RB%(1,I)
2190 NEXT I
2200 REM BEWERTUNG FUER DIE SCHWARZEN BAUERN (ANALOG ZU 2100-2190)
2204 FOR I=3 TO 10:FB%(2,I)=0:RB%(2,I)=0:NEXT I
2206 IF ES%(2) THEN GOTO 2212
2208 FOR I=3 TO 10:BV%(2,I)=KV%(I):NEXT I
2210 GOTO 2218
2212 FOR I=3 TO 10:BV%(2,I)=200:NEXT I
2218 IF (BR%(102)=SB%) AND (BR%(90)<>FF%) THEN HW%=HW%+1000
2220 IF (BR%(103)=SB%) AND (BR%(91)<>FF%) THEN HW%=HW%+1000
2222 FOR I=3 TO 10
2224 IF BZ%(2,I)=0 THEN GOTO 2290
2228 IF (BZ%(2,I-1)=0)AND(BZ%(2,I+1)=0) THEN HW%=HW%+1500
2232 IF BZ%(2,I)<2 THEN GOTO 2237
2233 IF BZ%(2,I)=2 THEN HW%=HW%+400:GOTO 2237
2234 HW%=HW%+4000
2237 S%=BF%(2,I,1)
2242 IF (BZ%(1,I-1)<>0)AND(BF%(1,I-1,1)<S%-1) THEN GOTO 2260
2244 IF (BZ%(1,I  )<>0)AND(BF%(1,I  ,1)<S%  ) THEN GOTO 2260
2246 IF (BZ%(1,I+1)<>0)AND(BF%(1,I+1,1)<S%  ) THEN GOTO 2260
2248 FB%(2,I)=-1
2250 A1%=-115
2252 IF BR%(S%-12)>FF% THEN A1%=-80
2254 IF (BR%(S%+11)=SB%)OR(BR%(S%+13)=SB%) THEN A1%=A1%-35
2256 IF ES%(1) OR ES%(2) THEN A1%=A1%*2
2257 A2%=10-INT(S%/12)
2258 HW%=HW%+A1%*A2%*A2%
2260 FOR J=1 TO BZ%(2,I)
2262 S%=BF%(2,I,J)
2266 HW%=HW%-(8-INT(S%/12))*BV%(2,I)
2278 IF (BR%(S%-1)=SB%)OR(BR%(S%+11)=SB%) THEN GOTO 2286
2280 IF (BR%(S%+1)=SB%)OR(BR%(S%+13)=SB%) THEN GOTO 2286
2282 IF BR%(S%-25)=WB% THEN RB%(2,I)=RB%(2,I)+200
2284 IF BR%(S%-23)=WB% THEN RB%(2,I)=RB%(2,I)+200
2286 NEXT J
2288 HW%=HW%+RB%(2,I)
```

```
2290 NEXT I
2294 RETURN
2300 REM BEWERTUNG FUER DIE WEISSEN FIGUREN
2302 A1%=0:A2%=1:A3%=-1:A4%=-1:A9%=-1
2304 A5%=1000:IF ES%(1) OR ES%(2) THEN A5%=1500
2308 FOR I=2 TO FZ%(1)
2310 S%=FS%(1,I)
2312 ON (FI%(1,I)-FF%) GOTO 2360,2317,2329,2337,2357
2314 GOTO 2360
2316 REM SPRINGER
2317 A9%=A9%+1
2318 IF S%<39 THEN A1%=A1%+1:HW%=HW%-470
2320 A6%=INT(S%/12)
2322 A6%=ABS(6.5-(S%-12*A6%))+ABS(5.5-A6%)
2324 HW%=HW%+60*(6-2*A6%)
2326 GOTO 2360
2328 REM LAEUFER
2329 A9%=A9%+1
2330 IF S%<39 THEN A1%=A1%+1:HW%=HW%-550
2332 IF A4% THEN A4%=0:GOTO 2360
2334 HW%=HW%+400:GOTO 2360
2336 REM TUERME
2337 A9%=A9%+1
2338 IF (S%<99)OR(S%>106) THEN GOTO 2342
2340 HW%=HW%+500:IF FS%(2,1)>106 THEN HW%=HW%+600
2342 A6%=S%-12*INT(S%/12)
2344 IF BZ%(1,A6%)>0 THEN GOTO 2348
2346 HW%=HW%+150:IF BZ%(2,A6%)=0 THEN HW%=HW%+250
2348 IF A3%<0 THEN A3%=A6%:GOTO 2352
2350 IF A6%=A3% THEN HW%=HW%+800
2352 IF FB%(1,A6%) OR FB%(2,A6%) THEN HW%=HW%+A5%
2354 GOTO 2360
2356 REM DAMEN
2357 A9%=A9%+3
2358 A2%=S%
2360 NEXT I
2362 IF (A1%>0)AND(A2%>46) THEN HW%=HW%-250*A1%
2364 RF%(1)=A9%
2400 REM BEWERTUNG FUER DIE SCHWARZEN FIGUREN (ANALOG ZU 2302-2364)
2402 A1%=0:A2%=144:A3%=-1:A4%=-1:A9%=1
```

```
2404 A5%=-1000:IF ES%(1) OR ES%(2) THEN A5%=-1500
2408 FOR I=2 TO FZ%(2)
2410 S%=FS%(2,I)
2412 ON (FF%-FI%(2,I)) GOTO 2460,2417,2429,2437,2457
2414 GOTO 2460
2417 A9%=A9%-1
2418 IF S%>106 THEN A1%=A1%+1:HW%=HW%+470
2420 A6%=INT(S%/12)
2422 A6%=ABS(6.5-(S%-12*A6%))+ABS(5.5-A6%)
2424 HW%=HW%-60*(6-2*A6%)
2426 GOTO 2460
2429 A9%=A9%-1
2430 IF S%>106 THEN A1%=A1%+1:HW%=HW%+550
2432 IF A4% THEN A4%=0:GOTO 2460
2434 HW%=HW%-400:GOTO 2460
2437 A9%=A9%-1
2438 IF (S%>46)OR(S%<39) THEN GOTO 2442
2440 HW%=HW%-500:IF FS%(1,1)<39 THEN HW%=HW%-600
2442 A6%=S%-12*INT(S%/12)
2444 IF BZ%(2,A6%)>0 THEN GOTO 2448
2446 HW%=HW%-150:IF BZ%(1,A6%)=0 THEN HW%=HW%-250
2448 IF A3%<0 THEN A3%=A6%:GOTO 2452
2450 IF A6%=A3% THEN HW%=HW%-800
2452 IF FB%(1,A6%) OR FB%(2,A6%) THEN HW%=HW%+A5%
2454 GOTO 2460
2457 A9%=A9%-3
2458 A2%=S%
2460 NEXT I
2462 IF (A1%>0)AND(A2%<99) THEN HW%=HW%+250*A1%
2464 RF%(2)=A9%
2466 RETURN
2500 REM BEWERTUNG DER KOENIGSSTANDORTE
2504 A6%=24:A7%=36:SF%=WB%
2510 REM FUER WEISS UND SCHWARZ GEMEINSAMER BEWERTUNGSTEIL
2512 FOR I=1 TO 2:REM ORGANISIERT ALS SCHLEIFE
2514 S%=FS%(I,1):A2%=INT(S%/12):A1%=S%-12*A2%:A3%=7-A1%
2516 IF ES%(I) THEN GOTO 2598
2518 REM EROEFFNUNGS- UND MITTELSPIELPOSITIONEN (-> KOENIGSSICHERHEIT)
2522 A4%=FS%(3-I,1):A5%=A4%-12*INT(A4%/12)
2524 IF (ABS(A3%)>1)OR(ABS(7-A5%)<=1) THEN KW%=0:GOTO 2532
```

```
2526 IF (BZ%(1,A1%)>0)OR(BZ%(2,A1%)>0) THEN KW%=100:GOTO 2532
2528 KW%=200
2530 REM EXPONIERTE KOENIGSSTELLUNG
2532 IF ABS(A2%-INT(A6%/12))>1 THEN HW%=HW%+RF%(3-I)*(KW%+2000):GOTO
2632
2534 A8%=1000:A9%=1000
2536 IF A3%<0 THEN GOTO 2558
2538 REM UNTERSUCHUNG DES DAMENFLUEGELS
2540 A8%=0
2542 FOR J=3 TO 5:REM BAUERNSCHUTZ
2544 IF BR%(A7%+J)=SF% THEN GOTO 2550
2546 IF BZ%(I,J)>0 THEN A8%=A8%+50:GOTO 2550
2548 A8%=A8%+100
2550 NEXT J
2552 IF A1%<=5 THEN HW%=HW%+RF%(3-I)*A8%:GOTO 2632
2553 IF RO%(I,1) THEN A8%=A8%+50:GOTO 2555
2554 A8%=A8%+200
2555 FOR J=4 TO 6:REM ENTWICKLUNGSDAUER
2556 IF BR%(A6%+J)<>FF% THEN A8%=A8%+50
2557 NEXT J
2558 IF A3%>0 THEN GOTO 2584
2560 REM UNTERSUCHUNG DES KOENIGSFLUEGELS
2562 A9%=0
2564 FOR J=8 TO 10
2566 IF BR%(A7%+J)=SF% THEN GOTO 2572
2568 IF BZ%(I,J)>0 THEN A9%=A9%+50:GOTO 2572
2570 A9%=A9%+100
2572 NEXT J
2574 IF A1%>=9 THEN HW%=HW%+RF%(3-I)*A9%:GOTO 2632
2575 IF RO%(I,2) THEN A9%=A9%+50:GOTO 2577
2576 A9%=A9%+200
2577 FOR J=8 TO 9
2578 IF BR%(A6%+J)<>FF% THEN A9%=A9%+50
2579 NEXT J
2580 REM AUSWAHL EINER BRETTHAELFTE ZUR BEWERTUNG
2584 IF A8%<A9% THEN A9%=A8%
2585 HW%=HW%+RF%(3-I)*(KW%+A9%)
2590 GOTO 2632
2592 REM ENDSPIELPOSITIONEN (-> KOENIGSAKTIVITAET)
2596 REM ZENTRALISIERUNG
```

```
2598 KW%=400*(ABS(6.5-A1%)+ABS(5.5-A2%))
2601 REM ABSTAND ZU EIGENEN UND GEGNERISCHEN BAUERN
2602 A6%=0:A7%=0:A9%=WB%
2604 FOR J=1 TO 2
2606 FOR K=2 TO FZ%(J)
2608 IF FI%(J,K)<>A9% THEN GOTO 2624
2610 ZF%=FS%(J,K):A5%=INT(ZF%/12):A4%=ZF%-12*A5%
2612 IF FB%(J,A4%) THEN A8%=6:GOTO 2618
2614 IF RB%(J,A4%)<>0 THEN A8%=3:GOTO 2618
2616 A8%=2
2618 A4%=ABS(A1%-A4%):A5%=ABS(A2%-A5%)
2620 A6%=A6%+A8%
2622 A8%=A8%*(A4%+A5%)
2623 A7%=A7%+A8%
2624 NEXT K
2626 A9%=SB%
2628 NEXT J
2630 IF A6%>0 THEN KW%=KW%+600*(A7%/A6%-6)
2631 HW%=HW%+(FF%-SF%)*KW%
2632 A6%=108:A7%=96:SF%=SB%
2634 NEXT I:REM ENDE DER BEWERTUNG FUER EINE FARBE
2636 RETURN
3000 REM STELLUNGSANALYSE
3012 REM ENDSPIELABFRAGEN
3014 ES%(1)=(MS%(2)>-2000)
3016 ES%(2)=(MS%(1)< 2000)
3018 REM BESTIMMUNG DER BAUERNSTANDORTE
3020 FOR I=1 TO 2
3022 FOR J=1 TO 11
3024 BZ%(I,J)=0
3026 NEXT J
3028 NEXT I
3030 FOR I=36 TO 96
3032 FOR J=3 TO 10
3034 IF BR%(I+J)<>WB% THEN GOTO 3040
3036 BZ%(1,1)=BZ%(1,1)+1:BZ%(1,J)=BZ%(1,J)+1:BF%(1,J,BZ%(1,J))=I+J
3038 GOTO 3044
3040 IF BR%(I+J)<>SB% THEN GOTO 3044
3042 BZ%(2,1)=BZ%(2,1)+1:BZ%(2,J)=BZ%(2,J)+1:BF%(2,J,BZ%(2,J))=I+J
3044 NEXT J
```

```
3046 I=I+11:NEXT I
3048 REM BESTIMMUNG DES STELLUNGSTYPS
3050 IF NOT(ES%(1) OR ES%(2)) THEN GOTO 3068
3052 IF (BZ%(1,1)+BZ%(2,1))>0 THEN GOTO 3068
3054 IF MB%<500 THEN GOTO 3060
3056 IF MS%(2)<-650 THEN GOTO 3068
3058 PT%=3:GOTO 3082:REM MATTFUEHRUNG GEGEN SCHWARZ
3060 IF MB%>-500 THEN GOTO 3066
3062 IF MS%(1)>650 THEN GOTO 3068
3064 PT%=1:GOTO 3082:REM MATTFUEHRUNG GEGEN WEISS
3066 IF (MS%(1)<=325)AND(MS%(2)>=-325) THEN PT%=2:GOTO 3082:REM REMIS
3068 PT%=4:REM NORMALE BEWERTUNG
3082 RETURN
3100 REM MATTSETZEN DES SCHWARZEN KOENIGS
3102 A2%=INT(FS%(1,1)/12):A1%=FS%(1,1)-12*A2%
3104 A9%=INT(FS%(2,1)/12):A8%=FS%(2,1)-12*A9%
3106 REM ZENTRALISIERUNG DES SCHWARZEN KOENIGS
3108 HW%=94*(ABS(6.5-A8%)+ABS(5.5-A9%))
3110 REM ABSTAND VON KOENIG UND SPRINGERN (WEISS) ZUM SCHWARZEN KOENIG
3112 A3%=2
3114 A5%=ABS(A1%-A8%):A6%=ABS(A2%-A9%)
3116 HW%=HW%+16*(14-(A5%+A6%))
3118 FOR I=A3% TO FZ%(1)
3120 IF FI%(1,I)<>WS% THEN GOTO 3128
3122 A3%=I+1
3124 A2%=INT(FS%(1,I)/12):A1%=FS%(1,I)-12*A2%
3126 GOTO 3114
3128 NEXT I
3130 REM GESAMTBEWERTUNG DER STELLUNG
3132 PW%=MB%+INT(HW%/10)
3134 RETURN
3200 REM MATTSETZEN DES WEISSEN KOENIGS (ANALOG ZU 3100-3134)
3202 A2%=INT(FS%(2,1)/12):A1%=FS%(2,1)-12*A2%
3204 A9%=INT(FS%(1,1)/12):A8%=FS%(1,1)-12*A9%
3208 HW%=94*(ABS(6.5-A8%)+ABS(5.5-A9%))
3212 A3%=2
3214 A5%=ABS(A1%-A8%):A6%=ABS(A2%-A9%)
3216 HW%=HW%+16*(14-(A5%+A6%))
3218 FOR I=A3% TO FZ%(2)
3220 IF FI%(2,I)<>SS% THEN GOTO 3228
```

```
3222 A3%=I+1
3224 A2%=INT(FS%(2,I)/12):A1%=FS%(2,I)-12*A2%
3226 GOTO 3214
3228 NEXT I
3232 PW%=MB%-INT(HW%/10)
3234 RETURN
3999 REM@P
4000 REM STELLUNG IM STACK RETTEN
4004 ST%(T%,1)=ZA%: ST%(T%,8)=NF%
4006 ST%(T%,2)=S%
4008 ST%(T%,3)=Z%
4010 ST%(T%,4)=ZF%
4012 ST%(T%,5)=F%
4014 ST%(T%,6)=FA%(S%)
4016 ON ZA%-19GOTO 4050,4022,4030,4018,4040
4018 REM SCHLAGZUG
4020 ST%(T%,7)=FA%(Z%): GOTO 4050
4022 REM ROCHADE
4024 IF Z%>S% THEN A1%=Z%+1:GOTO4028
4026 A1%=Z%-2
4028 ST%(T%,7) = A1%:GOTO 4050
4030 REM EN-PASSANT
4032 IF ZF%>FF% THEN A1%=Z%-12:GOTO4036
4034 A1%=Z%+12
4036 ST%(T%,7)=FA%(A1%)
4038 GOTO 4050
4040 REM UMWANDLUNG
4042 ST%(T%,7)=FA%(Z%)
4044 REM ENDE BRETT RETTEN
4050 ST%(T%,11)=I1%: ST%(T%,12)=I2%
4052 ST%(T%,13)=ZV%(1):ST%(T%,14)=ZV%(2)
4054 ST%(T%,15)=ZV%(3):ST%(T%,16)=ZV%(4)
4056 ST%(T%,17)=ZV%(5): ST%(T%,18)=ZV%(6)
4100 ST%(T%,20)=RO%(1,1):ST%(T%,21)=RO%(1,2)
4102 ST%(T%,22)=RO%(2,1):ST%(T%,23)=RO%(2,2)
4104 ST%(T%,24)=EP%
4120 REM FUER DIE SUCHE
4122 ST%(T%,30)=VS%
4124 ST%(T%,31)=MW%
4126 ST%(T%,32)=SH%
```

```
4128 ST%(T%,33)=GZ%
4130 ST%(T%,34)=AL%
4132 ST%(T%,35)=BE%
4134 ST%(T%,36)=MT%
4140 REM FUER DIE BEWERTUNG
4142 ST%(T%,40)=PW%
4144 ST%(T%,41)=MS%(1)
4146 ST%(T%,42)=MS%(2)
4148 ST%(T%,43)=MB%
4198 RETURN
4300 REM VERTIEFEN
4302 A1%=2+W%
4304 A2%=FA%(S%)
4306 BR%(S%)=FF%:FA%(S%)=0
4308 IF BR%(Z%)=FF% THEN GOTO 4316
4310 A3%=FA%(Z%)
4311 MS%(3-A1%)=MS%(3-A1%)-FW%(BR%(Z%))
4312 FI%(3-A1%,A3%)=FF%
4314 FS%(3-A1%,A3%)=0
4316 IF ZA%<>22 THEN GOTO 4330
4322 A5%=FA%(EP%)
4323 MS%(3-A1%)=MS%(3-A1%)-FW%(BR%(EP%))
4324 BR%(EP%)=FF%:FA%(EP%)=0
4326 FI%(3-A1%,A5%)=FF%
4328 FS%(3-A1%,A5%)=0
4330 BR%(Z%)=ZF%
4332 FA%(Z%)=A2%
4334 FS%(A1%,A2%)=Z%
4335 IF ZA%<>24 THEN GOTO 4338
4336 MS%(A1%)=MS%(A1%)+FW%(NF%)-FW%(BR%(Z%)):BR%(Z%)=NF%:FI%(A1%,A2%)
=NF%
4338 MB%=MB%+ZG%
4339 IF ZA%<>21 THEN GOTO 4360
4340 A3%=Z%-1:A4%=Z%+1
4341 IF Z%<S% THEN A3%=Z%+1: A4%=Z%-2
4342 BR%(A3%) = BR%(A4%): BR%(A4%)=FF%
4344 FA%(A3%) = FA%(A4%): FA%(A4%)=0
4346 FS%(A1%,FA%(A3%))=A3%
4348 RO%(A1%,1)=0: RO%(A1%,2)=0
4360 IF T%>3 OR NOT GZ% THEN GZ%=0:GOTO4362
```

```
4361 GZ%=(HV%(1,T%,1)=S%)AND(HV%(1,T%,2)=Z%)AND(HV%(1,T%+1,1)<>0)
4362 IF T%<4 THEN HV%(T%+1,T%+1,1)=0
4364 IF (ZV%(5)=Z%)AND(ZV%(6)>=22) THEN MT%=MT%+1
4366 ZV%(5)=Z%: ZV%(6)=ZA%
4370 EP%=0
4372 IF ((ZF%=WB%)OR(ZF%=SB%))AND(ABS(Z%-S%)=24) THEN EP%=Z%
4380 IF NOT W% THEN GOTO 4390:REM ROCHADEN
4382 IF ZF%=WK% THEN RO%(1,1)=0:RO%(1,2)=0:GOTO 4387
4384 IF ZF%<>WT% THEN GOTO 4387
4385 IF S%=27 THEN RO%(1,1)=0
4386 IF S%=34 THEN RO%(1,2)=0
4387 IFF%=ST%ANDZ%=111THENRO%(2,1)=0
4388 IFF%=ST%ANDZ%=118THENRO%(2,2)=0
4389 GOTO 4400
4390 IF ZF%=SK% THEN RO%(2,1)=0:RO%(2,2)=0:GOTO 4397
4394 IF ZF%<>ST% THEN GOTO 4397
4395 IF S%=111THEN RO%(2,1)=0
4396 IF S%=118THEN RO%(2,2)=0
4397 IFF%=WT%ANDZ%=27 THENRO%(1,1)=0
4398 IFF%=WT%ANDZ%=34 THENRO%(1,2)=0
4400 REM ROCHADERECHTE FERTIG
4460 T%=T%+1
4462 W%=NOT W%
4480 IF NOT DR% THEN GOTO 4496
4481 PRINT LEFT$(BL$,T%);
4482 A0%(1)=S%:A0%(2)=Z%
4484 GOSUB 11600
4494 PRINT "(";AL%;BE%;")"
4496 SI%(3)=SI%(3)+1
4498 RETURN
4500 REM ZUG RUECKGAENGIG MACHEN
4502 W%= NOT W%
4504 T%=T%-1
4506 EP%=ST%(T%,24)
4510 ZA%=ST%(T%,1): NF%=ST%(T%,8)
4512 S%=ST%(T%,2)
4514 Z%=ST%(T%,3)
4516 ZF%=ST%(T%,4)
4518 F%=ST%(T%,5)
4520 FA%(S%)=ST%(T%,6)
```

```
4522 A1%=ST%(T%,7)
4530 BR%(S%)=ZF%
4532 FS%(W%+2,FA%(S%))=S%
4534 IF ZA%=24 THEN FI%(W%+2,FA%(S%))=ZF%
4536 BR%(Z%)=F%
4538 IF F%=FF% THEN GOTO 4546
4540 FA%(Z%)=A1%
4542 FI%(1-W%,A1%)=F%
4544 FS%(1-W%,A1%)=Z%
4546 IF ZA%<>22 THEN GOTO 4560
4548 A2% = WB%
4550 IF W% THEN A2%=SB%
4552 FA%(EP%)=A1%
4554 FI%(1-W%,A1%)=A2%
4556 FS%(1-W%,A1%)=EP%
4558 BR%(EP%)=A2%
4560 IF ZA%<>21 THEN GOTO 4598
4562 A3%=Z%+1: A4%=Z%-1
4564 IF Z%<S% THEN A3%=Z%-2: A4%=Z%+1
4566 BR%(A3%)=BR%(A4%): BR%(A4%)=FF%
4568 FA%(A3%)=FA%(A4%): FA%(A4%)=0
4570 FS%(W%+2,FA%(A3%))=A3%
4598 REM ENDE BRETT UND MATERIAL
4600 I1%=ST%(T%,11): I2%=ST%(T%,12)
4602 ZV%(1)=ST%(T%,13):ZV%(2)=ST%(T%,14)
4603 ZV%(3)=ST%(T%,15):ZV%(4)=ST%(T%,16)
4604 ZV%(5)=ST%(T%,17): ZV%(6)=ST%(T%,18)
4610 RO%(1,1)=ST%(T%,20):RO%(1,2)=ST%(T%,21)
4612 RO%(2,1)=ST%(T%,22):RO%(2,2)=ST%(T%,23)
4620 REM FUER DIE SUCHE
4622 VS%=ST%(T%,30)
4624 MW%=ST%(T%,31)
4626 SH%=ST%(T%,32)
4628 GZ%=ST%(T%,33)
4630 AL%=ST%(T%,34)
4632 BE%=ST%(T%,35)
4634 MT%=ST%(T%,36)
4640 REM FUER DIE BEWERTUNG
4642 PW%=ST%(T%,40)
4644 MS%(1)=ST%(T%,41)
```

```
4646 MS%(2)=ST%(T%,42)
4648 MB%=ST%(T%,43)
4698 RETURN
5000 REM HILFSROUTINEN DER SUCHE
5100 REM DER NAECHSTE ZUG
5104 IF NOT ((ZV%(4)<6) OR VS%) THEN S%=0: GOTO 5290
5106 ON ZV%(4) GOTO 5110,5130,5190,5150,5150,5170
5110 REM GEMERKTER BESTER ZUG
5112 IF NOT GZ% THEN ZV%(4)=2: GOTO5104
5114 S%=HV%(1,T%,1)
5116 Z%=HV%(1,T%,2)
5117 NF%=HV%(1,T%,3)
5118 GOSUB 5300
5120 ZV%(4)=2
5122 IF NOT A1% THEN GOTO 5104
5124 GOTO 5290
5130 REM SCHLAGZUEGE, BESONDERE
5132 A1%=ZV%(2)+1
5134 IF A1%>I2% THEN ZV%(4)=4:GOTO 5104
5136 S%=Z2%(A1%,1):Z%=Z2%(A1%,2):NF%=Z2%(A1%,3)
5138 ZV%(2)=A1%
5140 IF S%=0ORZ%<>ZV%(5)THEN GOTO 5132
5142 Z2%(A1%,1)=0: GOTO 5290
5150 REM KILLERZUG
5152 IF ZV%(4)=4 THEN A1%=1:ZV%(4)=5:GOTO 5154
5153 A1%=3:ZV%(4)=3
5154 S%=KI%(T%,A1%)
5155 IF S%=0 THEN GOTO 5104
5156 Z%=KI%(T%,A1%+1)
5160 GOSUB 5300
5162 IF NOT A1% THEN GOTO 5104
5166 GOTO 5290
5170 REM RUHIGE ZUEGE
5172 A1%=ZV%(1)+1
5174 IF A1%>I1% THEN S%=0: GOTO 5290
5176 S%=Z1%(A1%,1):Z%=Z1%(A1%,2)
5178 ZV%(1)=A1%
5180 IF S%=0 THEN GOTO 5172
5182 GOTO 5290
5190 REM SCHLAGZUEGE, NORMALE
```

```
5192 A1%=ZV%(3)+1
5194 IF A1%>12% THEN ZV%(4)=6:GOTO 5104
5196 S%=Z2%(A1%,1):Z%=Z2%(A1%,2):NF%=Z2%(A1%,3)
5198 ZV%(3)=A1%
5200 IF S%=0ORZ%=ZV%(5)THEN GOTO 5192
5202 GOTO 5290
5290 RETURN
5300 REM SPEZIALZUG STREICHEN
5310 REM A) SCHLAGZUG
5312 A1%=0
5314 IF ZV%(3)>=I2% THEN GOTO 5340
5316 FOR J=ZV%(3)+1 TO I2%
5318 IF Z2%(J,1)<>S% THEN GOTO 5338
5320 IF Z2%(J,2)<>Z% THEN GOTO 5338
5322 IF (BR%(S%)<>WB%)OR(Z%<111) THEN GOTO 5328
5324 IF NF%=Z2%(J,3) THEN GOTO 5332
5326 GOTO 5338
5328 IF (BR%(S%)=SB%)AND(Z%<39) THEN GOTO 5324
5330 NF%=Z2%(J,3)
5332 A1%=-1
5334 Z2%(J,1)=0
5336 GOTO 5370
5338 NEXT J
5340 REM B) RUHIGER ZUG
5342 A1%=0: IF NOT VS% THEN GOTO 5370
5344 IF ZV%(1)>=I1% THEN GOTO 5370
5346 FOR J=ZV%(1)+1 TO I1%
5348 IF Z1%(J,1)<>S% THEN GOTO 5358
5350 IF Z1%(J,2)<>Z% THEN GOTO 5358
5352 A1%=-1
5354 Z1%(J,1)=0
5356 GOTO 5370
5358 NEXT J
5370 RETURN
5400 REM NEUEN BESTEN ZUG MERKEN
5410 IF T%>=4 THEN GOTO 5424
5412 FOR J=T%+1 TO 4
5414 HV%(T%,J,1)=HV%(T%+1,J,1)
5416 HV%(T%,J,2)=HV%(T%+1,J,2)
5418 HV%(T%,J,3)=HV%(T%+1,J,3)
```

```
5420 HV%(T%+1,J,1)=0
5422 NEXT J
5424 IF T%>4 THEN GOTO 5440
5426 HV%(T%,T%,1)=S%
5428 HV%(T%,T%,2)=Z%
5430 HV%(T%,T%,3)=NF%
5440 RETURN
5500 REM HV% LOESCHEN
5502 FOR J=1 TO 4
5504 FOR J1=1 TO 4
5506 HV%(J,J1,1)=0
5508 NEXT J1
5510 NEXT J
5512 RETURN
5520 REM KI% INITIALISIEREN
5522 FOR J=1 TO 4
5524 KI%(J,1)=0
5526 NEXT J
5528 RETURN
5600 REM IST DER GEGNERISCHE KOENIG ILLEGAL IM SCHACH?
5602 A1%=0
5604 IF ZV%(2)>=I2% THEN GOTO 5616
5606 A2%=FS%(1-W%,1)
5610 FOR J=ZV%(2)+1 TO I2%
5612 IF Z2%(J,2)=A2% THEN A1%=-1:GOTO 5616
5614 NEXT J
5616 RETURN
5700 REM KILLERZUG EINTRAGEN
5705 IF Z%=ZV%(5) THEN GOTO 5725
5710 IF KI%(T%,1)=S% AND KI%(T%,2)=Z% THEN GOTO 5725
5715 KI%(T%,3)=KI%(T%,1):KI%(T%,4)=KI%(T%,2)
5720 KI%(T%,1)=S%:KI%(T%,2)=Z%
5725 RETURN
6000 REM DIE SUCHE
6100 REM SUCHE FUER WEISS
6105 GOSUB 1900: REM ZUEGE GENERIEREN
6107 GOSUB 5600: REM ILLEGALE STELLUNG?
6108 IF A1% THEN MW%= 16001-T%:GOTO6280
6110 A1%=FS%(1,1):A2%=0:GOSUB 8400
6111 SH%=A1%:REM SCHACH?
```

```
6112 VS%= T%<MT%
6115 IF VS% THEN MW%=-16000+T%:GOTO6135
6120 GOSUB 2000: REM BEWERTUNG
6121 IF DR%THEN PRINTLEFT$(BL$,T%);"BEWERTUNG";PW%
6126 IF SH% THEN MW%=-16000+T%:GOTO6135
6130 MW%=PW%
6135 IF T%>8 THEN GOTO6280
6136 IF MW%<BE% GOTO 6139
6137 IF DR%THEN PRINT LEFT$(BL$,T%);"FPW1"
6138 GOTO 6280
6139 IF MW%>AL% THEN AL%=MW%
6140 GOSUB 5100: REM NAECHSTER ZUG
6145 IF S%=0 THEN GOTO 6230
6150 GOSUB 8005: REM ZUG ANALYSIEREN
6151 IF ZA%=0 THEN GOTO 6220
6152 IF T%<=MT% THEN GOTO 6156
6153 IF PW%+ZG%+80>AL% THEN GOTO 6156
6154 AW%=PW%+ZG%+80: IF DR% THEN PRINT LEFT$(BL$,T%+1);"FPW2"
6155 GOTO 6185
6156 GOSUB 4000: REM STELLUNG RETTEN
6160 GOSUB 4300: REM ZUG VERTIEFEN
6170 GOSUB 6300: REM REKURSIVER AUFRUF
6171 AW%=MW%
6175 GOSUB 4500: REM ZUG ZURUECKNEHMEN
6185 IF AW%<=MW% THEN GOTO 6220
6190 MW%=AW%
6195 IF MW%<= AL% THEN GOTO 6220
6200 GOSUB 5400: REM HAUPTVARIANTE
6205 GOSUB 5700: REM KILLERZUG
6210 IF MW%<BE% THEN GOTO 6215
6211 IF DR% THEN PRINTLEFT$(BL$,T%+1);"CUTOFF -------"
6212 GOTO6260
6215 AL%=MW%
6219 REM EXTRA ZEITABBRUCH
6220 IF T%=1 AND MT%>2 AND SI%(1)<MW% AND SI%(3)>(ZE%+ZE%) THEN GOTO
6260
6225 GOTO 6140
6230 REM SUCHE FERTIG?
6235 IF VS% OR (NOT SH%) THEN GOTO 6260
6236 IF MW%>=PW% THEN GOTO 6260
```

```
6237 IF PW%<=AL% THEN MW%=PW%:GOTO 6260
6238 IF PW%<BE% THEN BE%=PW%
6239 VS%=-1:GOTO 6140:REM SCHACHMATT?
6260 P%=0: REM PATT?
6264 IF MW%=-16000+T% THEN IF NOT SH% THEN MW%=0:P%=-1
6280 IF DR% THEN PRINT LEFT$(BL$,T%);"WERT=";MW%
6290 RETURN
6300 REM SUCHE FUER SCHWARZ (ANALOG ZU 6100-6290)
6305 GOSUB 1900
6307 GOSUB 5600
6308 IF A1% THEN MW%=-16001+T%:GOTO6480
6310 A1%=FS%(2,1):A2%=-1:GOSUB 8400
6311 SH%=A1%
6312 VS%= T%<MT%
6315 IF VS% THEN MW%= 16000-T%:GOTO6335
6320 GOSUB 2000
6321 IF DR% THEN PRINTLEFT$(BL$,T%);"BEWERTUNG";PW%
6326 IF SH% THEN MW%= 16000-T%:GOTO6335
6330 MW%=PW%
6335 IF T%>8 THEN GOTO6480
6336 IF MW%>AL% THEN GOTO 6339
6337 IF DR% THEN PRINT LEFT$(BL$,T%);"FPS1"
6338 GOTO 6480
6339 IF MW%<BE% THEN BE%=MW%
6340 GOSUB 5100
6345 IF S%=0 THEN GOTO 6430
6350 GOSUB 8050
6351 IF ZA%=0 THEN GOTO 6420
6352 IF T%<=MT% THEN GOTO 6356
6353 IF PW%+ZG%-80<BE% THEN GOTO 6356
6354 AW%=PW%+ZG%-80:IF DR% THEN PRINT LEFT$(BL$,T%+1);"FPS2"
6355 GOTO 6385
6356 GOSUB 4000
6360 GOSUB 4300
6370 GOSUB 6100
6371 AW%=MW%
6375 GOSUB 4500
6385 IF AW%>=MW% THEN GOTO 6420
6390 MW%=AW%
6395 IF MW%>= BE% THEN GOTO 6420
```

```
6400 GOSUB 5400
6405 GOSUB 5700
6410 IF MW%>AL% THEN GOTO 6415
6411 IF DR% THEN PRINT LEFT$(BL$,T%+1);"CUTOFF ------"
6412 GOTO 6460
6415 IF MW%<BE% THEN BE%=MW%
6420 IF T%=1 AND MT%>2 AND SI%(2)>MW% AND SI%(3)>(ZE%+ZE%) THEN GOTO
6460
6425 GOTO 6340
6430 REM SUCHE FERTIG?
6435 IF VS% OR (NOT SH%) THEN GOTO 6460
6436 IF MW%<=PW% THEN GOTO 6460
6437 IF PW%>=BE% THEN MW%=PW%:GOTO 6460
6438 IF PW%>AL% THEN AL%=PW%
6439 VS%=-1:GOTO 6340
6460 P%=0
6464 IF MW%= 16000-T% THEN IF NOT SH% THEN MW%=0:P%=-1
6480 IF DR% THEN PRINT LEFT$(BL$,T%);"WERT=";MW%
6490 RETURN
6800 REM AUFRUF DER SUCHE
6810 GOSUB 5500: REM HV% INITIALISIEREN
6812 GOSUB 5520: REM KI% INITIALISIEREN
6816 MT%=1
6817 AL%=-15000: BE%=15000
6818 INPUT "TESTAUSDRUCKE";ES$: DR%=ES$="J"
6820 IF SI%(3)>ZE% AND HV%(1,1,1)<>0 THEN GOTO 6960
6830 TI$="000000"
6835 T%=1: GZ%=HV%(1,1,1)<>0
6836 SI%(1)=AL%: SI%(2)=BE%: SI%(3)=0
6837 I1%=0: I2%=0
6840 IF W% THEN GOTO 6850
6845 GOSUB 6300: GOTO 6855
6850 GOSUB 6100
6855 GOSUB 11800
6856 PRINT SI%(3);"STELLUNGEN UNTERSUCHT,ZEIT=";TI$
6859 IF MW%>15000 OR MW%<-15000 OR P% THEN GOTO 6960
6860 IF (MW%>SI%(1)) AND (MW%<SI%(2)) THEN GOTO 6900
6865 PRINT "SUCHE WIEDERHOLT"
6870 AL%=-15000: BE%=15000
6875 GOTO 6835
```

```
6900 AL%=MW%-100: BE%=MW%+100
6950 MT%=MT%+1
6955 GOTO 6820
6960 REM SUCHE FERTIG
6998 RETURN
8000 REM BESTIMMUNG DES ZUGES
8005 REM ZUG VON WEISS
8010 ZF%=BR%(S%):F%=BR%(Z%):ZG%=-FW%(F%)
8012 IF ZF%<>WB% THEN GOTO 8022
8014 IF Z%>110 THEN ZA%=24:ZG%=ZG%+FW%(NF%)-FW%(WB%):GOTO 8036
8016 IF F%<>FF% THEN GOTO 8034
8018 IF Z%-S%=11 OR Z%-S%=13 THEN ZA%=22:NF%=FA%(Z%-12):ZG%=-FW%(SB%)
:GOTO 8036
8020 GOTO 8032
8022 IF ZF%<>WK% THEN GOTO 8032
8023 IF ABS(Z%-S%)<>2 THEN GOTO 8032
8024 ZA%=21:A2%=0
8025 A1%=31: GOSUB8400: IF A1% THEN ZA%=0: GOTO 8036
8026 IF Z%<S% THEN GOTO 8030
8027 NF%=FA%(34)
8028 A1%=32: GOSUB 8400: IF A1% THEN ZA%=0
8029 GOTO 8036
8030 A1%=30:GOSUB8400:IFA1%THENZA%=0:GOTO8036
8031 NF%=FA%(27): GOTO8036
8032 IF F%=FF% THEN ZA%=20:GOTO 8036
8034 ZA%=23
8036 RETURN
8050 REM ZUG VON SCHWARZ
8060 ZF%=BR%(S%):F%=BR%(Z%):ZG%=-FW%(F%)
8062 IF ZF%<>SB% THEN GOTO 8072
8064 IF Z%< 35 THEN ZA%=24:ZG%=ZG%+FW%(NF%)-FW%(SB%):GOTO 8086
8066 IF F%<>FF% THEN GOTO 8084
8068 IF S%-Z%=11 OR S%-Z%=13 THEN ZA%=22:NF%=FA%(Z%+12):ZG%=-FW%(WB%)
:GOTO 8086
8070 GOTO 8082
8072 IF ZF%<>SK% THEN GOTO 8082
8073 IF ABS(Z%-S%)<>2 THEN GOTO 8082
8074 ZA%=21:A2%=-1
8075 A1%=115: GOSUB8400: IF A1% THEN ZA%=0: GOTO 8086
8076 IF Z%<S% THEN GOTO 8080
```

```
8077 NF%=FA%(118)
8078 A1%=116: GOSUB 8400: IF A1% THEN ZA%=0
8079 GOTO 8086
8080 A1%=114:GOSUB8400:IFA1%THENZA%=0:GOTO8086
8081 NF%=FA%(111): GOTO8086
8082 IF F%=FF% THEN ZA%=20:GOTO 8086
8084 ZA%=23
8086 RETURN
8400 REM KOENIG IM SCHACH?
8402 REM@M
8404 IF NOT A2% THEN GOTO 8408
8406 IF BR%(A1%-11)=WB% OR BR%(A1%-13)=WB% THEN GOTO 8440
8407 GOTO 8410
8408 IF BR%(A1%+11)=SB% OR BR%(A1%+13)=SB% THEN GOTO 8440
8410 A3%=SS%: IF A2% THEN A3%=WS%
8412 FOR J=1 TO 8
8414 IFBR%(A1%+SW%(J))=A3%THENGOTO8440
8416 NEXT J
8418 IF A2%THEN A4%=WD%:A5%=WL%:A6%=WT%:GOTO 8422
8420 A4%=SD%:A5%=SL%:A6%=ST%
8422 FOR J=1 TO 8
8424 A3%=A1%
8426 A3%=A3%+DW%(J): A7%=BR%(A3%)
8428 IF A7%=FF% THEN GOTO 8426
8430 IF A7%=A4% THEN GOTO 8440
8432 IF J<5 AND A7%=A5% THEN GOTO 8440
8434 IF J>4 AND A7%=A6% THEN GOTO 8440
8436 NEXT J
8438 A1%=0: GOTO 8442
8440 A1%=-1
8442 REM@P
8444 RETURN
9000 REM INITIALISIEREN DER ANFANGSSTELLUNG
9012 REM BRETT UND FIGURENADRESSEN LOESCHEN
9013 FOR I=1 TO 24:BR%(I)=KF%:NEXT I
9014 FOR I=24 TO 108
9015 FOR J= 1 TO  2:BR%(I+J)=KF%:NEXT J
9016 FOR J= 3 TO 10:BR%(I+J)=FF%:NEXT J
9017 FOR J=11 TO 12:BR%(I+J)=KF%:NEXT J
9018 I=I+11:NEXT I
```

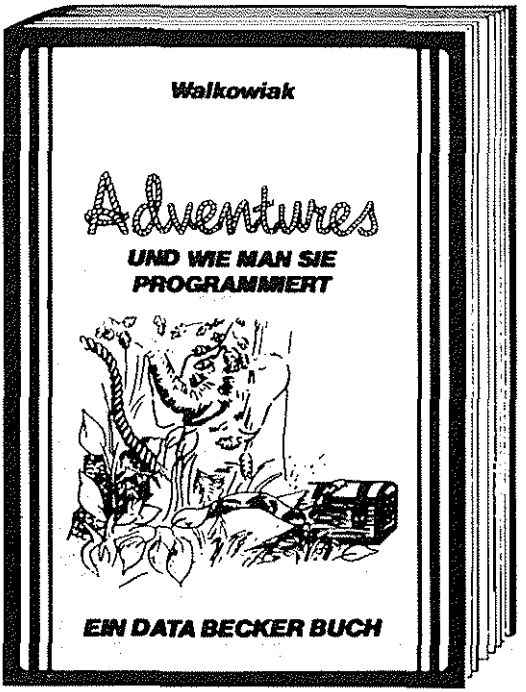
```
9019 FOR I=121 TO 144:BR%(I)=KF%:NEXT I
9020 FOR I=1 TO 118:FA%(I)=0:NEXT I
9021 REM MATERIALLISTEN LOESCHEN
9022 FOR I=1 TO 2
9023 FOR J=1 TO 16:FI%(I,J)=FF%:FS%(I,J)=1:NEXT J
9026 NEXT I
9027 REM STELLUNGSEINGABE
9028 PRINT "WAEHLEN SIE DIE ANFANGSSTELLUNG"
9029 PRINT "G=GRUNDSTELLUNG, SONST BELIEBIG"
9031 ES$="":INPUT ES$
9032 IF ES$<>"G" THEN GOTO 9058
9033 REM ANFANGSSTELLUNG=GRUNDSTELLUNG
9034 FOR I=1 TO 2
9035 FOR J=1 TO 16:READ FI%(I,J),FS%(I,J):NEXT J
9038 FZ%(I)=16
9039 NEXT I
9040 FOR I= 27 TO  34:READ BR%(I),FA%(I):NEXT I
9041 FOR I= 39 TO  46:READ BR%(I),FA%(I):NEXT I
9042 FOR I= 99 TO 106:READ BR%(I),FA%(I):NEXT I
9043 FOR I=111 TO 118:READ BR%(I),FA%(I):NEXT I
9044 MS%(1)=4000:MS%(2)=-4000:MB%=0
9045 W%=-1:EP%=0
9046 FOR I=1 TO 2
9047 FOR J=1 TO 2:RO%(I,J)=-1:NEXT J
9050 NEXT I
9051 GOTO 9202
9056 REM STELLUNGSEINGABE UEBER TASTATUR
9057 REM DATEN WEGLESEN
9058 FOR J=1 TO 128:READ A1%:NEXT J
9062 PRINT "GEBEN SIE DIE FIGURENSTANDORTE EIN"
9065 PRINT "(FELDANGABEN IN BRETTNOTATION (A1-H8))."
9067 PRINT "MIT DER LEEREN EINGABE "
9068 PRINT "(<ENTER> BZW. <RETURN> DRUECKEN)"
9069 PRINT "WECHSELN SIE ZUM NAECHSTEN FIGURENTYP."
9070 REM EINGABE DER FIGUREN
9072 F%=13: A5%=-1
9074 FOR I=1 TO 2
9076 FZ%(I)=0:MS%(I)=0
9078 PRINT MID$(ZK$,(I-1)*8+1,8)
9080 FOR K=2 TO 7
```

```
9082 ES$="":PRINT MID$(ZK$,K*8+1,8);:INPUT ES$
9084 GOSUB 9900
9086 IF S%=0 THEN GOTO 9094
9088 IF S%=1 THEN K=K-1:GOTO 9096
9090 GOSUB 9920
9092 IF (K>2) OR A1% THEN K=K-1:GOTO 9096
9094 F%=F%+A5%
9096 NEXT K
9098 F%=1:A5%=1
9100 NEXT I
9102 MB%=MS%(1)+MS%(2)
9154 REM EINGABE DES ANZUGRECHTS
9155 PRINT"WELCHER SPIELER IST AM ZUG?"
9156 PRINT WS$;:ES$="":INPUT ES$
9159 W%=ES$="W"
9162 REM EINGABE DER ROCHADEMOEGLICHKEITEN
9163 PRINT "WELCHE ROCHADEMOEGLICHKEITEN GIBT ES?"
9164 PRINT "ANTWORTEN SIE AUF EVENTUELLE FRAGEN MIT"
9165 PRINT "M=MOEGLICH, U=UNMOEGLICH"
9167 IF (BR%(31)<>13) OR (BR%(27)<>11) THEN GOTO 9173
9168 INPUT"LANGE ROCHADE VON WEISS";ES$
9169 RO%(1,1)=ES$="M"
9173 IF (BR%(31)<>13) OR (BR%(34)<>11) THEN GOTO 9179
9174 INPUT"KURZE ROCHADE VON WEISS";ES$
9175 RO%(1,2)=ES$="M"
9179 IF (BR%(115)<>1) OR (BR%(111)<>3) THEN GOTO 9185
9180 INPUT "LANGE ROCHADE VON SCHWARZ";ES$
9181 RO%(2,1)=ES$="M"
9185 IF (BR%(115)<>1) OR (BR%(118)<>3) THEN GOTO 9191
9186 INPUT "KURZE ROCHADE VON SCHWARZ";ES$
9187 RO%(2,2)=ES$="M"
9191 EP%=0
9200 REM PARTIEPARAMETER
9202 PRINT "MIT WELCHER FARBE MOECHTEN SIE SPIELEN?"
9204 PRINT WS$;:ES$="":INPUT ES$
9206 GW%=ES$="W"
9214 INPUT "GEBEN SIE BITTE DIE SPIELSTUFE EIN      (1-12)";ZE%
9216 ZE%=3*(2^ZE%)
9298 RETURN:REM ENDE DER STELLUNGSEINGABE
9300 REM EINLESEN EINES ZUGES VOM GEGNER
```

```
9301 I1%=0:I2%=0:T%=1
9302 Z$="":INPUT "GEBEN SIE BITTE IHREN ZUG EIN";Z$
9304 REM TEST AUF KORREKTE EINGABE
9305 IF LEN(Z$)<5 THEN PRINT "FALSCHE EINGABE!":S%=1:GOTO 9362
9306 ES$=MID$(Z$,4,2):GOSUB 9905:IF S%=1 THEN GOTO 9362
9308 A0%(2)=S%
9309 ES$=LEFT$(Z$,2):GOSUB 9905:IF S%=1 THEN GOTO 9362
9310 A0%(1)=S%
9311 IF MID$(Z$,7,1)="" THEN NF%=FF%:GOTO 9336
9312 ES$="DTLS"
9313 IF NOT W% THEN GOTO 9324
9314 IF MID$(Z$,7,1)<>"W" THEN GOTO 9332
9316 FOR I=1 TO 4
9318 IF MID$(Z$,8,1)=MID$(ES$,I,1) THEN NF%=13-I:GOTO 9336
9320 NEXT I
9322 GOTO 9332
9324 IF MID$(Z$,7,1)<>"S" THEN GOTO 9332
9326 FOR I=1 TO 4
9328 IF MID$(Z$,8,1)=MID$(ES$,I,1) THEN NF%=1+I:GOTO 9334
9330 NEXT I
9332 PRINT "FALSCHE FIGURANGABE!":S%=1:GOTO 9362
9334 REM TEST AUF AUSFUEHRBARKEIT (SPIELREGELN)
9336 IF (I1%+I2%)=0 THEN GOSUB 1900
9337 S%=A0%(1):Z%=A0%(2):VS%=-1
9338 GOSUB 5300
9340 IF NOT A1% THEN GOTO 9358
9342 ON (W%+2) GOSUB 8005,8050
9344 IF ZA%=0 THEN GOTO 9358
9346 GOSUB 4000
9348 GOSUB 4300
9350 GOSUB 1900
9352 GOSUB 5600
9354 IF NOT A1% THEN RETURN:REM KORREKTE ZUGEINGABE
9355 REM BEHANDLUNG EINES EINGABEFEHLERS
9356 GOSUB 4500
9358 S%=1:PRINT "ZUG NICHT AUSFUEHRBAR!"
9362 PRINT "MOECHTEN SIE DIE ZUGEINGABE WIEDERHOLEN?"
9364 ES$="":INPUT "(J=JA, SONST BELIEBIG)";ES$
9366 IF LEFT$(ES$,1)<>"J" THEN RETURN
9368 PRINT "BENOETIGEN SIE INFORMATION ZUR EINGABE?"
```

```
9370 ES$="":INPUT "(J=JA, SONST BELIEBIG)";ES$
9372 IF LEFT$(ES$,1)<>"J" THEN GOTO 9302
9373 PRINT
9374 PRINT "SIE MUESSEN URSPRUNGS- UND ZIELFELD"
9376 PRINT "IN BRETTNOTATION (A1-H8) SOWIE"
9378 PRINT "BEI BAUERNUMWANDLUNGEN ZUSAETZLICH"
9380 PRINT "DIE NEUE FIGUR DURCH ENTSPR. ABKUERZUNG"
9382 PRINT "(Z.B. WD=WEISSE DAME) EINGEBEN."
9384 PRINT "DIE FELD- UND FIGURANGABEN SIND JEWEILS"
9386 PRINT "DURCH EIN LEERZEICHEN ZU TRENNEN."
9387 PRINT
9388 GOTO 9302
9900 REM UMWANDLUNG BRETTNOTATION -> INTERNE DARSTELLUNG
9901 IF LEN(ES$)>0 THEN GOTO 9904
9902 IF (F%<>1) AND (F%<>13) THEN S%=0:RETURN
9903 S%=1:PRINT "FEHLENDE FELDANGABE FUER DEN KOENIG!":RETURN
9904 IF LEN(ES$)<>2 THEN GOTO 9912
9905 A1%=3+ASC(LEFT$(ES$,1))-ASC("A")
9906 A2%=2+ASC(MID$(ES$,2,1))-ASC("1")
9908 IF (A1%<3)OR(A1%>10)OR(A2%<2)OR(A2%>9) THEN GOTO 9912
9910 S%=12*A2%+A1%
9911 RETURN
9912 PRINT "FALSCHE FELDANGABE!":S%=1
9913 RETURN
9920 REM FUELLEN VON BRETT UND MATERIALLISTEN
9921 IF FZ%(I)=16 THEN GOTO 9930
9922 IF BR%(S%)<>FF% THEN GOTO 9931
9923 FZ%(I)=FZ%(I)+1
9924 FI%(I,FZ%(I))=F%
9925 FS%(I,FZ%(I))=S%
9926 BR%(S%)=F%
9927 FA%(S%)=FZ%(I)
9928 IF ABS(FF%-F%)<6 THEN MS%(I)=MS%(I)+FW%(F%)
9929 A1%=0:RETURN
9930 PRINT "MAXIMALE FIGURENANZAHL ERREICHT!":GOTO 9932
9931 PRINT "FELD SCHON BESETZT!"
9932 A1%=-1:RETURN
11000 REM TESTAUSDRUCKE
11200 REM DAS BRETT
11203 PRINT"------------------------"
```

```
11205 FOR J= 25TO 109
11206 PRINT "!";
11210 FOR J1=2 TO 9
11215 A1%=BR%(J1+134-J)
11216 GOSUB 11400
11217 PRINT"!";
11220 NEXT J1
11225 PRINT
11226 J=J+11
11233 PRINT"-----------------------"
11235 NEXT J
11240 PRINT
11251 RETURN
11400 REM FIGUR DRUCKEN, A1% IST FIGUR
11402 A2%=2*(A1%-1)+1
11404 PRINT MID$("SKSDSTSLSSSB  WBWSWLWTWDWK",A2%,2);
11406 RETURN
11500 REM FELD DRUCKEN, A1% IST FELD
11502 A2%=INT((A1%-1)/12)-1
11504 A3%=A1%-((A2%+1)*12)-2
11506 PRINT MID$("ABCDEFGH",A3%,1); MID$("12345678",A2%,1);
11508 RETURN
11600 REM ZUGAUSGABE
11602 A1%=A0%(1):GOSUB 11500
11604 PRINT " - ";
11606 A1%=A0%(2):GOSUB 11500
11607 PRINT "  ";
11608 RETURN
11800 REM HAUPTVARIANTE AUSDRUCKEN
11801 PRINT "HAUPTVARIANTE"
11802 FOR J=1 TO 4
11804 IF HV%(1,J,1) =0 THEN GOTO 11816
11806 FOR K=1 TO 2:A0%(K)=HV%(1,J,K):NEXT K
11808 GOSUB 11600
11812 PRINT ",";
11814 NEXT J
11816 PRINT
11818 RETURN
```



Ein faszinierender Führer in die Welt der Abenteuerspiele. Hier läßt sich ein arrivierter Autor in die Karten gucken. Er zeigt, wie Adventures funktionieren und wie man sie programmiert. Der Clou des Buches ist neben fertigen Adventures zum Abtippen ein kompletter ADVENTURE-GENERATOR, mit dem das Selbsterstellen zum Kinderspiel wird. Achtung: dieses Buch macht süchtig!

Walkowiak
Adventures – und wie man sie programmiert
225 Seiten, DM 39,–
ISBN 3-89011-043-6

BASIC leichtgemacht! Das bietet dieser komplette Programmierkurs in der preiswerten Buchform. Mit ihm lernen Sie von Grund auf das Beherrschen der einzelnen Befehle und ihre Anwendungen. Vieles über die Grundlagen des Programmierens, über BIT, BYTE und ASCII-Code, Programmablaufpläne, Unterprogramme und Menuetechniken. Alles was Sie für Ihre zukünftigen Programmiererfolge benötigen.

Kampow
Das BASIC-Trainingsbuch zum Commodore 64
308 Seiten, DM 39,–
ISBN 3-89011-023-1

Künstliche Intelligenz, ein faszinierendes Thema! Eine ausführliche Einführung in deren Theorie und Einsatzmöglichkeiten. Mit historischem Abriß über die „denkenden" und „lebenden" Maschinen und BASIC-Anwendungsbeispielen für den C-64: Expertensystem, Such- und Auskunftsprogramm, selbstlernende Programme, Computer-Kunst und Simulationen. Ihr C-64 ist intelligenter als Sie denken!

Voß
Einführung in die künstliche Intelligenz
395 Seiten, DM 49,–
ISBN 3-89011-081-9

Sie wollten schon immer mal ein Spiel selbst programmieren? Hier ist für Sie das Top-Buch! Zugeschnitten auf den C-64. Schrittweise lernen Sie, wie man Pac Man durchs Labyrinth schleust oder wie Captain Future spannende Abenteuer in fremden Galaxien überlebt. Viele Beispiele, Listings und Tips. Auch mit wenig Programmier-Praxis stellen sich schnell überraschende Erfolge ein!

Linden
Superspiele – selbst gemacht
235 Seiten, DM 29,–
ISBN 3-89011-087-8

LADEN, STARTEN - KLAR!

Sie wollen sich das mühsame Abtippen ersparen? Oder sofort gegen das Schachprogramm spielen? Auf der Diskette zum Buch finden Sie je zwei Versionen des Schachprogramms: das Original-BASIC-Listing und die compilierte, spielbereite Fassung.

Mit der nebenstehenden Post-Zahlkarte einfach die Diskette zum Buch bestellen – individuell für Ihren Rechner: C64/128 und IBM PC und Kompatible.

Bitte unbedingt den Rechnertyp auf der Rückseite der Zahlkarte ankreuzen!

Diskette zum Buch „Das Große Computerschach-Buch“ je DM 39,--

Empfängerabschnitt

DM 39,-- | Pf ---

für Postgirokonto Nr. 789 - 436

Absender (mit Postleitzahl)

Verwendungszweck: Diskette z. Buch 376 117

DM 39,-- | Pf ---

für Postgirokonto Nr. 789 - 436

Absender

Für Vermerke des Absenders

Zahlkarte (Mit Schreibmaschine, Tinte oder Kugelschreiber deutlich ausfüllen)

DM 39,-- | Pf ---

(DM-Betrag in Buchstaben wiederholen) Neununddreißig---

für DATA BECKER GmbH

Merowingerstr. 30

4000 Düsseldorf

Postgiroamt Essen

Postgirokonto Nr. 789 - 436

Postvermerk

Einlieferungsschein – Bitte sorgfältig aufbewahren –

DM 39,-- | Pf ---

für DATA BECKER GmbH

Merowingerstr. 30

4000 Düsseldorf

Postgirokonto Nr. 789 - 436

Postvermerk

Einlieferungsschein

(nicht zu Mitteilungen an den Empfänger benutzen)

Gebühr für die Zahlkarte
(wird bei der Einlieferung bar erhoben)

bis 10 DM ..... 90 Pf
über 10 DM (unbeschränkt) 1,50 DM

Bedienen Sie sich der Vorteile
eines eigenen Postgirokontos!
Auskunft hierüber erteilt jedes Postamt

Feld
für
postdienstliche
Zwecke

Für Mitteilungen an den Empfänger

Diskette zum Buch
„Das große Computerschach-Buch"
376 117
für folgende Rechner:

- ○ IBM-PC und Kompatible
- ○ Commodore 64/128

DAS STEHT DRIN:

Wie funktioniert eigentlich ein Schachprogramm? Die wenigsten wissen, mit welchen Algorithmen man dem Computer das königliche Spiel beibringen kann. Dieses Buch informiert über alle Aspekte der Schachprogrammierung und enthält ein komplettes Schachprogramm in BASIC, das auf anschauliche Weise die Probleme der Programmierung und ihre Lösungen darstellt. Abgerundet wird das Buch durch eine kleine Geschichte des Computerschach und eine Menge Tips zum Thema: Wie spielt man Schach gegen den Computer?

Aus dem Inhalt:

- Programme, Partien und Personen
- Strategiespiele auf dem Computer
- Stack und Rekursion in BASIC
- Brettdarstellung und Zuggenerierung
- Suchalgorithmen in Schachprogrammen
- Bewertungsfunktionen
- Komplettes Schachprogramm in BASIC
- Ausführliche Dokumentation zum Programm
- Testverfahren für Schachprogramme
- 7 goldene Regeln zum Spiel gegen Computer

UND GESCHRIEBEN HABEN DIESES BUCH:

Rainer Bartel ist Fachjournalist und kam über das Thema Schachcomputer zur EDV; er hat sich darauf spezialisiert, Schachprogramme zu testen und gegen sie zu gewinnen. Hans-Joachim Kraas ist Informatiker und aktiver Schachspieler. Sein Spezialgebiet sind die Bewertungsfunktionen, die letztlich über die Spielstärke eines Programmes entscheiden. Zusammen mit Günther Schrüfer, ebenfalls Informatiker und aktiver Schachspieler, hat er das Großrechner-Schachprogramm BOBBY entwickelt, das mit beachtlichen Partien an der Computerschachweltmeisterschaft 1983 in New York teilnahm. Günther Schrüfer ist einer der besten Kenner der Algorithmen von Schachprogrammen.

ISBN 3-89011-117-3