Diplomarbeit
Institut für Informationsverarbeitung

Betreuer: Ao. Univ. Prof. Dr. Volkmar Haase
Autor: Herwig Rehatschek
Graz im Juli 1992

Inhaltsverzeichnis:

0. Vorwort

Schon als ich in die Mittelschule ging, begann in mir ein Traum zu reifen, der Traum einmal ein Programm zu schreiben, das Schachspielen kann. Schach lernte ich schon mit etwa 8 Jahren, damals wurden mir aber nur die Regeln (und von denen nicht alle!) erklärt, keinerlei Strategie und mein Onkel freute sich immer diebisch, wenn er mich besiegen konnte. Doch schon bald kaufte ich mir einen kleinen Schachcomputer, dessen Spielstärke zwar in Hinblick auf die hohen Anforderungen in meiner Arbeit unter jeder Kritik steht, für mich reichte es damals aber. Vor allem lernte ich auch die grundsätzlichen Regeln wie En-Passant und wann man denn nun rochieren darf, und wann nicht. Auch so einfache Dinge wie "Gabeln" und "Fesselungen" lernte ich durch dieses Gerät kennen. Und schon bald schlug ich meinen Onkel - jetzt spielt er übrigens keine einzige Partie mehr mit mir. Im Jahre 1984 kaufte ich mir dann meinen ersten Homecomputer, den damals so berühmten C-64. Schachprogramme ließen nicht lange auf sich warten - und sie waren um einiges stärker, als mein kleiner Schachcomputer (dies ist insofern verwunderlich, daß mein Schachcomputer nur auf das Spielen ausgelegt ist, während sich der C-64 auch noch um die Graphik kümmern mußte). Doch an den Programmen störten mich einige Dinge immer wieder. U.a. gab es z.B. kein einziges (!) Programm, daß bei einer sich schon abzeichnenden Niederlage aufgab (das konnte sogar mein kleiner Schachcomputer). Auch die positionelle Bewertung schien mir an manchen Stellen sehr fehlerhaft und schon damals begann sich der Wunsch zu verfestigen, einmal ein eigenes Programm zu schreiben, das die obigen Nachteile nicht aufweisen sollte und das in seinen Spieleigenschaften so variabel wie möglich sein sollte (d.h. möglichst viele, das Spielverhalten des Programms beeinflussende Parameter sollten vom Benutzer veränderbar und frei wählbar sein). Doch erst jetzt, kurz vor meiner 2. Diplomprüfung konnte ich den Grundstein für mein Programm legen. Zwar hatte ich mir schon früher Literatur zu diesem Thema zugelegt, doch mir fehlte einfach die Zeit. Schließlich fing ich in meiner Freizeit an, einfach darauflos zu programmieren, und nach und nach entstand schließlich Salomon - sicher nicht das Non-plus-Ultra, aber doch ein Programm mit beachtlicher Spielstärke. Eigentlich schrieb ich das Programm nur aus persönlichem Interesse, vor einem Monat kam ich allerdings auf die Idee, mein Programm als Diplomarbeit zu verwenden - der Aufwand war ja ungemein groß. Und an dieser Stelle möchte ich mich in aller Form bei Herrn Prof. Haase bedanken (Spezialist für künstliche Intelligenz an der TU-Graz), der von Anfang an reges Interesse zeigte, und Salomon schließlich als Diplomarbeit akzeptierte. Besonderen Dank möchte ich hier an dieser Stelle auch an meine Eltern aussprechen, die mir all die lange Zeit über die alltäglichen Sorgen abnahmen und mein Studium auch finanziell ermöglichten und natürlich auch an meine liebe Freundin, die ich in letzter Zeit (und wahrscheinlich auch noch ein wenig in der Zukunft) sehr oft mit Problemen der Programmierung nervte, und die dabei nie die Geduld verlor. Sie unterstützte mich nach Kräften und half mir auch entscheidend in der Testphase.

⇨ Jetzt bleibt mir nur noch viel Spaß beim Lesen dieser Arbeit zu wünschen, und ich bin überzeugt, daß jeder Leser, der nicht gerade ein Großmeister des Schachspiels ist, sicher seine Spielstärke verbessern kann und außerdem noch interessante Tips findet.

Graz den 12.Juli 1992

1. Übersicht über die Geschichte des Computerschachs

1.1 Ursprung des Schachs und erste Theoretiker

Einer alten Sage zur Folge sollte der Ursprung des königlichen Spiels in China zu finden sein. Dort hat sich eines Tages der amtierende Kaiser gelangweilt und seinen Untertanen aufgetragen, ein Spiel für ihn zu erfinden, das seiner würdig ist und das so komplex ist. daß für längere Zeit seine Langeweile vertrieben werden könne. Damals befand sich auch ein pfiffiger junger Schreiner im Lande, der den Aufruf des Kaisers zufällig auf einem seiner zahlreichen Reisen durchs Land aufschnappte. Und da eine hohe Belohnung ausgesetzt war (welcher Art diese war, wurde nicht erwähnt), setzte sich der junge Schreiner am Abend nach der Arbeit hin und begann zu überlegen: Kaiser und alle hohen Beamten interessierten sich in der Regel für Strategie. Speziell in den damaligen Zeiten, es war die Zeit der Mongolenkriege, war natürlich militärische Strategie stark in den Vordergrund gerückt - und somit war es klar: das Spiel mußte eine Menge Strategie beinhalten und außerdem sollte es Heere repräsentieren. Und so entstand das berühmte Brettspiel, das selbst nach Jahrtausenden noch immer populär ist, denn die Komplexität diese Spiels ist so hoch, daß selbst mit heutigen Mitteln diesem Spiel noch nicht beizukommen ist (d.h. bis in alle Einzelheiten analysieren). Der Schreiner meldete sich schließlich beim Kaiser, und wurde auch sogleich empfangen. Seine Majestät war entzückt! Mit so etwas hatte er nicht gerechnet, und so stellte er dem Schreiner frei, welcher Art seine Belohnung sein sollte. Und da sagte der einfache aber hochintelligente Mann folgendes: "Keine große Belohnung will ich - zumindest kein Gold. Meine Belohnung soll in Weizenkörnern bestehen, und zwar will ich für jedes der Felder meines Spiels zuerst ein Korn, dann zwei, dann vier, sechzehn und so weiter." Der Kaiser mußte aus Leibeskräften lachen, mit soviel, wie es ihm schien, Dummheit, hatte er fürwahr nicht gerechnet. Doch schon bald verging seiner Majestät das Lachen, ja blieb förmlich im Halse stecken, denn allein auf das letzte Feld mußte er eine neunzehnstellige (!) Anzahl von Weizenkörnern ($=2^{64}$) legen - die Kornkammern des ganzen Landes waren erschöpft und der pfiffige Schreiner ein gemachter Mann.
Bis ins 18. Jahrhundert gab es leider so gut wie überhaupt keine Theoretiker, zwar hatte ein spanischer Mönch namens Ruy Lopez eine umfangreiche Analyse von Eröffnungen erstellt (es gibt noch heute eine Eröffnung, die nach ihm benannt ist), aber Schach war in den damaligen Tagen nur auf den Adel und den Klerus beschränkt, dem breiten Volk war das königliche Spiel nicht zugänglich. Michail Botwinik datiert den Beginn des modernen Schachs mit dem Gewinn der Weltmeisterschaft von Steinitz gegen Hermann Zuckertort 1871. Steinitz meinte, wie alles in der Natur müsse es auch beim Schach für alles eine Ursache und Systematik geben. Auch dem Schachspiel sind gewisse Prinzipien zu Grunde gelegt, die man erlernen und beachten muß, will man ein Meister dieses Spiels werden.
Und diese Weisheit schlug sich sehr bald nieder - in wahren Bergen von Schachliteratur. Vor allem Eröffnung und Endspiel wurden bis in alle Einzelheiten analysiert,

und wer heute nicht die wichtigsten Eröffnungen kann, wird sich in Turnieren nie richtig durchsetzen können, auch wenn er ein glänzender Schachspieler ist.

1.2 Die Theoretiker für das moderne Computerschach : Shannon, Turing, von Neumann

Der Deutsche Konrad Zuse hat einen gewaltigen Anteil an der Entwicklung einer programmgesteuerten Rechenanlage beigesteuert. Schon bei der Planung spekulierte er mit der Entwicklung eines Schachprogramms, verwirklicht hat er jedoch nie eines. Wahrscheinlich war auch die Rechenleistung viel zu gering, um ein vernünftiges Programm zu verwirklichen, vom Aufwand der Eingabe (binär!!) ganz zu schweigen.
Urväter aller Schachalgorithmen sind jedoch die Mathematiker von Neumann und Morgenstern, die 1944 ihre Spieletheorie veröffentlichten. Diese besagt im wesentlichen, daß sich alle Spiele nach mathematischen Kriterien klassifizieren lassen. Demnach gehört Schach zu den endlichen zwei Personen Nullsummenspielen mit vollständiger Information. Doch was bedeutet Nullsummenspiel? Jeder aktuellen Stellung (current state) des Spiels wird eine sogenannte Bewertungssumme zugewiesen, die über den Spielstand Auskunft geben soll. Da es zwei Spieler gibt, gibt es selbstverständlich auch Bewertungen aus zwei verschieden Sichten. Nullsumme bedeutet nun, daß es zu jedem current state für den aktuellen Spieler einen Zug gibt, der den Vorteil des Gegners, den er gerade mit dem letzten Zug erreicht hat, wieder aufhebt. D.h. jeder Spieler trachtet in erster Linie danach, den gerade errungenen Vorteil des Gegners wieder aufzuheben und wenn möglich, sich einen eigenen Vorteil zu verschaffen[1]. Das Minimaxtheorem war geschaffen! Vollständige Information bedeutete, daß zu jedem Zeitpunkt jeder der beiden Spieler alle Informationen über die möglichen Züge des Gegners hat.
Am 9. März 1949 erschien in einem Wissenschaftsmagazin ein Meilenstein in der Computerschachgeschichte: Claude Shannon, Mitarbeiter der Bell Telephone Company in New Jersey, veröffentlichte ein bis heute gültiges Konzept für die Schachprogrammierung. Leider konnte er seine Theorie nie in die Tat umsetzen, zu schwach waren damals noch die Rechenleistungen. Auch die noch übliche binäre Eingabe erlaubte kaum die Entwicklung eines komplexen Programms.
Folgende Dinge hat Shannon, der ja auch als Gründer der modernen Informationsverarbeitung gilt, bereits geplant:
Zuerst beschäftigte ihn einmal die Darstellung des Brettes im Computer. Seinem Vorschlag zu Folge könne man das Brett durch 64 Worte zu je 64 Bit beschreiben. Jedes Feld enthält Informationen über den Zustand. Züge könne man dadurch realisieren, daß man jeder Figur eine bestimmte mathematische Operation zuordnet, die wenn sie auf das Feld angewandt wird, das Zielfeld ergibt. Die Felder sind von 1-64 durchnummeriert, wie Postfächer. Diese Idee der eindimensionalen Brettdarstel-

[1] Siehe auch Kapitel 2.2 Minimaximierung !

lung wird mit einigen geringen Abweichungen auch noch heute verwendet, näheres dazu später.
Die zweite Frage, die sich Shannon stellte, war wie kann ein Computer den besten Zug finden? Hierzu klassifizierte Shannon alle möglichen Vorgehensweisen in zwei Teilbereiche:

- Typ A oder Brute Force Programme
- Typ B oder selektiv suchende Programme

Brute Force (rohe Gewalt) Programme untersuchen alle möglichen Züge in einer Stellung, die Betonung liegt auf **alle.** Selektiv suchende Programme versuchen schon von Anfang an gewisse Züge als gut herauszukristallisieren und nur diese zu betrachten (diese Vorgehensweise ist die eines Menschen, denn auch er betrachtet nicht alle Züge, sondern nur einige wenige Varianten). Das Problem bei den Typ B Programmen liegt, wie man unschwer errät, natürlich in der zentralen Frage, welche Züge sind gut? Und diese Frage ist bis heute noch nicht geklärt - so sind ja sogar Schachmeister sich oft uneinig, welcher Zug denn nun der beste ist - und wenn es nicht einmal der Mensch eindeutig sagen kann, wie soll es dann ein Computer wissen, der ja vom Menschen programmiert wurde. Dennoch gibt es einige Ansätze in der Richtung, welche Züge denn nun interessant sind, ich werde darauf näher im Kapitel 2.4.5 Variable Suchtiefe bei guten Zügen eingehen.
Demzufolge sind die meisten kommerziell erhältlichen Programme Typ-A Programme, deren Hauptaugenmerk darauf liegt, zwar alle Züge zu betrachten, die aber trotzdem versuchen, den exponentiell wachsenden Spielbaum (wenn auch nur linear) zu stutzen. Mehr dazu im Kapitel 2.
Ein weiterer großer Vorreiter in Sachen Computerschach war der britische Mathematiker Alan Turing. Auch er war vornehmlich an der Künstlichen Intelligenz interessiert und Schach war einfach ein tolles Versuchsfeld dafür. Turing's Ideen ähneln sehr denen von Shannon, er ging aber noch einen Schritt weiter: er schrieb auch ein Programm! Da er aber weder die Geduld noch die Möglichkeiten besaß, dieses Programm auf einem Computer laufen zu lassen, ließ er ganze Partien von Hand (!) durchrechnen. Schach ist übrigens auch das einzige Spiel, das bis dato den Turing-Test erfolgreich bestritten hat, wenn auch Gary Kasparow einen kleinen Strich durch die Rechnung machte. Der Test hat folgende Aussage:

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

Bei einem Simultanschachveranstaltung, die der deutsche Großmeister Dr. Helmut Pfleger in Hamburg 1983 durchführte, hatte man drei Schachcomputer eingeschmuggelt, deren Bedienpersonal über Minifunkgeräte verständigt wurde. Nach der Veranstaltung wurde Herrn Pfleger das Einschmuggeln mitgeteilt und er wurde aufgefordert herauszufinden, welche der Partien denn nun von den Computern gespielt wurden. Er mußte passen! Selbst Experten, denen man die

Partienotationen vorlegte, konnten keine eindeutige Zuordnung treffen. Lediglich Gary Kasparow, der spätere Weltmeister, konnte bei einer der Partien eindeutige sagen, daß sie von einem Computer gespielt wurde. Somit entwickelte sich Schach zu einem äußerst interessanten Teilgebiet der künstlichen Intelligenz, in dem noch lange nicht alle Möglichkeiten ausgeschöpft sein werden.

1.3 Die ersten Schachautomaten

Man möchte kaum glauben, wie lange es schon sogenannte Schachautomaten gibt. Am Ende des 18. Jahrhunderts war die Feinmechanik ein beliebtes Hobby. So z.B. der sächsische Baron von Kempelen, der auch solche Dinge wie einen sich selbst deckenden Tisch (wie er u.a. auch in König Ludwigs II. Schloß Neuschwanstein zu finden ist) gebaut hat. Das Geschirr war einfach auf der Unterseite befestigt und kam auf Knopfdruck per Federkraft an die Oberseite. Für ihn eher zu den unwichtigen Erfindungen zählend, aber einen gewaltigen kommerziellen Erfolg einbringend, war der sogenannte *Türke*, ein "Schachautomat". Der aufmerksame Leser wird bestimmt sofort verstanden haben, warum ich Schachautomat unter Anführungszeichen setzte. Denn wenn selbst zu Zeiten von Turing noch kein vernünftiger Computer existierte, der schnell genug war, ein Schachprogramm zu verwirklichen, wie um Himmels willen sollte es dann einem Kempelen im 18. Jahrhundert gelingen, einen Schachautomaten zu konstruieren, noch dazu in einer Zeit, wo man von Computern noch nicht die geringste Ahnung hatte? Nun, die Lösung ist sehr einfach: Der Türke war im wahrsten Sinne des Wortes getürkt! Es handelte sich um eine quadratische Kiste (ca. 1.5x1.5 m und 90 cm hoch), mit einem Schachbrett auf der Oberseite. Außerdem befand sich dort noch eine Puppe, die die Figuren mechanisch bewegte. Die Kiste war mit allerhand Mechanik "verziert", die sich natürlich auch geheimnisvoll bewegte. 1770 präsentierte Kempelen seinen Automaten dem staunenden Wiener Hof und überzeugte das Publikum, daß die Kiste leer war, indem er hier und dort eine Klappe öffnete und Zahnräder und Federn zum Vorschein brachte. In Wirklichkeit befand sich in der Kiste ein kleinwüchsiger Spieler, dessen Name leider nicht überliefert wurde, der seine Gegner reihenweise schlug. Daß es sich dabei um einen Zwerg gehandelt hat, gehört dem Reich der Fabel an, denn in späteren Jahren tat sich vor allem der Meister Allgaier als versteckter Spieler hervor. Und von Kleinwüchsigkeit ist bei ihm keine Rede.

Der erste Käufer des Türken war Friedrich der Große, doch er entdeckte den Schwindel sehr bald und anschließend verstaubte der Türke in einem Abstellraum für ca. 20 Jahre.

Erst 1806 ließ sich Napoleon Bonaparte, der Eroberer Berlins, von seiner Armee den Türken vorstellen. In dessen Inneren befand sich der oben erwähnte Meister Allgaier. Der große Stratege und Feldherr korsischer Abstammung verlor natürlich gegen einen der größten Spieler seiner Zeit.

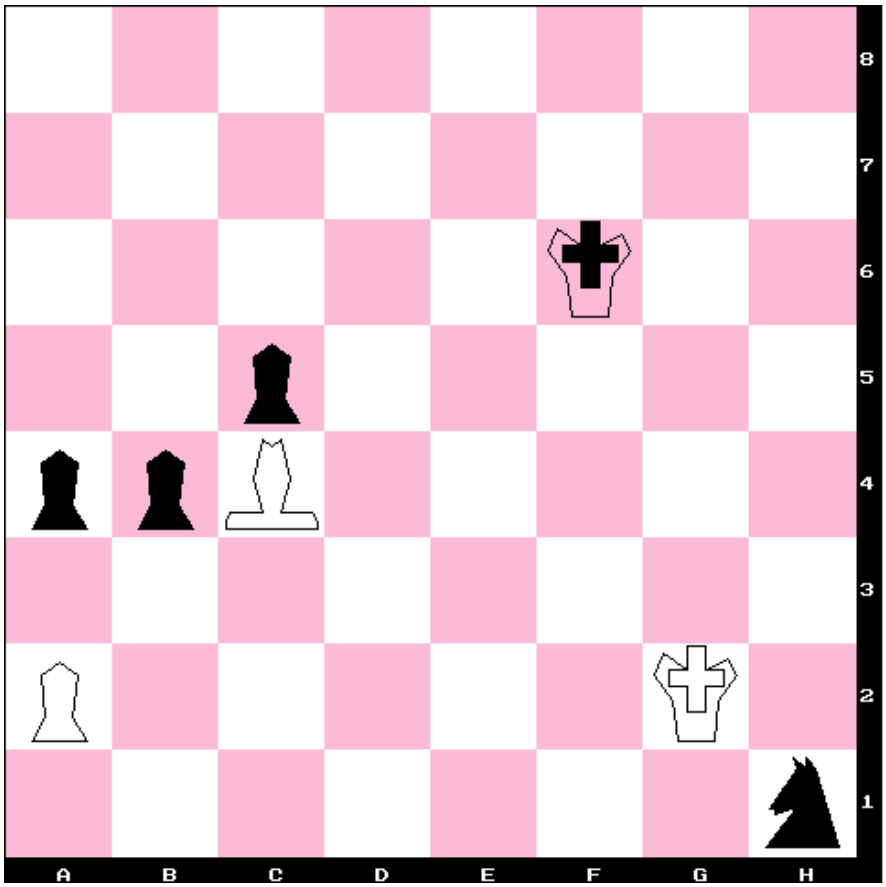
Danach ging die nun immerhin schon 40 Jahre alte Maschine auf Tournee nach Europa, wo sie tolle kommerzielle Erfolge erzielte. Später landete der Türke in einem Museum, wo er schließlich 1854 einem Feuer zum Opfer fiel. Natürlich hatte er schon zu Lebzeiten viele Nachahmer angeregt, aber niemand war im Stande, solch ein mechanisches Wunderwerk exakt nachzubauen, man bedenke, daß die Puppe immerhin alle Züge mechanisch ausgeführt hat und daß die Zugübermittlung ins Innere auch mechanisch erfolgte. Der Türke war auf jeden Fall für die damaligen Verhältnisse eine technische Sensation.

Weit ernsthafter, wenn auch unscheinbarer waren die Bemühungen des spanischen Mathematikers Torres y Quevedo, der um 1890 mit einem mechanischen Schachau-

tomaten experimentierte. Dieser Automat - den übrigens keiner, der über ihn geschrieben hatte jemals zu Gesicht bekam - hatte bereits sehr viele Ähnlichkeiten mit einem heutigen Schachcomputer. Zumindest soll er ein Spielfeld besessen haben, daß das Start- und das Zielfeld der zu ziehenden Figur angezeigt haben soll. Der einzige Haken bei der Sache war, daß Quevedos Schachautomat nur König Turmendspiele spielen konnte, und selbst diese nicht immer auf dem kürzesten Wege.
Diese erste wirkliche Schachmaschine zeigte erste verblüffende Parallelen zu den späteren Computern auf. Etwa zur selben Zeit wie Quevedo arbeitete das sagenhafte Mathematikgenie Charles Babbage, in Zusammenarbeit mit der nicht minder legendären Lady Ada Lovelace (die Programmiersprache ADA wurde nach ihr benannt), an seiner mechanischen Rechenmaschine. Diese Maschine nahm, obwohl sie nie vollendet wurde, schon eine ganze Menge der digitalen Signalverarbeitung vorweg.
Nach Torres y Quevedos Erfindung war es erst einmal 60 Jahre völlig ruhig, bis 1941 Konrad Zuse seinen Z3 herausbrachte, die erste digitale Rechenmaschine. Ab nun begannen die menschlichen Spekulationen eines nichtmenschlichen Schachspielers erneut.

1.4 Die ersten Großrechnerprogramme

Die ersten Versuche, Schach auf einem Computer zu verwirklichen, scheiterten an den noch zu geringen Rechenleistungen. In einem amerikanischen Atombombenzentrum versuchte man erstmals, Turings Ansätze zu verwirklichen. Die Programmierung fand auf einem Computer namens Maniac stand, was soviel heißt wie der Verrückte (was für ein treffender Name!). Das Programm spielte erbärmlich schlecht. Selbst als man das Brett auf 6x6 Felder reduzierte (man ließ die Läufer weg) dauerte es noch etliche Stunden, bis überhaupt ein Zug gefunden wurde.
So wurde das erste Schachprogramm erst auf Maniacs Nachfolger, MANIAC II zum Laufen gebracht. Von der Spielstärke her war das Programm allerdings eine Katastrophe.
Das erste Programm, das alle Regeln beherrschte und eine Partie korrekt zu Ende spielen konnte, wurde um 1955 auf einem IBM 704 geschrieben. Edward Lasker, einem engagierten Schachspieler ist der Name sicherlich ein Begriff (internationaler Schachgroßmeister), räumte dem Programm beachtliche Amateurspielstärke ein. Das Programm bediente sich notgedrungen einer Shannon Typ-B Strategie (geringe Rechenleistung) und konnte außerdem auch nur zwei Halbzüge tief sehen.
Die ersten ansehnlichen Erfolge erreichte schließlich Herbert Simon von der Carnegie-Mellon Universität in Pittsburgh mit seinem CP-1. Das Programm war in einer Hochsprache geschrieben und erzielte in der Eröffnung und im Mittelspiel beachtliche Erfolge. Die Schwächen des Programms lagen, sowie bei vielen anderen Programmen auch noch heute, im Endspiel. Hier machte das Programm ernsthafte Fehler, die man als Mensch sehr leicht ausnützen konnte. Erstmals in der Geschichte des Computerschachs stieß man auf den sogenannten "Horizonteffekt". Um ihn zu erklären betrachten sie einmal folgende Stellung:



Schwarz kann in dieser Stellung leicht gewinnen, wenn der König mit f6-e5 zentralisiert wird. D.h. Schwarz muß folgendes spielen: 1. .. K f6-e5 2. K g2xh1 K e5-d4 3. L c4-g8 K d4-c3 4. K h1-g2 b4-b3 und mindestens ein Bauer wird zur Dame! Völlig daneben wäre natürlich 1. .. b4-b3 ?? denn nach 2. a2xb3 a4xb3 3. Lc4xb3 hält Weiß ohne weiteres ein Remis. Viele Schachprogramme spielen leider auf niedriger Stufe diesen Unglückszug. Wie kommt das?
Das Programm findet in der Regel nach 1. .. b4-b3 die Varianten, daß Weiß entweder den Bauern gewinnt oder den Springer. Da das Programm aber nach der Minimax Methode[2] arbeitet, nimmt es automatisch an, daß Weiß den besten Zug macht, also den Springer schlägt. Also ist b3 richtig , denn weiter wird auf niedriger Spielstufe nicht gerechnet!! Der Horizont liegt hier also bei vier Halbzügen, weiter kann das Programm nicht blicken. Dieses Problem macht sich eben vor allem im Endspiel bemerkbar, denn gerade hier denkt der Schachmeister nicht selten 15 bis 20 Halbzüge tief. Erst die modernen Schachprogramme können auch im Endspiel sehr tief rechnen (z.B. Sargon V rechnete bei einer Endspielteststellung[3] in Null Komma nichts 14 (!) Halbzüge tief!).
Ein bereits fortgeschritteneres Programm wurde 1966 von einem amerikanischen Studenten namens Richard Greenblatt geschrieben. Ursprünglich war er an einem

[2] Siehe Kapitel 2.2 Minimaximierung
[3] Siehe Kapitel 5.4 Endspiele!

Projekt zur Erforschung der künstlichen Intelligenz beteiligt. Sein Programm MACK HACK VI hat allerdings nicht unmittelbar damit zu tun. Greenblatt ging es eher darum, die bereits bekannten Algorithmen mit besten Mitteln zu verwirklichen. Dazu stand ihm eine PDP-6 mit 256kB RAM zur Verfügung. Das Programm selbst schrieb er mit einem Macroassembler (MIDAS), der eine Fülle von Software Tools zum Debugging und zum Strukturieren der Daten aufweisen konnte. Greenblatt ging mit ungeheurem Ehrgeiz an die Arbeit, und in unglaublichen 3 Monaten (zum Vergleich: ich entwickelte an meinem Salomon fast ein Jahr!) war das Programm fertig. Erstmals waren Dinge wie:

- Optische Anzeige des Schachbrettes und Notation
- Eingabe der Züge in der üblichen Notationsform
- Optische Anzeige über die Bewertung
- Protokolle über die Auswahl der Züge
- statistische Funktionen über die Rechenzeit
- Anzahl der berechneten Züge

verwirklicht worden. Der Zweck war, das Programm laufend zu verbessern. Auch dies war bisher einmalig in der Geschichte des Computerschachs. Ein sehr großer Teil des Programms diente lediglich dazu herauszufinden, welcher Teil der Züge denn überhaupt betrachtet werden solle - es handelte sich also um ein Shannon-B Programm. Das Programm nahm 1967 an einer Schachmeisterschaft in Massachusetts teil und erreichte ein Remis bei fünf Partien (vier Niederlagen). Dieses Remis ist als erster Erfolg eines Programms gegen einen menschlichen Gegner bekannt, wo ein Programm unter Turnierbedingungen spielte.
MAC HACK wurde bis Mitte der siebziger Jahre laufend verbessert und das Programm lief auf allen Rechnern der PDP Serie im Time Sharing Verfahren. Die Idee, Schach am Computer zu verwirklichen bekam somit gehörigen Auftrieb, denn es spielten an den Geräten vorwiegend Leute, die auch die Funktionsweise des Programms verstanden. Leider nahm MAC HACK nie bei den US-Computermeisterschaften oder an einer Weltmeisterschaft teil, somit ist die Einordnung des Programms äußerst schwierig. Aber immerhin wurde MAC HACK Ehrenmitglied im US-Schachverband für seine besonderen Dienste um das Computerschach.

Bereits im Jahre 1968 hatte ein Student namens Larry Atkin an der Universität Northwestern an einem ziemlich schlecht spielenden Schachprogramm gearbeitet, als David Slate, ein ziemlich guter Schachspieler davon hörte und sich beteiligte. 1970 entstand dann schließlich CHESS 3.0, das auch an der US-Computermeisterschaft teilnahm. Das Programm war vollständig in Assembler verfaßt worden und lief auf einer CDC 6400. Das Programm gewann diese Meisterschaften.
Auch der Nachfolger CHESS 3.5 war bei den Meisterschaften 1971 siegreich und die Version 3.6 gewann schließlich auch noch 1972. Aber schon während dieser zwei Jahre hatten Slate und Atkin begonnen, einen von der 3er Serie abweichenden Ansatz zu verwirklichen. Bisher waren alle Programme vom Shannon Typ-B, mit der letzten Version schien den beiden das Konzept ausgereizt, CHESS 4.0 entstand, ein Shannon Typ-A Programm.

Das Konzept wurde ein voller Erfolg. CHESS 4.0 sowie die Nachfolger CHESS 4.5 und 4.6 dominierten das Computerschach von 1974 bis 1978. CHESS 4.6 wurde 1977 in Toronto sogar Weltmeister! Dazu aber später mehr.
Vorläufig wollen wir uns einmal eine Partie gegen den schottischen internationalen Großmeister David Levy anschauen. David Levy ist einer der Pioniere des Computerschachs, immer wieder wird er bei Computerschach-Turnieren als Leiter angeworben. Außerdem ist er einer der Gründer und Sprecher der ICCA (International Computer Chess Association).
Im Jahre 1968 kam es während einer Cocktailparty zu einer heißen Diskussion über die Spielfähigkeiten von Computerprogrammen zwischen Levy und zwei Professoren, die meinten, es würde keine zehn Jahre mehr dauern, bis es ein Programm gäbe, das ihn - David Levy - schlagen würde. Dies entlockte Levy nur ein mildes Lächeln, man schloß also eine Wette ab. Jeder der Kontrahenten setzte 250 Dollar ein. Diese Summe wurde wenig später von Seymour Papert (dem Erfinder der Sprache LOGO) auf 750 Dollar erhöht und in den folgenden Jahren stieg die Summe dann durch weitere Mitwetter auf gar 1250 Dollar.
Die Wette machte zwar in den Medien Schlagzeilen, doch bis 1977 fand sich keine Gelegenheit, einen Kampf auszutragen. Am 1. April 1977 trat dann David Levy in Pittsburgh (Pensylvania) zu einem Match gegen CHESS 4.5 an. Die Bedingungen waren für Levy günstig: Es waren zwei Partien angesetzt, von denen CHESS 4.5 beide gewinnen mußte. Levy mußte lediglich zwei Remis erreichen, um zu "gewinnen", oder einfach eine Partie für sich entscheiden. Levy gewann die erste Partie und damit das Turnier einfach:

1. e2-e4	c7-c5	18. e3-h6	c7-b6+	35. h4-h5	g6-g5
2. g1-f3	d7-d6	19 . g1-h1	b6-d4	36. c3-d5	a3-a2
3. d2-d4	c5xd4	20. d2xd4	g7xd4	37. e3-a3	e6xd5
4. f3xd4	g8-f6	21. f1-f3	d4-g7	38. e4xd5	c5xc2
5. b1-c3	g8-f6	22. h6xg7	g8xg7	39. e2-d1	c2-d2
6. f2-f3	f8-g7	23. a1-b1	c8-b6	40. h1-h2	c8-c1
7. c1-e3	0-0	24. f3-f1	f8-b8	41. d1-b3	a2-a1D
8. d1-d2	b8-c6	25. b1-d1	f7-f6	42. a3xa1	c1xa1
9. f1-c4	a7-a6	26. a3-a4	a6-a5	43.g3-e3 Weiß gab auf	
10. d4xc6	b7xc6	27. b4-b5	c6xb5		
11. 0-0	f6-d7	28. a4xb5	b8-c8	Weiß : CHESS 4.5	
12. f3-f4	d7-b6	29. d1-d3	c8-c5	Schwarz : David Levy	
13. c4-e2	c8-e6	30. d3-g3	a8-c8		
14. b2-b3	b6-c8	31. f1-f3	a5-a4		
15. a2-a3	d8-a5	32. h2-h4	a4-a3		
16. b3-b4	a5-c7	33. f5xg6	h7xg6		
17. f4-f5	e6-d7	34. f3-e3	d7-e6		

David Levy spielte hier ganz bewußt nach dem Motto "Wer wenig tut, macht wenig falsch". Nur zu gut wußte Levy um die Unfähigkeit von Programmen Bescheid, langfristige Pläne zu schmieden. Und genau so errang er auch relativ einfach einen Sieg. Selbst heute ist das noch eine sehr gute Strategie gegen Schachcomputer,

wenngleich sie auch nicht immer mehr wirkt, zu tief rechnen einige Programme schon voraus.
Das zweite und letzte Turnier fand 1978 in Toronto (Kanada) statt. Es sollte das alles entscheidende Turnier werden, die Wettsumme war inzwischen auf 2500 Dollar angestiegen, es ging also für Levy um ein ganz hübsches Sümmchen. Die Partie war auf sechs Spiele angesetzt, und CHESS 4.7 sollte gewonnen haben, wenn es mehr als drei Punkte aus diesen sechs Spielen holte. Auch dieses Turnier wurde von Levy schließlich siegreich beendet, wenngleich es diesmal nicht so leicht war, denn in der ersten Partie kam der Meister gehörig ins Schwitzen, als nämlich CHESS 4.7 mit zwei Bauern und starkem Angriff in Führung ging. Aber schließlich konnte sich Levy doch noch nach haarsträubender Verteidigung in ein Remis retten.

In der Zwischenzeit hatte die Konkurrenz aber auch nicht geschlafen, CHESS erlebte nur noch kleinere Aufschwünge mit den Versionen 4.7 und 5.0. An vielen Instituten in den USA aber auch in vielen Westeuropäischen Ländern wurde eine Menge Zeit und Geld in die Forschung investiert. Auch in der ehemaligen Sowjetunion arbeitete Michail Botvinnik, der Ex-Weltmeister an seinem Über-Superschachprogramm PIONIER, doch bis heute hat er noch keine lauffähige Version zusammengebracht. Die Erfolge der Sowjetunion liegen vielmehr in der Frühzeit des Computerschachs, da hatte die UdSSR nämlich das erste Weltmeisterprogramm KAISSA entwickelt.
In den Jahren 1966 und 1967 fand ein Schachwettkampf statt, der einiges Aufsehen erregte: Ein Programm der Stanford University spielte gegen ein Programm namens ITEP, das am Kybernetik Institut der Universität Moskau entwickelt worden war (von Michail Domoskoj und Wladimir Arlasorow). ITEP beendete drei von vier Partien siegreich, eine Partie endete mit einem Remis. Das Stanford Programm war durch seine selektive Suche (zu oft wurden wichtige Züge einfach abgeschnitten) immer wieder schnell in Nachteil geraten.
Aus ITEP entstand 1971 das Programm KAISSA (benannt nach der Göttin des Schachs), das auch gegen Menschen beachtliche Erfolge erzielte. Sogar der damalige russische Weltmeister Spasski mußte die Spielstärke von KAISSA anerkennen.
KAISSA war für seine Zeit eines der modernsten Programme auf der Welt. So untersuchte es jeden Zug fünf bis sieben Halbzüge tief und anschließend wurde noch eine Ruhesuche[4] nachgestellt, die die Schlagkombinationen noch tiefer analysierte. So arbeiten heute beinahe alle ernstzunehmenden Schachprogramme. KAISSA verfügte auch über eine ganze Menge an Schachwissen, sodaß es auch in positioneller Hinsicht erstaunliche Fähigkeiten besaß. Leider hatte KAISSA ein großes Handikap: sie neigte gerne zu unausrottbaren Patzern, die menschliche Gegner sehr leicht ausnutzen konnten, Computer taten sich hier allerdings ein wenig schwerer.
So wurde KAISSA im Jahre 1974 Weltmeister unter den Schachprogrammen, wobei sie von den Turnierregeln (Schweizer System) begünstigt wurde; so mußte sie z.B. keine einzige Partie gegen das so starke CHESS 4.6 spielen. KAISSA

[4] Siehe auch Kapitel 2.4.3 Die Suche

gewann seine Partien recht leicht und durfte sich somit als erstes Programm Computerweltmeister nennen.
1977 in Toronto war es dann allerdings umgekehrt: CHESS 4.6 gewann alle vier Partien, mußte aber wieder nicht gegen KAISSA spielen. Kurze Zeit später wurde jedoch diese Partie im Rahmen eines mit Spannung erwarteten Schaukampfes nachgeholt und CHESS 4.6 gewann beide Partien, KAISSA hatte zu arge Schwächen im Endspiel.
Also selbst Ende der siebziger Jahre waren die Programme noch äußerst schwach in diesem Stadium der Partie, obwohl die grundsätzlichen Algorithmen bekannt waren. Mehr Erfolg versprach man sich nur von einer Erhöhung der Rechenleistung.

Und 1978 war es dann soweit: Das Programm BELLE schockte die Computerschachwelt. BELLE war zunächst eigentlich ein Programm unter vielen anderen. Bereits 1972 wurde es für PDP-11 Rechner implementiert, interessant ist vor allem, daß das Programm unter UNIX lief und vom Vater dieses legendären Betriebssystems Ken Thompson höchst persönlich geschrieben worden war. Thompson, er arbeitet bei den Bell Telephone Labaratories, hatte in den folgenden Jahren beinahe unbegrenzte Möglichkeiten zur Verfügung. Seine Arbeitgeber versprachen sich nämlich mit dem Fortschritt des Programms auch Lösungen zu anderen komplexen Vorgängen.
Thompson wurde rasch klar, daß eine reine Software Verbesserung maximal Gleichstand mit der Konkurrenz bedeutete, mehr aber nicht. Also war die einzige Lösung eine geeignete Hardware zu entwerfen. Und so entwarf Thompson den ersten reinen Computer für Schachzwecke, der anfänglich aus 25 zusammenwirkenden Chips bestand. Sofort wurden bestimmte Funktionen erheblich beschleunigt.
Im Jahre 1978 verfügte BELLE über einen eigenen Schachprozessor und 325 Chips, der Prozessor war in spezifischem Mikrocode programmiert worden. Die Steuerung desselben übernahm ein PDP-11 Rechner, der in C programmiert war. Das Programm selbst hatte eine Länge von 90kB, dazu kamen 5kB und eine 128kB umfassende Tabelle.
Im Jahre 1980 schließlich wurde BELLE in einer Version Computerweltmeister, die bereits 1700 Chips (!) nur für Schachzwecke enthielt. BELLE war ein Brute Force Programm (Shannon Typ A), das im Mittelspiel durchschnittlich 8 Halbzüge tief rechnete, im Endspiel aber zwischen 10 und 30 Halbzügen tief blickte. In den Jahren 1980 bis 1983 war BELLE daher absolut konkurrenzlos.
Gefahr drohte eigentlich nur von anderen Giganten, und die ließen nicht allzu lange auf sich warten. BLITZ hieß das nächste Supercomputer Programm, das genauso wie BELLE zuerst eher unscheinbar war. Bereits 1976 hatten Robert Hyatt, Albert Gower und Harry Nelson an den amerikanischen Computerschachmeisterschaften teilgenommen und Platz vier erreicht. BLITZ ist ebenfalls ein Shannon Typ-A Programm, mit einer ausgefeilten Bewertungsfunktion (sie umfaßt mehrere tausend Statements), und genau das war das Problem: die ausgefeilteste Bewertung hilft nichts, wenn der Computer nicht in der Lage ist, sie auch in annehmbarer Zeit zu durchlaufen. Und so beschlossen die Entwickler, BLITZ auf dem damals größten Rechner, der CRAY ONE laufen zu lassen.

Die CRAY ONE, ein Vektorrechner mit damals unglaublichen 80 MFLOPS (Spitze, wenn alle Pipelines voll: 240 MFLOPS), verschaffte BLITZ die nötige Rechenpower. So war er nun in der Lage, 20000 Stellungen in der Sekunde zu untersuchen, was für damalige Verhältnisse astronomisch war (für heutige Verhältnisse zum Vergleich: der Chessmaster 3000 von Software Tools kann ca. 2000 Stellungen pro Sekunde auf einem normalen 386SX/16MHz untersuchen, Preisverhältnis ca. 1:1000000). Außerdem benutzte das Programm einen 120Mb umfassenden Hashing-Table. Dabei handelt es sich um eine besonders organisierte Datenstruktur (Zugriff in O(1) Schritten), in der bereits untersuchte Stellungen gespeichert wurden. Traf BLITZ während der Berechnungen wieder auf eine gleiche Stellung, rief er nicht mehr die zeitaufwendige Bewertung auf, sondern verwendete gleich die Ergebnisse der Tabelle, was natürlich zu einer ungemeinen Beschleunigung beitrug. Ein kleines Detail am Rande: Wenn BLITZ auf der CRAY ONE lief, dann funktionierte dies selbstverständlich nicht mehr im Time-Sharing, BLITZ benötigte die gesamte Rechenpower der CRAY ONE (!!), und dies kostete, wenn man nur die Rechenzeit beachtet, stattliche 500.000 Dollar (5 Stunden bei rund 30 Dollar pro Sekunde Rechenzeit)!
Am 9. November 1981 war es dann soweit, BLITZ und BELLE trafen das erste Mal aufeinander. Doch während BELLE zu diesem Zeitpunkt absolut auf der Höhe war, lief BLITZ nicht hundertprozentig zufriedenstellend. Und nach dem 42. Zug hatten die Operatoren ein Einsehen, sie beendeten den Kampf. BLITZ, der mit weiß gespielt hatte, war um eine volle Dame im Rückstand und neun Züge vor dem Matt.
Doch im Jahre 1983 schaffte es BLITZ schließlich dann, er wurde Weltmeister und die nächsten drei Jahre gab es keine Konkurrenz. Doch inzwischen kamen unvermutet von anderer Seite starke Angriffe, die Mikrocomputer waren im Anmarsch. Allen voran MEPHISTO, vom nun schon legendären Schachprogrammierer Richard Lang (auch die PC-Version von PSION entstand unter seiner Mitwirkung und ist noch immer das stärkste Programm, das ich auf dem PC besitze). Bis zum Auftauchen von DEEP-THOUGHT war MEPHISTO uneingeschränkter Weltmeister.

1.5 Die Mikrocomputer kommen

Es gleicht im Schach der Naturgeschichte, denn wie die großen Dinosaurier ausgestorben sind, so wurden auch die Supercomputer Giganten BELLE und BLITZ von den "Kleinen" verdrängt.
Im Jahre 1977 lief das erste Mal ein Schachprogramm auf einem Mikrocomputer, das auch anständige Partien spielen konnte. Es handelte sich um das Programm SARGON von Dan und Kathe Spracklen. Zunächst hatte das nun mittlerweile schon berühmt gewordene Schachehepaar eigentlich gar nicht die Absicht, irgendwie groß ins Geschäft einzusteigen. Große Teile von SARGON wurden, da Dan und Kathe anfangs gar keinen Mikrocomputer besaßen, in einem Pseudocode geschrieben. Schließlich erwarb das Ehepaar einen Z-80 Rechner und erlernte dessen Maschinencode.

Das Übertragen des Pseudocodes stellte sich aber dann als das kleinere Problem heraus, viel schwieriger und zeitaufwendiger gestaltete sich das Debugging. Ende Januar 1978 wählte das Programm endlich nur mehr legale Züge aus, mehr aber jedoch nicht! Doch dann gelang es den beiden in einem unglaublichen Kraftakt das Programm für die Mikrocomputermeisterschaft im März 1978 fertigzustellen. SARGON gewann das Turnier überlegen mit fünf von fünf möglichen Punkten und war ab sofort berühmt. Das Programm Listing wurde wenig später in Buchform veröffentlicht. Aber das war erst die Spitze des Eisberges. SARGON wurde laufend verbessert und tauchte auch unter den verschiedensten Namen überall in der Mikrocomputer und PC Welt auf. So wurde es z.B. unter dem Namen SARGON II eines der verbreitetsten Schachprogramme für den C-64 und APPLE Computer (auch ich habe in 64'er Zeiten oft gegen dieses Programm gespielt, seine Spielstärke ist tatsächlich beachtlich. Während SARGON II noch über eine äußerst bescheidene Eröffnungsbibliothek verfügte, so wurde SARGON III (auf IBM PC) schon mit 68.000 Halbzügen ausgestattet, was schon ganz schön ist (lediglich der Chessmaster 2100, 3000 kann da noch mithalten, er dürfte mindestens ebenso viele haben, die man noch dazu alle abrufen und üben kann).
SARGON taucht wie gesagt auch in zahlreichen Mikrocomputern auf, wie z.B.: BORIS, SENSORY CHALLENGER, ELITE, SENSORY 9, PRESTIGE (Fidelity) und in vielen anderen. Am 30. Oktober 1979 in Detroit (USA), trat SARGON bereits gegen BELLE an. Zwar verlor SARGON die Partie gegen den übermächtigen Gegner im 67. Zug, doch BELLE hatte wahrlich nicht geglänzt. Und bereits 1985 wurde der Sieg "gerächt": Das FIDELITY Programm ELITE XC hat die erste offene US-Meisterschaft für Schachprogramme siegreich beendet. BELLE wurde nur fünfter und konnte gegen Programme wie MEPHISTO Modular (erstmals trat ein Seriengerät an!) und Novag Y nur ein Remis erreichen.
Der Grund liegt wohl in der steigenden Leistungsfähigkeit der Mikrocomputer. Die Mikrocomputer der 80'er Jahre waren wesentlich leistungsfähiger als die Großrechner der siebziger Jahre. Außerdem waren die Algorithmen der Schachprogrammierung nun weitgehend bekannt und ausgereifter.
Die Weiterentwicklung der Programme kann nun auf zwei Arten geschehen:

- Von der Hardware Seite her, also mehr Rechenleistung
- Und von der Software Seite her, bessere Algorithmen, CUT-OFF Möglichkeiten etc.

Zum ersten Punkt ist anzumerken, daß die Rechenleistung der Mikrocomputer noch immer ständig wächst, vorläufig ist ein Ende noch nicht abzusehen (noch in diesem Jahr will INTEL den 80586 vorstellen), alles wird schneller, leistungsfähiger, kleiner, billiger. Natürlich wird man auch hier einmal an eine Grenze stoßen, aber sie scheint noch weit weg zu sein. Viel Potential liegt bestimmt auch in der immer mehr aufkommenden Multi-Prozessor Technik, die auch noch vieles in Petto hat - ich verweise auf das nächste Kapitel.

Vom softwaremäßigen Standpunkt kann man sich bis jetzt nur auf Alpha-Beta Cut, Forward pruning und Ruhesuche[5] stützen, mehr ist derzeit bei Shannon Typ-A Programmen nicht möglich (zumindest wurde nicht mehr veröffentlicht - ich traue mich wetten, daß im internen Wettkampf der einzelnen Firmen schon mehr erarbeitet wurde, aber diese Details, die die Spielstärke von Programmen natürlich wesentlich steigern, werden ja nicht der breiten Öffentlichkeit zugänglich gemacht. Warum ich auf diese Vermutung komme ist unter anderem, daß z.B. SARGON V in ca. 20 Sekunden im Endspiel 14 Halbzüge tief vorausberechnen kann - zum Vergleich: mein Salomon benötigt für die selbe Rechentiefe auf der selben Hardware Plattform fast eine dreiviertel Stunde!).

1.6 Deep Thought greift nach dem Weltmeistertitel

DEEP THOUGHT ist der nun schon seit etlichen Jahren amtierende Weltmeister, **derzeit** aber nur auf dem Computersektor. Doch warum lege ich so Wert auf die Betonung von derzeit?
DEEP THOUGHT wurde in den USA von 2 Amerikanern und einem Japaner entwickelt. Ein Amerikaner ist ein recht guter Schachspieler und kümmerte sich um die Bewertung, der zweite um die Organisation und Modularisierung. Der Japaner schuf die Hardware. Wie sie schon richtig vermuten, ist DEEP THOUGHT auf einem eigenen Computer, ähnlich wie BELLE, geschaffen worden. Doch die Rechenleistungen stehen zu BELLE in keinem Vergleich. DEEP THOUGHT ist in der Lage, mehrere Millionen (!!) Stellungen pro Sekunde zu untersuchen (zur Erinnerung: BLITZ schaffte auf der CRAY ONE "nur" 20.000 pro Sekunde). Außerdem stehen riesige Speichermedien zur Verfügung (Größenordnung: Giga-Byte), die eine riesige Eröffnungsbibliothek beinhalten und seit kurzem auch eine Endspieldatenbank (!). Die große Schwäche der Programme, das Endspiel, ist somit zu großen Teilen einfach weggewischt, denn viele Endspiele laufen nach ganz bestimmten Mustern ab, die man bis ins letzte Detail analysieren kann und Zug für Zug in eine Datenbank aufnehmen kann. DEEP THOUGHT benötigt im Falle eines in der Datenbank vorhandenen Endspiels also gar keine Rechenzeit mehr, sondern ruft gleich wie in der Eröffnung, die richtigen Züge einfach ab!
Ein weiteres interessantes, aber hoch umstrittenes Konzept, wurde in DEEP THOUGHT verwirklicht. Um die riesigen Speichermedien auch wirklich auszunützen, tat man sich die unglaubliche Arbeit an, unzählige gespielte Partien von Großmeistern und Weltmeistern einzugeben. DEEP-THOUGHT sieht vor jedem Zug nach, ob er denn vielleicht schon eine Partie gespeichert hat, die so abgelaufen ist. Wenn ja, braucht er wieder keine Rechenleistung für die Zugberechnung verschwenden. In sogenannten Ruhezeiten, also da wo DEEP THOUGHT gerade keine Partien spielt, ist er ständig damit beschäftigt, vorhandene Partien zu analysieren und in die Datenbank aufzunehmen.

---

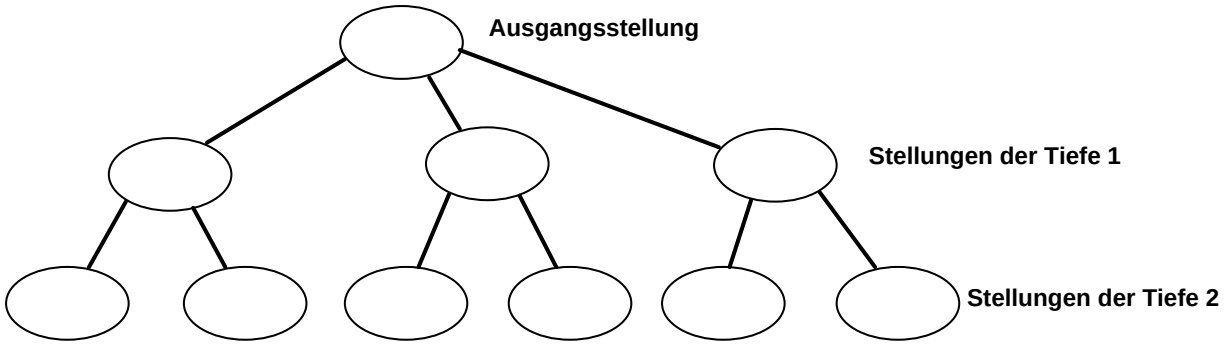
[5] Siehe auch Kapitel 2.4.3 Die Suche!

Doch was bringt nun das Multiprozessor Konzept? In der bisherigen Version war DEEP THOUGHT in der Lage, durchschnittlich 10-12 Halbzüge vorauszublicken, damit erreichte er eine (leider inoffizielle) ELO-Zahl von 2600 (!!) Punkten (Großmeister haben ca. 2200-2400, Gary Kasparow hält derzeit bei ca. 2800). Und das schlug sich auch in einer beinahe unglaublichen Spielstärke nieder. 99% aller Schachspieler auf der ganzen Welt haben keine Chance mehr gegen ihn. Selbst internationale Großmeister mußten sich dem Programm kopfschüttelnd geschlagen geben (auch David Levy, den ich weiter vorne mit seiner berühmten Wette vorgestellt habe, ist hier hinzuzuzählen). Ein kleines Detail am Rande: In einer Partie gegen einen Großmeister kündigte DEEP THOUGHT ein Matt in 17 (in Worten: in siebzehn) Zügen an, und führte es dann auch durch!! Die Stellung war übrigens noch durchaus einem Mittelspiel zuzuordnen.
Und DEEP THOUGHT wird ständig weiterentwickelt. Bereits 1993 ist eine neue Version geplant, mit noch mehr Rechenpower, noch mehr Prozessoren (etliche tausend, die von einem koordiniert werden), noch mehr Leistung. Mit diesem neuen DEEP THOUGHT sollte es dann möglich sein, bis zu 14 Halbzüge (im Schnitt!!) vorauszurechnen, in einer ELO Zahl ausgedrückt bedeutet das etwa 3000 ELO Punkte - diese Zahl, die auch gleichzeitig die höchste ELO-Zahl ist, die man überhaupt erreichen kann, hatte bisher noch kein Mensch, nicht einmal Ansatzweise erreicht. Bis heute kamen sogar erst zwei Schachspieler überhaupt über die astronomische Schranke von 2800 hinaus (nämlich der mittlerweile schon legendäre Amerikaner Bobby Fischer, der einzige Amerikaner, der je Schachweltmeister wurde und nun der amtierende Weltmeister Gary Kasparow).
D.h. eigentlich ist es nur mehr eine Frage der Zeit, bis wir in den Medien die Sensationsmeldung verlautbart bekommen, daß ein Computer Schachweltmeister ist, und die Chancen stehen gut, daß dieser Computer DEEP THOUGHT heißen wird. Wenn die Entwicklung also weiterhin mit solch Riesenschritten vorangeht, prognostiziere ich, daß obiger Fall in spätestens zehn Jahren eintreten wird, im Gegensatz zu David Levy werde ich meine Wette aber ganz bestimmt gewinnen!

2. Schachprogrammierung und Algorithmen

2.1 Darstellung eines Spiels im Computer

Das erste Problem, das bei der Programmierung eines Spiels grundsätzlich auftaucht, ist die sogenannte Zerlegung in Einzelzustände (current states), wie sie ja auch der Mensch durchführt - er betrachtet einzelne, augenblickliche Stellungen und bewertet diese nach gewissen Richtlinien (ein Anfänger anders als ein Meister, darin liegt eben meist der große Unterschied!). Beim Spiel Schach tut man sich mit der Zerlegung in Current states sehr einfach, da jeder ausgeführte Zug einen neuen Current state erzeugt, die Darstellung des Problems wird also quasi gleich mit den Spielregeln mitgeliefert (daß es nicht immer so einfach geht, zeigt z.B. das berühmte Problem des Farmers, der einen Fuchs, eine Gans und einen Kohlkopf über einen Fluß bringen soll, und dabei aber immer nur Platz für jeweils ein Ding hat. Weiters ißt der Fuchs die Gans, wenn er alleine mit ihr zurückbleibt, und die Gans den Kohl. Die Darstellung dieses Problems als Baum (kreisfrei) kostet schon einiges an Denkarbeit und Grundwissen in den wichtigsten Bereichen der KI. Zum Glück ist jedoch der Graph des Schachspiels kreisfrei, und wir brauchen uns nicht mit solchen Dingen herumzuschlagen).
Doch was ist nun ein Suchbaum? Oft taucht dieser Begriff auf, doch als Nichttechniker weiß man oft nichts damit anzufangen. Nun es handelt sich hierbei um die obenerwähnte Darstellung des Spieles Schach im Computer, und zwar geschieht das mit einer "Datenstruktur" (wir werden später noch kennenlernen, daß man eigentlich gar keine eigene Datenstruktur benötigt, sondern nur eine gescheite Programmiersprache, in der das Konzept der Rekursion verwirklicht ist), die aufgrund ihres Aussehens den Namen Baum bekam. Ob dieser Name gerechtfertigt ist, können sie ja einmal anhand der folgenden Abbildung beurteilen:



Hier ist ein einfacher vollständiger binärer Baum dargestellt (d.h. jeder Knoten (Vater) hat genau zwei Nachfolger (Söhne), außer den Blättern, die sich in der letzten Ebene befinden). Und genauso läßt sich Schach im Computer darstellen! Wenngleich es auch kein binärer Baum mehr sein wird, jeder Vater wird eben soviele Söhne haben, wie es gerade Züge in der aktuellen Stellung gibt (und das sind im Schnitt 40!), das Programm muß jetzt nur noch den Suchbaum durchlaufen und in geeigneter Weise (aus **zwei** Sichten, das ist auch ein wesentlicher Punkt!) die

beste Endstellung finden (Endstellungen befinden sich, wie man nur unschwer errät, natürlich in den Blättern). Zuerst könnte man einmal davon ausgehen, daß jede Schachpartie mit einem Remis endet (wie es z.B. die Theoretiker von Neumann und Morgenstern sahen). Diese Ansicht teile ich nicht unbedingt, da ja beim Schach immer einer zwangsläufig beginnen muß (Weiß). D.h. weiß ist von Anfang an um einen Zug vorne, also ständig im Vorteil, wenn man so will. Die Aufgabe von Schwarz besteht "lediglich" darin, diesen Vorteil mit dem (wohlgemerkt) nächsten Zug zunichte zu machen, und wenn möglich vielleicht etwaige Fehler von Weiß zu nutzen. Theoretisch gesehen müßte eigentlich Weiß immer gewinnen, sofern Weiß fehlerlos spielt (und wer tut das schon).
Meine Behauptung ist daher, leider läßt sie sich mit derzeitigen Mitteln noch lange nicht beweisen, daß es im Schach einen Zug in der Eröffnung gibt, mit dem man hundert prozentig jede Partie gewinnt, egal was Schwarz auch erwidert. Doch um diese Behauptung zu beweisen, müßte man alle nur denkbaren Partien durchspielen, bis man auf ein Matt stößt, und dies wird noch sehr sehr lange nicht möglich sein (in der Technik hat sich der bekannte James Bond Titel "Sag niemals nie" schon oft bewahrheitet, somit bin ich lieber vorsichtig ...), denn die Zahl der möglichen Stellungen beläuft sich auf eine achtzehnstellige Zahl, man bräuchte alle Zeit der Welt, um das alles berechnen zu können.
Fassen wir also kurz zusammen: hätten wir nicht das Problem einer Zeitschranke, so könnten wir mit dem oben vorgestellten Konzept eines Suchbaumes und noch ein paar Methoden, die in den folgenden Kapiteln vorgestellt werden, ein Schachprogramm verwirklichen, daß immer den besten Zug findet. Wie groß das Problem tatsächlich ist, soll folgende Tabelle verdeutlichen (es wurde angenommen, daß pro Tiefe 40 Züge möglich sind):

Tiefe	Stellungen
1	40
2	1600
3	64000
4	2560000
5	102400000

Die Anzahl der Stellungen wächst also, wie ja auch zu erwarten war, nicht linear sondern exponentiell, und das ist das zentrale Problem, mit dem sich jedes Schachprogramm der Shannon Typ-A Serie (und nur solche sind derzeit ernstzunehmen) herumschlagen muß. Die Explosion der Anzahl der zu untersuchenden Stellungen ist einfach enorm, wie jedoch macht das ein Großmeister, wenn schon nicht einmal ein rascher Computer mit der Vielzahl der Stellungen fertig wird?? Die Lösung, mit der man leider als Schachprogrammierer sehr wenig anfangen kann (seufz), besteht, wie man sicherlich leicht verstehen wird, in einer selektiven Suche. D.h. ein Meister dieses Spiels sieht sich eine Stellung einmal an, und dann kann er (teils aus Erfahrung aus unzähligen gespielten Partien, teils aus theoretischen Wissen und teils auch aus Intuition) relativ bald sagen, welche Züge denn nun eigentlich interessant sind. Und diese (und **nur** diese) betrachtet er dann ein wenig genauer, d.h.

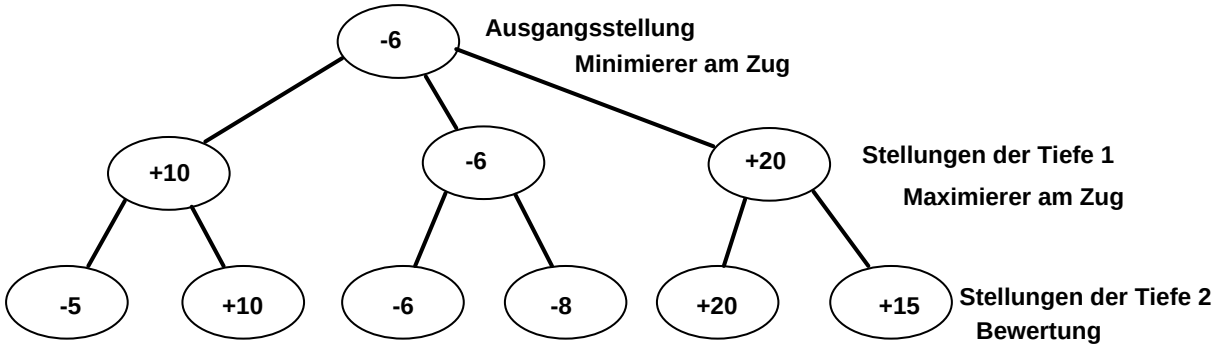
rechnet mehrere Züge voraus (Groß- bzw. Weltmeister betrachten ihre Hauptvarianten nicht selten 10 bis 12 Halbzüge tief - daher nehmen die Spieler auch bei jeder WM etliche kg ab (!), so enorm ist die Konzentration). Computer sind (derzeit und voraussichtlich in den nächsten 10 Jahren auch) nicht so "klug", sie müssen tatsächlich alle Züge betrachten, auch wenn sie noch so unsinnig sind (spielt sich z.B. der Kampf fast zur Gänze nur mehr am Königsflügel ab, so wird man doch wohl kaum Züge wie a2-a3 o.ä. betrachten, sondern eben zusehen, daß man sich entweder verteidigt oder den Angriff siegreich zu Ende bringt, je nachdem auf welcher Seite man spielt. Bauern auf der a-Linie interessieren da wahrscheinlich (es sei denn, es handelt sich hierbei um einen Freibauern!) i.a. nur peripher).
Daß aber dennoch nicht alle Züge betrachtet werden müssen, verdanken wir diversen CUT-OFF Möglichkeiten, allen voran das sogenannte Alpha-Beta Cut, dem wir uns im übernächsten Kapitel noch ausführlich widmen werden. Aber es gibt noch einiges mehr - man kann nur sagen, zum Glück, denn ansonsten wären Programme auf heutigen Mikrocomputern trotzdem noch nicht vernünftig einsetzbar.
Zunächst aber wollen wir uns einmal dem Problem widmen, eine Stellung "von zwei Seiten" aus zu betrachten (genauer: jede Tiefe muß vom jeweiligen Gegner betrachtet werden). Dies führt auf das Konzept der Minimaximierung.

2.2 Die Minimaximierung

Wie kommt es zu diesem ominösen Begriff, der doch in sich selbst ein Widerspruch zu sein scheint, denn entweder kann man nur minimieren oder nur maximieren, aber doch nicht beides zugleich - oder etwa doch?!
Zunächst möchte ich eine kleine Anekdote vorausschicken, die von meinem russischen KI Experten Dr. Nick Sherbakov stammt, und die auch er seinem Vortrag über Minimaximierung (was ein gebräuchlicher Begriff der künstlichen Intelligenz ist), voranstellte. Die kleine Geschichte stammt aus der Ukraine und lautet in etwa so:
Ein Ukrainer wurde befragt, was er denn mache, wenn er einen (kleinen) Laib Brot bekäme (man muß die derzeitige Hungersnot auch mitberücksichtigen). Er meinte, er äße ihn sofort auf. Die Frage wurde mit zwei, bzw. drei Laiben wiederholt, und die Antwort blieb immer dieselbe. Da wurde es dem Reporter zu bunt, und er fragte pfiffig (natürlich auf russisch, ich möchte hier aber auf Grund der besseren Lesbarkeit und aus Mangel an der geeigneten Tastatur darauf verzichten): "Ja und was tust du, wenn ich dir einhundert Laibe Brot gebe?" - Daraufhin dachte der Russe einige Zeit nach, dann antwortete er: "Ich beiße in alle Brote hinein!"
Eine glänzende Antwort und eine tolle Einführung in die Methodik der Minimaximierung! Das Grundkonzept ist hiermit nämlich schon erklärt: Minimaximierung heißt, daß es **zwei** Kontrahenten gibt, die zwei **verschiedene** Ziele (sogenannte goals, wie man in der KI-Welt auch sagt) ansteuern. Diese goals sind aber nur durch ein gemeinsames Netzwerk, dargestellt z.B. durch einen Suchbaum (die goals sind dann die Blätter), zu erreichen, und es versteht sich wohl von selbst, daß jeder der Kontrahenten alles nur erdenkliche unternimmt, daß (a) der andere sein goal nicht erreicht und daß (b) er selbst sein goal erreicht. Wie man sich leicht ausmalen kann,

wird in der Regel keiner der beiden sein Ziel erreichen, sofern nicht einer der beiden einen Fehler macht, oder schon mit einem unaufholbaren Vorsprung bzw. Nachteil (z.B. beim Schach die Vorgabe einer Dame), ins Rennen geht.
Wie funktioniert diese Methode nun konkret? Zuerst wollen wir den beiden Kontrahenten einmal Namen geben, und zwar wollen wir den einen Minimierer und den zweiten Maximierer taufen (welch Überraschung ...). Weiters wollen wir annehmen, daß für unser Spiel (z.B. TIC TAC TOE) eine Art Bewertung in Form von Zahlenwerten besteht, und zwar in der Form, daß eine Null eine ausgeglichene Stellung repräsentiert (keiner ist im Vorteil), eine negative Zahl einen Vorteil für den Minimierer und eine positive Zahl einen Vorteil für den Maximierer darstellt (symmetrische Bewertung). Man errät wohl unschwer, daß jeder der beiden natürlich nur solche Folgestellungen zu der aktuellen wählen wird, die seinen Punktewert verbessern, so wird z.B. der Minimierer immer danach trachten, eine Stellung zu ergattern, deren Bewertungswert möglichst negativ ist, zumindest den kleinsten Wert (auch wenn er positiv ist) wird er an sich reißen, der Maximierer versucht natürlich das Gegenteil. Und das Endergebnis des Ganzen? Ist natürlich der Wert Null, eine ausgeglichene Stellung. Beide ziehen gleichstark in verschiedene Richtungen (actio ist gleich reactio, wie Newton schon in seinem 3. Axiom treffend bemerkte), das Ergebnis ist, daß "gar nichts" passiert. Genau deswegen hat von Neumann, als er seine Spieletheorie veröffentlichte Schach als nutzloses Spiel abgestempelt, denn in Wirklichkeit passiere ja gar nichts. Nun das ist natürlich nur zum Teil richtig (sonst würde ich wohl kaum eine solche Diplomarbeit verfassen ...), denn der einzige (aber riesengroße Haken) an dieser Geschichte ist nämlich, daß das Konzept der Minimaximierung nur (und genau) dann aufgeht, wenn beide immer den optimalen Zug wählen (so kann man z.B. leicht beweisen, daß das Spiel TIC TAC TOE immer unentschieden endet, probieren sie es einmal!) und genau das ist das Problem: von Neumann hatte die unglaubliche **Komplexität** des Spieles nicht in Betracht gezogen. Und genau hier setzt die Schachprogrammierung an: nicht alle Züge können untersucht werden, aber z.B. bis in die Tiefe 4 lassen sich alle Züge in vernünftiger Zeit berechnen, alles was darüber liegt wird einfach abgeschnitten und von einer mehr oder weniger guten Bewertung[6] (hier liegt die eigentliche Intelligenz des Programms!!) ein Stellungswert errechnet, der dann eine Aussage darüber trifft, wie gut oder schlecht der jeweilige Spieler gerade steht.
Betrachten sie zur Verdeutlichung des eben Erklärten einmal folgendes Diagramm:

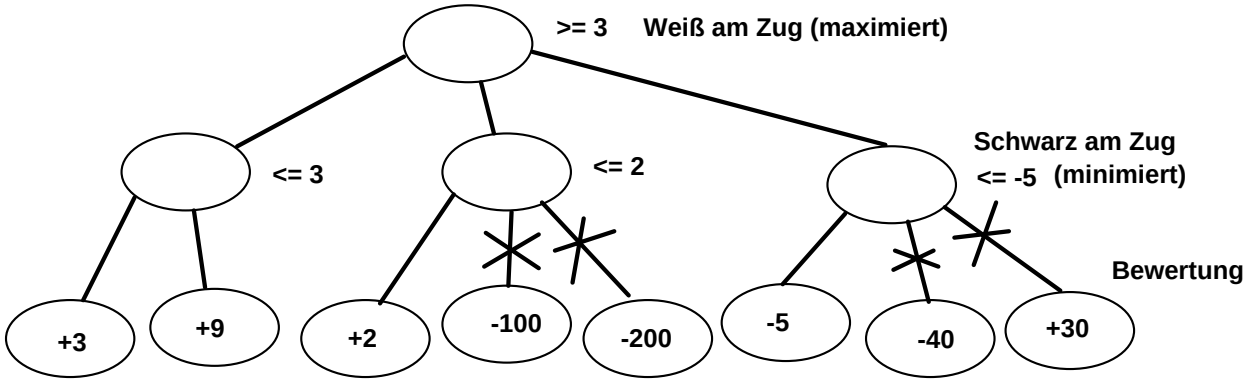
[6] Siehe auch Kapitel 2.4.4 Die Bewertung



D.h. im Endeffekt wird in obigem Diagramm der Minimierer jenen Zug wählen, der ihm -6 Punkte einbringt, aus seiner Sicht also der beste. Man sieht sehr deutlich, daß der Maximierer hier Gelegenheit hätte, sogar 20 Punkte zu erreichen, doch der Minimierer verhindert dies natürlich, indem er den mittleren Zweig wählt, dafür verhindert der Maximierer aber wieder, daß der Minimierer sogar -8 Punkte bekommt, denn in dieser Ebene ist er am Zug, und er wählt den aus seiner Sicht besten Zug, und das ist der mit -6 Punkten! Und somit haben sie bereits alles verstanden, was es zum Thema Minimaximierung zu erklären gibt. Nach diesem Prinzip arbeiten übrigens alle zwei Personen Spiele, so auch Schach. Doch nun wollen wir uns einem Thema widmen, das der Explosion des Suchbaumes entgegenwirkt, wenngleich nur linear, stellt es doch eines der zentralen Themen in der Künstlichen Intelligenz überhaupt dar (den Bereich "Suche und Lösung von Problemen mit Suchbäumen" betreffend), und als Programmierer eines Schachprogramms kommt man darum nicht herum. Die Rede ist vom:

2.3 Alpha-Beta Algorithmus

In den vorigen zwei Kapitel haben wir uns mit den Begriffen "Suchbaum" und "Minimaximierung" vertraut gemacht, wobei ich zur Suche noch eine kleine Zusatzanmerkung machen möchte: die übliche Methode beim Schach, den Suchbaum zu durchlaufen, ist die sogenannte Tiefensuche (im Gegensatz zur Breitensuche), d.h. es wird zuerst solange ein Ast in der Tiefe abgesucht, bis man zu einem Blatt gelangt (Bewertung), dann wieder solange Ebenen hoch, bis man wieder in die Tiefe gelangt. Denkbar wäre auch die sogenannte Breitensuche, die zuerst einmal alle Söhne in der nächsten Ebene untersucht, und erst dann eine Ebene weiter nach unten eilt. Selbstverständlich kann man beide Methoden auch kombinieren, wie und warum das sinnvoll ist, werden wir im Kapitel 2.4.3, der Suche kennenlernen.
Jetzt wollen wir aber noch einmal den Suchbaum betrachten, der ja exponentiell von Ebene zu Ebene anwächst. Und ähnlich wie ein guter Schachspieler können auch Computer bei der Suche etliches an Zeit sparen, indem sie einfach ein wenig "mitdenken". Betrachten wir einmal folgendes Beispiel:



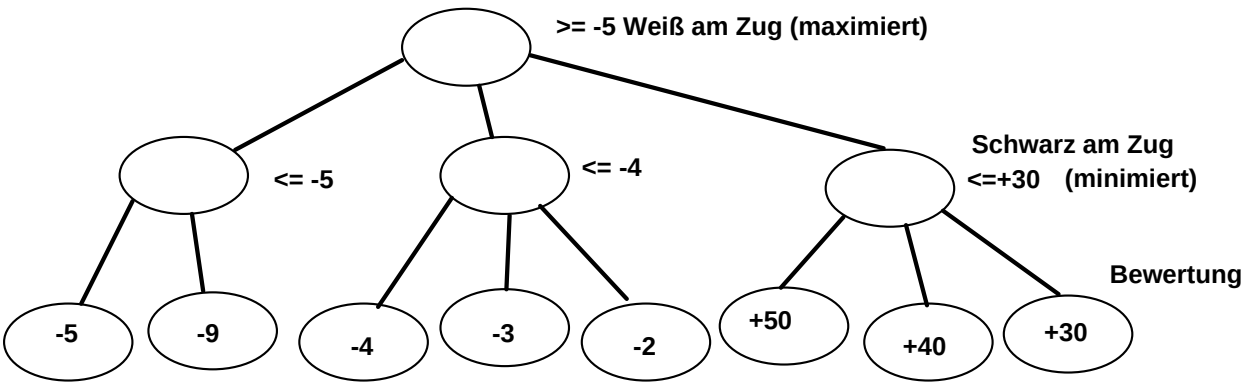
Wir wollen annehmen, daß Weiß in der aktuellen Stellung am Zug ist und gleichzeitig den Maximierer darstellen soll, Schwarz ist der Minimierer. Doch wie kann man nun gewisse Äste im Baum "abschneiden", so wie es im obigen Beispiel bereits eingezeichnet ist? Dazu wollen wir weiters annehmen, daß der Suchbaum von links nach rechts durchlaufen wird, und zwar per Tiefensuche. Zuerst ist also Weiß am Zug, er wählt zunächst einmal den Ast, der am weitesten links ist, als nächstes ist der Schwarze an der Reihe, der in der Folge dann zwischen zwei Zügen wählen kann, die die Werte +3 bzw. +9 besitzen. Da aber Schwarz der Minimierer ist, wählt er selbstverständlich den Zug mit +3 und es geht wieder nach oben, bis in die Wurzel zurück. Dort kann jetzt schon der erste "Denkvorgang" stattfinden. Weiß kann nun etwa sagen: "Aha, jetzt habe ich schon **mindestens** einen Wert von +3." Dies merkt sich das Programm - ebenso macht das auch ein guter Schachspieler. Wenn er z.B. sieht, daß er mit einem Zug einen Läufer des Gegners gewinnen kann, so merkt er sich diesen Zug auch, er weiß somit, daß er einen Läufer schon sicher hat. Doch weiter im Beispiel: als nächstes betrachtet Weiß den mittleren Ast, und Schwarz hat dann in der Folge die Wahl zwischen drei weiteren Zügen (mit den Werten: +2, -100, -200). Besonders der dritte Zug scheint sehr verlockend für Schwarz zu sein, doch diesen Zug braucht Schwarz gar nicht mehr betrachten (!), denn schon nachdem der erste Zug mit einem Wert von +2 betrachtet worden ist steht folgendes fest: die Suche "weiß" bereits, daß Weiß schon einen Wert von mindestens +3 hat, da aber Schwarz nun zu minimieren beginnt, hat Schwarz schon mindestens einen Wert von +2, einen größeren Wert wird er (sofern er noch ganz bei Trost ist) bestimmt nicht wählen, denn +2 hat er **sicher**! So hat man sich in diesem (recht einfachen Fall) zumindest zwei (äußerst zeitaufwendige) Bewertungen gespart, denn egal welche Werte in den Knoten auch stehen, Weiß wird diesen Zug ohnehin nie wählen!

Und genau dasselbe Spiel geht beim dritten Zug von Weiß vonstatten. Wieder beginnt Schwarz zu minimieren, und trifft schon bei der ersten Bewertung auf einen Wert von -5 (und freut sich ...) - und wieder braucht er die beiden anderen Züge gar nicht mehr betrachten, denn Weiß hat ja schon sicher +3, den Wert -5 nimmt er nicht freiwillig.

Bleibt eigentlich nur noch zu klären, warum man die ganze Geschichte Alpha-Beta Algorithmus (oder auch Alpha-Beta Cutting) nennt. Das ist relativ einfach: Genauso wie wir bei der Minimaximierung Weiß den Maximierer und Schwarz den

Minimierer tauften, so taufen wir nun **Weiß Alpha** und **Schwarz Beta**. Wobei diese Namen allerdings Variablen darstellen sollen, in denen für **jede** Suchtiefe (die Betonung liegt auf jede, globale Variablen tun es hier nicht!) die jeweiligen Werte gespeichert sind, die Weiß (Alpha) bzw. Schwarz (Beta) schon sicher haben. Im obigen Bild hatten wir nur sogenannte **Alpha Cut-Offs**, den die gestrichenen Äste wurden deswegen abgeschnitten, weil der **aktuelle Wert kleiner als Alpha** war. Hätten wir den Suchbaum noch um eine Tiefe erweitert, wären auch sogenannte **Beta Cut-Offs** möglich gewesen, die ganz analog dadurch zustande kommen, daß der **aktuelle Wert größer als Beta** ist, d.h. daß Schwarz einen solchen Zug nicht wählen würde, da er den Wert von Beta ja schon ganz sicher hat.
In der Folge spricht man dann auch vom sogenannten "Alpha-Beta Fenster", da ja Alpha stets kleiner gleich Beta sein muß (wie man sich leicht überzeugen kann), stellt man den Alpha-Beta Algorithmus auch so dar:

Die Gerade soll alle Werte repräsentieren, die von der Bewertung geliefert werden können, theoretisch also von minus Unendlich bis plus Unendlich (wobei sich der PC mit solchen Dingen ein wenig schwer tut, für Salomon werden wir dann später als Grenzen -16000 und +16000 definieren). Auf der Geraden sind die jeweiligen Werte von Alpha und Beta aufgetragen, und man beachte: nur die Werte interessieren uns, die größer als Alpha und kleiner als Beta sind, also jene die innerhalb des Fensters liegen, der andere Teil kann getrost weggeworfen werden. Ganz gleich handelt übrigens auch ein guter Schachspieler. Findet er z.B. einen Zug, bei dem er seine Dame verliert, so wird er bestimmt diese Variante nicht mehr weiterverfolgen, auch wenn er in der Folge vielleicht gar noch Matt gesetzt wird, der Verlust der Dame ist schon katastrophal genug!
Noch eine Sache ist beim Alpha Beta Algorithmus äußerst interessant, betrachten sie dazu doch einmal das folgende Diagramm, daß dem ersten fast gleich ist, bis auf die Werte:



Und dem aufmerksamen Leser sticht doch bestimmt etwas sofort ins Auge - wo sind denn plötzlich all die schönen CUT-OFFS geblieben?? Nun die sind auf einmal alle weg, denn im obigen Suchbaum gibt es keine einzige Cut Off Möglichkeit! Überzeugen wir uns davon:
Zuerst durchläuft Weiß, wie auch im vorigen Beispiel, den linken Teilbaum, und kommt schließlich zu dem Ergebnis, daß es einen mindestens einen Wert >= -5 bekommen wird (Alpha:=-5). Jetzt wird der mittlere Ast untersucht, Schwarz hat nun drei Züge zur Auswahl - und nun sieht man schon das Problem: untersucht man die Züge so wie wir es anfangs definiert haben von links nach rechts, so erhält man einfach keinen Wert, der kleiner als Alpha (=-5) ist, ja ganz im Gegenteil: jede Stellung würde Alpha sogar noch verbessern, und würde uns Schwarz nicht einen Strich durch die Rechnung machen, kämen wir sogar auf einen Wert von -2! So können wir aber immerhin (nachdem Schwarz minimiert hat) unser Alpha auf -4 verbessern, aber der Preis war hoch: wir mußten **alle** Stellungen ohne Ausnahme untersuchen!! Und jetzt kommt der letzte Ast an die Reihe, und wieder dasselbe: Schwarz findet zunächst +50 (Weiß jubiliert schon ...), und das ist doch wohl eindeutig größer als Alpha, also wieder nichts mit dem Schere Ansetzen. Und das gleiche Schicksal ereilt uns dann auch bei den Werten +40 und +30. Schwarz wird natürlich den Zug mit dem Wert +30 wählen, und erst jetzt hat Weiß seinen besten Zug gefunden, es ist derjenige, der im rechten Teilbaum ganz rechts angeordnet ist, und das schlimmste an dieser Angelegenheit: wir konnten tatsächlich keinen einzigen Cut Off machen, immer fanden wir die guten Züge viel zu spät.
Und genau das ist auch des Pudels Kern, wie in Goethes Faust so treffend bemerkt wird, wir haben die schlechtesten Züge (aus der Sicht von Weiß) immer zuerst betrachtet, die Züge waren also (im Gegensatz zum ersten Beispiel) gerade umgekehrt sortiert, deswegen konnten wir kein einziges Mal die Schere benutzen, jedesmal fanden wir im nächsten Teilbaum einen besseren Zug als im vorhergehenden. Doch wie macht denn das ein richtiger Schachspieler? Bei dem läuft die Sache etwa so ab: zunächst einmal werden von der aktuellen Stellung die interessanten Züge herauskristallisiert, nehmen wir folgenden einfachen Fall an (der sich auch leicht auf den Computer übertragen läßt): es gibt in der aktuellen Stellung genau drei mögliche Züge, und zwar kann man mit dem ersten einen Bauern, mit dem zweiten einen Läufer und mit dem dritten eine Dame gewinnen. Nun liegt es doch auf der Hand, welchen Zug der Spieler als erstes betrachten wird, natürlich den, der die Dame schlägt!! Hier kann man ja am meisten gewinnen, und dieser Zug wird auch in der Regel die anderen Züge widerlegen, es sei denn man wird durch das Schlagen der Dame plötzlich Matt gesetzt, dann wird der Spieler wohl den zweit besten Zug betrachten (den, der den Läufer gewinnt) usw. Und was hat der Schachmeister in Wirklichkeit getan?
Die Lösung heißt **Sortieren**!! Der Schachmeister hat die Züge nur ihren Werten nach geordnet, und hätte er nicht, wie in unserem Beispiel, beim ersten Zug gleich ein Matt entdeckt, dann hätte er sofort ein Maximum an Cut Offs erreicht. Denn wie wir ja bereits wissen: man erhält ein Maximum an Cut Offs, wenn wir den besten Zug zuerst betrachten. Und diese einfache Möglichkeit der Sortierung (wesentlich schwieriger wäre es z.B. schon, wenn in der aktuellen Stellung überhaupt kein Schlagzug existiert, man also praktisch nach positionellen Kriterien sortieren

müßte) kann auch ein Computer bewerkstelligen, indem er einfach eine sogenannte Vorbewertung durchführt (wie sie auch mein Salomon implementiert hat), in der einfach festgestellt wird, welcher Zug am meisten bringen könnte (ich rede hier nur von materiellen Kriterien - positionelle Kriterien sind noch Sache der Schachmeister, hier können Computer noch nicht soviel mitreden), und dieser wird in der Folge dann auch als erster betrachtet. Somit erreicht man auf eine einfache Art und Weise eine maximale Anzahl von Cut Offs (doch jede Medaille hat zwei Seiten, die Kehrseite der Medaille werden wir dann im Kapitel 2.4.3 kennenlernen, vorläufig können wir uns noch über das Ergebnis uneingeschränkt freuen!).

Hier vielleicht noch eine kleine Statistik, was das Alpha-Beta Cut reell für einen Gewinn bringen kann, wobei in der Tabelle der jeweils beste bzw. der schlechteste Wert dargestellt ist, der Gewinn liegt wohl meist irgendwo in der Mitte) :

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

Mittlerweile haben wir aber soviel Grundwissen erworben, daß wir uns dem eigentlichen Thema dieser Arbeit widmen können: der Schachprogrammierung. Im nächsten Kapitel werden wir nach und nach, wie mit einem Baukasten die einzelnen Module, aus denen ein Schachprogramm besteht, kennen- und verstehen lernen.

2.4 Grundsätzlicher Aufbau eines Schachprogramms

2.4.1 Darstellung des Bretts und der Figuren

Der erste wichtige Teil ist es, einmal festzulegen, wie wir im Computer (a) das Spielfeld und (b) die Figuren darstellen. So mancher Leser wird sich vielleicht schon gefragt haben, warum Shannon[7] eine so seltsame Darstellung gewählt hat, die so sehr Postfächern ähnelt. Zu Erinnerung: Shannon hat die Felder einfach von 1 bis 64 durchnummeriert und ich hatte bemerkt, daß diese Darstellung (mit einer kleinen Erweiterung) noch bis heute verwendet wird. Hätte man mich mit dem Problem der Brettdarstellung konfrontiert, so wäre ich zuerst einmal auf eine ganz andere Datenstruktur gekommen, nämlich auf ein zweidimensionales Array, sowie es (wohlgemerkt) der menschlichen Anschauung am nächsten kommt. Eine zweidimensionale Darstellung könnte etwa wie folgt aussehen:

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8
5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8
6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8
7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8
8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8

Wobei die erste Zahl für die jeweilige Zeile (Schachbrett 1-8) und die zweite Zahl für die jeweilige Spalte (Schachbrett a-h) steht. Für einen Menschen ist obige Darstellungsform einfach ideal, denn genauso sieht er ja auch das Brett. Wir werden aber sofort sehen, daß diese Datenstruktur für einen Computer gar nicht so gut geeignet ist. Versuchen wir doch einmal Züge auszuführen, z.B. wollen wir einen Bauernzug betrachten, der nach links oben schlägt. Folgende Operationen sind dazu notwendig, wenn wir z.B. den Zug von (5,4) auf (4,5), also d4xe5 betrachten:

Zeile:=Zeile-1
Spalte:=Spalte+1

Wir merken uns also: **zwei** Rechenoperationen sind nötig, um **einen** Zug auszuführen. Wenn jetzt wer behaupten würde, daß dies auf jeden Fall notwendig ist, der irrt! Mit Shannons Darstellungsmethode können wir genau die Hälfte an Rechenoperationen sparen, es ist demnach nur mehr eine einzige notwendig (der Vorteil liegt natürlich auf der Hand, denn Züge müssen in der Regel äußerst viele generiert werden, und da fällt es sehr wohl ins Gewicht, wieviele Operationen dafür notwendig sind!). Allerdings hat Shannons Modell in der ursprünglichen Form auch einen

[7] siehe Kapitel 1.2

Nachteil. Doch zunächst will ich einmal die Darstellungsweise, wie sie heute verwendet wird im folgenden Diagramm aufführen:

133	134	135	136	137	138	139	140	141	142	143	144
121	122	123	124	125	126	127	128	129	130	131	132
109	110	111	112	113	114	115	116	117	118	119	120
97	98	99	100	101	102	103	104	105	106	107	108
85	86	87	88	89	90	91	92	93	94	95	96
73	74	75	76	77	78	79	80	81	82	83	84
61	62	63	64	65	66	67	68	69	70	71	72
49	50	51	52	53	54	55	56	57	58	59	60
37	38	39	40	41	42	43	44	45	46	47	48
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

Bei dieser Form der Darstellung wird das tatsächliche Spielfeld eingebettet, konkret bedeutet das folgendes: das Feld Nummer 27 stellt a1 dar, Nummer 34 a8, weiters stellt dann 39 a2, 40 b2 usw. dar, die restlichen Eckpunkte sind dann Feld Nummer 111 für a8 und Nummer 118 für h8. Auf den ersten Blick scheinen nur Nachteile zu existieren. So z.B. haben wir jetzt 144 Postfächer und nicht nur 64 wie Shannon und vor allem, wie wollen wir denn eine Rechenoperation sparen?
Betrachten wir also obiges Beispiel noch einmal: wir wollen den Zug d4xe5 ausführen, nach unserer neuen Darstellungsform also von Feld 66 auf Feld 79 gehen. Und tatsächlich, es ist lediglich **eine** Addition von +13 zum Index des Feldes notwendig, um das gewünschte zu erreichen! Selbstverständlich funktioniert dies auch für die anderen Richtungen, z.B. links unten ⇔ -13 oder zwei Felder gerade nach oben ⇔ +24 usw., auch Springerzüge sind auf dieselbe Art und Weise möglich. Soweit so gut, doch warum brauchen wir 144 Einträge in unserem Array, würden es nicht auch 64 tun, so wie es der gute alte Shannon vorschlug? Wenn man die Darstellung allerdings auf 64 Felder begrenzt, ergibt sich ein Problem, daß ich schon weiter oben erwähnt habe (vielleicht haben sie ja in der Zwischenzeit schon ausgetüftelt, um welches es sich dabei handelt?). Nehmen wir doch einmal an, der Computer hat einen Turm auf h8 stehen und beginnt nun mit der Zuggenerierung. Für einen Zug nach rechts muß er einfach +1 zu den Indizes hinzuzählen, und schon hat er das Zielfeld - doch halt werden sie sagen, von h8 kann man doch gar nicht mehr nach rechts gehen, dabei würde ja das Spielfeld verlassen werden (ich weiß das klingt verrückt) - vollkommen richtig, doch man bedenke, der Computer hat nur ein **ein**-dimensionales Array mit 64 Indizes (wenn wir weiters annehmen, wir hätten Shannons Darstellungsform gewählt), die Betonung liegt auf der Silbe ein! Im letzten Diagramm habe ich nur aufgrund der leichteren Verständlichkeit das eindimensionale Array zweidimensional gezeichnet, in Wirklichkeit handelt es sich natürlich um eine lineare Verkettung von Zahlen, die einfach der Reihe nach im Speicher stehen, die Information über die Brettgrenzen, die wir ja bei der zweidimensionalen

Darstellung praktisch implizit mitgeliefert bekommen hatten, sind nun plötzlich weg! Und somit ist es gar nicht sonderlich verwunderlich, wenn das Programm plötzlich einen Zug von Feld 34 (h8) auf Feld 35 (das wäre a2, hätten wir keine eingebettete Darstellung, vgl. Diagramm!) macht, was natürlich völlig unsinnig ist. "Also muß man eben eine geeignete Abfrage machen..." könnte wer entgegnen - könnte man bestimmt, aber damit würden wir ja genau den Vorteil wieder einbüßen, den wir gegenüber der zweidimensionalen Darstellung gewonnen haben, nämlich das Einsparen einer Rechenoperation. Denn was hilft es, eine Addition (Subtraktion) weniger zu haben, wenn wir jetzt dafür ständig prüfen müßten, ob wir gerade das Spielfeld verlassen? Nun es gibt eine etwas elegantere Lösung als eine ständige Abfrage, und das ist eben die Einbettung des Feldes in einen Rahmen von zwei Feldern Breite. Und das funktioniert so: wir belegen einfach die Feldinhalte derjenigen Indizes, die sich außerhalb des Spielfeldes befinden, also illegal sind, mit einem besonderen Wert, der uns sofort erkennen läßt, ob wir uns noch innerhalb des Spielfeldes befinden - so einfach ist die Sache! Die letzte Frage, die noch zu klären wäre ist, warum wir denn das Feld mit einem Rahmen von zwei Feldern umgeben, ein Feld würde es doch zumindest auf den ersten Blick auch tun. Leider nicht, denn bei einem Rahmen von nur einem Feld Breite würden wir bei Springerzügen (wobei der Springer irgendwo am Brettrand steht) noch immer das Problem haben, einen Zug ins Leere (sprich wieder auf einem legalen Feld zu landen) zu generieren, wie brauchen also einen Rahmen von zwei Feldern, nicht mehr aber auch nicht weniger.
Der Name dieser Darstellungsform ist übrigens: **eingebettete eindimensionale Brettdarstellung,** und sie findet in allen modernen Schachprogrammen Verwendung, selbstverständlich macht auch Salomon™ davon Gebrauch.
Zuletzt bleibt nur noch zu klären, was denn nun eigentlich der Inhalt des Feldes ist (bisher sprachen wir ja immer nur von Indizes). Der Inhalt ist selbstverständlich eine Information, die über den aktuellen Zustand des Feldes Auskunft gibt, und diese Info wird in der folgenden Tabelle dargestellt:

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

14	unerlaubtes Feld

Damit können wir ein Schachbrett komplett in den Speicher eines Computers übertragen, und das war auch Ziel dieses Kapitels.

2.4.2 Der Zuggenerator

Nachdem wir ja nun wissen, wie das Spielbrett bzw. die Figuren im Computer dargestellt werden, können wir uns jetzt dem ersten wirklichen Baustein eines Schachprogramms widmen, der noch dazu ein äußerst zeitkritischer ist: dem Modul, das die möglichen Züge in einer Stellung generiert, dem sogenannten Zuggenerator. Nach dem letzten Kapitel gibt es eigentlich zum Generieren der Züge nicht mehr viel hinzuzufügen - Züge entstehen durch einfache Additionen bzw. Subtraktionen, die erhaltenen Züge (Startfeld, Zielfeld, Art) werden in geeignete Listen eingetragen und dem Such-Modul übergeben. Dabei sagt die Art aus, ob es sich um eine Verwandlung handelt, und wenn ja, dann hat Art den Wert den entsprechenden Figur (z.B. 2 für eine schwarze Dame). Auch En-Passant Züge lassen sich auf diese Weise erkennen, mehr dazu aber in Kapitel 4.1, hier soll nur eine Übersicht geboten werden.
Was aber noch sehr wohl wichtig ist: der Zuggenerator ist nicht intelligent (zumindest derjenige, der im Salomon™ Verwendung findet), d.h. es werden alle möglichen Züge einer Stellung generiert, auch wenn dieser Zug u.U. gar nicht möglich ist (z.B. eine Rochade, während man im Schach steht, oder ein Schlagzug, der den gegnerischen König schlägt etc.). Der Grund, warum ich keinen intelligenten Zuggenerator programmiert habe, ist das damit verbundene Zeitproblem. Es bedeutet nämlich einen ganz beträchtlichen Aufwand z.B. festzustellen, ob eine Figur gefesselt ist (d.h. wenn sie weggezogen wird, ob der eigene König dann im Schach steht), oder ob denn nun eine Rochade erlaubt ist. Dieser Aufwand schlägt sich gerade beim Zuggenerator enorm in der Zeit nieder, weil er ja praktisch bei jedem Betrachten eines Zuges aufgerufen werden muß. Ob ein Zug auch tatsächlich ausgeführt werden kann, entscheidet daher die Suche (wird im nächsten Kapitel genauer behandelt), denn die Chance ist gerade bei Zügen, die nichts schlagen (sogenannte Ruhezüge) enorm groß, daß sie per Alpha-Beta Cut abgeschnitten werden, wir hätten also im Zuggenerator völlig umsonst Rechenzeit verschenkt!
Ebenso wird verfahren, wenn ein Spieler im Schach steht, auch dies erledigt die Suche einfach mit. Denn falls wir das in den Zuggenerator mitaufnehmen würden, ständen wir sogleich vor dem Problem, einmal sämtliche möglichen Züge zu betrachten, die das Schach verhindern würden (Figur vorstellen bzw. die schachgebende Figur schlagen). Liegt allerdings ein Doppelschach vor, dann hilft dies alles nichts, wir müssen einen Königszug machen. All dies ist extrem zeitaufwendig, und deswegen habe ich zu einer anderen Möglichkeit gegriffen. Schließlich kann die Suche doch relativ einfach feststellen, ob eine Stellung illegal ist, d.h. es werden einfach alle Schlagzüge des Gegners betrachtet, und schlägt einer davon den eigenen König, dann wird ein Wert vergeben, der gleich einem Matt ist, und die Sache ist erledigt. Keine aufwendige Fesselungssuche, keine Doppelschachfeststellung, wenig neuer Programmieraufwand (lediglich eine Routine mehr). Die Suche muß sich die Züge ohnehin ansehen, und so wird lediglich die Suche ein wenig aufwendiger, der Zuggenerator bleibt aber schnell. Folgende Routinen sollte übrigens i.a. ein Zuggenerator enthalten:

- Generierung von Damenzügen
- Generierung von Läuferzügen
- Generierung von Springerzügen
- Generierung von Bauernzügen
- Generierung von Königszügen
- Generierung von En-Passant Zügen
- Generierung von Rochaden

Auch der Zuggenerator von Salomon™ besteht aus obigen Routinen, etwas genauer werden sie dann im Kapitel 4.1 beschrieben, nun wollen wir aber zum nächsten Baustein eines Schachprogramms schreiten, der eine zentrale - da steuernde - Rolle spielt.

2.4.3 Die Suche

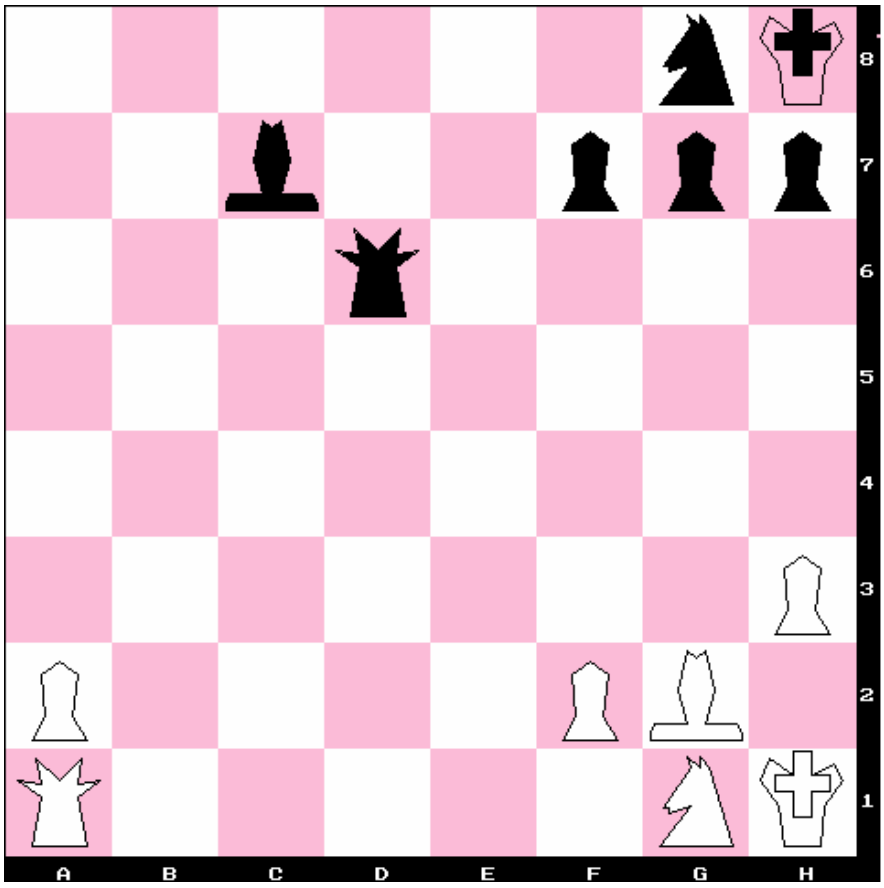
Die Suche ist einer der kritischsten Teile eines Schachprogramms. Denn hier fällt die Entscheidung, welche Züge bzw. Varianten das Programm untersuchen soll, bzw. welchen Zug der Computer dann letztenendlich spielen soll. Selbstverständlich kann die Suche nur funktionieren, wenn auch die anderen Bausteine vorhanden sind, von denen wir einen im letzten Kapitel schon etwas genauer kennengelernt haben, vom dritten Baustein, der Bewertung, wissen wir nur, daß er nach gewissen Richtlinien einen Wert berechnet, der für die aktuelle Stellung repräsentativ ist. Wir wollen, wie auch im Kapitel über die Minimaximierung, von einer symmetrischen Bewertungsfunktion[8] ausgehen. Für das Verständnis dieses Kapitels ist es also nicht so wichtig zu wissen, wie die Bewertung im einzelnen funktioniert, sondern was sie leistet, das genügt schon.
Das Modul der Suche an und für sich besteht (zumindest bei Salomon) aus einer rekursiven Prozedur, die sowohl für Weiß als auch für Schwarz geeignet ist (dafür sorgt ein Schalter). Prinzipiell hat die Suche die Aufgabe Züge, die sie vom Zuggenerator erhält, probeweise auszuführen, diese neue Stellung dann bewerten zu lassen, und sie dann wieder zurückzunehmen.
Wichtige Teile der Suche haben wir uns ohnehin schon erarbeitet, dazu zählt die Minimaximierung und der Alpha-Beta Algorithmus, letzterer diente ja dazu, einer Stellungsexplosion entgegenzuwirken, allerdings funktionierte er nur dann richtig, wenn man die Züge in der richtigen Reihenfolge betrachtete, und genau das ist eine der ersten Hauptaufgaben der Suche: geeignetes Sortieren.
Sortieren kann auf verschiedene Weisen geschehen, einerseits kann man nach heuristischen Kriterien die Züge auswählen, oder man kann wirklich sortieren, d.h. eine Art Vorbewertung ist nötig.
Die erste Methode ist sicherlich interessant, denn sie bietet den Vorteil einer gewissen Zeitersparnis - doch der Nachteil kann natürlich sein, daß wir u.U. nur wenige

[8] Zur Erinnerung : diese liefert für gute Stellungen von Weiß positive Werte, negative für Schwarz und Null für ausgeglichene Stellungen.

Cut Offs zusammenbekommen. Dennoch möchte ich hier kurz darauf eingehen, nach welchen Kriterien man Züge auswählen kann (dies könnte man natürlich auch in einem Shannon Typ-B Programm verwenden - je nach noch vorhandener Zeit werden Züge aller Kriterien bzw. wenn die Zeit knapp zu werden droht, nur mehr Züge der Kriterien mit der höchsten Priorität gewählt). Hier eine mögliche Liste:

1. Züge aus der Hauptvariante
2. Schlagzüge, die die zuletzt gezogene Figur schlagen
3. Killerzüge
4. Sonstige Schlagzüge
5. Sonstige ruhige Züge

Das erste Charakteristikum ist natürlich das beste, denn in der Hauptvariante speichert die Suche jene Züge, die sich bis in eine gewisse Tiefe bewährt haben, also ist die Wahrscheinlichkeit nicht unbedingt gering, daß dies auch in der nächsten Tiefe der Fall sein wird. Auch das zweite Unterscheidungsmerkmal ist aus "dem Leben" gegriffen, denn wenn eine Figur geschlagen wird ist häufig der nächste Zug einer, der zurückschlägt[9] , somit ist dieser der beste im Moment. Was bedeutet wohl der Begriff Killerzüge? Dazu sollten sie einmal folgende Stellung betrachten:



[9] wenn der nächste kein solcher ist, verliert man i.a. ja eine Figur - und das sollte der eher seltenere Fall sein

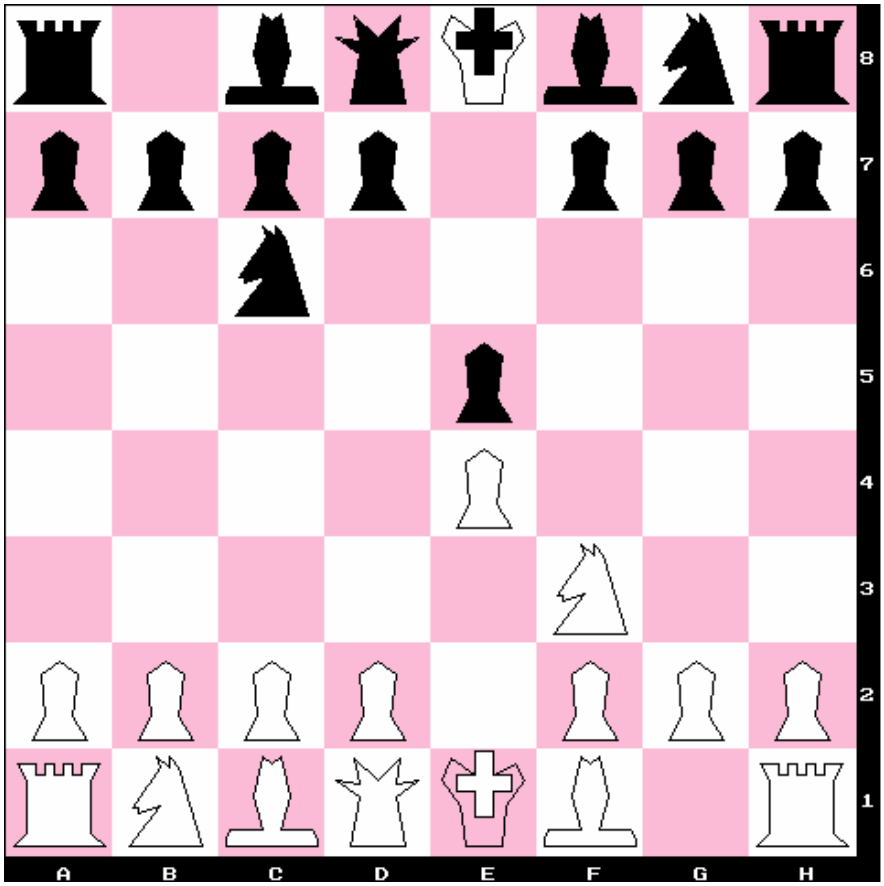
Weiß ist am Zug und muß dringendst etwas gegen Dh2++ unternehmen. In der Tiefe 1 werden von der Suche nacheinander alle möglichen Züge untersucht, sagen wir es wird a3 zuerst ausgeführt. Nun werden alle schwarzen Gegenzüge untersucht auch Dh2++ wird gefunden, spätestens jetzt läuten alle Alarmglocken - ein Matt droht. Ein Mensch würde ab sofort natürlich nur noch Züge untersuchen, die die drohende Niederlage noch irgendwie verhindern können, und auch von unserer Suche können wir diese Intelligenz erwarten. Es wäre also schön, wenn auf den nächsten getesteten Zug, sagen wir a4, nun sofort Dh2++ als erstes betrachtet würde - damit wäre nämlich a4 sofort widerlegt, und wie bräuchten keinen einzigen Gegenzug mehr untersuchen[10]. Das einzige was wir tun müssen ist also, Züge die sich in der Vorstellung bewährt haben, zu merken, sie werden dann in der nächsten Tiefe zuerst gewählt.
Als nächstes sollen dann die normalen Schlagzüge untersucht werden, denn diese stellen ja immer eine gewisse Instabilität im Spiel dar. Auch hier ist die Wahrscheinlichkeit groß, einen recht großen Gewinn zu verbuchen[11]. Zu guter Letzt werden diejenigen Züge untersucht, wo der geringste Punktezuwachs zu erwarten ist: die sogenannten Ruhezüge. Hier kann sich nur im positionellen Bereich etwas ändern, und da Programme keine Großmeister sind, wird die Position nicht so stark bewertet wie das Material (i.a. übersteigt die positionelle Bewertung nie den Wert eines Bauern - mit einigen Ausnahmen, wie wir im nächsten Kapitel sehen werden) - so auch bei Salomon™. Damit erreicht man auf ganz einfache Weise, daß das Programm auf seine Figuren achtgibt - langfristige Pläne, und die sind bei hohen positionellen Wertungen unumgänglich, kann es ohnehin nicht schmieden, denn dazu ist die maximale Suchtiefe viel zu gering.

Die zweite Methode ist das richtige Sortieren, wie es auch im Salomon™ implementiert ist (hier findet ein Quicksort Algorithmus Verwendung). Hierbei erspart man sich die Auswahl nach verschiedenen Zugmerkmalen, dies geschieht sozusagen implizit und wesentlich genauer, da ja jeder Zug einen gewissen Wert erhält, der Auskunft darüber gibt, wie erfolgversprechend er ist. Doch wie erhält man diese Werte? Hierzu ist eine sogenannte Vorbewertung nötig, die einmal alle möglichen Züge in einer Stellung probeweise ausführt und dann schaut, was tatsächlich dazugewonnen wird. Dabei werden nur materielle Kriterien, Schachgebote und Hauptvariante berücksichtigt - eine positionelle Bewertung wäre viel zu zeitaufwendig. Die Kehrseite der Medaille ist nun (um auf meine Bemerkung in Kapitel 2.3 zurückzukommen), daß erstens einmal das Sortieren auch Zeit kostet (O (n^2)) und zweitens die Ausführung der Züge irgendwie doppelt geschieht, denn die Suche muß anschließend ja auch die Züge probeweise ausführen. Dafür werden aber diese Dinge, die ich bei der vorigen Möglichkeit erwähnte, automatisch ausgeführt - und auch Schachgebote (die ja auch oft ein Matt

[10] Alpha Cut Off
[11] Nämlich dann, wenn man eine Figur gewinnen sollte

einleiten) werden blitzartig erkannt. Natürlich werden auch hier die Züge der Hauptvariante ganz nach vorne geschoben[12].

Doch damit sind die Aufgaben der Suche (noch lange nicht - ächz) getilgt! Betrachten sie einmal das nächste Diagramm:



Eine typische Eröffnungsstellung. Nehmen wir einmal an, das Programm sucht nur bis zur Tiefe 1 - na erraten sie schon, welchen Zug es dann spiele würde?! Sf3xe5 jedem Schachspieler unter ihnen werden sofort die Haare zu Berge stehen, denn der Bauer ist ja gedeckt - doch der Computer berechnet eben nur seine eigenen Züge, nicht die möglichen Erwiderungen, und dabei bringt eben Sf3xe5 den meisten Gewinn, einen Bauern nämlich. Daß der Springer dann auch verloren geht, sieht das Programm nicht - oder etwa doch? Versuchen sie doch einmal diese Stellung Salomon™ zu geben (Automatic Depth Increase OFF, look ahead level 1) - er macht unter Garantie nicht diesen Zug, aber auch beim Chessmaster 2100 läßt sich das schön testen, wenn sie die maximale Suchtiefe mit 1 begrenzen. Auch er wird sicher nicht den Unglückszug Sf3xe5 spielen[13].

---

[12] Bei Salomon bekommen sie einfach einen ziemlich hohen Wert zugewiesen

[13] Sollten sie dennoch ein Programm besitzen, das Sf3xe5 spielt, dann ab in die Mülltonne damit!

Irgend etwas scheint es also hier noch zu geben, daß dies verhindert, die Suche also mit weiterer Intelligenz ausstattet. Man nennt dieses etwas die sogenannte **Ruhesuche**. Meiner Meinung nach ist der Name höchst unglücklich gewählt, denn er gibt Anlaß zu argen Mißverständnissen. In der Ruhesuche werden nämlich mit Nichten die ruhigen Züge betrachtet, sondern ganz im Gegenteil: **nur** die **Schlagzüge**. Und genau aus diesem Grunde macht jedes vernünftige Programm den obigen Fehler auf keinen Fall. Die Bewertung darf nämlich nur sogenannte ruhige Stellungen bewerten[14]. Schlagzüge stellen immer einen gewissen Unruheherd dar (wie wir im letzten Diagramm ja festgestellt haben), den es zu beseitigen gibt. Also noch einmal: haben wir mit einer Stellung den Suchhorizont erreicht, so dürfen wir nur dann auch wirklich aufhören (sprich bewerten), wenn es keine Schlagzüge mehr gibt. Ansonsten wird eine Ruhesuche angeschlossen, die alle Schlagzüge durchgeht, bis eine ruhige Stellung erreicht wird, die wir schließlich bewerten dürfen.

Zwar sind Schlagzüge weit weniger häufig als die Ruhezüge, doch der Suchaufwand kann beträchtlich ansteigen (Spitzen bis Tiefe 23 habe ich schon erlebt), vor allem wenn es sehr viele Schlagmöglichkeiten gibt. Aber auch in der Ruhesuche kann der Alpha-Beta Algorithmus gewinnbringend eingesetzt werden, und zwar in Zusammenhang mit dem sogenannten **forward pruning**. Hier macht man sich folgende Tatsache zunutze: man bewertet zunächst einmal die Stellung, die am Anfang der Ruhesuche steht - diesen Wert hat der gerade am Zug befindliche Spieler schon ganz sicher, er muß ja den (die) Schlagzug (-züge) nicht ausführen! Nehmen wir weiter an, daß Weiß gerade am Zug ist, und der Wert ist bereits größer als Beta, also rechts außerhalb des Alpha-Beta Fensters, dann kann die Ruhesuche völlig entfallen, denn Schlagzüge würden den Wert nur noch weiter aus dem Fenster treiben[15], würden also ohnehin später abgeschnitten werden! Ist der Wert größer als Alpha, dann können wir zumindest Alpha auf diesen Wert setzen und in der Folge das normale Alpha-Beta Cut Off anwenden.

Der Name forward pruning stammt daher, daß unser Cut Off nicht auf tatsächlichen Untersuchungen (wie der Alpha-Beta Cut Off) beruht, sondern auf einer Vorausüberlegung, und ist daher auch nicht ganz mit dem Alpha-Beta Cut Off verträglich. Darauf will ich etwas später noch genauer eingehen.

Vorerst will ich noch eine weitere Möglichkeit des Cut Offs vorstellen, die ebenfalls in der Ruhesuche (aber nicht in der normalen Suche!) angewandt werden kann, auch diese Möglichkeit fällt unter die Kategorie des forward pruning. Nehmen wir einmal an, die Ruhesuche erreicht eine Stellung (Weiß am Zug), die den Wert -500 erhält, Alpha und Beta sollen zu diesem Zeitpunkt auf 0 und 200 stehen. Sollen wir nun einen Zug untersuchen, der einen schwarzen Bauern[16] schlägt? Die Antwort ist: Nein! Der Stellungswert könnte maximal auf -300 (100 Punkte der Bauer + 100 Punkte von der positionellen Bewertung) verbessert werden, im Alpha-Beta Fenster

---

[14] deswegen spricht man auch von einer Ruhesuche, d.h. Schlagzüge werden solange verfolgt, bis die Stellung ruhig ist, also keine Schlagzüge mehr möglich sind.

[15] Klar, denn Weiß wird wohl nicht freiwillig einen Zug wählen, der seine Stellung verschlechtert, also wieder näher an Beta treibt

[16] Der Wert eines Bauern ist bei Salomon 100 Punkte, genaueres dann im nächsten Kapitel

aber landen wir gewiß nicht. Also ersparen wir uns an dieser Stelle die weitere Verfolgung dieser Variante, und schätzen den Zug einfach ab (in diesem Fall bekommt er also z.B. den Wert -300), damit haben wir wieder einiges gespart!
Leider ergibt sich in der Ruhesuche auch ein Problem, mit dem ich bei Salomon sehr sehr lange gekämpft habe, erst sehr spät gelang es mir schließlich, doch noch eine Lösung zu finden. Die Schwierigkeit ist die folgende: da in der Ruhesuche **keine ruhigen Züge** (jetzt sieht man einmal, wie verwirrend der Begriff ist!) untersucht werden, hat man ein ziemliches Problem bei der Feststellung eines Matt. Meistens kann ein Matt (bzw. Schachgebot) ja durch einen ruhigen Zug (sei es nun durch wegfahren oder auch nur eine Figur vorstellen) verhindert werden, in der Ruhesuche, wie wir sie aber bisher kennengelernt haben, kann es aber zu einer äußerst folgenschweren Fehlentscheidung kommen und zwar genau dann, wenn ein Schachgebot nicht durch einen Schlagzug aufgehoben werden kann. In solch einem Fall wird die Stellung als Matt erkannt!! Und das führt i.a. zu einer mittleren Katastrophe. Das Programm versucht nun natürlich mit allen Mitteln der vermeintlich drohenden Niederlage entgegenzuwirken - haarsträubende, völlig sinnlose Opfer sind meist die Folge. Es droht ja Matt, also muß man u.U. schon eine Dame opfern. Was glauben sie, wie oft ich mir die Haare raufte (sie sind mittlerweile grau - aber damals habe ich mir wenigstens den Friseur erspart), wenn Salomon schon wieder scheinbar völlig sinnlos Figuren opferte, bis es mir endlich gelang, den wahren Grund für diese Verzweiflungstaten zu finden. Salomon war einfach nicht in der Lage, in der Ruhesuche ein Matt korrekt zu beurteilen - eine Katastrophe! Schließlich arbeitete ich ein Konzept aus, wobei ich aber den argen Fehler beging, mir zuwenig Zeit zu nehmen und klarerweise wurde mir sofort die Rechnung präsentiert: Salomon spielte noch mieser als vorher.
Nach vielen Stunden Programmierarbeit und viel Verlust an Gehirnschmalz, gelang es mir dann doch, eine zufriedenstellende Lösung zu finden. Zuerst war es einmal wichtig zu wissen, ob sich die Suche überhaupt schon in der Ruhesuche befand, wenn nicht war ein Matt eindeutig feststellbar (dazu hat man ja schließlich auch die Suche) und ich war fertig. Befand man sich allerdings gerade in der Ruhesuche, dann mußte einiges mehr geschehen. Nehmen wir an, Weiß gibt mit einem (Schlag-) Zug Schwarz ein Schach, nun liegt es am Schwarzen festzustellen, ob er denn Matt ist, oder ob die Sache vielleicht doch harmloser ist. Dazu muß der Schwarze zunächst einmal den Zuggenerator aufrufen, um alle seine möglichen Züge in der aktuellen Stellung zu erhalten (selbstverständlich auch die Ruhezüge!). Als nächsten Schritt führt er eine Bewertung seiner Stellung durch, den erhaltenen Wert merkt er sich, denn er ist insofern wichtig, als daß er die (im Moment) oberste Schranke darstellt, was der Schwarze überhaupt (noch) erreichen kann, falls tatsächlich ein Matt vorliegt, dann wird er diesen Wert natürlich nicht mehr erreichen. Ist die Drohung aber harmlos, vielleicht aber doch. Also wird der Schwarze nun der Reihe nach alle seine Gegenzüge (wieder nach oben erklärter Vorbewertung) betrachten und erstens schauen, ob der aktuelle Wert größer oder gleich dem gemerkten Wert ist - wenn ja, kann er aufhören, es liegt kein Matt vor, Schwarz hat den bestmöglichen Wert schon gefunden -, ist dies nicht der Fall, so kann er zumindest noch schauen, ob der Wert des gerade untersuchten Zuges seinen aktuellen Wert (den er in einer eigenen Variablen mitführt und zu Beginn mit dem

denkbar schlechtesten Wert, also einem Matt für Schwarz, initialisiert wurde) verbessert. Wenn ja, darf er den aktuellen Wert durch den Zug Wert ersetzen. Hat Schwarz alle Züge untersucht, und sein mitgeführter aktueller Stellungs Minimaxwert hat noch immer den Wert eines Matts, dann (und **nur** dann) liegt tatsächlich ein Matt vor! Man sieht, der Aufwand in der Ruhesuche ein Matt festzustellen ist nicht gerade gering.
Jetzt wollen wir uns aber auf die letzte Aufgabe der Suche konzentrieren, der sogenannten Breitensuche[17]. Diese Suche ist dem eigentlichen rekursiven Modul übergeordnet, übernimmt also quasi die Steuerung. Aber nicht nur das, sie leistet noch mehr[18]!
Die normale Suchmethode eines Schachprogramms ist ja die Tiefensuche, wie wollen aber Tiefen- und Breitensuche kombinieren, und damit die Effizienz der Suche noch um ein weiteres Maß steigern. Nehmen wir einmal an, wir hätten eine maximale Suchtiefe von 4 eingestellt, so würde nach der Tiefensuche das Programm sofort einmal bis Tiefe 4 rechnen. Dies ist aber nicht unbedingt gut, denn wertvolle Informationen gehen dabei verloren, die gewinnbringend genutzt werden könnten. Es ist viel besser, wir lassen unser Programm (auch Salomon™ und jedes mir bekannte Schachprogramm arbeitet nach diesem Prinzip) zuerst einmal bis Tiefe 1, dann bis Tiefe 2 usw. rechnen[19], also eigentlich in die Breite, innerhalb der Breite wird aber in die Tiefe gerechnet. Wir inkrementieren also die maximale Suchtiefe nach jedem Durchlauf um eins, bis wir jene Tiefe erreicht haben, die der Spieler als maximale Tiefe vorgegeben hat (oder bis die Zeit ausgeht). Sofort könnte man einwenden, daß man ja jetzt die ganze Arbeit doppelt oder sogar dreifach macht - dies ist aber ganz und gar nicht richtig! Oben sprach ich schon von wertvollen Informationen, die bei einer sofortigen Tiefensuche bis zum Suchhorizont verlorengehen würden, ohne mich aber genauer zu konkretisieren. Mit jeder Suche erhalten wir Informationen, die wir für die nächste Iteration sinnvoll verwenden können. So bekommen wir z.B. mit jeder Suche eine neue Hauptvariante, für die wir gleich einmal sorgen, daß sie in der nächsten Suchtiefe zuerst behandelt wird (eine Flut von neuen Alpha-Beta Cut Offs ist die Folge). Außerdem erhalten wir einen Minimaxwert, von dem zu erwarten ist, daß er sich auch in der nächsten Suchtiefe nicht viel verändern wird[20], wir schätzen also einfach das Alpha-Beta Fenster ab! Das geschieht so, daß wir uns einen geeigneten Wert, wollen wir ihn (wie auch im Salomon) mit MaxGewinn bezeichnen, überlegen und Alpha auf Minimaxwert-MaxGewinn und Beta auf Minimaxwert+MaxGewinn setzen. Mit diesem eingeengten Alpha-Beta Fenster erhalten wir natürlich viel mehr Cut Offs, als mit einem weiten Fenster - man gewinnt die scheinbare Mehrarbeit, die man durch diese Form der Suche leistet, also mehrfach in Form von unzähligen Cut Offs wieder zurück. Und je kleiner man MaxGewinn wählt, desto mehr Cut Offs werden

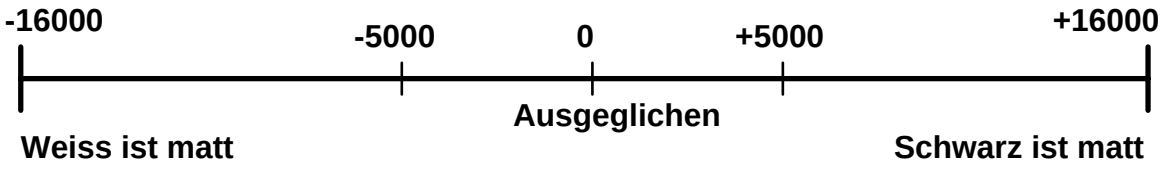
---

[17] wird in der Literatur auch manchmal als "Iterative Suche" bezeichnet
[18] Sollten sie mit dem Begriff Breitensuche nichts anfangen, im Kapitel 2.1 ging ich kurz darauf ein
[19] Hiermit wird auch der Name "Iterative Suche" ein wenig klarer
[20] Falls wir da aber auf ein Matt stoßen, haben wir Pech gehabt!

gemacht, doch damit wächst auch die Wahrscheinlichkeit, daß ein guter Zug übersehen wird, in solch einem Fall erhält die Breitensuche (ich habe schon in der Fußnote eine derartige Andeutung gemacht) einen Minimaxwert, der außerhalb des Alpha-Beta Fensters liegt. In so einem Fall kann leider keine Aussage getroffen werden, es bleibt nichts anderes übrig, als die Suche mit dem weiten Alpha-Beta Fenster zu wiederholen, nur so kann der Zug gefunden werden. Dieser Fall tritt aber nur recht selten ein (z.B. wenn ein Matt gefunden wird, oder eine Figur verloren geht bzw. gewonnen werden kann), und somit ist der Mehraufwand im durchschnittlichen Fall vollkommen gerechtfertigt!
Nun ist die allgemeine Betrachtung, die zum Verständnis dieses Bausteins notwendig war, abgeschlossen, und wir wollen zum nächsten Baustein schreiten, der Bewertung.

2.4.4 Die Bewertung

In diesem Kapitel wollen wir uns mit jenem Modul vertraut machen, das die eigentliche Intelligenz eines Schachprogramms ausmacht, der sogenannten Stellungsbewertung. Hier wird nach gewissen Richtlinien, auf die wir sofort genauer eingehen, entschieden, wie gut oder auch wie schlecht die augenblickliche Stellung für die jeweilige Farbe ist. Dabei wollen wir eine sogenannte *symmetrische Bewertungsfunktion* betrachten, also ein Modul das sowohl von Schwarz als auch von Weiß aufgerufen werden kann - ist Weiß gerade im Vorteil, so wird ein positiver Wert zurückgeliefert, ist Schwarz im Vorteil, ein negativer.
Als erstes wollen wir einmal klären, in welcher Größenordnung die Werte der Bewertungsfunktion sein sollen, betrachten sie einmal folgendes Diagramm:



Extremwerte sollen also für Mattstellungen vergeben werden - wobei die Mattfeststellung **nicht** Aufgabe der Bewertung ist, sondern die der Suche (s.o.). Der Baustein, der hier betrachtet wird, liefert also nur Werte zwischen + /- 5000. Wie macht er das nun?
Zunächst wollen wir die Bewertung einmal in eine **materielle** und eine **positionelle** Komponente unterteilen. Wobei sich die materielle Bewertung vorwiegend um die vorhandenen Figuren auf beiden Seiten kümmert, stellen sich bei der positionellen Bewertung die folgenden Fragen: (a) Welche schachtheoretischen Kenntnisse werden benötigt, damit ein Computer eine Stellung bewerten kann und (b) wie stellt man dieses Wissen in Form von Algorithmen dar.
Beginnen wir also mit der **materiellen Bewertung**. Jeder gute Schachspieler wird in der Frage, was die einzelnen Figuren "wert" sind, äußerst gut Bescheid wissen (so z.B. wird ein Meister dieses Spiels unter normalen Umständen niemals zwei

Türme gegen eine Dame tauschen, oder zwei Leichtfiguren gegen einen Turm), auch wir wollen unserem Programm dieses Wissen in irgendeiner Form vermitteln, und es liegt nahe, es mit Zahlenwerten zu tun. Die Werte der einzelnen Figuren sind in der nachfolgenden Tabelle zusammengefaßt:

Figurentyp	Wert
König	-
Dame	900
Turm	500
Läufer	325
Springer	325
Bauer	100

Diese Werte finden auch in Salomon™ Verwendung, und sind durchaus üblich. Auch in anderen Schachprogrammen werden sie ähnliche Werte finden, wenngleich dort statt 900 z.B. 9 verwendet wird (Chessmaster 2100), das Ergebnis ist aber letztenendlichs dasselbe. Auch wird das Programm so nun nicht mehr zwei Türme gegen eine Dame tauschen etc., wie man sich durch einfaches aufsummieren leicht überzeugen kann. Und genau das macht auch die materielle Bewertung, es werden einfach die Figurenwerte der beiden Seiten aufsummiert und mit den erhaltenen Werten wird eine sogenannte **Materialbalance** gebildet, das funktioniert so:

Materialbalance := Materialsumme Weiß - Materialsumme Schwarz

Damit wird erreicht, daß bei einem Vorteil von Weiß ein positiver Wert und bei einem Vorteil von Schwarz ein negativer Wert zurückgeliefert wird - also genau das, was wir eigentlich wollten. Manche haben sich vielleicht gewundert, warum der König keinen Materialwert bekommen hat - nun der König ist eine Figur, die im ganzen Spiel ohnehin nicht geschlagen werden darf, vergäben wir einen Wert für den König, würde er bei Bildung der Materialbalance wieder herausfallen, da er ja bei beiden Seiten stets konstant bleibt, somit können wir uns das von vornherein sparen.
Material ist nicht das ganze Spiel über gleich viel wert, so kann z.B. ein schwarzfeldriger Läufer völlig umsonst sein, wenn alle gegnerischen Figuren (insbesondere Bauern) auf weißen Feldern stehen, in diesem Falle könnte man einen beträchtlichen Abzug am Figurenwert des Läufers vornehmen. Aber auch Bauern haben im Endspiel eine zentrale Rolle, gilt es doch als vornehmliches Ziel, wenigstens einen, sofern möglich[21], in eine Dame zu verwandeln. Den Figurenwert eines Bauern kann man durch einen speziellen Bonus erhöhen. Dieser soll genau dann vergeben werden, wenn ein Spieler einen materiellen Vorteil hat, und sich schon im Endspiel befindet (wie man das feststellt wird weiter unten geklärt), und zwar nach folgender Formel:

Bonus := Minimum (Bauernanzahl des Spielers , 3)*10

[21] es könnte ja auch ein Patt drohen!

Die Minimumsbildung ist dazu gut, daß der Bonus in gewissen Grenzen gehalten wird. Der Bonus sorgt nun dafür, daß die Bauern der einen Seite jetzt praktisch mehr wert sind, und nur noch gegen gegnerische Bauern getauscht werden, wenn die Verringerung des Bonus durch eine steigende positionelle Bewertung kompensiert wird.
Damit haben wir die materielle Bewertung auch schon abgeschlossen, daß dies natürlich noch lange nicht alles sein kann, ist klar. Denn wenn unser Programm nur materiell denken würde, dann wären z.B. in einer Eröffnung die Züge h4 und e4 völlig gleichwertig (keine Änderung vom Material her), auch so grundlegende Regeln wie Zentralisierung eines Springers und möglichst rasche Rochade würden völlig unter den Tisch fallen. Gerade für diese Dinge sorgt die sogenannte **positionelle Bewertung**.
Das Hauptaugenmerk liegt hier darin, aus der aktuellen Stellung sogenannte Stellungsmerkmale (Indizien) herauszusuchen, die auf mögliche Vor- bzw. Nachteile einer der beiden Seiten schließen lassen. Für einen Meister des Schachspiels existiert natürlich eine große Vielzahl an diesen Indizien, viele beruhen aber einfach auf Intuition, und lassen sich somit gar nicht programmieren. Außerdem stehen wir auch unter einem enormen Zeitdruck, d.h. wir müssen viel mehr Stellungen betrachten, als z.B. ein Großmeister es tun würde, und genau aus diesem Grunde konnte ich auch im Salomon™ nicht alles implementieren, was ich nun hier an Bewertungsmöglichkeiten vorstellen werde. Die Berechnungen würden einfach zu lange dauern, und das müßte mit einer weitaus geringeren Suchtiefe teuer bezahlt werden.
Vorderhand wollen wir erst einmal eine Einteilung der verschiedenen Stellungsklassen treffen, wie wir sie ja schon ungefähr im Diagramm der Einleitung getroffen haben:

- Entschiedene Stellungen (Matt, Patt, Remis)
- "fast" entschiedene Stellungen (positionelle Bewertung für eine Mattführung)
- sonstige Stellungen

Die Bewertung von **entschiedenen Stellungen** soll vollkommen vom Suche-Baustein durchgeführt werden, hier ist die Feststellung eines Matts, Patts, Remis recht einfach, da ja ohnehin alle Züge geprüft werden. Folgende Werte werden vergeben : +16000 für ein Mattsetzen von Schwarz, wird Schwarz in n Zügen mattgesetzt, dann wird ein Wert von 16000-n vergeben. Wird Weiß mattgesetzt, so wird analog ein Wert von -16000 vergeben, bzw. -16000+n, falls es in n Zügen geschieht (dazu muß das Programm natürlich mindestens n+1 Züge vorausrechnen[22]). Die Feststellung eines Patts kann einfach dadurch erfolgen, daß man sich vor der Suche der Gegenzüge einer Stellung merkt, ob man gerade im Schach steht, bekommt man dann nach der Untersuchung einen Minimaxwert, der ein Matt ankündigt, braucht man nur mehr zu schauen, ob man vorher im Schach war - wenn ja ⇨ Matt, wenn nein ⇨ Patt. Die Feststellung eines Remis übernimmt im Salomon™ eine spezielle

[22] n+1 deswegen, weil bei einem Matt auch noch alle gegnerischen Züge geprüft werden müssen

Hilfsroutine. Hier wird sowohl auf klassisches Remis (nur mehr zwei Könige) als auch auf technisches Remis (eine Seite hat zuwenig Material, um ein Matt zu forcieren) getestet. Falls der Test positiv verläuft, wird der Stellung selbstverständlich der Wert Null zugewiesen.

Fast entschiedene Stellungen sollen aber sehr wohl in der Bewertung berücksichtigt werden, wenngleich hier auch eine Sparversion des gesamten Moduls ausreicht. Klären wir einstweilen erst einmal, wann überhaupt dieses spezielle Sparmodul aktiviert werden soll. Drei Punkte sind hierbei zu beachten:

1. Die Materialbalance liefert einen Wert, dessen Betrag mindestens dem Wert eines Turmes entspricht (sonst Remis)
2. Die Materialsumme des mattzusetzenden Spielers übersteigt nicht den Wert von zwei Leichtfiguren (sonst Verteidigungsmöglichkeiten zu groß)
3. Es sind in der Stellung keine Bauern mehr vorhanden. (Forderung für beide Seiten, einerseits soll das Verteidigungspotential nicht mehr erhöht werden können, andererseits ist es einfacher, zuerst die vorhandenen Bauern umzuwandeln und dann erst den Mattangriff zu starten)

Um die Mattführung erfolgreich durchführen zu können, müssen wir nun zwei Ziele verfolgen, die auch in der Regel von Menschen beachtet werden. Zum einen müssen wir danach trachten, den gegnerischen Monarchen möglichst weit vom Zentrum abzudrängen (denn nur am Rand ist er i.a. auch mattsetzbar), zum anderen muß der eigene Monarch dem Gegner einigermaßen dicht auf den Fersen bleiben (um erstens ein Geschlagenwerden der Figur zu verhindern und zweitens der Figur zu helfen, Fluchtwege abzuschneiden. Um das erste Ziel verwirklichen zu können, brauchen wir also nur den Abstand des gegnerischen Monarchen zur Brettmitte (dies geschieht im wesentlichen dadurch, daß wir uns Reihe und Spalte berechnen, diese dann von der Brettmitte (4.5,4.5) abziehen und die Absolutbeträge aufsummieren - genaueres im Kapitel 4.3) auszurechnen und mit einem entsprechenden Wichtungskoeffizienten zu versehen (Wichtungskoeffizienten sind einfach Multiplikatoren, deren Betrag aussagt, wie stark der aktuelle Wert in die Bewertung eingehen soll - Salomon™ ist in dieser Hinsicht völlig variabel, d.h. sämtliche Wichtungskoeffizienten können vom Spieler willkürlich gesetzt und gespeichert werden. Somit ist der Spielstil des Programms stark beeinflußbar). Für das zweite Ziel berechnen wir einfach den Abstand zwischen den beiden Königen und subtrahieren ihn vom maximal möglichen Abstand der beiden Monarchen. Damit haben wir erreicht, daß die positionelle Bewertung zunimmt, wenn der Abstand der Könige abnimmt. Bleibt nur noch zu sagen, wie hoch wir diese Kriterien gewichten wollen: die Standardwerte, die im Salomon™ verwendet werden sind 9.4 für das Abdrängen des Monarchen und 1.6 für den Abstand der beiden Monarchen, somit wird das Abdrängen also zum vordergründigen Ziel erklärt. Nun sind auch die fast entschiedenen Stellungen abgeschlossen.
Die sogenannten **restlichen Stellungen** werden uns nun aber ein wenig länger beschäftigen. Eigentlich lassen sich diese noch einmal unterteilen, und zwar nach dem

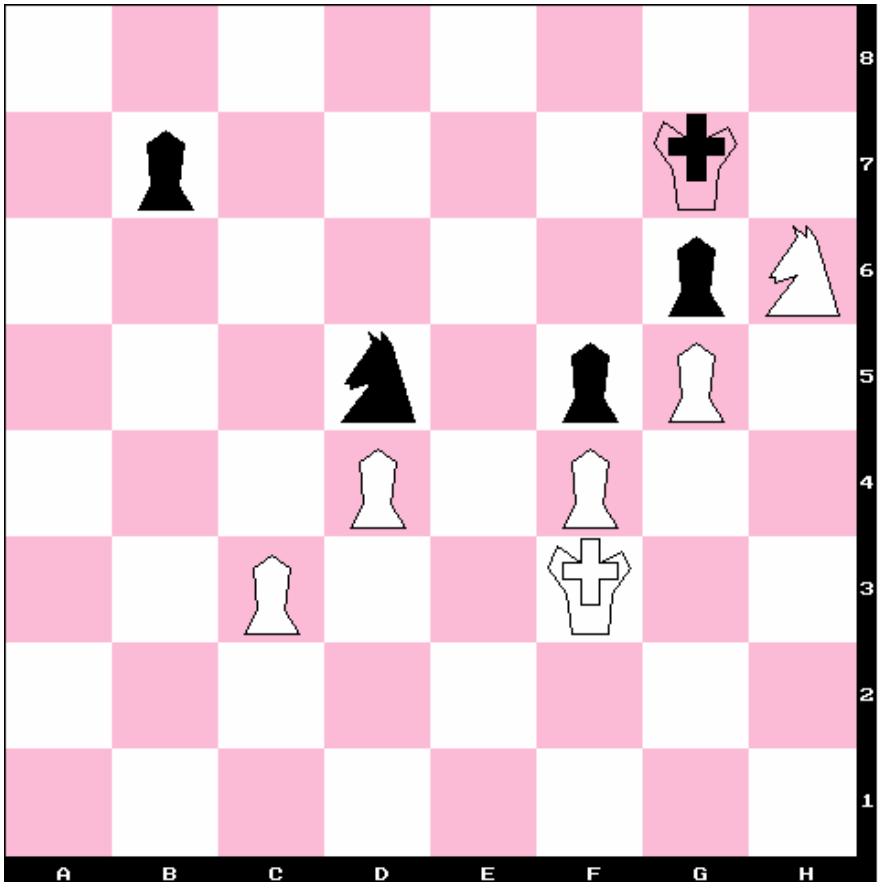
Partiestadium. Hier wäre einmal die Eröffnung zu nennen, in der beide Spieler probieren, ihre Figuren zu entwickeln und das Zentrum zu erobern. Die Eröffnung ist in der Regel abgeschlossen, wenn beide Seiten rochiert (die Rochade soll bezwekken, den König in die Nähe eines geschützten Eckfeldes zu bringen, wo die Verteidigungsmöglichkeiten besser sind und wo er seinen eigenen Figuren nicht im Wege steht) haben, und alle Figuren bis auf den König und die beiden Türme, die Grundlinie verlassen haben. Aber auch hier gibt es Ausnahmen, die sehr stark von der gewählten Eröffnungsvariante abhängen. So können gewisse Figuren erst sehr spät ins Geschehen eingreifen (z.B. steht bei der Pirc-Ufimzef Verteidigung vorwiegend das in Sicherheitbringen des schwarzen Monarchen im Vordergrund gegenüber einem möglichst raschen Entwicklungsausgleich), bzw. auch mehrmals gezogen werden. Der Übergang ins sogenannte Mittelspiel kann also auch fließend erfolgen.
Das Mittelspiel ist dann von beiden Seiten geprägt, einen positionellen oder sogar einen materiellen Vorteil zu erringen (so kann man z.B. schon im Mittelspiel solche Bauernstrukturen schaffen, für die eine hohe Wahrscheinlichkeit besteht, daß man sie dann später im Endspiel bis zur Verwandlung durchbekommt), seltener ist das Ziel, den gegnerischen König direkt mattzusetzen. Diese Bestrebungen setzen sich aus **taktischen** und aus **strategischen** Kampfelementen zusammen. Durch taktische Züge unternimmt man den Versuch, Material zu gewinnen (oder gar die Partie) ohne daß es dem Gegner gelingt, dies zu verhindern. Eine solche Folge von taktischen Zügen nennt man **Kombination**. Häufig werden solche Kombinationen durch die Opferung eigenen Materials eingeleitet, wobei dann soviele Drohungen auf das gegnerische Material gewonnen werden, daß sie der Gegner nicht mehr alle parieren kann. Man gewinnt also sein Opfer mit Zinsen zurück! Und genau das ist die Stärke von Computerprogrammen (natürlich auch die von Salomon™). Denn gerade der Computer rechnet die Schlagkombinationen besonders genau durch und analysiert die möglichen Züge so tief als es nur geht (Zeitschranke), und der Erfolg der Kombinationen läßt sich einfach an der Materialbalance ablesen - somit gleich einmal ein Tip am Rande, falls genau sie immer gegen ihr Schachprogramm verlieren: lassen sie sich auf keine taktischen Verwicklungen ein, spielen sie schön ruhig, versuchen sie möglichst langfristige Pläne zu fassen, und retten sie sich wenn möglich ins Endspiel, so werden sie die meisten Programme schnell in Verlegenheit bringen!
Doch die Erfolgsaussichten hängen in den meisten Stellungen - getreu dem Motto: "Ohne Fleiß kein Preis!" - vom strategischen Geschick der beiden Kontrahenten ab. Denn gute Kombinationen kann man in der Regel erst dann anbringen (von taktischen Fehlern des Gegners jetzt einmal abgesehen), wenn man sich mit viel Ausdauer und Zähigkeit zunächst einmal Angriffsmöglichkeiten geschaffen hat. Es muß versucht werden, günstige Aufstellungen für die eigenen Figuren zu finden und ernsthafte Schwächen in der gegnerischen Formation zu provozieren (z.B. Doppelbauern, Springer am Rand etc.) bzw. zu erkennen. Genau auf diesem Gebiet haben alle Schachprogramme ihre Schwachstellen (wenngleich Salomon™ auch einige interessante Ansätze in dieser Richtung implementiert hat, wie z.B. Kontroll Zentrum Analyse - wird im Kapitel 4.3 näher erläutert), muß man doch meist auf heuristische Algorithmen zurückgreifen, welche oft zu lückenhaft oder einfach zu grob

bewerten und eine große Suchtiefe kann diese Schwäche meistens nur unzureichend kompensieren. Was wir dennoch an Möglichkeiten haben, werden wir gleich kennenlernen.
Fällt im Mittelspiel noch keine Entscheidung, so geht die Partie ins letzte Stadium, dem sogenannten Endspiel. Das Endspiel erkennt man i.a. daran, daß beide Seiten in materieller Hinsicht schon relativ reduziert sind. Auch bei Salomon stelle ich auf diese Weise fest, ob das Endspiel erreicht wurde. Und zwar ist dies genau dann der Fall, wenn die Materialsumme des Spielers ≤ 2000 ist (dies entspricht nach unseren Werten etwa zwei Türmen einem Läufer und sieben Bauern). Selbstverständlich müssen wir die Endspielfrage für beide Spieler getrennt beantworten, denn ein Merkmal des Endspiels ist es wohl, daß nun der König aktiv am Kampf mitwirken darf (das Material ist ja reduziert genug), doch dies darf nur dann geschehen, wenn der **Gegner** bereits im Endspiel ist (und nicht der Spieler selbst!), denn ansonsten würde der König ins offene Messer laufen!!
Häufig wird das Endspiel auch von demjenigen Spieler angestrebt, der einen materiellen Vorteil hat, denn gerade im Endspiel kann sich ein materieller Vorteil (besonders wenn es sich um Bauern handelt) u.U. weitaus stärker bemerkbar machen, als im Mittelspiel, zumal eine gewisse Vereinfachung in der Zugvielfalt vorliegt. Das Endspiel ist vor allem durch die Bauern geprägt. Hier wirken sich natürlich Fehler in der Bauernformation besonders stark aus (Doppelbauer, isolierter Bauer, rückständiger Bauer) - denn beide Seiten trachten danach sogenannte *Freibauern* zu bilden (das sind solche Bauern, die von keinem gegnerischen Bauern mehr gestoppt werden können - für deren Blockierung also wohl oder übel eine (oder sogar mehrere) Figuren herhalten müssen).

Bei einem guten Schachprogramm muß man also dafür Sorge tragen, daß die jeweiligen Wichtungskoeffizienten dem Partiestadium angepaßt sind. In der Regel wird bei der Eröffnung eine sogenannte Eröffnungsbibliothek verwendet (eine Datenbank, die alle gebräuchlichen Eröffnungen gespeichert hat). Das Programm braucht also nur die richtige Erwiderung zu suchen, Rechenzeit wird keine benötigt (der Chessmaster 3000 hat bei einigen Eröffnungen sogar bis zu 35 (!!) Halbzüge gespeichert). Beim Mittelspiel und beim Endspiel (sofern keine Endspieldatenbank vorhanden ist - bisher konnte ich noch nie gegen solch ein Programm spielen) ist das Programm dann auf sich gestellt.
Auch Salomon unterscheidet zwischen Mittel- und Endspiel, und dementsprechend verändert er eine Bewertung dynamisch.
Es gibt ein weiteres Merkmal, das bei einer strategischen Planung berücksichtigt werden müßte, den sogenannten *Stellungscharakter*. Hierbei sind die Zugmöglichkeiten beider Spieler ausschlaggebend, meist beeinflußt von der gegenseitigen Bauernstruktur. Normalerweise unterscheidet man hier zwischen *offenen* und *geschlossenen* Stellungen. Bei ersteren existieren in der Regel offene, d.h. durch keinen gegnerischen Bauern versperrte Linien, über die rasche Standortwechsel möglich sind. Dementsprechend kann sich in solchen Stellungen das Kampfgeschehen rasch verlagern, da die Figuren sehr kurzfristig ihre Angriffsziele ändern können. Ganz im Gegenteil dazu die geschlossenen Stellungen. Hier ist meist durch die Bauernformation bedingt, daß die sonst so langschrittigen Damen Türme und Läufer in ihren

Aktionsradien stark eingeengt sind (da durch Bauern behindert). Daher gestaltet sich auch eine Umgruppierung der Angriffsziele als äußerst schwierig, die Aufgaben der Figuren sind meist auf ein bis zwei Funktionen beschränkt (oft irgendwelche Bauern zusätzlich zu decken, sodaß der Gegner nicht eindringen kann). In solchen (endspielähnlichen) Stellungen darf der König natürlich früher aktiviert werden, da die gegnerischen Angriffsmöglichkeiten äußerst gering sind. Außerdem kommt Bauernzügen ein verstärktes Gewicht zu, denn sie bereiten oft die Öffnung von Linien vor, über die dann z.B. ein Doppelturm in das gegnerische Lager eindringen kann. Ein Programm tut sich aber mit solch positionellen Abschätzungen äußerst schwer, da hier ja meist langfristige Pläne Fuß greifen, und die mögliche Pointe in der Suche nicht mehr erkannt werden kann. So ist der Stellungscharakter eigentlich in keinem mir bekannten Schachprogramm berücksichtigt worden (von der enormen zusätzlichen Rechenzeit ganz abgesehen), und da es auch in der Literatur keine konkreten Vorschläge gab, verzichtete ich bei Salomon auf solch eine Art der Bewertung, die nur äußerst zweifelhafte Erfolge verspricht.

Die Berechnung der Materialbalance gab uns Auskunft über die generelle Schlagkraft der Figuren, und dieses Maß war völlig unabhängig von ihrem Standort. Es liegt daher nahe, in der positionellen Bewertung auf die Figurenstandorte einzugehen und deren Wirkungsradien zu berücksichtigen. Der Wirkungsradius der Figuren wird gleichzeitig von drei Komponenten beeinflußt, ich will ein Beispiel zur Bewertung von Freibauern anführen, an dem sich dies gut verdeutlichen läßt:



1. Komponente : *Bedeutung des Standortes selbst*

Je weiter ein Bauer vorgerückt ist, desto intensiver wird für den Gegner die Bedrohung und desto eher muß er sich etwas einfallen lassen (u.U. sogar Figuren opfern). Stünde beispielsweise der weiße Bauer nicht auf d4 sondern auf d7, so wären die schwarzen Kräfte (Springer und König) wesentlich stärker an den Bauern gebunden als jetzt, wahrscheinlich müßte Schwarz sogar den Bauern mit dem Springer blokkieren, was ihn natürlich für weitere Aktionen wertlos macht.

2. Komponente : *Gegenseitige Beeinflussung von Figuren beider Seiten*

Eine wichtige Maßnahme, den Vormarsch eines Freibauern zu unterbinden besteht darin, das vor ihm liegende Feld zu blockieren (oder wenigstens zu kontrollieren). In der Diagrammstellung wird diese Aufgabe vom schwarzen Springer auf d5 wahrgenommen.

3. Komponente: *Zusammenwirken der Figuren einer Farbe*

Ohne Unterstützung kann ein Freibauer nur in den wenigsten Fällen eine Umwandlung realisieren, es bedarf dazu meist einer Unterstützung von anderen Figuren - ideal ist natürlich eine Unterstützung von weiteren Bauern, die sich links bzw. rechts von ihm befinden. Eine solche "Bauernwalze" ist dann nur mehr sehr schwer (und meist nur mit großen Materialopfern) aufzuhalten. Im Diagramm wird der Freibauer auf d4 durch seinen Kollegen auf c3 unterstützt, der durch den Vorstoß c3-c4 den lästigen Kontrahenten auf d5 vertreiben kann.

Mit diesen Charakteristiken haben wir allerdings bisher nur eine sogenannte **statische Bewertung** erreicht - wir haben quasi eine Momentanaufnahme gemacht, selbst kurzfristige Änderungen werden auf diese Art nicht erkannt. Macht Weiß z.B. in der Folge den Vorstoß c3-c4 muß Schwarz wohl mit Sd5-e7 antworten. Mit den bisher kennengelernten Mitteln gilt nun der Freibauer als nicht mehr blockiert und wird dementsprechend hoch bewertet. Daß der Freibauer aber nicht ohne weiteres weiterziehen kann, wird erst in der Suche zwei Halbzüge tiefer erkannt (d4-d5 Se7xd5). Es liegt also nahe, zusätzlich zu den statischen Komponenten noch eine **dynamische Komponente** aufzunehmen, die zumindest die Zugmöglichkeiten in der aktuellen Stellung ein wenig mitberücksichtigt.
Einen praktischen Ansatz in dieser Richtung veröffentlichte im Jahre 1950 C.E. Shannon[23]. Um den dynamischen Charakter einer Stellung mit in die Bewertung einzubeziehen schlug Shannon vor, die legalen Zugmöglichkeiten beider Seiten einfach aufzusummieren und ihre Differenz gewichtet in die Gesamtbewertung miteinzubeziehen. Die Betrachtung von legalen (also den eigenen König nicht ins Schach bringenden) Zügen ist deshalb wichtig, um eventuelle Fesselungen zu bestrafen, da durch sie die sogenannte *Mobilität* verringert wird. Das Ziel dieser Bewertung war es, ein Maß für die Chancen der freien Kräfteentfaltung zu finden, da ja mit einer anwachsenden Zugmöglichkeit auch die Wahrscheinlichkeit steigt, eine strategisch gute Aufstellung zu erreichen (und evtl. eine Kombination anzubringen).
Wenn man diese Art der Bewertung allerdings etwas genauer betrachtet, ist die Idee zwar anerkennenswert, doch die Zusammenfassung aller Figuren (ohne Rücksicht auf Art und Feldregion) erscheint doch ein wenig zu grob. Und gerade diese Aspekte spielen keine unwesentliche Rolle. Zu den generell wichtigen Bereichen des Brettes, deren Kontrollierung durch Figuren erstrebenswert ist, gehören das Zentrum (d4, e4, d5, e5) und auch das Gebiet um die Königsstellung. Figuren ver-

[23] in seinen Artikeln SHA1, SHA2

größern durch die Beherrschung des Zentrums einerseits ihren Aktionsradius, andererseits können sie auch Angriffe auf den gegnerischen Königsflügel besser vorbereiten bzw. zum Schutz des eigenen Monarchen zurückeilen. Beispiele für einzelne bedeutsame Felder sind etwa die Felder von Freibauern (s.o.) oder auch die Standorte jener Figuren, die aus irgendwelchen Gründen (sei es nun wegen einer Fesselung, weil sie an einer Verteidigung mitwirken müssen oder weil sie sonst in einer Form blockiert sind) ihren Platz nicht verlassen dürfen.
Wesentlich ist auch der Einfluß des *Figurentyps*. So wird ein guter Schachspieler eher danach trachten, das Zentrum mit Bauern zu beherrschen (also den Figuren, die den niedrigsten Wert besitzen), als z.B. mit einer Dame, denn einerseits kann die Dame von jeder anderen Figur (natürlich mit Ausnahme des Königs) wieder vertrieben werden (deswegen ist es auch nicht sonderlich schlau, die so wertvolle Monarchin zu früh ins Kampfgeschehen eingreifen zu lassen) sofern die gegnerische Figur gedeckt ist (Ausnahme: Springer!) und andererseits müssen die Bauern ohnehin vorwärts streben und sind außerdem durch ihre äußerst bescheidenen Zugmöglichkeiten für statische Kontrollen besser geeignet. D.h. es ist eine nach dem Figurentyp gewichtete Kontrolle des Raumes gefragt - und zwar je wenig wertvoller eine Figur ist, desto besser ist es.
Bisher summieren wir einfach die Kontrollen, allerdings abhängig vom Figurentyp, auf. Erstrebenswert wäre es auch, sogenannte Zwischenergebnisse der Bewertung zu erhalten, also wie es um die Kontrollen jedes einzelnen Feldes steht. Diese Aussage wäre viel feiner, auch lassen sich somit Fragen wie "steht der König im Schach?" sofort beantworten . Die Datenstruktur ist recht einfach, man braucht lediglich 2x64 Listen zu je 16 Einträgen (zwei für Weiß und Schwarz, zu je 64 Feldern und 16 ist die maximale Anzahl der Figuren einer Seite), doch die Implementierung gestaltet sich äußerst aufwendig und zeitkostspielig. So müssen doch bei jedem probeweisen Ausführen eines Zuges der Suche die Felder upgedated werden, was sich als ziemlich lästig herausstellt. Aber die Listen bringen natürlich auch jede Menge Vorteile. Wie schon oben erwähnt, kann man die Frage des "im Schach stehen" sofort beantworten, man braucht lediglich in der dem Königsstandort gegnerischen Liste nachschauen, ob dort ein Eintrag vorhanden ist. Verwenden wir die Listen hingegen nicht, so bleibt uns nichts anderes übrig als entweder alle gegnerischen Züge zu untersuchen oder vom Königsstandort ausgehend alle für ein Schachgebot in Frage kommenden Felder nach gegnerischen Figuren abzusuchen[24].
Auch die Suche ließe sich ein wenig vereinfachen (aber nicht vollkommen ersetzen), da wir recht leicht feststellen können, ob beliebige Figuren angegriffen werden, oder ob sichere Fluchtfelder z.B. für eine angegriffene Dame existieren. Aber diese Frage ist nicht immer ganz zuverlässig beantwortbar, denn ein Fluchtfeld kann trotz einer gegnerischen Kontrolle sicher sein, die gegnerische Figur könnte ja gefesselt sein. Wir sehen, daß eine Benutzung von solchen Listen zwar die Suche nicht ersetzt, man daraus aber recht gute Vorinformationen gewinnen kann. Für mich war allerdings der Rechenaufwand zu groß und die gewonnene Information zu gering, deshalb verzichtete ich auf eine solche Art der dynamischen Bewertung.

[24] Dieses Verfahren findet auch im Salomon Verwendung

Dafür habe ich ein sogenanntes *Field Control Analysing* implementiert, das das eben gesagte auf etwas vereinfachte Weise berechnet, allerdings nur in ganz bestimmten Fällen. Mehr dazu im Kapitel 4.3.
Schreiten wir zu einem weiteren interessanten Stellungsmerkmal, der sogenannten **Aktivität**. Unter diesem Begriff faßt man alle *taktischen* Möglichkeiten zusammen, die in einer aktuellen Stellung gegeben sind. Zu diesen zählen sowohl Züge, die die Materialbalance beeinflussen als auch Angriffe auf den König (insbesondere besonders bedrohliche Abzugsschachs oder Doppelschachs). Diese aktiven Spielfortsetzungen geben zusammen mit dem beherrschten Raum ein Maß für die Initiative des Spielers an. Eine gute Abschätzung der Aktivität trägt also wesentlich zu einer präzisen Bewertung bei. Leider ist eine dementsprechende Bewertungsmethode gar nicht so leicht zu finden, nur zu oft müssen komplizierteste Sachverhältnisse geklärt werden (z.B. wie groß ist die Bedrohung des Königs nach einem Schachgebot oder wie stark ist der drohende Materialverlust, wenn eine Figur von mehreren gegnerischen Figuren angegriffen und gleichzeitig von mehreren Figuren verteidigt wird. Es müssen also meist ganze Zugfolgen betrachtet werden, eine Aufgabe, die eigentlich die der Suche ist. Im Jahre 1961 wurde zwar ein Algorithmus zur statischen Vorhersage von Abtauschfolgen von einem Mann namens M. Smith entwickelt[25], doch erstens ist der Rechenaufwand nicht unbeträchtlich und zweitens konnte das Verfahren auch zu Fehlurteilen führen, weswegen ich hier nicht mehr näher darauf eingehen möchte. Welche Verfahren nun tatsächlich im Salomon™ Verwendung finden, erfahren sie im Kapitel 4.3.

Wir wollen uns jetzt dem nächsten Schwerpunkt der Bewertung zuwenden, der **Figurenbewertung**. Sechs verschiedene Figurentypen gilt es zu betrachten, beginnen wir mit der

Bauernbewertung:

Bauern sind Figuren, die ihre Standorte nur in kleinen Schritten verändern können, ihre Formationen weisen daher recht dauerhafte Eigenschaften auf. Die Bewertung der Stärken und Schwächen von Bauernformationen ist also ein wesentlicher Faktor bei der Beurteilung einer Position. Sehr häufig wird dadurch der weitere Spielverlauf beeinflußt. Gute Schachspieler sind meist bemüht, ihre Angriffe gegen Schwachstellen der gegnerischen Bauernstruktur zu richten, auf der anderen Seite eignen sich starke Bauernketten auch hervorragend, um Lücken in die gegnerische Verteidigung zu schlagen, oder den König vor Attacken zu schützen. Shannons theoretischer Ansatz beinhaltet drei Punkte, die in eine Bauernbewertung aufgenommen werden sollten, es sind dies:

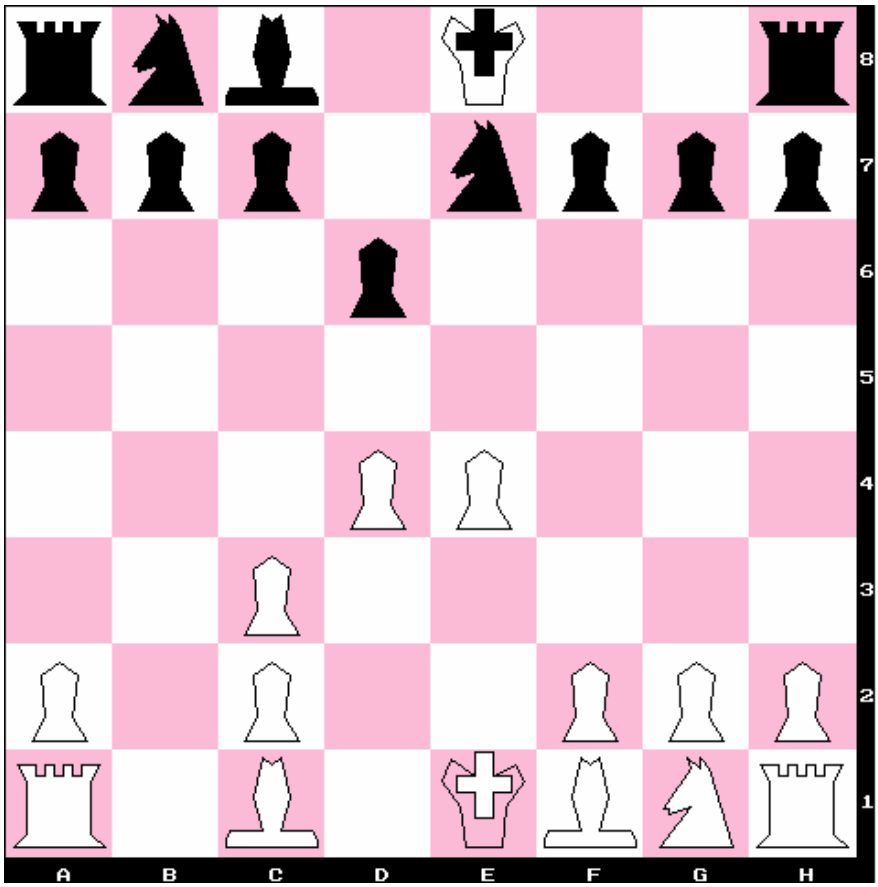
[25] er verwendete dieses Verfahren auch in seinem Schachprogramm SOMA, das aber nur bis zur Tiefe 1 suchte, und es daher mehr als nötig hatte

- Doppelbauern:
Unter Doppelbauern versteht man zwei Bauern, die auf der selben Linie stehen. Dies kann geschehen, wenn man mit einem Bauern schlägt. Solche Konfigurationen (es wären auch drei Bauern hintereinander und mehr denkbar, je mehr desto schlechter) stellen einen Nachteil dar, denn Doppelbauern sind einerseits schwerer gegen Angriffe zu schützen (der gegenseitige Schutz ist ja dahin) und andererseits sind zwei Bauern blockiert, wenn nur der vordere blockiert ist.

- isolierte Bauern:
Hierbei handelt es sich um Bauern, die links und rechts neben sich (bzw. am Rand eben nur links oder nur rechts) keine Kollegen mehr haben. Auch isolierte Bauern sind ein beliebtes Angriffsziel, denn sie sind nicht mehr durch eigene Bauern zu decken.

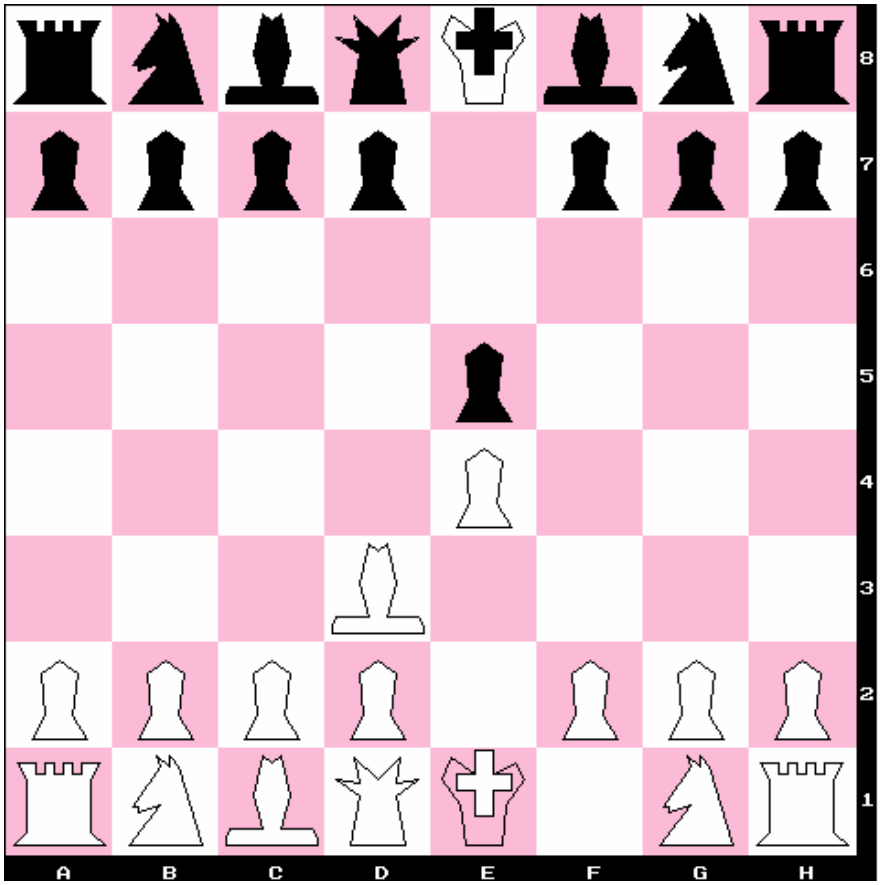
- rückständige Bauern:
Vier Punkte zeichnen einen rückständigen Bauern aus:
1. Er steht auf einer halboffenen Linie
2. Er ist nicht durch eigene Bauern gedeckt und kann auch nicht mehr durch das Vorrücken eigener Bauern gedeckt werden
3. Das vor ihm liegende Feld wird nicht von eigenen Bauern kontrolliert
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.
Shannon schlug in seinen theoretischen Ansätzen vor, jede eigene Bauernschwäche zu erfassen und mit einem Punkteabzug in der Größenordnung eines halben Bauern zu bestrafen, jede Bauernschwäche des Gegners hingegen als Bonus dazu zu addieren. Aufsummiert werden die Bauernschwächen auch in den heutigen Schachprogrammen, doch natürlich wird der Bonus (bzw. Abzug) etwas geringer gehalten, um z.B. Fehleinschätzungen der folgenden Art zu vermeiden:



NachstehendeSchwächen der Bauernstruktur von Weiß werden bei der Analyse entdeckt werden: erstens hat Weiß einen isolierten Bauern auf a2 und zweitens einen Doppelbauern auf der c-Linie. Schwarz hat einen Bauern weniger. Nach Shannons Ansatz wäre die Stellung allerdings ausgeglichen, denn zwei Bauernschwächen aufsummiert ergeben ja den Wert eines ganzen Bauern! Selbstverständlich ist diese Stellung natürlich nicht ausgeglichen. Weiß hat nicht nur einen Mehrbauern, sondern auch eine wesentlich bessere Zentrumskontrolle (nicht zuletzt wegen dem Doppelbauern) die außerdem eine wesentliche freiere Figurenentwicklung fördert.
Shannon wußte, obwohl er nie die Gelegenheit hatte, seine Überlegungen auf einem Computer auszuprobieren, schon damals um die Schwachstellen seiner Bewertung Bescheid. So hat er im Anhang noch weitere Kriterien aufgeführt, Bauern zu bewerten - eine numerische Abschätzung dieser Kriterien gab er allerdings nicht an. Diese beiden Erweiterung betrafen die (eben erwähnte) *Zentrumskontrolle* der Bauern bzw. die Bewertung von *Freibauern*. Die drei wichtigsten Faktoren für eine präzise Bewertung von Freibauern sind Standort (d.h. Entfernung vom Umwandlungsfeld), die Möglichkeit, den Freibauern durch eigene Figuren zu unterstützen bzw. die Möglichkeiten, den Freibauern durch gegnerische Figuren aufzuhalten. Außerdem bietet sich an, die Bedeutung von Freibauern im Endspiel zu erhöhen, denn gerade hier sind die Chancen auf Grund des reduzierten Materials groß, ein Umwandlungsfeld zu erreichen.
Eine wichtige Eigenschaft ließ Shannon allerdings unberücksichtigt: *das Vorrücken der Bauern*. Ungefähr zur gleichen Zeit wie Shannon (Anfang der fünfziger Jahre) hatte Turing bereits ein Schachprogramm geschrieben, das allerdings nie auf einem

Rechner lief. Dennoch ist eine gespielte Partie bekannt, die Turing mühsam (ich berichtete schon in der Einleitung darüber) von Hand durchrechnete. In diesem Programm war ein Vorrücken der Bauern mitberücksichtigt, wenn auch aus heutiger Sicht ein wenig zu primitiv. Alle Linien wurden mit dem selben Bonus versehen, was natürlich nicht optimal war (z.B. riß das Programm bedenkenlos seinen Königsflügel auf und somit wurde die wichtige Schutzfunktion der Bauern zunichte gemacht). Vor allem in der Eröffnung und im Mittelspiel sollte man folgendes tun: ein hoher Bonus für das Vorrücken der Zentrumsbauern auf d2 und e2, ein geringerer Bonus für das Vorrücken der f- bzw. c-Bauern, da sie ja schon praktisch zum Damen bzw. Königsflügel gehören und gar keinen Bonus (bzw. sogar Abzüge) für das Vorrücken der g,h bzw. a,b Bauern (dabei würden ja die Flügel aufgerissen, die dem König Schutz bieten sollen (einer davon)). Im Endspiel hingegen können wir einen konstanten Bonus für das Vorrücken aller Bauern vergeben, hier ist ja das vorrangige Ziel, ein Umwandlungsfeld zu erreichen, alle Bauern sind somit "gleich gut".
Abschließend wollen wir noch ein eröffnungsspezifisches Problem betrachten. Mit dem bisher gelernten sind noch durchaus Stellungen wie die folgende möglich:



Weiß hat sich mit der Blockierung des Zentrumsbauern auf d2 selbst eingeschlossen, denn eine Entwicklung des Läufers c1 ist nun nicht mehr ohne weiteres möglich, von einer möglichen besseren Zentrumskontrolle unter zu Hilfenahme des d Bauern einmal ganz zu schweigen. Auch solche ungünstigen Aufstellungen der Figuren müssen in der Bauenbewertung mitberücksichtigt und dementsprechend

bestraft werden. Wie hoch nun die einzelnen Punktevergaben sind, werden wir im Kapitel 4.3 erfahren. Jetzt wollen wir fortfahren mit der Bewertung der Figuren.

Die Figurenbewertung:

Über die Bewertung von Figuren gibt es keine ausgearbeiteten Theorien (während es über die Kunst der Bauernführung sogar Bücher gibt). Die hier gezeigten Möglichkeiten beschränken sich daher nur auf einige heuristische Grundlagen. Der Grund einer fehlenden Theorie liegt wohl darin begründet, daß die Figuren einem äußerst vielfältigen Zusammenspiel ausgesetzt sind, das auch meist von der Bauernstruktur abhängig ist, also ein starres Bewertungsschema gar nicht zuläßt. So ist auch in den meisten Schachprogrammen die Figurenbewertung äußerst mager ausgefallen im Vergleich zur Bauern- und Königsbewertung. Dafür hat sich allerdings bei der Implementierung der Kriterien ein quasi Standard gebildet, der beinahe von allen Schachprogrammen berücksichtigt wird. Beginnen wir mit der **Springerbewertung**:

Zwei Aspekte erscheinen in Zusammenhang mit einem Springer wesentlich. Zum einen hat ein Springer einen kleinen Wirkungsbereich, es ist daher vorteilhaft, ihn in Zentrumsnähe aufzustellen. Nur von dort kann er seine (ohnehin beschränkte) Bewegungsfreiheit voll ausnutzen. Demnach muß also zuerst einmal sein Zentrumsabstand in die Bewertung einfließen. Zum anderen ist auch die Entwicklung sehr wichtig. Demnach wollen wir ein Zurückbleiben des Springers auf der Grundlinie mit einem konstanten Punkteabzug bestrafen.

Läuferbewertung:

Beim Läufer gilt es vorerst einmal seinen Aktionsradius abzuschätzen, also seine *Mobilität*. Dies können wir dadurch erreichen, indem wir für alle möglichen Felder die er erreichen kann, einen Punktebonus vergeben. Und da wir beim Zügegenerieren ohnehin alle möglichen Züge berechnen, können wir das hier gleich miterledigen. Ebenso wird mit der Mobilität von Dame und Turm verfahren, wobei die Punkte in einem Verhältnis 1:3:4 (für Dame, Turm, Läufer) vergeben werden, dadurch erreichen wir, daß die Mobilität der Dame nicht überbewertet wird. Im Zentrum erhält man übrigens einen größeren Punktezuwachs als am Rand, somit wird erreicht, daß die Figuren danach streben, das Zentrum zu kontrollieren. Den zweiten Aspekt, den wir bei der Läuferbewertung berücksichtigen wollen, ist ein Bonus für ein Läuferpaar. Vor allem in offenen Stellungen kann man damit u.U. starke Angriffe führen. Zu guter Letzt wird noch ein Verweilen auf der Grundlinie (analog zum Springer) mit einem konstanten Punkteabzug bestraft.

Turmbewertung:

Bei diesem Figurentyp wollen wir einige vorteilhafte Aufstellungen mit einem Punktebonus versehen. Der erste Punkt ist ein Turm auf einer offenen oder halboffenen Linie, wobei der erstere Fall natürlich höher bewertet werden soll. Auf offenen Linien kann man die im Mittelspiel eher auf Verteidigungsaufgaben beschränkten Türme hervorragend aktivieren, und daher ist es oft ein vorrangiges Ziel, solche

offenen[26] Linien mit einem Turm "in Besitz" zu nehmen. Denn meist sind damit aussichtsreiche Vorstöße ins gegnerische Lager möglich, vor allem wenn es gelingt die Türme zu verdoppeln. Auch dies soll mit einem Extrabonus belohnt werden. Solch eine Bewertung für offene Linien hat normalerweise jedes ernst zu nehmende Programm implementiert. Ein Programm, daß sich besonders auf freie Linien stürzt ist PSION, eines der stärksten Programme, die ich auf dem PC besitze. Dieses Wissen kann man natürlich dazu nützen, das Programm in Nachteil zu bringen, indem man beispielsweise eine Linie in Besitz nimmt. Dies soll aber nur ein Ablenkungsmanöver sein, inzwischen kann man einen anderen Angriff vorbereiten, denn PSION probiert meist mit allen Mitteln, zuerst einmal die Linie wieder zurückzuerobern.
Was bei einer Turmbewertung noch nicht unwesentlich erscheint, ist das Eindringen eines Turmes auf der siebenten (bzw. zweiten) Linie, vor allem wenn sich der gegnerische König noch auf der Grundlinie befindet, denn dann ist die Mattgefahr relativ groß. Für beide Aspekte wird ein Extrabonus vergeben.
Des weiteren wollen wir eine Beziehung zwischen Türmen und Freibauern herstellen. So sind Türme aufgrund ihrer Gangart bestens geeignet, Freibauern zu unterstützen bzw. gegnerische aufzuhalten. Dieses Kriterium wird in Endspielen mit einem höheren Punktezuwachs belohnt als im Mittelspiel. Einen Punktebonus gibt es genau dann, wenn sich ein Turm auf gleicher Linie mit einem Freibauern befindet.

Damenbewertung:
Die Mobilität der Dame haben wir bereits beim Zügegenerieren mitberücksichtigt. Es gibt aber noch einen Aspekt, den man bei der Damenbewertung unbedingt nicht außer Acht lassen sollte. So weiß doch jeder gute Schachspieler, daß ein zu frühes Aktivieren der Dame äußerst schlecht ist. Da die Dame (mit Ausnahme des Königs) die wertvollste Figur ist, muß sie jedem Angriff weichen. Daher ist es für den Gegner ein leichtes, jede Menge Tempos zu gewinnen, indem die Dame immer wieder vertrieben wird. Nur wenige Programme haben dieses Merkmal berücksichtigt, so fuhr z.B. das sonst so starke SARGON V wie wild mit seiner Dame auf dem Brett umher, nachdem ich die Eröffnungsbibliothek deaktiviert hatte. Im Salomon™ wird eine Dame, die über die zweite Reihe hinauseilt (bzw. für Schwarz über die 7. Reihe) mit einem Punkteabzug bestraft, der proportional der Anzahl der nichtentwickelten Leichtfiguren ist.

Damit haben wir die Figurenbewertung beendet mit Ausnahme der wichtigsten Figur im ganzen Spiel, dem König. Die Bewertung des Königs ist eine der wichtigsten Aufgaben der Bewertungsfunktion.

---

[26] halboffen bedeutet, daß sich zwar kein eigener Bauer mehr auf der Linie befindet, dafür aber ein gegnerischer bei einer offenen Linie ist dementsprechend kein Bauer mehr auf der Linie vorhanden

Königsbewertung:
Die hohe Bedeutung des Königs im Schachspiel ist unbestritten, so ist es doch das Ziel des ganzen Spieles, den gegnerischen Monarchen mattzusetzen. Aber der König muß nicht unbedingt nur ein Angriffsziel sein, ganz im Gegenteil, vor allem im Endspiel kann man auch mit ihm Angriffe führen. Er wird zu einer bedeutsamen Figur, und in Bauernendspielen ist er gar die einzige Figur, die die eigenen Bauern unterstützen bzw. die gegnerischen aufhalten kann. Die Entscheidung, ob man sich im Endspiel befindet nimmt man (wie schon zuvor einmal beschrieben) aufgrund der Materialsumme des Gegners vor. Ist diese nämlich unter 2000 Punkten, dann kann der König aus seinem Versteck hervorkommen (man nennt das **Aktivierung des Königs**), ohne der direkten Gefahr ausgesetzt zu sein ins offene Messer zu laufen. Es ist daher klar, daß wir für Eröffnung bzw. Mittelspiel und Endspiel zwei unterschiedliche Typen von Bewertungen vornehmen müssen, wir wollen sie mit **Königssicherheit** und **Königsaktivität** bezeichnen. Beginnen wollen wir mit der

Königssicherheit:

Ein geeigneter Ansatz stammt (na was glauben sie?) von Shannon. Er schlug vor, der Bewertung der Königssicherheit den Bauernschutz sowie Kontrollen in der Nähe des Königsstandortes zu Grunde zu legen. Und in der Tat: fehlende oder vorgerückte Bauern am Flügel, wohin der König rochiert hat, bzw. Kontrollen der Nachbarfelder des Königs sind häufig Indizien dafür, daß der König unsicher steht. Im Grundsatz verschieden, aber trotzdem zu ähnlichen Ergebnissen führend ist die Methode von Turing: er stellte einfach eine (imaginäre) Dame auf den Königsstandort und berechnete ihre Mobilität, je größer diese war, desto schlechter war natürlich die Sicherheit des Königs. Ein Vorteil dieses Verfahrens ist, daß es auch schon zu Beginn, also noch vor der Rochade angewandt werden kann, hier wird das Programm nämlich danach trachten, möglichst früh zu rochieren, um die durch die Anfangszüge entstandene große Mobilität des Königs wieder zu verringern.
Schauen wir uns also nun nach dieser kleinen Übersicht an, was man zu einer Beurteilung der Königssicherheit alles berücksichtigen muß:

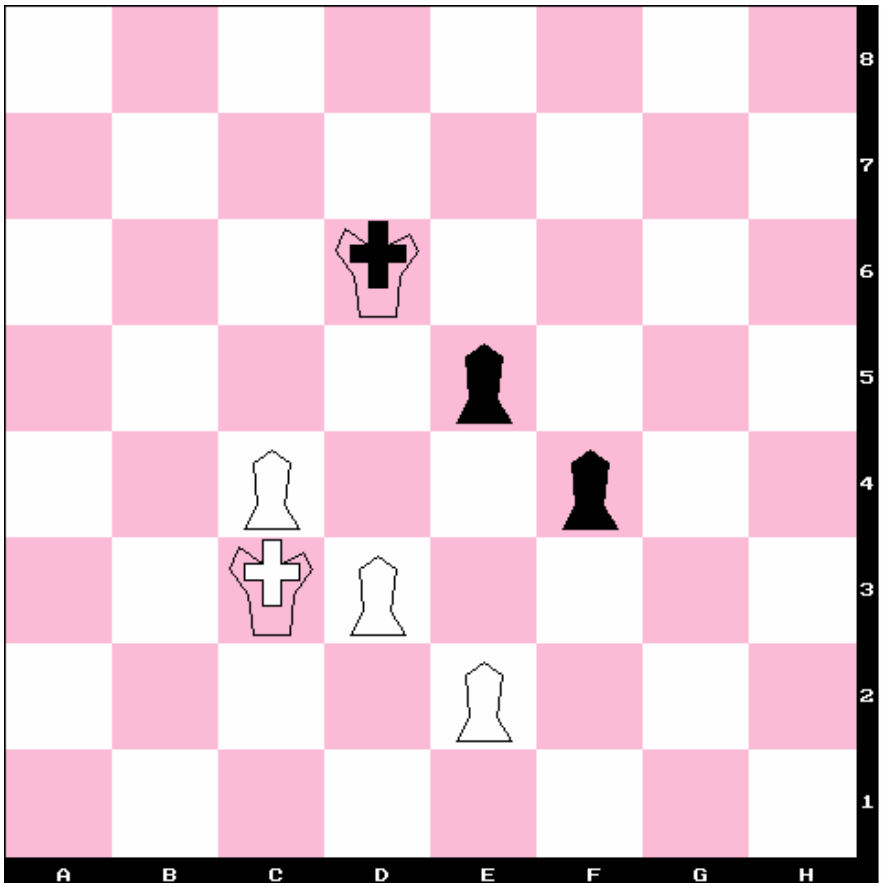
1. Anzahl der noch vorhandenen Figuren mit besonderer Berücksichtigung der Dame. Damit wird die Wichtigkeit einer Betrachtung der Königssicherheit bestimmt (es wird ein Wichtungsfaktor dynamisch errechnet). Außerdem wird ein Spieler, der eine geschwächte Königsstellung besitzt danach trachten, möglichst seine Figuren abzutauschen, denn damit verringern sich die Chancen, einen Königsangriff erfolgreich durchzuführen.
2. Relation der beiden Königsstandorte: es gibt einen Punkteabzug, wenn der gegnerische König sich bereits in Sicherheit gebracht hat, und der eigene König noch immer auf der e-Linie steht. Der Punkteabzug erhöht sich, falls die e-Linie offen ist, da dann die Angriffsgefahr zunimmt.
3. Anzahl der Felder, die der König momentan noch von einem sicheren Platz entfernt ist (für weiß also Entfernung von c1 bzw. g1). Dies ist eigentlich nur dann von Bedeutung, wenn der König seine Rochademöglichkeiten verloren hat. Daher gibt es in solchen Fällen einen Punkteabzug solange, bis sich der König in Sicherheit bringen konnte.

4. Reihe des Königsstandortes: dieser Punkt ist von großem Interesse, wenn der König gedenkt, auf Wanderschaft zu gehen. Dies kann z.B. durch ein Figurenopfer geschehen, wobei der König gezwungen ist, zurückzuschlagen. In der Folge versucht der Gegner den feindlichen Monarchen ins Zentrum zu zwingen (wo er natürlich allen Angriffen aufs Schärfste ausgesetzt ist). Punkteabzüge werden vorgenommen, wenn der König sich über die zweite (bzw. über die siebente) Linie hinaus in Richtung Zentrum bewegt. Wegen der großen Wichtigkeit dieses Aspektes werden hier Punkteabzüge in der Größenordnung eines Bauern und mehr vorgenommen.
5. Anzahl der noch auszuführenden Züge, um die Rochadestellung zu erreichen. Dadurch fließt die Entwicklungsdauer in die Königsbewertung ein, da ein gegnerischer Angriff umso gefährlicher wird, umso mehr Züge benötigt werden, um eine gesicherte Stellung zu erreichen.
6. Gegnerische Figuren, die bis zur Grundlinie vorgedrungen sind. Solche Figuren erschweren normalerweise die Manöver, den König in Sicherheit zu bringen, ungemein.
7. Bauernschutz auf dem Königsflügel: Fehlende Bauern werden mit einem höheren Punkteabzug bestraft als vorgerückte Bauern. Beide Aspekte weisen aber auf eine erhöhte Unsicherheit des Flügels hin.
8. Die lange Rochade wird durch einen geringen Punkteabzug etwas gehandicapt. Denn eine Rochade auf den Damenflügel ist in vielen Fällen etwas unsicherer als eine auf den Königsflügel.

Mitberücksichtigen kann man schließlich noch die Abstände der Türme, Damen und Springer vom König, dies trägt auch zu einer präzisen Königsbewertung bei. Jetzt aber genug zum Thema Sicherheit, wollen wir nun die Königsaktivität betrachten.

Königsaktivität:

Zwei Aspekte scheinen sich hier in den Vordergrund zu drängen. Da wäre zuerst einmal die **Zentralisierung des Königs** zu nennen. Jeder gute Schachspieler wird in Endspielpositionen daran bemüht sein, seinen König in der Nähe des Zentrums zu plazieren, denn nur von hier kann er alle Orte des Spielfeldes besonders schnell erreichen. Das zweite Ziel bezieht sich auf die **Abstände des Königs zu eigenen und gegnerischen Bauern**. Dieses Maß wird bestimmt, indem alle Reihen und Linienabstände zu den Bauernstandorten aufsummiert werden und dieses Ergebnis dann durch die Anzahl der vorhandenen Bauern dividiert wird. Dieser mittlere gewichtete Abstand fließt in die Endspielbewertung mit ein. Wie das nun konkret vor sich geht, wollen wir uns an einem Beispiel erarbeiten. Betrachten sie dazu folgende Stellung:



Bestimmen wir nun die weiße Königsaktivität. Dazu wollen wir zuerst einmal alle Abstände zwischen Bauern und weißem König ausrechnen. Um die Abstände zu berechnen, brauchen wir nur zwei mathematische Operationen: die sogenannte ganzzahlige Division (DIV) und die Modulo Operation (MOD), und schon können wir von der internen, eindimensionalen Darstellung wieder auf die zweidimensionale umrechnen. Das geht so:

Reihe := internes Feld DIV 12 ⇨ wir erhalten Reihenwerte von 2 bis 9
Linie := internes Feld MOD 12 ⇨ wir erhalten Linienwerte von 3 bis 10

Demnach ist also der Brettmittelpunkt bei (5.5/6.5), was bei der Zentralisierung verwendet wird. Für den Abstand berechnen wir nur:

Abstand := ABS (König Reihe - Bauern Reihe) + ABS (König Linie - Bauern Linie)

Die Abstände sind in der folgenden Tabelle zusammengefaßt (wobei die Bedeutung der Bauern auch schon mit den entsprechenden Punktezuwächsen vermerkt ist)

Figur	Standort	Bedeutung	Abst.z.w.König
weißer Bauer	C4	6 (Freibauer)	1
weißer Bauer	D3	2 (ohne Merkmal)	1
weißer Bauer	E2	3 (rückständig)	3
schwarzer Bauer	F4	2 (ohne Merkmal)	4
schwarzer Bauer	E5	3 (rückständig)	4

Der mittlere bezüglich der Bedeutung der Bauern gewichtete Abstand berechnet sich dann so:

(6x1 + 2x1 + 3x3 + 3x4 + 2x4) : (6 + 2 + 3 + 3 + 2) = 37 : 16 = 2.3125

Da der Abstandswert so klein wie möglich gehalten werden sollte, wirkt es sich positiv aus, daß der König so nahe wie möglich bei seinem Freibauern auf c4 verweilt. Die endgültige Bewertung wird dann nach folgender Formel vorgenommen:

Abstandswert := -6 x (mittlerer gewichteter Abstand - 6)

Es ergibt sich also insgesamt eine positive Abschätzung, wenn es dem König gelingt, den mittleren Abstand kleiner als 6 zu halten, ansonsten ergibt sich ein Punkteabzug.
Somit haben wir die letzte wichtige Funktion der Bewertung zu Ende besprochen, und wir kommen nun zum letzten Punkt der Schachprogrammierung, der variablen Suchtiefe.

2.4.5 Variable Suchtiefe bei guten Zügen

Wie ein guter Schachspieler sollte auch ein Schachprogramm, zumindest Ansatzweise die Fähigkeit besitzen, gewisse (interessante) Varianten tiefer zu untersuchen als sonst. Allgemein betrachtet bieten sich dazu zwei Verfahren an:

1. Wir untersuchen weniger interessante Varianten weniger tief als normal
2. Wir untersuchen besonders interessante Varianten tiefer als normal

Der erste Ansatz hat das große Problem, daß das Programm u.U. etwas übersieht. Dieser Ansatz würde dem eines Shannon Typ-B Programms gleichkommen. Und da es bis heute noch keine Vernünftigen Ansätze gibt, welche Züge denn nun interessant sind[27], erweist sich dieser Ansatz als äußerst gefährlich. Eine einfache Bewertungsfunktion kann eben nicht das Gehirn eines Schachmeisters modellieren,

[27] gäbe es einen solchen Ansatz, würde die Spielstärke der Programme wohl sprunghaft ansteigen!

und daher wäre es mit den jetzigen Mitteln beinahe dilettantisch zu behaupten, ein Computer könne wissen, welche Züge gut sind.
Der zweite Ansatz ist bezüglich der obigen Bedenken unproblematisch, es werden wie bisher alle Züge bis zur Suchtiefe untersucht. Der Nachteil ist allerdings: der Suchaufwand steigt, und zwar umso mehr, je mehr Züge das Prädikat "besonders interessant" bekommen. Zwei Versuche sind mir in dieser Richtung bekannt, wovon einer von meiner Wenigkeit stammt.
Der erste Versuch basiert auf einem Vorschlag von Prof. Berliner anläßlich der Konferenz "Advances in Computer Chess IV" im Jahre 1984. Er meinte, daß Schlagzüge, die zurückschlagen, als "besonders interessant" gelten sollten. Diese Züge machen in der Realität etwa einen Anteil von 2.5% aus, der Aufwand scheint verträglich.
Mein Ansatz vertieft eigentlich das Naheliegenste, was man sich nur vorstellen kann, nämlich die Hauptvariante. Gerade diese Züge gelten doch als besonders wichtig, denn der erste Zug der Hauptvariante wird schließlich nach Beendigung der Suche gespielt. Also ist es doch eine zusätzliche Vorsichtsmaßnahme, die Variante noch einmal um einen Halbzug zu vertiefen, vielleicht findet man ja noch eine Entgegnung[28], die die aktuelle Hauptvariante umwirft oder gar unterstreicht. Der Aufwand ist auf jeden Fall denkbar gering, denn nur in der letzten Suchtiefe wird die aktuelle Hauptvariante um einen Halbzug vertieft.

Mit diesem abschließenden Kapitel haben wir unser Wissen über moderne Schachprogramme vervollständigt. Beinahe alle kommerziellen Programme, seien es nun spezielle Schachcomputer oder auch Schachprogramme für den PC, funktionieren nach den eben genannten Prinzipien. Sie haben somit das Rüstzeug erarbeitet, ein eigenes Schachprogramm zu verwirklichen, vielleicht werden sie auch Weltmeister mit ihm, wer weiß...

[28] z.B. eine Gabel, die man bisher aufgrund der geringen Suchtiefe nicht gesehen hatte - so einen Fall hatte ich schon: Salomon spielte gegen SARGON V und verlor aus genau diesem Grund

3. Das Programm Salomon und seine Bedienung

Salomon wurde vollständig in Turbo Pascal 6.0 von Borland geschrieben. Die Entwicklungsdauer belief sich ca. auf ein halbes Jahr, wobei ich immer wieder Unterbrechungen hatte. Eigentlich wollte ich ein solches Programm aus purem Interesse entwickeln. Daß es schließlich eine Diplomarbeit wurde, habe ich Prof. Haase zu verdanken.
Das Programm enthält keine Assembler Routinen, und ist daher etwas langsam im Vergleich zu kommerziellen Programmen. Die Spielstärke ist jedoch beträchtlich (sofern man Salomon nur genügend Bedenkzeit gibt), so erreichte ich gegen den Chessmaster 3000 immerhin ein Remis, obwohl Salomon zu diesem Zeitpunkt noch nicht völlig zufriedenstellend lief. Das Listing ist gut dokumentiert, es sollte also keine Probleme bereiten, das Programm in Zusammenhang mit dieser Arbeit zu verstehen und selbständig zu erweitern.
Als **Hardware Voraussetzungen**, um Salomon laufen lassen zu können, braucht man einen Computer der AT Klasse (80286 oder höher), 640KB Ram und eine VGA Karte (256 KB Screen Memory reicht) oder eine Hercules Karte. Die Erkennung der Graphikkarte läuft vollautomatisch ab, bei einer Hercules Karte kann man jedoch *nicht* mit F1 die Felderkontrollen ansehen, da es ja nur zwei Farben gibt. Wählt man diesen Menüpunkt dennoch an, sieht man nur eine Farbe.

3.1 Das Hauptmenü

3.1.1 Play with Salomon

Dieser Menüpunkt wird wohl der häufigste sein, den man aufruft. Er ermöglicht, wie der Name schon sagt, das Spielen gegen Salomon. Sobald man diesen Menüpunkt anwählt (dies kann, falls vorhanden, auch mit einer Maus geschehen) schaltet Salomon in den Graphikmodus um und stellt als erste Frage (sofern nicht zuvor ein Spielstand geladen wurde - Erklärung erfolgt im nächsten Kapitel), welche Farbe sie spielen wollen (Taste W für White oder Taste B für Black drücken). Dann müssen sie noch mit Hilfe der Tasten "+" und "-" den gewünschten Suchhorizont (look ahead level) einstellen. Dieser wird, falls keine Zeitbegrenzung vorgegeben wurde (wie das funktioniert, erfahren sie im übernächsten Kapitel), in jedem Falle bis zum Ende durchgerechnet. In der Regel spielt Salomon ab Level 3 relativ vernünftig.
Nach dieser Eingabe können sie schon loslegen (bzw. Salomon legt los, wenn sie mit Schwarz spielen). Falls sie beginnen, müssen sie jetzt einen Zug eingeben. Dies geschieht in der üblichen Koordinaten-Notationsform (z.B. e2-e4 oder g1-f3 etc.). Die Eingabe von Rochaden erfolgt hierbei einfach durch die Eingabe des Königszuges (also z.B. um die kleine Rochade von Weiß auszuführen, müßte man e1-g1 eingeben). Den Bindestrich zwischen den Koordinaten fügt Salomon automatisch ein, illegale Eingaben (etwa a9 o.ä.) werden nicht zugelassen, illegale Züge werden mit einer entsprechenden Meldung im Message Fenster zurückgewiesen. Auch illegale

Rochaden werden mit einer entsprechenden Meldung quittiert ("You are not allowed to castle"), das tritt genau dann ein, wenn einer der folgenden Fälle erfüllt ist:

- Sie versuchen zu rochieren und ihr König steht im Schach
- Sie versuchen über ein Feld zu rochieren, das von einem gegnerischen Stein bedroht ist
- Sie haben ihr Rochaderecht entweder durch einen Königszug oder einen Turmzug verspielt

Haben sie ihren Zug ausgefüllt, so erscheint im Message Fenster "I am thinking", Salomon berechnet nun seinen Zug. Jetzt haben sie die Möglichkeit, mit der Space Taste in die Denkvorgänge von Salomon Einsicht zu nehmen, wobei sie folgende Dinge sehen können:

- **Best line** ist die aktuelle Hauptvariante
- eine vereinfachte Brettdarstellung, wo man sieht, was Salomon gerade probiert
- **looking ahead** yy/ xx gibt an, daß gerade yy Halbzüge vorausberechnet wird, xx gibt die aktuelle Tiefe an (kann bei Ruhesuche auch größer als yy sein)
- **Positions seen** ist die aktuelle Anzahl der bereits untersuchten Stellungen
- **Alpha**, **Beta** geben das aktuelle Alpha, Beta Fenster an
- **Best value**, best move geben den besten Zug bzw. den Wert des Zuges an
- **Actual value** gibt den Wert des gerade untersuchten Zuges an
- **Salomons judgement** oft the game gibt Auskunft über den Spielstand
- Im Turniermodus werden die verschiedenen Zeitschranken eingeblendet
- ganz unten sehen sie, welche Zugfolgen Salomon gerade betrachtet

Mit einem weiteren Druck auf Space schalten sie wieder in den Graphikmodus zurück (dies geschieht auch automatisch, sobald Salomon seinen Zug ausführt).
Außer der Zugeingabe haben sie noch die Möglichkeit, die Taste F1 zu drücken, man erhält danach eine farbkodierte Anzeige über die Felderkontrollen (wird im Kapitel 4.3 behandelt). Mit der Taste F2 kann man die Partie aufgeben, dies kann man auch dann tun, wenn man die Partie nur einfach beenden will, und zu einem späteren Zeitpunkt fortsetzen will. Man landet wieder im Hauptmenü, wo man nun mit "Save Game" den aktuelle Spielstand speichern kann.

3.1.2 Load Game, Save Game

Diese beiden Menüpunkte dienen zum Laden bzw. Speichern von Spielständen. Jedesmal, wenn man einen Spielstand ladet, und anschließend mit "Play with Salomon" zu spielen beginnt, wird zu Beginn noch eine zusätzliche Frage gestellt, nämlich welche Farbe am Zug ist. Sie ist ebenfalls mit den Taste W bzw. B zu beantworten. Der Rest verläuft analog zum vorigen Kapitel.
Speichern kann man Partien folgendermaßen:

- Man beendet die laufende Partie mit F2 (funktioniert nur, wenn man selbst am Zug ist) und wählt dann "Save game". Daraufhin gibt man einen Filenamen an - fertig
- Man gibt mit Setup board eine beliebige Stellung ein (wird im Kapitel 3.1.4 erklärt), verläßt anschließend den Menüpunkt und wählt jetzt "Save game", der Rest analog wie oben

Beim Speichern wird zusätzlich geprüft, ob ein File mit gleichem Namen bereits existiert, und wenn ja, eine entsprechende Warnung ausgegeben. Sie können nun wählen, ob sie das File überschreiben wollen oder ob sie einen anderen Filenamen eingeben möchten.

3.1.3 Set response time

Hier können sie einen Zeithorizont setzen. Geben sie eine beliebige Sekundenanzahl ein, nach der Salomon spätestens antworten soll. Im allgemeinen wird Salomon diese Zeit nicht überschreiten, es sei denn er muß die Suche wiederholen[29]. Salomon kann seine Suche auch vor der eingestellten Bedenkzeit beenden, nämlich genau dann, wenn er seinen Suchhorizont bezüglich maximaler Tiefe (look ahead level) erreicht hat. Wollen sie, daß Salomon seine ihm gegebene Zeit in jedem Falle ausnutzt, so brauchen sie lediglich einen dementsprechend hohen look ahead level einzustellen. Geben sie hingegen die Zeit Null Sekunden ein, so sucht Salomon in jedem Fall bis zur maximal vorgegebenen Suchtiefe.

3.1.4 Set up board

Dieser Menüpunkt befähigt sie, beliebige Stellungen einzugeben. Sie brauchen lediglich die Fragen nach den Standorten der Figuren richtig zu beantworten. Dabei können sie mit F1 jederzeit den derzeitigen Stand auf dem Brett begutachten. Salomon macht eine automatische Überprüfung, ob die Eingaben korrekt sind. Überprüft wird:

- sind mehr als 16 Figuren einer Farbe eingegeben worden?
- sind mehr als 8 Bauern einer Farbe eingegeben worden?
- ist die Summe der verwandelten Figuren und der noch vorhandenen Bauern ≤ 8?
- ist das eingegebene Feld schon besetzt?

Ist eine der obigen Bedingungen erfüllt, wird das mit einer entsprechenden Meldung quittiert und je nach Fehler muß die Eingabe u.U. wiederholt werden.

3.1.5 Tournament

[29] Siehe Kapitel 2.4.3

Sobald sie Tournament anwählen, können sie festlegen, wie lange Salomon für eine bestimmte Anzahl von Zügen an Zeit brauchen soll (üblich in Turnieren sind z.B. 40 Züge in zwei Stunden o.ä.). Damit ist Salomon auch in der Lage, an einem öffentlichen Turnier teilzunehmen. Geben sie einfach die Zeit in Stunden, Minuten und Sekunden ein, abschließend die Anzahl der Züge. Fertig! Salomon ist jetzt automatisch im Turniermodus. Wollen sie diesen wieder verlassen, so müssen sie mit F2 (wie üblich möglich, wenn sie am Zuge sind) zurück ins Hauptmenü gehen. Der Turniermodus ist nun abgeschaltet. Wenn sie jetzt auch noch die Partie im normalen Modus fortsetzen möchten, so müssen sie jetzt sofort (!) speichern und wieder laden. Die Partie kann dann in einem beliebigen Modus fortgesetzt werden.
Die Auswahl der Denkzeit erfolgt von Salomon dynamisch. So verfügt er über drei wichtige Zeitschranken: einer minimalen Denkzeit, einer durchschnittlichen Denkzeit und einer maximalen Denkzeit.
Am einfachsten errechnet sich die durchschnittliche Denkzeit: dazu dividiert man einfach die gesamte Zeit durch die Anzahl der Züge, die das Turnier dauern soll. Die minimale Denkzeit beträgt ein Drittel der durchschnittlichen Rechenzeit, die maximale Denkzeit das Doppelte der durchschnittlichen Rechenzeit. Ist die maximale Rechenzeit aber größer als die noch verbliebene Gesamtzeit, so wird die maximale Rechenzeit auf einen Wert gesetzt, der um 1.5 mal verbleibende Zugzahl kleiner ist als die verbliebene Restzeit.
Und wie verwendet Salomon diese Zeiten? Die minimale Denkzeit ist relevant, wenn Salomon bis dahin schon einen akzeptablen Zug gefunden hat (positiver Wert), ansonsten wird weiter bis zur durchschnittlichen Rechenzeit gesucht. Ist bis zu dieser Stelle nur ein Zug gefunden worden, der äußerst schlecht ist, dann wird noch bis zur maximalen Rechenzeit gesucht.
Nach jedem ausgeführten Zug berechnet Salomon die Denkzeiten neu, und je nachdem wie lange er für den letzten Zug gebraucht hat, werden sich die Werte vergrößern bzw. verkleinern. Aus denselben Gründen wie bei set response time kann Salomon bei einer Wiederholung der Suche **nicht** abbrechen. Das kann u.U. bewirken, daß Salomon enorm viel Zeit verliert. Aber anders läßt sich das nicht lösen, falls man es nicht tut, werden u.a. auch Matts nicht richtig erkannt.
Ist man im Tournament Modus, kann man wie üblich mit der Space Taste in den Denkvorgang von Salomon Einblick nehmen. Zusätzlich werden jetzt auch noch die noch verbleibende, die minimale, die maximale und die durchschnittliche Zeit angezeigt (time left to complete tournament, minimal time, maximal time, average time). Außerdem ist auch ein Richtwert zu sehen, wie lange der aktuelle Denkvorgang schon dauert (thinking time).

3.2 Das Untermenü für Advanced Features

Dieser Menüpunkt faßt alle Möglichkeiten zusammen, die viele Schachprogramme nicht anbieten. So ist z.B. kein Schachprogramm auf dem Markt, das man in seiner Spielweise so beeinflussen kann, wie Salomon. Der Chessmaster 3000 hat zwar ansatzweise dieses feature, allerdings beschränkt sich dies nur darauf, wie hoch die einzelnen Figurenwerte sein sollen (ich finde das ziemlich blödsinnig), man hat also

beispielsweise die Möglichkeit, einen Bauern höher zu bewerten als eine Dame (mir ist nach wie vor nicht klar, worin der Clou liegt). Außerdem kann man noch einstellen, wie Remis behandelt werden soll. Es ist durchaus möglich, ein Remis wie einen Sieg (!) sehen zu lassen. Wie dumm das wirklich ist, soll folgendes Beispiel verdeutlichen: so nahm der Chessmaster 3000 in einer Partie gegen mich, in der ich einen Zug vor dem Matt war, ohne mit der Wimper zu zucken ein Remis (!!) an. Was sich die Programmierer dabei wohl gedacht haben - vielleicht, daß ihr Programm so gut ist, daß man so einen Menüpunkt unbedingt braucht, um überhaupt gewinnen zu können. Hier soll sich jeder Leser sein Urteil aber selber bilden.

3.2.1 Field Control Analysing

Durch Anwahl dieses Punktes kann man Field Control Analysing ein- bzw. ausschalten. Was der Begriff genau bedeutet, wird in Kapitel 4.2 näher erläutert.

3.2.2 Control Center Analysing

Die Anwahl bewirkt dasselbe wie oben. Die Bedeutung wird ebenfalls in Kapitel 4.3 erklärt.

3.2.3 Automatic Depth Increase

Schaltet man Automatic Depth Increase ein, so erhöht Salomon automatisch (sofern er noch Zeit hat und eine Zeitschranke vorgegeben wurde) seine Suchtiefe um einen Halbzug, falls bei der aktuellen Suchtiefe noch nicht genug Stellungen untersucht wurden. Dies wirkt sich vor allem in Endspielen aus oder auch in Stellungen, wo es nur wenige Züge gibt (z.B. Schachgebot). In solchen Situationen kann Salomon meist mit der Betrachtung weniger Stellungen den Suchhorizont erreichen, es liegt daher nahe, noch einen Halbzug tiefer zu suchen (analog macht dies übrigens auch der Mensch).

3.2.4 Resign enable

Wie ein guter Schachspieler merkt auch Salomon, wenn er verloren hat. Aktiviert man diesen Menüpunkt, so gibt Salomon vorzeitig auf, sobald er merkt, daß er keine Chance mehr hat. Deaktiviert man diesen Menüpunkt, so spielt Salomon solange, bis er ein Matt sieht, dann gibt er in jedem Fall auf.

3.2.5 Change Play Style

Mit diesem Menüpunkt können sie nun vollständig in die Bewertung des Programms eingreifen und alle wichtigen Parameter (Wichtungskoeffizienten, Bonuspunkte, Abzüge) in gewissen Grenzen frei wählen. Kein Programm am Markt bietet eine solche Möglichkeit - dies ist auch klar, denn damit würde man ja der Konkurrenz verraten, wie das eigene Programm arbeitet, und das ist natürlich unerwünscht.

Zu jedem Parameter, den man ändern kann erscheint ein kurzer Text, der Aufschluß darüber gibt, was der jeweilige Parameter bewirkt. Drückt man nur die Eingabe Taste, bleibt der Standardwert (der auch zu Beginn immer angezeigt wird) bestehen, ansonsten kann man innerhalb der angezeigten Grenzen jeden beliebigen Wert einstellen. Man kann sich auf diese Weise seinen eigenen individuellen Schachgegner zusammenstellen! Mit F1 kann man die Eingabe vorzeitig abbrechen.
Ich möchte an dieser Stelle darauf hinweisen, daß eine Änderung der Standardwerte nur dann sinnvoll ist, wenn man die Kapitel 2 und 4 dieser Arbeit genau durchgearbeitet und verstanden hat, ansonsten dürfte es Schwierigkeiten beim Verständnis um die Bedeutung der einzelnen Werte geben. Nach dem Durcharbeiten der beiden Kapitel sollte es aber keine Probleme mehr geben.

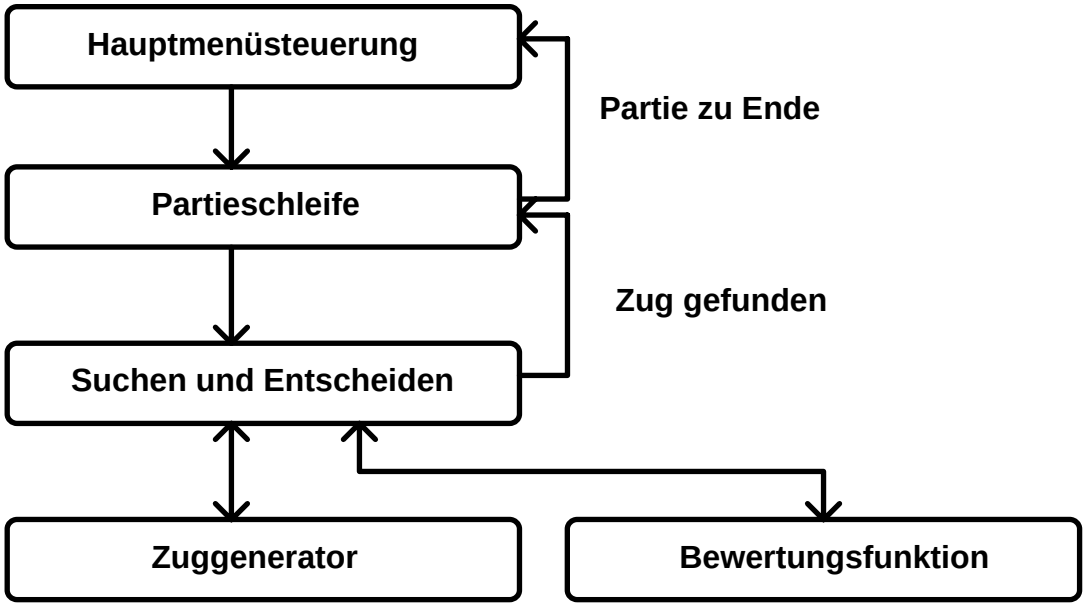
3.2.6 Load, Save Play Style

Mit diesen Punkten können sie einen eben eingegebenen Spielstil speichern, bzw. einen bereits gesicherten Stil laden. Hierzu brauchen sie nur einen geeigneten Filenamen angeben (z.B. "MYSTYLE") und die Eingabetaste drücken. Nach dem Laden bzw. nach Eingabe eines Spielstils ist dieser sofort aktiv und wird von Salomon verwendet. Wollen sie wieder den ursprünglichen Spielstil von Salomon herstellen, so müssen sie "Load Play Style" anwählen und den Filenamen "STANDARD" eintippen. Salomon spielt dann wieder mit den Default Werten. Selbiges erreichen sie übrigens auch, wenn sie Salomon neu starten.

Somit ist die Bedienung von Salomon abgeschlossen, die Bedeutung der einzelnen Menüpunkte sollte jetzt keinerlei Kopfzerbrechen mehr bereiten. Im nächsten Abschnitt werde ich darauf eingehen, welche der im Kapitel 2 kennengelernten Möglichkeiten bei Salomon implementiert wurden. Auch die Defaultwerte für die Wichtungskoeffizienten, Bonuspunkte und Abzüge werden angeführt.

4. Beschreibung der Module von Salomon

Ziel dieses Kapitels ist es, alle in Salomon implementierten Routinen vorzustellen und deren Funktionsweise ein wenig näher zu erläutern. Das soll aber nicht heißen, daß ich jede einzelne Programmzeile nun erklären werde (dies wäre bei über 8000(!) Zeilen auch ziemlich sinnlos), sondern vielmehr möchte ich einen guten Überblick über das Programm und seine Module geben. Es sollte also nach dem Kapitel 2 und dem jetzigen Kapitel keine Schwierigkeiten geben, den ohnehin gut dokumentierten Source Text zu verstehen. Beginnen wir mit einem Diagramm, welches die grundsätzlichen Bausteine und deren Zusammenwirken aufzeigt:



Sobald man Salomon startet, landet man in der Hauptmenüsteuerung, von wo aus alle anderen Module aufgerufen werden. Von hier werden auch die "Advanced features" eingestellt. Ruft man "Play with Salomon" auf, wird die Partieschleife aktiv[30]. Hier wird der gesamte Ablauf einer Partie gesteuert. Folgende Aufgaben gilt es zu bewältigen:

- Graphik initialisieren und das Spielfeld ausgeben
- Spieler nach seiner Wunschfarbe bzw. nach dem look ahead level fragen
- Hauptschleife anspringen, die erst wieder verlassen wird, wenn die Partie in irgendeiner Form beendet wird
- Die Suche aufrufen
- Den von der Suche bekommenen Zug darauf analysieren, ob in irgendeiner Form die Partie als beendet angesehen werden kann. Dies kann sein bei Matt, Remis, Patt, Spieler hat aufgegeben, Turnierzeit wurde überschritten, Salomon gibt auf

[30] Diese Routine findet man im Hauptprogramm (SALOMON.PAS) unter dem Namen "Play_Chess"

- Falls der Turniermodus aktiv ist, werden dynamisch die minimale, maximale und durchschnittliche Denkzeit berechnet
- Die Breitensuche und die damit verbundene Vorbelegung des Alpha-Beta Fensters[31] bzw. eine notwendige Wiederholung der Suche ist ebenfalls Teil der Hauptschleife
- Gilt die Partie als beendet, wird die Kontrolle wieder an die Hauptmenüsteuerung zurückgegeben und der Graphikmodus abgeschaltet.

Direkt von der Partieschleife wird auch das Such- und Entscheidungsmodul aufgerufen, das wir uns einmal näher ansehen wollen.

4.1 Die Suche

Die Suche hat eine Flut von verschiedenen Aufgaben zu erfüllen, die wir schon im Kapitel 2.4.3 kennengelernt haben. Um folgende Punkte kümmert sich die Suche von Salomon:

- Feststellung, ob ein Zeithorizont gesetzt wurde, wenn ja, wird bei der entsprechenden Zeit die Suche abgebrochen, sofern in der Hauptvariante mindestens ein Zug steht.
- Feststellung, ob der Turniermodus aktiv ist. Wenn ja, wird die Denkzeit nach minimaler, durchschnittlicher bzw. maximaler Rechenzeit eingeteilt. Die minimale Denkzeit wird genau dann verwendet, wenn der Vorteil gegenüber der letzten Stellung mehr als 25 Punkte beträgt (Konstante BewSchranke in der Unit CHESS.PAS), die durchschnittliche Zeit greift Fuß, wenn der aktuelle Nachteil nicht mehr als einen Bauern ausmacht (gegenüber der letzten Stellung) - in der Konstante BewSchrankeMittel (Unit CHESS.PAS) definiert. Ist der Nachteil mehr als ein Bauer, dann wird die maximale Denkzeit ausgenutzt.
- Aufruf des Zuggenerators
- Speicherung der aktuellen Mobilität von Dame, Turm und Läufer in einem Array. Die Mobilitäten werden ja, wie wir bereits wissen, vom Zuggenerator mitberechnet und dann von der Bewertung verwendet.
- Vorbewertung der Züge sowie eine Sortierung mit Quicksort. Damit soll ein Maximum an Alpha-Beta Cut Offs erreicht werden. Ist die Ruhesuche im Gange werden selbstverständlich nur mehr die Schlagzüge bewertet bzw. sortiert.
- Wenn der gerade geholte Zug eine Rochade ist, wird geprüft, ob sie möglich ist (dies erledigt eine kleine Hilfsroutine "Rochade_moeglich"). Wenn nicht, wird sofort der nächste Zug geholt.
- Feststellung, ob die gerade übergebene Stellung illegal ist, d.h. ob der gegnerische König geschlagen werden kann[32]. Wenn ja, wird ein Mattwert zurückgegeben, und die aktuelle Rekursionstiefe beendet.

---

[31] Siehe Kapitel 2.4.3

- Schauen, ob der eigene König im Schach steht und falls noch die volle Suche im Gange ist, wird ein Mattwert zurückgegeben (die volle Suche kann ein Matt auf jeden Fall feststellen). Ist die Ruhesuche bereits im Gange, dann muß nach dem Verfahren, das im Kapitel 2.4.3 vorgestellt wurde analysiert werden, ob es ein Matt ist, oder nur ein harmloses Schach.
- Pattfeststellung. Dies macht Salomon so: Bevor die Züge durchprobiert werden, wird noch geschaut, ob ein Schachgebot vorliegt. Dies wird in der Variablen SchachGebot vermerkt. Hier wird nun ausgenutzt, daß Matt und Patt sehr ähnlich sind, denn in beiden Situationen ist kein legaler Zug mehr möglich. Man erhält also als Bewertungswert auf jeden Fall einen Extremwert (z.B., wenn Weiß am Zuge ist, -15998, was einem Matt in zwei Zügen entsprechen würde). Nach dem Ausprobieren der Züge braucht man also lediglich feststellen, ob so ein Extremwert vorliegt (der MMWert wird vor dem Ausprobieren der Züge mit dem denkbar schlechtesten Wert vorbelegt, das wäre für Schwarz etwa 16000-Tiefe) und wenn ja, ob vorher ein Schachgebot vorlag. Wenn ja ⇨ Matt, wenn nein ⇨ Patt!
- Umschalten zwischen voller Suche und Ruhesuche. Dies geschieht genau dann, wenn die Tiefe größer als die Abbruchbedingung wird. Ab dann werden nur mehr die Schlagzüge untersucht. Außerdem wird forward pruning nun angewandt. Selbstverständlich sind beide Arten des forward prunings implementiert.
- Probeweise Ausführung des Zuges, dabei gibt es zwei unterschiedliche Routinen für Schlagzüge und normale Züge (daher kann auch die Ruhesuche einfach durchgeführt werden, man verwendet eben dann nur mehr eine Routine). Man muß sich alle Daten, die man für die Ausführung des Zuges verändert merken, um sie dann bei Rückkehr in die aktuelle Rekursionstiefe wieder rückgängig zu machen. Dies ist gar nicht einmal so einfach, bedenke man doch die vielen Sonderzüge wie Umwandlungen, En-Passant, Rochaden. All diese Züge muß man einer Sonderbehandlung unterziehen, dies schlägt sich auch in einer dementsprechenden Länge des Codes nieder. Welche Datenstrukturen den aktuellen Brettzustand beschreiben, werde ich im nächsten Kapitel vermitteln.
- Aufruf der nächsten Rekursionstiefe
- Schauen, ob der aktuelle Zug den bisher besten Wert (gespeichert in der Variable MMWert - die Variable MiniMaxWert gibt den Wert des eben ausgeführten Zuges an) verbessert hat, wenn ja dann neuen besten Zug vermerken und die Hauptvariante berechnen
- Beachtung, ob der aktuelle Wert zu einem Alpha oder Beta Cut Off führt. Wenn ja, cutten. Wenn nein, schauen, ob Alpha bzw. Beta wenigstens verbessert werden können[33]
- Feststellung, ob der Spieler Einsicht in den Denkvorgang nehmen will. Wenn ja, in den Textmodus umschalten und "Window on the Search anzeigen"[34]

---

[32] Schachgebote und Matts werden wie im Kapitel 2.4.3 schon besprochen in der Suche erkannt und nicht in der Bewertung!

[33] Damit ist eine weitere Verengung des Alpha-Beta Fensters verbunden, also mehr Cut Offs!

[34] Die Ausgabe nimmt dann die Hilfsroutine Ausgabe_Feld vor

- Ist die Suche beendet, steht in der Hauptvariante (in der Suche ist die entsprechende Variable HAUPTVAR) die berechnete Zugfolge von Salomon.

Sie sehen, die Aufgaben sind vielfältig. Das Modul ist auch entsprechend umfangreich und ist in der Unit CHESS.PAS zu finden unter dem Namen "Suche". Innerhalb dieses Moduls befindet sich dann erst die eigentlich rekursive Prozedur (Name: "Suche_Rek"), die äußere Hülle wird für Initialisierungen benötigt.

4.2 Der Zuggenerator

Welche Routinen grundsätzlich in einem Zuggenerator verwirklicht sind, haben wir schon im Kapitel 2.4.2 geklärt. Als besonders aufwendig gilt die **Bauernzuggenerierung**. Hier muß festgestellt werden, ob man ein oder zwei Felder ziehen darf, ob eine Umwandlung vorliegt und wenn ja, alle möglichen Züge in die Zugliste eintragen, ob eine Figur vor ihnen steht, wenn ja dürfen sie nicht vorwärts ziehen, ob eine Figur schräg neben ihnen steht, wenn ja dürfen sie schlagen.
All diese Umstände machen das Programmieren zu einer Strafarbeit!!
Viel einfacher geht es da schon mit den **Springerzügen**. Man trägt sich einfach in ein Array (Name ist SpringerWeiten) alle Springmöglichkeiten ein (gemeint sind die entsprechenden Additions- bzw. Subtraktionsterme zu den Indizes, um alle möglichen Standorte zu erreichen) und durchläuft alles in einer kleinen Schleife.
Ebenso einfach lassen sich **Damen-, Turm- und Läuferzüge** generieren. Auch hier werden alle acht möglichen Richtungen in ein Array geschrieben (Name ist DamenWeiten) und dann in einer Schleife durchlaufen. Da Turm und Läufer in ihren Zugmöglichkeiten quasi "Teilmengen" der Dame sind, braucht man in der Schleife nur den Figurentyp abfragen und die für die entsprechende Figur gültigen Indizes herauszukristallisieren (für den Turm wären das eben alle jenen Indizes, die eine Fortbewegung in einer Geraden ermöglichen). Die Berechnung kann in einer einzigen Schleife erfolgen.
Ähnlich lassen sich auch die **Königszüge** bestimmen. Hier muß lediglich beachtet werden, daß der König immer nur um ein Feld fahren darf, ansonsten ist alles gleich wie oben.
Das **Generieren von Rochadezügen** erfolgt, wie bereits erwähnt, unintelligent. D.h. es wird an dieser Stelle weder geprüft, ob der König gerade im Schach steht, noch ob eines der Felder, über die der König wandert, von einer gegnerischen Figur bedroht ist. Diese Aufgabe wird von der Suche erledigt. Hier werden lediglich zwei Dinge geprüft:

1. Ist eine Rochade überhaupt noch möglich? Dies geschieht mit Hilfe eines boole'schen Arrays (Name: RochadenFeld), dessen Einträge von der Suche laufend aktualisiert werden (z.B. wenn Weiß den Zug Ta1-a2 macht, ist die kleine Rochade nicht mehr möglich)
2. Sind die Felder frei, über die eine Rochade ausgeführt wird?

Kann man beide Fragen mit "Ja" beantworten, so ist eine Rochade möglich und sie wird in die Zugliste eingetragen.
Auch für die **En-Passantzug Generierung** wird die Hilfe der Suche benötigt. Hierzu wird eine globale Variable namens EnPassantFeld verwendet, deren normaler Wert 0 ist. Wird jetzt aber ein Bauernzug (probeweise) ausgeführt, der um zwei Felder vorrückt (genau dann ist ja ein En-Passantzug möglich!), so wird von der Suche die Variable EnPassantFeld auf den Wert des Zielfeldes des Bauern gesetzt. Der Zuggenerator braucht jetzt nur mehr nachzusehen, ob sich ein (gegnerischer) Bauer neben diesem Feld befindet, wenn ja, dann kann man einen En-Passant Zug eintragen, so einfach ist das.

Doch jetzt einmal konkret dazu, in welche Datenstruktur die Züge überhaupt eingetragen werden. Die Züge werden vom Zuggenerator in zwei verschiedene Listen eingetragen, und zwar in eine Schlagzugliste und in eine Ruhezugliste. Dies bedeutet für den Zuggenerator überhaupt keinen Mehraufwand, denn bei der Generierung der Züge muß ohnehin immer das Zielfeld überprüft werden. Helfen tun die beiden Listen aber ungemein, so ist z.B. jetzt die Ruhesuche einfach möglich (auch die Sortierung wird beschleunigt).
Die Einträge der Schlagzugliste bekommen auch noch einen Eintrag mit auf den Weg, der sich "Art" nennt. Hier steht drinnen, um welchen Zug es sich handelt (auch dies ist ohne Mehraufwand möglich, der Zuggenerator weiß ja, was er tut). Zwei Arten werden grundsätzlich unterschieden: einerseits die Umwandlungen, andererseits die En-Passantzüge. Bei den Umwandlungen steht im Art Eintrag der Figurenkode für die eingetauschte Figur (also z.B. 12 für eine schwarze Dame). Bei einem En-Passant Zug hingegen wird der Kode des gegnerischen Bauern eingetragen, also schlägt Weiß En-Passant, hat Art den Wert 6 (Figurenkode eines schwarzen Bauers). Mit diesen Einträgen wird in der Suche der Aufwand vertretbar, einen Zug auszuführen, wobei wie gesagt Ruhe- und Schlagzüge getrennt behandelt werden.

Jetzt bleibt nur noch zu klären, welche Datenstrukturen verwendet werden, um den aktuellen Stand auf dem Brett zu repräsentieren. Hierzu möchte ich gleich bemerken, daß die gewählten Datenstrukturen auf jeden Fall redundant in Bezug auf ihren Inhalt sind, diese Redundanz trägt aber wesentlich zu einer besseren Effizienz der einzelnen Module bei und somit zu einer Geschwindigkeitssteigerung. Ich habe somit die Methode "größere Geschwindigkeit auf Kosten von mehr Speicherplatz" gewählt, was in den jetzigen Zeiten ein durchaus übliches Konzept darstellt. Speicher ist heutzutage kein Problem mehr.
Die erste Datenstruktur (Name im Salomon: "Brett") repräsentiert das **Brett** selbst. Es ist ein eindimensionales Feld, mit 144 Einträgen - völlig analog wie im Kapitel 2.1 geschildert wurde, ich brauche darüber also keine weiteren Worte verlieren.
Weiters gibt es ein zweidimensionales Array, daß sich **Figuren Liste** nennt (Name im Salomon: "FigurenListe"). In der ersten Dimension befinden sich alle Figuren, die Weiß noch hat (also maximal 16 Einträge), in der zweiten Dimension alle von Schwarz. Der erste Eintrag der FigurenListe ist jeweils der König, dieser Eintrag bleibt bis zum Ende der Partie bestehen.

Damit man weiß, bis zu welchem Index das Feld (in beiden Dimensionen extra - schließlich kann ja Schwarz mehr Figuren besitzen als Weiß bzw. umgekehrt) noch gültig ist, gibt es ein weiteres Array, das die jeweilige **Anzahl der Figuren** der beiden Farben enthält (Name im Salomon: FigurenAnzahl). Wird nun z.B. ein Schlagzug ausgeführt, so gibt die Suche (gegebenenfalls) automatisch den letzten Eintrag der Liste in das durch den Schlagzug entstandene Loch. Somit sind stets die Einträge bis hin zum Wert Figuren Anzahl gültig.
Das nächste Feld gibt Informationen bezüglich des **Figurenstandortes** (Name im Salomon: "FigurenStandort"). Hier ist wichtig zu vermerken, daß die Figuren Liste und die Liste der Figurenstandorte stets konsistent sind, d.h. wenn man z.B. an der zweiten Stelle der Figuren Liste eine Dame hat, so findet man ebenfalls an der zweiten Stelle der Standort Liste den jeweiligen Platz der Dame auf dem Brett.
Will man nun von einer beliebigen Figur am Brett wissen, wo sie in der Figuren Liste steht, so hat man noch die **Figuren Adressen zu** Verfügung. Mit diesem Array kann man zu jedem Feld auf dem Brett feststellen, an welcher Stelle in der Figuren Liste die jeweilige Figur vermerkt ist - 0 wenn das Feld leer ist.

Diese Datenstrukturen beschreiben zusammenwirkend den aktuellen Zustand des Brettes. Sie werden an vielen Stellen des Programms verwendet, die Redundanz zahlt sich auf jeden Fall aus. Jetzt wollen wir aber zum interessantesten Teil der Modulbeschreibungen schreiten, zum eigentlichen Gehirn von Salomon, der Stellungsbewertung.

4.3 Die Bewertung

Im Kapitel 2.4.4 haben wir bereits eine Unzahl von Bewertungsmöglichkeiten kennengelernt. In diesem Abschnitt werden wir die Bewertungsfunktion von Salomon näher durchleuchten.
Es ist klar, daß die positionelle Bewertung äußerst viel Zeit in Anspruch nimmt, darum also so wenig oft wie möglich aufgerufen werden sollte. Gleich zu Beginn der Bewertungsroutine wird abgefragt, ob sich die Suche noch in Stadium der vollen Suche oder bereits in der Ruhesuche befindet. Ist letzteres eingetroffen, so spielen positionelle Kriterien nur mehr eine unterordnende Rolle, bei Schlagzügen interessiert i.a. nur der mögliche Materialgewinn bzw. -verlust. Somit erfolgt bei Tiefen, die größer als der aktuelle Suchhorizont sind, nur eine Berechnung der Materialbalance, und dies geht äußerst schnell.
Wie wir bereits wissen, ist von der Bewertungsfunktion meist nur ein Punktezuwachs von maximal etwa 120 Punkten zu erwarten. Wir können sofort zu Beginn der Bewertung abschätzen, ob wir überhaupt ins Alpha-Beta Fenster gelangen können. Wenn nicht, wird die positionelle Bewertung nur abgeschätzt, die eigentliche Routine haben wir uns wieder gespart! Ein Cut Off wird mit hoher Wahrscheinlichkeit ohnehin in der Suche vorgenommen, warum sollten wir sinnlos Rechenzeit verschenken?
Kommt das Programm über die ersten Abfragen hinweg, wird es spannend. Wir haben nämlich eine Stellung gefunden, die es tatsächlich positionell zu bewerten gilt.

Zuerst gilt es einmal zu klären, ob sich die Spieler schon im Endspiel befinden. Dies geschieht, wie bereits im Kapitel 2.4.4 gesagt, für beide Spieler getrennt. Das Ergebnis steht dann in einem Array, das sich "StAn"[35] nennt. Der erste Index ist - wie üblich - für Weiß, der zweite für Schwarz. Ist der Inhalt des Feldes eine 1, dann ist der jeweilige Spieler noch nicht im Endspiel[36], wenn zwei drinnen steht, dann schon.
Jetzt gilt es festzustellen, ob Salomon Mattsetzen soll. Die drei Voraussetzungen haben wir bereits im Kapitel 2.4.4 kennengelernt, ebenso wie dieser Vorgang vor sich geht. Völlig analog (auch die Wichtungskoeffizienten sind dieselben) wird auch im Salomon verfahren. Je nachdem mit welcher Farbe Salomon spielt, wählt er eine der beiden Mattroutinen aus und beginnt den Matt-Prozeß. Eine weitere Bewertung ist nicht mehr nötig, der Durchlauf dieser Sparversion genügt bereits.
Der nächste Schritt, falls kein Matt-Prozeß läuft, ist dann eine Analyse des (aktuellen) Brettes. Folgende Dinge gilt es zu erfassen:

- Anzahl der Bauern pro Linie und deren genaue Standorte
- Anzahl und genaue Standorte von Springer, Läufer, Dame(n), Türmen
- Standorte der Könige

Diese Analyse könnte im ersten Gedanken auf diese Art erfolgen: wir durchlaufen in einer Schleife einfach alle 64 möglichen Felder des Feldes. Damit erhalten wir obige Informationen sicher (um die Anzahl der Bauern pro Linie einfacher zu erhalten, durchlaufen wir das Feld vorzugsweise einfach Linienweise). Diese Brute Force Methode habe ich auch anfangs verwendet - doch wie man sich leicht überzeugen kann, sie ist mindestens zu 100% redundant! Schließlich verfügen beide Spieler zusammen maximal über 32 Figuren, wir bräuchten also höchstens 32 Felder[37] abscannen und nicht 64! Das Problem ist nur, wie bekommen wir die richtigen Felder?
Jetzt macht sich eben unsere Redundanz bezüglich der Repräsentation des aktuellen Brettzustandes bezahlt (wie im letzten Kapitel geschildert)! Mit Hilfe der Figurenliste, der Figuren Standorte und der Figuren Anzahl läßt sich tatsächlich alles rekonstruieren, was wir benötigen. Und es sind nur zwei Schleifen[38] mit insgesamt maximal 32 Durchgängen notwendig (eben dann, wenn beide Seiten noch alle Figuren haben). Wir haben also einen optimalen Algorithmus gefunden, besser geht es nicht.
Zur sogenannten Analyse des aktuellen Brettzustandes gehört auch noch die Anzahl der nichtentwickelten Leichtfiguren zu zählen, dies wird bei der Damenbewertung benötigt.

---

[35] für Stellungsanalyse
[36] zur Erinnerung: dann ist die Materialsumme **des Gegners** größer als 2000 Punkte
[37] meist sogar weniger, das Material wird i.a. ja recht schnell abnehmen
[38] jeweils eine über die Figuren Liste von Weiß bzw. Schwarz

Nach der Analyse folgt ein großer Block, der nur der **Bauernbewertung** gewidmet ist. Dieser Block verrichtet die folgenden Aufgaben:

- Finden von blockierten Zentrumsbauern und Abzug von 10 Punkten, falls einer gefunden wird

- Finden von isolierten Bauern mit einem konstanten Abzug von 15 Punkten bei Entdeckung solcher
- Aufspürung von Mehrfachbauern. Abzüge: 4 Punkte für einen Doppelbauern, 40 Punkte für drei oder noch mehr Bauern in einer Linie (bei solchen Konstellationen geht meist mindestens ein Bauer verloren)

- Suche nach Freibauern. Für jeden gefundenen Freibauern werden Bonuspunkte nach der folgenden Formel vergeben:

  BonusWert := Endspielfaktor x Umwandlungsfaktor x Standort x Standort

  Wobei der Endspielfaktor (von der Stellungs Analyse im Array StAn) im Mittelspiel auf 1, im Endspiel auf 2 gesetzt wird (wie oben bereits erklärt).
  Im Umwandlungsfaktor sind der Grundwert zur Wichtung (1.15), die Unterstützung von eigenen Bauern auf den Nachbarlinien (0.35 je Bauer) und die Blockade des Bauern durch eine fremde Figur (-0.35) enthalten.
  Bezüglich des Standortes interessiert natürlich nur die Reihe. Wegen der ständig zunehmenden Wahrscheinlichkeit einer Umwandlung bei Vorrücken, geht dieser Faktor auch quadratisch ein.

- Bewertung für das Vorrücken der Bauern, getrennt nach Eröffnung / Mittelspiel und Endspiel. Für die Eröffnung bzw. Mittelspiel werden pro Reihe folgende Pluspunkte vergeben[39]:

  (0.0 , 0.0 , 3.9 , 5.4 , 7.0 , 2.3 , 0.0 , 0.0)

  und im Endspiel:

  (2.0 , 2.0 , 2.0 , 2.0 , 2.0 , 2.0 , 2.0 , 2.0)

- Auffindung von rückständigen Bauer. Abzüge von 2 Punkten, wenn das vor ihm liegende Feld von einem gegnerischen Bauern kontrolliert wird und ein Abzug von 4 Punkten, wenn zwei Kontrollen bestehen.

Die nächste Aufgabe der Bewertung ist die **Figurenbewertung**. Sie ist in die folgenden Bereiche unterteilt:

---

[39] der erste Wert entspricht jeweils der a-Linie, der zweite der b-Linie usw.

- Bewertung von Springern: zwei Dinge galt es hier zu berücksichtigen, einerseits den Zentrumsabstand und andererseits die Entwicklung. Gehen wir davon aus, daß sich das Zentrum bei (5.5 / 6.5) befindet[40], kann man die Zentrumsbewertung nach folgender Formel durchführen:

  Abstandswert := 0.6 x (6 - 2 x Zentrumsabstand)
  Wobei sich der Zentrumsabstand wie folgt berechnet:

  Zentrumsabstand := ABS (Springer Reihe - 5.5) + ABS (Springer Spalte - 6.5)

  Die Ergebnisse der Zentrumsformel sind Null, falls einer der Standorte c3, f3, c6, f6 gewählt wurde, positiv, falls man sich dem Zentrum nähert und negativ, falls man irgendwo am Rande steht (probieren sie es doch einmal aus!). Entwicklungsrückstände werden mit einem Punkteabzug von 4.7 bestraft.

- Läuferbewertung: ein Läuferpaar wird mit 4 Punkten belohnt, ein Nichtentwickeln mit 5.5 Punkten bestraft

- Turmbewertung:
  Für die Besetzung von offenen Linien wird ein Bonus von 4 Punkten vergeben, für halboffene Linien werden 1.5 Punkte gutgeschrieben.
  Einer Turmverdopplung wird durch 8 Punkte Rechnung getragen, und hat sich schließlich ein Turm auf der 7. (bzw. 2.) Reihe eingenistet, so ist dies 5 Punkte wert - befindet sich außerdem der gegnerische König noch auf der Grundlinie, werden noch einmal 6 Punkte auf das Bonuskonto verbucht.
  Um die Beziehung zwischen Turm und Freibauern herzustellen, vergeben wir außerdem noch in der Eröffnung / Mittelspiel 10 und im Endspiel 15 Punkte, wenn sich ein Turm auf der selben Linie wie ein Freibauer befindet. Einerseits trägt dies nämlich ausgezeichnet zu einer Unterstützung bei, andererseits kann man dadurch auch die gegnerischen Freibauern u.U. aufhalten bzw. behindern.

- Bei der Bewertung der Dame wird nur berücksichtigt, daß ein zu frühes Aktivieren sehr schädlich für die Gesundheit ist. Dem wird durch folgenden Abzug Rechnung getragen:

  Abzug := 2.5 * Anzahl der nicht entwickelten Leichtfiguren

- Der letzte große Abschnitt ist selbstverständlich der **Königssicherheit** bzw. der **Königsaktivität** gewidmet. Beginnen wir mit der Königssicherheit:

  Ziel der Bewertungsfunktion soll es sein, einen Wert zu errechnen, der Auskunft über die Unsicherheit gibt. Ist also alles Ok, dann wird der Wert Null sein, sonst einen Wert proportional der Unsicherheit aufweisen.

---

[40] wie man auf diesen Wert kommt, habe ich im Kapitel 2.4.4 (Königsaktivität) erklärt

Am Anfang der Routine werden zwei Wichtungsfaktoren berechnet, die dann in der Folge Verwendung finden. Dies sind:

1. Figurensumme:
Dies ist ein Kriterium, das in jede Königsbewertung einfließen muß, völlig egal, wo sich der König gerade befindet. Dabei werden Bauern überhaupt nicht, Damen jedoch mit dem Wert 3 berücksichtigt. Insgesamt wird der Wert darauf genormt, daß zu Beginn 8 herauskommt.

2. Königsrelation:
Falls Salomon schon rochiert hat, oder beide Könige noch nicht, wird die Königsrelation auf Null gesetzt. Falls Salomon noch nicht rochiert hat, der Spieler aber schon, wird die Königsrelation genau dann auf 1 gesetzt, falls die e-Linie geschlossen oder halboffen ist, sonst auf zwei. Dieser Wert spiegelt auch die Wichtigkeit einer Rochade wieder.

Als nächstes wird eine Fallunterscheidung zwischen den folgenden vier Fällen vorgenommen, die dann auch extra (nur jeweils einer) bewertet werden. Dies sind:

1. Exponierte Königsstellung:
Steht der König weder auf der 1. noch auf der zweiten Reihe, und der Gegner befindet sich noch nicht im Endspiel, dann wird eine Sonderbewertung vorgenommen, die zu einer drastischen Bestrafung führt. Der König ist in solchen Situationen nämlich einer sehr großen Gefahr ausgesetzt, vor allem, wenn er ins Zentrum gedrängt wird. Die Verteidigung ist dann kaum mehr möglich, die Mattgefahr äußerst hoch. Die Bewertung erfolgt nach der Formel:

Abzug := Figuren Summe x (Königsrelation + 20)

Damit können Abzüge in der Größenordnung von 1.5 bis 2 Bauern zustande kommen.

2. Königsstellung vor Ausführung der Rochade:
Hat der König noch keine Rochade gemacht, müssen wird die Unsicherheit beider Flügel errechnen, um uns dann später (bei der Ausführung der Rochade) für den sichereren Flügel entscheiden zu können. Salomon berechnet hier zwei Werte (im Programm WertLinks, WertRechts), die umso höher sind, je unsicherer der jeweilige Flügel ist. Folgende Kriterien fließen in die Abschätzung ein:

- Bauernschutz:
Erhöhung des Wertes um 0.5 Punkte, falls kein Bauer mehr auf der 2. (bzw. 7.) Linie und Erhöhung um 1 Punkt, falls der Bauer überhaupt fehlt (am Königsflügel werden g und h Bauer am Damenflügel a,b und c Bauer untersucht).

-Entwicklungsdauer
Erhöhung des Wertes um jeweils 0.5 Punkte, falls sich auf den Feldern, über die die Rochade ausgeführt wird, noch Figuren befinden.

Außerdem wird, falls das Rochaderecht verspielt wurde, der Abstand zu den jeweiligen Rochadefeldern berechnet und mit einem Faktor 1.5 gewichtet. Zusätzlich bekommt man noch 0.5 Punkte verpaßt, falls eine Rochade möglich ist, man sie aber noch nicht ausgeführt hat bzw. 2 Punkte, wenn man sein Rochaderecht verspielt hat.

Die Berechnung der Werte erfolgt für Damen- und Königsflügel getrennt, die Höhe des Abzuges wird dann nach der folgenden Formel berechnet:

Abzug := FigurenSumme x (Königsrelation + Min(Damenflügel,Königsflügel))

Wobei "Min" Minimumsbildung bedeutet - wir haben uns ja noch für keinen Flügel entschieden.

3. Königsstellung nach Ausführung der langen Rochade:
Hier brauchen wir lediglich den Unsicherheitsfaktor des Damenflügels zu berechnen, außerdem gilt nach Ausführung der Rochade generell, daß der Wert der Königsrelation gleich Null ist. Die Bewertung kann also nach folgender Formel vorgenommen werden:

Abzug := Figuren Summe x Damenflügel Wert

4. Königsstellung nach Ausführung der kurzen Rochade:
Analog zum obigen Fall erfolgt die Bewertung einfach nach:

Abzug := Figuren Summe x Königsflügel Wert

Mit dieser Fallunterscheidung ist es uns nun sehr gut möglich, eine den Stellungen angepaßte Bewertung vorzunehmen. Wie gesagt, aber nur in der Eröffnung und im Mittelspiel, im Endspiel wollen wir die sogenannte **Königsaktivität** bewerten.

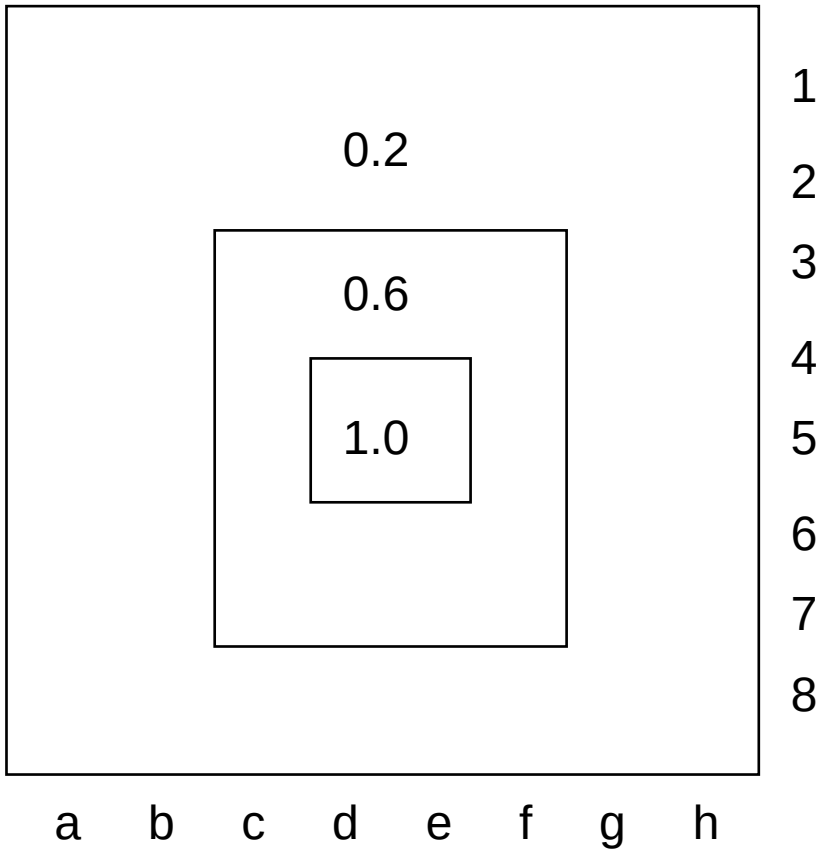
Zur Königsaktivität brauche ich eigentlich nicht mehr allzuviele Worte zu verlieren, eine genaue Beschreibung gab es bereits in Kapitel 2.4.4. Nur soviel zur Auffrischung: Königsaktivität bedeutet einerseits ein Streben hin zum Zentrum andererseits gelten die Bauern als Attraktoren, und zwar gewichtet nach ihrer Bedeutung[41].

Zu guter Letzt wird in der Bewertung auch noch ein Teil vom **Control Center Analysing** durchgeführt. Dieses steht eng in Zusammenhang mit dem bereits oft zitierten **Field Control Analysing**, wobei ich mit dessen Erklärung beginnen möchte, die andere Methode ergibt sich nämlich quasi daraus.
Rufen wir uns dazu noch einmal den Ansatz von Shannon ins Gedächtnis, eine Stellung dynamisch zu bewerten. Er untersuchte alle 64 Felder des Brettes auf Figurenkontrollen, wobei gefesselte Figuren nichts kontrollieren konnten. Wer im Endeffekt mehr Kontrollen aufwies, der stand letztenendlichs auch elastischer. Die Kontrollen sind also in den meisten Fällen ein sehr gutes Maß für die Dynamik einer Stellung (je dynamischer man steht, desto eher hat man auch die Chance, eine gute Kombination anzubringen).

---

[41] Die Hierachie war: 1. Freibauer, 2. rückständige Bauern 3. normale Bauern

Im wesentlichen leistete mein Field Control Analysing dasselbe wie Shannons Ansatz, gefesselte Figuren berücksichtige ich allerdings nicht, der Rechenaufwand würde sich nicht lohnen, es kostet schon die normale Berechnung der Kontrollen eine immense Rechenzeit. Dafür habe ich aber Shannons Ansatz, der bei genauerer Betrachtung doch ein wenig grob erscheint[42], ziemlich verfeinert, ohne allerdings den Rechenaufwand explodieren zu lassen (er bleibt in etwa gleich).
Um die Zentrumsnähe der einzelnen Figuren zu berücksichtigen, wird folgende Idee verwendet, die auch zur Berechnung der Figurenmobilität dient. Man teilt hierzu das Brett in drei Abschnitte, wie sie im Diagramm unten dargestellt sind. Jeder Abschnitt stellt einen Wichtungskoeffizienten dar.



Wie man unschwer erkennen kann gilt: je mehr Felder die Figur in der Nähe des Zentrums kontrolliert, desto mehr Punkte erhält sie. Um die Bedeutung der Figuren auch noch mitzuberücksichtigen, wurden die Figuren 4:3:2:1 für Bauer : Springer, Läufer, Turm : Dame : König gewichtet. Damit haben wir auf einfache Weise die Figurenart **und** die Brettregion berücksichtigt, also den Ansatz von Shannon um ein Vielfaches verfeinert. Wie die einzelnen Kontrollen verteilt sind, zeigt Salomon™ auch graphisch an, wenn man während der Zugeingabe die F1 Taste drückt. Dabei werden am Rande die Farbcodes ausgegeben, wie "stark" die Kontrollen tatsächlich sind. Dies ist immer von der Seite des Spielers betrachtet - d.h. grüne Farbtöne deuten auf eigene Kontrollen hin, rote Farbtöne auf Kontrollen von Salomon™.

---

[42] So wird doch z.B. die Figurenart nicht berücksichtigt und auch nicht die Zentrumsnähe

Der Grundwert für die Gewichtung des Field Control Analysing ist übrigens 1.0, wählt man den Faktor zu hoch, dann versucht das Programm verzweifelt irgendwelche Kontrollen zu erringen, vernachlässigt u.U. auch dabei die Königssicherheit. Deswegen warne ich an dieser Stelle vor einer Überbewertung diese Kriteriums, es handelt sich schließlich nur um eine Heuristik und um nichts Handfestes. Gerade aus diesem Grunde wird das Field Control Analysing nur genau dann aufgerufen, wenn eine neue Hauptvariante aufgestellt werden soll. Somit ist es in der Lage, die neue Hauptvariante doch noch umzuwerfen, oder sie eben sogar noch zu unterstreichen.
Das Control Center Analysing wird im Prinzip von der Routine des Field Control Analysing gleich mitberechnet. Hier geht es darum, ein von der gegnerischen Seite besonders stark kontrolliertes Feld[43] **in der eigenen Hälfte** zu entdecken und etwas dagegen zu unternehmen. Solche stark kontrollierten Felder deuten in der Praxis oft auf die Vorbereitung von starken Angriffen hin, so nehmen z.B. bei einem Königsangriff die Kontrollen in der Nähe des Königs meist rapide zu. Es wird also dringendst angeraten, etwas zu unternehmen.
Unternommen wird dann erst etwas in der Bewertung, nachdem nach Ausführung des Spielerzuges ein mögliches Zentrum berechnet wurde. In der Bewertung wird mit Ausnahme der Bauern der Abstand aller Figuren zu diesem Zentrum bestimmt und entsprechend der Figurenart wird ein Abzug vorgenommen, bei Damen 3 x, bei Türmen 2x und bei Läufern, Springern 1x. Der Grundwert der Wichtung ist 0.8, die Stärke der Kontrollen muß mindestens den Wert 3 aufweisen. Auch hier sei aus den gleichen Gründen wie oben vor einer Überbewertung gewarnt!

4.4 Graphik und Hilfsroutinen

Als Graphik findet bei Salomon™ eine einfache Rastergraphik Verwendung, die auch relativ leicht zu skalieren ist. Die Rastergröße der einzelnen Figuren beträgt 60x60 Pixel, die einzelnen Raster sind als konstante Arrays am Beginn der Unit GRAPHIK.PAS gespeichert und können natürlich nach Belieben geändert werden. Hier findet auch die automatische Erkennung der installierten Graphikkarte statt. Falls weder eine VGA noch eine Hercules Karte erkannt wird, gibt es einen Programmabbruch mit einer entsprechenden Meldung. Damit die Graphikkarte angesprochen werden kann, müssen sich im aktuellen Directory der entsprechende Graphiktreiber[44] und die Schriftart TRIP.CHR befinden. Wollen sie diesen aus irgendeinem Grund nicht im aktuellen Directory haben (z.B. weil sie schon Turbo Pascal auf ihrer Festplatte haben, und gleich diese Treiber benutzen wollen), dann müssen sie ein ASCII File namens SALOMON.BGI erzeugen, in das sie die vollständige Pfadangabe schreiben. Salomon™ sucht beim Start automatisch nach diesem File und verwendet dann auch die dortigen Angaben. Findet Salomon dieses File nicht,

[43] eben ein Zentrum
[44] Entweder EGAVGA.BGI oder HERC.BGI

erwartet er im *aktuellen Directory* den entsprechenden Treiber. Die einzelnen Routinen des Moduls erfüllen im wesentlichen die folgenden Aufgaben:

- Ausgabe des Brettes samt Figuren und Message Fenster
- Erkennen des Graphikmodus und Setzen der Variablen Monochrome, damit in entsprechenden Programmteilen auf eine monochrome Graphikkarte reagiert werden kann (andere Bildschirmmaße, Farben etc.)
- Ziehen einer Figur (Art, Start, Ziel)
- Ausgabe einer Meldung im Message Fenster
- Graphik einschalten, wobei eine Datei namens SALOMON.BGI eingelesen wird, wo der gültige Pfad zu den Dateien EGAVGA.BGI und TRIP.CHR drinnenstehen muß. Existiert diese Datei nicht, so sucht Salomon™ noch im aktuellen Directory nach den obigen Files, wird er auch dort nicht findig, gibt es einen Fatal Error Abbruch.
- Graphik ausschalten
- Eingabe des Zuges im Graphikmodus
- Ausgabe der Kontrollen samt Farbkode

Hilfsroutinen befinden sich in der Unit CHESS.PAS. An Hilfsroutinen existieren:

- Koenig_im_Schach: Diese Prozedur stellt fest, ob der als Parameter übergebene König gerade angegriffen wird. Die Feststellung erfolgt, indem man einfach vom Königsstandort ausgehend alle Richtungen (inklusive Springerweiten) nach gegnerischen Figuren abscannt.
- Illegale_Stellung: Diese Routine überprüft, ob der König des nicht am Zuge befindlichen Spielers geschlagen werden kann. Dazu braucht lediglich die aktuelle Schlagzugliste abgeklappert zu werden.
- Remis: Stellt fest, ob ein klassisches Remis (nur mehr zwei Könige) oder auch ob ein technisches Remis (z.B. nur mehr ein Springer) aufgetreten ist.
- Rochade_moeglich: Überprüft, ob vom gerade am Zuge befindlichen Spieler eine Rochade ausgeführt werden darf. Hierzu wird zuerst einmal geprüft, ob der Spieler im Schach steht, dann ob die Durchzugsfelder bedroht sind.

Somit haben wir das Ende des 4. Abschnittes erreicht. Sie haben jetzt das nötige Rüstzeug erlernt, ein eigenes Programm zu entwickeln, selbstverständlich auch eigene Ideen einfließen zu lassen, vielleicht entdecken ja gerade sie eine Methode, um z.B. den Stellungscharakter mit in die Bewertung aufzunehmen o.ä.
Das letzte Kapitel ist vor allem für jene interessant, die ihr eigenes Schachprogramm testen und es auch besiegen wollen. Speziell im letzten Abschnitt des 5. Kapitels möchte ich noch ein paar Tips geben, wie man die meisten am Markt befindlichen Programme gehörig ins Schwitzen bringen kann.

5. Teststellungen für die Bewertung der Spielstärke von Programmen

Mittlerweile ist der Markt überschwemmt mit Schachprogrammen, viele Benutzer werden oft vor ein großes Problem gestellt, sollen sie sich für ein Programm entscheiden. Das teuerste muß nicht immer das beste sein - teuer ist in den heutigen Zeiten meist mit einer guten Benutzeroberfläche gleichzusetzen, in vielen Fällen ist dann die Spielstärke aber gering. Jeder, der ein Schachprogramm kaufen will, muß sich also zuerst einmal überlegen, ob er ein besonders spielstarkes oder ein Programm mit einer tollen Bedienung haben will. Beides vereint habe ich noch nicht entdeckt, obwohl SARGON V ein heißer Anwärter darauf ist. Leider hat das Programm einige schwere Programmierfehler (so hat SARGON während des Spiels plötzlich angefangen, seine eigenen Figuren zu schlagen und gleich darauf verschwanden meine Figuren spurlos vom Brett ...). In der nachfolgenden Tabelle habe ich versucht, alle gängigen Programme für den PC zusammenzufassen und zu bewerten, und zwar sowohl nach Spielstärke als auch nach Benutzeroberfläche

Name	Firma	Oberfläche	Spielstärke	Features
Chessmaster 2100	Sw. Toolworks	sehr gut	mittel	Lernprogramm
Chessmaster 3000	Sw. Toolworks	sehr gut	mittel	Lernprogramm
Sargon III	Activision	befriedigend	gut	
Sargon V	Activision	sehr gut	gut	Lernprogramm
Cyrus	Intell.Chess Sw.	gut	schwach	gute Menüs
Chess 88	Don Berg	mies	sehr schwach	
Psion	Psion Ltd.	ausreichend	hervorragend	
Battle Chess	Interplay	gut	sehr schwach	Unterhaltung
Turbo Chess	Borland	schlecht	schwach	

Die beiden Chessmasterprogramme haben in beiden (!) Versionen einen schweren Fehler in der Eröffnungsbibliothek, man gewinnt praktisch jedesmal, wenn man folgendes spielt:

Weiß: Chessmaster 2100/3000
Schwarz: Spieler
Sizilianische Verteidigung

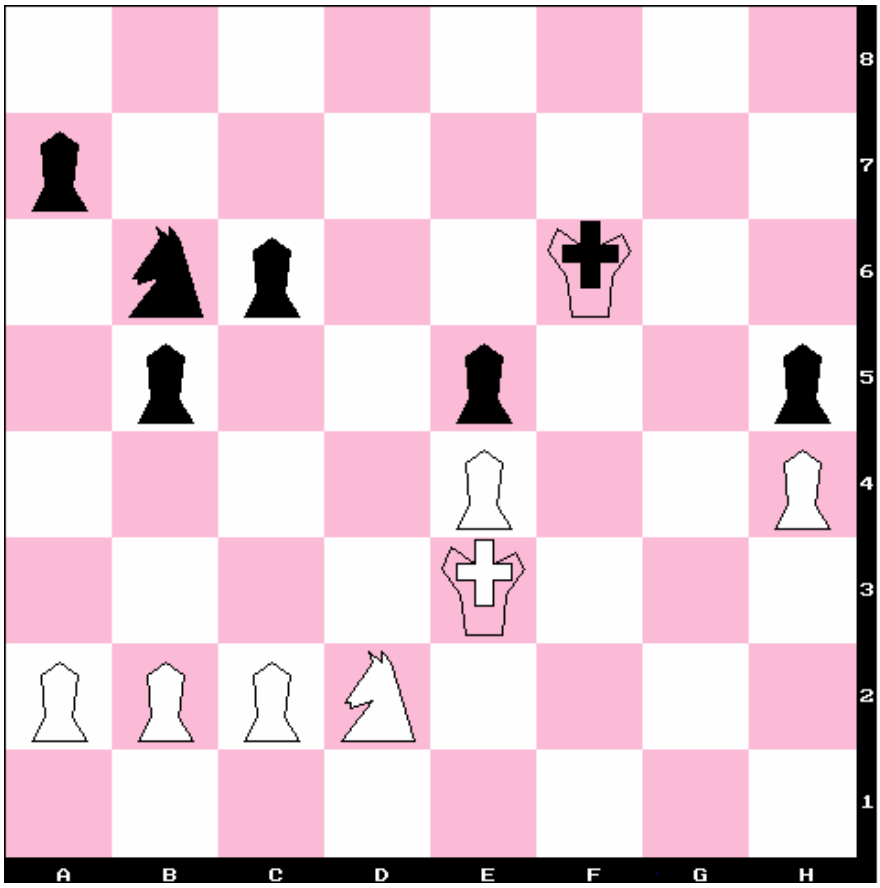
1. e2-e4	c7-c5	6. e4-e5	f6-e4 gilt als nicht besonders für Schwarz
2. g1-f3	e7-e6	7.d1-g4!	e4xc3
3. d2-d4	c5xd4	8. g4-f4??	c3-d5+!! ein schwerer Fehler!!
4. f3xd4	g8-f6	9. c2-c3	d5xf4 Damenverlust und Ende...
5. b1-c3	f8-b4		

Diesen Partieverlauf kann man jederzeit nachvollziehen, außerdem wählt der Chessmaster bei Play-Style "Best" immer diese Eröffnung. Man gewinnt immer!

Battle Chess ist eigentlich eher ein Unterhaltungsprogramm (man kann zusehen, wie sich die Figuren bekämpfen), dementsprechend schlecht spielt es auch Schach.
In den folgenden Kapiteln möchte ich nun ein paar Bewertungsmethoden aufführen, mit denen man eigentlich jedes Schachprogramm testen kann. Mit Hilfe dieser Tests kann man dann auch recht gute Aussagen über die Spielstärke des jeweiligen Programms treffen.

5.1 Positionelle Bewertung

Ein sehr kritischer Bereich der Bewertung ist, wie wir bereits kennengelernt haben, die positionelle Bewertung. Hier ist das "Schachwissen" des Programms zusammengefaßt, hier werden in ruhigen Stellungen die Züge ausgewählt. Auf Grund der hohen Rechenzeitanforderungen bewerten aber viele Programme nicht bis in beliebige Tiefen positionell[45], sondern nur bis z.B. in die Tiefe 2. Wird allerdings nur in der ersten Tiefe positionell bewertet, dann wird das Programm nicht sonderlich stark spielen - es würde mir dann besonders mülltonnenverdächtig erscheinen. Mit folgender Teststellung läßt sich feststellen, ob ihr Programm auch weiter als bis zur Tiefe 1 positionelle Bewertungen vornimmt:



[45] Das macht auch Salomon nicht! In der Ruhesuche werden nur mehr Materialabschätzungen gemacht - etwas anderes ist dort auch von geringem Interesse

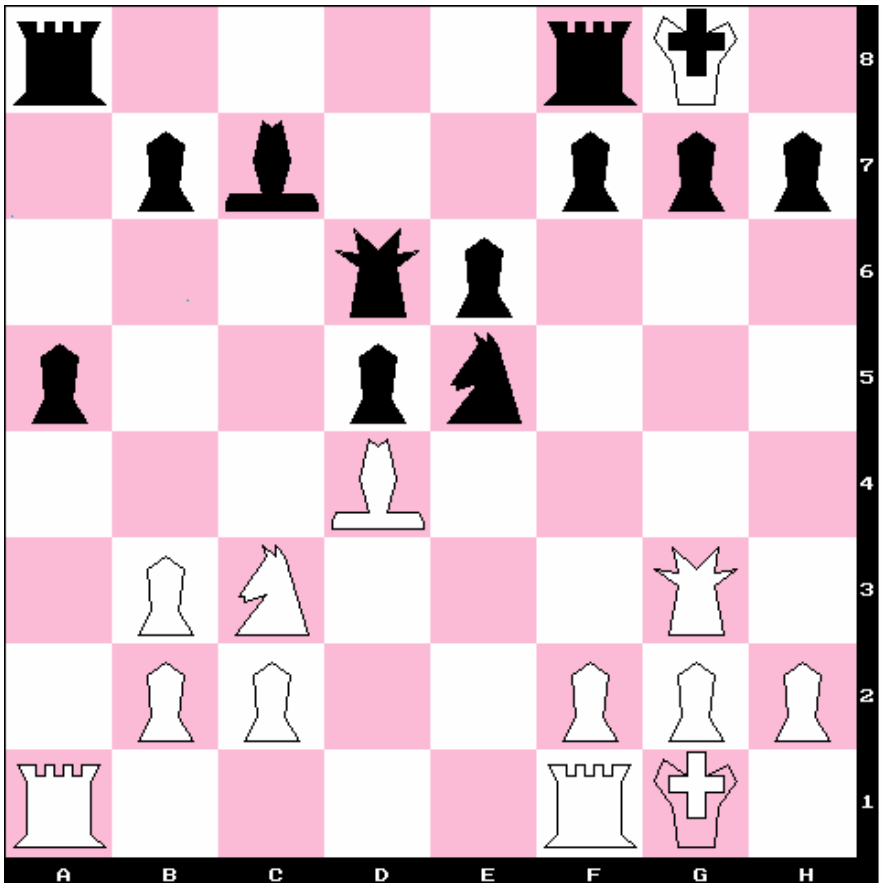
Schwarz am Zug sollte diese Stellung leicht Remis halten. Legt man diese Aufgabe jedoch einem Programm vor, das nur auf erster Ebene positionelle Bewertungen vornimmt, passiert folgender Katastrophenzug:

1. ... Sc4+??

Damit ist die schwarze Stellung völlig ruiniert! Nach 2. Sd2xc4 b5xc4 verliert Schwarz auch noch den Bauern auf c4 (weißer König ist schneller!) und bald auch die Partie. Anders jedoch bei Programmen, die auch tiefer positionelle Bewertungen anstellen: z.B. Salomon™ (Level 2) spielt 1. ... Ke6, was das Remis auf jeden Fall nicht in Frage stellt. Sargon antwortet mit 1. ... a7-a5.
Warum spielt ein Programm, das nur auf erster Ebene positionelle Bewertungen anstellt, den Zug 1. ... Sc4+? Das kommt daher: mit Sc4+ erreicht das Programm ein starkes Feld, von dem es mehrere weiße Figuren bedrohen kann, in positioneller Hinsicht erscheint der Zug also super, in den folgenden Tiefen werden nur mehr materielle Betrachtungen angestellt, das Programm findet 2. Sd2xc4 b5xc4, also einen materiellen Gleichstand. Somit gilt für das Programm Sc4+ als der beste Zug!

5.2 Kombinatorik

Ein weiteres Kapitel zum Thema Teststellungen bietet die Kombinatorik an. Dies ist eigentlich die große Stärke von Schachprogrammen, sofern sie tief genug rechnen können. Folgende Teststellung können sie ihrem Programm einmal vorlegen:



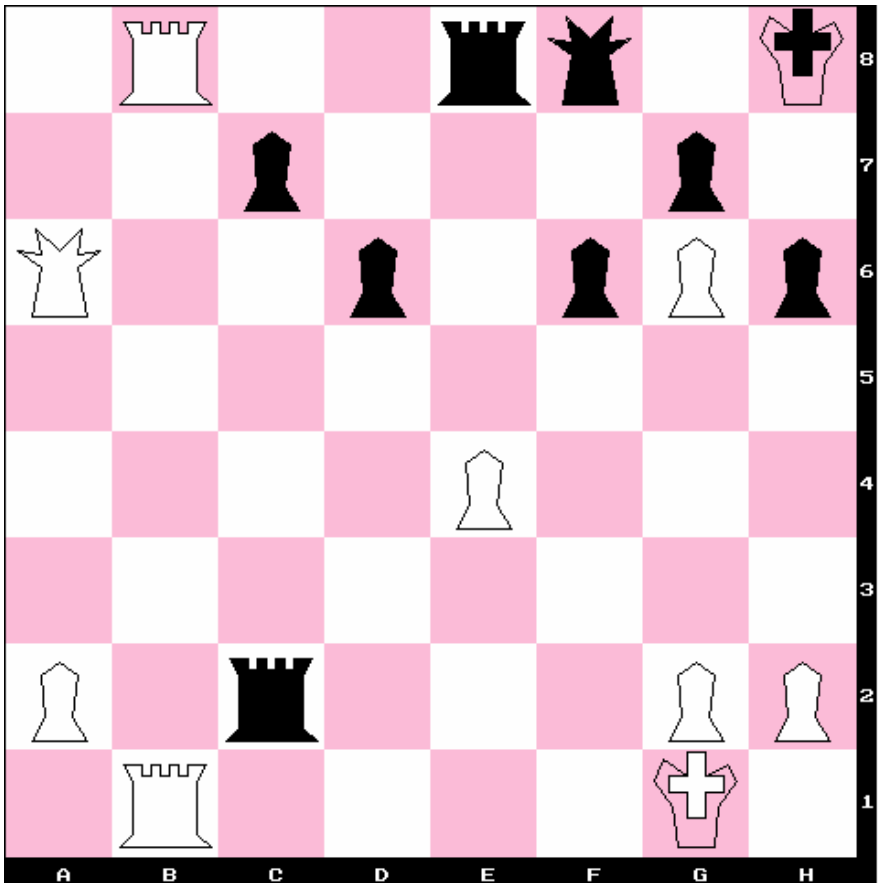
Die Ausgangslage ist sehr verzwickt. Weiß am Zug muß allerhand Drohungen und Fesselungen berücksichtigen. So droht z.B. ein Matt, falls Schwarz mit dem Springer von e5 flüchten würde[46], auch Sb5 (zwei Drohungen) und Lxe5 bzw. Dxe5 sollten in Betracht gezogen werden. Programme, die nicht sonderlich tief rechnen (nur zwei Halbzüge voraus) werden wahrscheinlich Lxe5? spielen, was aber nichts einbringt. In Wirklichkeit (na, wissen sie schon welcher Zug der richtige ist?!) ist der Zug Sb5 der Beginn eines Gewinnweges:

Weiß: Ich
Schwarz: Chessmaster 2100

1. Sc3-b5! Dd6-b4 3. Sc7xa8
2. Sb5xc7 Db4xd4

Und Weiß hat eine Qualität gewonnen! Aber auch die folgende Stellung können sie einmal ihrem Programm vorlegen, sie entstand während des berühmten Turniers von Hastings 1919. Der spätere Weltmeister Jose Raoul Capablanca spielte gegen einen ziemlich unbekannten Briten:

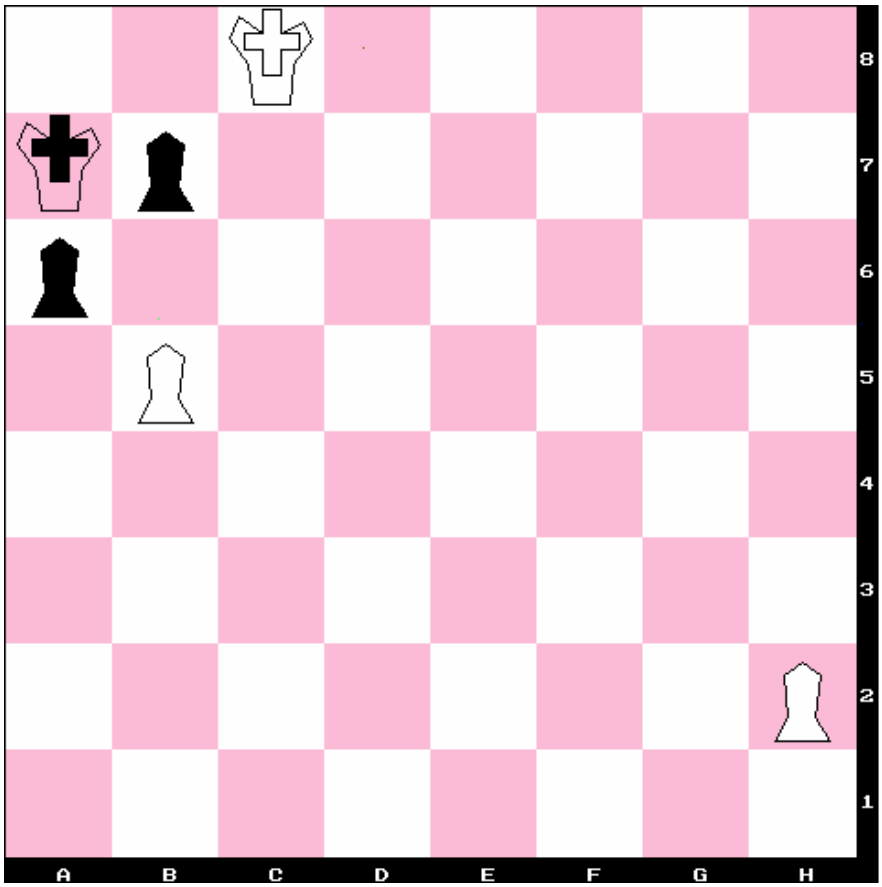
[46] wobei allerdings f4? mit Sg6 beantwortet werden würde!



Scheinbar plausibel spielte Capablanca hier Da6-a8?, was allerdings einen Bauernverlust nach sich zieht. Geben sie doch einmal ihrem Programm (oder Salomon™) diese Stellung und probieren sie zuerst, welchen Zug das Programm machen würde und dann, was das Programm auf Da8? antwortet.
Wie sie vielleicht schon bemerkt haben, kann man Da8? nämlich mit Tc2xa2! beantworten, womit mit einem Mehrbauern das Endspiel schon gesichert sein könnte. Salomon™ (Level 1) spielt übrigens nicht Da8? sondern Tb8xe8, und falls man ihn mit Schwarz spielen läßt, findet er die richtige Erwiderung Tc2xa2 auch auf niedrigster Spielstufe!

5.3 Rechentiefe

Ein weiteres wichtiges Kriterium für die Spielstärke von Programmen stellt sicherlich die maximale Rechentiefe dar. Je weiter ein Programm vorausrechnen kann, desto besser ist es natürlich. Vor allem in Endspielen ist das oft sehr wichtig, hier ist die Anzahl der Möglichkeiten meist so beschränkt, daß mit dem gleichen Rechenaufwand wie im Mittelspiel viel weiter vorausgerechnet werden kann. Folgende Teststellung hätte ich anzubieten:



Was würden sie hier (Weiß am Zug) spielen? Doch nicht etwa b5xa6?? Denn dieser Zug verliert:

1.	b5xa6??	b7-b5!!	5. h6-h7	b2-b1D
2.	h2-h4	b5-b4	6. h7-h8D	Db1-b8+!
3.	h4-h5	b4-b3		
4.	h5-h6	b3-b2		

Aber Weiß kann auch gewinnen (wissen sie auch schon wie?):

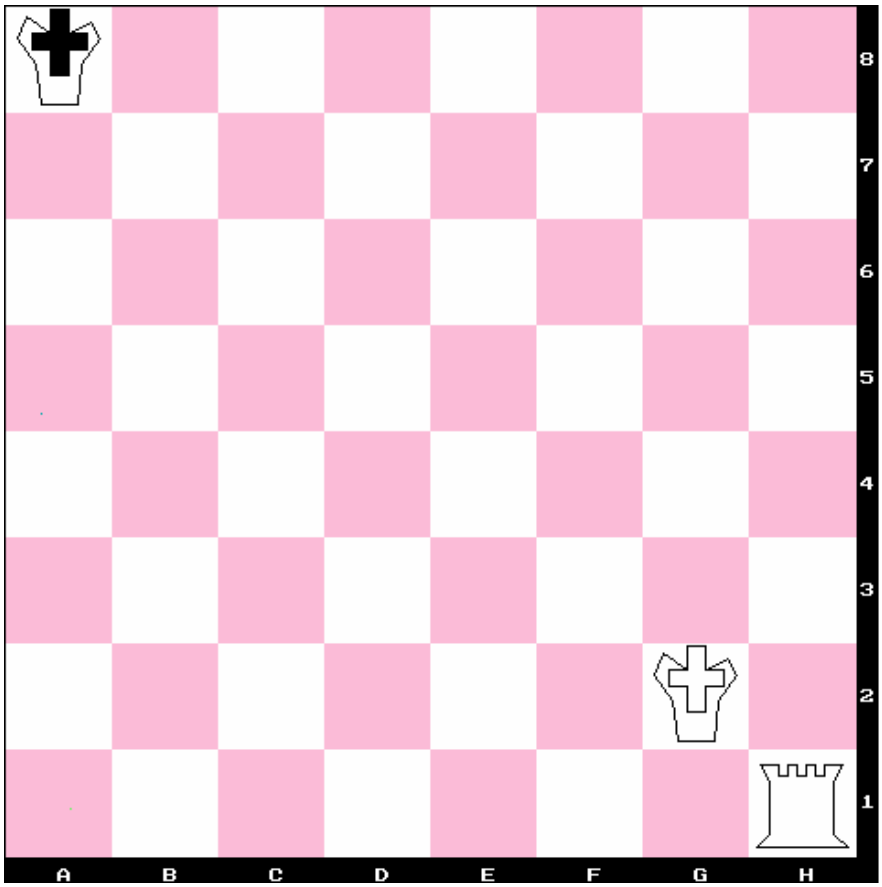
1.	b5-b6+!!	Ka7xb6	5. h6-h7	a3-a2
2.	h2-h4	a6-a5	6. h7-h8D	
3.	h4-h5	a5-a4		
4.	h5-h6	a4-a3		

Um diese Folge zu erkennen müßte das Programm aber mindestens 13 Halbzüge vorausrechnen, und das ist selbst nach heutigen Maßstäben noch gewaltig! Geben sie ihrem Programm also genügend Zeit[47] und probieren sie die obige Stellung. Falls es die richtige Entgegnung findet, rechnet ihr Programm also mindestens 13 Halbzüge tief

5.4 Endspiele

Bezüglich Endspiel kann man von den kommerziellen Programmen nicht allzuviel erwarten, trotzdem sollte ein Programm zumindest Damen- und Turm-Endspiele siegreich beenden können. Versuchen sie doch einmal folgende Stellung einzugeben, Weiß am Zug, sie spielen Schwarz:

[47] Wollen sie die Stellung mit Salomon™ testen, so stellen sie mindestens Level 13 ein und geben sie ihm ca. 45 Minuten Zeit. Auch er wird dann den richtigen Zug finden!

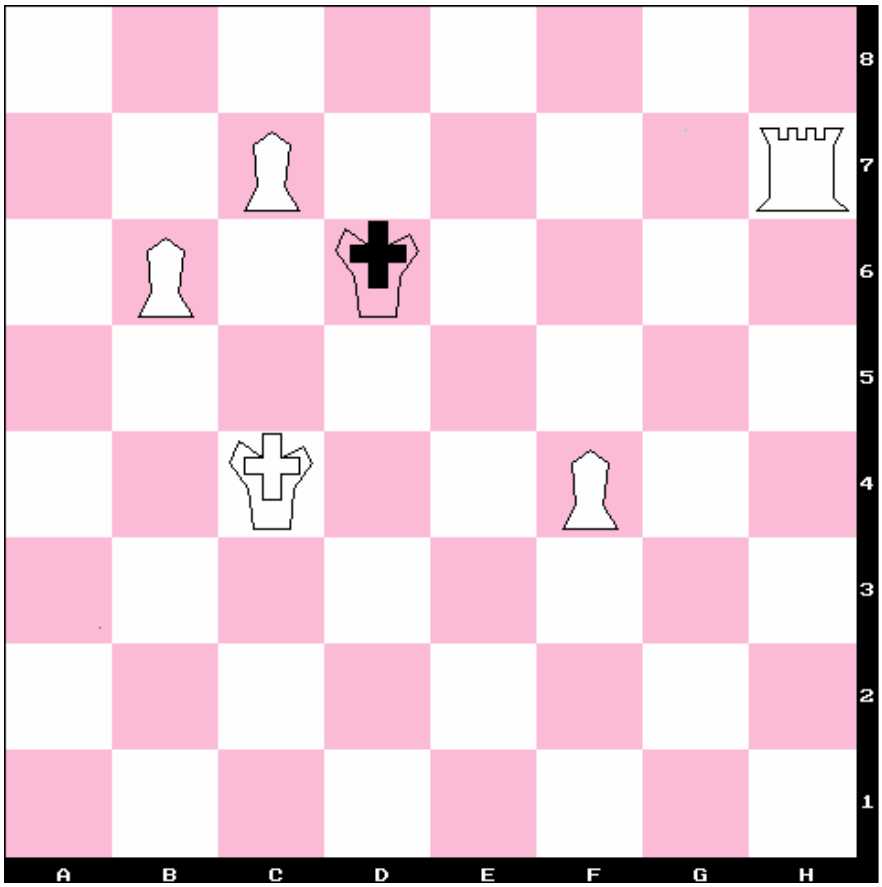


Falls das Programm nie weiter als 4 Züge vorausrechnet, wird es nie ein Matt forcieren können. Aber auch grundsätzliche Dinge wie das Abdrängen des Königs an den Rand und Unterstützung des Turmes mit dem eigenen König sollten vom Programm (auch auf niedrigstem Niveau!) beherrscht werden[48]. Selbstverständlich können sie auch mit Salomon™ diesen Test starten, das Programm verfügt ja, wie wir bereits wissen, über eine eigene Mattroutine in der Bewertung, die genau das oben Geforderte (nämlich abdrängen und Unterstützung) leistet.

5.5 Verwandlungen

Auch Verwandlungen sind ein heikles Thema im Computerschach, so gab es doch (z.B. auf dem C-64) genug Programme, die konstant nur Damen eintauschten, egal welche Stellung auch vorlag. Selbstverständlich sollte ein gutes Programm auch eine der Stellung entsprechende Figur eintauschen, also z.B. kein Patt erzeugen, wenn es sich schon im Vorteil befindet. Auch dazu gibt es eine Teststellung:

[48] Testen sie die Stellung mit Salomon™, so sollten ab Level 4 oder 5 keine Probleme auftreten



Weiß am Zuge darf auf keinen Fall c7-c8D spielen, denn das wäre ein Patt (und da würde sich Schwarz wohl ziemlich freuen!). Der richtige Zug ist vielmehr:

1. c7-c8L!! Kd6-c6
2. Th7-h6++

Der richtige Zug hat also sofort ein Matt zur Folge. Auch Salomon™ erkennt dies (Level 3) und kündigt dies sogar an! Gibt man ihm einen geringeren Level, spielt er c7-c8T aber **nicht** den obigen Katastrophenzug. Auch SARGON V und die beiden Chessmaster Programme überstanden obigen Test problemlos, die Antwort kam geradezu blitzartig.

5.6 Tips für den Sieg gegen den Computer

Wenn gerade sie frustriert sind, weil ihr Programm sie ständig besiegt, dann ist das genau der richtige Abschnitt für sie! In diesem letzten Kapitel möchte ich noch ein paar Tips geben, wie man gegen Programme spielen sollte, damit man sie auch schlägt[49]. Auch bei unseren internen Clubmeisterschaften spielte ein Schachcomputer mit (ich glaube er war von Fidelity) - und das unglaubliche geschah: ein Compu-

[49] Da Programme immer öfter bei Turnieren auftauchen, ist die Wahrscheinlichkeit gar nicht so gering, daß man einmal gegen einen Computer antreten muß

ter gewann unsere Meisterschaft (allerdings spielte ich damals nicht mit ...)!! Was sollte man also alles beachten, spielt man einmal in einem Turnier gegen einen Computer?

Regel 1: *Spielen sie ungewöhnliche Eröffnungen*

Je eher das Programm aus seiner Bibliothek gerissen wird, desto eher muß es Rechenzeit aufwenden, und die ist vor allem in der Eröffnung und im Mittelspiel enorm. Gerade hier sind die Stellungen sehr komplex und die Möglichkeiten mannigfaltig!

Regel 2: *Spielen sie Gambitvarianten*

Gerade in positioneller Hinsicht haben die Programme ihre Schwächen, sie werden sich also in der Regel auf Gambitbauern regelrecht stürzen (sofern diese Eröffnungsvariante nicht in der Bibliothek ist) und sich damit natürlich unwiderruflich in positionellen Nachteil begeben. Diesen zu nutzen liegt nun an ihnen! So erzielt man z.B. mit dem nordischen Gambit gegen viele Programme erstaunliche Erfolge (auch gegen Psion), gegen andere wiederum hilft es gar nichts, sie wissen auf Grund ihrer Bibliothek, wie man wieder ausgleicht.
Aber auch das Königsgambit kann sehr von Vorteil sein (eine äußerst beliebte Eröffnung von mir).

Regel 3: *Vermeiden sie im Mittelspiel taktische Verwicklungen*

"Wer wenig tut, macht wenig Fehler" heißt es doch so schön. Also provozieren sie nicht verzwickte Stellungen wie z.B. im Kapitel 5.2. Hier ist das Programm besonders stark, Schlagzüge werden ganz genau analysiert. Spielen sie also möglichst ruhig.

Regel 4: *Provozieren sie in verwickelten Situationen Abtäusche*

Sollten sie dennoch einmal in eine knifflige Situation rutschen, so versuchen sie möglichst schnell durch Abtäusche die Stellung zu "reinigen". Wenn möglich retten sie sich ins Endspiel, hier liegt meistens die große Schwäche aller Programme! Vor allem wenn sie schon einen kleinen Vorteil errungen haben (z.B. einen Doppelbauern in der Bauernstruktur des Computers), sollten sie diese Regel ganz besonders beherzigen.

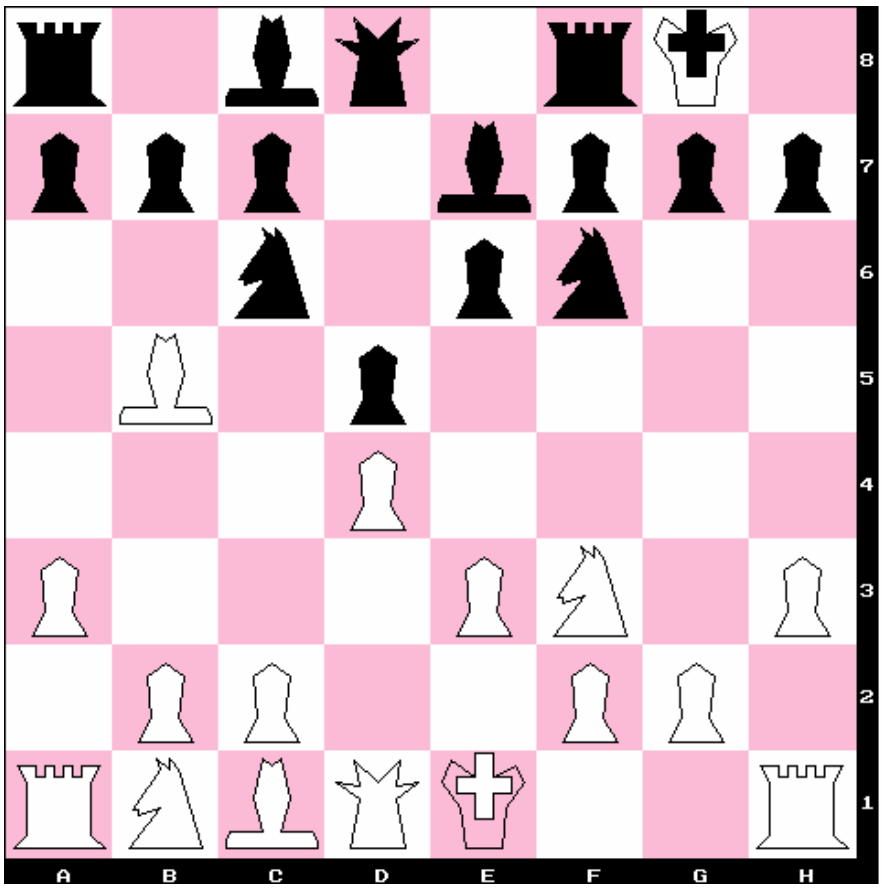
Regel 5: *Stellen sie ihrem Programm Fallen*

Gerade im Mittelspiel ist die Stellung oft so komplex, daß das Programm nicht allzuweit blicken kann. Es bietet sich also Gelegenheit, den Horizonteffekt auszunutzen, indem man raffinierte Fallen stellt, deren Pointe das Programm nicht mehr sehen kann. Eine solche Falle möchte ich ihnen gerne vorstellen - und ich möchte auch gleich anfügen: jedes mir bekannte Programm ist noch darauf hereingefallen!!

Die Falle nennt sich: *Trojanisches Pferd.* Der Name ist, wie sie gleich sehen werden, äußerst treffend gewählt:

Weiß: Spieler
Schwarz: Computer

1. d2-d4	Sg8-f6
2. e2-e3	d7-d5
3. h2-h3	Sb8-c6
4. Lf1-b5	e7-e6
5. a2-a3	Lf8-e7
6. Sg1-f3	0-0



Bis jetzt war es noch nicht sehr aufregend, ja scheint Weiß geradezu unsinnig zu spielen, da völlig passiv und planlos. Doch es kommt ganz anders:

7.	Sf3-g5?!	h7-h6
8.	h2-h4!!	h6xg5?? ein grober Fehler, wie sich gleich zeigen wird!
9.	h4xg5	Sf6-e4
10.	Dd1h5!!	f7-f5 hilft auch nicht mehr...
11.	g5-g6	Le7-b4+ Verzweiflung...
12.	a3xb4	Sc6xb4
13.	Dh5-h7++	

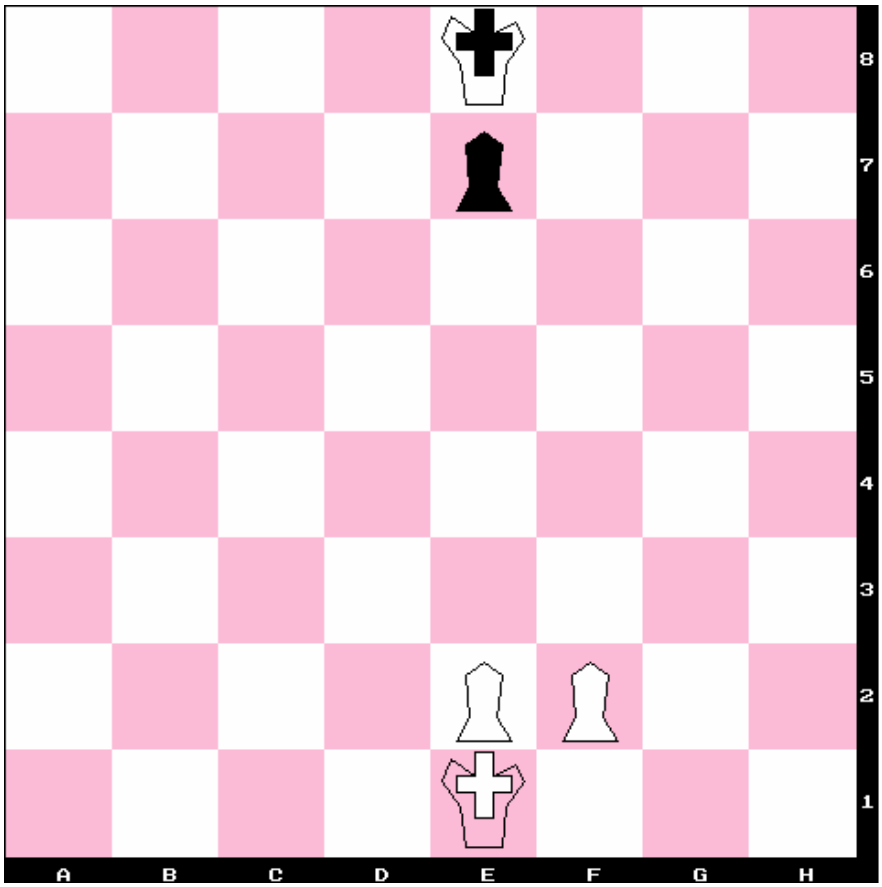
So schnell kann sich das Blatt wenden, aus einer scheinbar harmlosen Eröffnung wurde plötzlich ein Matt forciert!! Und alles wurde mit dem Springerzug Sf3-g5 eingeleitet, deswegen auch der Name trojanisches Pferd.
Warum kann so etwas passieren? Beachten wir zunächst: der Mensch verfolgt einen eindeutigen Plan und zwar vom ersten Zuge an, der stets auf Matt ausgerichtet ist. Das Matt soll mit Hilfe der geöffneten h-Linie erfolgen, dazu muß er allerdings einige Teilziele verwirklichen. Erstens muß Schwarz kurz rochieren, zweitens muß der schwarze Springer auf f6 vorhanden sein, drittens soll sich kein schwarzer Läufer im Felderrechteck f8-h8-h3-f3 befinden und viertens muß der eigene Springer auf f3 plaziert sein.
Um diese Ziele zu erreichen, macht Weiß so seltsame Züge wie 3. h2-h3 verhindert z.B. Lc8-g4 oder 5. a2-a3 gaukelt dem Programm einen Tempogewinn vor, den es auch sogleich mit der (von uns gewünschten!) Rochade zu nutzen versucht.
Mit Sf3-g5 schließlich wird die eigentliche Falle eingeleitet, Schwarz kann nun mit h7-h6 den lästigen Springer vertreiben und gleichzeitig noch ein Tempo gewinnen, was er natürlich als gut erachtet und tut. Mit h2-h4 kann man dann freie Bahn auf der h-Linie schaffen, den Springer zu gewinnen ist für das Programm natürlich viel zu verlockend. Die Folgen des Springerklaus kann das Programm aber innerhalb seines Horizonts nicht sehen, denn es folgen keine weiteren Schlagzüge auf diesem Feld. Nichtsahnend laufen hier alle mir bekannten Programme in ihr Verderben, zu komplex ist die Stellung, um derartig tief analysieren zu können! Sobald die weiße Dame auf h5 erscheint ist es aus, dann gibt es keine Rettung mehr.
Eine weitere Möglichkeit, Fallen zu stellen, wären z.B. Fesselungen. Vor allem die älteren SARGON Programme nahmen diese oft zu leicht und schlitterten dadurch oft in wilde Angriffe.

Regel 6: *Spielen sie im Endspiel nach einem konsequenten Plan*

Pläne sind meistens situationsbezogen, ein genereller Plan für das Endspiel könnte aber beispielsweise der folgende sein:

"Ich will versuchen, meine Bauern zur achten Reihe zu bringen und gleichzeitig den gegnerischen König daran hindern, diese zu schlagen"

Doch gerade im Endspiel erkennt man die Meister dieses Spiels, hier zeigt sich oft, wie gut ein Schachspieler wirklich ist. Bis zu 20 Halbzüge und mehr gilt es doch in vielen Stellungen vorauszurechnen, und auch die berühmte Intuition spielt hier natürlich mit. Dennoch möchte ich ein kleines Beispiel bringen, wie man einen Plan im Endspiel fassen kann. Gegeben sei die folgende Stellung:



In dieser Stellung könnte man folgenden Plan fassen:

"Der weiße König muß vor seine Bauern gelangen, um den schwarzen Monarchen zu verdrängen. Da dies der schwarze König ebenfalls versucht, muß Weiß schneller sein. Erreicht der weiße König den schwarzen Bauern, bevor der schwarze König hinter die weißen kommt, gewinnt er."

Und so könnte eine mögliche Fortsetzung aussehen:

1. Ke1-d2	e7-e5	8. f3-f4	Kf7-e7
2. Kd2-d3	Ke8-f7	9. f4-f5	Ke7-f7
3. Kd3-c4	Kf7-f6	10. Ke5-f4	Kf7-e7
4. e2-e4	Kf6-g5	11. e4-e5	Ke7-f7
5. Kc4-d5	Kg5-f6	12. e5-e6+	Kf7-f6
6. f2-f3	Kf6-g6	13. Kf4-e4	Kf6-e7
7. Kd5xe5	Kg6-f7	14. Ke4-e5	... usw., Weiß gewinnt

Schwarz hat hier nicht ganz optimal gespielt, viel mehr hätte er aber dennoch nicht herausholen können. Weiß kann seinen Mehrbauern durchbringen, aber eben nur, wenn ein fester Plan vorhanden ist. Hätte Weiß unbedacht gespielt, wäre z.B. auch folgender Spielverlauf denkbar gewesen:

1. Ke1-d2 Kd8-d7
2. Kd2-d3 Kd7-d6
3. f2-f4 ...

Dies ist entgegen dem weißen Plan, den König **vor** die Bauern zu bringen, aber schauen wir weiter...

3. ... Kd6-d5
4. e2-e4+ ...

Auch dieser Zug fördert nicht den Plan, sondern macht quasi einen Strich durch ihn. Auf diese Weise kann der weiße König nicht vor seine Bauern gelangen!

4. ... Kd5-d6
5. Kd3-d4 ...

Und jetzt hat Weiß auch noch den Gewinn verschenkt, der weiße Plan wurde nicht konsequent ausgeführt und ist daher gescheitert. Es folgt nämlich:

5. ... e7-e5+
6. Kd4-e3 e5xf4+
7. Ke3xf4 Kd6-e6
8. e4-e5 ...

Meister des Schachspiels erkennen sofort, daß man sich nun zurückziehen muß, will man ein Remis halten...

8. ... Ke6-e7

Und Schwarz hat das Beste, was in dieser Situation für ihn noch möglich war, erreicht, nämlich ein Remis! Sie sehen, so leicht kann man im Endspiel einen Sieg verschenken, geht man nicht konsequent nach einem Plan vor.

Diese Tips sollten übrigens nicht nur gegen Computer angewandt werden, auch ihre Spielstärke läßt sich damit bestimmt verbessern (sofern sie nicht gerade ein Großmeister sind). Wenngleich auch gegen Menschen das eine oder andere Rezept (z.B. Horizonteffekt) nicht wirkt, viele Regeln lassen sich aber dennoch auch gegen Menschen anwenden, probieren sie es doch aus!
Somit sind wir am Ende angelangt, sie haben nun das grundsätzliche Verständnis, selbst ein Programm zu entwickeln bzw. käufliche Programme zu "verstehen". Außerdem haben sie nun eine ganze Reihe von Teststellungen, mit denen sie ihr Programm auf seine Spielstärke testen können - ich hoffe allerdings, daß zumindest der eine oder andere auch sein eigenes Schwachwissen ein wenig erweitern konnte.

6. Literaturverzeichnis

Leider gibt es zum Thema "Computerschach" wenig Literatur, vor allem keine deutschsprachige. Vieles floß auch aus eigener Erfahrung in diese Arbeit ein. Trotzdem seien hier einige Referenzen aufgelistet, die allerdings teilweise recht schwer zu bekommen sind.

H.J.Kraas, G.Schrüfer — *"Das große Computerschachbuch"*
Data Becker Verlag, Düsseldorf 1985

C.E.Shannon [SHA1] — *"Programming a Computer for playing Chess"*
Philosophical Magazine 41/1950
Seiten 256 - 275

C.E.Shannon [SHA2] — *"Programming a Computer to play Chess"*
Scientific American 2/1950
Seiten 48 - 51

D.Slate, L.Atkin — *"CHESS 4.5- The Northwestern University Chess Prg."*
Springer Verlag, New York 1978

A.Bernstein — *"A Chessplaying Program For IBM 704 Computer"*
Proclamation Western Joint Computer Conference 1958
Seiten 157 - 159

D.Levy — *"Chess and Computers"*
Batsford Chess Books, London 1976

P.Frey — *"Chess Skill in Man and Machine"*
Springer Verlag, New York 1978

M.Gittel — *"SARGON - Porträt eines Schachprogrammes"*
Eigenverlag, Salzgitter 1983

G.Pachmann, — *"Computerschach"*
ECON Verlag, München 1980

M.Newborn D.Kopec — *"21st ACM Computer Chess Championship"*
Communications of the ACM 11/1991
Seiten 85 - 92

M.Newborn D.Kopec — *"20 th ACM Computer Chess Championship"*
Communications of the ACM 7/1990
Seiten 93 - 104

B.Owsnicki *"Repräsentation von positionellem Schachwissen mit KI"*

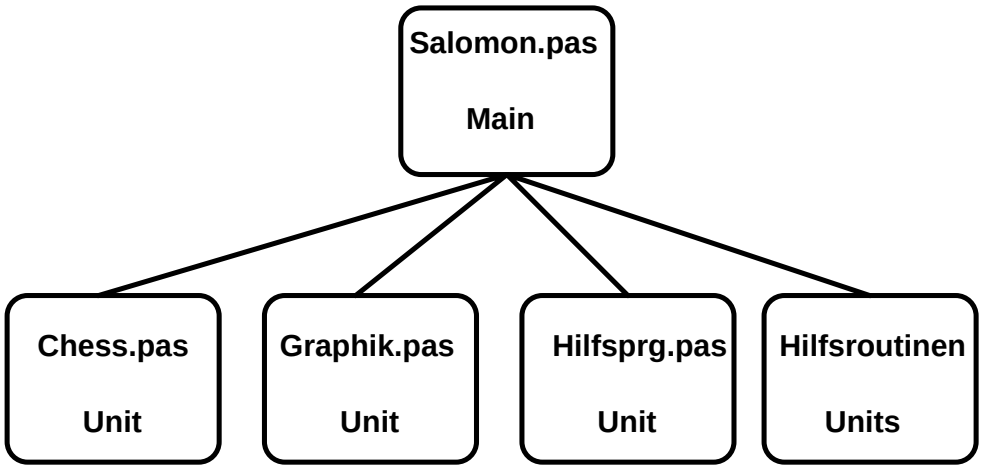
Dissertation, TU-Hamburg 1985

Siegbert Tarrasch *"Das Schachspiel"*
Carl Habel Verlag, 10.Auflage, Darmstadt 1972

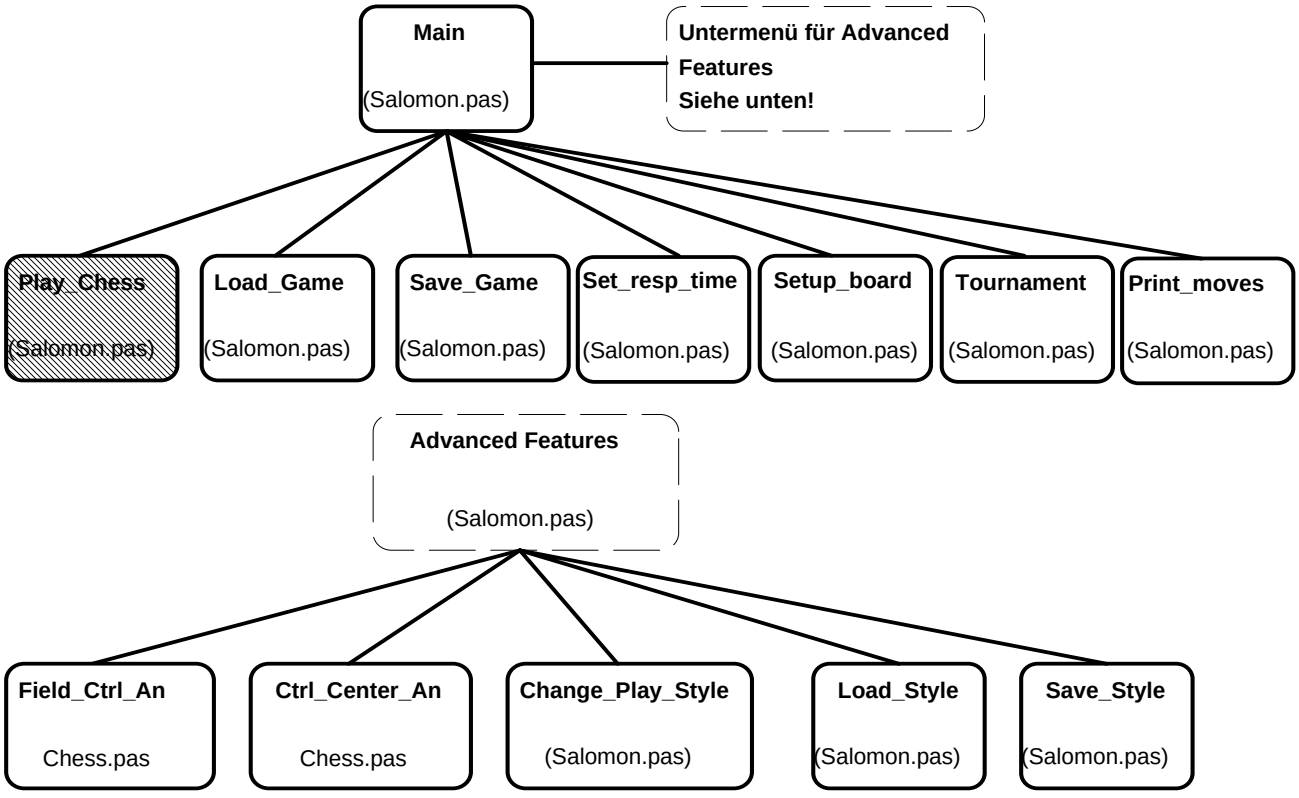
M.M.Botvinnik *"Meine neuen Ideen zur Schachprogrammierung"*
Springer Verlag, Berlin-Heidelberg-New York 1982

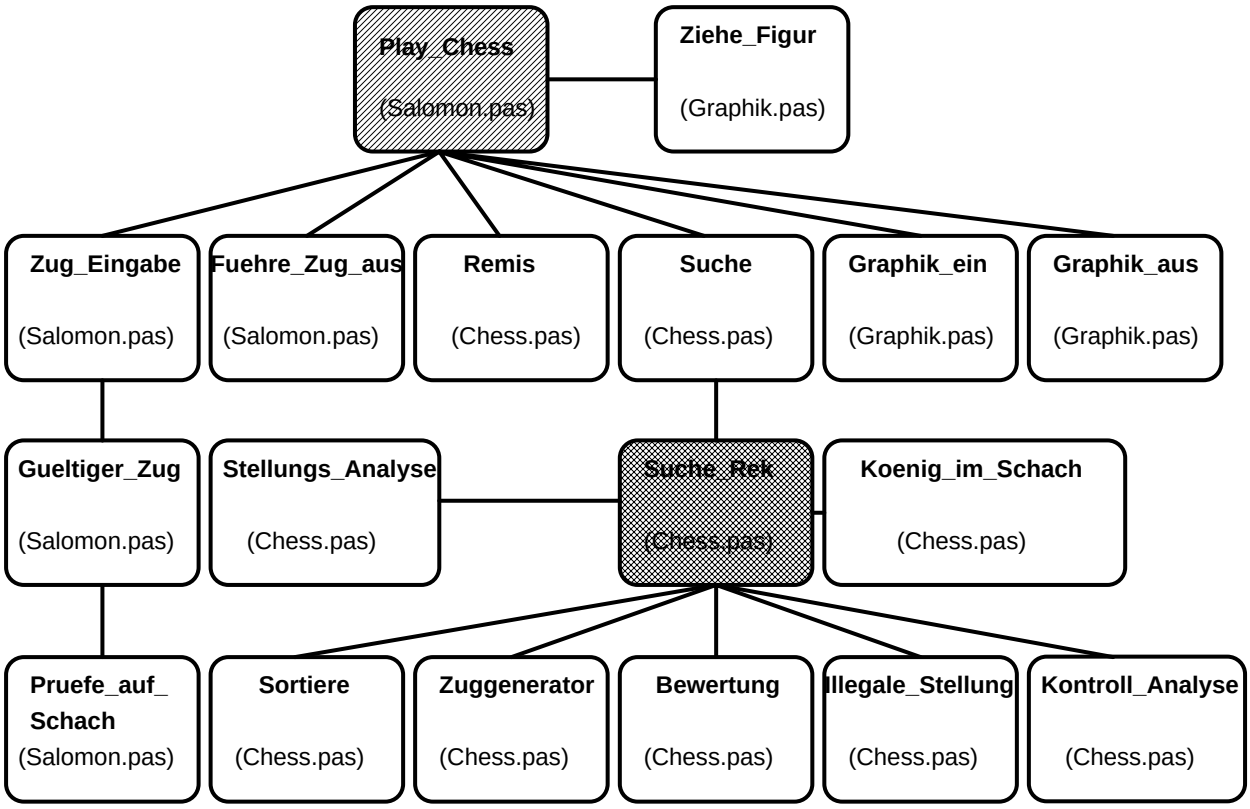
1. Anhang: Genaue Struktur des Programmes

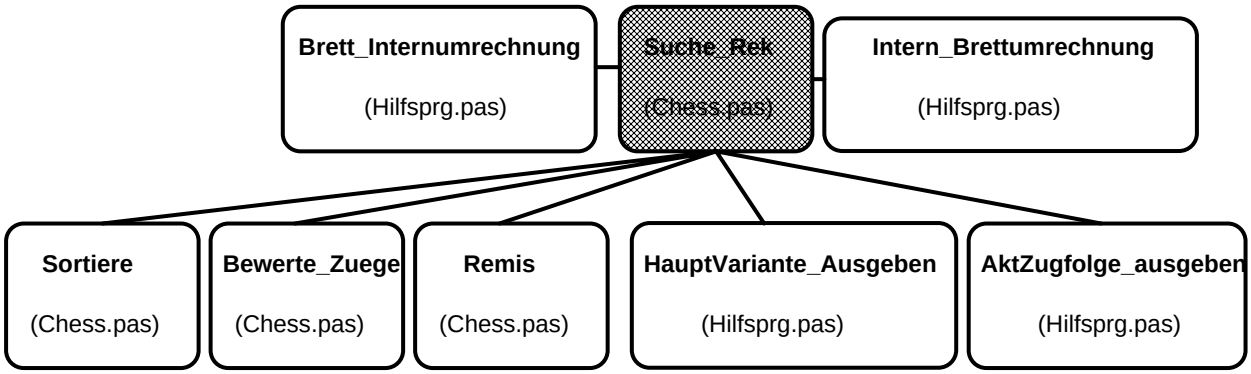
Die folgende Abbildung gibt die Programmteile (samt Hierachie) von Salomon™ wieder. Die Hilfsroutinen bestehen wiederum aus Units, die im einzelnen folgendermaßen heißen: *Mouse.pas, Utility.pas, Util_ii.pas, Windunit.pas*



In den nächsten Abbildungen sehen sie nun eine Auflistung der Routinen und deren Zusammenspiel (in Klammer ist das Programmodul vermerkt, welches die jeweilige Routine enthält):




Play_Chess
(Salomon.pas)
Ziehe_Figur
(Graphik.pas)
Zug_Eingabe
(Salomon.pas)
Fuehre_Zug_aus
(Salomon.pas)
Remis
(Chess.pas)
Suche
(Chess.pas)
Graphik_ein
(Graphik.pas)
Graphik_aus
(Graphik.pas)
Gueltiger_Zug
(Salomon.pas)
Stellungs_Analyse
(Chess.pas)
Suche_Rek
(Chess.pas)
Koenig_im_Schach
(Chess.pas)
Pruefe_auf_
Schach
(Salomon.pas)
Sortiere
(Chess.pas)
Zuggenerator
(Chess.pas)
Bewertung
(Chess.pas)
Illegale_Stellung
(Chess.pas)
Kontroll_Analyse
(Chess.pas)


Brett_Internumrechnung
(Hilfsprg.pas)
Suche_Rek
(Chess.pas)
Intern_Brettumrechnung
(Hilfsprg.pas)
Sortiere
(Chess.pas)
Bewerte_Zuege
(Chess.pas)
Remis
(Chess.pas)
HauptVariante_Ausgeben
(Hilfsprg.pas)
AktZugfolge_ausgeben
(Hilfsprg.pas)

2. Anhang: Leistungsbetrachtungen

In den folgenden Tabellen habe ich die wichtigsten Routinen, die während der Zugberechnung aufgerufen werden zusammengestellt. Außerdem wurde eine Zeitmessung vorgenommen, wieviel Rechenzeit (in %) eine Routine benötigt wird. Diese Messungen wurden mit dem *Turbo Profiler 1.1* von Borland durchgeführt, und zwar sind eine Anfangsstellung (Tiefe 3), eine Mittelstellung (Tiefe 2) und eine Endstellung (Tiefe 3) betrachtet worden. Zur Erinnerung: Die Routine *Bewerte_Zuege* dient zum **Vor**bewerten der Züge (werden anschließend mit Quicksort sortiert), die Routine *Bewertung* ist dann die eigentliche Routine, welche die positionelle Wertung vornimmt!

Eröffnungsstellung:

Name der Routine	Benötigte Zeit (Sekunden)	Prozent
Suche_Rekursiv	1.4439	49
Bewertung	1.1301	38
Zuggenerator	0.0472	2
Sortiere	0.0000	<1
Bewerte_Zuege	0.2832	10

In der Eröffnungsphase hat die Bewertung noch einen Löwenanteil, da von der Suche noch nicht genug Züge abgeschnitten werden können. Die Suche leistet übrigens in allen drei Phasen des Spiels die meiste Arbeit, diese reicht bekanntlich ja vom Alpha-Beta Cut über Feststellung eines Remis, Patts, Matts bis hin zur Eliminierung von illegalen Stellungen und Erkennung von Schachgeboten. Besonders kurz ist der Zuggenerator, der ja auf Zeit optimiert wurde (und nicht auf Intelligenz).

Mittelspielstellung:

Name der Routine	Benötigte Zeit (Sekunden)	Prozent
Suche_Rekursiv	5.6061	53
Bewertung	0.4233	5
Zuggenerator	1.2298	12
Sortiere	0.0133	<1
Bewerte_Zuege	3.1156	29

Die Bewertung ist deutlich zurückgegangen, die Suche tut ihre Wirkung. Besonders hervorstechend ist auch die Routine Bewerte_Zuege, ohne die allerdings die Suche in die Irre geführt werden würde (das Vorbewerten kostet eben ganz schön Zeit, dadurch wird aber auch die Anzahl der untersuchten Stellungen drastisch reduziert).

Endspielstellung:

Name der Routine	Benötigte Zeit (Sekunden)	Prozent
Suche_Rekursiv	0.5996	74
Bewertung	0.0853	9
Zuggenerator	0.0168	4
Sortiere	0.0000	<1
Bewerte_Zuege	0.1037	12

Auch hier hat die Bewertung keinen großen Anteil mehr, da die Anzahl der Figuren sehr reduziert wurde. Außerdem läuft die Mattroutine relativ rasch ab. Wieder ist deutlich zu sehen, daß die Hauptaufgabe in der Suche liegt. Hier werden die wichtigen Züge herausfiltriert!

Betrachtet man diese Tabellen, so kann man sicher einmal sagen, daß in der Eröffnungsphase eine Eröffnungsbibliothek ungemein helfen würde. Denn gerade hier wird eine Menge Rechenzeit investiert, die man (bei Turnierpartien) wesentlich effizienter im Mittel- oder Endspiel einsetzen könnte. Leider ist der Aufwand (für eine wirklich gute Bibliothek - und nur eine solche nützt etwas - müßte man tausende von Stellungen eingeben) sehr beträchtlich, und deswegen habe ich bei Salomon™ bisweilen darauf verzichtet.
Der zweite Verbesserungspunkt ist wohl eine Beschleunigung der Suche, die ja in allen Phasen den größten Anteil der Rechenzeit in Anspruch nimmt. Hier könnte eine mögliche Verbesserung (genügend Speicher vorausgesetzt) ein Hashingtable sein, somit müßten bereits untersuchte Varianten nicht immer wieder neu kreiert werden (das probeweise Zugausführen kostet sehr viel Zeit!), sondern würden einfach aus dem Speicher geladen werden - der Geschwindigkeitszuwachs wäre sicherlich enorm.
Weiters wäre eine Art *permanent brain* denkbar, d.h. Salomon™ denkt ebenso während der Zeit des Gegners. Auch das ist nicht gerade leicht, da man hier zwei Tasks (Zugeingabe des Spielers und Suche) gleichzeitig laufen lassen muß. Eine mögliche Implementation könnte so aussehen, daß einfach der beste Erwiderungszug aus der Hauptvariante genommen und probeweise ausgeführt wird. Macht der Spieler dann tatsächlich diesen Zug, so kann Salomon™ die Suche gleich an dieser Stelle fortsetzen[50], bzw. falls der Spieler einen anderen Zug ausführt, wird das ganze einfach vergessen und ein normaler Suchgang gestartet.
Eine letzte Verbesserung wäre auch in der Sortierung der Züge anzubringen. Vielleicht sind ja gerade sie ein Schachgenie und glauben, daß man die Züge in einer ganz anderen Reihenfolge betrachten müßte als es Salomon tut (um mehr Cut-Offs zu erhalten). Bitte, ihrer Kreativität sind keine Grenzen gesetzt! Versuchen sie ihre eigene Ideen zu verwirklichen, hier liegt sicher noch eine große Schwachstelle von Schachprogrammen.

[50] Falls die Tiefe schon groß genug ist, kann er eventuell *sofort* antworten

3. Anhang: Durchschnittliche Antwortzeiten

In der folgenden Tabelle habe ich die durchschnittlichen Antwortzeiten von Salomon™ (für die ersten fünf Spielstufen) zusammengefaßt. Die Ergebnisse stammen von einem 80486 / 50Mhz Computer (mit Coprozessor). Auf kleineren Prozessoren wird es natürlich etwas länger dauern (speziell auf 80286 basierenden Systemen). Außerdem muß noch angemerkt werden, daß diese Zeiten natürlich nicht immer dieselben sind, speziell im Mittelspiel braucht Salomon™ oft ein wenig länger für die Analyse, da es hier eine große Vielfalt an Stellungen zu untersuchen gibt. Dafür wird es aber im Endspiel dann wieder schneller - bzw. falls man eine fixe Zeit eingestellt hat, wird die Suchtiefe größer. Die Werte in der Tabelle sind also **nicht** fixe Antwortzeiten, sondern nur als Orientierungshilfe gedacht.

Spielstufe	Antwortzeit
1	1 Sekunde
2	5 Sekunden
3	20 Sekunden
4	3 Minuten
5	9 Minuten

4. Anhang: Durchgespielte Partie

Die vorliegende Partie hat Salomon™ am 4. Oktober 1992 gegen den Chessmaster von Software Toolworks gespielt. Salomon™ war insofern im Nachteil, als daß er keine Eröffnungsbibliothek besitzt. Bedenkzeit hatten beide im Schnitt ca. 10 Sekunden (Spielstärke 3 bei Salomon).

Datum: 04-10-1992
Weiss : Salomon
Schwarz: Chessmaster 2100

1. d2-d4	Sg8-f6

Der Chessmaster versucht eine dameninidische Verteidigung aus seiner Bibliothek zu spielen, doch ...

2. Sb1-d2	d7-d5

... Salomon spielt unkonventionell (normalerweise folgt 2. c4 e6 3. Sc3 Lb4 [Nimzowitsch Indisch] usw.) und wirft den Chessmaster aus seiner Bibliothek! Der Grund warum Salomon™ Sd2 Sc3 vorzog (was ja eigentlich natürlicher erscheint) war der Bauernvorstoß c4, der ja auch ohne Deckung des Springers funktioniert hätte (Damengambit), aber ein Gambit spielt ein Programm normalerweise nicht freiwillig!

3. c2-c4	Sb8-d7
4. c4xd5!	Sf6xd5
5. e2-e4	...

Und das Zentrum gehört Salomon™! Die zwei Bauern sind recht stark.

5. ...	Sd5-f6
6. e4-e5	Sf6-d5
7. Lf1-d3	e7-e6
8. Sg1-h3	c7-c5

Sh3 sollte den Bauernvorstoß f4 ermöglichen, denn der Chessmaster greift bereits das Zentrum mit c5! an.

9. Dd1-g4	c5xd4
10. Dg4xd4	Lf8-c5
11. De4-d4	Dd8-c7
12. Sd2-f3	...

Angriff des Chessmasters auf den e5 Bauern, Salomon™ verteidigt sich hier gut. Der Springerzug deckt nicht nur den e5 Bauern, sondern macht auch gleichzeitig die Läuferlinie frei.

12. ...	Lc5-b4+!
13. Lc1-d2	Lb4xd2
14. Ke1xd2	...

Das kleine Zwischenschach des Chessmasters hatte es in sich, so schnell kann man eine Rochade vermiesen - denn Sf3xd2 verbietet sich wegen Sd7xe5! Jetzt wird sich zeigen, wie gut Salomon™ sich noch verteidigen kann, Schwarz steht nun natürlich besser, wenngleich auch etwas unterentwickelt.

14. ...	Sd7-c5
15. Ld3-b5+	Lc8-d7

Salomon™ versucht offensichtlich dasselbe wie sein Gegner, doch der Chessmaster hat ein wenig besser aufgepaßt ...

16. Lb5xd7	Dc7xd7
17. De4-d4	Ta8-c8
18. Th1-c1	Dd7-b5

Nicht ungefährlich! Vor allem daß die weiße Dame vor ihrem König steht, und noch dazu auf einer offenen Linie - es droht nun ständig Td8!

19. g2-g3	Db5-a5+
20. Kd2-d1	0-0
21. Kd1-e2	...

Nun versucht sich Salomon™ aber schleunigst von der offenen d-Linie zurückzuziehen, aber zu spät!

21. ...	Da5-b5+
22. Ke2-d2	Db5-b4+?
23. De4xb4	Sd5xb4

Ein Patzer des Chessmasters! Damit konnte Salomon™ den gewaltigen Druck der Dame loswerden, der Tausch kam also einzig und alleine Weiß zugute!

24. Tc1-c4	Tf8-d8+

Jetzt ist Td8+ nicht mehr so gefährlich wie zuvor ...

25. Kd2-e2	Sc5-d3
26. Tc4xc8	Td8xc8

27. a2-a3! ...

Salomon™ vertreibt das lästige Springerpaar.

27. ... Sb4-c2
28. Ta1-b1 Tc8-d8
29. Sh3-g5! ...

Eine Falle, auf die der Chessmaster auch prompt hereinfällt, verzweifelt versucht er nämlich seine Springer zu retten, doch dies ist nicht sehr gesund ...

29. ... Sc2-d4+?

Ein Fehler, den Salomon einfach großartig ausnutzt!

30. Sf3xd4 Td8xd4
31. Tb1-d1!! Td4-d5

Td1 war hervorragend, doch auf den ersten Blick erscheint Td5 die entsprechende Verteidigung zu sein, denn es droht Txe5+ mit Bauerngewinn, aber Salomon™ hat die Stellung tief eingeschätzt ...

32. Sg5xh7! Td5xe5+
33. Ke2xd3 Te5-d5+
34. Kd3-e2 Td5xd1??

Das hätte ich nicht getan, denn das war der Anfang vom Ende! Salomon™ zeigt nun im Großmeister Stil warum ...

35. Sh7-f6+!! ...

Brilliant! Ein Großmeister hätte nicht besser gespielt. Zumindest ich hätte im Reflex sofort den Turm auf d1 geschlagen, der Springer auf h7 erschien mir ohnehin verloren. Nicht nur, daß er dem Chessmaster einen Doppelbauern anhängt, Salomon schafft sich selbst auch noch einen Freibauern - und das ist tödlich...

35. ... g7xf6
36. Ke2xd1 e6-e5
37. h2-h4 f6-f5
38. b2-b4 f7-f6
39. Kd1-d2 e5-e4
40. Kd2-e3 Kg8-f7?

Kg7 wäre wohl besser gewesen ...

41. h4-h5! Kf7-g7

Sagte ich es nicht? Aber jetzt ist es schon vorbei, bei einer so miesen Stellung darf man sich keine Fehler mehr erlauben (nach 40. ... Kg7 41. Kf4 Kg6 hätte Schwarz noch länger Widerstand leisten können).

42. Ke3-f4 a7-a5

Der weiße Monarch schreitet zum letzten Gefecht, beim Chessmaster zeichnet sich bereits reine Verzweiflung ab!

43. b4xa5 Kg7-h6
44. Kf4xf5 Kh6xh5

Den Freibauern konnte er noch loswerden, aber was hilft das jetzt noch?

45. a3-a4 e4-e3
46. f2xe3 Kh5-h6
47. Kf5xf6 Kh6-h5
48. Kf6-f5 Kh5-h6
49. g3-g4 Kh6-g7
50. g4-g5 Kg7-h7
51. e3-e4 Kh7-g7
52. g5-g6 Kg7-g8

Die erdrückende Übermacht kommt, der Chessmaster ist chancenlos, zwei Bauern kann er mit seinem König nicht mehr stoppen ...

53. e4-e5 Kg8-g7
54. e5-e6 Kg7-f8
55. Kf5-e5!

Am einfachsten! Der Chessmaster gibt an dieser Stelle auch auf, den auf Kg7 folgt e7! (bzw. 55. ... Ke7 56. g7!) und eine Umwandlung ist nicht mehr zu verhindern.

5. Anhang: Verbesserungen ab Version 3.7

Salomon™ wurde von mir ständig weiterentwickelt, viele Fehler konnte ich korrigieren. Auch die Optimierung zeitkritischer Routinen (vor allem in der Suche) lag mir sehr am Herzen, und die Erfolge blieben letzten Endes nicht aus. So konnte ich z.B. mit der sogenannten Killerheuristik (Erklärung folgt!) die Anzahl der untersuchten Stellungen drastisch vermindern. Doch nun die Verbesserungen im Einzelnen:

+ **Der neue Menüpunkt Print Moves:**

Diese Erneuerung gestattet es, gespielte Partien auch auf einem Drucker auszugeben. Außerdem wird beim Speichern einer Partie auch eine Datei selben Namens wie die Partie mit der Endung ".MOV" angelegt, in der die Züge der Partie im normalen ASCII-Format gesichert werden. Was allerdings noch nicht funktioniert ist, daß man nach Laden des Spielstandes wieder alle Züge zur Verfügung hat, in so einem Falle beginnt man wieder beim ersten Zug[51]. Der Ausdruck sollte eigentlich problemlos auf allen Druckern durchführbar sein, die über eine IBM-Proprinter Druckemulation verfügen, besondere Graphikzeichen werden nicht verwendet. Falls das Drucken aber dennoch Probleme bereiten sollte, steht ihnen noch immer der Weg über "Save Game" zur Verfügung, beim Ausdruck der ASCII-Datei können sie dann ihre eigenen Druckertreiber verwenden.

+ **Die Zugzurücknahme:**

Was mich schon die längste Zeit gestört hatte war, daß man eine Partie stur weiterspielen mußte, hatte man sich einmal vertippt und gar die Dame dabei verloren. Es bestand zwar die Möglichkeit aufzugeben und den letzten Spielstand wieder per Hand einzutippen, aber das ist natürlich nur eine Notlösung gewesen, die noch dazu mit der Zeit gewaltig nervt. Also habe ich mich entschlossen -mit nicht unerheblichem Aufwand, wenn ich das einmal so am Rande bemerken darf...- eine Zugzurücknahme, sowie sie eigentlich in allen professionellen Schachprogrammen implementiert ist, einzubauen. Das Schwierigste war eigentlich zu eruieren, welche Informationen man benötigt, um einen Zug rückgängig zu machen bzw. wie man diese Infos möglichst kompakt speichert. Schließlich habe ich das folgende Konzept ausgetüftelt: Eine Zügeliste existierte ohnehin bereits (für das Ausdrucken der Züge notwendig!), aber diese Information war natürlich zu wenig, wenn man die vier Sonderfälle (Schlagzüge, Rochade, Bauernumwandlung und En Passant) betrachtete. Also mußte ich der Zugliste noch ein Byte an Info (für Weiß und Schwarz) und jeweils ein Bit für Weiß bzw. Schwarz hinzufügen. Das erste Byte

[51] Ab Version 3.10 ist das möglich, Salomon merkt sich nun auch, wer an der Reihe war

wird intern in ein HI und ein LO Byte geteilt, wobei das LO-Byte für Weiß ist und das HI-Byte für Schwarz. Es wird nun durch die verschiedenen Werte des Info-Bytes zwischen den folgenden Fällen unterschieden (Bsp. für Weiß):

- Info-Byte = 0:

D.h. der zuletzt gemachte Zug war ein Ruhezug, man muß lediglich Start und Ziel in der Zügeliste vertauschen und den Zug dann ausführen. Diese Aktion muß man auch bei den folgenden Fällen ausführen, zusätzlich muß aber noch etwas mehr geschehen.

- Info-Byte = 2..6:

Der letzte (weiße) Zug war ein Schlagzug, das Info-Byte zeigt an, welche schwarze Figur eliminiert wurde. Man muß also lediglich die Figur auf dem Zielfeld wiederherstellen.

- Info-Byte = 14:

Der letzte Zug war ein En Passant Zug, in diesem Fall muß das En Passant Feld richtig gesetzt werden und ein Bauer an der richtigen Stelle[52] eingesetzt werden.

- Info-Byte = 15:

Der letzte Zug war eine Rochade, man muß den entsprechenden Turmzug auch noch ausführen und die Rochaderechte wieder neu vergeben.

- Jeweils ein Info Bit für Weiß und Schwarz, ob eine Umwandlung stattfand:

Ist dieses Bit gesetzt, dann muß anstelle der vorhandenen Figur ein Bauer eingesetzt werden, war die Umwandlung zusätzlich noch ein Schlagzug, wurde bereits mit Hilfe des Info-Bytes die geschlagene Figur rekonstruiert.

Die Zugzurücknahme funktioniert übrigens mit CTRL-T, falls man an der Reihe ist. Einige Dinge gilt es allerdings zu beachten: Nimmt man nur einen Zug zurück (also den des Computers), so hat man automatisch einen **Seitenwechsel** vorgenommen (praktisch, nicht? So habe ich zwei Fliegen mit einer Klappe geschlagen!), spielt also nun die Seite von Salomon™. Nimmt man hingegen zwei Züge zurück ist wieder alles beim Alten, man spielt wieder dieselbe Seite wie zu Spielbeginn. Wird ein Zug zurückgenommen, während Salomon™ noch Züge aus seiner Bibliothek nimmt (das erkennt man an den besonders raschen Antwortzeiten!), so wird in der Folge **kein** Zug mehr aus der Bibliothek genommen[53].

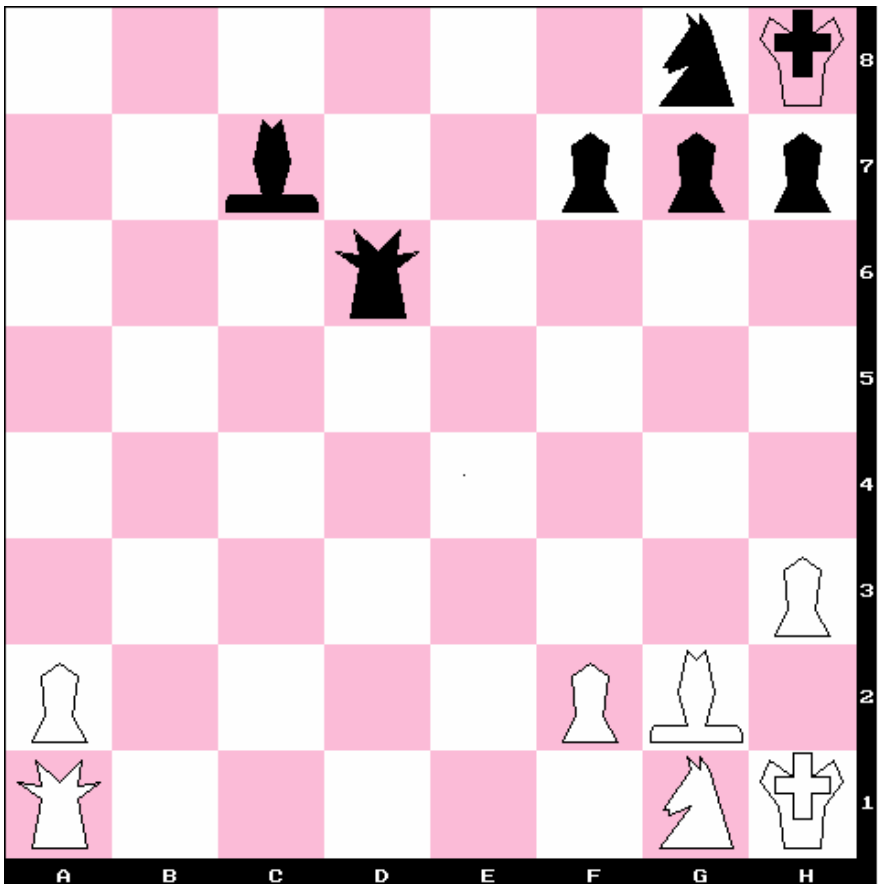
---

[52] Für Weiß wäre diese Stelle z.B. ein Feld vor dem alten Zielfeld des weißen Bauern

[53] Da die Bibliothek erst nachher hinzugefügt wurde, wäre das aus programmtechnischen Gründen nur mit erheblichem Mehraufwand an Code möglich. Da man aber in ernstzunehmenden Tunieren ohnehin keine Züge zurücknehmen darf, steht dies eigentlich nicht zur Debatte

+ **Killerheuristik:**

Betrachten sie hierzu doch einmal folgende Stellung:



Weiß muß dringend etwas gegen Dh2++ unternehmen (z.B. Sf3). Sobald nun die Suche das Matt entdeckt, wäre es doch schön wenn z.B. auf den generierten Zug a3 sofort der Zug Dh2++ betrachtet werden würde, a3 ist dann sofort widerlegt, die Suche kann sofort einen Cut-Off machen. Um dies zu erreichen braucht man nur ein zusätzliches Feld (Killer_Zuege) einführen, in dem man sich die besten Züge einer Stellung merkt (pro Tiefe). Derzeit werden fünf Züge pro Tiefe gespeichert, man kann diesen Wert allerdings beliebig ändern indem man die Variable MaxKiller-Zuege in CHESS.PAS auf einen anderen Wert setzt. Speichert man allerdings zu viele Züge, so wird der Verwaltungsaufwand beträchtlich, und der Wert der Heuristik zweifelhaft. Vielleicht werden sogar weniger Stellungen untersucht, aber die Zeit erhöht sich auch drastisch! Zu hohe Werte sollten also tunlichst unterlassen werden. Ach ja, falls die Liste für eine Tiefe voll ist (das steht im Feld Killer_Index[Tiefe]), dann sucht eine Routine den schlechtesten Zug aus der Liste heraus und ersetzt ihn durch den aktuellen Wert, falls dieser besser ist. Man sieht schon: ist die Liste zu lange, bedeutet das eine längere Suchzeit, noch dazu ist ja nicht gesagt, daß in der aktuellen Stellung auch jener Zug der Beste ist, er könnte im schlimmsten Falle auch der schlechteste sein, dann gibt es überhaupt keinen Cut-Off! Die Methode ist eben nur eine Heuristik und kein zuverlässiges Kriterium,

aber im Schnitt treten solche Fälle dann doch recht selten auf, sodaß die Geschwindigkeit von Salomon™ doch erheblich gesteigert wurde.

+ **Die Eröffnungsbibliothek:**

Jedes professionelle Programm verfügt heute über eine Eröffnungsbibliothek, und so habe ich mich auch bei Salomon™ entschlossen, so ein Modul (SALBOOK.PAS) mit sehr, sehr viel Arbeit zu implementieren. Denn gerade in der Eröffnung wurde oft sehr viel Rechenzeit investiert, die später dann (wenn dringend benötigt) fehlte. Außerdem fällt das Programm nun nicht mehr den zahlreichen Gambit-Varianten zum Opfer indem es verzweifelt versucht irgendeinen Bauern zu decken und dabei natürlich seine Position katastrophal verschlechtert. Die Varianten selbst liegen im ASCII-Format (File SALMON.BOK) vor und können beliebig ergänzt werden, wenn man die folgenden Regeln beachtet:

- Kommentare beginnen mit einem "!" und können beliebig oft eingesetzt werden, allerdings **nicht** in einer Zeile wo ein Zug oder ein Schlüsselwort steht!

- Eine neue Variante beginnt immer mit "Name", worauf nach einem Leerzeichen maximal zwanzig Buchstaben folgen dürfen, die die entsprechende Eröffnung benennen.

- Leerzeilen sind nicht erlaubt.

- Eine Variante muß **korrekte** Züge enthalten, Salomon™ prüft **nicht** die Gültigkeit der Züge[54]!

- Es dürfen keine zwei Varianten existieren, wovon die eine die andere **vollständig** enthält. Tritt dennoch so ein Fall auf, reagiert Salomon™ mit einer entsprechenden Fehlermeldung (Name der Eröffnung) und bricht ab. Sie müssen nun auf Grund des Namens die Eröffnung ausfindig machen und eliminieren, sie ist ohnehin zweimal gespeichert.

- Am Ende der Bibliothek steht das Schlüsselwort "End"

Beachten sie diese paar Regeln, so können sie die Bibliothek nach Belieben erweitern. Weiters wurde im Menü "Advanced Features" ein neuer Menüpunkt "Opening library" hinzugefügt, mit dem sie einerseits sehen, wieviele Varianten Salomon™ geladen hat, andererseits können sie je nach Belieben entweder die Bibliothek gänzlich oder auch nur die Random-Suche abschalten. Die Anzahl der Varianten die geladen werden hängt davon ab, wieviel Hauptspeicher noch zur Verfügung steht. Salomon™ erkennt selbständig, wann es eng wird und lädt dann entsprechend

---

[54] Das kann man glaube ich wirklich voraussetzen, wir wollen ja schließlich Rechenzeit gewinnen und nicht durch aufwendige Gültigkeitstests schon wieder verschwenden

weniger Varianten (es müssen nämlich auch noch der BGI Treiber und der Graphik Zeichensatz im Hauptspeicher Platz finden, bei 640k Arbeitsspeicher aber können sie getrost sehr lange tippen...). Die Bibliothek ist prinzipiell so konzipiert (und das war bei Gott nicht einfach!), daß Salomon™ vollkommen eigenständig einen Variantenwechsel erkennt (z.B. von Ruy Lopez Morphy Verteidigung zur Abtauschvariante) und auch darauf reagiert (ersteres ist selbstverständlich trivial, zweiteres wirft schon einige Probleme auf). Aber dem nicht genug, Salomon™ sucht sich nicht nur (falls natürlich vorhanden) eine neue Variante, er wählt zusätzlich auch noch **zufällig** zwischen mehreren Varianten aus, falls es an der entsprechenden Stelle mehr als eine Fortsetzung gibt. Falls man allerdings die Random-Suche abschaltet, nimmt er die erste gefundene. Deswegen sollte man die am wenigsten riskante Fortsetzung immer an den Anfang reihen, denn falls die Random-Suche abgeschaltet wird, spielt Salomon™ immer diese.
Salomon™ zeigt zusätzlich noch den Namen der jeweiligen Eröffnung für ca. drei Sekunden an, selbstverständlich auch dann, falls die Variante gewechselt wurde. Wird kein Fortsetzungszug mehr in der Bibliothek gefunden, wird das Spiel wie üblich weitergespielt.
Die Speicherung der einzelnen Eröffnungsvarianten erfolgt in einem dynamischen zweidimensionalen Array, in dessen eine Dimension die verschiedenen Eröffnungen gespeichert werden und in die andere Dimension die Züge die dazugehören. Zusätzlich wird das ASCII-Format in das interne Format übersetzt, sodaß nur vier Byte an Speicher für einen Zug gebraucht werden.

+ **Verbesserung der Königssicherheit:**

Ab Version 3.10 wurde die Beurteilung der Königssicherheit noch ein wenig verfeinert. So werden zum einen nicht nur vorgerückte Bauern als negativ gewertet, sondern zusätzlich wird auch noch betrachtet, ob die jeweilige Linie halboffen ist. Trifft letzteres auch noch zu, wird ein höherer Punkteabzug vorgenommen, da der König ja wesentlich unsicherer steht (Bedrohung durch Doppeltürme etc.).
Äquivalent dazu wird bei fehlenden Flügelbauern geprüft, ob die jeweilige Linie offen ist - das ist dann natürlich ganz schlecht, der König steht ganz besonders unsicher. Der Aufwand ist gering, entsprechende Felder, in denen Informationen über offene bzw. halboffene Linien sind, werden ohnehin von Bauernbewertung und Turmbewertung benutzt, ich habe nur vorhandenes Wissen noch ein wenig besser ausgenutzt.
Was allerdings sehr wohl Rechenzeit verschlingt ist die Prüfung, ob die Felder rund um den König von gegnerischen Figuren kontrolliert werden. Dies leistet die Routine Check_King_Savety in HILFSPRG.PAS. Sie dividiert ganzzahlig die Werte der jeweiligen Figuren durch 100 und liefert dann die Summe zurück - je höher selbige ist, desto angegriffener scheint der Flügel zu sein. Man muß den erhaltenen Wert nun nur noch so wichten, daß z.B. der Angriff einer Dame nicht sofort dazu verleitet, den Flügelbauern vorzuziehen (dann wäre ja zumindest die Kontrolle um den König weg!), denn dies ist ja meist mit einer eklatanten Schwächung des

Königsflügels verbunden. Selbstverständlich findet diese (aufwendige) Prüfung im Endspiel nicht mehr statt.