- Thema -

Die Entstehung eines Schachprogramms und dessen Grundlagen

Geschrieben von: Enrico Gebert
Daniel Schumann
Manuel Maier

Stammkurs: BGI02a

Stammkursleiterin: Ina Boer

Betreuer: Richard Brömel

Inhaltsverzeichnis

Vorwort

In unserer Seminarfacharbeit haben wir uns mit dem Thema der Entwicklung eines Schachprogramms und seiner Grundlagen auseinander gesetzt. In der Freizeit sind wir aktive Schachspieler, weiterhin ist das Programmieren eine Leidenschaft von uns. Zudem sehen wir in der Umsetzung des Spiels in ein lauffähiges Programm eine große Herausforderung an unsere Fähigkeiten. Auch wollen wir dem Leser mit dieser Arbeit die Faszination am „Spiel der Könige“ näher bringen.
Um einen Ausgangspunkt für die Entwicklung des Programms zu schaffen, haben wir uns zunächst eingehend mit der Geschichte und den Regeln des Schachs vertraut gemacht. Außerdem beschäftigten wir uns noch umfassend mit der Strategie und der Taktik des Spieles. Daraus entwickelten wir die verschiedenen Algorithmen, die die Grundlage des Programms bilden.

Um den Textfluss der Kapitel Regeln sowie Spielverlauf nicht zu stören, haben wir Handreichungen beigelegt. Auf diesen sind verschiedenen Diagrammen zur Erläuterung abgebildet.
Zusätzlich ist unserer Arbeit eine CD beigefügt, auf der sich das Schachprogramm und weiter Information für interessierte Leser befinden.

I. Geschichte

1. Legenden [i]

Die Entstehung des Schachs war ein jahrhunderte langer Prozess. Ein namentlich festgehaltener Erfinder dieser Sportart ist unbekannt. Es gibt viele verschiedene Legenden, in denen beschrieben wird, wie das Schachspiel entstanden sei. Die wichtigsten Legenden wurden erstmals von Herzog August, dem II. von Braunschweig und Lüneburg in deutscher Sprache gedruckt.

Eine dieser Legende besagt, dass das Schachspiel aus dem ägyptischen Pharaonenreich stammt. Jedoch kann dies nicht bewiesen werden, da nur bekannt ist, dass die Ägypter einige verschiedene Brettspiele kannten. In einer anderen Sage wird die Entstehung des Schachs den Griechen zugeschrieben. Die griechischen Truppen sollen zurzeit der Belagerung Trojas ihre „Langweile“ mit Schachspielen vertrieben haben. Ein Beweis dafür ist in Rom zu finden. Dort kann man eine Amphora[1] des Exekias aus der Zeit der Belagerung um das 5. Jahrhundert betrachten. Auf dieser Tonvase sind zwei Griechen zu sehen, welche mit einem Brettspiel beschäftigt sind. Es muss sich aber nicht um Schach handeln, es kann auch ein anderes Spiel gewesen sein. Aus diesem Grund, ist diese Legende nicht eindeutig zu beweisen. In spätmittelalterlichen Spanien entstand eine andere Legende, die auf den Perserkönig Xerxes zurückführt. Da wurde das Brettspiel Axedres genannt. Es gibt auch Verbindungen der persischen Mittelrolle und der Zeit von Gustavus Selenus.

Eine andere Version zur Entstehung des Schachs besagt, dass ein scheußlicher Tyrann, König Evilmerolach, durch das Schachspiel zur Güte bewegt worden ist. In einer weiteren Legende heißt es, dass das Schachbrett eine Nachbildung der viereckigen Stadt Babylon sein soll.

In jedem europäischen Land hat es den Geschichtsverständnissen angepasste Legenden gegeben.

Aus persischen und arabischen Quellen wird sichtbar, dass Indien das Ursprungsland ist.

Van der Linde, ein Geschichtsforscher, erarbeitete eine Quellenkritik zur Entstehung des Schachs. Er stützte sich nur auf schriftliche Überlieferungen und schrieb es in seinem Buch „Geschichte und Literatur des Schachspiels“ nieder.

---

[1] Ist ein großes Tongefäß mit zwei Henkeln.

Jedoch kann man sich nicht nur auf materiell greifbare Schriften beziehen, das heißt sich nicht nur wie van der Linde zu spezialisieren. Man muss jeder Legende nachgehen, auch wenn sie nur mündlich überliefert wurde.

Die persische – arabische Erzählung besagt, dass das Schach in Indien entstanden ist. Die ersten Aufzeichnungen machte Masudi. In seinen Schriften konnte man auch die Weizenkornlegende finden. Sie erzählt von dem weisen Erfinder des Schachspiels, der von seinem König nur Weizenkörner als Lohn verlangt hatte. Dies wurde jedoch besonders berechnet. Es sollte für das erste Feld des Schachbrettes ein Weizenkorn berechnet werden, für das zweite zwei, für das dritte vier, für das vierte sechzehn und so weiter. Da das Brett 64 Felder hatte, kam man am Ende fälschlicherweise auf eine Zahl vom 18˙446˙744˙073˙709˙551˙616 Weizenkörner. Heute weis man, dass es ein Korn weniger seinen muss, da sich auf dem ersten Feld nur eines befindet. So viele Körner hätte der König dem Erfinder zahlen müssen. Mit dieser Anzahl an Weizen konnte man ganz Europa und Teile Afrikas bedecken. Durch diese Geschichte wurde Masudi zum Märchenerzähler ernannt. Er war ein sehr intelligenter Mensch, er wusste viel über Indien zu berichten. Er benennt Balhit als den Erfinder des Schachs. Es soll zur Wende des dritten zum zweiten Jahrhunderts vor unserer Zeit entstanden sein.

2. Entstehung in Indien [ii]

In indischen Büchern schrieb man im 7. Jahrhundert zum ersten Mal über ein Schachspiel. In der Zeit von 606 bis 647/48, als König Shir Harsha an der Macht war, hatte sich das Schach schon in Indien verbreitet. Es wurde als ein Gleichnis für Friedfertigkeiten benutzt. Jedoch ist Schach schon viel älter, wie wir jetzt wissen.

Die Entstehung des Schachs wurde der Zeit Buddhas zugeschrieben. Es gab das Schach wie wir es heute kennen noch nicht, sondern es wurde mit Würfeln gespielt. Das ist nicht der einzige Unterschied zum heutigen Spiel, Schach wurde auch zu viert gespielt. Nun könne man annehmen, dass Schach ein Glücksspiel war. Man hat den Würfel so geworfen, dass man die gewünschte Augenzahl bekam. Fiel eine unerwünschte Zahl, hat man den Würfel einfach aufgefangen. Man musste blitzschnell entscheiden, ob die gewünschte Zahl fällt oder ob man erneut werfen musste. Der Würfel hatte nur vier Zahlen. Die Eins war für den König oder für den Bauern. Würfelte man eine Zwei, so konnte der Elefant[2] bewegt werden, bei einer Drei das Pferd und bei einer Vier der Turm. Anders als im heutigen Regelwerk konnte der

[2] Ist heute der Läufer.

Turm ein Feld in der waagerechten oder senkrechten überspringen und der Elefant konnte nur in das übernächste diagonale Feld springen. Das Ziel war es, wie beim heutigen Schach auch, die gegnerischen Könige zu werfen.

Die Anzahl der Felder eines Schachspiels ist nicht zufällig festgelegt wurden. Die 64 Schachfelder kam zu Stande, in dem man die Acht mit sich selbst multipliziert. Die Acht ist in Indien eine heilige Zahl, genau wie die Vier. Daher ist es kein Wunder, das es beim Vierschach (Ur – Schach), für jeden der vier Spieler, acht Figuren gab. Von der Brettmitte konnte der König und der Läufer acht Felder erreichen, der Elefant konnte auch genau auf acht gelangen und der Wagen konnte 64 umfassen oder als springender Turm 16 Felder erreichen. Hier wird ersichtlich, dass das indische Schach nach den heiligen Zahlen aufgebaut ist.

Beim Vierschach haben vier Könige gespielt, davon haben immer die zwei Gegenüberliegenden zusammen gespielt. Beide durften über das Heer, also über die Figuren des Anderen mitbestimmen. Man hatte auf einmal zwei Spieler, die die Figuren ziehen konnten. Es gab auch zwei Könige. Daraus ist höchst wahrscheinlich das Zweischach entstanden, so wie wir es heute kennen und lieben. Ein König wurde erst zum Wesir[3] und später dann zur Dame.

3. Der Weg nach Europa [iii]

Das Schach musste aus Indien eine große Distanz überwinden, damit es in Europa gespielt werden konnte. Daran hatten die Perser einen sehr großen Verdienst. Persien war seit je her das Verbindungsstück zwischen Asien und Europa. Die Perser haben die Schachbegriffe in ihre Sprache übersetzt. In Indien bezeichnete man beispielsweise den König mit „rāja“ und in Persien sagte man „šhāh“. Zum anderen wurden bei den Persern eine Bezeichnungen „ruhk“ gewählt, dies kann man heute aber immer noch nicht genau übersetzen. Zuerst wurde das Schachspiel im Hauptland der Perser gespielt.

Der Philologe Georg Bossong hat eindeutig den Weg des Schachspiels beschrieben. Es kam von Indien und wurde durch die Perser in den arabischen Ländern verbreitet. Schließlich kam das Schach von dort nach Europa. Die Europäer konnten sich nicht viel unter den Bezeichnungen der Perser vorstellen und bezeichneten es einfach nach dem Ausspruch „Schach“. Die Schachspieler, aus Persien, sagten „Schach“, wenn sie

[3] War eine Art Minister.

den gegnerischen König bedrohten. Aus dieser Zeit muss auch der Begriff „Matt“ kommen, was soviel heißt wie hilflos oder Hilflosigkeit.

Schach ist ungefähr zur selben Zeit nach Europa gekommen, wie auch der Islam. Dort begegneten sich beide Religionen (Christentum und Islam) das erste Mal. Hierzu gibt es wieder einige Legenden. Eine wäre zum Beispiel, dass Karl der Große schon Schach gespielt haben soll. Das war ca. im Jahre 800. Karl soll ein Schachbrett vom Kalifen Raschid geschenkt bekommen haben. Die 14 Figuren sind heute noch in Osnabrück zu sehen. Es könnte sich natürlich auch um die Spielfiguren eines anderen Schachspielers handeln, da nichts eindeutig bewiesen ist.

Wie genau oder wann kam das Schachspiel nach Europa? Genaue Angaben gibt es leider nicht.

Die Geschichtsforscher gehen davon aus, das Schach über Gibraltar nach Europa kam. Über die Pyrenäen gelangte es nach West- und Mitteleuropa. Seit dem 8. Jahrhundert trafen nämlich in Nordspanien und Südfrankreich die Franken und die Araber aufeinander. Aus diesem Grund ist die Wahrscheinlichkeit sehr groß, dass auf diesem Weg das Schachspiel nach Europa kam. Ein Beweis dafür sind die Spielfiguren, die man an den Stellen des Krieges gefunden hat. Es waren Figuren aus Bergkristallen. Diese sind noch vollständig in verschiedenen Museen zu betrachten.

Es ist aber auch belegt, dass vor dem Jahre 900 in Italien schon Schach gespielt wurde. Durch den Handel, den Moslems und Christen mit einander trieben, ist es möglich gewesen, dass das Schach auch auf diesem Wege nach Europa gelangte. Man fand die gleichen Bergkristallfiguren wie in Spanien als Grabbeilage. Nach dieser Legende müssten die Spielfiguren um 800 in Italien gewesen sein.

Nach Deutschland gelangte das Spiel voraussichtlich über Italien. In den Jahren 1030 bis 1050 ist das erst Mal die Rede vom Schach in Deutschland.

Das Schach kann natürlich auch auf anderem Wege nach Europa gekommen sein. Eine Möglichkeit wäre der Seeweg, speziell hätten es die Wikinger sein können, die den Europäern dieses Spiel lieferten.

4. Die Anfänge der Schachcomputer

Die Mathematiker und Wissenschaftler hatten schon immer ein sehr großes Interesse am Schachspielen. Eine interessante Aussage machte Gottfried Wilhelm Leibnitz, als man ihn nach dem Nutzen des Schachs fragte. Er sagte, dass man durch dieses Spiel seine Denkfähigkeit und Erfindungsgabe üben kann. Der Mathematiker Leonard Euler fand die Vorschrift, wie der Rösselsprung berechnet werden kann.

Freiherr von Racknitz Wolfgang von Kempelen entwickelte 1769 seinen so genannten Schachautomaten. Dieser bestand aus einer Puppe, welche die Figuren bewegte. Der zweite wichtige Bestandteil war ein Schrank mit einem Schachbrett. Diese Puppe war wie ein Türke angezogen und bestand wirklich nur aus Walzen und Hebeln. Deshalb war es unerklärlich das eine Maschine so gut spielen konnte. Der Türke besiegte sogar Napoleon im Jahre 1809, der dieses Spiel sehr gut beherrschte. Erst lange nach dem Tod von Kempelen wurde das Geheimnis des Schachautomaten entdeckt. Der Schrank hatte zwei Kammern, wovon man eine durch Spiegel verdeckte. In dieser hielt sich eine kleine Person auf. Sie übernahm das Schachspielen für den Türken [iv].

Alan Turing entwickelte 1947 den ersten Schachalgorithmus. Das Programm musste zu dieser Zeit noch von einem Menschen abgearbeitet werden, da es noch kein Computer gab. Die Bewertung der Züge musst dabei auch per Hand vorgenommen werden. Der Algorithmus hieß „Turochamp“. Da dieser noch nicht ausgereift war, konnte man noch die typischen Unarten erkennen [v].

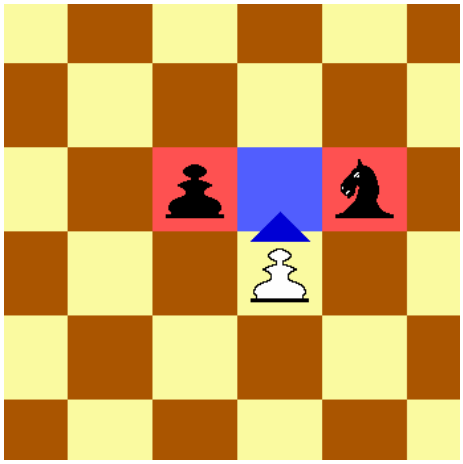
Alle weiteren erfundenen Schachcomputer konnten immer besser spielen. Nun trugen nicht nur Menschen Schachweltmeisterschaften aus sondern auch Computern. Diesen gelang sogar einige Male ein Sieg über berühmte Schachspieler. Ein britischer Internationaler Meister, David Lavy, verlor beispielsweise gegen das Schachprogramm „Chess 4,7“ [vi].

Der wohl bedeutendste Computer ist „Deep Blue“. Er erzielte 1996 einen Sieg und zwei Remis gegen den damals amtierenden Schachweltmeister, Garry Kasparow. Dieser gewann jedoch das Match gegen das Programm [vii].

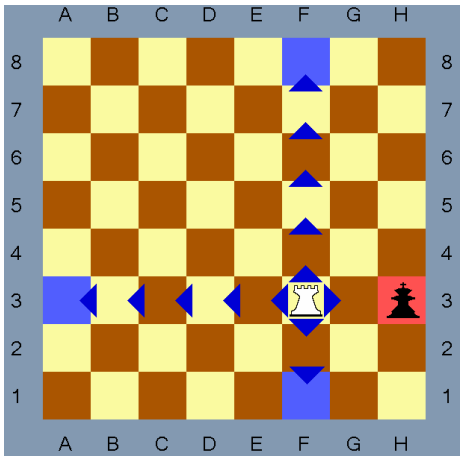
II. Regeln

1. Zug- und Schlagmöglichkeiten der Figuren [viii]

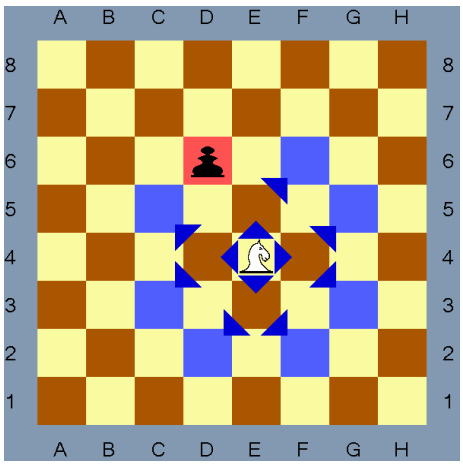
Die **Bauern** können immer nur ein Feld in die Richtung des Gegners bewegt werden. Der erste Zug des Bauern kann auch ein Doppelzug sein, bei dem er sich zwei Felder in Zugrichtung bewegen kann. Ein Bauer kann eine gegnerische Figur diagonal in seiner Laufrichtung schlagen.



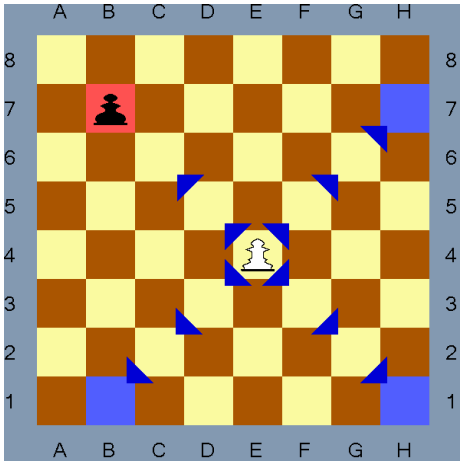
Der **Turm** kann immer nur waagerecht oder senkrecht gezogen werden, dabei darf er jedoch keine anderen Figuren überspringen. Der Turm darf die gegnerischen Figuren nur in seinen Zugrichtungen schlagen.



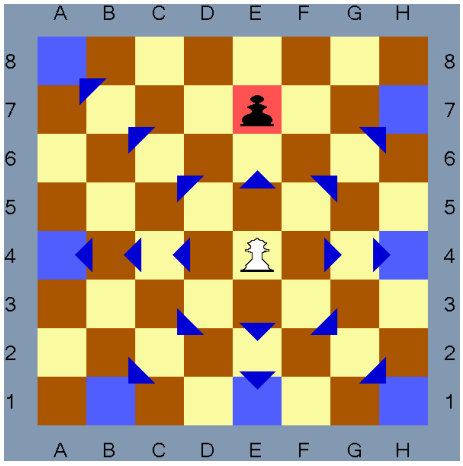
Der **Springer** zieht nach dem Rösselsprung, er bewegt sich dabei zwei Felder in eine Richtung (horizontal oder vertikal) und ein Feld in die dazu senkrechte Richtung. Der Springer kann im Gegensatz zu allen anderen Figuren über Figuren springen. Er schlägt durch den Rösselsprung, er nimmt dabei das Feld der geschlagenen Figur ein.



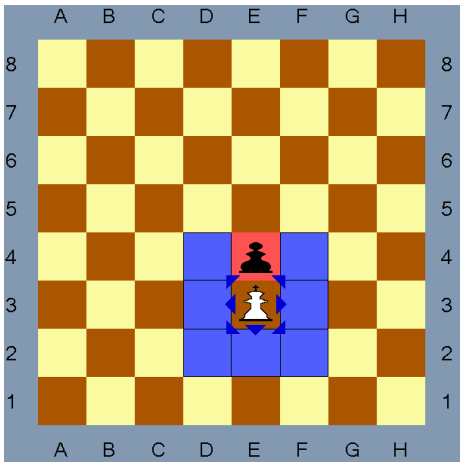
Der **Läufer** kann immer nur diagonal gezogen werden. Dabei bleibt er immer auf derselben Felderfarbe.



Die **Dame** ist in der Lage sowohl wie Turm als auch Läufer zu ziehen, das heißt sie kann immer waagerecht, senkrecht und diagonal gezogen werden. Die Dame kann die gegnerischen Figuren wie Turm und Läufer schlagen.

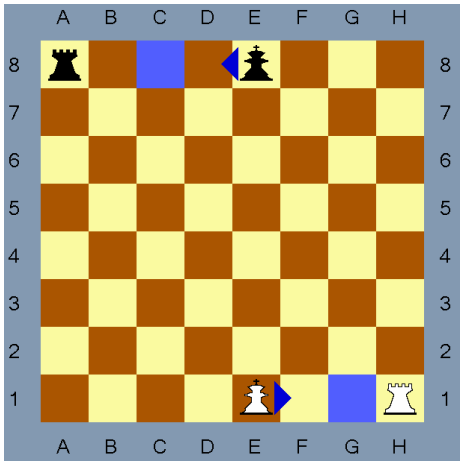


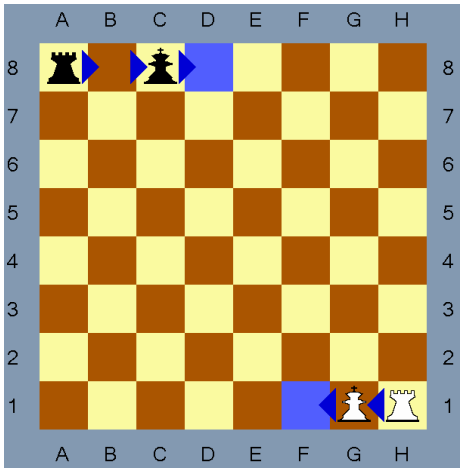
Der **König** kann immer nur ein Feld in jede Richtung ziehen. Jedoch bei der Rochade ist es ihm gestattet zwei Felder zu ziehen. Der König kann die unmittelbar benachbarten, gegnerischen Figuren schlagen, sofern er danach nicht bedroht wird, das heißt er darf nach seinem Zug nicht im Schach stehen.



2. Komplizierte Regeln [ix]

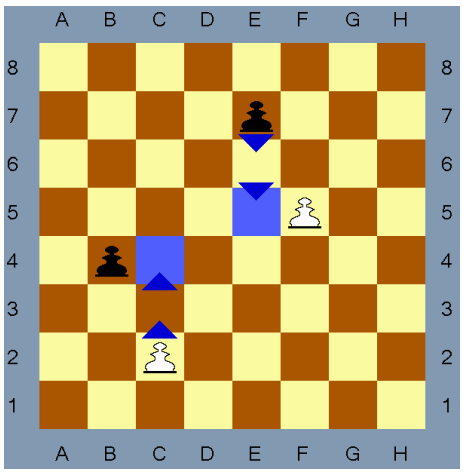
a) Rochade





Die Rochade ist ein Sonderzug von König und Turm, welche in einem Spiel nur einmal durchgeführt werden kann. Sie ist nur möglich wenn sich zwischen Turm und König keine Figur mehr befindet. Die Rochade ist jedoch nicht gestattet, wenn König oder Turm sich schon bewegt haben. Der König darf weder vor noch nach der Rochade bedroht sein, also im Schach stehen. Außerdem darf er kein Feld passieren, welches von einer gegnerischen Figur bedroht wird. Der König muss zwei Felder zu seinem Turm gezogen werden, damit der Turm „darüber springen“ kann. Er landet im benachbarten Feld des Königs. Es wird in lange und kurze Rochade unterschieden, man bezieht sich dabei auf den abstand zwischen Turm und König.

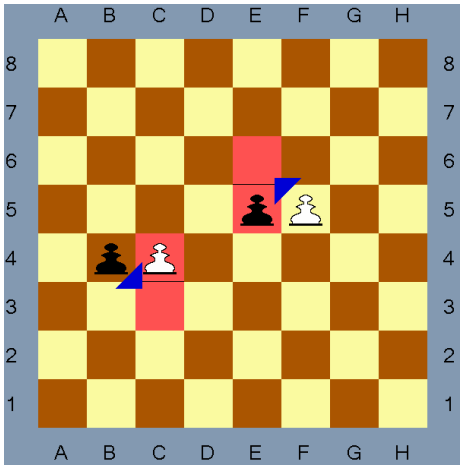
b) En passant[4]



En passant ist ein weiterer Sonderzug, er kann nur von den Bauern ausgeführt werden. Die Regel En passant verhindert, dass man aus einem Doppelzug des Bauern einen Vorteil erhält. Steht der am Zug befindliche Bauer nach einem Doppelzug neben dem gegnerischen Bauern, würde er sich vor dem Schlagen des gegnerischen Bauers schützen. Damit

[4] (frz. im Vorübergehen)

dieser Vorteil nicht zustande kommt, hat man die Regel En passant eingeführt. Sie besagt, wenn ein schwarzer Bauer nach einen Doppelzug neben einem weißen Bauern steht, so kann der Weiße ihn trotzdem schlagen. Dies geschieht indem er hinter den schwarzen Bauern zieh (schräg) und ihn vom Brett nimmt. Natürlich gilt diese Regel analog auch für die schwarzen Bauern.



c) Umwandlung

Der Bauer ist die einzige Figur, die im Rang aufsteigen kann, das heißt er kann in eine andere Figur, zum Beispiel in eine Dame, umgewandelt werden. Diese Umwandlung erfolgt, sobald ein Bauer die letzte Reihe der gegnerischen Seite des Brettes erreicht hat.

3. Weitere Begriffe [x]

Droht eine gegnerische Figur den König zu schlagen, so bezeichnet man dies als *Schach*. Da der König nicht vom Brett genommen werden darf, muss sich dieser entweder auf ein sicheres Feld bewegen oder es muss eine andere Figur so gestellt werden, dass der König nicht mehr bedroht wird.

Kann der König dem Schach nicht entfliehen, so ist er *Schachmatt*. Somit hat der Spieler, der den König *matt* gesetzt hat, dieses Spiel gewonnen.

Außer *Schachmatt* kann eine Partie auch *Remis* enden, das heißt die Partie ist unentschieden. Ein *Remis* kann durch ungenügendes Material, 3-malige Stellungswiederholung, 50-Züge-Regel, Patt oder durch Einigung beider Spieler zu Stande kommen.

Ungenügendes Material

Ist es für beide Spielpartner mit den übrigen Figuren nicht mehr möglich den gegnerischen König matt zusetzten, so endet diese Partie Remis. Beispiele sind König gegen König, König und Springer gegen König oder König und Läufer gegen König

3-malige Stellungswiederholung

Wiederholt sich eine Stellung dreimal mit dem gleichen Spieler am Zug, so kann der am Zug befindliche Spieler vom Gegner ein Remis fordern. Jedoch müssen sich die Stellungen nicht unbedingt direkt nacheinander wiederholen.

50-Züge-Regel

Wird innerhalb von 50 Zügen kein Bauer bewegt und keine Figur geschlagen, so geht diese Partie unentschieden aus.

Patt

Kann ein Spieler keinen legalen Zug mehr tätigen und befindet sich sein König nicht im Schach, so ist der Spieler patt. Das Spiel ist dadurch Remis.

Einigung beider Spieler

Eine Partie endet ebenfalls Remis, wenn sich beide Parteien darauf einigen.

4. Schachnotation [xi]

Wir wollen hier einen kurzen Überblick über die Notation im Schach geben, damit die Zugfolgen im Nachfolgenden Teil der Arbeit nachvollziehbar werden.

Das Schachbrett wird in acht mal acht Felder unterteilt. Die Buchstaben **a** bis **h** bezeichnen die Spalten und die Zahlen **1** bis **8** die Zeilen des Brettes. Für die Angabe der zu ziehenden Figur, wird dem Zug ihr Anfangsbuchstabe vorangestellt. Beim Bauern wird dieser jedoch weggelassen.

Um nun den Zug zu notieren, wird nur das Zielfeld der Figur angegeben, zum Beispiel e4 bedeutet, dass der Königsbauer auf e4 gezogen wird. Sf6 bedeutet, dass der Springer auf das Feld f6 zieht. Analog gilt dies für jede andere Figur auf dem Spielfeld. Im Falle, dass zwei Figuren derselben Art ein Zielfeld erreichen können, wird zusätzlich die Zeile oder Spalte des Ausgangsfeldes der zu bewegenden Figur mit angegeben. Beispielsweise Tab8 bedeutet, dass der Turm von a8 auf das Feld b8 gezogen wird.
Schlagzüge werden durch ein ’x’ vor dem Zugfeld angegeben. Ein Beispiel wäre exd5, was bedeutet, dass der e – Bauer die Figur auf dem Feld d5 schlägt.
Wenn durch einen Zug ein Schachgebot entsteht, wird dem Zug ein ’+’ angefügt. Bei Schachmatt ist dies ein ’++’.

III. Spielverlauf

1. Eröffnung

Es gibt viele verschiedene Möglichkeiten für Weiß ein Schachspiel zu beginnen. Man unterscheidet die Eröffnungen in Offene, Halboffene und Geschlossene Spiele. Alle Eröffnungsarten sollten die gleichen Ziele verfolgen. Man sollte seine Spielsteine so entwickeln, dass durch sie das Zentrum des Brettes bedroht wird. „Schwere Figuren“, wie Turm oder Dame brauchen offene Linien um ihr Kräfte vollständig zu entwickeln. Es sollten lieber Bauern oder leichte Figuren das Spiel eröffnen. Man muss seine Spielsteine so setzen, dass sie sich nicht gegenseitig behindern. Es ist immer gut, wenn sich Figuren einander decken.
Einige Eröffnungen die auch unser Schachprogramm beinhalte sind:

a) Offene Spiele [xii]

Alle offenen Spiele werden mit 1. e4 e5 eröffnet. Die weiteren Züge sind von Eröffnung zu Eröffnung verschieden.

Schottische Eröffnung (Diagramm 11)

1. e4 e5	2. Sf3 Sc6	3. d4 exd4	4. Sxd4 Sf6	5. Sc3 Lb4	6. Sxc6 bxc6

Italienische Eröffnung (Diagramm 12/13/14)

1. e4 e5	2. Sf3 Sc6	3. Lc4 Lc5	4. c3 Sf6	5. d4 exd4	6. cxd4 Lb6
			4. d3 d6	5. Sc3 Sf6	6. 0-0 0-0
			4. b4 Lxb4	5. c3 La5	6. 0-0 d6

Zweispringerspiel (Diagramm 15)

1. e4 e5	2. Sf3 Sc6	3. Lc4 Sf6	4. Sg5 d5	5. exd5 Sa5	6. Lb5+ c6

Russische Verteidigung (Diagramm 16)

1. e4 e5	2. Sf3 Sf6	3. Sxe5 d6	4. Sf3 Sxe4	5. De2 De7	6. d3 Sf6

Vierspringerspiel (Diagramm 17)

1. e4 e5	2. Sf3 Sf6	3. Sc3 Sc6	4. Lb5 Lb4	5. 0-0 0-0	6. d3 d6

Spanischer Eröffnung (Diagramm 18/19/20)

1. e4 e5	2. Sf3 Sc6	3. Lb5 a6	4. La4 Sf6	5. 0-0 Sxe4	6. d4 b5
				5. 0-0 Le7	6. Te1 d6
				5. 0-0 Le7	6. Te1 b5

Königsgambit (Diagramm 21/22/23)

1. e4 e5 2. f4 exf4 3. Sf3 Sf6
3. Sf3 g5
3. Sf3 Le7

b) Halboffene Spiele [xiii]

Bei halboffenen Spielen antwortet Schwarz auf e4 nicht mit e5, sondern mit einem anderen beliebigen Zug.

Französische Verteidigung (Diagramm 24/25)

1. e4 e6 2. d4 d5 3. Sc3 Sf6 4. Lg5 Le7 5. e5 Sfd7 6. Lxe7 Dxe7
3. Sc3 Lb4 4. e5 c5 5. a3 Lxc3+ 6. bxc3 Se7

Caro – Kann – Verteidigung (Diagramm 26/27/28)

1. e4 c6 2. d4 d5 3. exd5 cxd5 4. Ld3 Sc6 5. c3 Sf6 6. Lf4 Lg4
3. e5 Lf5 4. Ld3 Lxd 5. Dxd3 e6 6. f4 c5
3. Sc3 dxe4 4. Sxe4 Lf5 5. Sg3 Lg6 6. Sf3 Sd7

Skandinavische Verteidigung (Diagramm 29)

1. e4 d5 2. exd5 Dxd5 3. Sc3 Da5 4. d4 Sf6 5. Sf3 Lg4 6. h3 Lh5

Sizilianische Verteidigung (Diagramm 30/31)

1. e4 c5 2. Sf3 e6 3. d4 cxd4 4. Sxd4 Sf6 5. Sc3 d6 6. Le2 h6
2. Sf3 d6 3. d4 cxd4 4. Sxd4 Sf6 5. Sc3 g6 6. Le3 Lg7

Aljechin – Verteidigung (Diagramm 32/33)

1. e4 Sf6 2. e5 Sd5 3. c4 Sb6 4. d4 d6 5. f4 dxe5 6. fxe5 Sc6
3. d4 d6 4. Sf3 Lg4 5. Le2 e6 6. 0-0

c) Geschlossene Spiele [xiv]

Es gibt aber auch Eröffnungen, in denen Weiß mit einem anderen Zug als e4 beginnt, die nennt man Geschlossene Spiele.

Damengambit (Diagramm 34/35/36)

1. d4 d5 2. c4 dxc4 3. e3 Sf6 4. Lxc4 e6 5. Sf3 c5 6. 0-0 a6
2. c4 e6 3. Sc3 Sf6 4. Lg5 Sbd7 5. e3 Le7 6. Sf3 0-0
2. c4 c6 3. Sf3 Sf6 4. Sc3 dxc4 5. a4 Lf5 6. e3 e6

Englische Eröffnung (Diagramm 37/38)

1. c4 e5 2. Sc3 Sf6 3. Sf3 Sc6
1. d4 Sf6 2. c4 g6 3. Sc3 d5

Holländische Verteidigung (Diagramm 39)

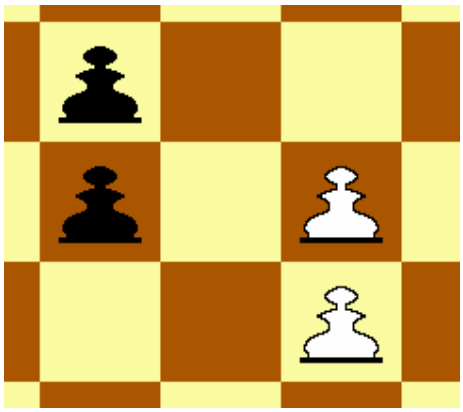
1. d4 e6 2. c4 f5

2. Mittelspiel

Nachdem die Spieler ihre eigenen Figuren entwickelt haben und der König rochiert hat, beginnt in den meisten Fällen das Mittelspiel. Es ist der interessanteste, aber auch der anspruchvollste Spielabschnitt einer Schachpartie. Die Spieler müssen daher eine große Kreativität und Ideenvielfalt besitzen. In den meisten Büchern der Schachliteratur wird das Mittelspiel in Strategie und Taktik geteilt.

a) Strategie

Die Strategie ist der langfristiger Plan, seine eigene Stellung zu verbessern. Dabei verwendet man allgemeine Erfahrungen des Schachspielens. Eine dieser Erkenntnisse ist die Kontrolle des Zentrums, das heißt, die Figuren eines Spielers besetzen die mittleren Felder des Schachbrettes oder können diese im nächsten Zug ereichen. Entsteht in einer Partie eine offene Linien[5], so ist es erstrebenswert diese sofort zu besetzen. Für diese Besetzung verwendet man Schwerfiguren[6]. Es ist unter anderem zweckmäßig die offenen Linien mit zwei sich unterstützenden Türmen zu beschlagnahmen. Zudem ist es nützlich den Wirkungsbereich der gegnerischen Figuren zu verkleinern. Eine weitere strategische Denkweise ist die Vermeidung eines oder mehrere Doppelbauern [7] . Diese haben die negative Eigenschaft sich gegenseitig zu behindern. Dagegen sind Freibauern sehr vorteilhaft. Sie können nicht von den gegnerischen Bauern geschlagen oder blockiert werden. Freibauern können daher leichte zur Endlinie gelangen.



b) Taktik [xv]

Die Taktik eignet sich für die Durchführung kurzfristiger Ziele, die den strategischen Plan realisieren. Man kann diese Ziele mit verschiedenen taktischen Elementen erreichen. Alle Taktiken verfolgen das Ziel, die gegnerischen Figuren durch verschiedene Manöver zu schlagen und somit einen Vorteil zu erspielen.

---

[5] Linien, die von keiner Figur besetzt sind.
[6] Sind Dame oder Turm.
[7] Zwei Bauern derselben Farbe auf einer Linie.

Bedroht ein Spielstein gleichzeitig mehrere Figuren des Gegenspielers, so spricht man von einer ***Gabel***. Diese taktische Idee kann von jeder Schachfigur ausgeführt werden. Kann eine der gegabelten Figuren Schach bieten, so ist es dadurch möglich einige Steine von der feindlichen Bedrohung zu befreien.

Diagramm 41

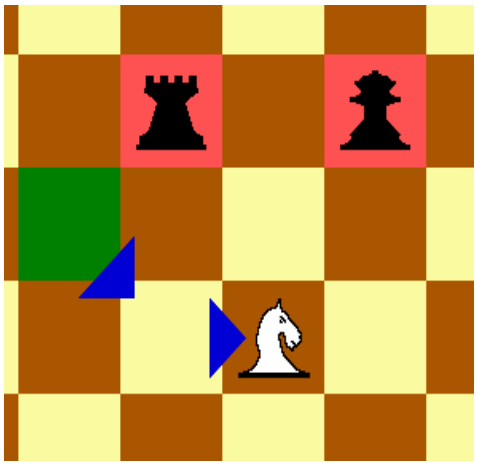


Diagramm 42

Ein ***Spieß*** ist die Bedrohung zweier gleichfarbiger Figuren auf einer Linie. Man sagt, dass der hinten stehende Spielstein „aufgespießt“ ist. Durch einen Spieß gelingt es, die hintere Schachfigur zu schlagen. Denn diese ist die weniger Wertvollere.

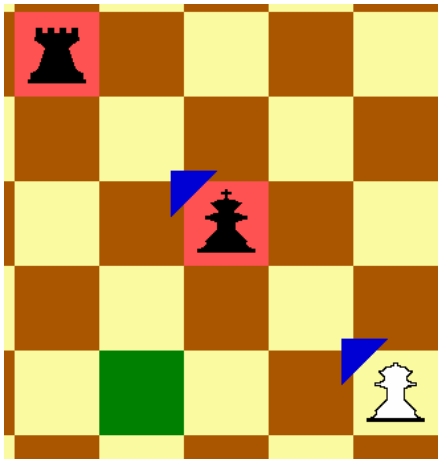
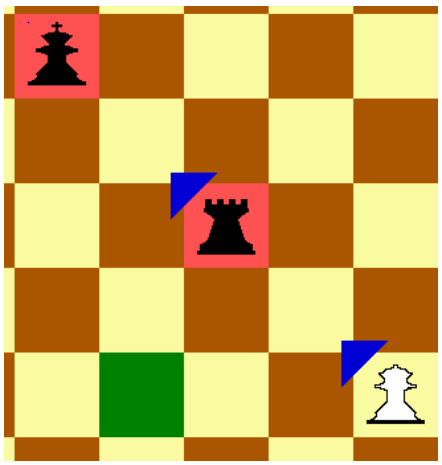
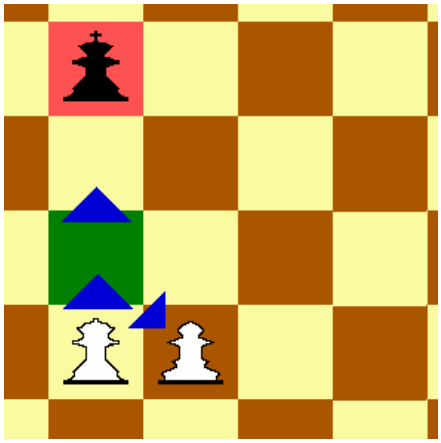


Diagramm 43

Eine ***Fesslung*** verfolgt letztlich das gleiche Prinzip wie der Spieß. Der einzige Unterschied ist, dass die wertvollere Figur hinter einer anderen steht. Die vordere kann sich daher nicht frei bewegen, da es sonst zu hohen Verlusten kommt.



Der ***Abzugsangriff*** ist ein hinterlistiger Feldzug. Dieser wird durch ein Schach geben besonders wirkungsvoll. Bei diesem Angriff muss man einen Spielstein zurückziehen, damit eine stärkere dahinter stehende Figur angreifen kann.



Eine Spielfigur schützt mehrere Steine derselben Farbe. Wird eine davon geschlagen und schlägt man mit dem Beschützer zurück, so ist die zweite zu sichernde Figur ohne Deckung. Bei diesem Ereignis spricht man von ***Überlastung***.

Das ***Opfer*** ist die anspruchvollste Taktik beim Schach, da dies sehr gut überlegt sein muss. Dabei überlässt man dem Gegner eine Figur und kann nicht sofort den gleich Wert des Steines durch ein eigenes Schlagen ausgleichen. Bei einem gut überlegten Opfer kann man nach kurzer Zeit den Verlust ausgleichen und sogar einen Vorteil daraus schlagen. Dieser kann sich bis zu einem Schach oder sogar einem Matt ausweiten.

3. Endspiel

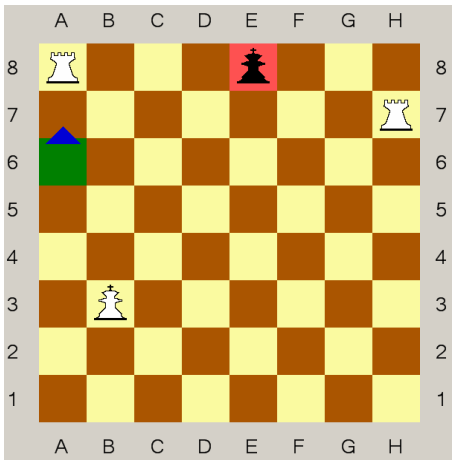
Die Grenze zwischen Mittel- und Endspiel ist fließend. Als gängige Definition fanden wir, dass das Endspiel beginnt, wenn weniger als acht Figuren auf dem Brett stehen. Dies ist aber auch von der Verteilung der Figuren abhängig. Man unterscheidet die Endspiele in elementare und komplexe.

a) Elementare Endspiele [xvi]

Bei diesen Endspielen kann der Spieler mit dem Materialvorteil immer gewinnen. Spielt er nicht fehlerfrei kann es im schlimmsten Fall zu einem Remis kommen.

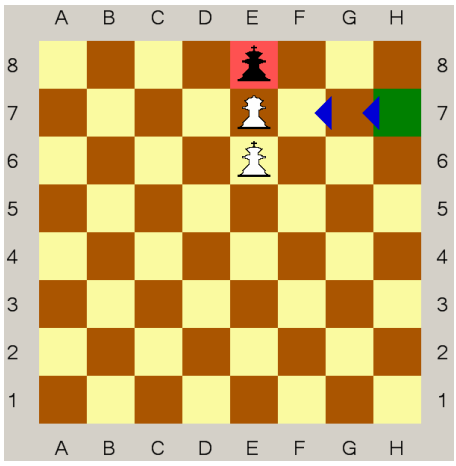
Matt mit zwei Türmen derselben Farbe

Zuerst müssen die Türme auf benachbarte Linien gebracht werden, wobei sie sich nicht gegenseitig behindern sollten. Durch abwechselndes Schach bieten der Türme, ist es möglich den gegnerischen König an den Brettrand zu treiben und diesen dort matt zu setzen.



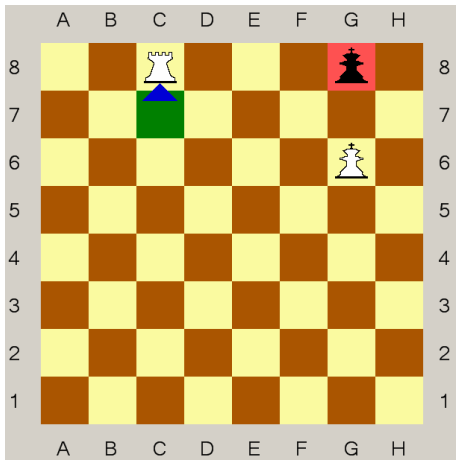
Matt mit einer Dame

Man versucht die Könige nah zusammen zubringen (ein Feld Abstand). Danach wird die Dame zu dem eigenen König gezogen, damit sie gedeckt ist und Schach sagen kann. Der gegnerische König wird an den Brettrand gejagt. Dort wird er letzten Endes zur Aufgabe gezwungen.



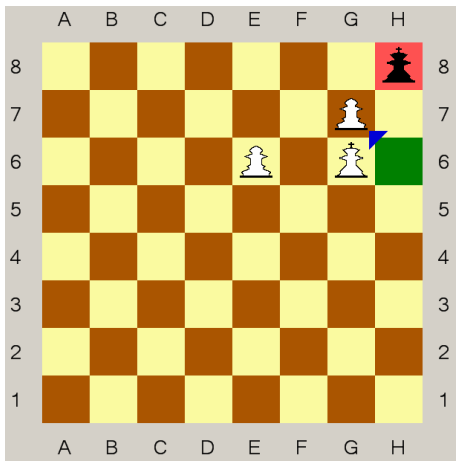
Matt mit einem Turm

Einleitend bringt man die Könige nah zusammen, so dass sie sich gegenüber stehen. Danach wird der Turm auf die Linie des gegnerischen Königs gebracht. Dieser wird durch wiederholen solcher Aktionen an den Brettrand getrieben. Dort kann der König nun matt gesetzt werden.



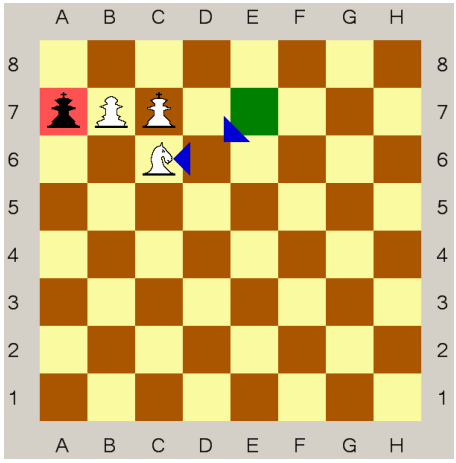
Matt mit zwei Läufern

Mit einem Läufer alleine kann man den Gegner nicht mehr matt setzten. Mit Zweien dagegen ist dies kein Problem. Beide Läufer arbeiten am besten zusammen, wenn sie nebeneinander stehen. Die Läufer teilen so das Schachbrett. Das Ziel dieser Teilung ist, so den gegnerischen König zum Spielbrettrand zu bewegen. Das reicht aber noch



nicht aus. Er muss in eine Ecke gebracht werden. Bei dieser Aktion muss auch der eigene König mit gezogen werden. Sobald man den Gegner in der Ecke festhalten kann, ist es so gut wie geschafft. Nun muss man nur noch darauf acht geben, dass diese Partie nicht Remis endet.

Matt mit Springer und Läufer

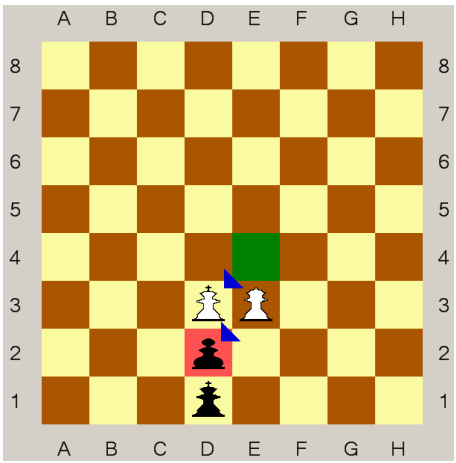


Dieses Endspiel ist wahrscheinlich das Kompliziertes aller elementaren Endspiele, es kommt dafür nicht all zu oft vor. Der gegnerische König muss wie bei einem Endspiel mit zwei Läufern in eine Ecke des Brettes getrieben werden. Dies ist aber nicht einfach und je nach Ausgangsstellung verschieden, von daher kann man es nur versuchen. Sobald man ihn in einer Ecke mit dem Läufer gefangen hat muss der König herangezogen werden um einen Ausbruch zu verhindern. Ist dies geschehen hat der Gegner nur noch zwei Felder wo er seinen König hin bewegen kann. Zuletzt kommt der Springer ins Spiel, er bietet Schach und der Läufer besiegelt das Matt. Bei diesem Endspiel ist die Gefahr groß, dass es Patt wird oder die 50 – Züge – Regel greift.

b) Komplexe Endspiele [xvii]

Bei diesen Endspielen hängt es von der Position der Figuren ab, ob ein Spieler gewinnt oder das Spiel Remis endet.

Dame gegen Bauer

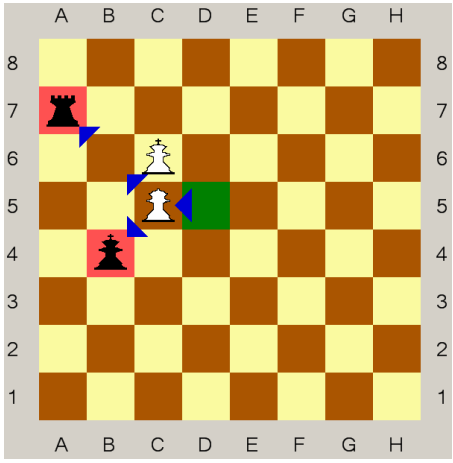


Der Bauer ist für die Dame im Normalfall kein Problem. Sie könnte ihn ohne Gegenwehr schlagen. Ist der Bauer aber kurz vor der Umwandlung, der gegnerische König in der Nähe und der Eigene zu weit vom Bauern entfernt, so muss dieses Endspiel mit sehr hohem Aufwand betrieben werden. Der eigene König muss an den Bauern herangezogen werden. Dies kann nur geschehen, wenn der Gegner den Bauern blockiert. Hat man dies geschafft, ist es

kein Problem mehr für die Dame ein elementares Endspiel daraus zu machen. Steht der Bauer jedoch auf der Läufer- oder Turmlinie, so endet die Partie meist mit Remis.

Dame gegen Turm

Der Vorteil liegt auf jeden Fall bei der Dame. Sie muss den generischen König von seinem Turm trennen. Danach kann man es sich aussuchen ob der allein stehende König sofort matt gesetzt wird oder ob man erst noch den Turm haben will. Für diesen Fall baut die Dame eine Gabel, das heißt Turm und König werden gleichzeitig bedroht.



Turm gegen Springer oder Läufer

Der Turm hat unter normalen Voraussetzungen keine Chance den Gegner matt zu setzten, weder gegen Läufer noch gegen Springer. Macht der Spieler mit König und Läufer bzw. Springer einen Fehler ist ein matt jedoch möglich. Der Turm kann den Gegner nur in die Ecke treiben, mehr aber nicht. Sobald ein Läufer im Spiel ist, so muss außerdem die richtige Ecke gewählt werden. Sie darf für den Läufer nicht zu erreichen sein, ansonsten endet diese Partie mit einem Matt.

IV. Die Entstehung des Schachprogramms

In diesem Teil unserer Seminarfacharbeit wollen wir nun die Entwicklung unseres Schachprogramms darlegen. Zuerst müssen dafür ein paar Grundfragen geklärt werden. Und zwar:

Was wird für die Umsetzung benötigt?
Wie beginnt man am besten mit der Umsetzung?
Welche Probleme können auftreten?

1. Grundlagen

Wir wollen nun die Fragen der Reihe nach beantworten, um damit eine Grundlage für die Entwicklung unseres Programms zu schaffen.

a) Was wird für die Umsetzung benötigt?

Zuerst muss man wissen, welche Informationen benötigt werden, um ein Schachprogramm zu entwickeln. Als Zweites muss man sich über die technischen Vorraussetzungen einigen, die für die Umsetzung benötigt werden.
Zu den Informationen: um ein Schachprogramm zu programmieren, ist es als erstes wichtig sich in den Regeln des Schachs auszukennen (siehe Kapitel II). Denn wie soll der Computer Schachspielen, wenn er nicht weiß, was er für Züge machen darf und welche nicht?
Als zweites ist es nötig den Spielverlauf genauer zu betrachten (siehe Kapitel III). Für die Umsetzung müssen bestimmte Zugfolgen im Eröffnungs-, Mittel- und Endspiel bekannt sein, um sinnvolle Züge von dem Programm erwarten zu können. Des Weiteren sind die Grundlagen in Sachen Taktik und Strategie wichtig, um geeignete Zugalgorithmen für das Programm zu erstellen.
Als nächstes sind Kenntnisse im EDV-Bereich, speziell der Softwareentwicklung, nötig. Womit wir auch zu den technischen Vorraussetzungen kommen. Um unser Ziel zu erreichen benötigen wir eine Entwicklungsumgebung, mit der wir umgehen können und die unseren Anforderungen entspricht. Wir haben uns für Microsoft Visual Studio .Net 2003 ™ entschieden, weil wir im Rahmen unserer Ausbildung mit dem Vorgänger Microsoft Visual Studio 6.0 ™ arbeiten und die Steuerung ähnlich ist. Die

Software haben wir über die Schule mit einer Schülerlizenz erworben, was ein weiteres Kriterium war, sie zu wählen. Als Programmiersprache haben wir C++ gewählt. Dies war nahe liegend, da wir schon Vorkenntnisse in dieser Sprache haben sowie unsere Ausbildung auf ihr basiert und sie gut für unsere Zwecke geeignet ist.
Zuletzt ist es wichtig sich über die Algorithmen Gedanken zu machen, mit dem das Programm arbeiten und seine Züge wählen soll. Dieses Thema ist von solcher Bedeutung für die Entwicklung, dass wir es später noch einmal eingehend behandeln werden.

Da die Vorrausetzungen für die ersten Schritte der Programmentwicklung geschaffen sind, kommen wir nun zur nächsten Frage.

b) Wie beginnt man am besten mit der Umsetzung?

Um dieses Problem zu klären, sollte man sich zu erst im Klaren sein, was mit dem zur Verfügung stehenden Mitteln machbar ist. Nach dieser Analyse folgt dann ein erstes Konzept, welches der erste Schritt zum Programm ist.
Wie schon erwähnt, haben wir uns für die Programmiersprache C++ entschieden, diese Sprache bietet uns die Möglichkeit der OOP, das heißt der Objektorientierten Programmierung. Wir sind also in der Lage mit dieser Programmiersprache eigene Objekte (Klassen) zu schreiben und zu entwickeln, die wir dann später nur noch miteinander verknüpfen (linken) müssen. Dies ermöglicht uns ein effektives Programmieren der einzelnen Funktionen des Programms, durch die Klassenkapselung ist es uns möglich, schnell und gezielt Veränderungen vorzunehmen, da die Klassen in sich geschlossen unabhängig sind und nur über die von uns zur Verfügung gestellten Schnittstellen ansprechbar sind. So verhindern wir unerwünschte Zugriffe auf Daten und ermöglichen uns ein getrenntes Arbeiten an dem Programm, was die Entwicklung beschleunigt.
Des Weiteren haben wir die Möglichkeit, durch die gewählte Entwicklungsumgebung, die Microsoft Fundation Classes (MFC) zu nutzen, dies sind vorgefertigte Klassen, die viele Funktionen, die man für die Programmierung benötigt, schon liefern. Man muss sich diese Klassen nur noch entsprechend anpassen, so ist zum Beispiel das Erstellen eines Fensters mittels MFC sehr einfach. Dieser Umstand führt uns nun auch zum Interface[8], über dass der Benutzer mit dem Programm kommunizieren soll.

[8] Schnittstelle zwischen Benutzer und Programm.

Wir werden mittels MFC ein Fenster mit einem Schachfeld entwickeln, über das die vom Spieler gewünschten Züge eingegeben und die vom Programm ermittelten Gegenzüge ausgegeben werden. Um dies zu erreichen, werden wir uns die verschiedenen Funktionen der MFC zu nutze machen.
Es gibt auch die Möglichkeit eigene Programmbibliotheken zu entwickeln, um sie dann über das Hauptprogramm aufzurufen, was weitere Performance und Kompatibilitäts- sowie Übersichtlichkeitprobleme löst, deswegen werden wir uns diese Funktion auch zu Nutze machen.

Aus den eben gezogenen Schlüssen, ist es uns nun möglich, einen ersten groben Entwurf unseres Programms zu machen, der dann schrittweise verfeinert und ausgebessert wird und als Grundlage für das Schreiben des Programms dient.
Zuerst müssen wir uns einig sein, wie wir unser Programm aufteilen. Dazu haben wir folgende Punkte überlegt:

- Klassenkapselung der Programmfunktionen
- Funktionen für Zuglogik, Zugkontrolle, Zugermittlung und so weiter über eine externe DLL[9] einbinden
- Zusatzinformationen, zum Beispiel das Eröffnungsbuch über externe Dateien einbinden
- Benutzerinterface wird Startpunkt

Des Weiteren wird das Benutzerinterface eine eigene Klasse erhalten, die dann über geeignete Schnittstellen mit der Kernel – DLL kommuniziert. Um uns über den groben Ablauf klar zu werden, haben wir als nächstes die Programmstrukturen entwickelt. Wir sind dabei zu folgendem Ergebnis gekommen:

Programmoberfläche

Initialisierung des Fensters
Anzeigen des Fensters
Warten auf Zugeingabe vom Benutzer
Zug an Kernel-DLL Senden
Gegenzug von Kernel-DLL holen
Zug Ausführen
SOLANGE BIS Spiel beendet
Ergebnis anzeigen

[9] Dynamic Link Library (engl. dynamisch einbindbare Programmbibliothek); Enthält Funktionen, die erst in den Arbeitsspeicher geladen werden, wenn sie benötigt werden.

Kernel – DLL:

Stellung Überprüfen	
Matt?	
1	Spiel beenden
0	Empfangenen Zug überprüfen
	Korrekt?
	1 Ausführen
	0 Nochmalige Eingabe eines Zugs fordern
	Gegenzug ermitteln
	Gegenzug an Hauptprogramm übermitteln

Diese beiden Struktogramme, zeigen den Grundablauf unseres Programms, aber geben noch keinen genauen Aufschluss darüber, wie die einzelnen Funktionen behandelt und ausgeführt werden sollen. Um die Struktur der Klassen und ihre Schnittstellen zu demonstrieren, werden wir später noch ein Diagramm entwickeln. Für die einzelnen Module und Prozeduren werden dann auch jeweils noch eigene Struktogramme entwickelt, doch sollen die beiden hier schon aufgeführten für das erste als grober Leitfaden dienen und somit reichen.

Jetzt haben wir eine Grundlage für die Umsetzung unseres Programms, doch bevor wir beginnen können, sollten wir uns noch mit möglichen Problemen beschäftigen, damit wir während des Entwicklungsprozesses nicht völlig unvorbereitet sind. Womit wir auch zur letzten Frage kommen.

c) Welche Probleme können auftreten?

Beim Entwickeln von Programmen können vor allem folgende Grundlegende Probleme Auftreten, auf die man sich einstellen sollte:

- Kompatibilität und Portierbarkeit
- Komplexität
- Fehler in Algorithmen

Nun zur Erklärung der angesprochenen Punkte. Was die Kompatibilität und Portierbarkeit von Programmen angeht, so ist damit gemeint ob die Einzelnen Teile

untereinander zusammenarbeiten, Das komplette Programm auf anderen Systemen lauffähig ist, bzw. sich vom Entwicklungssystem auf andere Systeme übertragen lässt.
Um von vornherein zu verhindern, dass es zur Inkompatibilität einzelner Programmteile kommt sollte man sich vor Beginn der eigentlichen Programmierarbeit auf bestimmte Schnittstellen zwischen den Programmteilen einigen, d.h. man macht sich aus in welcher Form Daten gesendet werden sollen und wie sie empfangen werden. Die Verarbeitung spielt hier noch keine Rolle. Zur Kompatibilität mit anderen Systemen ist zu sagen, dass man für das Vorhandensein der vom Programm benötigten Systemdateien sowie Ressourcen sorgen muss. Dies führt auch gleich weiter zur Portierbarkeit. Es ist nötig, dass das Programm so gestaltet ist, dass alle benötigten Dateien klar Strukturiert eingebunden werden, damit sie bei einem Datentransport bzw. bei einer Aktualisierung schnell getauscht oder ersetzt werden können. Dies kann nur gewährleistet werden, wenn die einzelnen Programmteile in austauschbaren DLLs implementiert sind. Sind die DLLs plattformunabhängig programmiert, ist es ohne größere Probleme möglich die Lauffähigkeit auf anderen Plattformen zu gewährleisten. Des Weiteren muss man darauf achten, dass bei einem Transport auch alle Dateien vorhanden sind. Wenn dies der Fall ist, ist auch die Lauffähigkeit auf anderen Systemen garantiert, sofern die Systemkonfiguration die Lauffähigkeit des Programms nicht einschränkt.
Ein weiteres, nicht außer acht zu lassendes Problem ist die Komplexität, das heißt je größer das Programm wird, desto mehr Daten müssen verwaltet werden und desto mehr Programmcode gibt es. Dies führt, bei einer nicht sauberen Programmierung und Strukturierung, zu Unübersichtlichkeit. Dadurch wird es schwerer das Programm zu erweitern, zu verbessern oder andere Veränderungen durchzuführen. Deswegen ist es von Anfang an nötig, klar Strukturiert und nach einem vorher abgesprochenen Standart zu programmieren, sowie den Quelltext zu kommentieren. Dies trägt zur Lesbarkeit des Codes bei und macht Änderungen leichter.
Als letztes Problem, auf welches man schon im Vorfeld Einfluss nehmen kann, sei hier noch auf die Algorithmen verwiesen. In den Algorithmen versteckt sich die eigentliche Arbeit des Programms, sie sind es, die ihm sagen, wann es was zu tun hat. Deswegen ist es von Anfang an nötig die Algorithmen auf ihre Konsistenz, das heißt ihre Richtigkeit, zu prüfen. Denn es ist sehr schwer Programmfehler, die durch fehlerhafte Algorithmen auftreten, zu beseitigen. Denn diese werden nicht durch Korrekturhilfen und Syntaxkontrollen angezeigt. Also sollte man immer die Algorithmen selbst einmal durchgehen, um die erwarteten Ergebnisse zu überprüfen. Eine Fehlersuche ist hier

sonst nur mit Hilfe des Debuggers möglich und dies kann ein sehr Zeitintensiver Prozess werden.
Dieser Abschnitt sollte einen Einblick in Probleme geben, die man schon vor dem Beginn der Arbeit am Programm, weitgehend einengen kann. In dem man dies tut, beschleunigt man die Entwicklung, da ein Teil der Zeit, die sonst für Fehlersuche, Datengleichschaltung, Schnittstellenanpassung und Codestrukturierung, die nachträglich sehr viel zeit kosten würden, eingespart werden kann.

Mit dem Abarbeiten der drei Fragen für die Grundlagen haben wir uns nun ein Fundament geschaffen, auf dem wir jetzt langsam anfangen können weiter in Richtung Umsetzung zu arbeiten. Als nächstes werden wir uns nun den Algorithmen widmen.

2. Schachalgorithmen

Ein Schachprogramm braucht einen Algorithmus, der es ihm ermöglicht selber gute Züge zu erzeugen. Dieser Schachalgorithmus besteht bei den heutigen Schachprogrammen aus einer Vielzahl von Komponenten. Die wichtigsten Bestanteile sind folgende:

- Zuggenerator
- Baumsuche
- Bewertungsfunktion

Mit Hilfe dieser Komponenten gelingt es dem Programm alle möglichen Züge zu berechnen. Dabei werden die Züge in Form eines Baumes gespeichert. Bei der Brute – force[10] – Methode werden alle Züge dieses Baums durch die Bewertungsfunktion auf ihre Spielstärke hin untersucht. Der beste Zug, der ermittelt werden konnte, wird dann vom Algorithmus zurückgegeben [xviii].

a) Der Zuggenerator

Der Zuggenerator ist das Herz des Algorithmus. Er bestimmt zu einer vorgegebenen Stellung alle legalen Züge. Der Algorithmus ruft diese Funktion besonders oft auf, so dass die Geschwindigkeit des Algorithmus maßgeblich von dieser Funktion abhängig ist.

[10] Engl. „Rohe Gewalt“; Lösungsmethode eines schweren Problems durch Ausprobieren aller Varianten.

b) Baumsuche [xix]

Die Baumsuche ist das Verfahren mit dem ein Schachalgorithmus einen passenden Zug sucht. Von einer Stammstellung (auch Wurzelstellung), die untersucht werden soll, zweigen sich alle möglichen Züge der am Zug befindlichen Partei ab, wobei der Zuggenerator aufgerufen wird. Sie bilden die Hauptäste des Baums. Von den Hauptästen verzweigen sich dann alle möglichen gegnerischen Züge und bilden dann die Nebenäste. Durch die Verzweigung entstehen Ebenen. Desto mehr Ebenen des Baums vom Algorithmus errechnet werden, desto größer ist seine Spielstärke. Am Ende dieser Verzweigung, in der letzten Ebene, liegen die Blattstellungen. Diese Blattstellungen werden durch die Bewertungsfunktion des Algorithmus bewertet.
Nimmt man an, dass in einer Partie jeder Spieler, wenn er am Zug ist, 40 Züge tätigen kann, so gibt es nach 2 Halbzügen 40 x 40 = 1600 Stellungen, nach 4 Halbzügen 40^4 = 2,6 Millionen Stellungen und nach 6 Halbzügen 40^6 = 4,1 Milliarden Stellungen. Wie man an diesen Zahlen sieht, wächst die Anzahl der Blattstellung exponentiell mit der Anzahl der Ebenen an.
Um die Breite des Baumes klein zu halten und somit die Endstellungen zu verringern, werden nicht immer alle möglichen Züge untersucht, sondern schwache Züge aussortiert. Damit ist es möglich in der gleichen Zeit mehr Ebenen zu berechnen und somit die Spielstärke drastisch zu erhöhen. Dieses Verfahren wird Alpha-Beta-Algorithmus genannt [xx].

Um von den Blattstellungen zu dem besten Zug der Stammstellung zu gelangen, wird ein weiterer Algorithmus angewendet. Der Minmax – Algorithmus.

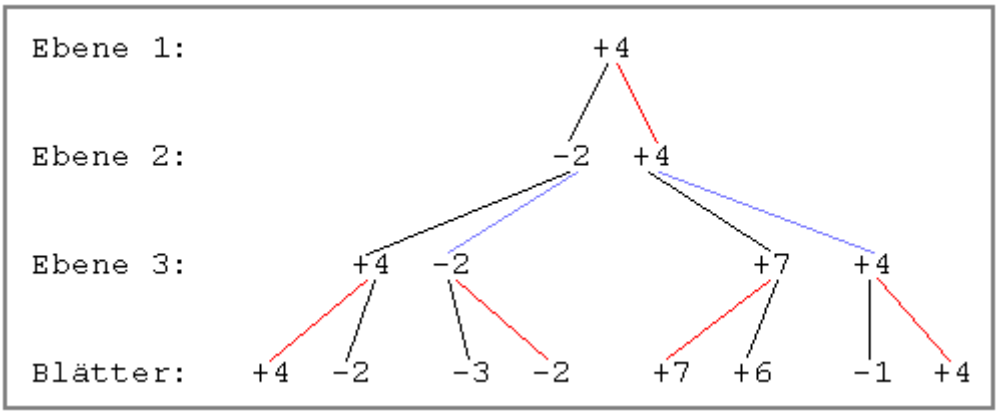


Diagramm Bewertungsbaum

Bei diesem werden die Stellungen der einzelnen Ebenen von den Blattstellungen aus in Richtung der Wurzelstellung bewertet. Dabei wird der stärkste Zug (hohe Bewertung) oder der schwächste Zug (niedrige Bewertung) übernommen, je nachdem welche Partei den Zug ausführen kann. (siehe Bild, welches die Bewertungen in Form eines

Baumes zeigt) Diese beiden Möglichkeiten werden Maximierung (im Bild die roten Linien) beziehungsweise Minimierung (im Bild die blauen Linien) genannt.

c) Die Bewertungsfunktion

Durch die Bewertungsfunktion werden die Endstellungen des Suchbaumes beurteilt. Eine positive Bewertung einer Stellung bedeutet einen Vorteil für Weiß, eine negative Bewertung einen Vorteil für Schwarz. Als Bewertungskriterien verwendet man meist schachliche Faustregeln oder andere Kriterien:

- Materialverhältnisse auf dem Brett (eine Dame ist besser als ein Läufer)
- Sicherheit des Königs (um ein Schachmatt zu verhindern)
- Beweglichkeit der Figuren (eine unbewegliche Figur kann ihre Spielstärke nicht entfalten)
- Zentrumskontrolle (viele Figuren können das Zentrum angreifen)
- offene Linie von Läufer, Turm und Dame (erhöhen die Beweglichkeit)
- Turmverdopplung (damit wird ein stärkerer Angriff ermöglicht, da sich die Türme gegenseitig schützen)
- Doppelbauern (wirken sich negativ aus, da sie sich behindern)
- Freibauern (sind ein Vorteil, da sie sich leichter in eine höherwertige Figur verwandeln lassen als normale Bauern)
- Zentralisierung des Königs im Endspiel (damit wird es erschwert den König matt zu setzen)

Die Bewertungsfunktion ist maßgeblich für die Spielstärke des Schachalgorithmus verantwortlich.

3. Umsetzung

Nachdem wir nun alle Grundlagen, die wir für die Umsetzung benötigen, geschaffen haben, wollen wir uns in dem letzten Teil unserer Arbeit der Dokumentation und der Entwicklung unseres Schachprogramms widmen.

Da es uns in dieser Seminarfacharbeit vor allem darum geht, zu zeigen, wie es möglich ist, dem Computer das Schachspielen beizubringen, haben wir uns auch entschieden unsere Entwicklung mit der Schach – Kernel DLL zu beginnen. Hierfür haben wir zu

allererst eine Klassenhierarchie aufgebaut, welche im Anhang zu finden ist, deren Klassen und Funktionen wir dann schrittweise programmiert und getestet haben.
Nachdem die Elemente des Kernels fertigt gestellt waren, haben wir eine einfache DOS – Ein- und Ausgabeoberfläche programmiert, um zu sehen, inwieweit das Programm seinen Zweck, das Schachspielen erfüllt. So war es uns möglich erste Fehler in den Bewertungsfunktionen und der Zuganalyse zu ermitteln.
Da die DOS – Oberfläche nicht gerade benutzerfreundlich war und die Tests mit ihr auch einen hohen Zeitaufwand hatten, haben wir während dieser Testphase begonnen die neue graphische Oberfläche zu entwickeln und in das Programm einzubinden. Am Anfang haben wir einfach ein Schachbrett gezeichnet, auf denen wir Buchstaben, die den Schachfiguren entsprachen bewegt haben. Diese Oberfläche haben wir dann weiter zum Testen und Verfeinern der Spiellogik des Programms genutzt. Während parallel die Entwicklung einer weiteren graphische Oberfläche gestartet wurde, die unseren Zielvorstellungen entsprechen sollte. Aus Zeitgründen wird auf der dieser Arbeit beigefügten CD aber nur eine Vorabversion enthalten sein. Die fertige Version wird mit allen Funktionen im Kolloquium zum Einsatz kommen. Dies liegt daran, dass uns die Arbeit an der Schach – Kernel DLL wichtiger war, da dies der eigentliche Kern des Programms ist und somit auch für das Schachspielen des Computers verantwortlich ist.

In unserem Konzept haben wir die Schnittstellen von vornherein berücksichtigt und die Klassen des Kernels und der Programmoberfläche so entwickelt, dass sie einen einheitlichen Standart einhalten, welcher in einem späteren Teil dieser Arbeit erklärt wird. Dies ermöglicht uns eine schnelle und unkomplizierte Zusammenführung von Oberfläche und Kernel, des Weiteren ermöglicht uns dies eine problemlose und unabhängige Weiterentwicklungen der einzelnen Programmteile.
Durch die Schnittstellen wird auch das Einlesen und Abspeichern von externen Daten, die unser Programm erzeugt oder benötigt, geregelt. Zu diesen Daten gehören Spielstände, die abgespeichert und wieder geladen werden können, verschiedene Problemstellungen die unser Programm analysieren und lösen kann, das Eröffnungsbuch, auf das unser Programm zurückgreifen kann und zuletzt auch die Bildinformationen für das dargestellte Fenster und die Schachfiguren.

Im nachfolgenden Teil werden wir nun noch einmal einzeln und genauer auf die Entwicklung und Gedanken hinter den einzelnen Programmkomponenten eingehen. Die Struktogramme der wichtigsten Funktionen können im Anhang von interessierten

Lesern eingesehen werden. Weiterhin befinden sich auf der beigefügten CD im Ordner ’Dokumentation\Quellcodes’ die Quellcodes für die einzelnen Funktionen und Klassen als Textdateien zum einsehen. In der Arbeit haben wir aus Verständlichkeitsgründen auf Codebeispiele verzichtet und stattdessen eine allgemeine Formulierung angestrebt.

a) Die Schach – Kernel – DLL

Die SchachKernel – DLL ist das Herzstück unseres Schachprogramms. Sie stellt Datentypen, Klassen und Funktionen bereit, die es dem Computer ermöglichen Schach zu spielen. In ihr sind Schnittellen definiert, die es dem Hauptprogramm (die grafische Oberfläche unseres Schachprogramms) ermöglicht auf die Datentypen, Klassen und Funktionen zu zugreifen. Durch diese Schnittstellen ist es dem Hauptprogramm möglich, mit der SchachKernel – DLL zu kommunizieren, die Ergebnisse der Funktionen zu verarbeiten und diese auf den Bildschirm auszugeben.

Wichtige Klassen der SchachKernel – DLL

Die *Feld* – Klasse speichert die Koordinaten eines Feldes auf dem Schachbrett. Mit Hilfe dieser Klasse ist es möglich gezielt auf ein bestimmtes Feld auf dem Schachbrett zu zugreifen, um beispielsweise Informationen über die Figur, die sich auf diesem Feld befindet, zu bekommen.

Mit der Klasse *Figur* wird ein Spielstein auf einem Feld beschrieben. Die Figur ist dabei eindeutig durch ihren Namen und ihre Farbe bestimmt. Damit ist gewährleiste das nur die Zugmöglichkeiten der entsprechenden Figur vom Zuggenerator als gültige Züge in Betracht gezogen werden.

Die Klasse *Zug* speichert einen Halbzug, das heißt, die Bewegung einer Figur eines Spielers. Über diese Klasse werden die von der SchachKernel – DLL berechneten Spielzüge an das Hauptprogramm weiter geleitet, dadurch ist diese Klasse für die Kommunikation der beiden Programmteile von großer Bedeutung.

Mit der *Brett* – Klasse wird das Spielbrett nachgebildet. Sie beinhaltet alle Informationen die eine Stellung auf einem Schachbrett eindeutig charakterisiert. In ihr wird die Position jeder Figur auf dem Spielbrett und spezielle Statusinformationen

(wie zum Beispiel, ob die Rochade noch erlaubt ist oder ob ein Bauer en passant schlagen darf) gespeichert.
Die *SchachBrett* – Klasse setzt die Regeln um und berechnet mit ihren Funktionen den nächsten Zug für den Computer. Sie überprüft ob ein vom Benutzer eingegebener Zug möglich ist und sorgt dafür, dass man bereits ausgeführte Züge wieder zurück nehmen kann.

b) Schnittstellen

Eine Schnittstelle ist in erster Linie eine Funktion die für den Datenaustausch zwischen einzelnen Programmteilen genutzt wird. Um eine Schnittstelle zu definieren muss man wissen, welche Daten zwischen den einzelnen Programmteilen ausgetauscht werden sollen. Weiterhin ist zu beachten, dass komplexe Programme sich nicht nur auf eine einzige Schnittstelle stützen können. Da es beim Transport vieler Daten zu einem Überfluss an unbenötigten Datensätzen kommt, werden diese in mehrere Funktionen aufgeteilt. Somit garantiert man, dass nur die benötigten Daten übermittelt werden und die Programmteile durch die Auswertung überflüssiger Datensätze nicht überlastet werden.
Vor dem Start der eigentlichen Arbeiten am Programm werden die Schnittstellen definiert, da man durch sie bestimmt, was die einzelnen Programmteile bewirken sollen. Durch die Festlegung der Schnittstellen ist man gezwungen sich auf einheitliche Richtlinie in der Benennung von Funktionen und Variablen festzulegen. Dies wird im Allgemeinen als Namenskonvention bezeichnet.
Die Namenskonvention dient dazu, die Kompatibilität der einzelnen Programmteile zu erhöhen und die spätere Lesbarkeit des Programmcodes zu verbessern. Wir haben uns diesbezüglich auf die im Folgenden erläuterte Konvention geeinigt.

Benennung von Klassen

Klassennamen bestehen aus Aneinanderreihungen mehrere Wörter die jeweils mit einem Großbuchstaben beginnen. Ein Beispiel für einen solchen Klassennamen ist SchachBrett.

Benennung von Membervariablen[11]

Membervariablen werden Analog zu Klassen benannt. Ein Beispiel hierzu ist AmZug.

[11] Definieren die Eigenschaften einer Klasse.

Benennung von Memberfunktionen[12]

Die Memberfunktionen werden in private und öffentliche Funktionen unterteilt. Für die privaten gilt, dass das erste Wort klein und jedes folgende groß geschrieben wird. Als Beispiel sei testZugSchach angeführt. Bei öffentlichen wird wie bei Klassen verfahren.

Benennung von Parametern[13]

Bei Parametern werden alle Wörter kleingeschrieben und jeweils durch einen Unterstrich getrennt. Ein Beispiel ist anzahl_ebenen.

Benennung von lokalen Funktionsvariablen[14]

Hier ist die Benennung wie bei privaten Memberfunktionen. Als Beispiel dient zugTemp.

Benennung von Konstanten[15]

Konstanten bestehen grundsätzlich aus Großbuchstaben. Eventuelle Worttrennungen werden durch Unterstriche vorgenommen was man bei der Konstante NONE_FARBE sieht.

Des Weiteren haben wir uns darauf geeinigt, dass in den Bezeichnern keine Sonderzeichen außer dem Unterstrich sowie keine Umlaute und Zahlen eingesetzt werden.

Die Anwendung von Schnittstellen gestaltet sich wie folgt. Als erstes wird festgelegt, welche Parameter eine Schnittstellenfunktion bekommt und welche Werte sie zurückliefern soll. Im Quelltext erkennt man diese Funktionen durch einem dem Namen vorangestelltes **Set** oder **Get**. Bei Set – Funktionen handelt es sich um Funktion, die Eigenschaften ändern. Sie erhalten nur Parameter und liefern auch nur eine Änderungsbestätigung zurück. Im Gegensatz die Get – Funktionen, diese liefern einen Wert von einer festgelegten Eigenschaft zurück. Hier gibt es zwei Fälle. Im ersten Fall handelt es sich um Funktionen, die ohne Parameter aufgerufen werden.

[12] Definieren die Funktionalität einer Klasse.
[13] Sind die beim Aufruf einer Funktion übergebenen Variablen.
[14] Variablen, die innerhalb einer Funktion deklariert sind.
[15] Variablen, deren Werte vor Programmstart festgelegt sind und während der Ausführung des Programms unveränderlich sind.

Beim Zweiten sind es Funktionen, die einen oder mehrere Parameter erhalten um die Eigenschaft näher einzugrenzen. Anhand ausgewählter Beispiele werden wir dies nun veranschaulichen.

Die Schnittstelle **SetAmZug(Farben farbe)** erhält einen Parameter vom Typ Farben. Dieser legt den am Zug befindlichen Spieler fest. Das nächste Beispiel ist die Schnittstelle **GetAmZug()**, die den gerade aktiven Spieler zurückgibt. Als letztes Beispiel wollen wir die Schnittstelle **GetFigur(Feld feld)** anführen. Diese liefert die Figur, die auf dem angegebenen Feld steht, zurück.

Es gibt aber auch Schnittstellen, die keine Werte ändern oder liefern sollen, sondern nur Ereignisse auslösen. Dies kann mit oder ohne Angabe von Parametern geschehen. Als Beispiel wollen wir hier die Schnittstelle **Setzen(Zug zug)** anführen. Diese bekommt einen Zug als Parameter übergeben und aktiviert dessen Ausführung im Schach – Kernel.

Nachdem die Parameter beziehungsweise die Aktionen der Schnittstellen definiert sind, kann man diese nun von jedem anderen Programmteil aufrufen. Damit ist es möglich auf die Funktionalität des durch die Schnittstellen beschriebenen Programmteils zu zugreifen.

c) Programmoberfläche

Die Oberfläche ist wichtig für die Kommunikation zwischen Programm und seinem Benutzer. Diese Verständigung geschieht durch ein Fenster oder durch eine Eingabemaske.

Wir haben uns entschieden, ein Fenster als Kommunikationsschnittstelle mit dem Benutzer einzusetzen. In dem Fenster wird ein Schachbrett dargestellt, auf dem die aktuelle Stellung des Spieles angezeigt wird. Des Weiteren dient das Feld dazu, dem Benutzer per Drag & Drop seine Züge durchzuführen. Er kann, wenn er am Zug ist, mit der Maus auf eine der Figuren klicken und diese dann bei gedrückter Maustaste auf das gewünschte Feld ziehen. Wenn er die Maus wieder loslässt, wird die Figur auf das Feld gesetzt. Der Zug des Spielers wird danach an den Schach – Kernel weitergeleitet. Dieser prüft ihn und liefert gegebenenfalls einen Gegenzug zurück, welcher dann auf dem Schachbrett dargestellt wird. Nun kann der Benutzer seinen nächsten Zug durchführen.

Das Fenster enthält daneben Informationen darüber, wer am Zug ist, ob das Programm einen Gegenzug ermittelt, welche Züge zuletzt durchgeführt wurden und

Sonderinformationen. Beispielsweise wenn der König im Schach steht. Außerdem zeigt es, wenn die Partie mittels Remis oder Matt beendet wurde.

Um diese Programmoberfläche zu umzusetzen, haben wir uns entschieden, das Schachfeld statisch, als Hindergrund in das Fenster zu zeichnen. Auf dieses Feld werden dann die Schachfiguren, die als externe Bilddateien eingeladen werden gezeichnet. So müssen wir immer nur die Positionen der aktuell bewegten Schachfiguren überprüfen und ihre Anzeige zur korrekten Darstellung aktualisieren.
In der Statusleiste des Fensters ist der am zugbefindliche Spieler und ob das Programm gerade einen Zug ermittelt, dargestellt. Neben dem Schachfeld befindet sich eine Textbox, in dieser werden die Zuginformationen der gemachten Züge angezeigt.

Zur Steuerung der Programmfunktionen haben wir ein Menü erstellt, das bequem mit der Maus gesteuert werden kann. Neben der Möglichkeit der Maussteuerung haben wir auch die Möglichkeit einprogrammiert, die Menübefehle über Shortcuts[16] aufzurufen. Dies erhöht den Komfort für den Benutzer.

Den Entwurf, sowie den Quelltext der programmtechnischen Umsetzung der Oberfläche befinden sich für den interessierten Leser im Anhang und können dort eingesehen werden.

[16] Kurztaste; Sie erleichtern die Programmsteuerung, indem man durch einen einfachen Tastendruck auf der Tastatur bestimmte Menübefehle schnell ausführen kann.

V. Die Entwicklungsgeschichte unseres Programms

In diesem Kapitel wollen wir dem Leser die einzelnen Etappen der Entwicklung unseres Programms vorstellen. Hierzu haben wir wichtige Programmversionen in chronologischer Reihenfolge aufgeführt und deren Merkmale dokumentiert.

1. Version 0.5.0.0

Diese Version war die erste lauffähige Testversion unseres Programms. Als Oberfläche diente eine einfache Konsole (Abb. 1). Hier fand noch keine Trennung der einzelnen Programmteile statt. Es wurde die erste Kernel – DLL verwendet, die die einfachen Zugregeln einhielt. Sie hat ihre Züge in der Zugtiefe von einem Halbzug berechnet. Die Spielstärke war dem entsprechend schlecht, aber wir waren froh, dass das Einhalten der Regeln funktionierte.

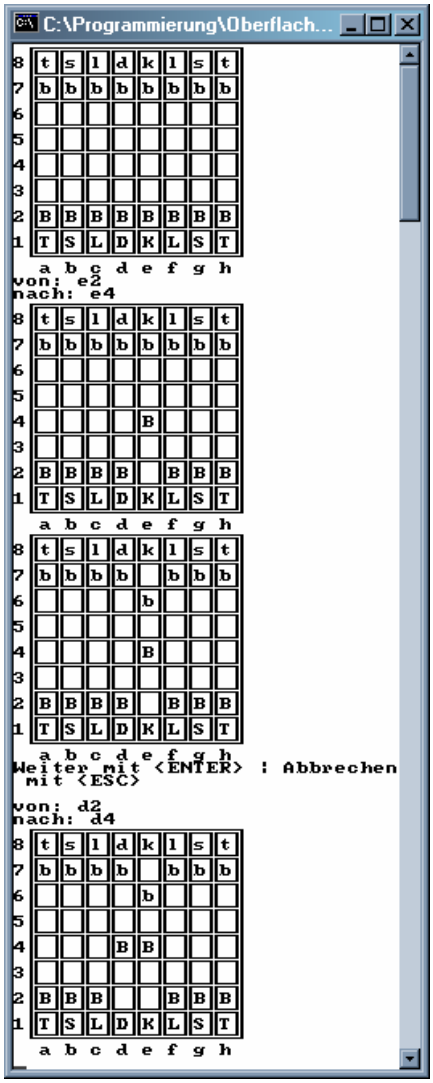


Abbildung 1

2. Version 0.5.3.1

Wie die Vorversionen, verwendete diese die Konsole als Oberfläche. Die grundlegenden Änderungen zu den früheren Versionen bestanden darin, dass diese auch die komplizierten Regeln wie Rochade, Bauernumwandlung und En passant beherrschte. Die Spielstärke blieb unverändert.

3. Version 0.6.1.2

Es handelte sich hierbei um eine Weiterentwicklung der Version 0.5.3.1, wobei die Verbesserung darin bestand, dass Kernel und Oberfläche ab jetzt getrennt verwaltet wurden.

4. Version 1.0.0.0

Das wichtigste war, dass wir unsere erste eigene grafische Oberfläche entwickelt hatten (Abb. 2). Von der Spielstärke unterschied sie sich nicht von den früheren Versionen. Durch diese Weiterentwicklung konnten die Tests angenehmer durchgeführt werden, auch wenn die Darstellung der Figuren nicht sehr gelungen war. Es traten auch noch einige Fehler auf.

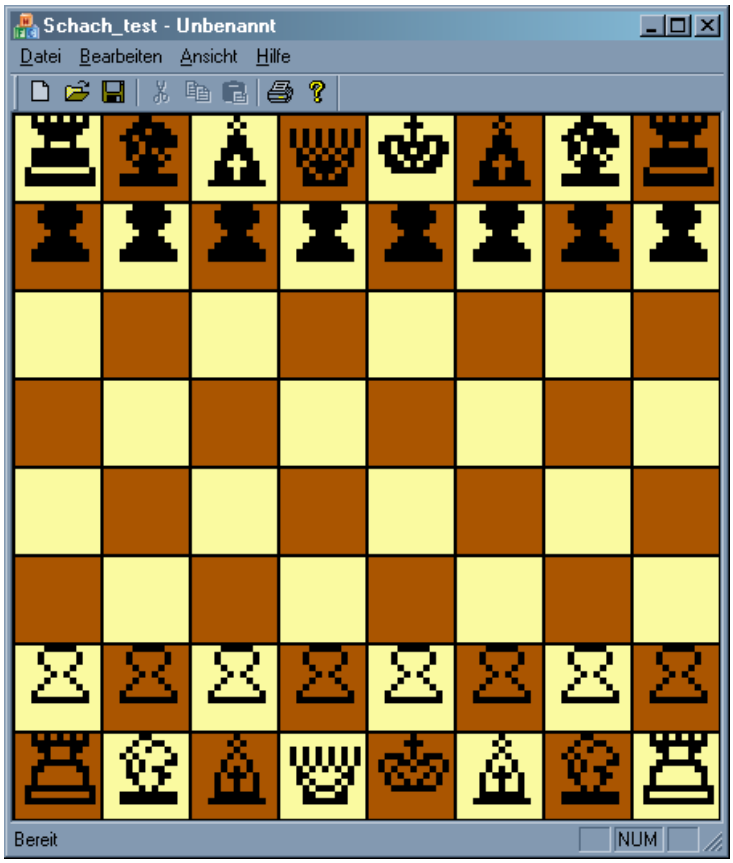


Abbildung 2

5. Version 1.2.7.4

Die Verbesserungen zum Vorgänger lagen darin, dass nahezu alle kleineren und größeren Fehler in der Anzeige und bei den Schnittstellen behoben wurden. Außerdem wurde nun in der Statusleiste der am Zug befindliche Spieler angezeigt. In den weiteren Versionen konnte nun die Oberfläche problemlos verwendet werden. Dies war nötig, damit der Kernel weiterentwickelt werden konnte. Parallel dazu haben wir die unabhängige Weiterentwicklung der Grafikoberfläche gestartet. Sie aber noch nicht eingebunden, um eventuelle Fehler, die die Entwicklung des Kernels stören könnten, zu vermeiden.

6. Version 2.0.5.7

In dieser Version gelang es uns, die Rechentiefe des Programms zu steigern. Es konnten nun die Anzahl an Halbzüge, die berechnet werden sollen, angegeben werden. Des Weiteren wurde die Bewertungsfunktion verbessert. Diese Maßnahmen führten zu einer erheblichen Steigerung der Spielstärke unseres Programms. Die weiteren Versionen der 2er Reihe zeichneten sich hauptsächlich durch Arbeiten an der Implementierung eines Eröffnungsbuches sowie kleinerer Optimierungen aus. Wir strebten damit die Erhöhung der Ausführungsgeschwindigkeit des Programms an.

7. Version 3.0.1.0

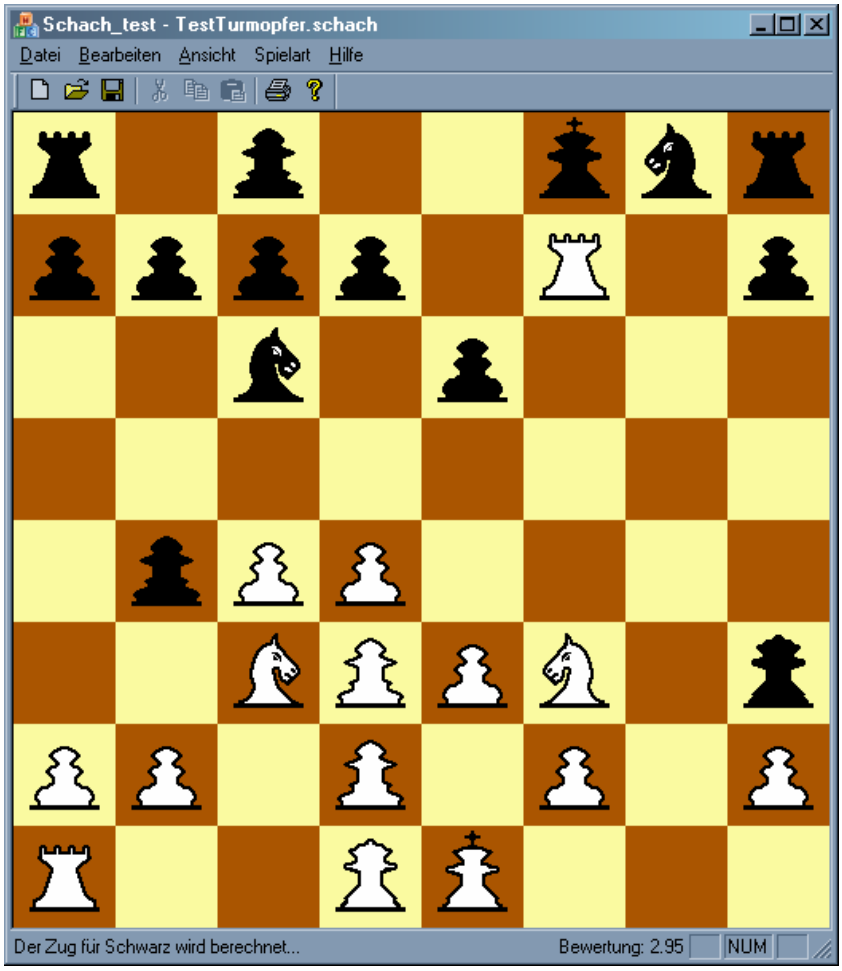


Abbildung 3

Diese neue Oberfläche, die ab diesem Zeitpunkt zum Einsatz kam (Abb. 3), zeichnet sich durch ein komplett neues Design aus. Der Schwerpunkt lag auf der Neugestaltung des Figurensatzes, der wie sein Vorgänger eine Eigenentwicklung ist. Zusätzlich wurden weitere Informationen der Statusleiste hinzugefügt, um dem Benutzer den Spielablauf zu vereinfachen. Des Weiteren ist eine komplett neue Steuerung für das Programm eingeführt worden, die benutzerfreundlicher ist. Die Oberfläche arbeitete problemlos mit der aktuellsten Version des Kernels zusammen, was für uns eine Bestätigung unseres Konzepts war.

8. Version 3.5.1.0

Ist die zurzeit aktuellste Erweiterung unseres Programms. Die hauptsächliche Änderung zum Vorläufer besteht in der Einführung des Multithreading[17]. Außerdem wurden ein paar noch auftretende Fehler behoben.

Zur Abgabe ist die ***Version 4.0.0.0*** angestrebt, welche ein neues verbessertes Eröffnungsbuch enthalten sowie eine Steigerung der Spielgeschwindigkeit aufweisen soll. Bis zum Kolloquium wird diese Version weiterentwickelt um eine höhere Spielstärke und mehr Komfort zu erreichen.

[17] Ist das parallele Ausführen der einzelnen Programmteile (Oberfläche und Kernel).

Nachwort

Durch die Arbeit an diesem Thema wurde uns die große Komplexität des Problems der Entwicklung eines Schachprogramms deutlich. Einem geübten Schachspieler fällt es leicht, aus einer Spielstellung heraus den richtigen Zug zu finden. Der Computer hingegen benötigt dafür viele Algorithmen, deren Entwicklung schwierig ist. Nach anfänglichen Problemen, ist es uns gelungen ein lauffähiges Programm zu entwickeln. Dieses spielt zumindest auf einem Anfängerniveau Schach.

Im Laufe der Arbeit an unserer Schachsoftware haben sich unsere Fähigkeiten im Programmieren verbessert. Diese Kenntnisse erwarben wir durch das Lösen der auftretenden Probleme. Zudem haben wir unsere eigene Spielstärke gesteigert. Die Verbesserung begründet sich auf die Entwicklung der Bewertungsfunktion unseres Programms. Eine eingehende Beschäftigung mit der Strategie und der Taktik prägte uns zusätzlich.

Wir wollen nun auf unser Kolloquium verweisen. Dort werden wir unser Programm vorstellen und auf die Tests eingehen. Sie waren zur Weiterentwicklung nötig. In unserer Seminarfacharbeit wurden die Versuche, aus Platzgründen nicht näher erläutert.

Anhang

Struktogramme

Statusbar

Datenstruktur:

Name	Typ	Verwendung
pDoc	CSchach_testDoc *	Zeiger auf das Hauptdokument
pFrameWnd	CMainFrame *	Zur Ansteuerung der Statusbar
text	CString	Auszugebender Text in der Stausbar

<table>
<tr><th colspan="4">void CSchach_testView::UpdateStatusBar(Farben farbe)</th></tr>
<tr><td colspan="4">Deklaration: Datenstruktur</td></tr>
<tr><td colspan="4">farbe GLEICH 'NONE_FARBE'?</td></tr>
<tr><td rowspan="11">1</td><td colspan="3">IST 'WEISS' am Zug?</td></tr>
<tr><td rowspan="5">1</td><td colspan="2">IST 'WEISS' im Schach?</td></tr>
<tr><td>1</td><td>text = "Weiß ist am Zug und steht im Schach"</td></tr>
<tr><td>0</td><td>text = "Weiß ist am Zug"</td></tr>
<tr><td colspan="2">text = text + "\t\tBewertung: " + pDoc->Schach.Bewertung()</td></tr>
<tr><td colspan="2">Ausgabe (text)</td></tr>
<tr><td rowspan="5">0</td><td colspan="2">IST 'SCHWARZ' im Schach?</td></tr>
<tr><td>1</td><td>text = "Schwarz ist am Zug und steht im Schach"</td></tr>
<tr><td>0</td><td>text = "Schwarz ist am Zug"</td></tr>
<tr><td colspan="2">text = text + "\t\tBewertung: " + pDoc->Schach.Bewertung()</td></tr>
<tr><td colspan="2">Ausgabe (text)</td></tr>
<tr><td>0</td><td colspan="3">---</td></tr>
<tr><td colspan="4">farbe GLEICH 'WEISS'?</td></tr>
<tr><td rowspan="3">1</td><td colspan="3">text = "Der Zug für Weiß wird berechnet..."</td></tr>
<tr><td colspan="3">text = text + "\t\tBewertung: " + pDoc->Schach.Bewertung()</td></tr>
<tr><td colspan="3">Ausgabe (text)</td></tr>
<tr><td>0</td><td colspan="3">---</td></tr>
<tr><td colspan="4">farbe GLEICH 'SCHWARZ'?</td></tr>
<tr><td rowspan="3">1</td><td colspan="3">text = "Der Zug für Schwarz wird berechnet..."</td></tr>
<tr><td colspan="3">text = text + "\t\tBewertung: " + pDoc->Schach.Bewertung()</td></tr>
<tr><td colspan="3">Ausgabe (text)</td></tr>
<tr><td>0</td><td colspan="3">---</td></tr>
</table>

Berechnung starten

Datenstruktur:

Name	Typ	Verwendung
pObject	Cschach_testDoc*	Zeiger auf Hauptdokument

UINT SetzenAufSchachbrett(LPVOID pParam)
Deklaration: Datenstruktur
Statusbar Aktualisieren
Setze 'berechnung' auf true
Zug berechnen (Angabe der Zugtiefe)
Setze 'berechnung' auf false
return 0

Bild laden

Datenstruktur:

Name	Typ	Verwendung
testbmp	CBitmap	Zu zeichnendes Bild
pOldBitmap	CBitmap *	Zeiger auf das Bild
dcMemory	CDC	Speichercontext für das Bild

void CSchach_testView::LadeBmp(CDC* pDC, int BMP_ID, int x, int y)

Deklaration: Datenstruktur	
Laden des Bildes mit der ID 'BMP_ID' Erfolgreich?	
1	Zuweisung der Bildinformation an 'testbmp'
	Selektieren des Bildes
	Zeichnen des bildes an Position (x, y)
	Bild zurücksetzen
0	Fehler: "Fehler beim Laden des Bildes %d\n",BMP_ID

Maussteuerung

void CSchach_testView::OnLButtonDown(UINT nFlags, CPoint point)

Wird Zug berechnet?		
1	return	
0	---	
Ist Computer am Zug?		
1	return	
0	---	
Auswahl?		
1	Schachmatt?	
	NONE_FARBE	Setze Ausgangsfeld
		Aktualisiere Bild
	WEISS	A: "Weiß ist Schachmatt"
	SCHWARZ	A: "Schwarz ist Schachmatt"
	SW	A: "Das Spiel ist Remis"
	default	---
0	Setze Zielfeld	
	ziehe Figur	
	Aktualisiere Bild	

Feld zeichnen

Datenstruktur:

Name	Typ	Verwendung
pDoc	Cschach_testDoc*	Zeiger auf Hauptdokument
b	int	Vertikale Koordinate des Schachbretts
z	int	Horizintale Koordinate des Schachbretts

void CSchach_testView::OnDraw(CDC* pDC)	
Deklaration: Datenstruktur	
b=0 (1) 7	
z=0 (1) 7	
Welche Figur steht auf Feld mit Koordinaten (b,z)	
LadeBmp(pDC,Figur,b*64,z*64)	
Rahmen Zeichnen?	
1	Wähle Figur
	Zeichne Rahmen
0	---
Statusbar aktualisieren?	
1	UpdateStatusBar(Aktueller Spieler)
0	---

Oberfläche – Setzen

Datenstruktur:

Name	Typ	Verwendung
result	bool	Gibt an, ob das Setzen erfolgreich war
tempZug	Zug	Temporäres Zugobjekt

bool CSchach_testDoc::Setzen(CSchach_testView *view, Feld vonFeld, Feld nachFeld)		
Deklaration: Datenstruktur		
Schachmatt?		
NONE_FARBE	Ist Computer am Zug?	
	1	AfxBeginThread(SetzenAufSchachbrett, this)
	0	Vom Spieler gewählten Zug ausführen
	result = Zug erfolgreich ausgeführt?	
	Bild Aktualisieren	
WEISS	A: "Weiß ist Schachmatt"	
SCHWARZ	A: "Schwarz ist Schachmatt"	
SW	A: "Das Spiel ist Remis"	
default	---	
Statusbar Aktualisieren		
return (result)		

Zug berechnen

Datenstruktur:

Name	Typ	Verwendung
zugNummerListe	Liste<size_t>	Zugnummern der besten Zügen
zugTemp	Zug	Rückgabezug
zuegeEroeffnung	Liste<Zug>	Züge aus dem Eröffnungsbuch
zuegeW	Liste<Zug>	Mögliche Züge für Weiß
zuegeS	Liste<Zug>	Mögliche Züge für Schwarz
farbe	Farben	Farbe die am Zug ist
baum	Knoten	Baum der generierten Züge

Zug SchachBrett::BerechneZug
(bool eroeffnungs_buch, size_t such_tiefe)

Deklaration: Datenstruktur			
Zufallsgenerator initialisieren			
Züge für Weiß und Schwarz ermitteln			
farbe ?			
WEISS	eroeffnungs_buch ?		
	1	Lade Züge aus Eröffnungsbuch	
		Aussortieren ungültiger Züge	
		zuegeEroeffnug nicht leer ?	
		1	Rückgabe: zufälliger Zug aus zugeEroeffnung
	baum aufbauen		
	Suche in dem Baum nach den besten Zügen		
	zugNummerListe nicht leer ?		
	1	Rückgabe: zufälliger Zug aus zugNummerListe	
SCHWARZ	eroeffnungs_buch ?		
	1	Lade Züge aus Eröffnungsbuch	
		Aussortieren ungültiger Züge	
		zuegeEroeffnug nicht leer ?	
		1	Rückgabe: zufälliger Zug aus zugeEroeffnung
	baum aufbauen		
	Suche in dem Baum nach den besten Zügen		
	zugNummerListe nicht leer ?		
	1	Rückgabe: zufälliger Zug aus zugNummerListe	

Kernel – Setzen

Datenstruktur:

Name	Typ	Verwendung
name	Namen	Name der gesetzten Figur
farbe	Farben	Farbe der gesetzten Figur

bool SchachBrett::Setzen(Zug zug)

Deklaration: Datenstruktur					
Züge für Weiß und Schwarz ermitteln					
farbe ?					
WEISS	Ist zug einer von den möglichen Zügen für Weiß ?				
	1	name ?			
		BAUER	Ist zug ein Doppelschritt eines Bauern ?		
			1	Flag für Doppelschritt setzen	
			0	Flag für Doppelschritt zurücksetzen	
			Ist zug ein Umwandlungszug ?		
			1	Umwandlungsfigur nicht vorgegeben ?	
				1	Umwandlungsdialog aufrufen
			Ist zug ein en passant Zug und möglich ?		
			1	en passant Zug vorbereiten	
			Zug des Bauern ausführen		
		KOENIG	Ist zug eine Rochade und diese möglich?		
			1	Rochade ausführen	
			0	Zug des Königs ausführen	
			Flag für Rochade zurücksetzen		
		TURM	Flag für Rochade zurücksetzen		
			Zug des Turms ausführen		
		sonst	Zug der Figur ausführen		
	0	Rückgabe: false			
SCHWARZ	Ist zug einer von den möglichen Zügen für Schwarz ?				
	1	name ?			
		BAUER	Ist zug ein Doppelschritt eines Bauern ?		
			1	Flag für Doppelschritt setzen	
			0	Flag für Doppelschritt zurücksetzen	
			Ist zug ein Umwandlungszug ?		
			1	Umwandlungsfigur nicht vorgegeben ?	
				1	Umwandlungsdialog aufrufen
			Ist zug ein en passant Zug und möglich ?		
			1	en passant Zug vorbereiten	
			Zug des Bauern ausführen		
		KOENIG	Ist zug eine Rochade und diese möglich?		
			1	Rochade ausführen	
			0	Zug des Königs ausführen	
			Flag für Rochade zurücksetzen		
		TURM	Flag für Rochade zurücksetzen		
			Zug des Turms ausführen		
		sonst	Zug der Figur ausführen		
	0	Rückgabe: false			
zug speichern					
Farbe des Spielers der am Zug ist ändern					
Rückgabe: true					

Bewertung

Datenstruktur:

Name	Typ	Verwendung
result	double	Berwertungsergebniss
zuegeW	Liste<Zug>	Mögliche Züge für Weiß
zuegeS	Liste<Zug>	Mögliche Züge für Schwarz
wertW	int	Materialwert für Weiß
wertS	int	Materialwert für Schwarz

double SchachBrett::Bewertung()		
Deklaration: Datenstruktur		
Schachmatt?		
WEISS		result = -1000
SCHWARZ		result = 1000
zuegeW = Mögliche Züge von Weiß		
zuegeS = Mögliche Züge von Schwarz		
Bestimme Materialwert für Schwarz (wertS) und Weiß (wertW)		
result = (zuegeW - zuegeS) / 20		
result = result + (wertW - WertS)		
Ist wenig Material?		
1	WEISS im Vorteil?	
	1	result = result + (7 - Abstand Könige)
	0	result = result - (7 - Abstand Könige)
Rückgabe: result		

Diagramm – Programmkommunikation

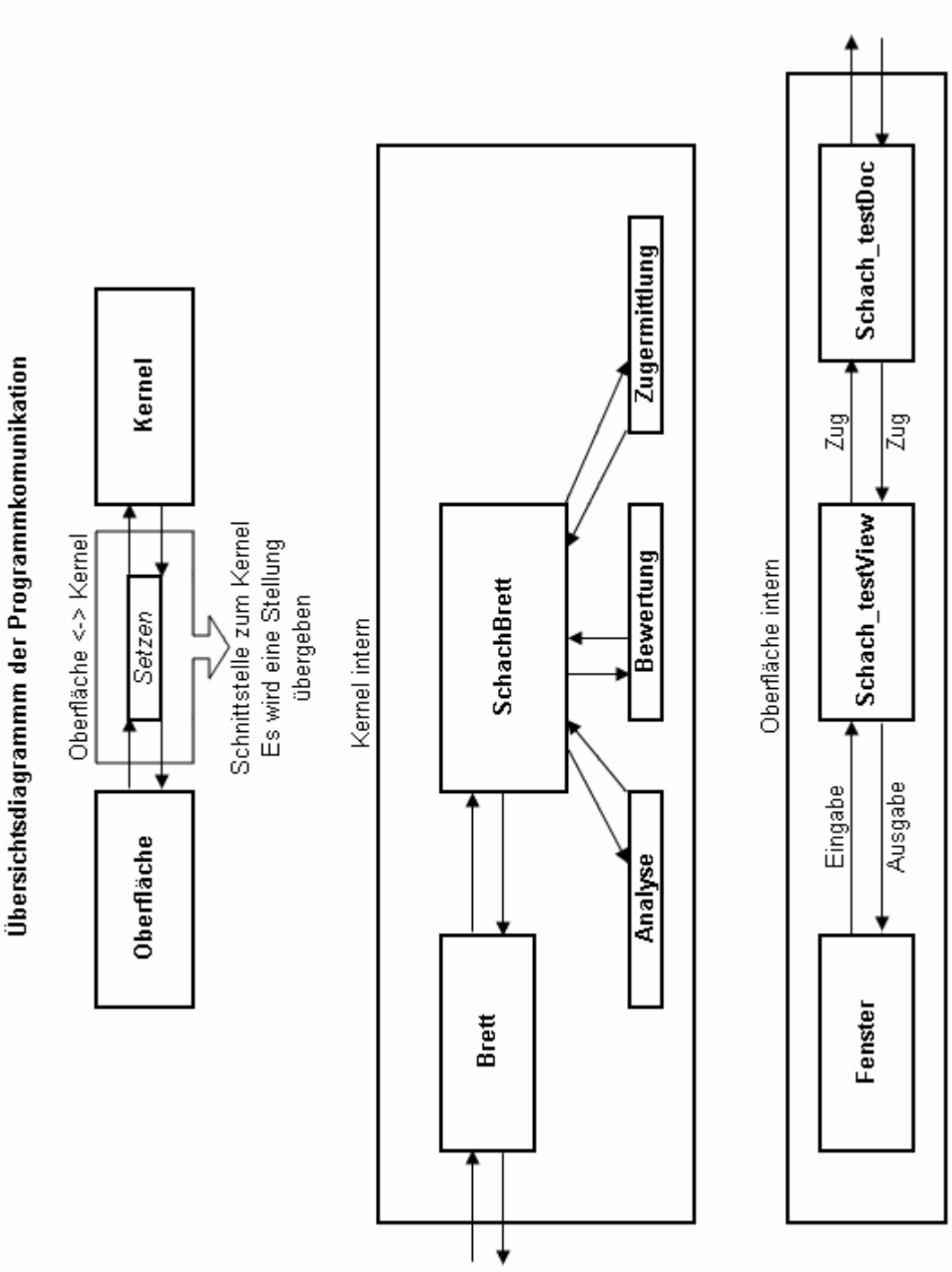


Diagramme Eröffnung

Der nachfolgende Anhang beinhaltet die Diagramme für die Endstellungen der Eröffnung, zur besseren Übersicht.

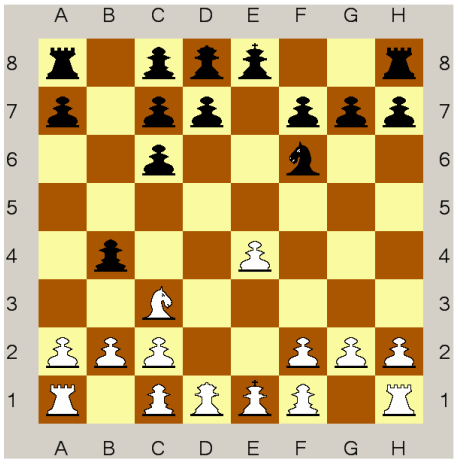


Diagramm 11

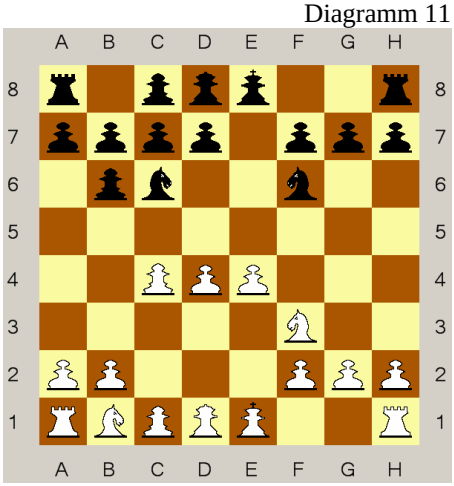


Diagramm 12

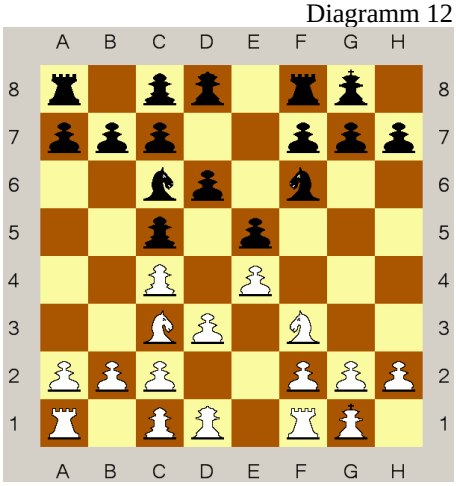


Diagramm 13

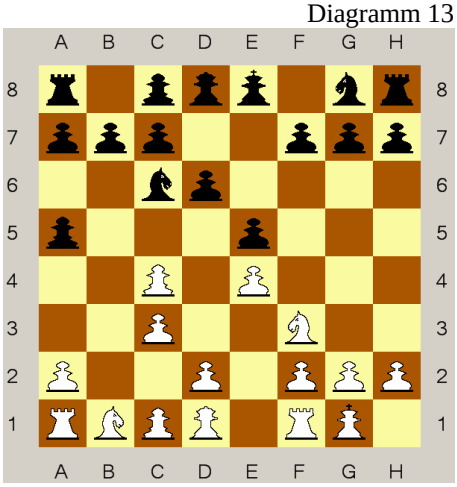


Diagramm 14

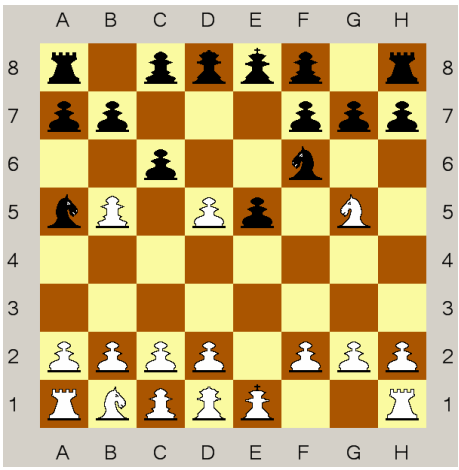


Diagramm 15

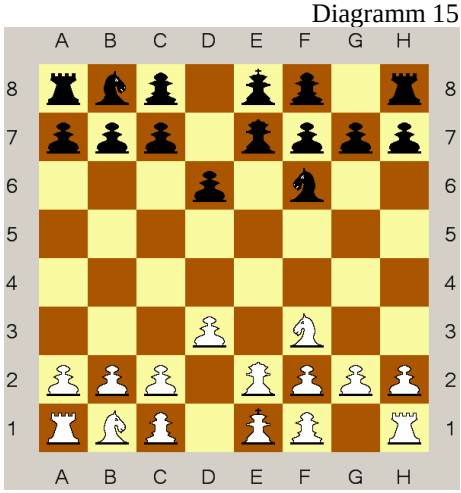


Diagramm 16

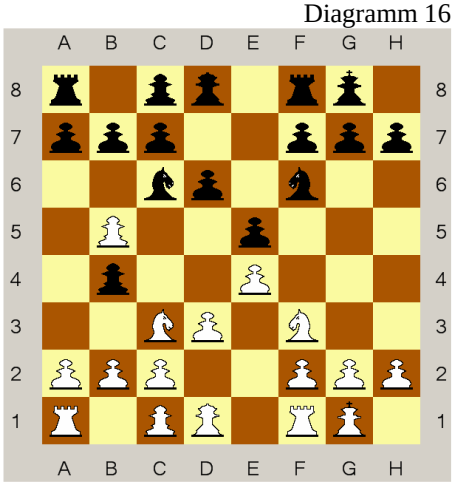


Diagramm 17

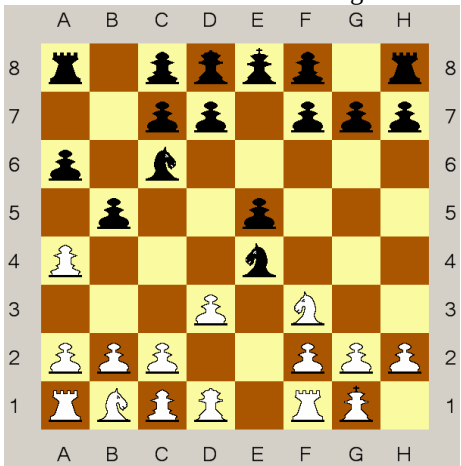


Diagramm 18

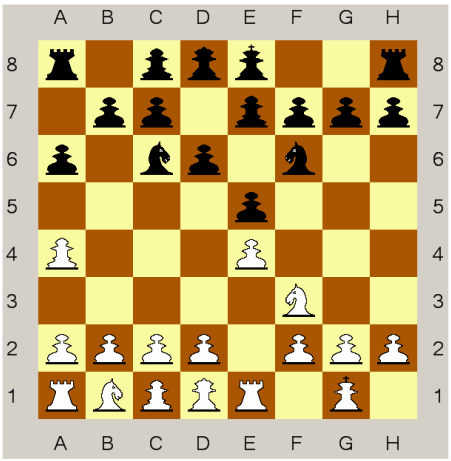


Diagramm 19

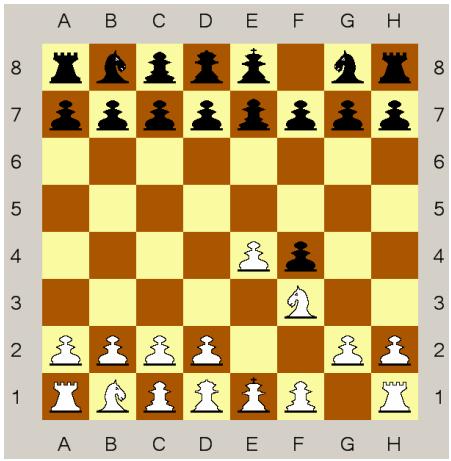


Diagramm 23

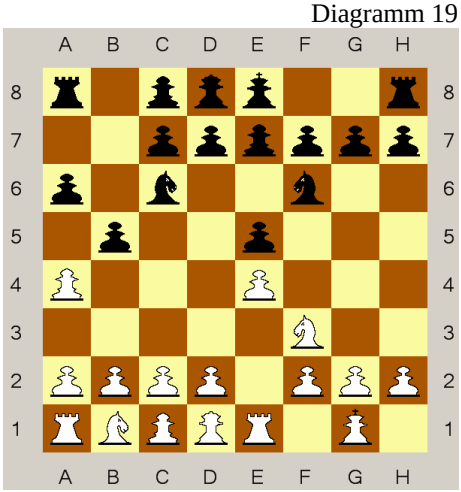


Diagramm 20

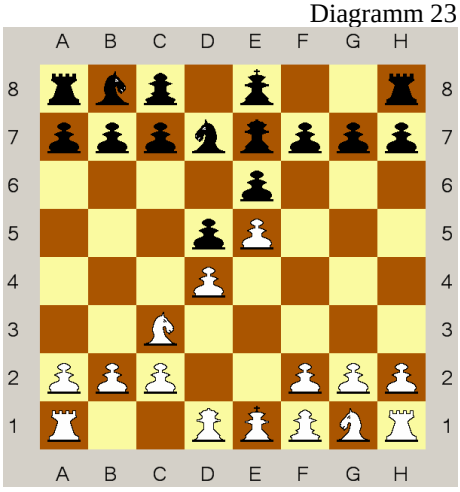


Diagramm 24

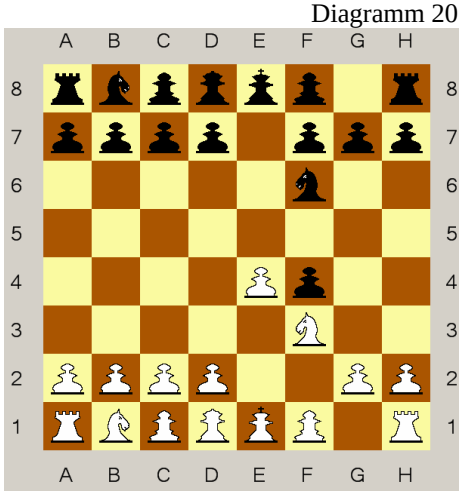


Diagramm 21

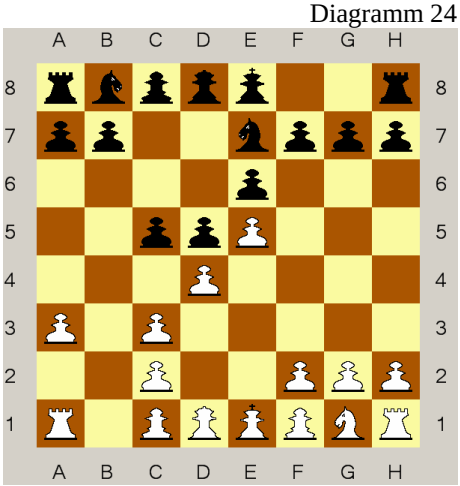


Diagramm 25

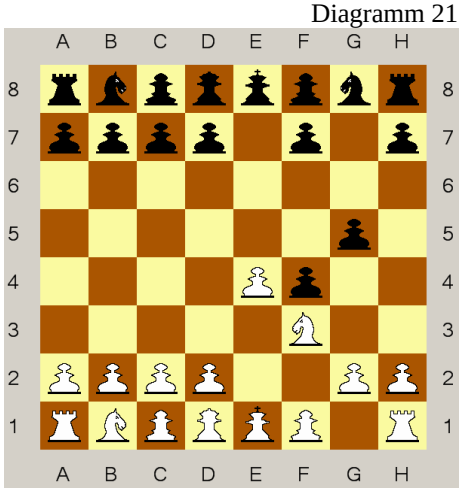


Diagramm 22

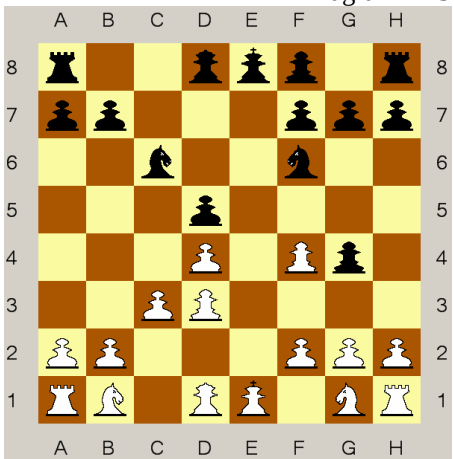


Diagramm 26

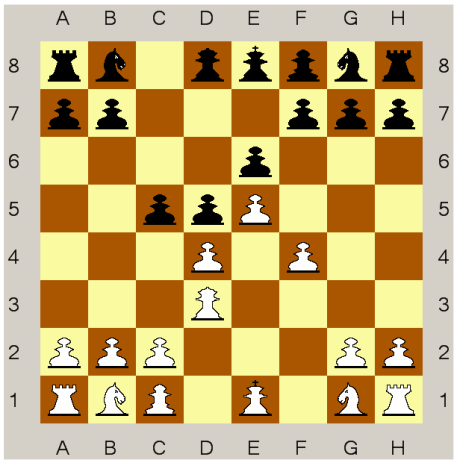


Diagramm 27

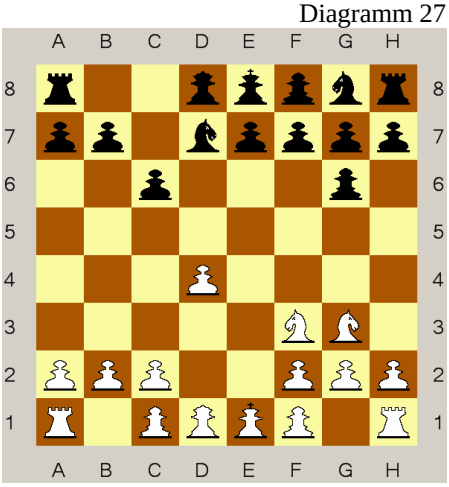


Diagramm 28

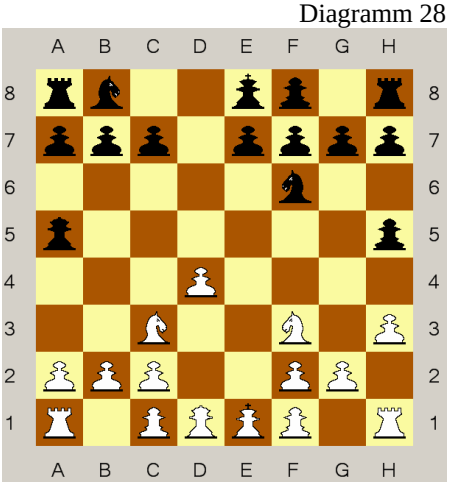


Diagramm 29

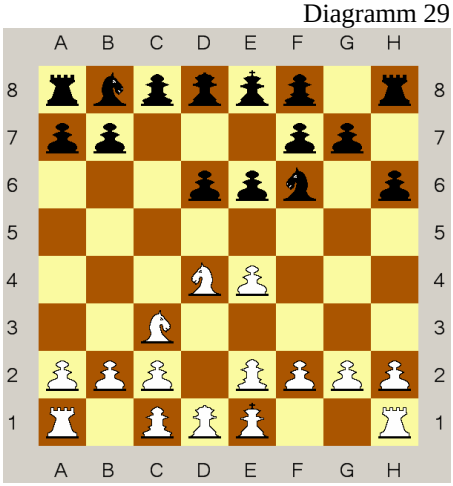


Diagramm 30

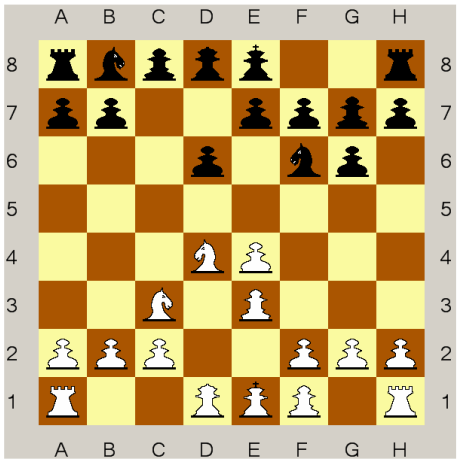


Diagramm 31

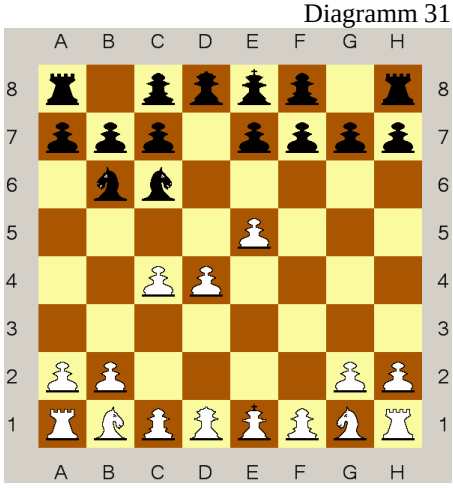


Diagramm 32

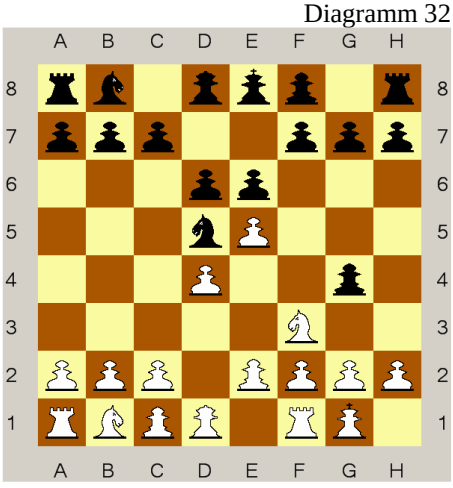


Diagramm 33

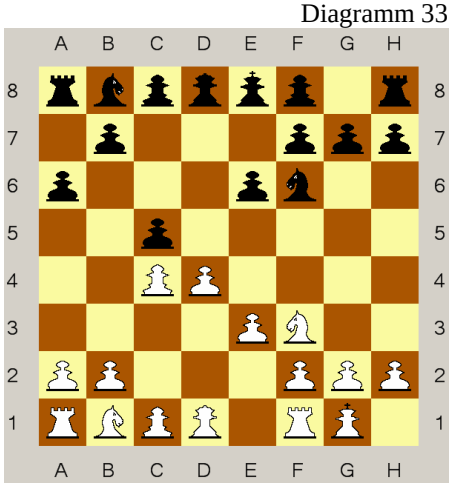


Diagramm 34

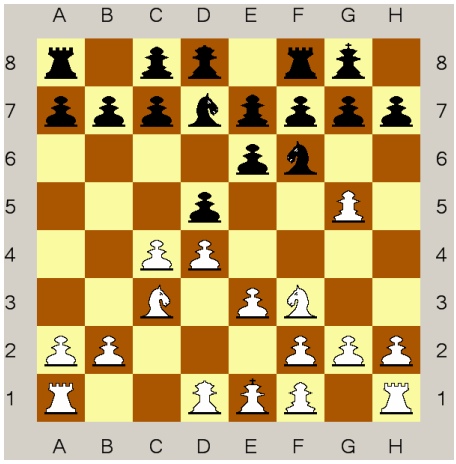


Diagramm 35

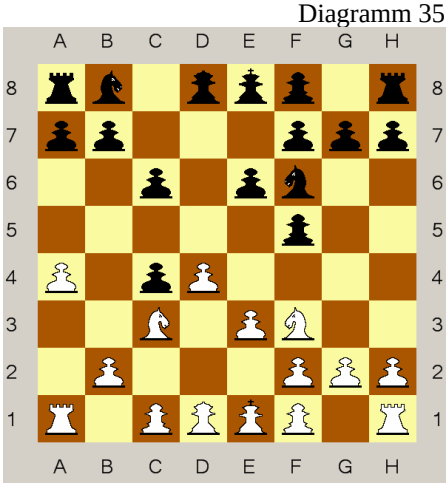


Diagramm 36

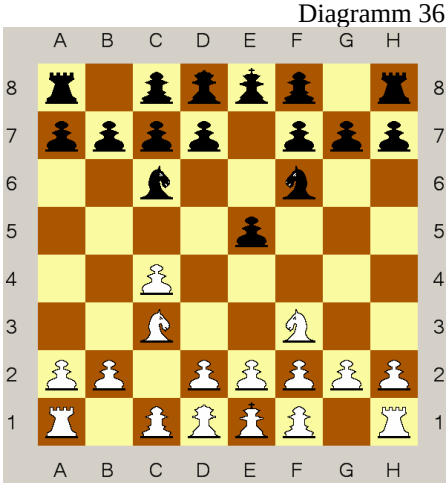


Diagramm 37

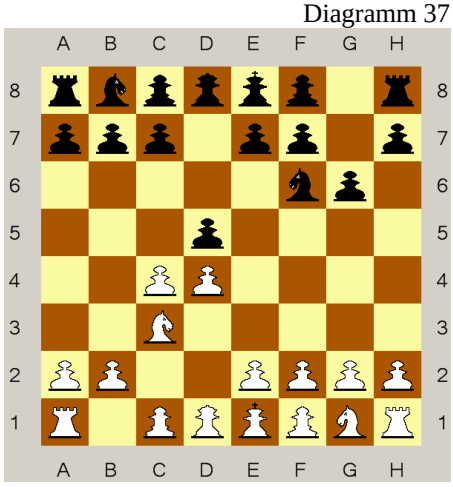


Diagramm 38

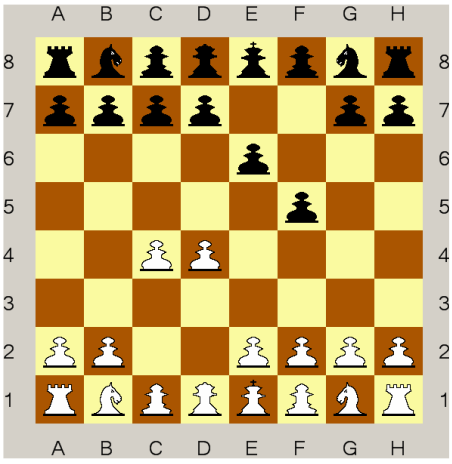


Diagramm 39

Legende für alle Diagramme

grüne Felder = Ausgangsfeld der Figur
rote Felder = bedrohte Felder
blaue Felder = mögliche Zielfelder
blaue Pfeile = Zugweise der Figur

CD – Inhalt

Diese Übersicht ist dazu gedacht, einen kurzen Überblick über die Inhalte der beigelegten CD zu geben. Bitte beachten sie, dass ’Laufwerk:\’ durch den Laufwerksbuchstaben ihres CD/DVD-ROM Laufwerks ersetzt werden muss (Standardmäßig ist dies ’D:\’), damit der Zugriff auf die jeweiligen Inhalte möglich wird.

Quellcodes

„Laufwerk:\Dokumentation\Quellcodes\“

Diagramme Kapitel II

„Laufwerk:\Dokumentation\KapIIDia\“

Diagramme Kapitel III

„Laufwerk:\Dokumentation\KapIIIDia\“

Struktogramme

„Laufwerk:\Dokumentation\Struktogramme\“

Übersichtsdiagramm Programmkommunikation Kernel – Oberfläche

„Laufwerk:\Dokumentation\KomDia\“

Stellungseditor „Naeaac Edit“[18] v0.0.2.1

„Laufwerk:\NaeaacEdit\v0021\“

Seminarfacharbeit Originalfassung

„Laufwerk:\semi\org\“

Seminarfacharbeit elektronische Fassung

„Laufwerk:\semi\elek\“

Schachprogramm „Naeaac Echecs“[19] v4.0.0.0

„Laufwerk:\NaeaacEchecs\v4000\“

Schachprogramm Betaversion v0.5.0.0

„Laufwerk:\NaeaacEchecs\v0500\“

Adobe Acrobat Reader

„Laufwerk:\sonst\Adobe\“

.Net Framework

„Laufwerk:\sonst\Framwork\“

Hinweis: Bitte lesen Sie zuerst die Datei „Laufwerk:\LiesMich.txt“ auf der CD. Dort werden alle Fragen zur Handhabung der CD beantwortet.

[18] © by Enrico Gebert, Manuel Maier & Daniel Schumann
[19] © by Enrico Gebert, Manuel Maier & Daniel Schumann

Bildnachweis

Kapitel IV:
- Struktogramme
  - Programmoberfläche: Erstellt von Enrico Gebert
  - Kernel – DLL: Erstellt von Enrico Gebert
- Diagramm Bewertungsbaum
  - http://www.matheboard.de/lexikon/index.php/Bild:040125_minmax.png

Kapitel V:
- Abbildung 1: Screenshot[20], erstellt von Enrico Gebert [Programmversion 0.5.0.0]
- Abbildung 2: Screenshot, erstellt von Enrico Gebert [Programmversion 1.0.0.0]
- Abbildung 3: Screenshot, erstellt von Enrico Gebert [Programmversion 3.0.1.0]

Diagramme 1 – 10:
- Erstellt von Daniel Schumann mit selbst erstelltem Stellungseditor.

Diagramme 11 – 51:
- Erstellt von Daniel Schumann mit erweiterter Version des selbst erstellten Stellungseditors.

Struktogramme:
- Statusbar: Erstellt von Enrico Gebert.
- Bild laden: Erstellt von Enrico Gebert.
- Maussteuerung: Erstellt von Enrico Gebert.
- Feld zeichnen: Erstellt von Enrico Gebert.
- Oberfläche – Setzen: Erstellt von Enrico Gebert.
- Berechnung starten: Erstellt von Enrico Gebert.
- Zug berechnen: Erstellt von Manuel Maier.
- Kernel – Setzen: Erstellt von Manuel Maier.
- Bewertung: Erstellt von Enrico Gebert und Manuel Maier.

Diagramm – Programmkommunikation:
- Erstellt von Enrico Gebert und Manuel Maier.

[20] Bildschirmaufnahme

Quellenverzeichnis

i Petzold,Joachim:SCHACH Eine Kulturgeschichte.Edition Leipzig 1986.S.9-16

ii Petzold,Joachim:SCHACH Eine Kulturgeschichte.Edition Leipzig 1986.S.17-40

iii Petzold,Joachim:SCHACH Eine Kulturgeschichte.Edition Leipzig 1986.S.41-54

iv Petzold,Joachim:SCHACH Eine Kulturgeschichte.Edition Leipzig 1986.S.200-203

v http://www.schachcomputer.at/gesch3.htm

vi http://www.schachcomputer.at/gesch13.htm

vii http://www.heise.de/newsticker/meldung/1021

viii Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7.Auflage:Sportverlag Berlin 1979.S.15-23

ix Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7. Auflage:Sportverlag Berlin 1979.S.25-27

x Williams,Gareth:Das erstaunliche Schachspiel,spielend lernen;Tips und Tricks;mit spannenden Beispielen.Bd.26.Hrsg.von IGM Viktor Kortschnoi,IGM Helmut Pfleger und EGM Rudolf Teschner.Hombrechtikon/Zürich:Edition Olms 1996.S.52-53

xi Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7Auflage:Sportverlag Berlin 1979.S.14-15

xii Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7Auflage:Sportverlag Berlin 1979.S.86-91

xiii Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7Auflage:Sportverlag Berlin 1979.S.92-96

xiv Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7Auflage:Sportverlag Berlin 1979.S.96-98

xv Williams,Gareth:Das erstaunliche Schachspiel,spielend lernen;Tips und Tricks;mit spannenden Beispielen.Bd.26.Hrsg.von IGM Viktor Kortschnoi,IGM Helmut Pfleger und EGM Rudolf Teschner.Hombrechtikon/Zürich:Edition Olms 1996.S.74-87

xvi Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7Auflage:Sportverlag Berlin 1979.S.28-38

xvii Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7Auflage:Sportverlag Berlin 1979.S.48-53

xviii http://www.matheboard.de/lexikon/index.php/Brute-Force-Methode

xix http://www.dietmar-schmitt.de/essays/Spiele/Spiele.htm

xx http://www.gerhard-lutz.de/al_beta.htm

Literaturverzeichnis

Primärliteratur

Awerbach,Juri;Beilin,Michail:ABC des Schachspiels,Ein Lehrbuch für die Anfängerausbildung.7Auflage:Sportverlag Berlin 1979.

Petzold,Joachim:SCHACH Eine Kulturgeschichte.Edition Leipzig 1986.

Williams,Gareth:Das erstaunliche Schachspiel,spielend lernen;Tips und Tricks;mit spannenden Beispielen.Bd.26.Hrsg.von IGM Viktor Kortschnoi,IGM Helmut Pfleger und EGM Rudolf Teschner.Hombrechtikon/Zürich:Edition Olms 1996.

Sekundärliteratur

Althöfer,Ingo:13 Jahre 3–Hirn,Meine Schach-Experimente mit Mensch-Maschine-Kombinationen.Jena:Prof. Dr. Ingo Althöfer 1998.

Borik,Otto (Hrsg.);Petzold,Joachim u.a.:Meyers Schachlexikon,Schachwissen für jedermann.Mannheim;Leipzig;Wien;Zürich:Meyers Lexikonverlag 1993.

Gik,Evgeni Jakovlevič:Schach und Mathematik/J.J. Gik [Aus d. Russ. von Wolfgang Hintze].Moskau:Verlag Mir;Leipzig;Jena;Berlin:Urania-Verlag 1986.

Schwachulla,Wolfram[red. Leitung]:Der Brockhaus: in einem Band.8.Auflage.Leipzig;Mannheim:Brockhaus 1998.

Vogt,Lothar;Knaak,Rainer:Königsindisch pro & [und] contra.Berlin:Sportverlag Berlin 1992.

Willms,Gerhard:C++, Das Grundlagenbuch.2.Auflage.Düsseldorf:DATA BECKER 2001.

Internetseiten

www.henkessoft3000.de
www.matheboard.de
www.dietmar-schmitt.de
www.minet.uni-jena.de
www.gerhard-lutz.de
http://de.wikipedia.org/wiki/Schachprogramm

Zusatzmaterialien

Microsoft Visual Studio .NET 2003-Dokumentation
MSDN Library

Sonstiges

Eidesstattliche Erklärung

„Ich versichere, dass ich die vorgelegte Seminarfacharbeit ohne unerlaubte Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.“
Jena, 30. Oktober 2004

Enrico Gebert

„Ich versichere, dass ich die vorgelegte Seminarfacharbeit ohne unerlaubte Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.“
Jena, 30. Oktober 2004

Daniel Schumann

„Ich versichere, dass ich die vorgelegte Seminarfacharbeit ohne unerlaubte Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.“
Jena, 30. Oktober 2004

Manuel Maier

Danksagung

Wir möchten uns hiermit bei allen Personen bedanken, die uns während der Erstellung dieser Seminarfacharbeit unterstützend zur Seite standen.

Spezieller Dank gilt unserem Betreuer Herrn Richard Brömel, Abteilungsleiter der Abteilung 4 des Staatlichen Berufsbildenden Schulzentrums Jena – Göschwitz, für die Hilfe und Hinweise zu unserer Seminarfacharbeit und die Bereitschaft zur Diskussion.

Des Weiteren bedanken wir uns bei unserer Stammkursleiterin, Frau Ina Boer, für die Konsultationen und hilfreiche Tipps zur Arbeit.