Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Informatik

Proseminar: Geschichte des Computers

WS 2013/2014

Dozenten: Julian Röder, Raúl Rojas, Till Zoppke

Schachprogrammierung

23.03.2014

Bearbeitet von:

Johannes Polster
johannes.polster@fu-berlin.de
3. Semester, Bachelor Informatik

Zusammenfassung

Schach wurde schon immer mit Intelligenz in Verbindung gebracht, deshalb ist es nicht verwunderlich, dass sich die künstliche Intelligenz Forschung ausführlich mit der Schachprogrammierung beschäftigt hat. Claude Shannon schlug 1950 zwei verschiedene Ansätze der Schachprogrammierung vor. Die Typ A, brute Force Methode und die Typ B Methode, welche versucht das menschliche Spiel zu imitieren. Die brute Force Programme haben sich durchgesetzt, da sie einfacher zu Programmieren sind und gut Schach spielen können. Inzwischen ist Schach aus dem Focus der KI Forschung verschwunden, da das Problem als gelöst gilt, seitdem Kasparow 1997 von dem Computer Deep Blue geschlagen wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Schach ist ein sehr altes Spiel. Es entstand vor über 1.500 Jahren, vermutlich in Indien. Heute ist es auf der ganzen Welt bekannt. Schach ist ein Spiel bei dem es nicht auf Glück ankommt. Um gut Schach spielen zu können, braucht man Kreativität, Phantasie, Erfahrung, Intuition und logisches Denken, alles Eigenschaften die die menschliche Intelligenz ausmachen. Sollte also eine Maschine das Schachspiel meistern, würde das bedeuten, dass diese intelligent ist? Herbert Simon, einer der einflussreichsten Sozialwissenschaftler des 20. Jahrhunderts sagte dazu: "If one could devise a successful chess machine, one would seem to have penetrated to the core of human intellectual endeavor." (Newell et al., 1958, S. 320). Deswegen ist es nicht verwunderlich, dass sich die Computer-Pioniere dem Problem widmeten eine "Schachmaschine" zu entwickeln.

Das Thema dieser Hausarbeit ist die Schachprogrammierung. Sie wurde im Rahmen des Seminars "Geschichte des Computers" geschrieben. In dieser Hausarbeit werde ich zuerst beschreiben, warum Schachcomputer so gründlich erforscht wurden, danach werde ich einen kurzen Überblick über die Geschichte der Schachprogrammierung geben und einen kurzen Exkurs zum Match Deep Blue gegen Kasparov halten. Als nächstes gehe ich auf die Funktionsweise von Schachprogrammen ein. Zum Schluss schreibe ich warum sich Minimax basierte Schachprogramme durchgesetzt haben und gebe dann noch ein Fazit.

2 Schach als Drosophila der KI

Als der Russische Mathematiker Alexander Kronrod sich 1965 am Soviet Institute of Theoretical and Experimental Physics, rechtfertigen musste, warum er so viel der damals teuren Computerzeit dafür verwendete um Schach zu spielen, verteidigte er sich indem er behauptete, Schach sei die Drosophila der künstlichen Intelligenz (Aleksander, 2001). Damit prägte er einen Begriff der immer wieder verwendet wurde wenn es um Computerschach ging. Die Drosophila Melanogaster ist eine Fruchtfliege, die eine sehr wichtige Rolle als Versuchsobjekt in der Genetik spielt. Sie gilt als das am besten erforschte Lebewesen auf der Welt. Wegen ihres einfaches Aufbaus konnte man die Fliege gründlich erforschen und die gewonnenen Erkenntnisse auf komplexere Systeme übertragen.

Wie die Fruchtfliege ist auch Schach einfach aufgebaut, es basiert auf wenige einfache Regeln, diese münden aber in ein komplexes Spiel. Kronrod, und andere KI Forscher seiner Zeit, hofften dass man durch das Erforschen von Schach fundamentale Einsichten in das menschliche Denken gewinnen könnte.

Schach eignete sich zusätzlich als Versuchsobjekt, weil sich die Regeln mathematisch beschreiben

lassen und da Schach ein sehr beliebtes Spiel war, konnten die Wissenschaftler auf eine schon vorhandene Infrastruktur zugreifen: Es gab Aufzeichnungen von vergangenen Partien und von Problemstellungen, die schon in maschinenlesbarer Form vorlagen. Außerdem gab es eine große Schachcomunity, mit Turnieren, welche die Forscher als Bühne nutzen konnten um ihre Fortschritte der Öffentlichkeit zu präsentieren. Mit der Elozahl gab es später noch die Möglichkeit zu messen wie gut ein Schachprogramm war, also sozusagen ein Maß für die Intelligenz eines Programms.

KI Forscher hatten also guten Grund sich ausführlich mit dem Thema Schach zu beschäftigen.

3 Die Geschichte der Schachprogrammierung

Schon lange beschäftigten sich die Menschen mit der Idee Schach spielender Maschinen. 1769 wurde der Schachtürke, ein Automat welcher sehr gut Schach spielen konnte, von Wolfgang von Kempelen erfunden. Er wurde bis 1836 in ganz Europa ausgestellt und erregte viel Aufsehen. Leider konnte der Automat nicht selber spielen, sondern wurde von einem, heimlich in der Maschine verstecktem, Menschen bedient (Jay, 2000). 1890 wurde eine elektromechanische Maschine erfunden, die Turm und König gegen König Endspiele lösen konnte. Da es sehr schwer ist nur auf mechanischer Ebene Schachmaschinen zu bauen dauerte es bis zu der Erfindung des Computers bis die ersten Schachprogramme erfunden wurden. Während des zweiten Weltkrieges schrieb Konrad Zuse das erste Schachprogramm, um den Funktionsumfang seiner neuen Programmiersprache, dem Plankalkül, zu demonstrieren. Die Sprache wurde jedoch erst in den 70er Jahren implementiert.

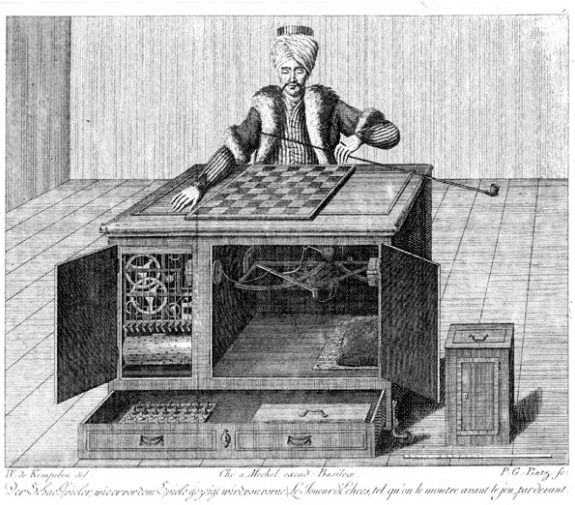

Abb. 1: Zeitgenössischer Kupferstich des von Wolfgang von Kempelen konstruierten „Schachtürken“. (Karl Gottlieb von Windisch, 1783)

1950 veröffentlichte Claude Shannon ein Papier, dass sehr wichtig für die Schachprogrammierung war (Claude Shannon, 1950). Darin beschrieb er wie man Schachprogramme aufbauen könnte, zum Beispiel wie man den Minimax Algorithmus in Schachprogrammen anwenden könnte. Die meisten Schachprogramme wurden nach dem vorgeschlagenen Prinzip entworfen.

1953 Schrieb Alan Turing ein Schachprogramm, welches man zu der Zeit jedoch nur auf Papier ausführen konnte (Nathan Ensmenger, 2012). Aber schon früher beschäftigte er sich mit den Möglichkeiten von Schachcomputern für die Künstliche Intelligenz. Herbert Simon wettete 1957, dass in

den nächsten 10 Jahren die besten Schachspieler regelmäßig von Computern geschlagen werden würden. Wie sich jedoch herausstellte, war das eine sehr optimistische Wette und er verlor sie. Die Wette zeigt, wie euphorische die KI Forscher zu der Zeit waren und wie sehr man das Problem der künstlichen Intelligenz unterschätzt hatte. Inzwischen gab es einige Programme die leidlich Schach spielen konnten. So wurde 1966 das Programm Mac Hack vorgestellt welches vom MIT entworfen wurde und offiziell an Turniere teilnehmen durfte. Es gewann sogar einige Spiele gegen relativ gute Spieler.

11 Jahre nach der Wette von Simon, wettete der International Master (IM) Levy mit einigen Informatikern für 1250 Pfund, dass es in den kommenden 10 Jahren kein Schachprogramm geben werde, welches ihn schlagen kann (Levy, 1980). Er gewann die Wette und verlängerte sie um weitere 10 Jahre. Diesmal sollte er verlieren, da er 1989 gegen den Schachcomputer Deep Thought spielte und chancenlos 4:0 verlor.

1976 wurde das Programm Microchess vorgestellt; es konnte relativ gut Schach spielen, obwohl es nur 1 Kilobyte groß war. Dadurch konnte es auf Personal Computer wie dem Apple 2 oder Commodore PET laufen und wurde so zur ersten Software die über 10,000 mal verkauft wurde.

1996 war es dann soweit und der amtierende Schachweltmeister Garri Kasparow wurde von IBM Deep Blue geschlagen. Heute haben die besten Schachprogramme eine von Menschen unerreichte Elozahl von über 3000.

4 Kasparow vs IBM Deep Blue

Garri Kasparow, der von 1985 bis 2000 amtierende Weltmeister und bis zu seinem Rückzug 2005 der am höchsten gerankte Schachspieler, hatte viel Erfahrung mit Spielen gegen Schachcomputer. 1980 behauptete er, nie von einem Schachprogramm geschlagen zu werden. 1996 spielte er ein Match über 6 Partien gegen Deep Blue, ein von IBM entwickelter Supercomputer, der extra darauf spezialisiert war gegen Kasparow zu spielen. Das Match wurde Medienwirksam vermarktet und erregt viel Aufsehen, auch außerhalb der Fachwelt. Das Kasparow das erste, spannende Spiel des Matches verlor, war eine Sensation. Zum ersten mal wurde ein amtierender Weltmeister von einem Computer geschlagen. Trotzdem konnte er noch die Ehre der Menschheit retten, da er das Match drehte und am Ende 4:2 gewann.

Abb. 2: Der Spiegel, Ausgabe 18 von 1997 ("DER SPIEGEL 18/1997," 1997)

Danach rüstete IBM Deep Blue weiter auf, sodass er bis zu 200 millionen Stellungen pro Sekunde berechnen konnte. 1997 wurde dann, unter großen Medienrummel, ein zweites Match ausgetragen, welches Kasparow 2.5 zu 3.5 verlor, sodass man endgültig sagen konnte, dass Computer besser Schach spielen könne als Menschen. Danach behauptete Kasparow, er habe Züge in dem Spiel von Deep Blue erkannt, zu denen kein Computer in der Lage wäre und das deswegen Menschen ihre Hände im Spiel gehabt haben müssten. Deswegen verlangte er ein Rematch. IBM, des Lügens bezichtigt, lehnte jedoch ab und zerlegte Deep Blue.

5 Schachprogrammierung

Es gibt nur eine endliche Anzahl an möglichen Schachpartien, da wegen der 50 Zug Regel eine Partie nicht unendlich dauern kann. Die 50 Zug Regel besagt, dass eine Partie unentschieden endet wenn innerhalb von 50 Zügen keine Figur geschlagen wird. Man könnte also theoretisch einen kompletten Spielbaum aufbauen, somit wäre das Problem gelöst, da man bei jedem Zug schauen kann welcher Zug zum Sieg führt.

Betrachtet man aber ein Spiel mit der durchschnittlichen Spiellänge von 42 Zügen pro Spieler, also insgesamt 84 Züge. Im Durchschnitt gibt es 38 mögliche legale Züge pro Zug. Dann müssten, um einen vollständigen Spielbaum aufzubauen, also 38^{84} Züge ausgewertet werden, welches ungefähr 10^{143} entspricht. Zum Vergleich: es gibt wahrscheinlich 10^{75} Atome im Universum. Es ist also ziemlich unmöglich einen kompletten Spielbaum aufzubauen. (Nathan Ensmenger, 2012)

Claude Shannon hat in seinem Paper von 1950 (Claude Shannon, 1950) zwei Lösungen für dieses Problem präsentiert, die er Typ A und Typ B Lösung nannte. Die Typ A Lösung besteht darin, den Baum nicht vollständig aufzubauen, sondern nur bis zu einer bestimmten Tiefe. Da dann in den meisten Blättern keine eindeutige Lösung steht (Sieg, Niederlage oder unentschieden) muss auf den Blättern eine Bewertungsfunktion ausgeführt werden, welche eine gegebene Stellung anhand von Heuristiken bewertet. Diese Lösung wird auch brute Force Methode genannt, da sie versucht mit purer Rechengewalt das Problem zu lösen.

Bei der Typ B Lösung, versucht man den Suchbaum zu verkleinern, indem nur ganz bestimmte Züge tiefer betrachtet werden. Diese Lösung orientiert sich an das menschliche Denken. Wenn wir Schach spielen, überlegen wir uns anhand von Patternmatching, welche Züge am vielversprechendsten sind, diese kalkulieren wir dann vielleicht 2 bis 4 Züge voraus. Die brute Force Methode hat sich gegenüber der Typ B Lösung durchgesetzt, da sie einfach viel effektiver und einfacher zu implementieren war, auch wenn die Typ B Lösung wahrscheinlich mehr Erkenntnisse für die KI Forschung gebracht hätte.

5.1 Aufbau von Schachprogrammen

In diesem Abschnitt gebe ich einen kurzen Überblick über den Aufbau von gängigen Schachprogrammen. Dabei werde ich nicht sehr in die Details gehen, Ziel dieses Abschnitts ist, sich grob vorstellen zu können, wie ein Schachprogramm vielleicht funktionieren könnte. Schachprogramme bestehen im wesentlichen aus drei verschiedenen Komponenten: dem Zuggenerator, der Suche und der Bewertungsfunktion.

5.1.1 Zuggenerator

Der Zuggenerator berechnet welche Züge eine Figur legal ausführen darf. Dafür braucht er eine Repräsentation des Bretts, um zu wissen wo die verschiedenen Figuren stehen. Es gibt verschiedene Datenstrukturen die dafür erfunden wurden. Ich werde hier näher auf die Bitboards eingehen (Hyatt, n.d.).

Ein Schachbrett ist 8 x 8 Felder groß, wenn man jede Reihe des Bretts aneinanderhängt hätte man ein 1 x 64 Brett und kann dieses als 64 Bit binäre Zahl interpretieren. Jede Stelle der Zahl repräsentiert also ein Feld. Wenn sich auf einem Feld eine bestimmte Figur befindet, steht an der Stelle der Zahl eine 1, ansonsten eine 0. Diese Zahl nennt man Bitboard. Für jede Figurenart hat man eine solche Zahl, also für die weißen Bauern, für die schwarzen Bauern usw. und kann diese dann mit boolesche Operationen verknüpfen. Will man zum Beispiel die Position aller weißen Figuren wissen, verknüpft man deren Bitboards mit "oder" und erhält so ein Bitboard mit einer 1 überall da, wo eine weiße Figur steht.

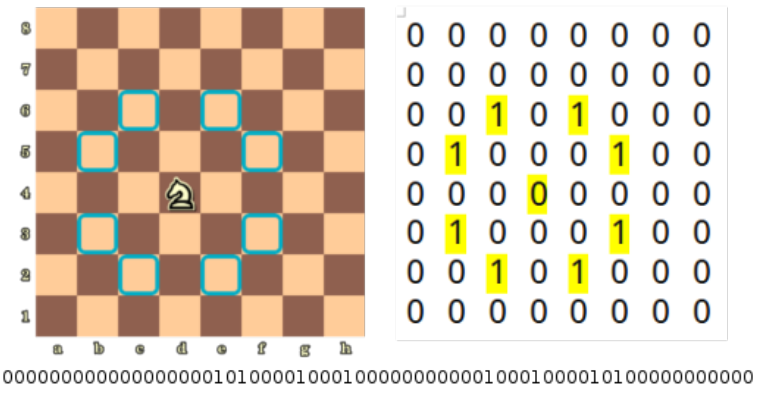

Abb. 3: Links: Mögliche Zielfelder des Springers auf d4. Rechts: Dazugehöriges Bitboard. Unten: Repräsentation des Bitboards als Zahl

So kann man auch die möglichen Züge einer Figur berechnen. Bei einem Springer kann man zum Beispiel dazu zu allen 64 Positionen, ein Bitboard speichern, welches die möglichen Zielfelder ausgibt (vgl. Abbildung 3). Der Springer darf nicht dahin springen wo eine weiße Figur steht, deshalb nimmt man das Bitboard mit allen weißen Figuren und "verneint" dieses. Jetzt hat man ein Bitboard bei dem überall eine 1 steht, wo keine weiße Figur ist. Jetzt muss man nur noch dieses Bitboard mit dem Bitboard welches die Möglichen Zielfelder des Springers spei-

chert mit "und" verknüpfen und schon hat man ein Bitboard auf dem überall eine 1 steht wo der Springer hinspringen darf.

Bei gleitenden Figuren ist das Generieren der Züge schwieriger, da hier keine eigenen Figuren im Weg stehen dürfen. Bitboards werden verwendet da sie auf 64 Bit Architekturen sehr effizient sind, da hier 64 Bit Register benutzt werden.

5.1.2 Bewertungsfunktion

Mit der Bewertungsfunktion versucht man zu berechnen wie gut eine gegebene Stellung ist. Dazu weist man der Stellung eine Zahl zu die von verschiedenen Faktoren abhängt. Jede Figur auf dem Brett hat einen Wert (vgl. Tabelle 1). Bei Schachprogrammen wird normalerweise etwas genauer differenziert und zusätzlich noch die Position einer Figur mit in die Bewertung einbezogen. So wird der Wert erhöht oder verringert je nachdem wie viel Bewegungsspielraum eine Figur hat. Diese Werte werden dann aufsummiert. Man kann zusätzlich noch einige andere Faktoren mit bewerten, zum Beispiel wie sicher der König ist oder wie viele Figuren durch Bauern gedeckt sind usw. Es gibt sehr aufwendige Bewertungsfunktionen, allerdings ist Geschwindigkeit bei brute Force Schachprogrammen sehr wichtig, sodass man oft einen Kompromiss finden muss. Am Ende hat man also einen Wert für die weiße und einen Wert für die schwarze Stellung. Diese beide Werte werden von einander abgezogen. Ist das Ergebnis positiv ergibt sich ein Vorteil für weiß, ist das Ergebnis negativ, ergibt sich ein Vorteil für schwarz. Die Bewertungsfunktion ist der einzige Teil eines konventionellen Schachprogramms bei dem der Programmierer etwas tiefere Schachkenntnisse haben sollte.

Bauer	1
Pferd	3
Läufer	3
Turm	5
Dame	9
König	Unendlich

Tabelle 1: Werte der Figuren in Bauerneinheiten

5.1.3 Suche

Ziel der Suche ist es, unter der Annahme, dass der Gegner auch für sich immer den vorteilhaftesten Zug wählt, den besten Zug für den Spieler welcher am Zug ist zu finden. Dafür wird der Minimax Algorithmus verwendet. Dazu muss zuerst der Suchbaum, mit Hilfe des Zuggenerators, bis zu einer gewissen Tiefe aufgebaut werden. Die Baumtiefe hängt von den verfügbaren Hardwareressourcen ab. Hier kann es zu dem sogenannte Horizonteffekt kommen. Der Horizonteffekt tritt auf, wenn der Baum gerade so nicht tief genug ist und deswegen wichtige Sachen übersehen werden (vgl. Abbildung 4) (Nathan Ensmenger, 2012).

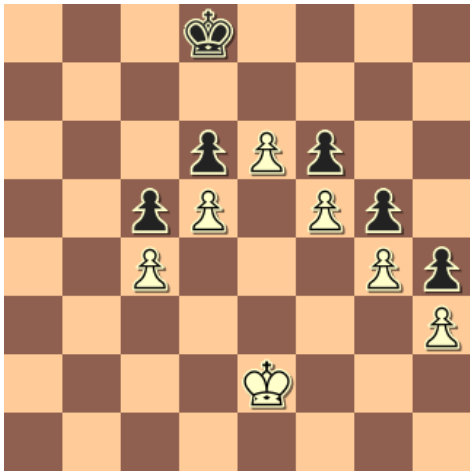

Abb. 4: Beispiel zum Horizonteffekt: Wird der Baum nicht tief genug aufgebaut, wird nicht erkannt das weiß eigentlich gewonnen hat (Problemstellung aus Michael Brudno, "Competitions, Controversies, and Computer Chess." (2006))

Am Anfang und am Ende einer Partie wird nicht der Minimax Algorithmus angewendet, sondern die besten Züge aus einer Datenbank ausgewählt, da dies effizienter ist.

Der Minimax Algorithmus weist im Suchbaum den Elternknoten jeweils das Minimum bzw. das Maximum der Kinderknoten zu, je nachdem wer bei dem Elternknoten am Zug ist. Ist weiß am Zug wird maximiert, bei schwarz wird minimiert. Die Werte in den Knoten kommen von den Bewertungsfunktion, welche auf den Blättern ausgeführt wurde. Auf allen Ebenen darüber füllt der Minimax Algorithmus die Werte aus. Ist der komplette Baum ausgefüllt, weiß der Spieler der am Zug ist, wie er ziehen muss um das für ihn beste Ergebnis zu erzielen (vgl. Abb. 5).

Meistens wird zusätzlich zum Minimax Algorithmus noch der Alpha Beta Algorithmus eingesetzt. Dieser verhindert, dass Zweige des Suchbaumes vom Minimax Algorithmus verfolgt werden, die auf keinen Fall von einem Spieler verfolgt werden würden. Dadurch wird der Baum erheblich kleiner und die ganze Suche somit schneller.

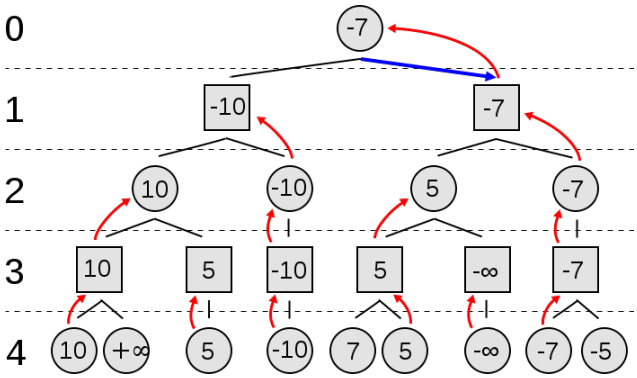

Abb. 5: Beispiel eines stark vereinfachten Suchbaumes. Weiß ist bei Ebene 0 und 2 am Zug und maximiert, schwarz minimiert bei Ebene 1 und 3. (Nuno Nogueira (Nmnogueira), 2006)

6 Vorteile von Minimax-Schachprogrammen

Brute Force Programme haben sich durchgesetzt, da sie einige Vorteile bieten; zum einen sind Minimax-Schachprogramme relativ einfach umzusetzen und der Programmierer braucht keine tiefe Schachkenntnisse (nur für die Bewertungsfunktion). Manche behaupten, je weniger der Programmierer über Schach weiß desto besser, da sich Leute die sich mit Schach auskennen oft verkünsteln und somit langsame Programme schreiben (Nathan Ensmenger, 2012).

Zum anderen skaliert die Leistung des Minimax Algorithmus linear mit der Leistung der CPU. Das hatte zur Folge, dass selbst wenn die KI Forscher nichts an den Programmen geändert haben, sie jedes Jahr aufs neue Erfolge feiern konnten, da Prozessoren kontinuierlich schneller werden. Außer-

dem sind die Minimax-Programme leicht erweiterbar, so kann man zum Beispiel einfach die Bewertungsfunktion austauschen. Der wichtigste Grund, warum sich brute Force Programme durchgesetzt haben ist jedoch, dass diese gut Schach spielen können.

7 Fazit

Schachprogramme können inzwischen sogar die besten Spieler schlagen. Also könnte man meinen, dass die Schachprogrammierung aus Sicht der KI Forscher ein voller Erfolg ist. Dies ist leider nicht der Fall, denn obwohl man das Problem als gelöst betrachten kann, hat die Schachprogrammierung die KI Forschung nicht wirklich weitergebracht. Die Hoffnung, dass Schachprogramme Intelligent sind, weil sie Schach spielen können, hat sich also nicht erfüllt, ebenso wenig die Hoffnung, dass Schach die Drosophila der Künstlichen Intelligenz werden könnte. Das Problem ist, dass brute Force Schachprogramme nicht wie Menschen spielen, und somit nicht geeignet sind, menschliche Intelligenz zu simulieren.

Schachprogrammierung ist deswegen aus dem Fokus der künstlichen Intelligenz Forschung verschwunden. Stattdessen erforscht man Spiele wie Go, bei dem es unmöglich ist, das Spiel mit brute Force zu Lösen, da es pro Zug einfach zu viele Möglichkeiten gibt. Ein anderes vielversprechendes Spiel ist Jeopordy, da man hier Erkenntnisse für das Verstehen der natürlichen Sprache gewinnen kann. Während man bei Jeopordy innerhalb vergleichsweise kurzer Zeit mit IBM Watson die stärksten Spieler schlagen und auch die gewonnen Erkenntnisse auf andere Gebiete übertragen konnte, hat es bei Schach sehr lange gedauert bis man so weit war. Schon in den 60er Jahren gab es Schachprogramme die relativ gute Spieler schlagen konnte, aber es dauerte noch über 30 Jahre bis der Beste Mensch geschlagen wurde. Und das nicht weil das Programm besonders Intelligent war, sondern nur, weil es sehr schnell rechnen konnte.

Quellenverzeichnis

Aleksander, I., 2001. How to build a mind: toward machines with imagination, Maps of the mind. Columbia University Press, New York.

Claude Shannon, 1950. Programming a computer for playing chess. Philosophical Magazine 41, 256–75.

DER SPIEGEL 18/1997 [WWW Document], n.d. URL http://www.spiegel.de/spiegel/print/index-1997-18.html (accessed 3.23.14).

Hyatt, P.R., n.d. bitmaps [WWW Document]. URL http://www.cis.uab.edu/info/faculty/hyatt/bitmaps.html (accessed 3.23.14).

Jay, R., 2000. The Automaton Chess Player, the Invisible Girl, and the Telephone. Jay's Journal of Anomalies 4.

Karl Gottlieb von Windisch, 1783. Kupferstich aus dem Buch „Briefe über den Schachspieler des Hrn. von Kempelen, nebst drei Kupferstichen die diese berühmte Maschine vorstellen." URL http://upload.wikimedia.org/wikipedia/commons/8/8b/Tuerkischer_schachspieler_windisch4.jpg

Levy, D.N.L., 1980. More chess and computers: the microcomputer revolution, the challenge match, Computer chess series. Computer Science Press, Potomac, Md.

Nathan Ensmenger, 2012. Is chess the drosophila of artificial intelligence? A social history of an algorithm. Social Studies of Science 42, 5–30.

Newell, A., Shaw, J.C., Simon, H.A., 1958. Chess-Playing Programs, and the Problem of Complexity. IBM J Research and Development 2, 320–335.

Nuno Nogueira (Nmnogueira), 2006. License: Creative Commons Attribution-Share Alike 2.5 Generic license. (http://creativecommons.org/licenses/by-sa/2.5/deed.en) URL http://upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Minimax.svg/701px-Minimax.svg.png