HU Berlin, Institut für Musikwissenschaft und Medienwissenschaft

Silvio Lippert, silvio.lippert@gmail.com
Einschreibenummer: 568103

Projektbericht

Wintersemester 2015/16

Schachcomputer und Computerschach

Projektbetreuung: Nikita Braguinski und Dr. Stefan Höltgen

Einleitung **2**
Dekonstruktivistisches Verstehen 3
Prinzipien des Schach 4
Computerschach 5
Konzept / Idee **7**
Fiktive Geschichte 9
Software 10
Listings 10
Hardware 12
Umsetzung **12**
code abtippen 13
code testen 15
Emulator 15
code verstehen 17
code manipulieren 19
Zusammenfassung **19**

"Dieses schreckliche Gefühl, dass ich gegen eine Maschine spielte, die niemals Fehler macht, zerschmetterte meinen Widerstand." Mark Tajmanov[1]

Einleitung

Idee des Projektes, Ziele und Konzept
Aspekte von Komplexität; Schachprogramme (Suchbaum-Verfahren, Pruning)

Dem Projekt *Schachcomputer und Computerschach* begleitend, fand das Seminar: *Schach dem Computer. Theorie, Geschichte und Implementierung des Computerschach*[2] statt, welches theoretische Grundlagen zur Geschichte, Theorie und Praxis des Computerschachs vertiefend behandelte: Computerschach als Forschungsfeld, auf dem Betrachtungen verschiedenster Aspekt der Mensch-Maschine-Interaktion, des Verhältnisses zwischen Mensch und Maschine selbst und der Entwicklung von Schachprogrammen angesiedelt werden können.

Seit sich Maschinen als Geistesleistung des Menschen vergegenwärtigen und in nahezu jeden Bereich des Lebens Einzug gehalten haben, ja seit es die Maschine an sich überhaupt gibt, ist das Verhältnis, welches der Mensch zu ihr einnimmt ein Virulentes. In diesem Spannungsfeld sollte sich das Projekt zum Seminar bestmöglich verorten. Im Seminar geriet ein, für mein Empfinden sehr vielversprechender Aspekt der die Unterscheidung zwischen Mensch und Maschine veranschaulicht, an zentrale Stelle. Dies ist der Umgang der Maschine (hier Schachcomputer/Schachprogramm) mit komplexen Aufgaben. Sowohl in der Bewältigung von entsprechenden Aufgaben, als auch in der Berechnung. Soweit die erste Annahme.

Die Betrachtung der Bewältigung von Komplexität wollte ich nun im Projekt ganz speziell am Objekt des Schachspiels vornehmen. Dazu war es zunächst notwendig, sich mit dem Spiel selbst, seinen Regeln (siehe Abschnitt: Prinzipien des Schach) und im Weiteren natürlich mit Maschinen, Maschinen die Schach spielen können, sowie deren Programmierung, Algorithmen und Implementierung auseinanderzusetzen.

Der hier verwendete Maschinenbegriff umfasst ganz selbstverständlich den Computer. Die ersten Computer wurden auch als Rechenmaschinen bezeichnet; vgl. *die analytische Maschine* von Charles Babbage[3]. Babbage versuchte bereits Anfang des 19. Jahrhunderts "eine Maschine zu konstruieren, die dem Menschen die Berechnung von Logarithmentafeln abnehmen sollte. Zur Demonstration der Leistungsfähigkeit einer solchen Rechenmaschine, die prinzipiell jede Funktion berechnen könnte, entschied er sich, eine Schachmaschine zu bauen."[4] Babbage scheiterte an der Komplexität und erst Torres y Quevedo baute im Jahre 1890 die erste echte Schachspielende Maschine. "Torres baute eine Maschine, die nur einen begrenzten Bereich des Schach abdeckte: der Automat war in der Lage ein elementares Endspiel (weißer König und Turm gegen schwarzen König) immer siegreich zu bestreiten, allerdings nicht immer auf

[1] Mark Jewgenjewitsch Taimanow; * 7. Februar 1926 in Charkow; † 28. November 2016 in Sankt Petersburg; war ein russischer Pianist und Schachspieler; aus Wikipedia: https://de.wikipedia.org/wiki/Mark_Jewgenjewitsch_Taimanow; Abruf: 12.03.2017, 13:34 Uhr

[2] https://agnes.hu-berlin.de/lupo/rds?state=verpublish&status=init&vmfile=no&publishid=104661&moduleCall=webInfo&publishConfFile=webInfo&publishSubDir=veranstaltung; Abruf: 20.01.2017, 12:45 Uhr

[3] https://de.wikipedia.org/wiki/Analytical_Engine; Abruf: 23.03.2017, 20:55 Uhr

[4] Bruns, Edmund:Das Schachspiel als Phänomen der Kulturgeschichte des 19. und 20. Jahrhunderts; Schriftenreihe der Stipendiatinnen und Stipendiaten der Friedrich-Ebert-Stiftung; 2003; S.301

dem kürzesten Weg. Die Maschine existiert noch und steht in der Polytechnischen Universität von Madrid.“[5]

Die erste Maschine die dann theoretisch und praktisch vollständiges Schach spielen konnte wurde erst 1936 durch den Logiker Alan Mathison Turing mit der *Papiermaschine*[6] Realität. “Das Schachspiel diente ihm dabei als eine Metapher des Intellekts ohne Bewusstsein. Er entwickelte zu dieser Zeit ein Programm für einen Computer, der noch gar nicht existierte.”[7] Interessant und wegweisend in diesem Zusammenhang ist vor allem auch Konrad Zuse mit der Erfindung des Z1-Z4. Er beabsichtigte, dass seine Rechenmaschinen auch Schach spielen können müssen[8]. Claude E. Shannon, ebenfalls ein Pionier der Computerwissenschaft darf an dieser Stelle nicht unerwähnt bleiben. Er “erörterte etwa zeitgleich mit Turing, jedoch von ihm unabhängig, die Möglichkeiten der Schachprogrammierung“[9]

Um zumindest etwas vollständig in der historischen Entwicklung des Schachspiels mittels Maschinen zu bleiben, wäre noch eine faszinierende, bereits 1769 konstruierte und gebaute Maschine, der Schachtürke des Mechanikers Wolfgang von Kempelen zu nennen. Obgleich es sich hier nicht um ein echten Schachspielende Maschine handelte, sondern ein Mensch das Schachspiel führte. Dennoch war hier bereits der Gedanke geboren, Maschinen Schach beizubringen und sie selbstständig agieren zu lassen.

Dekonstruktivistisches Verstehen

Zum Schachtürken hielt das Seminar einen sehr spannenden Text des amerikanischen Schriftstellers Edgar Allan Poe bereit. Dieser analysierte das Geheimnis und veröffentlichte eine mögliche Lösung in seinem 1836 geschriebenen Essay „Maelzel’s chess player“. Poe war motiviert den Scharlatan Mälzel, welcher den Schachautomaten von Kempelens Sohn nach dessen Tod abkaufte, zu entlarven. In seinem Essay ging Poe sehr dezidiert vor, er verglich den Schachtürken mit der analytischen Maschine von Babbage, analysierte seine Beobachtungen und stellte Überlegungen zu Bewegungsmustern und dem zeitlichen Ablauf des Spiels an. Poe empfahl sogar einen Hack, um das falsche Spiel aufzudecken, er schlug vor, den eigenen Zug kurz bevor der Schachtürke seinen Arm bewegen würde zu korrigieren und anhand der Bewegung des Arms zu erkennen, dass ein Mensch dahinter steckt. Von Edgar Allan Poe zum Schachcomputer vergingen noch Dekaden, aber sein kritischer Blick, seine Motivation ins Innere der Maschine blicken zu wollen, begründen in meinem Verständnis auch weiterhin heutiges dekonstruktivistisches Verstehen.

Der Mensch der die Maschine erdacht und erschaffen hat, versucht ihr Wesen und ihr Wirken zu verstehen und zu ergründen, was sie in ihrem Inneren zusammenhält! In einem Vortrag[10] von 1953 stellt Heidegger erhellend “Die Frage nach der Technik” und formt beispielsweise den Technikbegriff als ein dekonstruktivistisches Verstehen. Er sagt gleich zu Beginn: “Die

[5] Der Traum vom Computerschach - Eine kleine Geschichte des Computerschachs von Eric van Reem, Januar 2003, URL: http://www.scrkuppenheim.de/coko/computerschach.htm, Abruf: 12.05.2017, 13:45Uhr.
[6] http://www.turing.org.uk/index.html; Abruf: 03.04.2017, 12:34 Uhr
[7] Bruns, Edmund: Das Schachspiel als Phänomen der Kulturgeschichte des 19. und 20. Jahrhunderts; Schriftenreihe der Stipendiatinnen und Stipendiaten der Friedrich-Ebert-Stiftung; 2003; S.306
[8] Zuse, Konrad: Der Computer — Mein Lebenswerk; Springer-Verlag; 2013; S. 33
[9] Bruns, Edmund: Das Schachspiel als Phänomen der Kulturgeschichte des 19. und 20. Jahrhunderts; Schriftenreihe der Stipendiatinnen und Stipendiaten der Friedrich-Ebert-Stiftung; 2003; S.307
[10] Heidegger, Martin: Vortrag: Die Frage nach der Technik; gehalten am 18. November 1953 im Auditorium Maximum der Technischen Hochschule München, in der Reihe «Die Künste im technischen Zeitalter»; im Druck erschienen in Band III des Jahrbuches der Akademie; R. Oldenbourg München 1954, S. 70 ff.

Technik ist nicht das gleiche wie das Wesen der Technik“ und ergänzt dies: “So ist denn auch das Wesen der Technik ganz und gar nichts Technisches.“ Das Wesen der Technik ist für ihn vor allem mit der Frage verknüpft, *was* etwas ist. Er erkennt hierauf zwei Antworten und stellt sie uns mit erstens: Technik als Mittel zum Zweck und zweitens: Technik als Tun des Menschen zur Disposition.

Wie kommunizieren wir jedoch mit Computern? Die Sprache des Menschen ist erst einmal und grundsätzlich ein Prinzip für das Verständnis von Dingen. Das Nachdenken über Formulierungen führt zum Verständnis. Die Maschine weist bestimmt Fähigkeiten auf, wir ‘programmieren’ die entsprechende Sprache.

> *“Technik ist für Heidegger eine Weise des Entbergens. Sie schafft Erkenntnis und bringt neue Ideen hervor.“* [11]

Das Entbergen, wie Heidegger es nennt, das Tun um Verborgenes sichtbar, analysierbar zu machen, ist treibende Motivation dieses Projektes. Ein Sichtbarmachen des Computerprogramms.

Prinzipien des Schach

Um Computerschach besser zu verstehen, habe ich mir im nächsten Schritt einige der Grundprinzipien des Schachspiels vergegenwärtigt und diese etwas gewichtet. Ein Blick in die Spieltheorie[12]:

> *“Spieltheorie ist zunächst eine normative Theorie, die jedem einzelnen Entscheider in einer interaktiven Entscheidungssituation aufzuzeigen versucht, wie er seine eigenen (egoistischen) Interessen in dieser Situation rationalerweise am besten verfolgen kann.”* [13]

Ein Spiel ist demnach eine Entscheidungssituation, in welcher mindestens zwei Spieler interagieren. Bei der Interessen- und Motivationslage der Spieler wird davon ausgegangen, dass diese jeweils das Spiel gewinnen wollen. Weiterhin wird von drei Grundannahmen ausgegangen, ohne die gemeinhin die meisten Spiele nicht funktionieren würden: 1. Die Spieler halten sich an Regeln und befolgen diese, 2. die Spieler verhalten sich ‘vernünftig’[14], rational und 3. strategisch, in Bezug auf ihr Anliegen das Spiel zu gewinnen.
Wenn wir uns nun das Schachspiel ansehen, haben wir es zudem mit einem zufallsfreien Zwei-Personen-Spiel mit vollkommener Information zu tun, d.h. alle Züge sind bekannt. Das gewonnene Spiel und das verlorene Spiel für den Gegner ergeben zusammen Null.

> *“Laut Spieltheorie zählt das Schachspiel zu den sogenannten ‘Nullsummenspielen’ mit ‘perfekter Information’. Als ‘Nullsummenspiele’ bezeichnet man Spiele, bei denen die*

[11] Fuchs, Christian; http://cartoon.iguw.tuwien.ac.at/christian/technsoz/heidegger.html, Abruf: 23.04.2017, 12:34 Uhr
[12] nach Johann von Neumann, [1928]: Zur Theorie der Gesellschaftsspiele, Mathematische Annalen. Erstes Werk zur sogenannten Spieltheorie, die auch interaktive Entscheidungstheorie heißen könnte.
[13] Prof. Dr. Wolfgang Leininger und PD Dr. Erwin Amann in: Einführung in die Spieltheorie, S.6; https://www.ethz.ch/content/dam/ethz/special-interest/gess/chair-of-sociology-dam/documents/education/spieltheorie/literatur/Leininger%20Amann%20Einf%C3%BChrung%200708-ST1-Vorlesung-Skript.pdf; Abruf: 24.03.2017, 14:54 Uhr
[14] vgl.: https://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/sauer/geyer/Spieltheorie.pdf; S.5; Abruf: 12.05.2017, 12:34 Uhr

> *theoretischen Gewinnchancen für beide Seiten annähernd gleich sind – also zusammen genommen Null ergeben. 'Perfekte Information' bedeutet, dass das Spiel keinerlei Zufallsfaktoren beinhaltet und somit – im Prinzip – vollständig ausrechenbar ist."*[15]

Schach, ein im Prinzip vollständig ausrechenbares Spiel, aber eben nur theoretisch.

> *"Auch wenn sich die Fachliteratur nicht einig ist, wie hoch die Zahl der möglichen Partien tatsächlich ist, so findet man allein für die ersten 40 Züge keinen Wert, der die Zahl von 10^{120} nennenswert unterschreitet. Das reicht für die Feststellung, dass es allein für die ersten 40 Züge schon prinzipiell unmöglich sein wird, alle denkbaren Partien nacheinander durchrechnen, denn für annähernd 10^{120} Möglichkeiten sind mindestens ebenso viele Rechenoperationen notwendig. Für alle Partien, die länger als 40 Züge dauern, wären es sogar noch sehr viel mehr."*[16]

Wir haben es beim Schach also mit einem Spiel zu tun, dass eine extrem hohe Anzahl möglicher Spielzüge bereithält, bei relativ einfacher Grundstruktur. Das Berechnen aller Züge ist theoretisch zu bewerkstelligen, jedoch kann dies heutige Computertechnik nicht in einem vernünftigen Zeitrahmen leisten. Da dieser Umstand bekannt ist, werden andere Strategien, welche allerdings auch eine gewisse Suchtiefe erfordern, um einigermaßen gutes Schach zu spielen, verfolgt.

Computerschach

Beim Computerschach wird ein menschlicher Spieler durch eine Maschine, durch einen Computer ersetzt. Das Regelspiel bleibt gleich, die Figuren und das Brett ebenfalls und doch ist einiges anders. Computer spielen Schach mit anderen Voraussetzungen. Sie können zunehmend schneller (auch vernetzt) rechnen (Mooresches Gesetz), die Fehlerquote dabei liegt weit unter der des Menschen und sie spielen mittlerweile sehr stark.

Doch warum spielen Computer heutzutage i.d.R. viel stärkeres Schach als Menschen, als Großmeister es können? Zunächst werden die möglichen Züge formalisiert und eine Strategie entwickelt. Wie das geht, haben bereit Shannon und Turing gezeigt. Die entsprechenden Züge werden nach einer Rangfolge berechnet, die Berechnung an bestimmten Stellen gestoppt, maximale Gewinnstrategien verfolgt und mitunter mit Datenbanken gespeicherter Spielverläufe abgeglichen. Ein Grundproblem hierbei bleibt, dass alle Felder einzeln betrachtet und schrittweise abgearbeitet werden. Der geübte menschliche Schachspieler hingegen, ist in der Lage, das Schachbrett insgesamt auf einen Blick visuell zu erfassen und sich sehr schnell auf einen Bereich des Schachfeldes zu konzentrieren, wo die Aussicht besteht, beim nächsten Zug einen Vorteil zu erzielen. Dabei werden viele Schritte, durch die sich ein Computer 'mühsam' hindurch rechnen muss, von vornherein ausgeblendet, um sich auf Wesentliches zu konzentrieren.

Beim Spiel mit einem Menschen geht man eigentlich immer, sofern eine etwas elaborierte Spielstärke besteht, davon aus, dass eine bestimmte Strategie verfolgt wird. Wenn man mit einem Computer spielt, möchte man dies auch tun, ja man ist geneigt, dem Computer etwas

[15] http://www.sfbux.de/wp-content/uploads/artikel/berechenbarkeit.pdf; Seite 1; Abruf: 24.03.2017, 14:07 Uhr
[16] http://www.sfbux.de/wp-content/uploads/artikel/berechenbarkeit.pdf; Seite 3; Abruf: 24.03.2017, 14:07 Uhr

menschliches hineinzuinterpretieren. Sobald man jedoch ein relativ altes Computerprogramm wie das BASIC-Schachprogramm von Günter O. Hamann und Jan-Jürgen Eden[17] in der ersten Spielstärke ausprobiert, wird man sehr schnell feststellen, dass dies zwar so ist, aber nicht zwingend der Strategie eines Menschen[18] entspricht. Interessant war bei mir zu beobachten, dass ich versuchte habe, eine Strategie des Computers seines Algorithmuses zu verstehen und dagegen zu spielen. d.h. bestimmte Züge konnte ich ohne Bedenken tun, da ich wusste, das Programm würde darauf nicht 'antworten' weil die zu erwartenden Werte nicht berücksichtigt würden, denn der 'Gewinn' wäre zu klein gewesen. Das lässt ein komplett anderes Spiel zu. Eine Figur zu schlagen beispielsweise, steht unter dem Wert den König ins Schach zu setzen. Das ist zumindest anders, als ich spielen würde. Material zu gewinnen, wenn der König sich wieder elegant aus der Affäre ziehen, oder jemand anderes als Schutzschild zu benutzen, schiene mir angebrachter, zumal das Schach Setzen in Folge einfacher würde.

Der Mensch unterscheidet sich von seinem Spiel her vom Computer, aber ist er als Gegenspieler des Computers ein bloßer Antagonist mit destruktiven Anleihen? Hier ist nicht auf die Absicht ein Spiel gegen den Computer zu gewinnen einzugehen, eher geht es um das Verständnis, wie der Computer mit Komplexität verfährt. Gleichzeitig bleibt die Betrachtung, wie wir als Menschen, die ja den Computer erschaffen haben, der Komplexität begegnen? Der Sprung in die Diskussion um die Entwicklung künstlicher Intelligenz ist nicht groß, zumal es bei der Entwicklung von Schachprogrammen seit jeher um genau diesen Aspekt ging. Ist das Schachspiel als ein Indikator für die Entwicklung künstlicher Intelligenz zu sehen?

Alexander Kronod, ein russischer Schachspieler, stellte noch 1965 die These auf, dass das Schachspiel die Fruchtfliege der künstlichen Intelligenz sei[19]. Auch wenn seine These heute als widerlegt gilt, warf sie doch ein eröffnendes Licht auf die bestehende Diskussion.

> *"Operation of the brain and nervous system are viewed as machine-like in nature, and therefore human thinking eventually ought to be imitated by computer analogs."*[20]

Alfred Pfeiffer von der TU Chemnitz geht davon aus, das "Schachspielen gilt als eine Intelligenzleistung, wobei 'Intelligenz' ein Attribut ist, das man nach landläufiger Meinung künstlich noch nicht nachbilden kann."[21]

Im Grundsatz, löst eine Maschine Probleme anders als Menschen. Als Beispiel kann hier die Herangehensweise der Überprüfung möglicher Züge gesehen werden. Die Maschine prüft i.d.R. jeden möglichen Zug, das ist viel Rechenarbeit. Dabei wird formalisiert und algorithmisch berechnet. D.h. eine Unmenge an Rechenoperationen müssen angestellt werden. Dies ist der Part, den Computer können! Wenn man sich einfache Berechnungen ansieht, ist diese Erkenntnis recht trivial. Maschinen sind einfach schneller und präziser. Aber

[17] BASIC-Schachprogramm schreiben mit Schneider; Deutscher Betriebswirte-Verlag GmbH; 1986 - ich verwende dieses Buch, bzw. Schachprogramm für das Projekt "CHESSODY"
[18] an dieser Stelle bleibt festzuhalten, dass natürlich alle bisherigen Computer-Schachprogramm von Menschen geschrieben wurden und insofern stets eine menschliche Strategie impliziert ist, die sich jedoch den Grenzen der verwendeten Algorithmen unterworfen ist.
[19] https://chessprogramming.wikispaces.com/Alexander+Kronrod, Abruf: 19.04.2016; 13:48 Uhr
[20] Hearst, Eliot; Man and machine: Chess achievements and chess thinking;1983;S. 169
[21] http://www.informatik.tu-chemnitz.de/~apf//Informatik-Spielend/pcschach.html, Abruf: 20.4.2016; 05:10 Uhr

genau hier ist der Sprung in der Qualität der Schachprogramme zu vermuten, die mit besserer Algorithmisierung des Spielbaum-Suchverfahren, geschickterer Umsetzung des MINI-MAX-Prinzips und Alpha-Beta-Pruning arbeiten.

Schachprogramme haben folgende grundlegende funktionale Bestandteile (Komponenten): Zuggenerator, Engine. Moderne Schachprogramme verfügen zusätzlich oft auch über eine angeschlossene Bibliothek, eine Datenbank, die als künstliches Gedächtnis fungiert. Hier sind diverse Züge, Eröffnungen, ja ganze Spiele abgespeichert und abrufbereit. Entscheidend bleibt, wie dies von den verwendeten Algorithmen eingebunden wird.

Der MINI-MAX-Algorithmus analysiert den vollständigen Suchbaum. Dabei werden aber auch Knoten betrachtet, die in das Ergebnis (die Wahl des Zweiges an der Wurzel) nicht einfließen. Die Alpha-Beta-Suche versucht, möglichst viele dieser Knoten zu ignorieren. Die Anzahl der möglichen Halbzüge steigt mit der Suchtiefe exponentiell an (kombinatorische Explosion). Wie funktioniert das? Das Alpha-Beta-Pruning genannt, ist eine optimierte Variante des MINI-MAX-Suchverfahrens, also eines Algorithmus zur Bestimmung eines optimalen Zuges bei Spielen mit zwei gegnerischen Parteien. Während der Suche werden zwei Werte Alpha und Beta aktualisiert, die angeben, welches Ergebnis die Spieler bei optimaler Spielweise erzielen können. Mit Hilfe dieser Werte kann entschieden werden, welche Teile des Suchbaumes nicht untersucht werden müssen, weil sie das Ergebnis der Problemlösung nicht beeinflussen können.

Bei dem BASIC-Schachprogramm von Hamann und Eden wird die Alpha-Beta-Methode und davon speziell nur das Alpha-Pruning genutzt (dies ist vor allem der geringen Rechenkapazität der damaligen Computer geschuldet). Die einfache (nicht optimierte) Alpha-Beta-Suche liefert exakt dasselbe Ergebnis wie die MINI-MAX-Suche.

Konzept / Idee

Beschreibung der Idee, Motivation, vorbereitenden Maßnahmen, planerische Tätigkeiten sowie Organisatorisches

Die Grundintention war zunächst, ein Projekt zu gestalten, dass Aspekte des Seminars: *Schachcomputer und Computerschach* aufgreift und in kreativer Beschäftigung damit, eine Idee entsteht, deren Betrachtung Erkenntniswerte zum programatischen Ablauf eines Computer-Schachspiels schafft. Es ging um die Idee, den Kern eines Schachprogramms, den Aufbau, Ablauf, sowie Funktionsweise zu identifizieren. Falls es gelingt, sollte überdies ein bestehendes Programm modifiziert, ja im Programmablauf selbst Dinge verändert werden, die wiederum zu anderen Ergebnisse und Zügen führen und durch das Brechen von Regeln an seine Grenzen gebracht werden. Das Programm soll zudem mittels Sichtbarmachen von Ablaufstrukturen und Modifikation besser verstanden werden[22]. Dieses zunächst explorative und zugleich experimentelle Vorgehensweise, sich einem Schachprogramm derart zu näher, lässt sich auf das Entbergen Heideggers begründen.

[22] das Medium erkennt man erst, wenn es zusammenbricht, bzw. wenn ein Fehler entsteht

Ein Ziel des Projektes war es nun, genau die Stellen aufzuzeigen, an denen der Computer aus Sicht eines Gegenspielers ‘wirklich Schach spielt’, statt nur zu rechnen, was nach den Vorbetrachtungen an sich unmöglich ist, aber eventuell ließe sich etwas finden, was entsprechend interpretiert werden kann. In einem zweiten Schritt geht es darum zu erkennen, wie komplexe Aufgaben bewältigt werden. Dies anhand einer etwas älteren Software und einem bestehenden Schachprogramm zu tun, war aufgrund meines, nur in geringem Umfang vorhandenen Vorwissens bezüglich moderner Software, ein naheliegender Ansatz. Das Schachprogramm sollte zudem auch auf Hardware seiner Zeit laufen, was medienarchäologisch interessant ist, aber auch den Vorteil hat, dass die internen Abläufe dieser Hardware, verglichen mit heutigen Rechnern wesentlich langsamer sind und, in meiner Annahme, die Chance besteht, bessere Beobachtungen durchzuführen. Meiner Meinung nach, waren das mehrere Aspekte, die sich sehr gut ergänzten, bei Verwendung des richtigen Programms und der passenden Hardware.

Wie eingangs beschrieben, wollte ich nicht nur die Funktionsweise aufzeigen und ein in die Jahre gekommenes Programm lauffähig machen, sondern um etwas Besonderes zu kreieren habe ich mir überlegt, das Ganze in eine leicht verrückte Geschichte einzubetten. Eine kleine fiktive Geschichte, der so wie ich sie mir vorstelle, ein gewisser Grad an Realisierbarkeit innewohnt, dass sie so wirklich hätte passiert sein können. Die Geschichte sollte einen Rahmen bilden, in dem das Projekt verortet und dessen Darstellung gekoppelt werden kann, d.h. in geeigneter Weise auch fassbar und greifbar werden, ähnlich einer Hands-On-Installation. Diese wiederum wollte ich nicht zu einfach konzipieren und mehrere technische Aspekt einbringen, die nicht alltäglich verknüpft werden sollten.

Sobald Wissen in Geschichten verpackt ist, erreicht es in der Regel höherer Wirkungsgrade, Geschichten bleiben hängen und transportieren Inhalte besser, als bloße statische Erklärung es können. Ich suchte also nach einem Ansatzpunkt für eine Geschichte und ließ meine Gedanken schweifen. Dabei versetzte ich mich gedanklich zurück in die 80er Jahre, in meine Kindheit und rekapitulierte, womit ich mich zu jener Zeit nachhaltig beschäftigt habe. Elektrobasteln stand hoch im Kurs, auch spielte ich gern Schach mit meinem Vater. Ab der fünften Klasse besuchte ich fakultativ eine Computer-AG und lerne dort die Programmiersprache BASIC kennen. Es war mein erster Einblick in die Welt von Software und gesteuerter Technik, in all das, was uns heut ganz selbstverständlich umgibt. Wie bei vielen anderen Kindern auch, galt mein Interesse dem Weltraum und ich verfolgte gespannt jede Berichterstattung von der Raumstation MIR und den Erkundungen des Weltraums. Geschichten von Stanislav Lem, die us-amerikanische Fernsehserie Raumschiff Enterprise und später Filme wie 2001 Odyssee im Weltraum gaben meiner Vorstellung vom All immerwährend Impulse und formten grundlegend die Annahme, dass wir nur einen kleinen Teil, von dem was uns umgibt kennen und wir offen mit neuen Annahmen umgehen müssen und Dinge auseinandernehmen und erforschen sollten.

Fiktive Geschichte

Berlin, 70er Jahre: Wissenschaftler der Humboldt Universität (HU) beschäftigen sich mit Maschinenlernen, mit der Programmierung einer Art Künstlicher Intelligenz (KI). In mehreren experimentellen Forschungsreihen wird unter anderem Computerschach zu einem zentralen Schwerpunkt. Die Kombination mehrerer Algorithmen mit einem einfachen

BASIC-Schachprogramm zeigte – aufgrund schwacher Rechner – nur auf einzelne Schachprobleme bezogen, erste Erfolge. Die Schachprobleme haben nur drei bis vier Figuren und das Schachfeld ist auf 5x5 Felder begrenzt. Man sucht nach Möglichkeiten, komplexere Berechnungen anstellen zu können, doch es gab keine Rechner die das in absehbarer Zeit zu leisten vermochten. Man suchte nach einer Möglichkeit, spezielle Parameter in die Berechnungen mit einfließen zu lassen, an die man vorher nicht gedacht hatte, auf die man nicht so einfach kommt, die sich evtl. erst aus Beobachtungen des Großen Ganzen ergeben und dann wiederum zurück auf das Kleine führten.

Nun ergab sich die Möglichkeit, an einer Weltraummission teilzunehmen. Ein Forschungssatellit sollte für eine ca. 30 bis 50 Jahre währende Mission ins All geschickt werden und die HU bekam das Angebot, ein eigenes Forschungsprojekt mitzuschicken. An Bord befanden sich mehrere Rechner, auf denen in BASIC geschriebene Programme lauffähig waren. Das damalige Forscherteam unter der Leitung von Horst Völz[23] erkannte die Chance, die 'fehlenden' Parameter aus den Ergebnissen der Mission generieren zu können und wollten das mit in ihre Berechnungen einfließen lassen und das Programm permanent modifizieren. Ihrer Annahmen zufolge, könnten sich die Erkenntnisse aus der Beobachtung des Alls nützlich erweisen, die KI zu verbessern und das Ganze am Beispiel des Schachspiels zu demonstrieren. Außerdem war ihnen bewusst, dass bei einer Mission ins Weltall, nur modernste Computertechnik, die für die HU zu diesem Zeitpunkt nicht erschwinglich war, verwendet werden wird.

Die Forscher entwickelten ein Programm, dass auf BASIC basierte und in unzähligen Iterationsschleifen seinen Code selbstständig verbessern sollte, die Ergebnisse der Erkundungen ständig mit einfließen und so ein Schachspiel spielen könne, das bislang nicht denkbar war. Die Ergebnisse sollten zur Erde gefunkt und ausgewertet werden. Soweit die Annahme. Sie nannten das Projekt CHESSODY, in Anlehnung an den 1968 erschienene Film *2001: A Space Odyssey*.

Mit der politischen Wende Ende der 80er und Anfang der 90er Jahre wurden die Forschungsgelder gestrichen und schlussendlich geriet das Projekt in Vergessenheit. Bis sich dann eines Tages im Jahre 2016 Studenten der HU an das Projekt CHESSODY erinnerten und versuchten Kontakt zu dem Satelliten, der noch immer im All unterwegs war, herzustellen. Dies gelang! Der Satellit antwortete und schickte das modifizierte Schachprogramm per Funkwellen auf die Erde zurück. Das Programm konnte empfangen werden und nun wurde versucht geeignete Hardware zu finden, um das Schachspiel lauffähig zu machen. Fündig wurde man im Signallabor der Medienwissenschaften der HU.

Um die technische Seite des Projektes zu veranschaulichen, sollte das Ganze dann als FM-Transmitter und Empfänger-Station aufgebaut werden (in der DDR wurden tatsächlich BASIC-Programme per Radio über den Äther verteilt). Ein AMSTRAD[24]/Schneider CPC aus der Hoch-Zeit der BASIC-Programmierung dient als ausführender Rechner. Besucher können das Spiel ausführen und spielen und somit eine sichtbare Verbindung der fiktiven Geschichte mit der Konzeptidee herstellen.

[23] https://de.wikipedia.org/wiki/Horst_V%C3%B6lz; Abruf: 27.05.2017, 12:34 Uhr
[24] AMSTRAD = Alan Michael Sugar Trading Colour Personal Computer

Software

Schon zu Beginn des Projektes beschäftigte ich mich mit der Frage: welche Software, in welcher Programmiersprache für die Umsetzung des Projektes sollte es werden? Bei Andre Adrian habe ich dazu eine hilfreiche Antwort, zumindest für die Programmiersprache gefunden:

> *"Auf die Frage: 'Was ist die beste Programmiersprache für eine gewisse Aufgabe' gibt es immer zwei Antworten: 'Jede' und 'Ihre Lieblingsprogrammiersprache'. Jede Turing komplette Programmiersprache hat die gleiche Mächtigkeit."*[25]

Da ich selbst nur in begrenztem Umfang Erfahrungen mit dem Programmieren gemacht habe, suchte ich nach einem relativ einfachen Schachprogramm, welches ich mit meinen bisherigen Kenntnissen verstehen und nachbauen konnte. In der 5. Klasse, hatte ich meine erste Begegnung mit Computern, bei einem Informatikkurs, welcher fakultativ neben dem Schulunterricht angeboten wurde. Dort verbrachte ich einige Zeit mit einem KC 85/3 und lerne die Programmiersprache BASIC kennen. BASIC geriet also schon in die engere Auswahl.

Listings

Es gibt mittlerweile unzählige Schachprogramme, darunter sind einfache und weniger einfache, Programme mit viel und mit wenig Quellcode, einige mit angeschlossener Bibliothek oder lediglich einer Routine, die den nächsten Zug ermittelt (Zuggenerator).

Das Listing eines BASIC-Schachprogramms, welches mir von Stefan Höltgen empfohlen stammt aus dem Buch: *BASIC-Schachprogramm schreiben mit Schneider* von Hamann und Eden. Es ist ein Listing, wie es sehr typisch für die damalige Zeit war. Der Programmcode, in der Regel auf einem Rechner erstellt und lauffähig gemacht, dann ausgedruckt und veröffentlicht mit dem Ziel beim Leser folgende Handlung zu evozieren: den Code abtippen, wieder in einer Rechner einzuspeisen und das Programm starten.

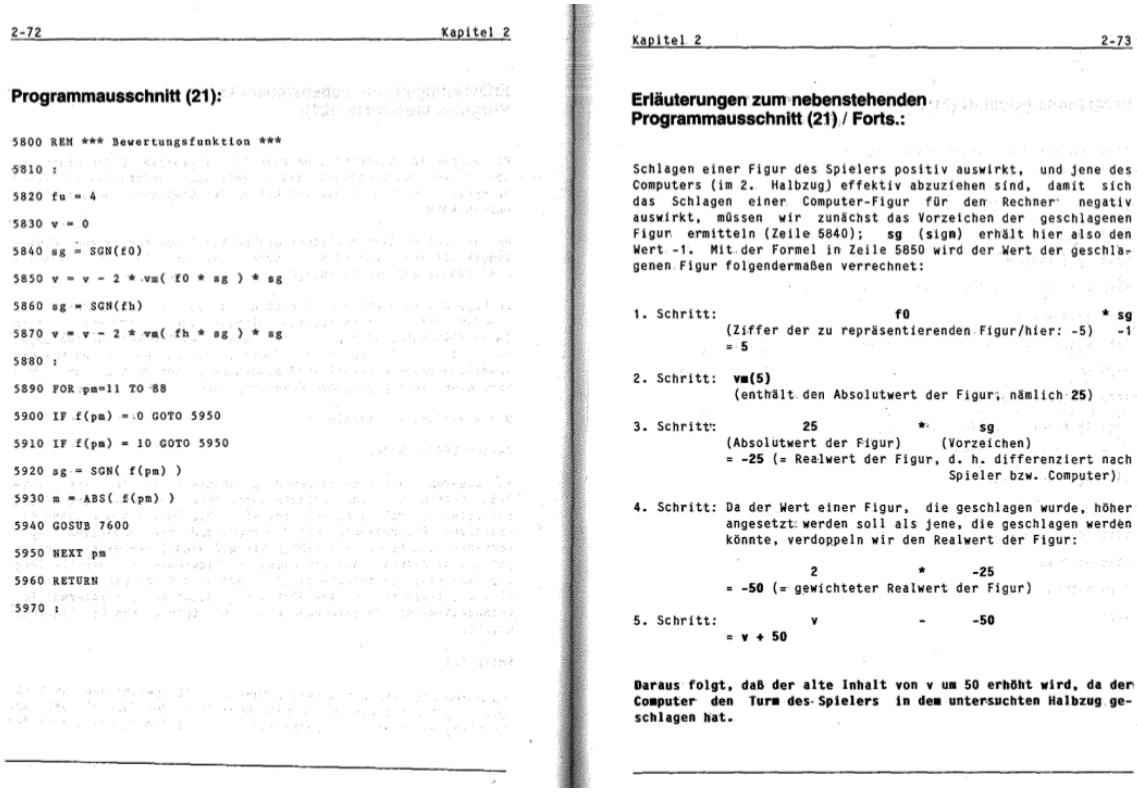
Das Transferieren der Software auf diesem Wege funktioniert gut, es überdauert auch viele Jahre, aber ist etwas umständlich. Zugegeben. Es findet ein Medienwechsel statt, der zwar nicht zwingend sein muss, um die Software zu verbreiten, denn Rechner waren auch zum damaligen Zeitpunkt schon fähig sich zu vernetzen und direkt miteinander zu kommunizieren, d.h. der Code hätte auch per Datenleitung übertragen werden können (natürlich findet hier auch ein Medienwechsel statt). Nur war das nicht zeitgemäß. Zeitgemäß waren Datasetten, Listings auf Papier (und evtl. Disketten)[26].

---

[25] http://www.andreadrian.de/schach/index.html; Absatz: Programmiersprache für Schachprogramme, Abruf: 24.03.2017, 12:53 Uhr

[26] Nicht unerwähnt sein lassen wollte ich die Speicherung von Software auf Schallplatte und die Übertragung im Rundfunk (DDR).

Die Wahl auf das Listing von Hamann und Eden fiel unter anderem wegen der sehr guten Erläuterungen der einzelnen Code-Abschnitte. Hamann und Eden lieferten ein komplett strukturiertes Schachprogramm mit erläuternden Kommentaren, welche direkt in den Code geschrieben wurden. Zusätzlich zu jedem Programmabschnitt gibt es auf der gegenüberliegenden Buchseite eine Dokumentation, die über die Kommentare im Code weit hinausgeht. Dies half das Programm, auch nach 30 Jahren gut nachvollziehen zu können.



2-72 Kapitel 2

Programmausschnitt (21):

```
5800 REM *** Bewertungsfunktion ***
5810 :
5820 fu = 4
5830 v = 0
5840 sg = SGN(f0)
5850 v = v - 2 * vm( f0 * sg ) * sg
5860 sg = SGN(fh)
5870 v = v - 2 * vm( fh * sg ) * sg
5880 :
5890 FOR pm=11 TO 88
5900 IF f(pm) = 0 GOTO 5950
5910 IF f(pm) = 10 GOTO 5950
5920 sg = SGN( f(pm) )
5930 m = ABS( f(pm) )
5940 GOSUB 7600
5950 NEXT pm
5960 RETURN
5970 :
```

Kapitel 2 2-73

Erläuterungen zum nebenstehenden Programmausschnitt (21) / Forts.:

Schlagen einer Figur des Spielers positiv auswirkt, und jene des Computers (im 2. Halbzug) effektiv abzuziehen sind, damit sich das Schlagen einer Computer-Figur für den Rechner negativ auswirkt, müssen wir zunächst das Vorzeichen der geschlagenen Figur ermitteln (Zeile 5840); **sg** (sign) erhält hier also den Wert -1. Mit der Formel in Zeile 5850 wird der Wert der geschlagenen Figur folgendermaßen verrechnet:

1. Schritt: **f0** * **sg**
(Ziffer der zu repräsentierenden Figur/hier: -5) -1
= **5**

2. Schritt: **vm(5)**
(enthält den Absolutwert der Figur, nämlich **25**)

3. Schritt: **25** * **sg**
(Absolutwert der Figur) (Vorzeichen)
= **-25** (= Realwert der Figur, d. h. differenziert nach Spieler bzw. Computer)

4. Schritt: Da der Wert einer Figur, die geschlagen wurde, höher angesetzt werden soll als jene, die geschlagen werden könnte, verdoppeln wir den Realwert der Figur:

2 * **-25**
= **-50** (= gewichteter Realwert der Figur)

5. Schritt: **v** - **-50**
= **v + 50**

Daraus folgt, daß der alte Inhalt von v um 50 erhöht wird, da der Computer den Turm des Spielers in dem untersuchten Halbzug geschlagen hat.

In heutiger Zeit drängen sich beim Abtippen einer Software sofort verschiedenste Fragen auf. Eine grundsätzliche wäre zunächst die Frage nach der Rechtmäßigkeit eines solchen Vorgehens, welches sich ohne größere Argumentationsbögen als Kopieren von Software definieren ließe. Dazu haben Hamann und Eden in ihrem Vorwort folgendes geschrieben:

> *"Der Reiz dieses Buches liegt also weniger in der Möglichkeit, die Ausgaben für einen 'spezialisierten' Schachcomputer oder ein teures Schachprogramm zu sparen. Der persönliche Gewinn ist vielmehr und vor allem darin zu sehen, daß das Schachspiel der Maschine seinen 'black-box'-Charakter verliert, d.h. der Spieler durchschaut mit der Übernahme des Programms die Strategie des Computers. Und wenn diese dem Anwender nicht behagt, kann er sie sogar verändern, weil BASIC eine Programmiersprache ist, die von fast allen Hobbyisten der Datenverarbeitung beherrscht wird."*

Damit wird klar, dass hier explizit das 'Kopieren', Verbreiten, ja gar das Verändern des Codes gewünscht wird. Wenn man so möchte, ein Vorläufer der Creativ Commons[27] Bewegung.

[27] 2001 in den USA als gemeinnützige Organisation gegründet; hält Standard-Lizenzverträge für Autoren bereit, um freie Inhalte zu schaffen

Hardware

Für die Hardware entschied ich mich für einen Rechner aus dem Bestand des Signallabors der HU, der zur Zeit der fiktiven Geschichte tatsächlich genutzt und für den das Listing des BASIC-Schachprogramms geschrieben wurde. Es ist ein Mainstream Modell, ein AMSTRAD CPC or CTM-644, µP: Zilog Z80A, 128 KB RAM, mit Locomotive-BASIC[28]

AMSTRAD baute mit dem *Colour Personal Computer* (CPC) einen Rechner, der den damals dominierenden Commodore C64 die Stirn bieten sollte und war damit recht erfolgreich. Insgesamt wurden über drei Millionen Rechner verkauft und sogar Lizenzen nach Frankreich, Spanien und Deutschland vergeben. In Deutschland wurde der AMSTRAD als Schneider CPC bekannt und beliebt.

Der im Signallabor der HU vorrätige Schneider CPC verfügt genau über die Spezifikation, die notwendig sind, um das Schachprogramm von Hamann und Eden erfolgreich laufen lassen zu können.

Umsetzung

Beschreibung der Projektdurchführung (mit Nennung besonderer Meilensteine und Schwierigkeiten, mit denen das Projekt konfrontiert wurde)

An dieser Stelle des Projektes lagen alle Voraussetzungen vor; ich hatte eine fiktive Geschichte, entsprechende Vorkenntnisse zu Computerschach und zum Schachspiel selbst, ein Listing und die notwendige Hardware. Der folgende Schritt war nur, das auf Papier gedruckte Listing des Schachprogramms wieder in eine digitale Form zu transferieren. Dieser Vorgang schien zunächst objektiv sehr einfach in seiner Realisation, barg jedoch Probleme und Herausforderung, die ich in meiner explorativen und experimentellen Herangehensweise nicht absehen konnte.

Das Listing hat einen Umfang von insgesamt 740 Zeilen BASIC-Code, wovon für das eigentliche Spiel 646 Zeilen notwendig sind. Ab Zeile 647 wurden noch zusätzliche grafische Elemente eingefügt, die schön, aber für das Spiel selbst entbehrlich sind. Vielmehr sollen die zusätzlichen Funktionen als Beispiele dienen, um zu zeigen, was der Anwender und BASIC-Programmierers sich selbst noch hinzuschreiben kann. Hamann und Eden haben genau dazu in ihrem Vorwort aufgefordert.

In den Zeilen 647 bis 708 werden aus den Buchstaben kleine Figuren und in den Zeilen 709 bis 740 wurde ein Blinken der zuletzt gezogenen Figur eingefügt, welches das Spiel erleichtert.

[28] https://www.musikundmedien.hu-berlin.de/de/medienwissenschaft/medientheorien/signallabor/fotos/p1040043.jpg, Abruf: 30.3.2017, 12:27 Uhr

code abtippen

Ich habe einmal gehört, dass gute Programmierer faul sind. Das ist natürlich nur übertragen zu verstehen, übertragen auf jede mögliche Vereinfachung und Erleichterung in der Arbeit. D.h. Unnötiges wird i.d.R. weggelassen. Das wollte ich auch tun und empfand als Erstes die durch REM im Code markierten Kommentare als unnötig. Nicht um das Programm zu verstehen, eher für die Funktion. Außerdem stehen die Kommentare im Listing und wären für mich jederzeit nachschlagbar. Diese Idee erwies sich jedoch als nicht durchdacht, denn dadurch formte sich zwangsläufig eine andere Struktur, als das Listing es vorgab und erschwerte den Blick und die Lokalisation bei der Fehlersuche, da mein abgetippter Code jeweils Lücken aufwies an den Stellen, wo eigentlich REM-Zeilen standen. Der Code hatte ein anderes visuelles Muster. Einen weiteren wichtigen Aspekt hätte ich ebenfalls ignoriert; die Antwort auf die Frage: warum Kommentare und Dokumentation wichtig sind?

Sie erleichtern das Verstehen, geben Hinweise und bauen Brücken von der formalisierten Syntax des Codes in den natürlichen[29] Sprachraum mit hoffentlich verständlicher Semantik, in dem sich die meisten Menschen bewegen. Verstehen lässt sich der Code auch ohne Kommentare, nur bedarf es dann oft Vorkenntnis und in meinem Fall, wesentlich mehr Zeit.

Um Zeit zu sparen, versuchte ich natürlich einfach den Code, der mir auch als PDF vorlag zu kopieren. Naheliegend. Jedoch ergaben sich so viele Fehler, die gleich auf den ersten Blick sichtbar waren, dass ich davon Abstand nehmen musste. Der Scan des Buches war an sich schon gut, aber sehr viele Zeichen wurden schlicht falsch interpretiert, so erkannte mein Rechner ein ‘l’ im Buch manchmal als eine ‘1’ oder ein ‘i’ usw. Die Korrektur hätte mit hoher Wahrscheinlichkeit mehr Zeit in Anspruch genommen, als das Abtippen.

Zeit war auch tatsächlich an anderer Stelle ein Problem, denn den Umständen als junger Vater zweier Söhne geschuldet, kam ich meist erst nachts dazu mich an den Rechner zu setzen und den Code Zeichen für Zeichen abzutippen. Insgesamt vier Nächte verbrachte ich vor meinem Computer, etwa acht Zeitstunden.

In den Nächten, die ich mit dem Abtippen des Code verbracht habe, konnte ich mehrere Beobachtungen an meinem Arbeitsverhalten machen. So liefen die ersten Zeilen recht gut, ich war aufmerksam und hielt mich immer genau an den Ursprungscode, doch dann schlichen sich Fehler ein. Fehler, die daraus resultieren, dass ich glaubte, den Code verstanden zu haben und schon immer etwas schneller schrieb. D.h. ich tippte schon mal das ein, was ich annahm, was als Nächstes kommen würde, weil sich manches zu wiederholen schien. Solche Fehler hab ich tatsächlich auch erst viel später gefunden.

Das Abtippen selbst ist eine monotone Sache, man richtet den Blick starr auf das Papier und tippt Zeichen für Zeichen ab. Maschinengleich. Und obwohl ich etwas BASIC-Vorerfahrung hatte, verstand ich nicht alles, was ich tippte und musste mich dem einfach hingeben. Ich dachte zwischenzeitlich kurz darüber nach, das Ganze auszulagern und jemanden zu bezahlen, der das für mich abtippt, suchte auch tatsächlich nochmal im Netz, ob eventuell

[29] die formalisierten Sprachen sind Teile des natürlichen Sprachraumes

schon jemand das Listing irgendwo veröffentlicht hat; erfolglos. So blieb mir nichts anderes übrig, als weiterzumachen und ganz ehrlich, wollte ich das auch schaffen! Zeile um Zeile, Zeichen um Zeichen immer in der Hoffnung möglichst wenig Fehler zu machen. An dieser Stelle, also beim Abtippen einer Software, eines Codes wurde mir klar, was es bedeutete, als zur Zeit der Listings die Menschen vor ihren Rechnern saßen und den Code Zeichen um Zeichen eingaben. Dies wurde vielfach getan, Tausende saßen so da, im Grunde tun dass viele Programmierer heute noch, nämlich fremde Code-Teile kopieren und in den eigenen Code einfügen. Nur heute geht das digital sehr einfach; Markieren → Zwischenablage → Einfügen. Fertig! Wie sieht das analog beim Abtippen aus? Markiert wird hier per visueller Wahrnehmung durch das Auge, als Zwischenablage fungierte der Kurzzeitspeicher unseres Gehirns (teilweise mit Übergang zum Langzeitspeicher, was unter Umstände zu Problemen führt s.o.) und das Einfügen erfolgt dann quasi biomechanisch per Finger auf Tastatur. Je nachdem wie geübt man dabei ist und wie schnell man tippen kann, dauert das eben so seine Zeit.

Kopieren/Einfügen (engl.: Copy/Paste) ist im weiteren Sinne als eine Kulturtechnik anzusehen, denn wir haben es hier mit einem Konzept zu tun, Aufgabenstellungen zu lösen, die grundlegende Fertigkeiten im Lesen und Schreiben erfordern, welche wiederum auch als Zivilisationstechniken bezeichnet werden. Schrift ist auch im engeren Sinne eine Kulturtechnik.

> *"Die Schrift als Kulturtechnik wird nicht durch ein neu auftretendes Medium – wie zum Beispiel den Computer – abgelöst, sondern wird durch dieses in ihrer Form und Funktion verändert."*[30]

Insofern bleibt für Copy/Paste festzustellen, dass wir mit Schrift operieren, mit einem System um Verständlichkeit zu erzeugen, Informationen zu transportieren. Genau das wird mit dem 'analogen' und dem 'digitalen' Copy/Paste im besten Fall erreicht. Auch wenn der Kopiervorgang selbst oft negativ konnotiert wird, was vor allem wirtschaftliche Gründe hat, sind wir hier bei einer Kulturtechnik angekommen, die verbreitete Anwendung findet.

> *"Der Mensch findet sich im Zeitalter des Abtipplistings so deutlich wie nie wieder danach zwischen Papier und Maschine wieder."*[31]

Ein weiterer interessanter Aspekt, ist der Zustand bzw. die Position, die man einnimmt, wenn man ein Listing von Papier abtippt. Man befindet sich direkt zwischen dem Papier und der Maschine, zwischen zwei Medien, die sonst nur über Scanner und Drucker, als technischer Geräte miteinander in direkter Verbindung stehen. Hier ist der Mensch mit seinen Fähig- oder Unfertigkeiten, mit seiner eigenen Geschwindigkeit und kulturellem Hintergrund das bindende Glied.

[30] Grube, Gernot; Kogge, Werner; Krämer, Sybille: Schrift: Kulturtechnik zwischen Auge, Hand und Maschine; Wilhelm Fink Verlag, 2005; S. 53
[31] Höltgen, Stefan: Input Source Code - Der Mensch zwischen Papier und Computer(spiel)*; 25.12.2015; http://www.simulationsraum.de/blog/2015/12/25/inputsourcecode; Abruf: 26.05.2017, 14:22 Uhr

code testen

Nach vier Nächten war der Code abgetippt und musste nun natürlich getestet werden. Es war zu erwarten, dass sich Fehler eingeschlichen haben, deren Lokalisation eventuell nicht einfach werden würde. Außerdem war ich total gespannt, wie das Schachspiel laufen wird und wie es sich spielt. Es handelt sich um ein Spiel aus den 80er Jahren, wo es, bedingt durch die damals zur Verfügung stehende Hardware eben andere ästhetisch Maßstäbe verwirklichte, als die heute der Fall wäre. Ich ging davon aus, dass das Spiel eher monochromatisch, pixelig und in seiner Anwendung eher umständlich würde.

Um die Software zu testen, konnte ich zwei Wege wählen. Einmal das Programm direkt auf dem Schneider CPC laden und ausführen oder in einem Emulator auf meinem eigenen Ubuntu-Rechner testen und im Erfolgsfall danach auf den Schneider CPC portieren. Letzteres erschien einfacher, denn mit einem Schneider CPC hab ich zuvor nie gearbeitet und auch noch keine Idee, wie der Code aus meinem Texteditor in eine BASIC lesbar Form gebracht werden könnte.

Emulator

Einen Emulator, der quasi als virtuelle Maschine den Rechner simuliert, auf dem das Listing laufen soll, zu nutzen ist zunächst eine gute Idee, nur muss man sich tatsächlich erst in die Welt der Emulatoren einarbeiten. Ich musste den Richtigen finden, einen der geeignet und gut in mein System (Ubuntu-Linux) implementiert ist, in welchem sich die Textdatei mit meinem Code öffnen oder kopieren lies und vor allem den richtigen Output für den Schneider CPC liefert. Diese verschiedenen Aspekte zu vereinen war nicht einfach und ist mir ohne fremde Hilfe auch nicht gelungen. Ich habe mich für die Emulatoren: JavaCPC[32] und Arnold[33] entschieden, auf meinem System installiert und den Code, den ich in einem Texteditor fertig abgetippt hatte hinein-kopiert, bzw. geladen. JavaCPC wurde in Java programmiert und benötigt das Java Runtime Environment (JRE). Arnold wurde in C geschrieben.

Beim Kopieren des Codes zu Arnold, trat zunächst ein Fehler auf, bei dem mich die Ursachensuche viel Zeit gekostet hat. Dieser Fehler hatte vor allem etwas mit BASIC zu tun, denn die Anführungszeichen, die ich im Texteditor gesetzt hatte, entsprachen nicht den Anführungszeichen in BASIC, d.h. visuell sah das Anführungszeichen auf dem Bildschirm zwar aus wie ein Anführungszeichen, jedoch war das nicht der Zeichensatz, den BASIC für ein Anführungszeichen verwendet. Nach einiger Suche fand ich das für den BASIC-Code passende Anführungszeichen und ersetzte es direkt in Arnold mit dem richtigen BASIC-Zeichensatz. D.h., auch wenn visuell der Code richtig abgetippt wurde, bedeutet das noch nicht, dass auch der verwendete Zeichensatz korrekt ist.

Mit Arnold liefen die Tests am besten. Jedoch startete das Schachprogramm nicht auf Anhieb. Nachdem ich einige der ersten

[32] https://sourceforge.net/projects/javacpc; Abruf: 21.05.2017, 12:34 Uhr

[33] http://arnold.emuunlim.com/faq.html und http://www.bannister.org/software/arnold.htm; Abruf: 20.05.2017, 12:34 Uhr

Tippfehler korrigiert hatte, erschien aber zumindest schon einmal das Spielfeld! BASIC hilft bei der Fehlersuche, denn die Zeile, in der das Programm stoppt und auf einen Fehler trifft, wird direkt auf dem Bildschirm angezeigt. In dem Bildbeispiel ist das ein Syntax-Fehler in Zeile 2030. Zur Zeile wird zusätzlich der Zeileninhalt aufgeführt. Im vorliegenden Fall:

```
2030 WINDOW SWAP 1,0
```

Das Problem an dieser Stelle war, dass das Fenster zum Wechseln nicht gefunden wurde, weil es an dieser Stelle einen Tippfehler gab. Anstatt eines Kommas hatte ich dort einen Punkt gesetzt. Von solchen Syntax-Fehlern gab es etliche in meinem Code. Nach und nach wurden diese mühsam mittels Abgleich der gedruckten Version korrigiert, auf dem Weg zum lauffähigen Spiel. In dem Bildbeispiel fällt auch auf, dass das Spielfeld mit den Figuren nicht korrekt initialisiert ist. Ein weiterer Fehler, der mühsam behoben wurde.

Nach einer Weile Korrekturarbeit lief das Programm schon, aber stürzte bei bestimmten Eingaben ab, dann stellte ich Überlegungen an, wie ich die Fehlersuche verbessern könnte. BASIC bietet die Möglichkeit mit Fenstern zu arbeiten und so in einem Unterfenster, den Code direkt anzuzeigen, bzw. die Stelle an der das Programm abstürzt. So definierte ich ein Fenster als Spielfeld und ein Unterfenster als 'Arbeitsbereich'. In diesem Unterfenster ließ sich nun der Code sehen, der Zeile für Zeile ausgeführt wird. Natürlich passiert das zu schnell um es sinnvoll visuell nachvollziehen zu können, daher suchte ich nach einer Möglichkeit diesen Ablauf zu entschleunigen. Fündig wurde ich leider nicht und behalf mich daher mit den BASIC-Befehlen: STOP[34] und CONT[35]. Insgesamt nahm die Fehlerkorrektur mehr Ressourcen in Anspruch, als ich es erwartet hatte, aber führt schlussendlich zum Etappenziel – ein im Arnold-Emulator lauffähiges Schachprogramm.

Der nächste Schritt hielt nun die Herausforderung bereit, aus Arnold heraus ein File zu erstellen, mit dem der Schneider CPC etwas anfangen kann. Der Schneider CPC kann sehr gut mit dem DSK[36] -Format umgehen, weshalb ich auch versuchte ein File in diesem Format zu erstellen. DSK ist ein *standard disk image format* mit der Endung .DSK. Jedoch gelang mir das nicht allein, sondern nur mit Hilfe eines befreundeten Entwicklers von Stefan Höltgen. Vielen Dank an dieser Stelle für das Zusenden der lauffähigen .DSK Datei!

Nun stand dem Testen des Schachspiels auf dem Schneider CPC nicht mehr viel im Wege, lediglich das Aufspielen war etwas trickreich. Aber auch hier half Stefan Höltgen und wir konnten gemeinsam im Signallabor der HU den Schneider CPC mit dem Programm starten. Es

[34] Der BASIC-Befehl STOP unterbricht die Abarbeitung eines BASIC-Programms und es wird die Meldung BREAK IN Zeilennummer und einem nachfolgenden READY. ausgegeben; https://www.c64-wiki.de/wiki/STOP; Abruf: 26.05.2017, 16:10 Uhr

[35] Der BASIC-Befehl CONT (engl.: continue) setzt die Abarbeitung eines unterbrochenen BASIC-Programms fort, das durch die BASIC-Befehle END oder STOP oder das Drücken der <RUN/STOP> -Taste angehalten wurde; https://www.c64-wiki.de/wiki/CONT; Abruf: 26.05.2017, 16:11 Uhr

[36] http://www.cpctech.org.uk/docs.html; Abruf: 26.05.2017, 17:18 Uhr

lief! Dank der vorhergehenden Emulation und Fehlerkorrektur auch problemlos.

code verstehen

Das Konzept für dieses Projekt sieht eine Manipulation des Codes vor. Dazu ist es nötig, den Code überhaupt zu verstehen. Seine Routinen, den Ablauf, die Struktur und natürlich die Stellen finden, an denen Veränderungen vorgenommen werden können und das Spiel trotzdem noch spielbar bleibt.

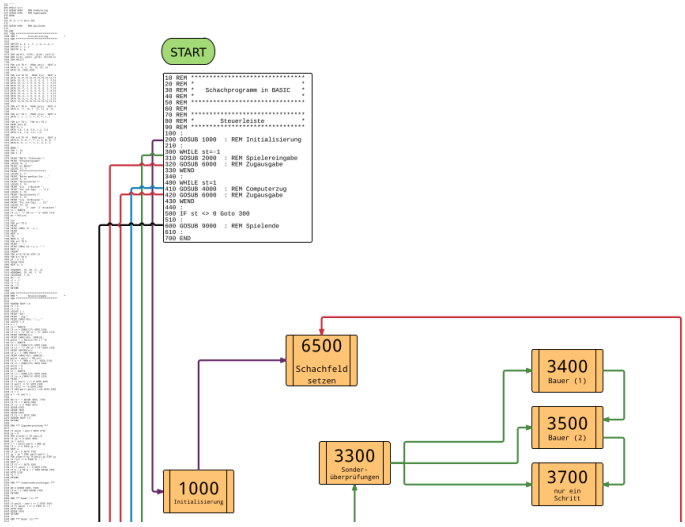
Das Verstehen des BASIC-Codes des Schachprogramms verlief in Etappen und ist als Prozess zu sehen. Zunächst halfen mir meine Grund-BASIC-Kenntnisse, die ich vor fast 30 Jahren erwarb. Erstaunlich wie präsent dieses Grundwissen plötzlich wieder wurde. Ich fühlte mich instantan in meine Schulzeit zurückversetzt und sah vor meinem geistigen Auge den KC 85/3 mit der Datasette und dem kleinen Junior s/w Fernsehapparat.

Da das gesamte Listing in der Grundversion insgesamt 740 Zeilen umfasst wirkt es auf den ersten Blick etwas unübersichtlich. Um die Struktur, die bei der Programmierung angewendet wurde zu erfassen und daraus das Verstehen abzuleiten, entschloss ich mich, dies graphisch darzustellen. Anhand einer übersichtlichen und farblich abgesetzten Graphik wollte ich die für die geplante Manipulation wichtigen Stellen lokalisieren. Im Verlauf der Graphikerstellung und mehrerer iterativer Schleifen, stellte sich heraus, dass es sinnvoll wäre, nicht nur das Programm in seiner Grundstruktur darzustellen, sondern auch die Sprungmarkenstruktur. Deshalb habe ich mir den gesamten Code unter dem Aspekt der Sprungmarken nochmal angesehen und dies durch verbindende Pfeile (mit Richtung) in die Graphik eingezeichnet.

Jetzt bot sich ein gänzlich strukturiertes Bild des Codes. Die einzelnen Programmteile wurden visuell gewichtet, ebenso wie die Abhängigkeiten untereinander. Auf einem Blick ist nun zu erkennen, welcher Teil sich auf welchen Teil bezieht und an welcher Stelle im Code wir uns bei den jeweiligen Schritten des Programmablaufs befinden. Das war für mich, für das Verstehen des Codes ein großer Schritt. Wenn ich nun mit jemandem über mein Vorhaben den Code evtl. zu manipulieren sprach, konnte ich direkt sagen, dass dies hier oder hier erfolgen könnte und was das in welchen anderen Teilen des Programms für die Rückgabewerte zur Folge hat.

Da sich der Bereich des Codes mit der Bezeichnung ‘COMPUTERZUG’ als Kern des Programms herausstellte, da hier die meisten eingezeichnete Pfeile konzentrierten, habe ich noch eine weitere Graphik angefertigt, die diesen Teil näher aufschlüsselte.

Bei näherer Betrachtung des Codes fiel mir zudem auf, dass es auffällig viele Subroutinen gibt. Um auch hier etwas mehr Verständnis mittels visueller Darstellung zu erreichen, gestaltete ich eine weiteren Graphik, welche diese Subroutinen umfassend abbildet.



Subroutinen insgesamt 24

GOSUB (BASIC-Befehl zum Aufruf einer Subroutine)

1000 Initialisierung
2000 Spielereingabe
3000 Zugüberprüfung
3300 Sonderüberprüfungen
3400 Bauernzug korrekt ? (1)
3500 Bauernzug korrekt ? (2)
3700 Überprüfung ob der Bauer nur einen zulässigen Schritt ausführen soll
4000 Computerzug
5100 Zugparameter einstellen
5400 Zugausgabe in Window#1
5600 Sortieren von fm(a,b)
5800 Bewertungsfunktion
6000 Zugausgabe
6500 Schachfeld setzen
6700 Interne Zugausführung
6900 Interne Zugrücknahme
7100 Rochadeüberprüfung
7400 Überprüfung König im Schach?
7600 Feldbestimmung
7800 Feldüberprüfung
7900 Zugerzeuger Computerzug
8200 Zugerzeuger Computerzug
8500 Positionswertung
9000 Spielende

Anhand der Darstellungen lässt sich nun der Programmablauf exakt nachvollziehen, was zu einem relativ umfassenden Verständnis des Codes führt und hilft die entsprechenden Stellen aufzuspüren, wo eventuell manipuliert werden kann.

code manipulieren

Der BASIC-Code bzw. die Programmierung erschien mir nach kurzer Einarbeitung ähnlich dem Schachspiel: einfache Strukturen, überschaubare Figuren (Befehle), viele kombinatorische Möglichkeiten und demzufolge eine Fülle an Möglichkeiten. Das veranlasste mich zu der Annahme, dass Manipulationen an einigen Programmteile relativ einfach zu realisieren wären. Die Manipulationen bzw. Modifikationen sollten wie folgt aussehen: 1. die Figuren können das Schachbrett auch nach links oder rechts verlassen und erscheinen dann auf der gegenüberliegenden Seite wieder; 2. das *en passant*-Schlagen sollte so verändert werden, dass ein Bauer berechtigt ist, einen vorbeiziehenden Turm oder Läufer oder die Dame zu schlagen. Der Bauer würde dann zusätzlich die Rolle der geschlagenen Figur einnehmen. Er würde zum Offizier befördert werden, wenn ein Offizier an ihm vorbeiziehen möchte und dabei geschlagen wird. Dies hätte eine größere Anzahl von Offizieren auf dem Feld zur Folge und brächte eine ungeahnte neue Dynamik ins Spiel.

Das Manipulieren des Codes sah ich zwar als realisierbar an, scheiterte jedoch daran. Gründe für das Scheitern waren zum einen: Zeitmangel und zum anderen aber auch ein Mangel an Programmierkenntnissen. Dennoch hat sich der Weg bis zu diesem Punkt des Projektes gelohnt, da ich mir so ein grundlegendes Verständnis zu dem Schachprogramm erarbeiten konnte.

Zusammenfassung

Warum gibt es dieses Projekt? Es ging um das Sichtbarmachen, das Entbergen wie Heidegger es ausdrücken würde. Das Entbergen von Vorgängen die sich der Kulturtechnik der Schrift, des *Copy/Paste*, der Technik allgemein und des Computers im Besonderen widmet. Es bestand für mich der Anspruch, Medienprozesse direkt erfahrbar zu machen, Medienkompetenz zu erhöhen und das Medienverständnis zu vertiefen; am Beispiel des Computerschachs. Dazu wurde ein altes Computerschach-Programm auf zeitgenössischer Hardware wiederbelebt, was gleichzeitig einen medienarchäologischer Aspekt im Umgang mit 'alter' Medientechnik bereithält. Das Programm wurde abgetippt und in mehreren Iterationen fehlerbereinigt. Mit dem Abtippen des Codes für das Schachprogramm erfolgt ein Perspektivenwechsel, welcher subjektive Erfahrungen ermöglichte, die in der Art nur in einem derartigen Projekt möglich würden. Die Position des Abtippenden zwischen abgedrucktem Listing auf Papier und dem ausführenden Rechner, das Direkte, Unmittelbare flankierte den epistemologischen Ansatz des Projektes auf wunderbare Weise.

Das Programm konnte mittels übersichtlicher Visualisierung der Ablaufstrukturen besser verstanden werden. Modifikationen, die das Verständnis der Strukturen und des Programmierens voraussetzen und gleichzeitig begründen waren geplant, wurde jedoch leider aus verschiedenen Gründen nicht realisiert und auch wenn die ursprüngliche Absicht das Schachprogramm zu manipulieren schlussendlich nicht umgesetzt wurde, gibt jetzt doch eine gute solide Vorarbeit für ein solches Vorhaben. Dies bleibt dann weiteren Projekten vorbehalten..
