

Mein Computer versteht mich

Die folgenden Seiten beschäftigen sich mit der Programmierung von Parsern – den Programmen, die den Spieler »verstehen«.

Was ist ein Parser? Schlägt man in einem Englisch-Deutsch-Wörterbuch nach, so findet man unter dem Begriff »to parse«: grammatisch zerlegen, analysieren. Oft wird statt Parser auch der Begriff Phraser: etwas ausdrücken oder formulieren, verwendet. In jedem Parser muß zwangsläufig auch ein Phraser enthalten sein. Der Parser ist in der Lage, eingegebene Sätze (zum Beispiel Adventure-Spiel-Befehle) grammatisch zu zerlegen, sie zu analysieren, um letztendlich eine mehr oder weniger intelligente Antwort zu geben. Es wird also erst »geparst« (Satz zerlegt), dann analysiert (Reaktion auf den Befehl) und schließlich »gephrast« (eine Antwort formuliert). Parser gewinnen im Bereich der Adventures und KI-Programmierung (KI=Künstliche Intelligenz) zunehmend an Bedeutung, denn letztendlich sind sie es schließlich, die eine Kommunikation zwischen Mensch und Maschine mittels der menschlichen Sprache ermöglichen. Wenn man sich einmal vor Augen hält, wie kompliziert die menschliche Ausdrucksweise mittels Sprache sein kann, dann kann man sich auch gut vorstellen, wie schwierig es ist, einen »perfekten« Parser zu programmieren. Trotzdem, die Fortschritte, die in letzter Zeit im Bereich Parser-Programmierung gemacht wurden, sind enorm; man bringt immer bessere Parser in immer weniger Speicherplatz unter. Inzwischen scheinen die Parser-Programmierer jedoch vor ernststen Schwierigkeiten zu stehen, besonders die, die Parser für die deutsche Sprache entwickeln wollen.

Zweideutigkeiten

Betrachten Sie einmal folgenden Satz:

»Moritz fotografiert die Frau mit dem Fotoapparat.«

Für uns ist der Fall eindeutig: Moritz hat einen Fotoapparat und fotografiert damit eine Frau. Der Satz kann jedoch auch völlig anders gedeutet und trotzdem völlig richtig werden: Moritz fotografiert eine Frau, die einen Fotoapparat bei sich hat. Dies ist nur eines von zahlreichen Beispielen. Uns soll dieses Problem jedoch nicht weiter berühren.

Unser Ziel ist es, einen Parser zu programmieren, der möglichst schnell ist, einen möglichst großen Wortschatz verwalten kann und dem es auch nicht an Intelligenz mangelt.

Der bekannteste Parser, den es gibt, ist der von Infocom. Zunächst möchte ich jedoch noch den Scott-Adams-Parser beschreiben. Scott Adams, Autor von bekannten Adventures wie Adventure, Pirate Cove, Voodoo Castle, The Hulk etc., hält auch heute noch an seinem 2-Wort-Parser fest. Ihm geht es wohl mehr um knifflige Aufgaben und weniger um großartigen Textumfang und Wortschatz. Bei dem 2-Wort-Parser handelt es sich um einen Parser, der nur Sätze versteht, die aus einem Verb und einem Objekt oder einer Richtungsangabe bestehen, zum Beispiel NIMM SCHWERT, WIRF SEIL etc.

Einen 2-Wort-Parser zu programmieren, bereitet wohl kaum jemand allzu große Schwierigkeiten. Auch die meisten Adventures, wie man sie vom Abtippen aus Zeitschriften kennt (zum Beispiel Zauberschloß), bedienen sich eines solchen Parsers. Der nächste Parser, der von sich Reden

machte, ist der, der in Melbourne Houses »The Hobbit« zur Geltung kam. Er versteht wirklich komplexe englische Sätze, Verknüpfungen sowie Unterhaltungen mit Personen, die im Spiel auftauchen. Beispiel: »Say to Gandalf please carry me«. Da sich das gesamte Hobbit-Programm auf einmal samt Grafik im Speicher befindet und nie nachlädt, läßt sich auf eine äußerst geschickte Programmierung schließen. Nun aber zum Infocom-Parser. Die Infocom-Adventures waren die ersten, die ständig auf Diskette zugegriffen. Dadurch konnte natürlich auch der Parser entsprechend besser gestaltet werden. Infocom-Spiele haben einen Wortschatz von bis zu 1000 Wörtern. Auch die Intelligenz des Parsers läßt nur äußerst selten zu wünschen übrig. Bis vor kurzem galt der Infocom-Parser als das Beste, was sich mit einem 64-KByte-RAM-Computer verwirklichen läßt. Vor noch nicht allzu langer Zeit wurde von Synapse/Broderbond ein Text-Adventure namens Mindwheel veröffentlicht; ausgerüstet mit einem BTZ-(Better than Zork) Parser, der angeblich bis zu 1500 Wörter verstehen soll. An dieser Stelle erlaube ich mir auch gleich zu erwähnen, daß wir im Rahmen dieses Kurses einen BABAZ-Parser (besser als besser als Zork) entwickeln werden. Ich bin mir sicher, daß dieser Parser wirklich das Optimalste ist, was sich mit 64 KByte RAM programmieren läßt. Allerdings hängt die Qualität eines Parsers keineswegs von der Größe des Wortschatzes ab. Werfen wir an dieser Stelle doch einmal einen Blick auf mein erstes Adventure »Gordon Saga«. Der Parser – so denke ich heute – war extrem schlecht programmiert. Trotzdem verstand er Sätze wie zum Beispiel »Ich will nach Norden gehen und das Schwert nehmen« genausogut wie »Nimm schnell das Schwert und renne dann sofort nach Norden«. Auch werden Sätze wie »Schließe die Tür« und sogar »Mache schnell die Tür zu« verstanden.

Besonders erstaunlich ist dies alles, wenn man weiß, daß der Wortschatz von Gordon Saga aus lächerlichen 85 Worten besteht. Sie sehen also: Auch mit geringem Wortschatzumfang lassen sich erstaunliche Parser programmieren.

Falls Sie noch keinerlei Vorstellung davon haben, wie ein Parser funktioniert, also wie Befehlssätze zerlegt, codiert und analysiert werden: alles darüber finden Sie im ersten Adventure-Kurs im 64'er-Sonderheft 2/85.

Der Grundaufbau eines Parsers

Das Grundschema eines jeden Parsers ist in Bild 1 veranschaulicht.

Die Grundlagen der Parsertechnik sind keineswegs speziell für den C64 oder einen 64-KByte-RAM-Computer ausgelegt. Sie gelten für jedes Computersystem – egal, wie groß und wie schnell.

Wie Sie aus Bild 1 entnehmen können, setzt sich der Grundaufbau eines Programmes, das Sprache verstehen soll, aus 4 oder 5 Teilen zusammen. Diese Teile wollen wir nun im einzelnen besprechen:

Befehlseingabe

Wie der Name schon sagt, geht es hier um die Eingabe eines Befehls. Dieser Befehl muß jedoch nicht unbedingt das Format »TUE DIES UND JENES« haben. Er kann genausogut eine Frage, eine Antwort oder eine Bemerkung sein. Unter einem Befehl versteht man also alles, was einem Computer mitgeteilt wird. Die Befehlseingabe besteht in der Regel beim C64 aus einer modifizierten INPUT-Routine. Bei anderen Computern, die bei ihrer INPUT-Routine von vorneherein Kommas und Anführungszeichen erlauben, muß erst gar nicht lange modifiziert werden. Eine modifizierte INPUT-Routine für den C64 stellt Listing 22 (im Artikel Dateiverwaltung) dar. In diesem Bereich des Parser-Programms läßt sich kaum noch etwas verbessern. Eine Befehlseingabe muß jedoch nicht unbedingt über Tastatur erfolgen. Manche Adventures können, zumindest teilweise, mit Joystick oder Maus gesteuert werden. Theoretisch wäre auch eine

Spracheingabe denkbar. Beide Systeme sind aber derart speicherintensiv, daß wir uns nicht näher damit beschäftigen werden.

Parser

Von der Befehlseingaberoutine aus gelangt der Befehlssatz (den ich auch in meinem zweiten Programmierkurs stets im String BE\$ ablegen will) in den Parser. Der Parser selbst besteht im Prinzip aus zwei Hälften: Befehlszerlegung und Befehls Codierung (siehe Bild 2):

Wer meinen ersten Adventure-Kurs gelesen hat, weiß bereits, was hiermit gemeint ist. In der ersten Hälfte wird der Befehlssatz zunächst in einzelne Worte zerlegt. Nun greift der Parser in der Codierungshälfte auf den Wortschatz zurück und codiert jedes einzelne Wort. Er stellt fest, daß es in seinem Wortschatz nicht enthalten ist. Diese Zerlegung/Codierung läßt sich auf verschiedene Art und Weise durchführen. Hierzu später mehr, denn zunächst müssen wir erst einmal wissen, wie ein Wortschatz aufgebaut ist.

Wortschatz

Unter einem Wortschatz versteht man alle Wörter, die ein Adventure beziehungsweise KI-Programm kennt. Bei Eliza jedoch ist der Wortschatz gleich Null – wenn man einmal von den Schlüsselbegriffen und Wörtern absieht. Ich muß an dieser Stelle jedoch auch zugeben, daß sich der Parser meiner Gordon Saga eines kleinen Tricks bedient – er gaukelt dem Spieler mehr Intelligenz vor, als er eigentlich besitzt. Gibt man bei einem Infocom-Spiel beispielsweise den Befehlssatz »Küsse das goldene Schwert« ein, so erhält man mit Sicherheit eine Antwort wie zum Beispiel »Das kann doch nicht Ihr Ernst sein!«. Der Infocom-Parser weiß also, daß es keinen Sinn ergibt, wenn man ein Schwert küßt (Es sei denn, das Schwert ist in Wirklichkeit gar kein Schwert, sondern eine hübsche, verzauberte Prinzessin – lechz – aber lassen wir das lieber!). Woher ein Parser so etwas weiß, werde ich Ihnen später ausführlich erklären!

Betrachten wir noch einmal den Gordon-Saga-Parser: Wenn im Spiel eine Tür vorkommt, so gibt es verschiedene Befehlssatz-Möglichkeiten, um diese Tür zu schließen:

1. SCHLIESSE DIE TÜR.
2. MACHE DIE TÜR ZU.
3. ICH WILL DIE TÜR ZU MACHEN.

Bis heute gibt es keinen anderen deutschen Parser, bei dem eine solche Eingabevielfalt erlaubt ist. Wie schafft der Gordon-Saga-Parser dies, wenn er doch nur über einen Miniwortschatz von 85 Worten verfügt?

Betrachten wir doch einmal einen kleinen Ausschnitt aus dem Wortschatz.

Dort steht gespeichert, daß TÜR ein Objekt ist, dem der Objektwert 1 zugeordnet ist.

Es ist auch festgelegt, daß SCHLIESSE und ZU Verben sind. Da beide die gleiche Bedeutung haben, wird beiden zum Beispiel der Verbwert 8 zugeordnet.

Wenn jetzt einer der 3 obigen Sätze nach den Verben ZU oder SCHLIESSE durchsucht wird, wird immer der Verbwert (beziehungsweise Verbzahl VE) 8 gefunden.

Bei der Objektsuche wird jedesmal der für TÜR entsprechende Wert 1 gefunden.

Der Parser hat somit zwei Werte gefunden:

Objektzahl = 1

und

Verbzahl = 8

Diese beiden Werte werden dem Analyse-Teil des Parsers übermittelt. Der weiß, daß eine Tür geschlossen werden soll, wenn VE=8 und OB=1. Er überprüft jetzt nur noch zwei Dinge:

- ob überhaupt eine Tür vorhanden ist und
- ob die Tür nicht bereits zu ist (man kann keine verschlossene Tür schließen).

Schließlich antwortet der Parser mit »Ok. Die Tür ist jetzt zu.«.

Gibt man bei Gordon Saga jedoch zum Beispiel einen Befehl ein, der unsinnig ist und »Schließe das Schwert« lautet, so schweigt der Parser beziehungsweise antwortet mit »Ich verstehe nicht, was Sie wollen«. Er erkennt also nicht – so wie der Infocom-Parser – daß es unsinnig ist, ein Schwert zu schließen. Entschuldigen Sie bitte, wenn ich hier etwas vom eigentlichen Thema Wortschatz abgewichen bin und schon etwas vorausgegriffen habe, aber Sie sehen ja selbst, wie eng die einzelnen Bereiche des Parsers miteinander verschlungen sind! Unser momentanes Problem ist es, wie man einen Wortschatz anlegt und wie man ihn im Computer unterbringt und verwaltet.

Es gibt verschiedene Möglichkeiten, einen Wortschatz zu programmieren. Die Zahl dieser Möglichkeiten hält sich jedoch in überschaubaren Grenzen.

Ganz anders ist dies bei der Analyse. Hier gibt es unheimlich viele Programmiermöglichkeiten. Der Analyse-Teil des Parsers ist nicht zuletzt auch der Teil, in dem die KI steckt. Die KI entscheidet auch, ob der Parser weiß, wovon er redet, oder ob er nur so tut, als verstünde er, was los ist. Sie sehen also schon, daß der »Parser« des ELIZA-Effektes keinesfalls mit einem Parser verglichen werden kann, den ein gutes Adventure haben soll. Wenngleich dies jedoch eine Kombination beider Möglichkeiten nicht ausschließen darf. Im Rahmen dieses Kurses werden alle Parserbereiche ausführlich behandelt. Es werden alle Möglichkeiten beschrieben, die sich mit einem C 64 und Floppy realisieren lassen. Was den Analyse-Teil anbelangt, so kann ich Ihnen jedoch nur einige Lösungsideen vorstellen (Kursteil Adventureprogrammierung). Jeder muß für sich selbst entscheiden, welches Analysekonzept für ihn das beste ist.

Ausgabe

Im Ausgabe-Teil des Parsers wird schließlich das Resultat der Analyse ausgegeben. Dies geschieht in der Regel in Form einer PRINT-Ausgabe auf dem Bildschirm.

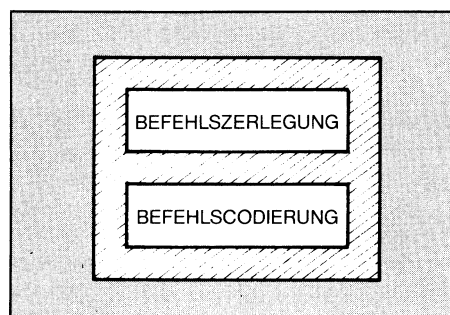


Bild 2. Grundfunktionen eines Parsers

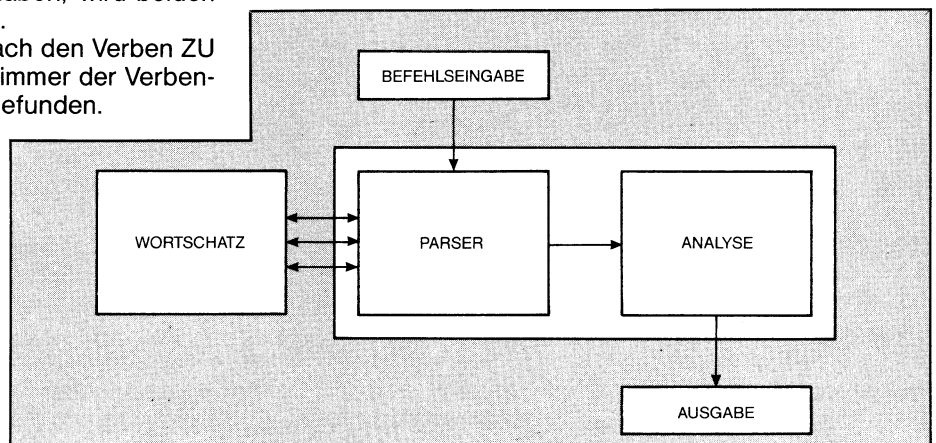


Bild 1. Aufbau eines Parsers

Parser 4.0 ist ein völlig Disketten-orientiertes System. Das heißt, der gesamte Wortschatz befindet sich in Form einer relativen Datei auf Diskette. Eine Diskettenseite faßt 170 KByte. Bei dem Wortschatz können wir davon ausgehen, daß die Wörter (Verben, Objekte etc.) eine Wortlänge von 20 Zeichen nicht überschreiten. Man kann theoretisch also Wortschätze mit über 6000 Wörtern verwalten. Praktisch ist dies jedoch Unsinn, da auf der Diskettenseite ja auch noch der

Text für das Adventure untergebracht werden muß. Realistisch ist es allerdings, mit Parser 4.0 Wortschätze mit etwa 500 Wörtern zu verwalten – denn dann ist noch genug Platz für Texte. Lassen Sie uns also jetzt gemeinsam den Disketten-orientierten Wortparser 4.0 entwickeln. Bevor man allerdings einen solchen Parser programmieren kann, braucht man erst einmal einen Wortschatz. Wie ist so ein Wortschatz aufgebaut? Die Liste in Tabelle 1 stellt einen solchen Relativen-

Tabelle 1. Wortschatz (relative Datei)								
1	abwaerts	2010	48	fliehen	1019	98	pruefen	1033
2	aufwaerts	2009	49	fragen	1066	99	putzen	1048
3	bedanken	1074	50	fuellen	1040	100	quetschen	1051
4	bedecken	1037	51	geben	1043	101	ra	2009
5	befestigen	1042	52	gebrauchen	1063	102	rasten	1027
6	bekleiden	1065	53	gehen	1001	103	rauchen	1079
7	belohnen	1046	54	gellen	1070	104	rauf	2009
8	benutzen	1063	55	gib	1043	105	reden	1073
9	berauben	1058	56	giessen	1050	106	reiben	1017
10	beruehren	1017	57	graben	1015	107	reinigen	1048
11	bestechen	1068	58	grabschen	1017	108	reisen	1052
12	betatschen	1017	59	greifen	1012	109	reiten	1029
13	betrachten	1011	60	haendigen	1043	110	rennen	1001
14	betreten	1018	61	heben	1044	111	ru	2010
15	betritt	1018	62	hinab	2010	112	rudere	1026
16	bewaessere	1050	63	hinauf	2009	113	rufen	1070
17	bewegen	1025	64	hinunter	2010	114	ruhen	1027
18	bezahlen	1046	65	hoch	2009	115	runter	2010
19	bieten	1043	66	holen	1012	116	s	2002
20	binden	1042	67	huepfen	1024	117	saeubere	1048
21	bitten	1066	68	informieren	1069	118	sagen	1073
22	blasen	1067	69	inventur	1013	119	saufen	1078
23	blockieren	1075	70	kaufen	1032	120	schauen	1011
24	brennen	1045	71	kitzle	1075	121	schenken	1043
25	bruellen	1070	72	klauen	1058	122	schicken	1055
26	buchstabieren	1016	73	koepfen	1057	123	schieben	1051
27	buddle	1015	74	kontrollieren	1063	124	schlafen	1028
28	danken	1074	75	kuessen	1071	125	schliessen	1035
29	dechiffrieren	1038	76	laufen	1001	126	schmecken	1059
30	druecken	1051	77	leeren	1039	127	schreien	1070
31	entfachen	1045	78	legen	1014	128	schuettle	1049
32	entkommen	1019	79	lesen	1016	129	schwimmen	1021
33	entkorken	1034	80	lies	1016	130	senden	1055
34	entleeren	1039	81	liste	1013	131	setzen	1030
35	entraetsle	1038	82	n	2001	132	so	2008
36	entriegle	1034	83	nehmen	1012	133	spielen	1072
37	entziffre	1038	84	nimm	1012	134	sprechen	1073
38	entzuenden	1045	85	no	2006	135	sprich	1073
39	erbitten	1066	86	norden	2001	136	springen	1024
40	ergreifen	1012	87	nordosten	2006	137	stecke	1047
41	ermorden	1057	88	nordwesten	2005	138	stehen	1031
42	erschliessen	1057	89	nw	2005	139	stehlen	1058
43	erstechen	1057	90	o	2004	140	stoepsle	1047
44	erwerben	1032	91	oeffnen	1034	141	strecken	1053
45	erwurb	1032	92	osten	2005	142	streichle	1017
46	erwuergen	1057	93	paddle	1026	143	suchen	1041
47	finden	1041	94	pennen	1028	144	sueden	2002
			95	pfeifen	1067	145	suedosten	2008
			96	polieren	1048	146	suedwesten	2007
			97	probieren	1059	147	summen	1067
						148	sw	2007
						149	taetschle	1017
						150	tauchen	1022
						151	toeten	1057
						152	tragen	1065
						153	tragen	1065
						154	trinken	1078
						155	umhuellen	1037
						156	unterrichten	1069
						157	untersuchen	1011
						158	verankere	1042
						159	verbinden	1042
						160	verbrennen	1045
						161	verdecken	1037
						162	verdrehen	1061
						163	verhuellen	1037
						164	verkaufe	1054
						165	verknotten	1042
						166	verkorken	1036
						167	verlassen	1020
						168	verlieren	1014
						169	verlocken	1068
						170	verloeschen	1062
						171	vernichten	1056
						172	verriegle	1036
						173	verruecken	1025
						174	verschieben	1025
						175	verschliessen	1036
						176	versperren	1036
						177	versuchen	1059
						178	w	2003
						179	wandere	1001
						180	warten	1027
						181	wecken	1076
						182	wedeln	1064
						183	werfen	1014
						184	westen	2003
						185	winden	1060
						186	winken	1064
						187	wirf	1014
						188	wuehlen	1015
						189	zerquetschen	1051
						190	zerreißen	1052
						191	zerren	1052
						192	zerschlagen	1056
						193	zerschmettere	1056
						194	zerstoeren	1056
						195	zertruemmere	1056
						196	ziehen	1025
						197	zuenden	1045

Datei-Wortschatz dar. Sie wurde mit Hilfe des Relativen-Dateien-Editors zusammengestellt und anschließend ausgedruckt.

Dieser Wortschatz besteht bis jetzt nur aus Verben und Himmelsrichtungsangaben. Diese Verben können als eine komplette Verbensammlung betrachtet werden, die man in einem Adventure benötigt. Das heißt, mit diesen Verben läßt sich nahezu jeder Befehlssatz ausdrücken (in Verbindung mit entsprechendem Objekt natürlich).

Wie liest man diese Wortschatztabelle? Stellen wir diese Frage erst einmal zurück! Nehmen Sie zunächst bitte einmal eine leere Diskette her und formatieren Sie diese. Dann legen Sie mit Hilfe des Relativen-Datei-Editors auf dieser Disk eine relative Datei mit folgenden Angaben an: 600 Sätze mit je 25 Zeichen. In diese relative Datei schreiben Sie dann die Wortschatz-Tabelle. Das geht einfach: Die linke Zahl in der Tabelle gibt jeweils die Satznummer der relativen Datei an. Danach kommt der Satzinhalt: Verbnamen und Codierungszahl. Bild 3 zeigt ganz genau, wie der erste Satz in der relativen Datei aussehen muß. Die Wörter (=Verben, Objekte etc.) dürfen also maximal 21 Zeichen lang sein. Die letzten vier Zeichen der Sätze sind für die Codierungszahl reserviert, deren Bedeutung uns im Moment noch nicht interessieren soll.

Beim »Abtippen« ist zu beachten, daß Sie nicht vom deutschen Zeichensatz Gebrauch machen. Das heißt, für »ae« darf nicht »ä« gesetzt werden und so weiter.

Wir haben jetzt also einen Wortschatz (bisher zwar nur Verben, aber das ist zunächst egal), auf den wir zurückgreifen können.

```

50000 REM ***** <227>
50001 REM * * <009>
50002 REM * WORT-PARSER 4.0 * <169>
50003 REM * * <011>
50004 REM * <C> 1986 BEI * <086>
50005 REM * * <013>
50006 REM * MICHAEL NICKLES * <082>
50007 REM * * <015>
50008 REM ***** <235>
50010 REM BEFEHLSATZZEINGABE -----
-----
-- <069>
50011 : <202>
50015 SL=80: REM BEFEHLSATZLAENGE <144>
50020 PRINT "{DOWN}": BE$="": POKE 198,0: POKE
211,0: POKE 214,22: SYS 58732: PRINT "{
YELLOW}"; <156>
50030 GET X$: IF PEEK(203)=1 THEN 50120 <226>
50040 IF X$="" THEN 50030 <171>
50050 IF LEN(BE$)=0 AND ASC(X$)=20 THEN 50
030 <148>
50060 I=ASC(X$): IF I<32 OR I>133 AND I<159
THEN IF I<>20 THEN 50030 <131>
50070 IF LEN(BE$)=SL AND I<>20 THEN 50030 <128>
50080 BE$=BE$+X$ <031>
50090 PRINT CHR$(20); X$; " "; <108>
50100 IF I=20 THEN BE$=LEFT$(BE$,LEN(BE$)-
2): GOTO 50030 <101>
50110 GOTO 50030 <001>
50120 PRINT "{LEFT,SPACE}" <227>

```

Listing 1. Befehlssatzzeingabe

```

51000 REM BEFEHLSZERLEGUNG-CODIERUNG -----
----- <174>
51001 : <176>
51010 VG$="NIMM": VG=1012 <081>
51020 REM SUCHEN OB VG$ IN BE$ VORKOMMT <057>
51030 FOR I=1 TO LEN(BE$)-LEN(VG$)+1 <010>
51040 : IF VG$=MID$(BE$,I,LEN(VG$)) THEN V
E=VG <088>
51050 NEXT I <077>
51060 IF VE=0 THEN PRINT VG$ " KOMMT NICHT
IN BE$ VOR" <092>
51070 IF VE=1012 THEN PRINT VG$ " KOMMT IN
BE$ VOR" <212>

```

Listing 2. Befehlszerlegung

In der Einleitung zum Kapitel Parsertechnik habe ich Ihnen bereits den Grundaufbau eines JEDEN Parsers vorgestellt. Am Anfang jedes Parsers kommt folglich die Befehlseingabe, die beim C64 in der Regel aus einer modifizierten INPUT-Routine besteht.

Nach Durchlauf der Befehlseingabe liegt der Befehl beziehungsweise Befehlssatz als String BE\$ vor. Lassen Sie mich nun wieder einmal von meinem alten Lieblingsbefehlssatz ausgehen: NIMM DAS SCHWERT. Also BE\$ = »NIMM DAS SCHWERT«. Dieser Befehlssatz soll jetzt anhand Vergleich mit dem Wortschatz codiert werden. Das heißt, dem VERB im Satz wird eine Verbzahl VE zugeordnet, dem Objekt eine Objektzahl OB, unbedeutende Wörter (zum Beispiel der, die, das und so weiter) werden ignoriert. Es gibt nun verschiedene Möglichkeiten, einen String beziehungsweise Befehlssatz zu codieren, nach Wörtern zu durchsuchen. Diese Möglichkeiten, ihre Vor- und Nachteile möchte ich Ihnen nun im folgenden einmal ganz genau vorstellen:

Möglichkeit 1

Bitte tippen Sie Listing 1 ab und ergänzen Sie es um Listing 2.

Bei dieser Befehlszerlegungs-Methode wird der Befehlssatz eigentlich gar nicht zerlegt. Es wird einfach überprüft, ob das Verb »NIMM« (stellvertretend für die GANZE Worttabelle) im Befehlssatz BE\$ enthalten ist. Dazu wird der gesamte Befehlsstring durchlaufen und mittels MID\$-Funktion überprüft, ob das Verb NIMM (=VG\$) im String vorkommt. Bild 4 demonstriert diesen Suchvorgang für den Befehlssatz »NIMM DAS SCHWERT«: das heißt, nehmen wir lieber als Befehlssatz »OTTO BITTE NIMM ES«, damit es nicht zu einfach wird. Aus Bild 4 wird auch deutlich, warum es legitim ist, anstelle von LEN(BE\$) LEN(BE\$)-LEN(VG\$)+1 zu setzen. Es ist schließlich logisch, daß ein Suchbegriff von 4 Zeichen Länge spätestens in den letzten 4 Zeichen von BE\$ gefunden werden muß.

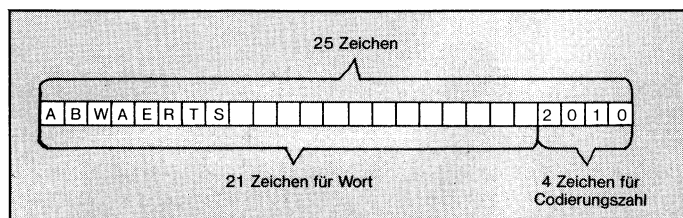


Bild 3. Satzaufbau in der relativen Datei

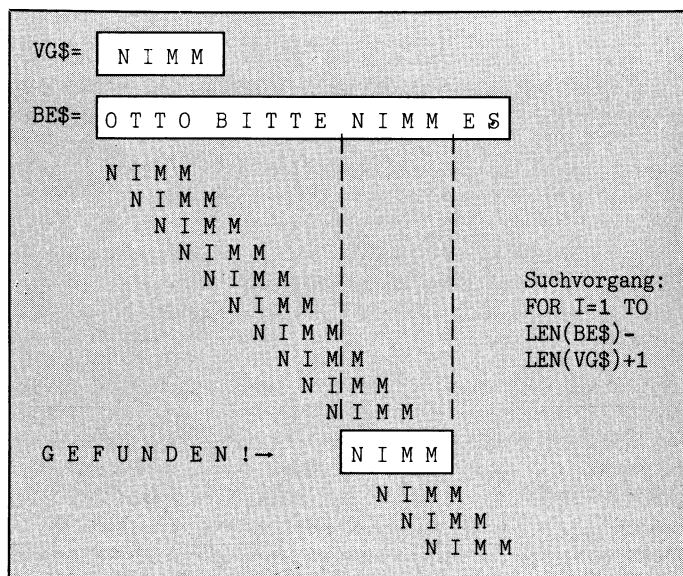


Bild 4. Beispiel für einen Suchvorgang

Achtung: Beim bisherigen Programm wird VG\$ im gesamten BE\$ gesucht. Es wird so zum Beispiel für BE\$=»OTTO BITTE NIMMES«, aber auch in »OTTOBITTENIMMES« gefunden. Will man nach absoluten Wörtern (zum Beispiel ELIZA-Schlüsselwörter) suchen, so muß sichergestellt sein, daß überprüft wird, ob sich am Ende und Anfang des gesuchten Wortes ein SPACE befindet.

Listing 2 muß für diesen Fall so modifiziert werden:

```
51015 BE$=BE$+" "+BE$+" "
51040 : IF " "+VG$+" " =MID$(BE$,I,LEN(VG$)+2) THEN
VE=VG
```

Methode 1 ist ganz die Methode, die bei ELIZA angewandt wird – nämlich dann, wenn die Schlüsselwörter gesucht werden. Das heißt, alle 38 Schlüsselwörter der ELIZA werden mittels dieser Methode in BE\$ gesucht. Nicht zuletzt aus diesem Grund braucht ELIZA ziemlich viel Zeit, bis sie zu einer Antwort kommt. Gehen wir nun einmal von einem Wortschatz für ein Adventure aus, der ca. 600 Wörter groß ist. Ich sagte oben ja bereits, daß das Verb NIMM stellvertretend für den ganzen Wortschatz gesucht wird. Tatsächlich müßten alle 600 Wörter im Befehlssatz gesucht werden. Dies würde viel zuviel Zeit in Anspruch nehmen und außerdem zu Schwierigkeiten führen, wenn sich ein Befehlssatz aus mehreren Einzelbefehlen, die durch UND oder KOMMA getrennt sind, zusammensetzt. Methode 1 ist im Bereich der Adventure-Parser-Programmierung folglich völlig unbrauchbar. Aber auch was die Anwendung bei ELIZA anbelangt, gibt es inzwischen eine wesentlich bessere Methode zum Suchen eines Strings VG\$ in einem String BE\$:

Möglichkeit 2

Ergänzen Sie diesmal Listing 1 mit Listing 3. Geben Sie auch hier als Befehlssatz BE\$ wieder »OTTO BITTE NIMMES« ein. Auch bei dieser Routine handelt es sich um eine Routine, die den Befehlsstring BE\$ durchläuft und VG\$ (wieder »NIMM«, stellvertretend für den gesamten Wortschatz) darin sucht. Allerdings erfolgt das Suchen nicht wie bei Methode 1 in »ein Zeichen«-Schritten. Bild 5 verdeutlicht den Suchvorgang bei der hier verwendeten Routine.

Während beim gleichen Befehlssatz BE\$ bei Methode 1 ganze 12 Schritte bis zum Finden von VG\$ in BE\$ benötigt wurden, sind es hier nur noch 4. Der in Methode 2 angewandte Algorithmus wurde übrigens erst vor kurzem erfunden und in der 64'er Ausgabe 2/85 im Kurs »Effektives Programmieren« beschrieben.

Wie funktioniert das in Bild 5? Nun, der Suchvorgang beginnt ganz klar am Anfang des Befehlssatzes. Besteht hier noch keine Übereinstimmung, so wird VGS einfach um eine bestimmte Zeichenanzahl nach rechts verschoben. Woher weiß das Programm, um wie viele Zeichen es bei Nicht-Übereinstimmung verschieben muß? Das Prinzip ist eigentlich ganz einfach: Zunächst wird eine Tabelle angelegt, aus der hervorgeht, wie viele Zeichen jeder Buchstabe im Such-

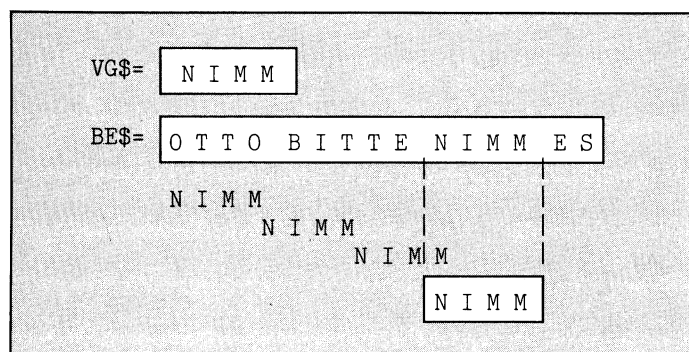


Bild 5. So kann schneller gesucht werden

wort VG\$ entfernt ist. Bei unserem Suchwort VG\$=»NIMM« gilt folgendes:

M = 0 entfernen. Bei doppelt auftretenden Buchstaben wie zum Beispiel M im Wort NIMM, wird der Buchstabe notiert, der dem Ende des Wortes näher ist. Für alle Buchstaben, die im Suchwort VG\$ nicht enthalten sind, wird als »Entfernungswert« die Länge von VG\$ eingetragen – also 4 bei VG\$=»NIMM«.

Für das Wort NIMM ergibt sich die folgende Tabelle:

VG\$ = NIMM		
A = 4	J = 4	S = 4
B = 4	K = 4	T = 4
C = 4	L = 4	U = 4
D = 4	M = 0	V = 4
E = 4	N = 3	W = 4
F = 4	O = 4	X = 4
G = 4	P = 4	Y = 4
H = 4	Q = 4	Z = 4
I = 2	R = 4	

Das Erstellen dieser Tabelle findet beim Programm in den Zeilen 51020–51052 statt. Aus dieser Tabelle kann nun immer ermittelt werden, um wie viele Zeichen VG\$ nach einem mißglückten Versuch nach rechts verschoben werden darf. Und das geht so: Betrachten Sie bitte wieder Bild 5. VG\$ wird von hinten nach vorn beziehungsweise von rechts nach links durchlaufen. Dabei wird jedes Zeichen mit der entsprechenden Stelle in BE\$ verglichen. Stimmen alle Zeichen überein, so ist das Wort VG\$ in BE\$ gefunden. Erfolgt jedoch bei einem Buchstabenvergleich keine Übereinstimmung, so darf BG\$ nach rechts verschoben werden – und zwar genau um die Zeichenzahl beziehungsweise zu dem Wert, der gemäß der Tabelle dem Zeichen von BE\$ zugeordnet ist, bei dem zum ersten Mal keine Übereinstimmung festgestellt wurde. Beim ersten Suchschritt wird NIMM mit OTTO verglichen. Es wird also schon beim letzten Buchstaben Übereinstimmung festgestellt, daß M ungleich O ist. Gemäß der Tabelle hat O den Wert 4. VG\$ darf somit um 4 Zeichen nach rechts verschoben werden. Allerdings spielt der Tabellenwert nicht die einzige Rolle beim Verschieben. Vom aus der Tabelle ermittelten Wert wird noch die Entfernung des Buchstabens vom Wortende abgezogen, bei der

```
10 DIM M%(26) <181>
51000 REM BEFEHLSZERLEGUNG-CODIERUNG ----- <174>
----- <176>
51001 : <176>
51010 VG$="NIMM":VG=1012 <081>
51020 REM M%-TABELLE ANLEGEN ----- <219>
51040 FOR I=1 TO 26:M%(I)=LEN(VG$):NEXT I <077>
51050 FOR I=1 TO LEN(VG$):M%(ASC(MID$(VG$,I,1))-64)=LEN(VG$)-I:NEXT I <101>
51052 FOR I=1 TO LEN(VG$):PRINT MID$(VG$,I,1),M%(ASC(MID$(VG$,I,1))-64):NEXT I <158>
51060 REM VG$ IN BE$ SUCHEN ----- <035>
51065 PRINT "{CLR,SPACE}";BE$ <120>
51070 PO=1 <025>
51080 FOR I=1 TO PO:PRINT " ";NEXT I:PRINT VG$ <104>
51085 IF VG$=MID$(BE$,PO,LEN(VG$)) THEN PRINT"GEFUNDEN !":END <214>
51090 FOR I=LEN(VG$) TO 1 STEP-1 <107>
51095 : IF MID$(BE$,PO+I,1)="" THEN PRINT"NICHT GEFUNDEN!":END <231>
51105 : IF MID$(VG$,I,1)=MID$(BE$,PO+I-1,1) THEN 51120 <218>
51109 : IF ASC(MID$(BE$,PO+I-1,1))=32 THEN PO=PO+LEN(VG$):I=0:GOTO 51120 <022>
51110 : PO=PO+M%(ASC(MID$(BE$,PO+I-1,1))-64)+LEN(VG$)-I:I=0 <216>
51120 NEXT I <147>
51130 GOTO 51080 <087>
```

Listing 3. Zweite Möglichkeit zur Befehlszerlegung

die erste Nichtübereinstimmung gefunden wurde. Dies ist nicht ganz einfach verständlich. Ich rate Ihnen daher eine eigenständige Analyse von Listing 3, Zeile 51060-51130. Danach wird Ihnen das Prinzip dieser Methode mit Sicherheit klar sein.

Man kann nicht so ohne weiteres behaupten, daß die zweite Methode der ersten überlegen ist. Auf den ersten Blick scheint Methode 2 zwar schneller zu sein, da weniger Verschiebungen notwendig sind. Dem steht jedoch eine wesentlich größere Rechenzeit als bei Methode 1 gegenüber. Am besten, Sie probieren selbst einmal aus, mittels TI\$-Zeitmessung, welche der beiden Methoden die geeignetere ist. Auf jeden Fall ist die Methode 2 äußerst interessant, nicht zuletzt deshalb, weil sie erst vor kurzem entdeckt wurde.

Möglichkeit 3

Wie Methode 1 eignet sich Methode 2 jedoch kaum zur Adventureprogrammierung, sondern vielmehr zur Entwicklung von ELIZA-ähnlichen Simulationen. Anders Methode 3:

Die 3. Methode entspricht weitgehend der Methode, die ich bereits in meinem ersten Adventure-Kurs verwendet habe. Es geht darum, den eingegebenen Befehlssatz BE\$ zunächst in einzelne Worte BE\$(0)-BE\$(N) zu zerlegen, wobei N für die Anzahl der im Befehlssatz vorkommenden Worte steht. Der Befehlssatz kann in BE\$(0)="NIMM" BE\$(1)="DAS" und BE\$(2)="SCHWERT" zerlegt werden.

BE\$(3) ist hier natürlich " ", wodurch das Satzende gekennzeichnet ist.

Nun wurde eine Wortzählvariable WZ auf 0 gesetzt. Sodann wurde das Wort BE\$(WZ) codiert – also im Wortschatz gesucht. Es wird so eine Verbzahl beziehungsweise Objektzahl etc. ermittelt. Wird das Wort nicht im Wortschatz gefunden, so kommt die Fehlermeldung »ICH KENNE DAS WORT...NICHT«, und der gesamte Codiervorgang wurde abgebrochen. Sodann wurde der Wortzähler WZ um 1 erhöht, und eine neue Suchrunde begann. Der ganze Satz wurde so also codiert. Der Suchvorgang wurde dann abgeschlossen, wenn BE\$(WZ)=» « war (also Satzende) oder BE\$(WZ)=»UND« beziehungsweise »«,« war (=Befehlssatz mit mehreren Befehlen). Wurde der Suchvorgang wegen letzterem unterbrochen, so wurde die Variable UD auf den Wert 1 gesetzt. Dadurch wußte das Programm, daß beim nächsten Mal, wenn der Spieler dran ist, die Befehlseingabe (INPUT-ROUTINE) übersprungen werden muß und die Codierung im alten Befehlssatz dort fortgesetzt werden mußte, wo man zuletzt wegen einem UND beziehungsweise KOMMA abgebrochen hatte.

In Bild 6a und 6b finden Sie nochmals die beiden Ablaufschemen, die diese Codiermethode beschreiben:

Wer meinen ersten Adventurekurs durchgearbeitet hat, dem sind diese beiden Ablaufschemen mit Sicherheit gut bekannt.

Obwohl die Ablaufschemen bereits über 1 Jahr alt sind und es inzwischen gravierende Verbesserungen im Bereich der Adventureprogrammierung gegeben hat (wie Sie schon bald

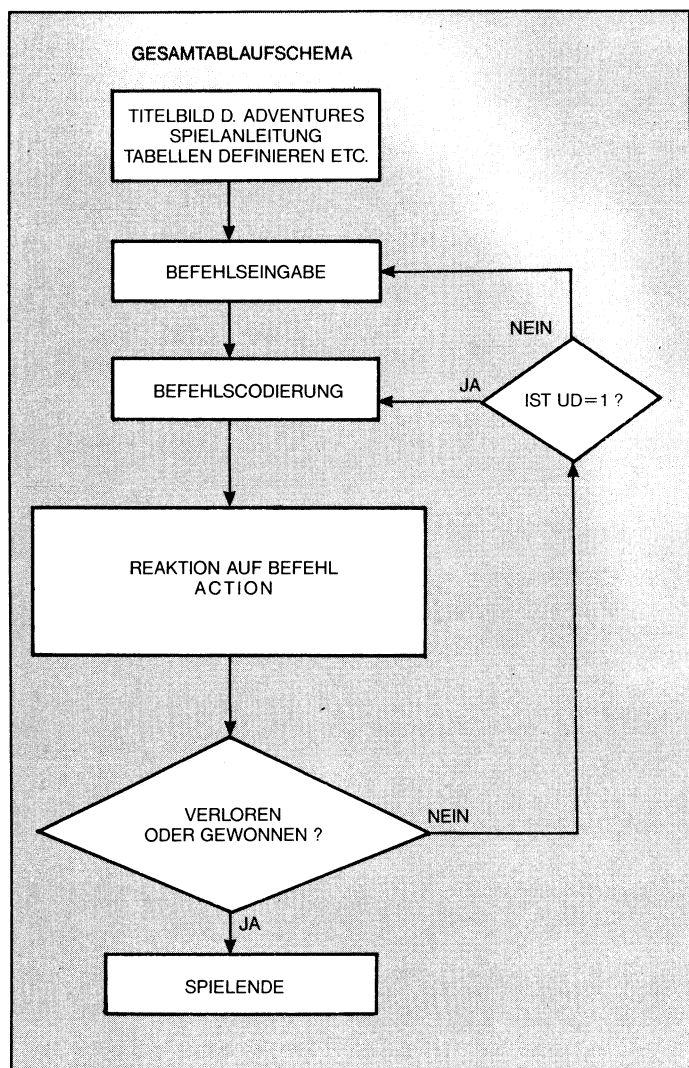


Bild 6a. Aufbau und Ablauf eines Spiels

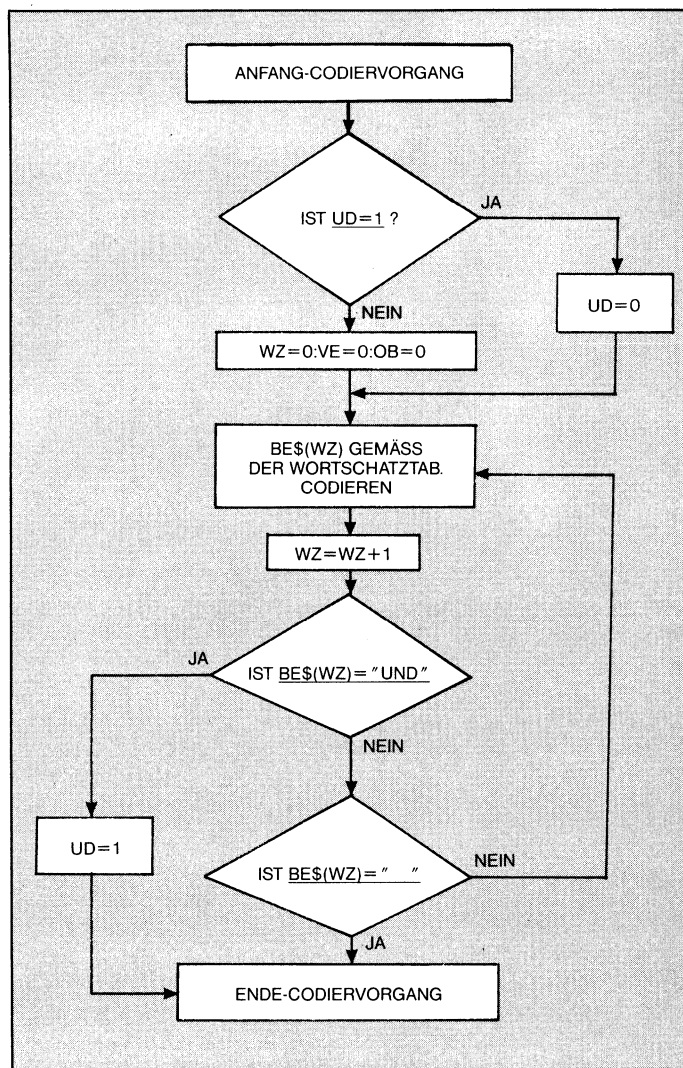


Bild 6b. Die Befehlscodierung im Detail

sehen werden !), haben sie keineswegs ihre Gültigkeit verloren.

Bild 6a stellt das Gesamtablaufschema eine Adventures dar. Sie können jedes Adventure-Programmierschema in dieses Schema quetschen. Unter Adventure läßt sich natürlich auch jedes andere KI-Programm verstehen, bei dem im Programm folgende Teile enthalten sind: Befehlseingaberoutine, Analyse des Befehlssatzes anhand eines gegebenen Wortschatzes, Interpretation der Analyse beziehungsweise Reaktion auf die Befehlseingabe.

Bild 6b beschreibt den Ablauf der Befehlskodierung nach Methode 3. Lassen Sie uns jetzt Methode 3 programmieren.

Ergänzen Sie dieses Mal Listing 1 bitte um Listing 4. Unser Parsersystem 4.0 ist nun insofern komplett, daß nach der Befehlseingabe der Befehlssatz BE\$ in einzelne Worte BE\$(1)-BE\$(N) zerlegt wird. Dabei ist festgelegt, daß der Befehlssatz aus maximal 10 einzelnen Wörtern bestehen darf. Dies läßt sich natürlich leicht abändern, aber 10 sollen uns zunächst genug sein zum Experimentieren. Die Wortzerlegung funktioniert äußerst primitiv: Es wird einfach BE\$ durchlaufen und nach den SPACES zwischen den einzelnen Wörtern gesucht. Eine Dokumentation zu Listing 4 erübrigt sich (hoffe ich doch !!!).

Jetzt muß der Codiervorgang gemäß Bild 6b programmiert werden. Ergänzen Sie das jetzige Programm bitte um Listing 5.

Auch Listing 5 läßt sich leicht verstehen. Auf Einbau von UND-Befehlstrennungen habe ich der Einfachheit halber zunächst verzichtet. Listing 5 bringt im Moment allerdings noch gar nichts. Wir müssen uns zunächst noch überlegen, wie wir das Wort in unserer relativen Wortschatzdatei finden beziehungsweise die einzelnen Befehlswörter BE\$(1)-BE\$(N). Im letzten Adventurekurs war dies noch ganz einfach: Da gab es für Verben und Objekte einzelne Tabellen (dimensionierte Stringfelder), in denen gesucht wurde. Jetzt befindet sich der gesamte Wortschatz, also Verben, Objekte, unwichtige Wörter, Richtungen etc; in einer relativen Datei, die alphabetisch sortiert ist. Nun, wie findet man Wörter in einer sortierten relativen Datei am schnellsten? Ganz klar – binäre Suchmethode. Wir müssen also in Listing 5 eine binäre Suchroutine einbauen. Eine solche Routine haben wir ja bereits im Kursabschnitt »Dateiverwaltung« kennengelernt. Bevor wir die binäre Suchroutine in Listing 5 einbauen, müssen wir uns allerdings noch einige Gedanken zur Organisation des Wortschatzes machen.

Kleine Wortschatzlehre

Werfen Sie an dieser Stelle bitte nochmals einen Blick auf Bild 3. Dort habe ich Ihnen gezeigt, wie sich die 25 Zeichen eines jeden Satzes der Wortschatz-Datei aufteilen: Die ersten 21 Zeichen sind für das Wort selbst reserviert. Also zum Beispiel für NIMM, ÖFFNE (Verben) oder auch HAUSTÜR, SCHRANK (Objekte). Die letzten 4 Zeichen jedes Satzes sind für eine Zahl – die Codierungszahl – reserviert. Was hat es nun mit dieser Codierungszahl auf sich? Ganz einfach, aus ihr läßt sich ermitteln, welcher Wortart das entsprechende Wort angehört und welche Zahl ihm als Codierung (zum Beispiel Verbzahl VE usw.) zugeordnet ist. Die 4stellige Zahl ist also vielmehr als zwei Zahlen beziehungsweise eine Ziffer und eine Zahl zu verstehen, die unmittelbar nebeneinander stehen. Bild 7 macht die genaue Unterteilung deutlich:

Die erste Ziffer der 4stelligen Zahl beschreibt somit die Wortart WA, während die letzten drei Ziffern den Wortcode WC bestimmen. Wir können also festlegen:

Wortart: WA = VAL(MID\$(A\$,22,1))

Wortcode: WC = VAL(MID\$(A\$,23,3))

A\$ steht dabei für einen beliebigen Satz der Wortschatz-

Datei. Das tatsächliche Wort ermittelt sich als

A\$=LEFT\$(A\$,21)

Allerdings hat das absolute Wort nicht immer die Länge von 21 Zeichen. Es wäre jedoch günstig, wenn in A\$ nur das Wort (keine SPACES mehr danach) stehen würde. Die SPACES kann man ganz einfach eliminieren:

```
9340 IFRIGHT$(A$,1)="cursor right" THEN A$=LEFT$(A$,LEN(A$-1)):GOTO 9340
```

Diese Zeile wiederholt sich so lange, bis in A\$ nur noch das Wort und keine SPACES mehr danach stehen. Eigentlich darf ich hier gar nicht von SPACES reden. Was wir auf dem Bildschirm als SPACE sehen, ist bei Sätzen, die von relativen Dateien, die mit unserem Relative-Dateien-Editor erstellt wurden, stammen, tatsächlich ein »cursor right«-Zeichen. Achten Sie also stets darauf, daß Sie in Sätzen aus unseren Dateien niemals nach einem SPACE, sondern vielmehr nach dem cursor-right-Zeichen suchen (vergleiche Beschreibung zum Relative-Dateien-Editor im Kapitel »relative Dateien«)!

Natürlich muß jetzt noch festgelegt werden, welche Wortartzahl welcher Wortart entspricht:

- 1 = Verb
- 2 = Richtung
- 3 = Objekt
- 4 = Aus-Wort

Bisher treten in unserem Wortschatz nur 1 und 2 als WA-Werte auf – kein Wunder, denn unser Ausgangs-Wortschatz besteht ja bisher nur aus Verben und Richtungen (die im Prinzip auch als Verben verstanden werden können, jedoch trotzdem einen eigenen WA-Wert benötigen, wie Sie gleich sehen werden).

Unter Aus-Wort verstehe ich jedes Wort, das im Befehlssatz auftreten kann, bei der Codierung jedoch keinerlei Rolle spielt. Gemeint sind Wörter wie zum Beispiel der, die, das, nach, vom und so weiter.

```
51000 REM BEFEHLSZERLEGUNG-CODIERUNG -----
-----
51001 : <174>
51005 REM BE$ IN BE$(1)-BE$(N) ZERLEGEN <176>
51010 FOR I=0 TO 10:BE$(I)="" :NEXT I <165>
51020 WZ=0 <183>
51030 FOR I=1 TO LEN(BE$) <059>
51040 : IF MID$(BE$,I,1)="" THEN WZ=WZ+1:GOTO 51060 <004>
51045 : IF WZ>10 THEN PRINT"EINGABE IST ZU LANG !":I=LEN(BE$)+1:GOTO 51060 <223>
51050 : BE$(WZ)=BE$(WZ)+MID$(BE$,I,1) <037>
51060 NEXT I <247>
<087>
```

Listing 4. Befehlskodierung nach Methode 3

```
51100 REM BE$(1)-BE$(N) CODIEREN <107>
51110 WZ=0 :REM WORTZAEHLER AUF 0 <094>
51120 VE=0:OB=0:REM ALTE ERG. LOESCHEN <247>
51130 REM WORT BE$(WZ) IM WORTSCHATZ SUCHE N <065>
51140 : <059>
51150 : <069>
51160 : <079>
51170 : <089>
51180 : <099>
51190 : <109>
51200 WZ=WZ+1 <241>
51220 IF BE$(WZ)="" THEN 52000 <197>
51230 GOTO 51130 <155>
52000 REM REAKTION AUF BEFEHL -----
-----
<118>
```

Listing 5. Der Codiervorgang nach Bild 6b

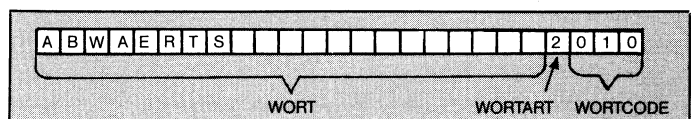


Bild 7. Die Wortcodierung im Datensatz

```

10 OPEN 1,8,3,"WORTSCHATZ":OPEN 15,8,15:RE
  M WORTSCHATZ-DATEI OEFFNEN <125>
20 INPUT SU$ <157>
30 GOTO 9500 <114>
9000 REM ZEIGER POSITIONIEREN -----
- <187>
9001 : <087>
9010 HB%=SA/256:LB%=SA-HB%*256 <068>
9020 PRINT#15,"P";CHR$(3);CHR$(LB%);CHR$(HB%) <005>
9030 INPUT#15,A: IF A=50 THEN PRINT"SATZNUM
  MER ZU HOCH" <088>
9040 RETURN <208>
9100 REM SATZ LESEN -----
- <178>
9101 : <187>
9110 IF A=50 THEN RETURN <082>
9120 INPUT#1,A$ <116>
9130 RETURN <042>
9200 REM SATZ SCHREIBEN -----
- <105>
9201 : <031>
9210 IF A=50 THEN RETURN <182>
9220 PRINT#1,A$ <042>
9230 RETURN <144>
9300 REM SATZ-TEILBEREICH LESEN -----
- <021>
9301 : <133>
9310 IF A=50 THEN RETURN <028>
9320 INPUT#1,A$ <062>
9330 WA=VAL(MID$(A$,22,1)):REM WORTART <075>
9332 WC=VAL(MID$(A$,23,3)):REM WORTCODE <100>
9335 A$=LEFT$(A$,21) :REM WORT <075>
9340 IF RIGHT$(A$,1)="(RIGHT)"THEN A$=LEFT
  $(A$,LEN(A$)-1):GOTO 9340 <171>
9345 PRINT A$,LEN(A$) <049>
9350 RETURN <008>
9500 REM BINAERE SUCHROUTINE -----
- <166>
9520 REM SU$=SUCHWORT <044>
9540 SZ=197:N=INT(LOG(SZ)/LOG(2))+1
  :REM MAXIMAL-FORMEL <224>
9560 SA=(2↑N)/2:REM MITTE DER GES.DAT <209>
9565 GOSUB 9000:GOSUB 9300
  :REM A$ LESEN <121>
9580 N=N-1 :REM 1.ABFRAGE <234>
9600 : <178>
9620 REM SU$ MIT A$ VERGLEICHEN ----- <251>
9623 : IF LEN(SU$)<3 THEN 9635 <189>
9625 : IF WA=1 AND SU$=LEFT$(A$,LEN(SU$))
  THEN PRINT"SU$=VERB:SATZ ";SA:END <223>
9630 : IF WA=3 AND SU$=RIGHT$(A$,LEN(SU$))
  THEN PRINT"SU$=OBJEKT:SATZ ";SA:END <172>
9635 : IF WA=2 AND SU$=A$
  THEN PRINT"SU$=RICHTUNG:SATZ ";SA:END <192>
9637 : IF WA=1 AND SU$=A$
  THEN PRINT"SU$=VERB:SATZ ";SA:END <094>
9640 : IF SU$<A$THEN SA=SA-(2↑(N-1)):GOSUB
  9000:GOSUB 9300:GOTO 9700 <060>
9660 : IF SA+(2↑(N-1))>SZ AND N>-1 THEN N=
  N-1:GOTO 9660 :REM "UEBERLAUF" <038>
9680 : SA=SA+(2↑(N-1)):GOSUB 9000:GOSUB 93
  00 :REM "HALBIEREN" <154>
9700 : <022>
9720 :N=N-1 :REM NAECHST. HALBIEREN <021>
9740 :IF N<0 THEN PRINT"SU$ GIBT'S NICHT!
  IN 1220":END :REM SU$ NICHT IN DAT
  EI <043>
9760 PRINT"SA=";SA;" N=";N
  :REM UNNOETIG !! <199>
9780 :GOTO 9600:REM NAECHST. SUCH-VERS. <069>

```

Listing 6. Die Binäre Suchmethode

NIMM			
SCHUETTLE	POLIEREN	NORDWESTEN	
SA=128 N=8	SA=96 N=5	SA=88 N=3	
HINUNTER	LIES	NIMM	
SA=64 N=6	SA=80 N=4	SA=84 N=2	

Bild 8. Binär gesucht = schnell gefunden

Bei dem Wortschatz, den wir im letzten Adventurekurs verwendet haben, mußten die Wörter nicht mit Zahlen gekennzeichnet werden. Auch Wörter mit gleicher Bedeutung wurden anders abgehandelt. Die nun angewandte Methode hat natürlich ihre Vorteile: Zum einen läßt sich der Wortschatz leichter um Worte erweitern, und zum anderen müssen nicht mehr mehrere Einzeltabellen verwaltet werden (für die verschiedenen Wortarten).

Eine gute Wortschatzverwaltung beeinflusst nicht zuletzt auch die Anwenderfreundlichkeit eines Parsers in positivem Maße. Ein anwendungsfreundlicher Parser ist ein Parser, der bei der Befehlssatz-Analyse äußerst tolerant vorgeht.

Es muß zum Beispiel anstelle des Verbs NIMM auch NEHME und NEHM sowie gegebenenfalls auch HOLE und ERGREIFE verstehen. Der Trick bei dieser Sache ist, daß mehr Worte verstanden werden, als eigentlich im Wortschatz stehen. Was das soll? Ganz einfach:

Nehmen wir einmal an, wir wollen in einem Adventure die NIMM-Routine programmieren:

```

IF VERB=NIMM AND OBJEKT X IST HIER THEN PRINT
"OK. ICH HABE OBJEKT X."

```

So könnte eine vereinfachte Syntax lauten. Dem Verb NIMM ist gemäß der Wortschatztabelle der Wortwert 12 zugeordnet (also VE=12). Wir können also sagen:

```

IF VE=12 AND OBJEKT X IST HIER THEN PRINT "OK.
ICH HABE OBJEKT X".

```

Was passiert, wenn der Spieler den Befehlssatz »HOLE OBJEKT X« eingibt? Nun, das Programm antwortet mit »OK. ICH...«, da dem Verb HOLE ebenfalls der Wortwert (beziehungsweise Verbwert) 12 zugeordnet ist. Unser Wortschatz ist ziemlich intelligent aufgebaut – intelligent insofern, daß Verben gleicher Bedeutung der gleiche Verbwert zugeordnet ist. Ein guter Parser sollte allerdings »NIMM DAS SCHWERT« genauso gut verstehen wie »ICH WILL DAS SCHWERT NEHMEN« und »NEHM DAS SCHWERT« (auch wenn es grammatikalisch nicht in Ordnung ist). Wie stellt man es nun an, daß NEHM genauso gut verstanden wird wie NEHMEN? Nun, ganz einfach. NEHM steckt schließlich in NEHMEN drin. Nicht zuletzt ist dies auch der Grund, warum im Wortschatz immer die längstmögliche Form eines Verbs gespeichert wird – insofern ist dies möglich. Die meisten routinierten Adventure-Spieler geben eh in der Syntax NIMM, VERLIER, GIB etc. ein. Wir können uns deshalb auch hauptsächlich an dieser Form orientieren. Nehmen wir einmal an, der Satz »NIMM DAS SCHWERT« soll codiert werden (welcher sonst). BE\$(0) ist dann ganz klar »NIMM«. Wir wollen BE\$(0) jetzt anhand des Wortschatzes codieren. Mittels der binären Suchmethode werden dann verschiedene Sätze vom Wortschatz eingelesen (als String A\$) und mit SU\$(=BE\$(0)) verglichen. Bild 8 zeigt, wie das Wort NIMM im Wortschatz nach der binären Suchmethode gefunden wird.

Wie wird A\$ eigentlich mit SU\$ verglichen?

Ein Vergleich IF A\$=SU\$ THEN PRINT »GEFUNDEN« wäre mit Sicherheit falsch, da man in diesem Fall niemals das Verb NEHM finden würde (im Wortschatz steht nur NEHME). Der korrekte Vergleich lautet daher:

```

IF SU$=LEFT$(A$,LEN(SU$)) THEN ? "GEFUNDEN"

```

Dadurch sind bei der Befehlseingabe auch Abkürzungen der Verben erlaubt – zum Beispiel UNTERSU statt UNTERSUCHEN beziehungsweise NEHM statt NEHMEN. Dies ist mit Sicherheit jedem sonnenklar. Was aber, wenn der Spieler nach Norden gehen will? Er wird dann mit größter Wahrscheinlichkeit BE\$=»N« eingeben. Aufgrund unserer bisherigen Abfragemethode IF SU\$=LEFT\$(A\$,...) wird das Programm annehmen, daß das erste Wort, das mit N beginnt, das richtige ist. Werfen Sie doch bitte einmal einen Blick in die Wortschatztabelle: Sie werden merken, daß dort für Norden sowohl »N« als auch »NORDEN« vermerkt ist. Nicht umsonst haben Richtungen deshalb einen anderen WA-Wert als Ver-

ben. Sicher erinnern Sie sich daran, daß A\$ vor dem Vergleich mit SU\$ präpariert wird. Zunächst wird A\$ mit einem Satz der Wortschatzdatei gefüllt. Dann werden Wortcode WC und Wortart WA ermittelt. Zuletzt steht in A\$ dann nur noch das absolute Wort. Dadurch, daß die Wortart beim Vergleichen bereits bekannt ist, kann folgender Vergleich programmiert werden:

```
IF WA=1 AND SU$=LEFT$(A$,LEN(SU$)) THEN ? "VERB GEFUNDEN"
```

```
IF WA=2 AND SU$=A$ THEN ? "RICHTUNG GEFUNDEN"
```

Sie sehen also, bei RICHTUNGEN muß SU\$ mit A\$ übereinstimmen und SU\$ nicht nur linksbündig in A\$ enthalten sein.

Wie muß nach Objekten gesucht werden? Wir wollen uns auch mit dieser Frage beschäftigen – auch wenn sich im Wortschatz bisher noch keine Objekte befinden.

Objektnamen setzen sich oft aus Hauptwörtern zusammen. So zum Beispiel HAUSTÜR, HOLZKISTE, WEINFLASCHE und so weiter. Will der Spieler eine Haustür öffnen, so kann er sagen »ÖFFNE DIE HAUSTÜR« oder auch »ÖFFNE DIE TÜR«. Der Parser muß also wissen, daß HAUSTÜR=TÜR, HOLZKISTE=KISTE und so weiter ist. Entscheidend zur Bestimmung eines Objektes ist stets das zweite Hauptwort (beziehungsweise das Hauptwort, das ganz rechts im Objektnamen steht, beziehungsweise das Objekt selbst, wenn es sich nicht aus mehreren Hauptwörtern zusammensetzt). Es bietet sich also an, bei der Objektsuche von »rechts her« zu vergleichen:

```
IF WA=3 AND SU$=RIGHT$(A$,LEN(SU$)) THEN ? "OBJEKT-GEFUNDEN"
```

Das war wieder mal ziemlich viel Theorie. Listing 6 setzt das eben Gelernte in die Praxis um.

Dokumentation:

10-30	Hier wird die Wortschatzdatei geöffnet. Vor Programmstart ist es also erforderlich, daß Sie die Diskette mit der Wortschatzdatei einlegen. Anschließend wird ein Suchbegriff SUS eingegeben.
9000-9230	Hier stehen die drei Unterprogramme, die beim Arbeiten mit relativen Dateien immer wieder benötigt werden: Positionieren, Lesen und Schreiben.
9300-9350	Hier wird ein Satz A\$ der Wortschatzdatei gelesen. Zunächst werden Wortcode WC und Wortart WA gelesen, dann die Codierzahl vom A\$ abgeschnitten und alle SPACES (cursor-rights) hinter dem absoluten Wort weggenommen (Zeile 9340).
9500-9780	Hier steht die binäre Suchroutine, wie ich sie Ihnen bereits im Kapitel Dateiverwaltung Listing 15 beschrieben habe. Die Routine wurde zwecks Zusammenarbeit mit relativer Datei entsprechend modifiziert.
9540	Für SZ muß die Anzahl der Worte, die im Wortschatz enthalten sind, eingesetzt werden. (Im Moment haben wir 197 Worte.)
9620-9637	Hier werden die Objekte, Verben und Richtungen gesucht.

0 REM *****	<131>		
1 REM * INDEX-MAKER - ERSTELLT DEN *	<097>		
2 REM * INDEX ZU EINEM BELIEBIGEN, *	<058>		
3 REM * ALPHABETISCH GEORDNETEN *	<134>		
4 REM * BEREICH EINER RELATIVEN DATEI *	<198>		
5 REM * <C> 1986 BEI MICHAEL NICKLES *	<163>		
6 REM *****	<137>		
10 PRINT "CLR,YELLOW,RVSON)INDEX-MAKER(RVDF	<022>		
20 DIM M%(26) : REM INDEX-TABELLE	<034>		
30 PRINT "DOWN)BITTE GEBEN SIE DEN NAMEN D	<039>		
40 PRINT "DATEI AN, VON DER EIN INDEX ERSTE	<250>		
50 PRINT "WERDEN SOLL!"	<202>		
60 POKE 198,0: INPUT "DOWN)";D\$	<225>		
70 OPEN 1,8,3,D\$: OPEN 15,8,15: REM DATEI OE	<197>		
80 PRINT "DOWN)BITTE GEBEN SIE DEN BEREICH	<143>		
90 PRINT "VON DEM DER INDEX ERSTELLT WERDEN	<061>		
100 POKE 198,0: INPUT "DOWN)ANFANG"; AN	<133>		
110 POKE 198,0: INPUT "DOWN)ENDE (2SPACE)"; E	<086>		
120 SA=AN:GOSUB 9000: IF A=50 THEN PRINT "AN	<236>		
130 SA=EN:GOSUB 9000: IF A=50 THEN PRINT "EN	<190>		
140 IF A=EN THEN PRINT "UNSINN!";GOTO 100	<244>		
1000 REM INDEX-ERSTELLEN			
1001 :	<138>		
1005 PRINT "DOWN)BITTE WARTEN !"	<016>		
1010 SA=AN:GOSUB 9000	<149>		
1020 IZ=0	<042>		
1030 FOR SA=AN TO EN	<246>		
1035 :	<251>		
1040 : GOSUB 9100:PRINT SA,A\$	<000>		
1050 : IF M%(ASC(LEFT\$(A\$,1))-64)=0 THEN M	<227>		
1080 :	<040>		
1100 NEXT SA	<012>		
2000 REM INDEX AUSGEBEN			
			<185>
			<199>
2001 :			<030>
2010 PRINT "CLR)INDEX-AUSGABE:"			
2020 PRINT "2DOWN)AUSGABE AUF BILDSCHIRM <			<068>
2030 PRINT "DOWN)ALS SEQ-FILE AUF DISK (2SP			<247>
2040 POKE 198,0			<170>
2050 GET I\$: IF I\$<>"1" AND I\$<>"2" THEN 2050			<094>
2060 IF I\$="1" THEN 2100			<023>
2070 IF I\$="2" THEN 2200			<170>
2100 REM AUSGABE AUF BILDSCHIRM			<197>
2101 :			<045>
2110 PRINT "CLR)"			<066>
2120 FOR I=1 TO 13:PRINT CHR\$(I+64)="M%(I			<086>
2130 END			<100>
2200 REM M%(1)-M%(26) ALS SEQFILE SPEICHER			<168>
2201 :			<145>
2210 POKE 198,0: INPUT "DOWN)FILENAME "; I\$:			
I\$=I\$+" ,S,W"			<043>
2220 OPEN 2,8,2,I\$			<006>
2230 FOR I=1 TO 26			<223>
2240 :PRINT#2,M%(I)			<103>
2250 NEXT I:CLOSE 2			<113>
2260 END			<230>
9000 REM ZEIGER POSITIONIEREN			
			<187>
9001 :			<087>
9010 HB%=SA/256:LB%=SA-HB%*256			<068>
9020 PRINT#15,"P";CHR\$(3);CHR\$(LB%);CHR\$(H			<005>
B%);CHR\$(1)			
9030 INPUT#15,A: IF A=50 THEN PRINT "SATZNUM			<088>
MER ZU HOCH"			<208>
9040 RETURN			
9100 REM SATZ LESEN			
			<178>
9101 :			<187>
9110 IF A=50 THEN RETURN			<082>
9120 INPUT#1,A\$			<116>
9130 RETURN			<042>

Listing 7. Mit diesem Programm kann der Index einer relativen Datei erstellt werden

9623

Wenn die Zeichenanzahl des Suchbegriffs weniger als 3 Zeichen beträgt, muß davon ausgegangen werden, daß es sich bei SU\$ um eine Richtungsangabe handelt. Das heißt auch, daß Verbenabkürzungen mindestens 3 Buchstaben betragen müssen. Man kann für INVENTUR also nicht die Abkürzung I eingeben – es sei denn, I wurde zusätzlich im Wortschatz definiert (Zeile 9637 sorgt dafür, daß solche Verbabkürzungen akzeptiert werden).

Ist LEN(SU\$) also kleiner als 3, so werden die Zeilen 9625 (Verbvergleich linksbündig) und Zeile 9630 (Objektvergleich rechtsbündig) übersprungen.

9625

Verbuche (linksbündig)

9630

Objektuche (rechtsbündig)

9635

Richtungssuche (absolut)

9637

Verbuche (absolut) – für Verben, deren Abkürzung auch weniger als 3 Zeichen betragen darf – müssen im Wortschatz extra verzeichnet werden!

Zum Abschluß von Methode 3 müßte Listing 6 nun eigentlich mit Listing 5 kombiniert werden. Ich möchte jedoch davon absehen, da Sie dies zum einen leicht selbst machen können und es zum anderen noch eine bessere Möglichkeit gibt, die im Prinzip eine Verbesserung des letzten Programms bedeutet:

Möglichkeit 4

Unser Wortschatz besteht aus 197 Worten. Um ein bestimmtes Wort mittels der binären Suchmethode in diesem Wortschatz zu finden, beziehungsweise um festzustellen, daß dieses Wort im Wortschatz nicht vorkommt, sind maximal 8 Lesezugriffe auf Disk erforderlich. Jeder Schreib- beziehungsweise Lesezugriff auf Disk kostet Zeit. Auch durch Assemblerprogrammierung läßt sich diese Zugriffszeit kaum verkürzen. Es gibt allerdings eine Suchmethode, bei der weniger Lesezugriffe aus Disk nötig sind als bei Möglichkeit 3.

Das Index-Binäre-Suchprinzip

Keine Angst, dieses Prinzip läßt sich einfacher verstehen, als der komplexe Name vermuten läßt. Beim Index-Binären-Suchprinzip handelt es sich lediglich um eine Verbesserung

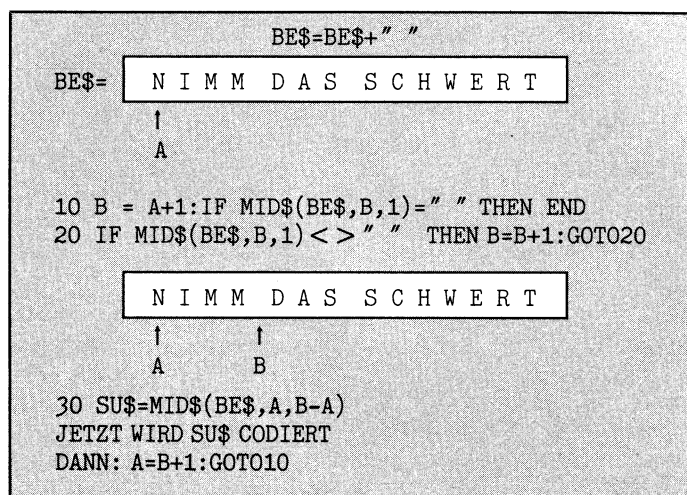


Bild 9. Befehlskodierung mit Methode 4

der binären Suchmethode. Gehen wir einmal davon aus, daß unser Suchbegriff `SU$=»NIMM«` ist. Unser Parser erkennt nicht auf Anhieb, daß `SU$` ein VERB ist. Er erkennt `SU$` erst dann als Verb an, wenn er es im Wortschatz gefunden hat. Etwas könnte der Parser allerdings schon vor Beginn des Suchvorgangs wissen – und zwar, daß der Suchbegriff mit dem Buchstaben N beginnt. Wozu also den gesamten Wortschatz durchsuchen, der ja eh alphabetisch geordnet ist? Es würde doch vollkommen ausreichen, den Bereich, in dem alle mit N beginnenden Wörter stehen, mittels der binären Suchmethode zu durchsuchen! Dazu benötigt man zunächst einen Index – eine Tabelle, in der steht, ab welchem Satz die mit A beginnenden Wörter stehen, ab welchem die mit B beginnenden, und so weiter. Das Programm in Listing 7 ist in der Lage, den Index zu einem beliebigen Bereich einer relativen Datei (der natürlich alphabetisch geordnet sein muß) zu erstellen.

Das Programm ist anwenderfreundlich – es erkennt unlogische Eingaben und erklärt sich nach Programmstart selbst.

Es lohnt sich eigentlich gar nicht, den Index-File jedesmal erst lange von Disk zu laden. Es ist günstiger, die Werte einfach in DATA-Zeilen im Listing abzulegen.

Durch Anwendung der Index-Binären-Methode sind nur wenig Lesezugriffe auf Disk notwendig, bis ein Wort gefunden ist: Die größte Wortgruppe unserer Wortschatzdatei stellen die Wörter dar, die mit S beginnen: Das sind 33 Worte. Im ungünstigsten Fall werden nur 6 Zugriffe benötigt, um ein Wort zu finden. Vorteilhaft an der neuen Methode ist außerdem, daß Wörter, die mit einem Buchstaben beginnen, der im Wortschatz als Anfangsbuchstabe überhaupt nicht auftaucht (zum Beispiel C,J,M,X und Y), erst gar nicht gesucht werden, weil von vorneherein klar ist, daß sie im Wortschatz nicht auftreten.

Bei Methode 4 wird allerdings auch noch etwas anderes verbessert: Das Zerlegen von `BE$` in `BE$(0)–BE$(N)`. Eigentlich ist es nämlich Quatsch, `BE$` in einzelne Worte zu zerlegen. Es reicht völlig aus, mit Zeigern zu arbeiten. Bild 9 verdeutlicht dies.

Das mit den Zeigern funktioniert ganz einfach: Zu Beginn wird Zeiger A auf das erste Zeichen von `BE$` gesetzt. Zeiger B läuft von Zeiger A aus nach rechts bis zum nächsten Space. Das erste Suchwort ist somit festgelegt. Nachdem dieses Wort codiert ist, wird Zeiger A auf das Zeichen rechts neben Zeiger B gesetzt – auf das erste Zeichen des neuen Suchwortes. Jetzt läuft Zeiger B wieder nach rechts und sucht erneut das Wortende, und so weiter.

Trifft Zeiger B auf kein Zeichen mehr, so ist er am Ende des Satzes angelangt – der Satz ist komplett codiert.

Damit das letzte Wort der Sätze noch codiert wird, muß am Befehlssatz `BE$` noch ein SPACE angehängt werden.

Bei Methode 4 soll auch noch etwas anderes berücksichtigt werden – das Satzende. Es gibt 3 Möglichkeiten, einen Satz zu beenden – mit einem Punkt, einem Ausrufezeichen, einem Fragezeichen oder einfach nichts.

Uns interessieren allerdings nur zwei Fälle: Fragezeichen oder nicht. Bevor `BE$` also codiert wird, muß künftig darauf geachtet werden, ob am Ende des Satzes ein Satzzeichen steht. Die Variable SN speichert das Satzzeichen (Punkt etc. = 0, Fragezeichen = 1). Dann wird `BE$` so gekürzt, daß das Zeichen wegfällt.

Nach dem Codiervorgang müssen folgende Werte in Form von Variablen vorliegen:

SN	Satzzeichen am Satzende
VE	Verbzahl
O	Objektzahl für erstes Objekt
O2	Objektzahl für zweites Objekt (falls vorhanden)
RI	Richtung
UD	UND-Speicher
AD	Adjektiv-Zahl

```

10 OPEN 1,8,3,"WORTSCHATZ":OPEN 15,8,15:RE
  M WORTSCHATZ-DATEI OEFFNEN <125>
20 DATA 1,3,0,28,31,47,51,60,68,0,70,76,0,
  82,90,93,100,101,116,149,155,158 <160>
30 DATA 178,0,0,189 <193>
35 DATA 198:REM BUCHSTABE NACH Z !?! <235>
40 DIM IN(26):FOR I=0 TO 26:READ IN(I):NEX
  T I <193>
50 GOSUB 50000 <062>
60 PRINT"SN="SN:PRINT"VE="VE:PRINT"O1="O1:
  PRINT"O2="O2:PRINT"UD="UD:PRINT"RI="RI <017>
70 PRINT"AD="AD:PRINT:GOTO 50 <111>
50000 REM ***** <227>
50001 REM * <009>
50002 REM * WORT-PARSER 4.0 * <169>
50003 REM * <011>
50004 REM * <C> 1986 BEI * <086>
50005 REM * <013>
50006 REM * MICHAEL NICKLES * <082>
50007 REM * <015>
50008 REM ***** <235>
50010 REM BEFEHLSSATZEINGABE -----
-----
50011 : <069>
50012 IF UD>0 THEN 50500:REM UND <202>
50015 SL=80:REM BEFEHLSSATZLAENGE <202>
50020 PRINT"DOWN":BE$="":POKE 198,0:POKE
  211,0:POKE 214,22:SYS 58732:PRINT"
  YELLOW"; <156>
50030 GET X$:IF PEEK(203)=1 THEN 50120 <226>
50040 IF X$="" THEN 50030 <171>
50050 IF LEN(BE$)=0 AND ASC(X$)=20 THEN 50
  030 <148>
50060 I=ASC(X$):IF I<32 OR I>133 AND I<159
  THEN IF I<>20 THEN 50030 <131>
50070 IF LEN(BE$)=SL AND I<>20 THEN 50030 <128>
50080 BE$=BE$+X$ <031>
50090 PRINT CHR$(20);X$;" "; <108>
50100 IF I=20 THEN BE$=LEFT$(BE$,LEN(BE$)-
  2):GOTO 50030 <101>
50110 GOTO 50030 <001>
50120 PRINT"LEFT,SPACE" <227>
50400 REM SATZZEICHEN SUCHEN -----
-----
50401 : <103>
50415 SN=0 <082>
50420 IF RIGHT$(BE$,1)=" " THEN BE$=LEFT$(B
  E$,LEN(BE$)-1):GOTO 50420 <233>
50430 I$=RIGHT$(BE$,1) <180>
50440 IF I$="," OR I$="!" THEN SN=0:BE$=LEFT
  $(BE$,LEN(BE$)-1):GOTO 50420 <010>
50450 IF I$="?" THEN SN=1:BE$=LEFT$(BE$,LEN
  (BE$)-1):GOTO 50420 <144>
50500 REM BE$-ZEIGER SETZEN -----
-----
50501 : <199>
50502 IF UD=0 THEN VE=0:RI=0:O1=0:O2=0 <184>
50504 IF UD>0 THEN RI=0:GOTO 50530 <012>
50505 IF LEFT$(BE$,1)=" " THEN BE$=RIGHT$(B
  E$,LEN(BE$)-1):GOTO 50505 <132>
50510 BE$=BE$+" " <232>
50520 ZA=1:REM ZEIGER A SETZEN <058>
50530 ZB=ZA+1 <180>
50540 IF MID$(BE$,ZB,1)=" " THEN UD=0:RETURN <213>
50550 IF MID$(BE$,ZB,1)<>" " THEN ZB=ZB+1:G
  OTO 50550 <175>
50590 SU$=MID$(BE$,ZA,ZB-ZA) <222>
50600 REM SU$ VORBEHANDLUNG -----
-----
50601 : <130>
50610 IF SU$="UND" THEN UD=1:GOTO 50630 <028>
50620 IF SU$="," THEN UD=2:GOTO 50630 <093>
50625 GOTO 50670 <061>
50630 ZA=ZB+1 <201>
50640 IF MID$(BE$,ZA,1)=" " THEN PRINT"UND W
  AS ?":GOTO 50530 <080>
50650 IF MID$(BE$,ZA,1)=" " THEN ZA=ZA+1:GO
  TO 50650 <153>
50660 RETURN <242>
50670 IF SU$="IHN" OR SU$="SIE" OR SU$="ES" T
  HEN O1=OM:GOTO 51300 <169>
51000 REM BINAERE SUCHROUTINE -----
-----
51001 : <113>

```

```

51007 REM ANFANG UND ENDE DES SUCHBEREICHE
  S ERMITTELN <176>
51008 I=ASC(LEFT$(SU$,1)) <117>
51009 IF I-65<0 OR I-65>25 THEN GOSUB 5150
  0:GOTO 50000 <176>
51010 AN=IN(I-65):REM ANFANG DES SUCHBEREI
  CHES <134>
51012 IZ=64 <164>
51013 IF IN(I-IZ)=0 THEN IZ=IZ-1:GOTO 5101
  3 <200>
51015 EN=IN(I-IZ)-1:REM ENDE DES SUCHBEREI
  CHES <086>
51016 IF AN=0 THEN GOSUB 51500:UD=0:GOTO 5
  0000 <070>
51020 SZ=197:N=INT(LOG(EN-AN+1)/LOG(2))+1
  :REM MAXIMAL-FORMEL <249>
51030 SA=AN-1+(2↑N)/2:REM MITTE DER GES.DA
  T <121>
51040 GOSUB 52100:REM A$ LESEN <073>
51050 N=N-1:REM 1.ABFRAGE <234>
51060 : <045>
51070 REM SU$ MIT A$ VERGLEICHEN ----- <235>
51080 : IF LEN(SU$)<3 THEN 51110 <042>
51090 : IF WA=1 AND SU$=LEFT$(A$,LEN(SU$))
  THEN VE=WC:GOTO 51300 <140>
51100 : IF WA<>3 OR SU$<>RIGHT$(A$,LEN(SU$
  )) THEN 51110 <157>
51102 : IF UD=1 OR UD=2 THEN UD=3:O1=0:O2
  =0 <161>
51104 : IF O1=0 THEN O1=WC:OM=O1:GOTO 51
  300 <042>
51105 : IF O2=0 THEN O2=WC:OM=O1:GOTO 51
  300 <133>
51110 : IF WA=2 AND SU$=A$ <170>
  THEN RI=WC:GOTO 51300 <140>
51115 : IF WA=5 AND SU$=A$ <247>
  THEN AD=WC:GOTO 51300
51120 : IF WA=1 AND SU$=A$ <017>
  THEN VE=WC:GOTO 51300
51125 : IF WA=4 AND SU$=A$ <204>
  THEN 51300
51130 : IF SU$<A$ THEN SA=SA-(2↑(N-1)):GOSU
  B 52100:GOTO 51160 <015>
51140 : IF SA+(2↑(N-1))>SA+EN-AN AND N>-1
  THEN N=N-1:GOTO 51140:REM 'UEBERLAU
  F' <204>
51150 : SA=SA+(2↑(N-1)):GOSUB 52100:REM '
  HALBIEREN' <141>
51160 : <079>
51170 N=N-1:REM NAECHST. HALBIEREN <068>
51180 IF N<0 THEN GOSUB 51500:UD=0:GOTO 5
  0000 <201>
51190 REM PRINT"SA=";SA;" N=";N
  :REM UNNOETIG !! <216>
51200 GOTO 51060:REM NAECHST. SUCH-VERS. <192>
51300 REM NAECHSTES WORT <027>
51310 ZA=ZB+1 <254>
51320 IF MID$(BE$,ZA,1)=" " THEN 50530 <128>
51330 IF MID$(BE$,ZA,1)=" " THEN ZA=ZA+1:GO
  TO 51330 <207>
51350 GOTO 50530 <131>
51500 REM ICH KENNE SU$ NICHT -----
-----
51501 : <232>
51510 I=INT(3*RND(1))+1 <168>
51520 ON I GOTO 51530,51540,51550 <080>
51530 X$=" "+ICH KENNE DAS WORT "+SU$+"
  NICHT.":GOTO 52000 <173>
51540 X$=" "+DAS WORT "+SU$+" IST MIR NI
  CHT BEKANNT.":GOTO 52000 <002>
51550 X$=" "+DEUTSCH IST LEIDER NUR MEINE
  ZWEITE SPRACHE. ICH KENNE DAS " <113>
51555 X$=X$+"WORT "+SU$+" NICHT.":GOTO 5
  2000 <038>
52000 REM FORMATIERTE TEXTAUSGABE -----
-----
52010 IF X$="" THEN 52090 <120>
52020 IF LEN(X$)<40 THEN PRINT X$:GOTO 520
  90 <213>
52030 IF LEN(X$)=40 THEN PRINT X$:GOTO 52
  090 <248>
52040 I$=LEFT$(X$,41):IC=0:FOR I=1 TO LEN(
  I$):IF MID$(I$,I,1)=" " THEN IC=I <148>

```

```

52050 NEXT I: IF IC=0 THEN PRINT X$: GOTO 52
090                                <064>
52060 I$=LEFT$(I$,IC-1): IF LEN(I$)=40 THEN
PRINT I$:                                <046>
52070 IF LEN(I$)<40 THEN PRINT I$        <116>
52080 X$=RIGHT$(X$,LEN(X$)-IC): GOTO 52010 <206>
52090 RETURN                                <075>
52100 REM SATZ-TEILBEREICH LESEN -----
-----
52101 :                                <254>
52110 HB%=SA/256: LB%=SA-HB%*256        <004>
52112 PRINT#15,"P";CHR$(3);CHR$(LB%);CHR$(
HB%);CHR$(1)                                <241>
52114 INPUT#15,A: IF A=50 THEN PRINT"SATZNU
MMER ZU HOCH"                                <170>
52120 IF A=50 THEN RETURN                <245>
52130 INPUT#1,A$                            <165>
52140 WA=VAL(MID$(A$,22,1))                <199>
52150 WC=VAL(MID$(A$,23,3))                <030>
52160 A$=LEFT$(A$,21)                        <089>
52170 IF RIGHT$(A$,1)="(RIGHT)" THEN A$=LEF
T$(A$,LEN(A$)-1): GOTO 52170                <202>
52180 RETURN                                <154>
                                <165>

```

Listing 8. Methode 4 als Programm

Zur Objektcodierung werden zwei Variablen O1 und O2 benötigt, da ein Befehl aus maximal zwei Objekten (direktes und indirektes) bestehen kann. Beispiel: »ÖFFNE DIE TÜR MIT DEM SCHLÜSSEL«. Im Befehl kann auch nur ein Verb auftreten. Gegebenenfalls kann auch noch eine Variable AD für Adjektiv zur näheren Beschreibung der Objekte verwendet werden. Adjektive haben die Wortzahl WZ=5.

Jetzt kann Methode 4 wie in Listing 8 programmiert werden:

Die REM-Zeilen können beim Abtippen nicht so ohne weiteres weggelassen werden!

Mit REM-Zeilen beansprucht Parser 4.0 zirka 3 KByte RAM Basic-Speicher. Entfernt man die REM-Zeilen unter Berücksichtigung der dann teils veränderten Anspruchsadressen, so werden nur zirka 2 KByte Basic-RAM benötigt.

Parser 4.0 ist ziemlich intelligent und zum größten Teil sogar idiotensicher. Das heißt, es werden die blödsinnigsten Befehlseingaben akzeptiert. Zum Beispiel:

"NIMM DAS SCHWERT"

" NIMM DAS SCHWERT!"

"NIMM SCHWERT, UNTERSUCHE ES"

Die UND-Routine ist auch eingebaut. Außerdem speichert Parser 4.0 jeweils das zuletzt eingegebene Objekt in der Variablen OM. Bei Eingabe eines der Schlüsselwörter IHN, SIE oder ES (zum Beispiel BE\$=»UNTERSUCHE ES«), so wird O gleich OM gesetzt. Der Clou ist auch, daß Sie einfach nur GOSUB 50000 eingeben müssen, wenn vom Spieler eine Befehlseingabe erwartet wird. Parser 4.0 erkennt automatisch, ob die INPUT-Routine angesprochen werden muß oder ob der letzte Befehlssatz noch nicht ganz abgehandelt ist (UND-FALL). Kann Parser 4.0 ein Wort nicht codieren, weil es im Wortschatz nicht vorkommt, so erfolgt »ICH KENNE DAS WORT NICHT«, und anschließend wird zur Befehlseingabe-Routine zurückgesprungen. Dies ist notwendig, damit ein Spieler niemals verlieren kann, wenn er bei einer entscheidenden Situation (zum Beispiel Kampf mit Monster) sich aus Versehen vertippt (zum Beispiel TOTE MONSTER statt TOETE MONSTER). Das Programm darf also erst dann weiterlaufen, wenn der Spieler eine korrekte Befehlseingabe gemacht hat.

Nach dem Rücksprung aus Parser 4.0 liegt der Satz codiert in den Variablen SN,VE,O1,O2,UD,AD vor. Natürlich können Sie Parser 4.0 jetzt noch nicht gründlich testen, da im Wortschatz momentan noch die Objekte fehlen. Zunächst rate ich Ihnen, die nun folgende Dokumentation gründlich zu studieren, damit Sie in der Lage sind, Parser 4.0 selbst zu modifizieren – denn dies ist beim Einbau in eigene Programme fast immer notwendig!

Dokumentation zu Parser 4.0

- 10 Hier wird die Wortschatz-Datei geöffnet. Das heißt es muß die Diskette eingelegt sein, auf der sich der Wortschatz befindet.
- 20-30 Hier stehen die DATAs zum Wortschatz-Index, die mit dem Index-Maker ermittelt wurden.
- 35 Tatsächlich besteht die Index-Tabelle nicht nur aus 26, sondern 27 Elementen. Das 27. Element steht für den Buchstaben, der nach Z kommt. Richtig, nach Z kommt kein Buchstabe mehr. Das 27. Element der Index-Tabelle errechnet sich vielmehr aus ANZAHL WÖRTER DER WORTTABELLE +1. Dies ist nötig, damit das Programm erkennt, von welchem bis zu welchem Satz die mit Z beginnenden Wörter zu finden sind. Der 27. Wert wird NICHT automatisch vom Index-Maker erstellt. Er muß als »von Hand« berechnet und eingesetzt werden.
- 40 Einlesen der Index-Datas in das Index-Feld IN(0)-IN(26)
- 50-70 Hier wird Parser 4.0 mittels GOSUB 50000 aufgerufen. Danach werden die codierten Werte, SN, VE, O1, O2, UD, RI, AD ausgegeben. Dann erfolgt Rücksprung zum Parser-Aufruf. Die Zeilen 50-70 dienen also nur zum Testen des Parsers.
- 50000-50120 Hier erfolgt die Befehlseingabe in Form einer erweiterten INPUT-Routine, wie ich sie Ihnen im Kursabschnitt Datei-Verwaltung bereits vorgestellt habe.
- 50012 Hier wird überprüft, ob die letzte Befehlseingabe bereits voll codiert worden ist oder ob sie nach einem UND abgebrochen wurde. ZB BE\$=»NIMM SCHWERT UND GEH NORDEN« wobei erst der Teilbefehl »NIMM SCHWERT« ausgeführt worden ist. Ist der Befehlssatz BE\$ also noch nicht komplett codiert, so wird die Befehlseingabe (und natürlich auch die Suche nach dem Satzzeichen, die ja bereits erfolgt ist) übersprungen, alle bisherigen codierten Werte VE,O1 etc. werden beibehalten, und es wird dort weiter codiert, wo zuletzt abgebrochen wurde – GOTO 50500.
- 50400-50450 Hier wird nach einem Satzzeichen am Ende von BE\$ gesucht.
- 50420 Alle unnötigen Leerzeichen, die dem Befehlssatz BE\$ anhängen, werden weggekürzt. Beispiel: AUS BE\$=»NIMM SCHWERT ! « wird BE\$=»NIMM SCHWERT!«.
- 50440-50450 Wird ein Satzzeichen gefunden, so wird es in der Variablen SN gespeichert (interessant ist dabei eher nur das Fragezeichen!). Dann wird das Satzzeichen weggekürzt. Also aus »NIMM SCHWERT ! « wird jetzt »NIMM SCHWERT«. All dies ist notwendig, damit bei einer Befehlseingabe BE\$=»NIMM SCHWERT!« nicht die Fehlermeldung »Ich kenne das Wort SCHWERT!« nicht auftritt. Denken Sie also stets daran, alle unnötigen Leerzeichen vor oder nach dem Befehl zu eliminieren. Einen Parser zu programmieren, ist wirklich eine verflixte Sache: Es gibt so viele Dinge, die berücksichtigt werden müssen, damit der Parser auch wirklich absturzfähig läuft. Das Wegkürzen der Leerstellen ist ein gutes Beispiel hierfür. Ein Tip: Wenn Sie glauben, den Parser endlich fertig zu haben, dann compilieren Sie ihn mit Hilfe eines Compilers wie zum Beispiel Austro Comp oder Pet Speed. Der Compiler überprüft alle Zeichen und weist auf eventuelle Syntax-Errors hin. Wird ein Parser-Programm fehlerfrei compiliert, so können Sie davon ausgehen, daß im Programm keine schwerwiegenden Fehler vorkommen!

- 50500-50600 Hier werden die Zeiger ZA und ZB auf den Befehlssatz BE\$ gesetzt oder weitergerückt.
- 50502 Wurde beim letzten Vorgang nicht wegen UND abgebrochen, so werden alle bisher codierten Werte auf 0 gesetzt.
- 50504 Wird hier festgestellt, daß der letzte Befehlssatz noch nicht komplett codiert worden ist (UD größer 0), so passiert folgendes: Die Richtungszahl RI wird auf 0 gesetzt (warum, erfahren Sie später noch!). Dann erfolgt ein Sprung zu Zeile 50530. Das heißt, alle Operationen oder Einstellungen, die vorgenommen werden, wenn ein neuer Befehlssatz codiert werden soll (also kein wegen UND abgebrochener), werden übersprungen.
- 50505-50520 Einstellung der Zeiger auf einen neuen Befehlssatz.
- 50505 Alle Leerstellen, die dem eigentlichen Befehl voranstehen, werden entfernt. Beispiel: Aus » NIMM SCHWERT« wird »NIMM SCHWERT«.
- 50510 BE\$ wird ein Leerzeichen angehängt. Aus BE\$ = »NIMM SCHWERT« wird also BE\$ = »NIMM SCHWERT «. Dies ist nötig, da später beim Suchvorgang das Auffinden eines Leerzeichens als Markierung eines Wortendes interpretiert wird.
- 50520 Zeiger ZA wird auf das erste Zeichen von BE\$ gesetzt.
- 50530 Zeiger ZB wird auf das Zeichen nach ZA gesetzt.
- 50540 Ist MID\$(BE\$,ZB,1) = " ", so wurde das Satzende erreicht. Der Codiervorgang wird mittels RETURN abgebrochen.
- 50550 Ist MID\$(BE\$,ZB,1) kein Leerzeichen, so wird Zeiger ZB um ein Zeichen nach rechts gerückt und es erfolgt ein Rücksprung zu Zeile 50550. Zeiger ZB wird also so lange nach rechts geschoben, bis entweder das Satzende oder ein Leerzeichen gefunden wird. Wird ein Leerzeichen gefunden, so ist das neue Suchwort festgelegt: SU\$=MID\$(BE\$,ZA,ZB-ZA).
- 50590 Das neue Suchwort, das von den Zeigern ZA und ZB auf BE\$ bestimmt ist, wird im String SU\$ abgelegt.
- 50600-50670 Es erfolgt eine »Vorbehandlung« des Suchwortes SU\$. Hierbei werden solche Wörter gesucht, die Parser-intern gespeichert sind und nicht auf der Wortschatz-relativen Datei verzeichnet sind.
- 50610 Hier wird überprüft, ob SU\$ gleich UND ist. Wenn ja, so ist klar, daß ein Teilbefehl von BE\$ codiert ist und später von hier aus weitercodiert werden muß. Zunächst muß jedoch der bisherig ermittelte Teilbefehl analysiert werden. Dazu wird die Variable UD auf 1 gesetzt und der Codiervorgang mittels RETURN abgebrochen.
- 50620 Das gleiche wie bei 50610 in Grün - nur, daß hier nach einem Komma gesucht wird. Die Komma-Suche hat allerdings einen Haken: Das Komma wird nur dann gefunden, wenn links und rechts von ihm mindestens je ein Leerzeichen steht. Bei Befehlssätzen wie zum Beispiel BE\$ = »NIMM SCHWERT,BIERFLASCHE« erfolgt eine Fehlermeldung à la »Ich kenne das Wort Bierflasche nicht«. Mit einem direkten Vergleich IF SU\$ = ", " ist es also nicht abgetan. Um den Parser nicht unnötig kompliziert zu machen, habe ich deshalb auf diese umfangreiche Abfrage verzichtet. Wie umfangreich diese Abfrage ist, werden Sie spätestens dann bemerken, wenn Sie versuchen, sie zu programmieren. Ich rate Ihnen, bei einem Basic-Parser auf diese Abfragerei zu verzichten, denn sie kostet viel zuviel kostbare Rechenzeit, und der Parser wird zu langsam!
- 50630-50660 Diese Zeilen treten dann in Kraft, wenn ein UND oder ein Komma gefunden worden ist. Zeiger ZA wird auf eine Stelle nach ZB gesetzt.
- 50540-50550 Hier erfolgt das bereits beschriebene Weiterrücken von Zeiger ZB nach rechts, bis ein Leerzeichen oder das Satzende gefunden wird. Wird hier ein Satzende festgestellt, so heißt dies, daß nach dem UND oder Komma nichts mehr kommt. Der Parser äußert sich im Fall zum Beispiel BE\$ = »NIMM SCHWERT UND« empört mit »Und was?«, wobei er wieder einen Hauch von Intelligenz beweist und den Spieler in Erstaunen versetzt ...
- 50670 Wird eines der Schlüsselwörter IHN, SIE oder ES gefunden, so weiß der Parser sofort, daß der Spieler sich auf das zuletzt erwähnte Objekt bezieht. Das zuletzt erwähnte Objekt wird immer in der Variablen OM gespeichert. Der Parser braucht hier also nichts weiter zu tun, als der Objektzahl 01 den Wert OM zuzuweisen und dann die binäre Suchroutine zu überspringen mit GOTO 51300.
- 51000-51330 Hier steht die binäre Suchmethode, mit deren Hilfe SU\$ in der relativen Wortschatzdatei gesucht wird.
- 51008 Der ASC-Wert des 1. Buchstabens vom Suchwort SU\$ wird in der Variablen I gespeichert.
- 51009 Wenn I nicht der ASC-Wert eines der Buchstaben von A-Z ist, also I-65 kleiner 0 oder I-65 größer 25, dann ist klar, daß das Suchwort nicht mit einem Buchstaben, sondern mit irgendeinem Zeichen beginnt, zum Beispiel SU\$ = "/%&%/ ". Es erfolgt dann eine »ICH KENNE DIESES WORT NICHT«-Meldung, und der Codiervorgang wird abgebrochen.
- 51010 Der Anfang des zu durchsuchenden Bereiches wird mittels der Index-Tabelle IN(0)-IN(26) ermittelt und in AN gespeichert.
- 51012-51015 Hier wird das Ende des Suchbereiches, EN ermittelt. Das Ende berechnet sich NICHT einfach durch EN=IN(I-64), da berücksichtigt werden muß, daß in der Index-Tabelle für nicht existierende Anfangsbuchstaben (zum Beispiel C) eine Null eingetragen ist. Es darf also nicht der unmittelbare nächste Buchstabe zur Ermittlung des Suchendes herangezogen werden, sondern vielmehr der nächste existente Buchstabe (Tabellenwert ungleich 0).
- 51016 Wenn AN=0 ist, so beginnt das Suchwort SU\$ mit einem Buchstaben, mit dem kein Wort des Wortschatzes beginnt. Es braucht dann gar nicht erst lange gesucht werden - Fehlermeldung ICH KENNE WORT NICHT.
- 51020-51200 Hier steht die binäre Suchroutine, die ich Ihnen bereits im Kapitel Dateiverwaltung beschrieben habe. Allerdings ist die Routine diesmal so modifiziert, daß sie mit relativen Dateien arbeitet und nicht mit Stringfeldern im Speicher.
- 52100-52180 Dieses Unterprogramm wird innerhalb der binären Suchroutine angesprungen. Es dient dazu, um aus einer relativen Datei einen Satz zu lesen. Diese Routine ist im wesentlichen eine Kombination unserer bisher verwendeten Unterprogramme zum Positionieren auf (GOSUB 9000) und Lesen (GOSUB 9100) eines Satzes aus einer relativen Datei.
- 52100-52130 Positionieren und anschließendes Lesen des Satzes A\$.
- 52140 Die Wortart WA wird ermittelt.
- 52150 Der Wortcode WC wird ermittelt.
- 52160 AS wird auf das eigentliche Wort gekürzt. Aus A\$=»abwaerts 2010« wird A\$=»abwaerts «.
- 52170 Alle unnötigen Leerstellen in A\$ werden gekürzt. Aus A\$=»abwaerts « wird also A\$=»abwaerts«. Dies ist eigentlich nur dann nötig, wenn es sich bei A\$

um ein Objekt handelt (WA=3), denn schließlich werden nur Objekte »rechtsbündig« verglichen beim Suchvorgang. Wer will, kann folgende Zeile einfügen (ohne Gewähr):

```
52165 IF WA=3 THEN RETURN
```

Warum ohne Gewähr? Während man einen Parser programmiert oder dokumentiert, entdeckt man ständig kleine Dinge, die man noch zu verbessern glaubt. Schnell sind drei, vier Änderungen gemacht, und plötzlich läuft der Parser nicht mehr so recht, obwohl alle Änderungen für sich allein betrachtet doch eigentlich funktionieren sollten!?! Ich rage Ihnen daher, an einem Parser-Programm nie mehr als eine Änderung auf einmal vorzunehmen und dann auch gleich alle Fälle auszuprobieren, um sicherzustellen, daß auch alles noch funktioniert...

51070-51125 Hier werden die einzelnen Wortzahlen VE,01 etc. ermittelt. Es wird also genau hier codiert.

51080 Verben werden, wie bereits erklärt, linksbündig gesucht beim Vergleich – also IF SU\$=LEFT\$(A\$, LEN(SU\$))...

Es sind somit auch Abkürzungen legitim – zum Beispiel NIM statt NIMM oder INV statt INVENTUR. Eine Abkürzung muß allerdings mindestens 3 Zeichen lang sein. Ist die Abkürzung kleiner als 3 Zeichen, so werden Versuche und Objektsuche übersprungen und der Parser stellt sich darauf ein, eine Richtungsangabe zu finden. Würde man auf diese Abfrage verzichten, so wüßte der Parser nicht, wie er sich im Fall einer Befehlseingabe BE\$=»N« verhalten soll. Er würde als Verbzahl VE das nächstbeste Verb ermitteln, das mit N beginnt, und anschließend auch noch die Richtungsangabe beziehungsweise Richtungsanzahl RI für die Richtung Norden ermitteln.

51090 Hier erfolgt die linksbündige Versuche.

51100 Hier erfolgt die Objektsuche – und zwar wie bereits erklärt rechtsbündig, damit zum Beispiel sowohl HOLZKISTE als auch KISTE verstanden werden, wenn der Objektname HOLZKISTE lautet.

51102 Jetzt wird's kompliziert zu erklären!

Es wird überprüft, ob bei der letzten Codierung wegen Auffinden eines UNDS beziehungsweise Kommas abgebrochen wurde (UD größer 0). Ist dies der Fall, so werden 01 und 02 gelöscht und UD auf 3 gesetzt, damit beim Auffinden eines zweiten Objekts nicht wieder gelöscht wird.

51104-51105 In einem Befehlssatz können maximal zwei Objekte gleichzeitig auftreten (beziehungsweise Teilbefehl) – vergleiche Wortschatzlehre! Es müssen also zwei Variablen zum Speichern von Objekten zur Verfügung stehen – 01 und 02.

Nehmen wir einmal BE\$=»ÖFFNE TÜR MIT DEM SCHLÜSSEL UND BETRETE DAS ZIMMER« an.

Zuerst wird ÖFFNE im Wortschatz gesucht und eine Verbzahl VE gefunden. Dann wird das Wort TÜR gesucht und gefunden. Das Objekt TÜR wird codiert in der 1. Objektzahl 01 gespeichert.

Das Wort MIT wird nicht berücksichtigt (ich setze hier einmal voraus, daß es im Wortschatz als AUS-Wort aufgeführt ist).

Jetzt wird das Objekt SCHLÜSSEL gesucht und gefunden. In der Variablen 01 kann sein Code nicht gespeichert werden, da 01 bereits belegt ist. In 02 ist allerdings noch Platz. Dem Parser ist also klar, daß ein zweites Objekt gefunden wurde und dies in 02 gespeichert werden muß. Nehmen wir jetzt einmal den unmöglichen Fall an, daß in einem Befehlssatz oder Teilbefehl 3 Objekte auftauchen. Das dritte Objekt wird vom Parser dann eiskalt ignoriert, und es

erfolgt keine Fehlermeldung. Wenn in einem Befehlssatz 3 Objekte verwendet werden, so ist klar, daß der Befehlssatz nur unsinnig sein kann, zum Beispiel »ERSCHLAGE MONSTER MIT DEM SCHWERT DAS AUF DEM STUHL LIEGT«. Ok, eigentlich ergibt dieser Satz einen Sinn, aber wir wissen, daß dies nicht die Eingabe-Syntax eines routinierten Adventure-Spielers ist. Man dürfte es dem Parser also nicht übel nehmen, wenn er sich mit »ICH VERSTEHE SIE NICHT« meldet. Man könnte also (wieder ohne Gewähr) noch folgende Zeile einbauen:

```
51103 IF 01<>0 AND 02<>0 THEN? »ICH VERSTEHE SIE NICHT«:GOTO50000.
```

Als nächstes Wort wird UND gefunden. Die Variable UD wird auf 1 gesetzt und der Codierungsvorgang abgebrochen. Der erste Teilbefehl des gesamten Befehlssatzes ist somit codiert und liegt anhand der Variablen VE,01,02 und sekundär UD vor.

Nach Analyse dieses Teilbefehls wird die Codierung fortgesetzt. Zunächst wird BETRETE als neues VERB ermittelt. Aus der Wortschatzlehre wissen Sie bereits, daß ein Befehlssatz maximal ein Verb enthalten kann. Stehen unsinnigerweise doch mal zwei Verben in einem Befehl, so speichert Parser 4.0 den Wert des Verbs in VE ab, das am weitesten rechts im Befehl steht.

Jetzt wird nach Ignorieren von DAS das neue Objekt ZIMMER gesucht. Wäre Zeile 51102 nicht vorhanden, so würde der Parser dieses Objekt nicht mehr codieren können, da erst die beiden Werte 01 und 02 gelöscht werden müssen.

Jetzt liegt der zweite Teilbefehl von BE\$ codiert vor. Und zwar diesmal in den Variablen VE und 01.

51110 Hier wird eine Richtungsangabe absolut gesucht.

51115 Adjektivsuche – ebenfalls absolut.

Zur Adjektivcodierung wurde nur die Variable AD reserviert. Ein Befehl kann also nur maximal ein Adjektiv enthalten. Dies langt in der Regel. Wer jedoch Adventures programmieren will, bei denen extrem komplizierte Befehlseingaben als BE\$=»ÖFFNE DIE GRÜNE TÜR MIT DEM GELBEN SCHLÜSSEL« erforderlich sind, der muß hier entsprechend modifizieren (analog zur Objektcodierung mittels 01 und 02).

51120 Absolute Versuche. Hier werden Verben gefunden, deren Satzlänge weniger als 3 Zeichen beträgt. Wollen Sie zum Beispiel I als Abkürzung für INVENTUR erlauben, so können Sie I extra in der Wortschatzdatei als Verb eintragen.

51125 AUS-Wörter wie der, die, das werden absolut gesucht und gegebenenfalls ignoriert.

51300-51350 Weiterrücken der Zeiger ZA und ZB auf nächste Wort. Eine neue Suchrunde wird eingeläutet.

51500-51555 Kleine Bierbraukunde

52000-52090 Wird hier endlich geklärt, was Lolo, Peter und Mike im Wald machen?

Variablenliste

IN(0) bis IN(26) Index-Tabelle

UD UND-Variable (größer 0 bei Codierungsabbruch wegen UND oder)

SL Maximal erlaubte Befehlssatzlänge (max. 255 Zeichen!)

BE\$ Gesamter Befehlssatz

X\$ Hilfsstring

I Hilfsvariable

SN Satzzeichenwert

I\$ Hilfsstring

VE Verbzahl

RI Richtungsanzahl

O1 1. Objektzahl

O2 2. Objektzahl
 ZA,ZB Zeiger zum Eingrenzen des Suchwortes SU\$ in BE\$
 SU\$ Suchwort
 IZ Hilfsvariable
 AN Anfang des Suchbereichs
 EN Ende " "
 SZ Anzahl der Worte in der Wortschatzdatei
 N Hilfsvariable
 SA Satz einer relativen Datei (Nummer)
 WA Wortart
 WC Wortcode
 HB% Higher Byte
 LB% Lower Byte
 A Fehlerkanal-Zahl
 A\$ Satz, der von relativer Datei gelesen wurde.

Das war's auch schon. Anhand dieser umfangreichen Dokumentation sollte es Ihnen nun nicht mehr schwerfallen, sich mit Parser 4.0 zurechtzufinden und eigene Änderungen am Programm durchzuführen. Mit Sicherheit haben Sie bald so manche Idee, was man noch verbessern kann. Ich rate Ihnen auf jeden Fall, an dieser Stelle erst dann mit dem folgenden Kursabschnitt fortzufahren, wenn Sie Parser 4.0 verstanden haben.

Für Profis: Parser 4.1

Parser 4.1 ist eine Verbesserung beziehungsweise eine schnellere Version von Parser 4.0. Das nun Folgende soll Ihnen lediglich als theoretisches Gedankengut dienen, wenn Sie Parser 4.0 selbständig verbessern wollen.

Parser 4.0 basiert auf dem index-binären Prinzip. Der Wortschatz muß daher alphabetisch geordnet sein, damit eine Index-Tabelle IN(0)-IN(26) erstellt werden kann. Aus dieser Tabelle kann man schnell erlesen, in welchem Satzbereich der relativen Datei Wörter mit gleichem Anfangsbuchstaben stehen. Durch diese Eingrenzung des Suchbereiches werden einzelne Suchwörter mit relativ wenigen Diskzugriffen gefunden. Die Idee von Parser 4.1 ist wie folgt: Auch hier findet die index-binäre Methode Verwendung. Allerdings wird diesmal nicht eine Indextabelle IN(0)-IN(26) verwendet, sondern vielmehr ein Index-Feld, das so angelegt wird:

DIM IN%(26,26)

In IN%(0,0)-IN%(26,0) steht hierbei wieder die gleiche Index-Tabelle, die bei Parser 4.0 in IN(0)-IN(26) steht.

Jetzt spielt allerdings auch noch der 2. Buchstabe des Suchwortes eine Rolle bei der Einschränkung des Suchbereiches.

In IN%(0,0)-IN%(0,26) zum Beispiel ist die Index-Tabelle des zweiten Buchstabens aller Wörter, die mit A beginnen, gespeichert.

Das heißt, in IN%(0,0) steht, wo alle Wörter der Wortschatzdatei beginnen, deren erste beide Buchstaben AA sind. In IN%(0,1) entsprechend AB, in IN%(1,3) entsprechend BD etc. Anhand eines Index-Feldes kann der Suchbereich folglich noch viel feiner eingeschränkt werden als bei einer Index-Tabelle. Das heißt, es werden weniger zeitraubende Disketten-Zugriffe zum Finden eines Suchwortes benötigt, und der gesamte Parser wird schneller.

Parser 4.0 braucht ziemlich lange, um festzustellen, daß er ein mit den beiden Buchstaben RT beginnendes Wort nicht kennt. Parser 4.1 hingegen benötigt nicht einmal einen einzigen Disketten-Zugriff. Er kann aus seinem Index-Feld auf Anhieb erlesen, daß die Buchstabenkombination RT im Wortschatz nicht vertreten ist. Parser 4.0 läßt sich schnell und relativ einfach zu Parser 4.1 aufrüsten. Zunächst müssen Sie den Index-Maker modifizieren, damit er Index-Felder erstellen kann. Der Rest dürfte für Sie dann ein Kinderspiel sein – wenn Sie Parser 4.0 verstanden haben.

Parser 5.0 – der Superparser

Parser 4.0 ist ja recht nett. Man kann relativ unkompliziert den Wortschatz erweitern und schnell Programmänderungen vornehmen. Trotzdem reicht Parser 4.0 in puncto Geschwindigkeit keineswegs an den INFOCOM-Parser heran. Parser 4.0 braucht im Durchschnitt 15 Sekunden zum Codieren eines Befehls, Parser 4.1 etwa die Hälfte. Mit dem Codieren ist es allerdings noch lange nicht abgetan – bis auf dem Bildschirm eine Antwort erscheint, muß die Codierung erst noch analysiert werden etc. Um eine INFOCOM ähnliche Geschwindigkeit zu erreichen, muß man daher einen möglichst schnellen Parser haben, um Zeit für die Analysenberechnung zu sparen. Wie macht's INFOCOM? Mit Sicherheit verwaltet INFOCOM seine 1000 Wörter NICHT in Form einer relativen Datei. Zwar beginnt nach jeder Befehlseingabe die Floppy zu laufen, jedoch nur, um den Antwortsatz einzulesen. Codiert ist er ruck, zuck. Um solche Geschwindigkeiten zu erreichen, hilf die schnellste Floppy beziehungsweise relative Datei-Programmierung nicht weiter. Der Wortschatz muß irgendwo im RAM untergebracht werden. Nehmen wir einmal an, wir möchten im Speicher 600 Wörter unterbringen. Für jedes Wort haben wir uns 21 Zeichen + 4 Zeichen für Wortart und Wortcode reserviert. Wir bräuchten also 600 mal 25 Byte RAM, um diesen Wortschatz im RAM zu verstauen. Also einen freien RAM-Bereich von zirka 15 KByte. Abgesehen vom Basic-RAM verfügt der C64 über freies

```

10 REM ***** <022>
20 REM * RAM-EDITOR * <053>
30 REM * <C> 1985 BEI * <149>
40 REM * MICHAEL NICKLES * <157>
50 REM ***** <062>
60 DIM M$(255),WZ(26) <152>
70 GOTO 51000 <082>
8000 REM ORDNUNGSFILE INDEX-SEQ -----
----- <087>
8005 PRINT"MOECHTEN SIE EINEN INDEX-FILE E
RSTELLEN?"; <144>
8006 GET I$:IF I$<>"J"AND I$<>"N"THEN 8006 <051>
8008 IF I$="N"THEN RETURN <240>
8019 INPUT"AB SPEICHERSTELLE ";E1 <124>
8020 INPUT"BIS(15SPACE)";E2:IF E1>E2 THEN
PRINT"UNSINN! :GOTO8005 <234>
8030 INPUT"SCHRIITWEITE-STEP ";SW <048>
8100 PRINT"DOWN)BITTE WARTEN!" <255>
8105 FOR I1=E1 TO E2 STEP SW <235>
8110 :FOR I2=1 TO 26 <222>
8112 : IF USR(I1)=64+I2 AND WZ(I2)=0 THEN
WZ(I2)=I1 <001>
8114 :NEXT I2 <198>
8120 NEXT I1 <093>
8400 REM WZ(N) ALS SEQ-FILE SPEICHERN ----
----- <010>
8410 POKE 198,0:INPUT"FILENAME ";I$:I$=I$+
",S,W" <231>
8420 OPEN 2,8,2,I$ <135>
8430 FOR I=1 TO 26 <249>
8440 :PRINT#2,WZ(I) <080>
8450 NEXT I:CLOSE 2 <219>
8500 RETURN <176>
9000 REM UP-POSITIONIEREN -----
----- <215>
9010 HB%=SA/256:LB%=SA-HB%*256 <068>
9020 PRINT#15,"P";CHR$(3);CHR$(LB%);CHR$(H
B%);CHR$(1) <005>
9030 INPUT#15,A:IF A=50 THEN PRINT"SATZNUM
MER ZU HOCH!":END <135>
9040 RETURN <208>
9100 REM UP-SATZ LESEN -----
----- <037>
9110 IF A=50 THEN RETURN <082>

```

Listing 9. Relative Dateien werden mit dem RAM-Editor in RAM-Dateien gewandelt

```

9120 INPUT#1,A$ <116>
9130 RETURN <042>
51000 REM RAM-EDITOR -----
-----
-- <113>
51029 POKE 53280,0:POKE 53281,0:PRINT" (WHI <100>
TE)"
51030 PRINT"(CLR,RVSON)RAM-EDITOR V. 1.5 (4 <049>
SPACE)VON MICHAEL NICKLES (RVDOFF)";
51040 REM MODIFIZIERTE PEEK-FUNKTION -----
-----
-- <234>
51045 DATA 32,247,183,120,169,52,133,1 <112>
51048 DATA 160,0,177,20,168,169,55,133 <243>
51050 DATA 1,88,76,162,179 <078>
51060 FOR I=704 TO 724:READ X:POKE I,X:S=S <178>
+X:NEXT I
51070 POKE 785,192:POKE 786,2:REM USR-ZEIG <022>
ER
51080 IF S<>2325 THEN PRINT"FEHLER IN DATA <036>
S (SHIFT-SPACE)!" :END
51130 REM ANFANG -----
-----
-- <174>
51131 REM GOTO8000 <127>
51132 PRINT"(DOWN)RAM EDITOR (2SPACE)KANN I <128>
N FREIEN RAM-BEREICHEN";
51134 PRINT"RELATIVE DATEIEN VERWALTEN." <066>
51135 PRINT"BITTE NAME DER RELATIVEN DATEI <145>
ANGEBEN,
51136 PRINT"VON DER DATEN INS RAM GESCHRIE <246>
BEN WERDEN";
51137 PRINT"SOLLEN (DISK EINLEGEN)!" <027>
51138 INPUT"(DOWN)FILENAME ";F$ <028>
51139 OPEN 15,8,15 :REM FLOPPYKANAL AN <130>
51140 PRINT#15,"I0":REM INITIALIZIEREN <098>
51141 OPEN 1,8,3,F$ :REM DATEI OEFFNEN <200>
51143 PRINT"BITTE GEBEN SIE ANFANG UND EN <041>
DE DES(5SPACE)BEREICHES AN, DER INS R
AM";
51144 PRINT" GESCHRIEBEN (3SPACE)WERDEN SOL <037>
L!";
51146 POKE 198,0:INPUT"(DOWN)ANFANG";AN <122>
51147 INPUT"ENDE (2SPACE)";EN:IF EN<=AN THE <235>
N PRINT"UNSINN !":GOTO 51146
51148 SA=AN:GOSUB 9000:GOSUB 9100 <143>
51149 PRINT"(DOWN)BITTE GEBEN SIE ANFANG U <088>
ND ENDE DES(5SPACE)RAM-BEREICHES AN,
IN DEM";
51150 PRINT" DIE DATEN(6SPACE)ABGELEGT WER <041>
DEN SOLLEN! ACHTUNG - DIESE ANGABEN
";
51151 PRINT"WERDEN NICHT UEBERPRUEFT ! <048>
51152 POKE 198,0:INPUT"(DOWN)ANFANG";E1 <029>
51153 INPUT"ENDE";E2 <149>
51154 IF E2<E1 THEN PRINT"UNSINN !":GOTO 5 <220>
1152 RUN
51159 GOSUB 60000 <122>
51200 REM INS RAM SCHREIBEN ----- <023>
51205 PRINT"(DOWN)BITTE WARTEN!" <179>
51210 IZ=E1 <177>
51215 FOR SA=AN TO EN:GOSUB 9000:GOSUB 910 <188>
0: REM LESE A$
51220 FOR II=1 TO 3 <176>
51225 :IF MA(II)=0 THEN 51250 <152>
51230 :IF TY%(II)=2 THEN GOSUB 51500:REM $ <097>
51240 :IF TY%(II)=1 THEN GOSUB 51600:REM V <213>
AL
51250 NEXT II <171>
51255 NEXT SA <128>
51270 REM SPEICHERN AUF DISK ----- <206>
51280 PRINT"(DOWN)BITTE DISKETTE EINLEGEN <133>
AUF DER RAM-
51281 PRINT"BEREICH GESPEICHERT WIRD! (RET <088>
URN)":POKE 198,0:WAIT 198,1
51290 POKE 198,0:INPUT"FILENAME";I$:IF I$= <075>
"" THEN 51290
51300 PRINT"(DOWN)BITTE WARTEN!" <018>
51315 FOR I=1 TO 15:CLOSE I:NEXT <206>
51320 OPEN 1,8,1,I$ <202>
51330 HB%=INT(E1/256):LB%=E1-HB%*256 <070>
51340 PRINT#1,CHR$(LB%);CHR$(HB%); <018>
51350 FOR I=E1 TO E2 <145>
51360 :PRINT#1,CHR$(USR(I)); <232>
51370 NEXT I:CLOSE 1 <146>
51375 GOSUB 8000 :REM INDEX-SEQ-FILE <115>
51380 PRINT"*** FERTIG ***":END <196>

51500 REM ALS STRING SPEICHERN ----- <221>
51501 IF LEN(A$)<MB(II)-MA(II)+1 THEN A$=A <221>
$+" ":GOTO 51501
51510 FOR I=MA(II) TO MB(II) <090>
51515 :IF ASC(MID$(A$,I,1))=29 THEN POKE I <169>
Z,32:IZ=IZ+1:GOTO 51520
51516 :POKE IZ,ASC(MID$(A$,I,1)):IZ=IZ+1 <240>
51520 NEXT I <039>
51550 RETURN <043>
51600 REM ALS VAL-WERT SPEICHERN ----- <009>
51610 POKE IZ,VAL(MID$(A$,MA(II),MB(II))) <197>
51620 IZ=IZ+1 <158>
51630 RETURN <123>
60000 REM MARKIEREN WIE DATIERT WIRD <108>
60010 I$="":FOR I=1 TO LEN(A$) <225>
60012 : IF ASC(MID$(A$,I,1))=29 THEN I$=I$ <149>
+" ":GOTO 60015
60014 : I$=I$+MID$(A$,I,1) <012>
60015 NEXT I <152>
60019 FOR I=1 TO LEN(I$):M$(I)="0":NEXT <123>
60020 IM=1:PRINT <131>
60025 PRINT I$:FOR I=1 TO LEN(I$):PRINT M$ <223>
(I);:NEXT:PRINT
60030 PRINT"(DOWN)EDIT (3SPACE)";MID$(I$,I <232>
M,1);"<(4SPACE,RVSON)"IM"(RVDOFF)"
60035 I=PEEK(203):IF I=64 THEN 60035 <110>
60040 IF I=2 AND PEEK(653)=0 AND IM<>LEN(I <174>
$) THEN IM=IM+1:GOTO 60070
60042 IF I=2 AND PEEK(653)=1 AND IM<>1 THE <119>
N IM=IM-1:GOTO 60070
60045 IF I=56 THEN M$(IM)="1":GOTO 60070 <041>
60047 IF I=59 AND IG=0 THEN M$(IM)="2":GOT <012>
O 60070
60050 IF I=8 AND IG=0 THEN M$(IM)="3":GOTO <164>
60070
60055 IF I=35 THEN M$(IM)="0":GOTO 60070 <178>
60056 IF I=1 THEN 60100 <013>
60060 GOTO 60035 <183>
60070 PRINT"(UP,25SPACE,25LEFT,3UP)";:GOTO <139>
60025
60100 REM UMRECHNEN <252>
60102 FOR I=1 TO 3:MA(I)=0:MB(I)=0:NEXT I <148>
60105 FOR II=1 TO 3:REM PRINT"(RVSON)II=( <194>
RVDOFF)";II
60110 :FOR I=1 TO LEN(I$):IF MA(II)=0 AND <059>
M$(I)=RIGHT$(STR$(II),1) THEN MA(II)=
I
60112 :REM PRINT"M$(I)=";M$(I);"(2SPACE)RI <051>
GHT$(RIGHT$(STR$(II),1)
60113 NEXT I <250>
60115 :IF MA(II)=0 THEN 60150 <245>
60120 : MB(II)=MA(II) <124>
60125 : MB(II)=MB(II)+1:REM PRINT,MB(II) <069>
60130 : IF M$(MB(II))<>RIGHT$(STR$(II),1 <217>
) THEN MB(II)=MB(II)-1:GOTO 60150
60135 : IF MB(II)=LEN(I$) THEN 60150 <074>
60140 : GOTO 60125 <248>
60150 NEXT II <179>
60200 PRINT"(CLR,RVSON)";I$:PRINT"(RVDOFF)W <143>
URDE IN FOLGENDE BEREICHE ZERLEGT: (D
OWN)"
60210 FOR I=1 TO 3 <108>
60215 IF MA(I)=0 THEN 60230 <051>
60220 PRINT"BEREICH";I;": (SPACE,RVSON)"MID <126>
$(I$,MA(I),MB(I)-MA(I)+1)"(RVDOFF)"
60230 NEXT I:PRINT <071>
60240 PRINT"BITTE GEBEN SIE NUN AN, OB DIE <037>
BEREICHE
60245 PRINT"ALS (SPACE,RVSON)V (RVDOFF)AL-WER <147>
T ODER ALS (SPACE,RVSON)S (RVDOFF)TRING
60250 PRINT"GESPEICHERT WERDEN SOLLEN! (DOW <028>
N)"
60260 FOR I=1 TO 3 <158>
60262 IF MA(I)=0 THEN 60290 <228>
60265 PRINT"(RVSON)"MID$(I$,MA(I),MB(I)-MA <229>
(I)+1)"(RVDOFF)"
60270 POKE 198,0:INPUT"(RVSON)V (RVDOFF,SPAC <045>
E) ODER (SPACE,RVSON)S (RVDOFF)";II$
60275 IF II$<>"V"AND II$<>"S" THEN 60270 <095>
60280 TY%(I)=1:IF II$="S" THEN TY%(I)=2 <110>
60290 NEXT I <173>
60300 RETURN <157>
61000 CLOSE 1:OPEN 1,8,15 <037>
61010 INPUT#1,A,B$,C,D <014>
61020 PRINT A,B$,C,D <126>
61030 CLOSE 1 <078>

```

Listing 9. RAM-Editor (Schluß)

Der RAM-Editor

40960 ABWAERTS

40883 AUFWAERTS

41006 BEDANKEN

41029 BEDECKEN

41052 BEFESTIGEN

WORT WZ WC

Hier stehen Zeichen die später als Zahl gelten

<pre>ram-editor v. 1. 5.</pre>	von michael nickles
<pre>ram-editor kann in freien ram-bereichen relative dateien verwalten bitte name der relativen datei angeben, von der daten ins ram geschrieben werden sollen (disk einlegen)!</pre>	} Disk mit relativer Datei einlegen und Filenames angeben
<pre>filename ? wortschatz bitte geben sie anfang und ende des bereiches an, der ins ram geschrieben werden soll!</pre>	} Anfang und Ende des Bereiches angeben
<pre>anfang? 1 ende ? 197</pre>	}
<pre>bitte geben sie anfang und ende des ram-bereiches an, in dem die daten abgelegt werden sollen! achtung - diese angaben werden nicht ueberprueft!</pre>	} RAM-Bereich von-bis angeben
<pre>anfang? 40960 ende? 45491</pre>	}
<pre>abwaerts 2010 111111111111111111112333</pre>	} Markieren
<pre>edit >0< 25</pre>	}
<pre>abwaerts 2010 wurde in folgende bereiche zerlegt:</pre>	}
<pre>bereich 1 : abwaerts bereich 2 : 2 bereich 3 : 010</pre>	}
<pre>bitte geben sie nun an, ob die bereiche als val-wert oder als string gespeichert werden sollen!</pre>	} Angeben, ob Bereich als VAL-Wert oder String abgehandelt wird
<pre>abwaerts v oder s? s 2 v oder s? v 010 v oder s? v</pre>	}
<pre>bitte warten!</pre>	}
<pre>bitte diskette einlegen, auf der ram-bereich gespeichert wird! {return filename? wortschatz</pre>	} Wortschatz als PRG-File speichern
<pre>bitte warten! moechten sie einen index-file erstellen? ab speicherstelle ? 40960 bis ? 45491 schrittweite-step ? 23</pre>	} Index erstellen
<pre>bitte warten! filename ? index ... fertig ...</pre>	

64'er

selbst ausrechnen. Wir wollen unseren Wortschatz ab Speicherstelle 40960 unterbringen. Warum der Editor nicht selbständig ermitteln kann, erkläre ich Ihnen gleich. Wir haben 197 Wörter und brauchen pro Wort 23 Byte RAM (wie wir bereits ermittelt haben). Es werden für den gesamten Wortschatz also $197 \text{ mal } 23 = 4531$ Byte benötigt. Das Ende des RAM-Bereiches errechnet sich also aus $40960 + 4531 = 45491$. Jetzt wird der erste Satz der relativen Datei eingeblendet. Anhand der Cursortasten left-right und der Zifferntasten 1 bis 0 können Sie jetzt einzelne Satzteile markieren. In Bild 11 können Sie sehen, wie im Falle unserer Wortschatz-Datei markiert werden muß. Der Markierungsvorgang wird mit der RETURN-Taste abgeschlossen. Jetzt erscheint auf dem Bildschirm, wie der eben markierte Satz in einzelne Bereiche zerlegt wurde. Es können maximal 3 Bereiche markiert werden – in beliebiger Reihenfolge! Sie müssen nun zu jedem einzelnen Bereich angeben, ob er als VAL-Wert oder als String gespeichert werden soll. Bei String wird der Bereich einfach in ASC-Zahlen umgewandelt, die ins RAM gePOKEt werden. Bei VAL wird aus dem Bereich ein VAL-Wert ermittelt (der nicht größer als 255 sein darf – gegebenenfalls müssen Sie in High- und Low-Byte aufteilen), der dann einfach an die entsprechende Stelle im RAM gePOKEt wird. Sie müssen nun eine Zeitlang warten. Dann werden Sie aufgefordert, einen Filenamen anzugeben, unter dem die Wortschatz-RAM-Datei als PRG-File auf Diskette gespeichert wird. Bitte legen Sie hier eine leere Diskette ein und geben Sie als File-

namen »Wortschatz« an. Jetzt dauert's wieder eine Weile. Nun kann ein Index-File erstellt werden, was bei Wortschatz-Dateien in der Regel notwendig ist. Es müssen wieder Anfang und Ende des RAM-Bereiches angegeben werden und außerdem die Schrittweite – dies ist die Länge jedes Satzes im RAM – in unserem Fall 23 (Byte). Jetzt speichern Sie den erstellten Index unter dem Filenamen »Index« ab, und fertig ist die RAM-Wortschatzdatei. Dies alles hört sich viel komplizierter an, als es eigentlich ist. Bitte vergleichen Sie stets mit Bild 11, damit Sie beim Erstellen der RAM-Datei nichts falsch machen!

Die Wortschatz-Datei kann jetzt mittels LOAD»WORTSCHATZ«,8,1 geladen werden.

Der Index ist als SEQ-File gespeichert und kann folgendermaßen eingelesen werden:

```
5 OPEN3,4
10 OPEN1,8,2,"INDEX"
20 FORI=1TO26
30 :INPUT #1,X
40 :PRINT #3,CHR$(64+I) " AB ADRESSE"X
50 NEXTI
60 CLOSE1
70 CLOSE3
```

Wenn Sie alles richtig gemacht haben, dann müssen Sie

```
10 IF I=0 THEN I=1:LOAD"WORTSCHATZ",8,1 <067>
20 DIM BE$(25) <194>
30 DIM IN(26):FOR I=1 TO 26:READ IN(I):NEXT I <215>
40 DATA 40960,41006,0,41581,41650 <044>
50 DATA 42018,42110,42317,42501,0 <219>
60 DATA 42547,42685,0,42823,43007 <140>
70 DATA 43076,43237,43260,43605,44364 <159>
80 DATA 44502,44571,45031,0,0,45284 <175>
100 REM MODIFIZIERTE PEEK-FUNKTION ----- <034>
----- <077>
101 : <077>
110 DATA 32,247,183,120,169,52,133,1 <234>
120 DATA 160,0,177,20,168,169,55,133 <116>
130 DATA 1,88,76,162,179 <215>
140 FOR I=704 TO 724:READ X:POKE I,X:S=S+X <059>
: NEXT I <159>
150 POKE 785,192:POKE 786,2:REM USR-ZEIGER <159>
160 IF S<>2325 THEN PRINT"FEHLER IN DATAS( <173>
SHIFT-SPACE)!" :END <212>
200 GOSUB 50000 <051>
210 PRINT"VE="VE:PRINT"O1="O1:PRINT"O2="O2 <166>
:PRINT"AD="AD:PRINT"RI="RI <093>
230 GOTO 200 <009>
50000 REM ***** <041>
50001 REM * WORT-PARSER 5.0 * <011>
50002 REM * <086>
50003 REM * <013>
50004 REM * <082>
50005 REM * <015>
50006 REM * MICHAEL NICKLES <101>
50007 REM *
50008 REM *****
50010 REM BEFEHLSSEINGABE ----- <076>
----- <202>
50011 : <111>
50015 J1=1:J2=10:J3=198:J4=54276:J5=54277: <068>
J6=203:J7=3000:J8=2:J9=20:J0=54273 <078>
50016 K1=54272:K2=89:K3=129:K4=47:K5=57:K6 <081>
=65:K7=90:K8=32:K9=34:K0=44 <076>
50017 PRINT <058>
50019 POKE 54296,15:REM UND ABFRAGEN <238>
50020 TT=: REM INPUT-INTERRUPT-ZAEHLER
50030 FOR I=J1 TO J2:BE$(I)="" :NEXT:POKE J <058>
3,,:BE$="" : REM SATZ LOESCHEN
50040 PRINT"(YELLOW)";:PRINT"e";
50050 POKE J4,,:POKE J5,,:GET X$:II=PEEK(J <238>
6):IF TT<J7 THEN 50060:REM TT=TT+1 !
```

```
!!! <010>
50055 FOR I=1 TO LEN(BE$)+J8:PRINT CHR$(J9 <233>
):NEXT:VE=0:RETURN <223>
50060 IF X$="" THEN 50050 <033>
50061 POKE J5,J1:POKE J0,J2:POKE K1,K2:POK <065>
E J4,K3
50070 TT=0 <158>
50080 IF II=1 THEN IF BE$<>"" THEN PRINT"(L <214>
EFT).":GOTO 50200
50085 I=ASC(X$):IF I>K4 AND I<K5 THEN 5010 <146>
0 <103>
50090 I=ASC(X$):IF I<K6 OR I>K7 THEN IF I< <072>
>K8 AND I<>J9 AND I<>K9 AND I<>K0 TH
EN 50050
50100 IF I=J9 AND LEN(BE$)=. THEN 50050
50110 IF I=J9 THEN PRINT"(2LEFT,2SPACE,2LE <168>
FT)";:BE$=LEFT$(BE$,LEN(BE$)-J1):GO <168>
TO 50050
50120 BE$=BE$+CHR$(I):IF LEN(BE$)>78 THEN <007>
BE$=LEFT$(BE$,78):GOTO 50050
50125 IF I=K9 THEN I=64
50130 PRINT"(LEFT)";:CHR$(I);"e";:GOTO 5005 <020>
0
50200 REM CODIERUNG ----- <138>
----- <127>
-- <243>
50201 : <124>
50250 REM WORT IM RAM SUCHEN ***** <230>
50254 I$=BE$ <120>
50255 IM=ASC(LEFT$(I$,1)):I=IN(IM-64) <205>
50262 REM BEGIN LOOP <157>
50264 :IF IM<>USR(I) THEN 50280
50265 :FOR II=1 TO LEN(I$):PRINT MID$(I$,I <241>
1,1),CHR$(USR(I+II-1))
50270 :IF ASC(MID$(I$,II,1))<>USR(I+II-1) <012>
THEN II=LEN(I$):GOTO 50276
50273 :IF II=LEN(I$) AND USR(I+21)=3 THEN <163>
OB=USR(I+22):OM=OB:PRINT"OBJEKT="OB:
END <079>
50274 :IF II=LEN(I$) AND USR(I+21)=2 THEN <002>
RI=USR(I+22):PRINT"RICHTUNG="RI:END
50275 :IF II=LEN(I$) AND USR(I+21)=1 THEN <076>
VE=USR(I+22):PRINT"VERB="VE:END
50276 :NEXT II
50278 I=I+23:GOTO 50264
50280 REM END LOOP <076>
50290 PRINT"ICH KENNE DAS WORT "BE$" NICHT <076>
!":END
```

Listing 10. Grundgerüst vom Parser 5.0

jetzt folgende Wertetabelle (bei Ausgabe auf Drucker) erhalten:

```
A AB ADRESSE 40960
B AB ADRESSE 41006
C AB ADRESSE 0
D AB ADRESSE 41581
E AB ADRESSE 41650
F AB ADRESSE 42018
G AB ADRESSE 42110
H AB ADRESSE 42317
I AB ADRESSE 42501
J AB ADRESSE 0
K AB ADRESSE 42547
L AB ADRESSE 42685
M AB ADRESSE 0
N AB ADRESSE 42823
O AB ADRESSE 43007
P AB ADRESSE 43076
Q AB ADRESSE 43237
R AB ADRESSE 43260
S AB ADRESSE 43605
T AB ADRESSE 44364
U AB ADRESSE 44502
V AB ADRESSE 44571
W AB ADRESSE 45031
X AB ADRESSE 0
Y AB ADRESSE 0
Z AB ADRESSE 45284
```

Jetzt steht der Programmierung von Parser 5.0, dem Superparser, nichts mehr im Wege. Im Prinzip ist es sogar ein leichtes, unseren Parser 4.0 so zu modifizieren, daß er statt mit einer relativen Datei mit einer RAM-Datei arbeitet. Dies ist jedoch nicht empfehlenswert, da es bei RAM-Dateien wesentlich effektivere Verwaltungsmöglichkeiten gibt, die allerdings etwas komplizierter als bei der relativen Disk-Verwaltung verlaufen.

Leider nicht komplett

Um Ihnen eine Enttäuschung zu bewahren, muß ich Sie bereits an dieser Stelle darauf aufmerksam machen, daß Sie auf diesen Seiten keineswegs einen kompletten Parser 5.0 finden. Es gibt einfach zu viele Möglichkeiten bei der Verwaltung von RAM-Wortschätzen, von denen Sie am besten selbst entscheiden, welche Ihnen am meisten behagt, nachdem ich Ihnen verschiedene vorgestellt habe.

Bitte geben Sie jedoch zunächst einmal Listing 10 ein.

Dieses Programm ist, wie bereits gesagt, noch kein kompletter Parser. Es soll Ihnen vielmehr als Grundgerüst eines Parsers 5.0 nach eigenen Vorstellungen dienen.

Dokumentation zum Listing:

- 10 Laden des Wortschatzes von Diskette ins RAM.
- 30 Indextabelle IN (1)–IN(26) initialisieren
- 40–80 Index-DATAs. Hier wurde nichts weiter gemacht, als der Index-File ausgelesen und in DATAs untergebracht.
- 100–160 Hier steckt eine für Parser 5.0 grundlegende Erweiterung: Eine modifizierte PEEK-Funktion, die es erlaubt, die Speicherzellen unter dem Basic-ROM auszulesen. Sie wissen mit Sicherheit bereits, daß beim POKEn in diesen Bereich immer ins RAM, aber beim normalen PEEKen immer der Speicherinhalt vom ROM ausgegeben wird.

Die erweiterte PEEK-Funktion wird mittels USR(X) aufgerufen. Wenn Sie zum Beispiel in Speicherzelle 60000 den Wert 1 schreiben mit POKE 60000,1, so erhalten Sie bei PRINT PEEK (60000) einen Wert, der nicht unbedingt der 1 entspricht. PRINT USR(60000) zeigt jedoch den Wert vom RAM: 1.

Diese Erweiterung ist für alle diejenigen unentbehrlich, die Parser 5.0 ausschließlich in Basic programmieren wollen.

- 200 Aufruf von Parser 5.0
- 50010–50130 Erweiterte INPUT-Routine. Diesmal mit schönem Eingabe-Ton.
- 50200 Das Herzstück von Parser 5.0 beginnt hier.
- 50290 Er erfolgt hier die Suche nach dem Wort. Das Programm ist sehr einfach gehalten. Es können nur Befehlssätze eingegeben beziehungsweise codiert werden, die nicht mehr als ein Wort enthalten. Sie können dies anhand des bisher erworbenen Wissens jedoch leicht ändern.
- 50254 Das Suchwort I\$ soll im RAM gesucht werden.
- 50255 In der Variablen IM wird der ASC-Wert des ersten Buchstabens vom Suchwort gespeichert. Anhand der Index-Tabelle wird jetzt ermittelt, ab welcher Speicherzelle (=I) die Wörter stehen, deren erster Buchstabe den ASC-Wert IM hat.
- 50262–50280 Hier steht die eigentliche Suchroutine.
- 50264 Wenn IM nicht gleich USR(I) ist, so heißt dies, daß an der RAM-Stelle I ein Wort steht, dessen erster Buchstabe nicht mit dem Suchwort I\$ übereinstimmt. Der Suchvorgang wird dann mittels GOTO 50290 abgebrochen, und es erfolgt die altbekannte Fehlermeldung »ICH KENNE DIESES WORT NICHT!«.
- 50265 Der gesamte Suchbegriff I\$ wird vom ersten bis zum letzten Zeichen durchlaufen. Es wird dabei verglichen, ob $ASC(MID$(I$,1,1)) - ASC(MID$(I$,LEN(I$),1))$ gleich $USR(I-1) - USR(I-1+LEN(I$))$ ist. Es werden also der Reihe nach die einzelnen Buchstaben des Suchwortes mit den entsprechenden Speicherzellen im RAM verglichen, von denen vermutet wird, daß sie dem Buchstaben entsprechen.
- 50270 Wird bei irgendeinem Buchstaben des Wortes eine Nichtübereinstimmung mit dem Wert im RAM festgestellt, so wird der Suchvorgang abgebrochen mit GOTO 50276, und der Zeiger I wird auf das nächste Wort im RAM gesetzt, dessen erster Buchstabe IM ist: $I = I + 23$ (23, da jedes Wort beziehungsweise jeder Satz 23 Byte lang ist).
- 50273–50275 Wurde bis zum letzten Buchstabenvergleich keine Nichtübereinstimmung festgestellt, so wurde das Suchwort im RAM gefunden. Anhand der Wortart $WA = USR(I + 21)$ wird dann die entsprechende Wortzahl $WA = USR(I + 22)$ ermittelt. Das Wort ist somit codiert.

Wenn Sie das Programm einmal laufen lassen, werden Sie überrascht sein, wie schnell die Wörter im RAM gefunden werden. Schon das Grundmodell von Parser 5.0 ist weitaus schneller als Parser 4.0 und Parser 4.1. Allerdings läßt sich die Verarbeitungsgeschwindigkeit noch erhöhen:

Man kann zum Beispiel eine index-binäre Suchroutine wie bei Parser 4.0 einbauen. Ergänzt man diese index-binäre Methode zu einer index-feld-binären Suchroutine, so erhält man einen Basic-Parser, wie man ihn sich schneller kaum noch vorstellen kann.

Ein index-feld-binärer Parser 5.0 kann an Geschwindigkeit nur noch mit Maschinensprache-Programmierung übertroffen

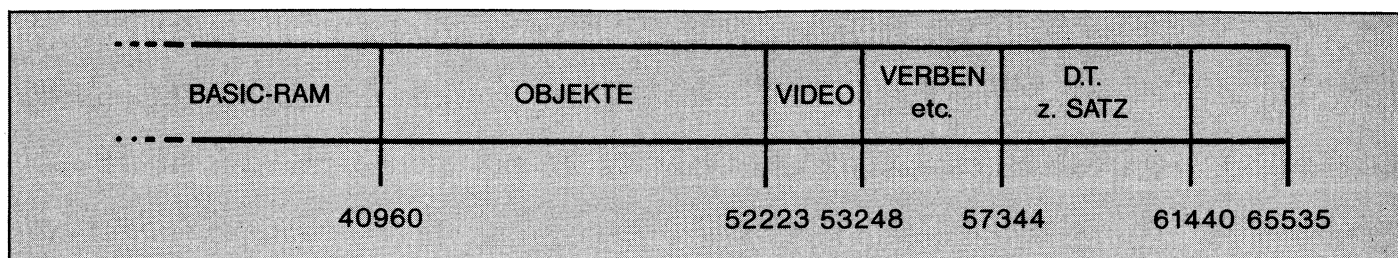


Bild 12. Speicherbelegung des C64, wenn die wichtigsten Routinen eingebaut sind.

fen werden. Allerdings kann man in puncto Wortschatz-Organisation noch einiges verbessern, wie die folgenden Beispiele zeigen:

Verbesserte Wortschatz-Organisation

Wie wir bereits festgestellt haben, besteht ein Wortschatz aus Verben, Objekten, Adjektiven und Richtungsangaben und AUS-Wörtern. Als Wortlänge haben wir uns bisher maximal 21 Zeichen reserviert. Sind aber wirklich 21 Zeichen nötig? – Mit Sicherheit nicht. Ein Verb zum Beispiel ist in 99% aller Fälle an 6 Buchstaben zu erkennen. Das heißt, kennt man die ersten 6 Buchstaben eines Verbs, so weiß man mit großer Wahrscheinlichkeit, welches Verb gemeint ist.

Betrachten wir doch einmal die ersten 6 Sätze unserer Wortschatz-Datei im Falle einer Verbkürzung auf 6 Zeichen:

```
abwaer 2010
aufwae 2009
bedank 1074
bedeck 1073
befest 1042
beklei 1065
belohn 1046
benutz 1063
beraub 1058
berueh 1017
```

Bei einer Umwandlung zur RAM-Datei benötigt man in diesem Fall nur noch 8 Byte – das heißt, man bringt fast dreimal soviele Verben unter wie bisher!

Es ist auch ein leichtes, eine neue Wortschatzdatei zu erstellen, bei der die Verben auf 6 Zeichen Länge begrenzt werden. Sie müssen beim Markieren mit dem RAM-Editor lediglich nur die ersten 6 Zeichen markieren. Bei der Suchroutine ist jetzt allerdings zu beachten, daß im Falle einer Verb-Suche nur noch die ersten 6 Buchstaben des Suchwortes im RAM gesucht werden. Und zwar linksbündig, wie es sich für die Verbsuche gehört. Analog für die Verben kann man auch die Adjektive und Richtungsangaben abhandeln. Bei den AUS-Wörtern, also den Wörtern, die eigentlich zur Codierung gar nicht benötigt werden (zum Beispiel der, die, das, den etc.), gilt dies im großen und ganzen auch.

Etwas schwieriger ist es mit den Objekten. Objekte können sehr lange Namen haben, wie zum Beispiel »Fernsehapparat« oder »Zeitmaschine«.

Behalten wir als Musterbeispiel einmal den Objektnamen »Zeitmaschine« bei. Objekte haben wir bei Parser 4.0 rechtsbündig gesucht (damit sowohl TÜR als auch HAUSTÜR verstanden wird etc.). Die letzten 6 Buchstaben von »Zeitmaschine« lauten »schine«, und daraus läßt sich wahrlich nicht erkennen, daß es sich um eine »Zeitmaschine« handelt. Würde man für die Objekte 10 Buchstaben reservieren, so käme man schon besser zurecht. Bei relativen Dateien muß jeder Satz die gleiche Länge haben, und es kann immer nur eine relative Datei gleichzeitig geöffnet sein. Deshalb waren wir bei Parser 4.0 und 4.1 auch gezwungen, eine große Wortlänge für die Verben zu reservieren, obwohl diese eigentlich nur für Objekte nötig ist. Eine Alternative wäre es gewesen,

Objekte und die anderen Wortarten zusammen, in zwei getrennten relativen Dateien unterzubringen. Das Öffnen und Schließen von relativen Dateien wäre dann jedoch unvermeidbar gewesen. Das hätte Parser 4.0 stark verlangsamt, weil das Öffnen von relativen Dateien viel Zeit in Anspruch nimmt. Also eine miserable oder keine Alternative. Anders bei RAM-Dateien. Von ihnen können beliebig viele gleichzeitig geöffnet werden beziehungsweise sind immer offen. Es bietet sich also an, für die Objekte einen anders organisierten RAM-Bereich herzunehmen. Dies ist auch der Grund, warum ich unseren bisherigen Wortschatz frei von Objekten, nur aus Verben bestehend, gelassen habe. Ich bin mir ziemlich sicher, daß Sie bei Ihren eigenen Adventures zu den vorhandenen 197 Verben und Richtungsangaben kaum noch mehr als 50 andere finden. Wenn Sie sich also Platz für 400 Wörter mit 6 Zeichen Länge (250 Verben + 150 Adjektive) RAM reservieren wollen, so brauchen Sie 400 mal 8 = 3200 Byte Speicherplatz. Für Verben, Richtungsangaben und Adjektive und eventuell auch noch AUS-Wörter können wir das freie RAM von 53248 bis 57343 (= Platz für 511 Wörter mit 8 Zeichen Länge – mit 2 Zeichen gerechnet für Wortart und Wortcode). Jetzt haben wir noch freies RAM von 40960 bis 52222 – also genug Platz für 938 Objekte bei Objektlänge = 11 Zeichen + Wortcode (auf die Wortart können wir verzichten, da wir wissen, daß sich in diesem Bereich nur Objekte befinden).

938 + 511, das bringt einen Wortschatz von 1449 Wörtern. Was soll also das unsinnige Infocom-Getue um den fantastischen 1000er-Wortschatz. 1449 Wörter, das sind mehr Wörter, als der englische Grundwortschatz hat, mit dem sich immerhin schon 70 % aller Dinge in Englisch formulieren lassen, sind garantiert genug für jedes nur erdenkliche Adventure, das auf einem C64 geschrieben werden soll. Jetzt liegt es nur noch an Ihnen, Parser 5.0 zu programmieren. Wenn Sie das gesamte Parser-Technik-Kapitel aufmerksam durchgearbeitet haben, dürfte Ihnen diese Programmierung keine Schwierigkeiten mehr bereiten (wobei Sie noch nicht einmal in Maschinensprache programmieren müssen)!

Bild 12 zeigt, wie es jetzt im RAM des C64 aussieht:

Der freie RAM-Bereich ist nahezu voll ausgenutzt mit dem Wortschatz und dem deutschen Zeichensatz. Besser kann man das RAM im Bereich der Adventureprogrammierung garantiert nicht mehr ausnutzen. Sogar die Grafikroutine (vergleiche Kapitel Grafikprogrammierung) läßt sich in dieses RAM-Schema noch integrieren – es geht dann nur Basic-RAM verloren sowie 1 KByte des Objekt-RAMs, wobei aber immer noch Platz für genügend Objekte vorhanden ist. Den RAM-Bereich von 61440 bis 65535 lasse ich Ihnen für eigene Ideen offen. Den Bereich von 1024 bis 2048, der durch das Verschieben des Bildschirms freigeworden ist, werde ich allerdings noch beanspruchen.

Wir haben es also geschafft – wir haben viele Basic-Routinen und Parser kennengelernt und das RAM des C64 optimal mit Wortschatz, Grafikfenster und deutschem Zeichensatz belegt. Es steht also nur noch aus, das ganze Wissen zusammenzupacken und daraus Adventures zu programmieren. Wie das geht, erfahren Sie im anschließenden Kapitel.

(M. Nickles/bs)