Abenteuer selbst programmiert

Abenteuerspiele auf dem Commodore 64 gehören mittlerweile zu den beliebtesten Freizeitbeschäftigungen mit einem Computer. Dieser Kurs wird Sie in die Lage versetzen, nahezu professionelle Abenteuerspiele selbst zu programmieren. Alles was Sie brauchen ist ein bißchen Basic-Erfahrung, gute Ideen, viel Fantasie und Spaß am Programmieren.

Eine unheimliche Stille liegt in der Luft...

Sie befinden sich in einem großen Raum, in dessen Mitte auf dem kalten Steinboden ein kleiner roter Teppich liegt. Eine kleine Fackel, die an der Wand in einer Halterung steckt, beleuchtet den Raum nur spärlich. Am Boden liegt ein zwei Meter langes stabiles Holzbrett.

Während Sie sich umschauen, fällt plötzlich ein schweres Eisenfallgitter hinter Ihnen herunter und versperrt den einzigen Ausgang.

Alle Ihre Versuche, das Fallgitter zu heben oder einen geheimen Ausgang zu finden, scheitern. Mit großem Schrecken vernehmen Sie plötzlich ein lautes Rattern — die Zimmerdecke bewegt sich langsam, aber sicher nach unten. Erst jetzt bemerken Sie die spitzen Eisenstangen, die von der Decke herunterragen! Sie erinnern sich sofort an den Zauberring, den ein Monster verloren hatte, nachdem Sie es getötet hatten. Sie greifen in Ihre Tasche, holen den Ring heraus und stecken ihn sich an den Finger. Sie drehen den Ring mit der Hoffnung, daß er Sie durch seine Zauberkraft befreien kann, aber es passiert nichts. Inzwischen hat sich die Decke bis auf etwa drei Meter Höhe gesenkt! Ihr nächster Gedanke ist das stabile Brett, Sie klemmen es zwischen Boden und Decke. Sie atmen auf, als Sie feststellen, daß das Brett die Decke, jedoch nicht das Rattern zum Stillstand bringt. Aber ihre Freude hat ein schnelles Ende, als das Brett dem ungeheueren großen Druck der Decke nachgibt und zerbricht. Die Decke ist nun so weit unten, daß Sie nicht mehr aufrecht stehen können. In letzter Sekunde kommt Ihnen schließlich der rettende Gedanke: Sie stürzen auf den kleinen roten Teppich zu...

Dies könnte eine von vielen Action-Szenen eines Abenteuerspiels sein, wie Sie es vielleicht schon bald selbst schreiben werden.

Was ist ein Abenteuerspiel oder englisch »Adventure« eigentlich? Eine einfache Antwort hierfür wäre zum Beispiel: »Das Gegenteil von einem Ballerspiel.« Während bei dem »Baller-«, »Weltraum-« oder »Grafik-« Action-Spiel flinke Finger und ein starrer Blick zum Bildschirm nötig sind, bezieht sich ein Adventure mehr auf scharfen Verstand.

Wer ein Adventure spielen will, muß viel Fantasie und Einfühlungsvermögen mitbringen. Bei einem Adventure stellt der Computer ein Fenster in eine andere Welt dar, deren Grenzen und Gesetze vom Programmierer gesetzt werden.

Am besten läßt sich das Prinzip eines Abenteuerspiels anhand eines Rollenspiels erklären.

Diese Rollen- beziehungsweise Fantasierollenspiele sind besonders in England unter dem Titel Dungeons & Dragons (Höhlen und Drachen) verbreitet. Diese Rollenspiele können sich manchmal über ein ganzes Wochenende hinziehen.
Wie ist solch ein Rollenspiel aufgebaut?

Da gibt es zum einen den Dungeon-Master (Höhlenmeister). Dieser ist der Ersteller des Spiels und somit auch der Spielleiter. Den einzelnen Spielern werden Charaktere wie Zauberer, Fee oder Zwerg etc. zugeteilt.

Außerdem gibt es Nichtspielercharaktere. Dies sind zum Beispiel Monster und Kreaturen, die in dem Spiel auftreten; auch sie werden vom Höhlenmeister gelenkt. Der Höhlenmeister ist der absolute Schiedsrichter. Er verteilt während des Spiels die Punkte. Falls sich im Spiel eine besondere Situation ergibt, so muß er improvisieren, um den Spielern gerecht zu werden.

Dies fordert besonders große Fantasie und Ideenreichtum von ihm. Ein Rollenspiel könnte folgendermaßen ablaufen:

Höhlenmeister: Ihr befindet euch in einer großen, trockenen Höhle mit Fackeln an den Wänden. Was macht ihr jetzt?

Die Spieler beraten sich nun.

Sprecher der Spieler: Wir untersuchen die Höhle genauer.
Der Höhlenmeister schaut nun in seinen Plänen und Notizen nach, ob in der Höhle sonst noch etwas zu finden ist.

Höhlenmeister: Ihr entdeckt eine Geheimtür. Was nun?
Die Spieler beraten sich.

Sprecher der Spieler: Wir öffnen die Tür und lassen unseren stärksten Mann vorausgehen.

An diesem Beispiel können Sie die Aufgaben des Höhlenmeisters erkennen. Unser Ziel soll nun sein, den Computer als Höhlenmeister arbeiten zu lassen. Hierzu geben wir ihm Pläne, Tabellen etc., aus denen er ersehen kann, welche Antworten er uns geben soll.

Das Problem hierbei ist, daß der Computer auch ein wenig an den Plänen, die wir ihm geben, manipulieren und während des Spiels improvisieren soll, damit keine Langeweile beim Spieler aufkommt.

Bevor wir uns an dieses Problem wagen, müssen wir lernen, wie man Pläne erstellt, also wie man sich ein Spielkonzept ausdenkt.

Sie können Ihren Computer also ruhig ausschalten, sich in einen bequemen Sessel zurücklehnen und aufmerksam die folgenden Kapitel lesen...

1. Kapitel: Die Speicher-Grenzen — oder wieviel Abenteuer paßt in den C 64 ?

Im Prinzip gibt es nur zwei Faktoren, die uns beim Erstellen von Abenteuerspielen einschränken: Zum einen die Grenzen unserer Fantasie und zum anderen die Speicherkapazität unseres C64.

Wieviel Abenteuer paßt in 38 KByte? Mit dieser Frage wollen wir uns nun einmal beschäftigen.

Vorweg möchte ich jedoch sagen, daß Sie Ihre Fantasie niemals aus Angst vor Speicherplatzmangel einschränken sollten.

Sollten Sie die Idee zu einem gigantischen Spiel haben, dann kürzen Sie es keinesfalls so lange, bis es in den C64 paßt. Teilen Sie es lieber in mehrere Programme auf, die nacheinander geladen und gespielt werden. Die Antwort auf die Frage hängt hauptsächlich von der Art des Spiels ab. Wir unterscheiden folgendermaßen:

a) Spiele mit überwiegend Grafik (Grafik-Adventure)
b) Spiele mit überwiegend Text (Text-Adventure)

Bei Grafikabenteuerspielen findet man für jeden Raum ein Bild. Unter diesem Bild ist ein kleines Textfenster zur Befehlseingabe, ein Beispiel hierfür wäre das Adventure »Zauberschloß« von Happy Software. Bei Textabenteuerspielen findet man anstelle von Bildern lange Texte, die jeden Raum ausführlich beschreiben — das Bild entsteht also in der Fantasie des Spielers, der den Text liest.

Als ich mein erstes Adventure schrieb, war ich besonders darauf aus, viele Räume zu haben, weil ich dachte, daß dies für ein sehr gutes Adventure unbedingt notwendig sei. Ich bemerkte jedoch bald, daß ich hier völlig falsch lag — das Resultat war ein Spiel mit 200 (!) Räumen, aber kaum Action. Alles was man konnte, war herumlaufen und laufen und laufen. Auch die Befehlsanalyse, auf die ich in einem später folgenden Kapitel ausführlich zu sprechen kommen werde, war viel zu knapp geraten.

Wir wollen unsere anfänglich gestellte Frage deshalb nun erweitern: Wie sieht ein gutes Adventure aus, und wie bringt man es auf 38 KByte? Das Wesentliche an jedem Adventure, das letztendlich auch über dessen Qualität entscheidet, ist der Komfort der Befehlseingabe und deren Analyse. Damit hängt natürlich auch der Wortschatz des Spiels zusammen. Damit es nun nicht zu kompliziert wird, ein Beispiel:

Wir stellen uns einfach ein Abenteuerspiel des Titels X vor.
Wenn X einen Befehl erwartet, fragt es »WAS NUN?«.
Wir geben nun folgendes ein: »Nimm Schwert«.
X antwortet uns: »Sie nehmen das Schwert«.

Geben wir jedoch »Nimm das Schwert« ein, so antwortet X: »Ich kenne (das) nicht.« oder »Sie können (das) nicht nehmen« oder einfach »Das geht nicht«. etc.

Was haben wir nun falsch gemacht?
Für uns ist »Nimm Schwert« und »Nimm das Schwert« gleichermaßen verständlich, obwohl »Nimm das Schwert« besseres Deutsch ist.

Warum versteht uns der Computer jedoch nicht?
Ganz einfach — Die Befehlsanalyse von X sieht so aus:

X versteht nur Befehle, die aus zwei Worten — VERB + OBJEKT bestehen.

Wenn wir also »Nimm das Schwert« eingeben, so nimmt X an, daß das Wort »das« das Objekt des Befehls ist.

X sieht in seiner Objekttabelle nach und kann »das« nicht finden.
X versteht den Befehl also nicht!

Spiel ohne Grenzen

Sie sehen also, was mit guter Befehlsanalyse gemeint ist: Das Spiel soll Sätze wie »Nimm das Schwert, den Ring und das Brett und gehe nach Norden« verstehen. Hierauf werden wir später zurückkommen.

Ein weiteres Kriterium für ein gutes Spiel ist das folgende: Der Spieler sollte viel mehr Möglichkeiten haben, als nur Gegenstände zu nehmen, zu verlieren und herumzulaufen. Gute Spiele müssen so viele Möglichkeiten bieten, daß der Spieler kaum an die Grenzen des Spiels stößt. Es ist nämlich äußerst frustrierend, wenn man ständig eine Antwort wie »Ich verstehe das nicht« nach einer Befehlseingabe erhält.

Wenn wir ein gutes Adventure-Programm schreiben wollen, so müssen wir dem Spieler die Möglichkeit bieten, sich nach Lust und Laune in unserer Abenteuerwelt auszutoben, auch dann, wenn der Spieler etwas macht, das mit der Lösung des Spiels kaum etwas zu tun hat.

Stellen wir uns einmal einen Raum vor, in dem eine große Kiste und ein Fenster ist.

Wir sollten dem Spieler dann folgende Dinge ermöglichen:

— Er kann die Kiste öffnen und schließen (wenn wir wollen, muß er dafür einen Schlüssel haben).
— Er kann Gegenstände in die Kiste legen.
— Er kann die Kiste mitnehmen, falls sie nicht zu schwer ist.
— Er kann sich in der Kiste verstecken (falls sie groß genug ist).
— Er kann die Kiste verschieben, wenn sie zu schwer zum Tragen ist.

Aber Achtung: Der Spieler kann nur in die Kiste gehen, wenn alle Gegenstände, die er hat, mit in die Kiste passen. Ansonsten muß er sie vorher ablegen.

— Er kann das Fenster öffnen und schließen sowie hinaussehen.
— Er kann aus dem Fenster springen, wenn ihm das Spiel zu langweilig wird, oder aus Angst vor einem Monster, das den Raum betritt.

Sie sehen also, wie viele Möglichkeiten in der Ausgangssituation (Kiste und Fenster) stecken. Falls die Kiste enorm groß ist, dann kann der Spieler sich vielleicht sogar mit einer Prinzessin, die er kurz zuvor gerettet hat, in ihr verstecken...

Stichwort »Prinzessin« — Wie Sie sehen, sind lebendige Personen in einem guten Adventure nicht fehl am Platz.

Allerdings ist es langweilig, wenn man nur auf Taubstumme trifft. Wenn Personen im Spiel vorkommen, so sollten sie mehr können, als nur herumstehen oder wortlos hinter dem Spieler herrennen. Ein Spiel, welches ein gutes Beispiel im Bezug auf »selbstdenkende« Spielcharaktere abgibt, ist das englische Adventure »The Hobbit« von Melbourne House. Wer dieses Spiel kennt (wahrscheinlich jeder Adventure-Freak), weiß, was Thorin, Gandalf, Elrond, Bard, der Butler oder Smaug, der Drache, im Spiel alles selbständig treiben. Leider ist beim Hobbit auch einiges zu kritisieren:

Sagt man seinem Freund Elrond »Lies die Karte«, so kann es manchmal vorkommen, daß dieser einfach nein sagt. Auch kann es passieren, daß man von Monstern gefangen wird und dann auf die Rettung durch einen Freund wartet — manchmal kommt dieser jedoch nicht.

Wir können also nun zusammenfassen, was ein gutes Adventure auszeichnet.

1. Der Spieler sollte so viele (auch unsinnige) Möglichkeiten haben, daß ihm so wenig wie nur irgend möglich auffällt, daß das Spiel Grenzen hat.

2. Das Spiel muß einen großen Wortschatz und eine großzügige Befehlsanalyse haben — das heißt, das Spiel soll nicht Worte allein erkennen, sondern den Sinn eines Satzes verstehen.

3. Das Spiel sollte zu jedem Raum einen ausführlich beschreibenden Text haben. Grafiken sollten diesen Text ab und zu unterstützen — es sollte jedoch nicht so sein, daß ein Spieler erst rätseln muß, was die Grafik darstellt, bevor er den Sinn erkennt. Grafiken sind nur Ergänzungen zu einem guten Text — sie können ihn nicht völlig ersetzen.

4. Wenn im Spiel Personen (Charaktere) auftreten, so sollten sie auch einen eigenen Willen haben, sowie sprechen und verstehen können. Dadurch besteht die Möglichkeit, eine lebendige Adventure-Welt zu erbauen und nicht eine tote, in der nur der Spieler umherläuft.

Nehmen wir als Beispiel noch einmal den Raum mit der Kiste und dem Fenster:

Angenommen, ein selbstdenkender Spielcharakter, wie ein Geist, kommt in diesen Raum, so hat er genau dieselben oder ähnliche Möglichkeiten wie der Spieler, wenn er sich in diesem Raum befindet.

Wir spinnen einmal weiter...

Angenommen, ein Geist betritt den Raum mit der Kiste (der Spieler ist gerade woanders).

Der Geist sieht also die Kiste und beschließt, sich in dieser zu verstecken. Er öffnet die Kiste also, geht hinein und schließt sie von innen. Wenn der Spieler nun in den Raum kommt, so bieten sich jetzt viele neue Möglichkeiten.

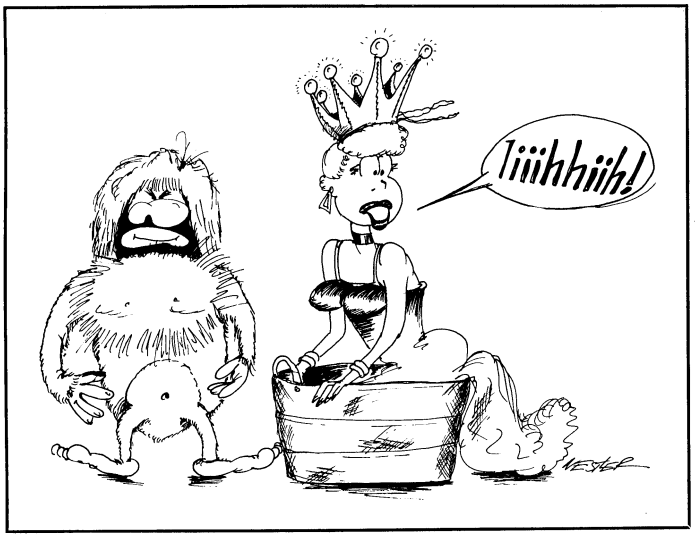
a) Der Spieler öffnet die Kiste:

— Der Geist erschreckt und flieht.
— Der Geist erschreckt den Spieler zu Tode.
— Die Kiste ist leer. Warum? Ganz einfach — dem Geist ist es zu langweilig in der Kiste geworden und er hat sie wieder verlassen, bevor der Spieler den Raum betreten hat, und ist nun bereits in einem anderen Raum.

b) Der Spieler öffnet die Kiste nicht, bleibt aber im Zimmer:
— Der Geist kommt aus der Kiste heraus...
— Der Spieler hört plötzlich ein Schluchzen und Weinen. Warum? Ganz klar, der dumme Geist hat sich selbst in der Kiste eingeschlossen, beziehungsweise diese ist zugeschnappt, er ist gefangen und bittet Sie, ihn herauszulassen.
Der Spieler hilft dem Geist:
I. Der Geist belohnt den Spieler großzügig
II. Der böse Geist tötet den Spieler trotzdem

c) Der Spieler gibt dem Geist einen Kuß:
— Es passiert nichts.
— Der Geist verschwindet empört.
— Abrakadabra — der Geist war gar kein Geist, sondern eine atemberaubend hübsche, verzauberte Prinzessin (Ist die Kiste enorm groß?).

d) Der Spieler öffnet die Kiste und erwischt den Geist beim Flirten mit einer Hexe.



Sie sehen also, wie viele Möglichkeiten in der Situation »Kiste, Fenster, Geist und Spieler« stecken, und es gibt noch tausende mehr.

Sicherlich haben Sie schon eigene Ideen ausgeheckt— was beweisen würde, daß Sie bereits Fortschritte machen.

Falls Sie noch Anregung benötigen, so stellen Sie sich einfach noch eine zusätzliche Person zu unserer Ausgangssituation vor.

Was passiert,
— wenn der Spieler auf diese Person trifft?
— wenn diese Person auf den Geist trifft?
— wenn sich die Person (zum Beispiel Hexe) und der Geist verbünden und gemeinsam gegen den Spieler vorgehen?

Nun aber vorerst genug des Geredes über Geister, Kisten etc. Schließlich sind wir schon fast am Ende dieses Kapitels angelangt, und unsere Frage, wieviel Adventure denn nun in 38 KByte passen, ist noch immer nicht beantwortet.

Es hat natürlich seinen Grund, warum ich die Antwort auf meine Frage so lange hinausgezögert habe — ich weiß keine Antwort.

Hoffentlich sind Sie mir nicht böse, aber die letzten Seiten sollten Ihnen nur begreiflich machen, warum eine konkrete Antwort unmöglich ist.

Vielleicht fragen Sie sich jetzt, warum jemand überhaupt eine Frage stellt, wenn er selbst keine Antwort zu geben vermag.

Auch dies hat seinen Grund!
Gerade ein Anfänger beziehungsweise jemand, der zum ersten Mal ein Adventure schreibt, ist nicht oder nur sehr schlecht in der Lage, abzuschätzen, wieviel Speicher für das Spiel benötigt wird, das er sich ausgedacht hat. Dieses Abschätzen will auch gelernt sein.

Doch dazu mehr, wenn wir den Computer einschalten und die ersten kleinen Programmteile entwickeln.

2. Kapitel: Ohne Drehbuch kein Film

Am ehesten kann man das Schreiben von Abenteuerspielen wohl mit dem Drehen eines Films vergleichen.

Während an einem Film jedoch viele mitarbeiten, müssen wir alle Arbeiten ganz alleine verrichten: Wir sind Drehbuchautor, Regisseur, und Kameramann in einem, um nur die wichtigsten zu nennen.

Lange bevor ein Filmregisseur zu drehen beginnen kann, braucht er zunächst eine Idee, was für einen Film er drehen will.

Genauso muß der Programmierer eines Adventures zuerst eine Idee aufgreifen, aus der später das Spiel entstehen soll. Dies ist wahrscheinlich der schwierigste Teil von allen. Es ist nicht leicht, ein wirklich neues und möglichst originelles Spielthema zu finden. Denken Sie hierbei nur einmal an die zahlreichen Spiele, bei denen eine Burg oder ein Schloß im Mittelpunkt steht: Zauberschloß, Burg des Schreckens, Schloß des Grauens etc. Genauso viele Spiele finden in Höhlen (Caves) statt.

Sie sollten deshalb bei ihren Überlegungen ein Spiel anstreben, das sich in einem ganzen Abenteuerland und nicht in einer Burg abspielt.

Im Prinzip sind folgende Spieltypen möglich:
— Märchen (Beispiel: Der Hobbit)
— Science-Fiction
— modernes Adventure (Beispiel: Dallas Quest)

Unter »Märchen« versteht man ein Spiel mit Zauberern, Hexen, Drachen etc. Bei »Science-Fiction« könnte es sich um ein Weltraum- oder ein Zukunftsabenteuer handeln.

Ein »modernes Adventure« spielt sich in unserer Zeit ab.

Selbstverständlich können sich diese Themen auch überschneiden, zum Beispiel wenn man mit einer Zeitmaschine reist.

Alle Möglichkeiten ausführlich zu erläutern, würde den Rahmen dieses Kurses sprengen. Ich will jedoch nicht fortfahren, ohne Ihnen als Beispiel einige Gedanken vorzuführen.

Angenommen, ich möchte ein Spiel schreiben, das sich in der Vergangenheit abspielt. Ich brauche also zu Beginn des Spiels eine Zeitmaschine, mit der der Spieler in die Vergangenheit reisen kann. Da es in der heutigen Zeit jedoch noch keine Zeitmaschine gibt, muß ich den Anfang des Spiels in die Zukunft verlegen. Unser erster Spielgedanke könnte sein: Im Jahre 2000 wird eine Zeitmaschine mit zwei Mann Besatzung in die Vergangenheit geschickt. So schön, so gut. Der Spieler gehört also zur Besatzung der Zeitmaschine. Damit er nicht alleine ist, geben wir ihm einen Begleiter, einen Wissenschaftler, mit auf den Weg (dieser Wissenschaftler wird im Programm als selbstdenkender Spielcharakter behandelt). Natürlich müssen wir dem Spieler auch eine Aufgabe mit auf den Weg geben. Sein Ziel könnte sein, herauszufinden, warum die Dinosaurier damals ausgestorben sind.

Damit hätten wir unseren ersten Gedanken — die Ausgangssituation — abgeschlossen. Ist dieser erste Schritt getan, so ist es nicht mehr sehr schwer, weitere Gedanken auszuhecken. Unser Spieler fliegt in die Vergangenheit. Nun müssen wir uns ausdenken, was ihn dort erwartet. Vielleicht

trifft er auf Steinzeitmenschen, oder gar Außerirdische. Er könnte von den Steinzeitmenschen gefangen werden. Nun sind Sie an dem Punkt angelangt, bei dem Sie sich voll und ganz in Ihre Fantasie stürzen müssen. Sicherlich kommen dann zahlreiche Ideen, was dem Spieler alles passieren kann, und wie er es zu meistern hat.

3. Kapitel: Von der Idee zum Spiel

Lange bevor ein Spiel in ein Programm umgesetzt wird, muß es in Form von Skizzen und Tabellen nahezu völlig fertig vorliegen.

Natürlich gibt es auch viele Programmierer, die sich an den Computer setzen und einfach drauflostippen — nach dem Motto erst Tippen, dann Denken.

Selbstverständlich kann auf diese Weise nur äußerst selten ein vernünftiges Programm entstehen:

Zum einen kommen Spiele dabei heraus, die so viele stilistische und programmbedingte Fehler enthalten, daß sich ein Spielen kaum noch lohnt. Solche Spiele wandern dann in die Schubladen und werden schnell vergessen. Zum anderen wird beim Programmieren nach einiger Zeit der Überblick verloren, so daß Korrekturen am Programm nur noch sehr mühselig vorzunehmen sind. Dann geht schnell die Lust am Weiterprogrammieren verloren, und das Spiel wandert ebenfalls in die Schublade.

Einige könnten nun einräumen, daß sie ihre Programme äußerst gut strukturiert haben und der Überblick somit gewährleistet ist.

Das mag schon stimmen, soweit es sich um die Programmierung eines Schießspiels etc. handelt. Jedoch aus eigener Erfahrung kann ich Ihnen versichern, daß es unmöglich ist, ein 38 KByte-Adventure bis auf die letzte Variable zu strukturieren — irgendwann findet man sich auch in der besten Struktur nicht mehr zurecht, außerdem sind strukturierte Programme oft viel länger als herkömmliche und verbrauchen somit den, besonders bei Adventurespielen, benötigten Speicherplatz.

Natürlich möchte ich von Ihnen nicht verlangen, jedes Adventure vor der Umsetzung zum Programm bis ins letzte, klitzekleinste Detail auszuarbeiten. Sie sollen vielmehr erfahren, welche Unterlagen man stets beim Programmieren neben sich liegen haben sollte, um problemlos und schnell Änderungen am Spiel vornehmen zu können.

Generell empfehle ich Ihnen für jedes Adventure ein Heft (Format A4) zu führen, in dem alle wichtigen Informationen und Tabellen zum Spiel enthalten sind.

Nun aber zum eigentlichen Thema. Am Anfang aller Arbeit steht, wie schon die Überschrift des Kapitels verrät, die Idee.

Wie kommt man nun aber zu einer guten, neuen Idee?

Hierzu bietet es sich zunächst an, sich einmal auf dem Softwaremarkt umzuschauen und dann einige Adventures zu spielen. Gute Adventures die ich Ihnen empfehlen kann, wären zum Beispiel »The Hobbit«, »Ulysses« und »Enchanter«. Äußerst interessant ist auch »Gordon Saga«. Dieses Spiel enthält völlig neue Elemente und zeichnet sich außerdem durch einen nahezu unerschöpflichen Wortschatz und unbegrenzte Spielmöglichkeiten aus. Natürlich sollen Sie diese Spiele nicht spielen, um die in ihnen enthaltenen Ideen in Ihrem eigenen Spiel zu kopieren — Sie sollen aus diesen Spielen lernen, wie andere Autoren ihre Ideen ausgebaut haben. Es macht auch nichts aus, wenn Sie an ein extrem schlechtes Adventure geraten, denn Sie werden aus den Fehlern dieser Spiele lernen und umzugehen, daß sich diese Fehler in den eigenen Spielen wiederholen.

Oft wird unter einem Adventure ein Spiel verstanden, in dem wie in Märchen Zauberer, Zwerge, Feen und Monster auftreten. Leider ist dieser Gedanke kaum zutreffend.

Unter Abenteuer sollte man vielmehr Erleben verstehen — Adventures sollten deshalb nicht nur Spiele sein, in denen der Spieler einen Zauberer besiegen oder Schätze finden muß. Ein Adventure ist einfach ein Spiel, das der Spieler selbst beeinflussen kann und dabei einiges erlebt.

Sehen Sie sich doch einmal die folgenden Themen an:

1. Ein kleiner Wanderzirkus kommt in die Stadt. In der Nacht bricht ein gefährlicher Löwe aus und macht die Stadt unsicher.
   Die Aufgabe des Spielers besteht nun darin, den Löwen zu finden und zu fangen.
2. Auf einem riesigen Reiseschiff (à la Traumschiff) befindet sich eine Gräfin. Aus ihrer Kabine werden plötzlich die gesamten Juwelenschmuckstücke entwendet. Der Täter kann das Schiff erst verlassen, wenn es im nächsten Hafen anlegt.
   Aufgabe des Spielers ist den Dieb zu entlarven, bevor die Schiffsreise beendet ist.
3. Eine Flaschenpost wird gefunden, die eine Seekarte enthält, auf der eine bisher unbekannte Insel eingezeichnet ist.
   Aufgabe des Spielers: Zur Insel reisen und sie erforschen.
4. Aufgabe des Spielers: Per Autostop von Frankfurt nach Rom zu trampen.
5. Man schreibt das Jahr 2002. Ein Raumschiff wird, mit tiefgefrorener Besatzung, zu einer langen Reise ins Weltall geschickt. Viele Jahre später landet das Schiff wieder auf der Erde, die sich völlig verändert hat. Der Spieler ist der Kommandant des Sternenschiffes.
   Aufgabe: Herausfinden, was mit der Erde geschehen ist.

Diese fünf Themen sind Beispiele, wie die zündende Idee zu einem Spiel aussehen könnte. Ich habe solche Beispiele gewählt, die noch einen Bezug zur Realität haben — sich also in unserer Zeit und Welt abspielen könnten.

Das einzige Problem besteht nur noch darin, diese zündende Idee zu haben. Glücklicherweise ist es einfach, sich Anregungen zu verschaffen. Es gibt zum Beispiel eine riesige Menge an spannenden Abenteuerbüchern.

Ich möchte Sie auffordern, sich einmal in Büchereien und Bibliotheken oder Buchhandlungen umzusehen. Sie werden sicherlich ein Buch finden, von dem Sie sich Anregung zu einem Spiel versprechen. Empfehlenswert sind auch die Fantasy-Buchreihen der verschiedenen Buchverleger.

Weitere Anregungen können Sie durch Kinofilme finden — denken Sie nur an die verschiedenen Filme, von denen bereits Programme entstanden sind (zum Beispiel »Der dunkle Kristall« oder »Ghost Buster«).

Wichtig ist, daß Sie wissen, daß es nicht nur auf die Ausgangsidee ankommt und diese deshalb einzigartig gut sein muß, sondern vielmehr darauf, was Sie aus Ihrer Idee machen. Wir wollen uns die fünf Themen, die ich Ihnen vorgestellt habe, nun noch einmal genauer betrachten:

1. Auf den ersten Blick scheint in der Idee, »Spieler muß ausgerissenen Löwen fangen«, nicht allzuviel zu stecken.
   Sicher ist auf jeden Fall, daß der Spieler einige Aufgaben zu bewältigen hat, bevor er den Löwen fangen kann.
   Der Spielort: Eine große Stadt. Damit das Spiel nicht zu monoton wird (der Spieler also einfach alle Straßen der Stadt abläuft, bis er den Löwen findet), setzen wir noch einige Parks oder Grünanlagen in die Stadt.
   Um den Löwen zu fangen, braucht der Spieler einen Käfig und frisches Fleisch, um den Löwen anzulocken. Den Käfig kann er nur mit einem Lastwagen befördern, den der Spieler zum Beispiel mieten muß. Der Lastwagen braucht natürlich ab und zu Diesel. Alle diese Dinge kosten jedoch Geld — und das hat der Spieler noch nicht. Vielleicht gibt es in der Stadt jedoch eine Spielhölle oder eine freundliche Bank, wo sich der Spieler

Geld leihen kann. Vielleicht gibt es aber auch ein paar Diebe, die auf das erworbene Geld scharf sind? Was passiert, wenn der Lastwagen unterwegs kaputt geht? Solche und viele andere kleine Zwischenfälle können dem Spieler widerfahren, bis er den Löwen endlich im Käfig hat.
Sie müssen zugeben, daß sich mehr und mehr ein Spiel aus unserer bescheidenen Anfangsidee entwickelt.
Natürlich müssen noch viele weitere Einfälle realisiert werden, bis man das Spiel als nahezu vollendet bezeichnen kann. Je mehr man sich jedoch in die Materie vertieft, desto einfacher wird es, die Gedanken zu fassen.

2. Lassen Sie uns noch einmal das zweite Beispiel betrachten. Der Spielort ist eindeutig ein Schiff — also Kabinen, Speisesaal, Aufenthaltsräume, Decks etc. Die Handlung kann man mit der eines Kriminalfalls vergleichen. In dieser Art von Adventure müssen also viele Personen mitspielen, die alle Motive und Alibis haben. Der Spieler kann die Personen beschatten und befragen — er übernimmt in etwa die Rolle eines Detektivs. Während der Suche nach dem Täter kann zur Steigerung der Spannung ein Mordanschlag vorkommen (dieser kann, muß aber nicht unbedingt gelingen) — zum Beispiel auch ein Mordanschlag auf eine andere Person auf dem Schiff (nicht den Spieler), die den Täter kennt und somit nach dessen Ansicht zu viel weiß.
Ein Spiel dieser Art in ein Programm zu packen, ist durchaus möglich, jedoch auch relativ schwierig. Das Programm muß alle Personen (außer den Spieler) steuern. Die Personen müssen sich bewegen, handeln und auf den Spieler eingehen (reagieren) können. Wie man solche kniffligen Programmteile schreibt, erfahren Sie im Programmierteil. Zunächst sollten Sie sich bei Ihren ersten Adventures jedoch auf Spiele beschränken, in denen nur möglichst wenige Personen auftreten.
3. Beispiel 3 ist typisch für ein Fantasie-Adventure. Durch eine Seekarte findet der Spieler den Weg zu einer unbekannten Insel. Man kann den Spieler — und damit das Adventure — gleich auf der Insel beginnen lassen, oder den Spieler erst eine Schiffsreise antreten lassen. Auf der Insel leben dann vielleicht noch längst für ausgestorben gehaltenene Tiergattungen (Saurier), sowie Steinzeitmenschen. Auf der Insel kann es Höhlensysteme, Flüsse, Dschungel, Berge, Seen und vielleicht auch einen Vulkan geben. Natürlich birgt die Insel ein Geheimnis, das der Spieler entdecken muß. Am einfachsten wäre es, einfach einen Schatz zu verstecken. Man kann dem Spiel jedoch auch Utopie verleihen: Der Spieler entdeckt im Inneren des Vulkans eine technisch hochentwickelte Apparatur, die die Lebewesen auf der Insel steuert — die Lebewesen erweisen sich hier als perfekte Roboter. Wer hat jedoch die Apparatur aufgestellt, und welchen Zweck beabsichtigt er? — Außerirdische? Ein verrückter Professor? Vielleicht ist die Insel nur eine Tarnung für eine geheime Raumschiff-Abschußstation.
Anhand dieses Beispiels können Sie gut erkennen, wie man überraschende Effekte erzielt, durch die der Spieler, der eigentlich glaubt, er müsse nur einen vergrabenen Schatz finden, völlig verblüfft wird.
4. Die Ausarbeitung dieses Themas erfordert besonders viel Einfalls- und Ideenreichtum. Auf den ersten Blick scheint auch in diesem Ausgangsthema nicht viel Abenteuer zu stecken. Tatsache ist jedoch, daß gerade diese Problematik uns extrem viele Möglichkeiten bietet.
Sicherlich ist es langweilig, wenn der Spieler nur die Aufgabe hat, an der Straße zu stehen und auf ein Auto zu warten, das ihn mitnimmt. Am Anfang des Spiels wollen wir den Spieler deshalb mit etwas Geld versehen und in ein Geschäft schicken, in dem er sich eine Ausrüstung kaufen kann. In dem Geschäft gibt es alles — vom Schlafsack über Kleidung bis hin zur Zahnbürste. Unser Spieler ist also ausgerüstet und begibt sich zur Autobahn, von der er lostrampen will.
Nun wird es Zeit, sich über den Fortgang des Spiels und dessen Aufbau Gedanken zu machen.
Wir wollen einmal festlegen, daß der Spieler 8 Tage Zeit für sein Unternehmen hat. Des weiteren legen wir fest, daß der Spieler während des Spiels niemals umkommt. Er muß höchstens kapitulieren, wenn ihm das Geld ausgeht, er abends keinen günstigen Schlafplatz findet (er hat auf keinen Fall genug Geld für ein auch noch so billiges Hotel bei sich) oder das Essen knapp wird.
Der Spieler steht also auf der Straße und hofft, daß ein Auto hält und ihn mitnimmt. Wenn ein Auto hält, sind folgende Dinge möglich:
— das Auto fährt in Richtung Rom
— das Auto fährt in eine abweichende Richtung
Der Spieler hat nun die Wahl, ob er mitfahren will oder nicht. Wir lassen mehrere mögliche Routen zu — welche möglich sind, kann man leicht durch einen Atlas mit Reisekarte erfahren.
Es ergeben sich hier also schon zahlreiche Kombinationsmöglichkeiten. Allerdings darf nicht alles dem Zufall überlassen werden. Wir wollen nun noch festlegen, daß der Spieler immer von einer Großstadt zur anderen gelangt. Eine mögliche Route wäre Frankfurt—Nürnberg—München—Verona—Florenz—Rom. Übernachten kann der Spieler in öffentlichen Parkanlagen, Bahnhöfen etc.
Nun müssen wir uns noch einfallen lassen, was ihm alles zustoßen kann. Er könnte einen Teil seines Geldes verlieren oder es wird gestohlen. Er kann gute und schlechte Bekanntschaften machen.
Sicherlich haben auch Sie schon Dinge erlebt, aus denen man ein Adventure machen kann. Ich will damit sagen, daß ein Adventure nicht unbedingt in einer anderen Welt oder Zeit ablaufen muß, um als Adventure anerkannt zu werden — lassen Sie Ihrer Fantasie freien Lauf...
5. Bei diesem Beispiel sollen Sie sich nun einmal selbst ein paar Gedanken machen. Versetzen Sie sich dazu einfach in die Lage des Spielers und verfahren Sie nach einem Gedankenschema, wie ich es in den letzten vier Beispielen vorgeführt habe.

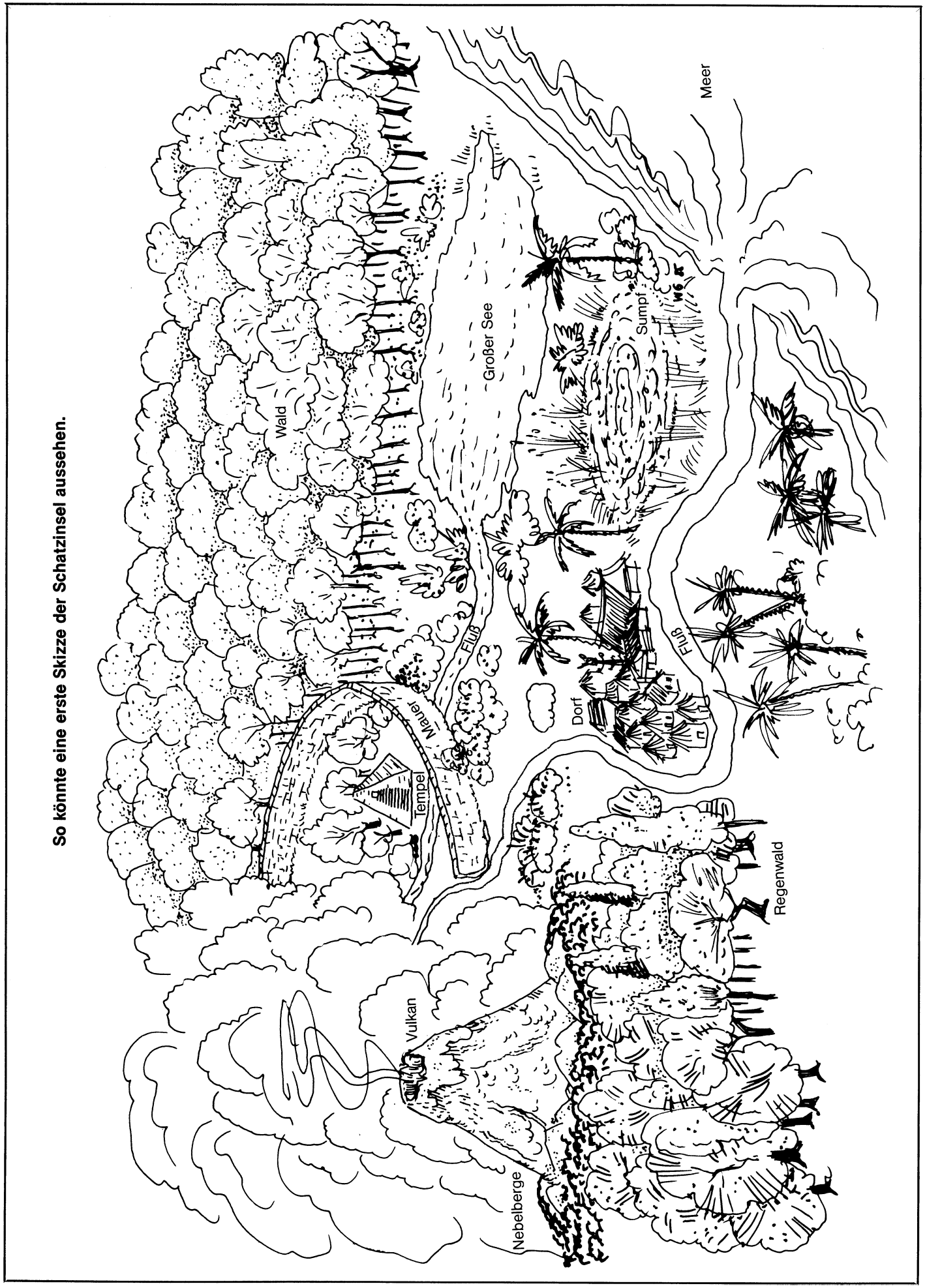
Nun möchte ich einmal ein Beispiel herausgreifen, an dem wir lernen wollen, welche Tabellen man erstellen muß und wie man eine Adventure-Karte zeichnet.

Wir nehmen dazu Beispiel 3 — Die geheimnisvolle Insel. Den zündenden Gedanken hatten wir ja bereits — aber wie arbeiten wir ihn zum Spiel um? Jetzt gleich Aufgaben und Spielwitz zu suchen, ist sehr schwierig. Wir wollen uns deshalb zunächst einmal Gedanken über die Umgebung der Insel machen.

Dazu denkt man einfach nach, welche Dinge auf einer Insel vorkommen. Zum einen ist da der Strand — das soll später auch die Stelle der Insel sein, bei der das Spiel beginnt. Der Spieler rudert mit einem kleinen Boot vom großen Schiff aus zum Strand. Vom Strand aus könnte er vielleicht in einen tropischen Regenwald oder Dschungel gelangen. Außerdem soll es auf unserer Fantasie-Insel ein Gebirge geben (die Insel ist sehr groß), in dessen Zentrum ein riesiger Vulkan aufragt. Außerdem erstrecken sich mehrere Flüsse über die Insel, deren Quellen im Gebirge liegen. Wo Flüsse und Vulkane sind, muß es eigentlich auch heiße Quellen und Lagunen geben.

Damit die Insel auch Leben erhält, soll sich auf ihr auch ein

So könnte eine erste Skizze der Schatzinsel aussehen.



kleines Dorf mit Eingeborenen befinden. Vielleicht auch mehrere Dörfer unterschiedlicher Stämme, aber dies soll uns zunächst nicht so sehr beschäftigen. Was wir nun brauchen ist eine erste Skizze. Man nehme also einfach einmal ein Blatt Papier und Stifte und male dann ein Bild, das in etwa wie auf Seite 12 aussehen könnte.

Es muß sich bei dieser ersten Skizze nicht um ein grafisches Meisterwerk handeln, an dem man stundenlang zeichnen muß. Es soll vielmehr ein Bild sein, das man mit einem Bleistift schnell auf ein Stück Papier kritzelt. Dieses Bild hat natürlich noch sehr wenig mit einer richtigen Adventure-Spielkarte gemeinsam. Die Größenverhältnisse auf dem Bild sind ebenfalls sehr vage. Es soll also nur ein Bild sein, das einen Uberblick über die angestrebte Adventure-Landschaft darstellt. Man kann an ihm auch in etwa erkennen, wo was liegt. Bei der vorgestellten Skizze von unserer Insel sind folgende Interpretationen möglich:

Ein riesiges Gebirge erstreckt sich über den westlichen Teil der Insel. Im Zentrum des Gebirges steht ein Vulkan. Vom Gebirge gehen auch mehrere Flüsse aus, die sich über die ganze Insel verzweigen. Einer dieser Flüsse mündet in einen großen See, der sich im östlichen Teil der Insel befindet. An einem der anderen Flüsse, der ins Meer fließt, liegt ein kleines Dorf mit Eingeborenen.

Im nordwestlichen Teil, am Fuß des Gebirges, befindet sich eine riesige Mauer. Hinter der Mauer liegt ein Tempel versteckt. Der restliche Teil der Insel besteht aus Wäldern und Sümpfen.

Sie haben sicher bereits erkannt, worin der Sinn unserer ersten Skizze liegt — man gewinnt einen bildlichen Eindruck über die Gegend, in der das Adventure später stattfinden soll. Des weiteren findet man im Rahmen der Bildbetrachtungen viele neue Ideen und Anregungen über den Verlauf des Adventures.

Worüber wir uns bisher noch keinerlei Gedanken gemacht haben, ist der Spielwitz — sprich die Aufgaben, die der Spieler lösen muß.

Der nächste Schritt besteht nun darin, die eigentliche Spielkarte zu entwerfen, mit deren Hilfe wir das Spiel später programmieren werden. Man kann darüber streiten, ob es besser ist, zuerst den Spielwitz und dann die Spielkarte auszuarbeiten oder umgekehrt. Ich halte es für sinnvoll, wenn man beides gleichzeitig behandelt, da Spielkarte und Spielwitz sehr eng miteinander zusammenhängen.

Eines der größten Probleme ist das folgende:

Wenn man die Spielkarte erst einmal ausgearbeitet hat, so ist es oft sehr schwierig und mühselig, später noch einmal Änderungen vorzunehmen.

Während man jedoch das Spiel programmiert, kommen einem noch viele Ideen, die den Spielwitz betreffen und die man noch mit einbauen will. Deshalb muß die Spielkarte sehr sorgfältig und sauber gezeichnet werden, damit die Übersicht nie verloren geht.

Wie geht man nun beim Entwurf der Spielkarte vor?

Die Gedankengänge zum Entwurf einer Spielkarte sind genau die Umkehrung derer, mit denen man ein Adventure löst.

Sicherlich haben auch Sie schon einmal ein Adventure gespielt und sich dabei Notizen gemacht, wo was ist. Es gibt natürlich auch Spieler, die sich keine Notizen machen, im Spiel immer wieder hin- und herlaufen und die Orientierung und somit die Freude am Spiel schnell verlieren.

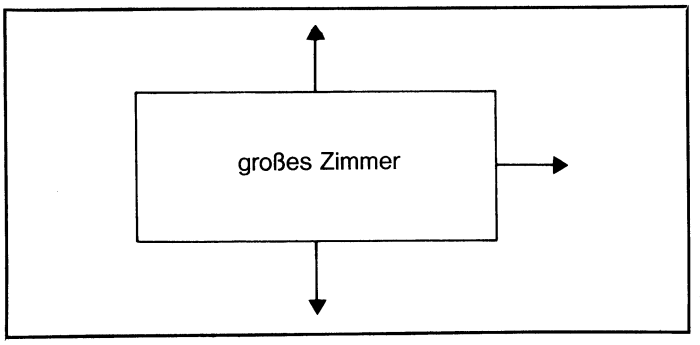
Ein gewissenhafter Abenteurer, der ein Adventure bis zum Ende bestehen will, geht jedoch ganz anders vor — er zeichnet einen Lageplan zum Spiel.

Wie macht er das?

Wir gehen einmal vom Beginn eines Spiels aus. Der Computer beziehungsweise das Programm liefert seinen ersten Lagebericht, der so lauten könnte:

SIE BEFINDEN SICH IN EINEM GROSSEN ZIMMER
IM NORDEN BEFINDET SICH EINE TUER, DIE OFFEN IST
MOEGLICHE RICHTUNGEN: S, N, O

Nun macht sich der gewissenhafte Abenteurer seine erste Notiz, die so aussieht:



Er macht also ein Kästchen, in dem er mit einem kurzen Wort den jeweiligen Raum beschreibt. In diesem Fall also »großes Zimmer«.

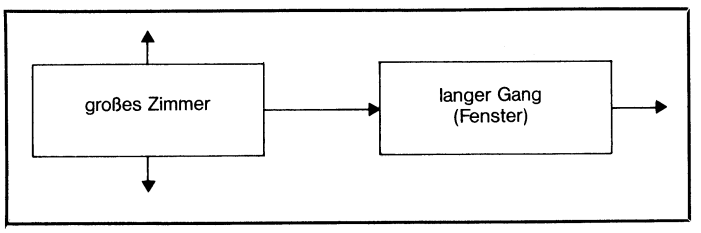
Von dem Kästchen gehen Pfeile in die verschiedenen möglichen Richtungen. Dabei ist es wichtig, daß auch Pfeile — nicht etwa nur Linien — markiert werden. Der Pfeil zeigt immer in die Richtung, in die man von diesem Raum aus gehen kann. Wenn man zum Beispiel nach Norden läuft, so ist es nicht sichergestellt, ob man anschließend durch Bewegung nach Süden wieder zum Ausgangsort gelangt. Deshalb also zunächst nur einen Pfeil.

Der nächste Schritt besteht also darin, die einzelnen Richtungen auszuprobieren. Wir wollen einmal nach Osten gehen.

Das Programm liefert daraufhin folgenden Lagebericht:

SIE BEFINDEN SICH IN EINEM LANGEN GANG
AM ENDE DES GANGES IST EIN KLEINES FENSTER
MOEGLICHE RICHTUNGEN: W, O

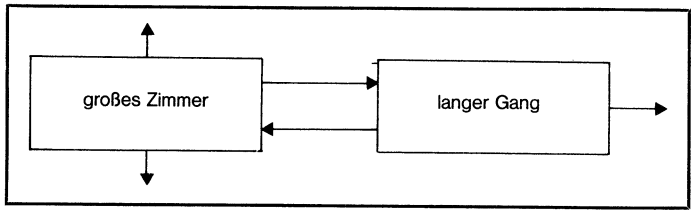
Damit können wir den Lageplan folgendermaßen ergänzen:



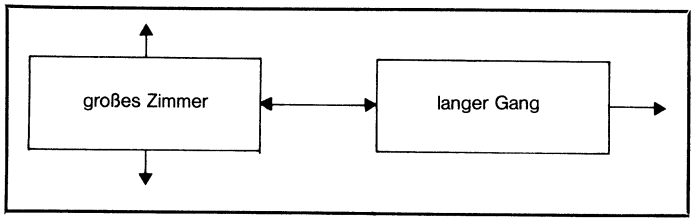
Aus der Lageberichtserstattung des Programms geht hervor, daß man nach Westen und nach Osten laufen kann (vom langen Gang aus).

Logischerweise müßte man nun eigentlich wieder in das große Zimmer gelangen, wenn man zurück nach Westen

läuft. Wir wollen dies nur ausprobieren und laufen zurück in westliche Richtung — und tatsächlich gelangen wir wieder in das große Zimmer. Wir können also einen zweiten Pfeil in die Skizze einzeichnen.



Sinnvoller ist es natürlich, anstelle der beiden Pfeile nur einen Pfeil mit zwei Spitzen einzuzeichnen. Also:



Lassen Sie uns jedoch auch den traurigen Fall betrachten, was passiert wäre, wenn wir nicht wieder in das große Zimmer gelangt wären. Dann wäre es schon etwas verwirrender einen vernünftigen Spielplan zu zeichnen.

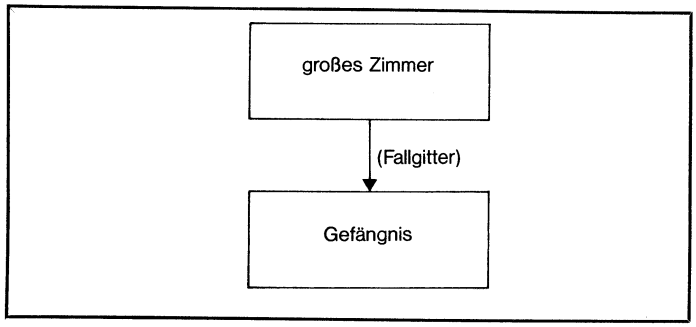
Es scheint viele Adventure-Programmierer zu geben, denen es eine wahre Freude bereitet, den Spieler durch unlogischen Pläne zu verwirren.

Auch in einigen sehr guten Adventures, zum Beispiel »The Hobbit« wird dieses Verwirrspiel getrieben. Eigentlich schade, denn bei einem guten Adventure, das vom Spieler viel Ideenreichtum bis zur Lösung erfordert, kann es äußerst frustrierend für den Spieler sein, wenn der Spielplan völlig unlogisch aufgebaut ist. Deshalb auch mein Rat an die angehenden Adventure-Programmierer: Schreiben Sie stets nur Adventures mit logischem Spielplan. Das Hauptziel eines Adventures soll schließlich sein, den Spieler zu unterhalten und nicht zu frustrieren. Der Spieler sollte so wenig wie nur irgend möglich vom Spiel abgelenkt werden, weder durch ständige Fehlermeldungen, die auf einen mangelhaften Wortschatz des Spiels schließen lassen, noch durch einen Spielplanaufbau, in dem man sich kaum zurechtfinden kann. Natürlich gibt es auch Fälle, bei denen man tatsächlich nicht mehr umkehren kann, um wieder an den Ausgangsort zu gelangen.

Dies ist dann der Fall, wenn

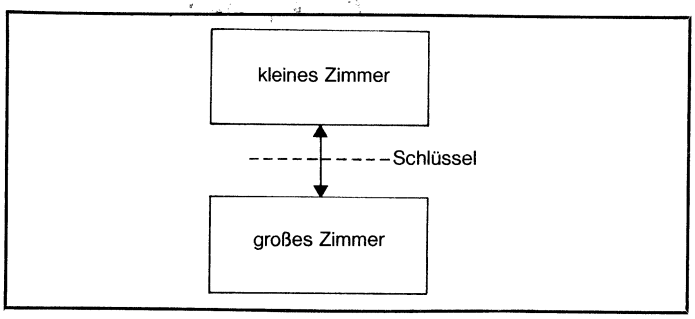
— man in eine Grube hinabstürzt.
— plötzlich ein Fallgitter herunterfällt und den Ausgang versperrt.
— hinter dem Spieler plötzlich eine Tür zufällt, die sich nicht mehr öffnen läßt.

Eine Szene, bei der ein Fallgitter herunterfällt, würde man folgendermaßen skizzieren:



Hier wurde an dem Pfeil noch kurz vermerkt, warum eine Umkehr nicht möglich ist. Wir wollen festlegen, daß in einem Fall, bei dem eine Umkehr unmöglich ist, immer eine kleine Begründung neben den Pfeil geschrieben wird. Zu dieser Begründung genügt meistens ein einziges Wort, wie Fallgitter.

Oftmals befinden sich Türen zwischen einzelnen Räumen. Diese Türen werden im Plan so skizziert:



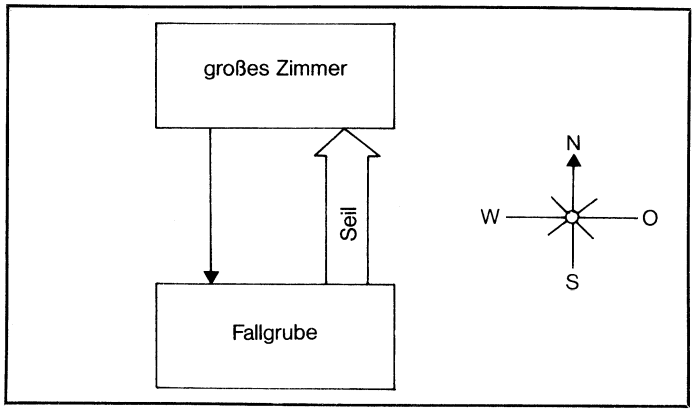
Wie Sie sehen, wird die Tür durch eine gestrichelte Linie dargestellt. Bei dieser Linie steht wieder eine Bemerkung: Schlüssel.

Diese Bemerkung stellt also die Bedingung dar, mit deren Hilfe man durch die Tür gelangt: Man braucht den Schlüssel, um durch diese Tür gehen zu können. Im Fall einer magischen Tür, bräuchte man vielleicht einen Zauberspruch oder bei einer elektronisch gesicherten Tür eine Identitätskarte.

Steht jedoch keine Bemerkung neben der gestrichelten Linie, so heißt dies, daß man die Tür ohne weiteres passieren kann — sie stellt in diesem Fall also kein Hindernis dar.

Lassen Sie uns nun folgenden Fall betrachten:

Der Spieler stürzt in eine Grube, die er nur unter Zuhilfenahme eines Seils wieder verlassen kann. Hier sieht die Skizze so aus:



Diese Skizze ist so zu verstehen:

Wenn man vom großen Zimmer aus in südliche Richtung läuft, so stürzt man in eine Fallgrube, die man nur mit einem Seil wieder verlassen kann. Die Bedingung »Seil« steht hierbei in einem großen Pfeil, der von der Grube zum großen Zimmer führt.

Man könnte die Bedingung auch neben einen einfachen Pfeil schreiben, durch den großen Pfeil ist der Sachverhalt jedoch besser ersichtlich.

Aus diesem Beispiel läßt sich auch etwas lernen — man sollte den Spieler nie vor vollendete Tatsachen stellen (beziehungsweise in aussichtslose Lagen bringen). Es gibt zahlreiche Adventures, die sich nach einer Bewegung des Spielers mit einem solchen Wortlaut melden:

SIE SIND IN EINE FALLGRUBE GESTUERZT UND HABEN SICH DAS GENICK GEBROCHEN!
WOLLEN SIE NOCHMAL SPIELEN?

Solche Spiele spiele ich in der Regel nur einmal.

Worin besteht der Witz eines solchen Falls?

Da läuft der Spieler herum, ist der Lösung des Adventures auf der Spur und bekommt plötzlich die Meldung, daß er verloren hat.

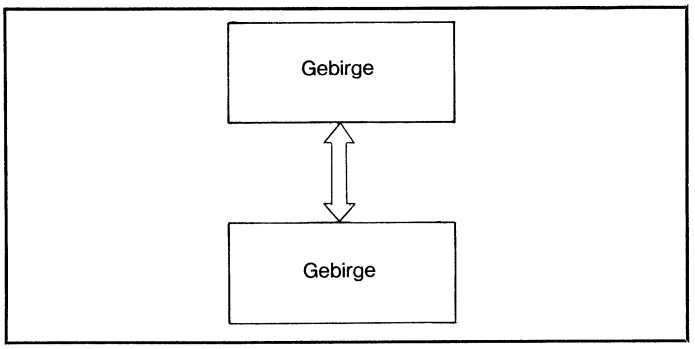
Bei einem guten Adventure sollte der Spieler nur auf Grund von eigenem Fehlverhalten verlieren. Dies müßte eigentlich das Grundmotto eines jeden Adventure-Autors sein.

Damit wären wir schon fast am Ende der Lageplan-Grundlagen.

Scheuen Sie sich nicht davor, zehn Himmelsrichtungen zu verwenden — N, S, O, W, NO, NW, SO, SW, RAUF, RUNTER — denn dadurch lassen sich viel interessantere Pläne erstellen, als mit nur vier Himmelsrichtungen.

Die Richtungen N, S, O, W, NO, NW, SO, SW werden auf der Lagekarte mit einfachen Pfeilen dargestellt. Die Richtungen RAUF und RUNTER hingegen mit Doppelpfeilen, damit keine Irrtümer auftreten können.

Ein Beispiel:



Anhand dieser Beispiele wissen Sie nun also, wie man Landkarten für Adventures zeichnet.

Doch nun zurück zu unserem Inselbild. Wie setzt man diese Skizze denn nun in einen sinnvoll aufgebauten Lageplan um?

Die Insel jetzt gleich in einen riesigen Lageplan umzusetzen, ist nicht sinnvoll. Es empfiehlt sich nun erst einmal Gedanken über den Spielverlauf zu machen.

Wir wollen davon ausgehen, daß sich der Spieler zu Beginn des Spiels auf dem Schiff befindet. Von dort aus begibt er sich mit Hilfe eines kleinen Ruderbootes zur Insel. Er kommt so also zum Strand der Insel.

Nun muß er eine Wald- und Sumpflandschaft durchqueren, um zum Dorf der Eingeborenen zu gelangen. Das Dorf liegt, wie aus der ersten Skizze ersichtlich, an einem Fluß. Trotzdem kann der Spieler mit seinem Ruderboot nicht vom Meer aus über den Fluß zum Dorf gelangen, da die Strömung zu stark ist und dem Meer entgegenfließt. Im Dorf der Eingeborenen, die friedlich sind, erfährt der Spieler ein Geheimnis, das so lauten könnte:

»Der seit Jahren erloschene Vulkan im Nebelgebirge strahlt seit einigen Wochen ab und zu ein merkwürdiges Licht aus. Dies flößt den Eingeborenen große Angst ein. Neugierige, die sich auf den gefährlichen Weg durch das Nebelgebirge zum Vulkan begeben haben, sind nie mehr zurückgekehrt. Außerdem berichtet eine alte Legende von einer riesigen, hunderte Meter langen Mauer, die sich irgendwo im nordwestlichen, unbewohnten Teil der Insel befindet. Niemand weiß, was sich hinter dieser Mauer verbirgt. Abenteurer haben erzählt, daß ein schwarzer, schneller Fluß durch die Mauern zum großen See fließt.«

Sie müssen zugeben, daß dieses Geheimnis schon ziemlich interessant ist. Was ist hinter der Mauer? Was ist mit dem Vulkan los? Wo sind die verschwundenen Abenteurer, die nie wieder heimgekehrt sind?

Diese Frage wird sich der Spieler jetzt stellen — und er wird versuchen wollen, diese Geheimnisse (also das Abenteuer) zu lösen.

Solche Einleitungen zu einem Adventure sind sehr empfehlenswert, da sie das Interesse des Spielers am Spiel steigern, obwohl sie nur sehr oberflächliche Informationen enthalten.

In unserem Insel-Adventure wird die Einleitungsstory im Dorf der Eingeborenen gegeben — also während des Spiels. Sinnvoller ist es allerdings eine solche Einleitung an den Anfang des Spiels zu stellen, man könnte sie dann in ein separates Titelprogramm schreiben. Das heißt, der Spieler lädt erst die Einleitung in den Computer und liest sie. Ist die Einleitung abgelaufen, dann wird das eigentliche Spiel geladen und gestartet. Dieser Tip gilt hauptsächlich für Datasetten-Anwender. Besitzern von Diskettenlaufwerken, die Spiele schreiben können, die selbständig Programmteile nachladen, stellt sich dieses Problem wohl kaum.

Der Spieler kennt also nun die Geheimnisse, die er lüften will. Er weiß natürlich noch nicht, daß sich hinter der hohen Mauer ein Tempel verbirgt. Wir wissen bereits, daß der erste Weg, den der Spieler zurücklegen muß, der Weg vom Inselstrand zum Dorf der Eingeborenen ist.

Aber wohin soll er von dort aus gehen? Im Prinzip haben wir noch zwei Möglichkeiten — das Nebelgebirge mit dem Vulkan und die hohe Mauer.

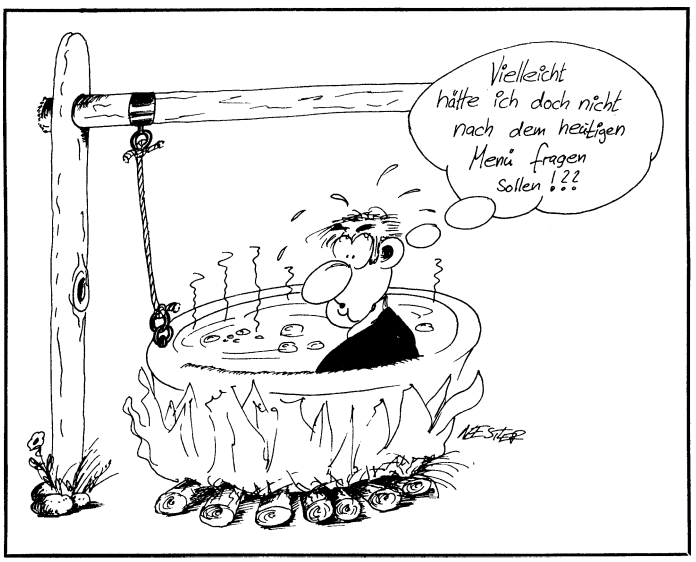
Ich muß hierbei anmerken, daß wir momentan nur den absolut richtigen Weg betrachten, also den Weg, den der Spieler verfolgen muß, um das Spiel so schnell wie möglich zu bezwingen. Irrwege werden im Moment noch nicht berücksichtigt. Dabei möchte ich noch klarstellen, daß der Spieler sich selbstverständlich auf der Insel frei bewegen kann — er kann, wenn er will, vom Dorf auch wieder durch den Dschungel zurück zum Strand, ins Boot steigen und zurück zum Schiff rudern.

Nun aber zurück zum absolut richtigen Weg. Nehmen wir einmal an, der Spieler begibt sich anschließend über das Nebelgebirge zum Vulkan.

Er ergründet dort das Geheimnis des Vulkans. Im Inneren des Vulkans befindet sich eine hochentwickelte, technische Fabrik, die völlig automatisiert ist. Sie wird von Robotersklaven betrieben, deren Herr ein wahnsinniger Professor ist, der die Weltherrschaft an sich reißen will.

Der Spieler ist also in der Fabrik eines Wahnsinnigen. Er erlebt dort wieder einige Abenteuer, und entdeckt schließlich einen Geheimgang, der unter dem Nebelgebirge hindurch zum Tempel hinter der Mauer führt. Der Spieler kommt also in den Tempel und muß dort wieder einige Nüsse knacken. Danach gelangt er über den schwarzen Fluß unter der Mauer hindurch zum großen See und von dort zum Strand und zurück zum Schiff — das Adventure ist gelöst.

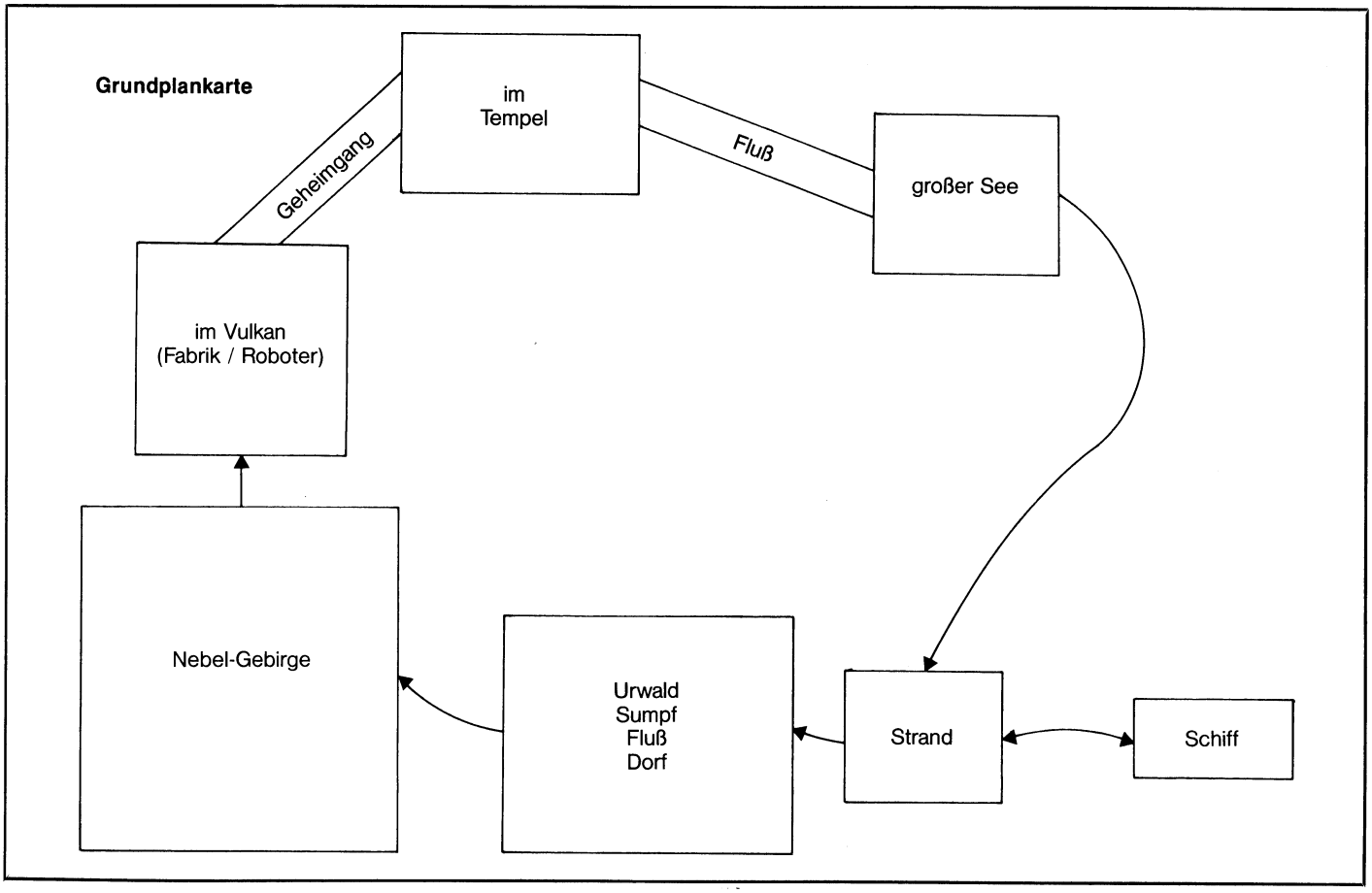
Buh, aufhören, könnten Sie jetzt vielleicht rufen, da unser Spielkonzept zwar schon sehr interessant klingt, jedoch noch keine Details und Action enthält. Wir sind bisher nur durch eine leere Landkarte gelaufen. Es wird bald Zeit, darüber nachzudenken, welche Abenteuer der Spieler beste-



hen muß und in welchen Bereichen der Insel sie versteckt liegen (einige Andeutungen habe ich ja bereits gemacht).

Vorher wollen wir jedoch eine neue Skizze zeichnen, oder besser gesagt unsere alte Inselskizze ein wenig überholen — sie in die Grundlandkarte umsetzen. Darunter versteht man eine Karte, die sehr grob nach dem vorgestellten Kästchen-und-Pfeil-Prinzip angefertigt wird. Diese Karte für unser Spiel sieht folgendermaßen aus:

Das Adventure »The Hobbit« hat zirka 80 Räume. Allerdings ist nicht jeder Raum mit Action gefüllt. Bei etwa 25 Räumen erhält man als Raumbeschreibung immer wieder die Anwort »Sie befinden sich in einem dunklen Höhlensystem.«. Man muß oft lange Strecken zurücklegen, um zu den einzelnen Aufgaben zu gelangen. Adventures, wie man sie aus Zeitschriften vom Abtippen her kennt, haben eine durchschnittliche Raumanzahl von zirka 30 Räumen.



Diese Landkarte ist natürlich noch sehr oberflächlich, zeigt aber dennoch bereits viel deutlicher als unsere erste Zeichenskizze, wie das Spiel aufgebaut ist.

In der Landkarte sind fünf große Bereiche eingezeichnet und ihre Verbindung zueinander. Diese Bereiche sind Urwald, Nebelgebirge, Vulkan, Tempel und der große See. Ich habe also beim Anfertigen dieser Karte nichts anderes gemacht, als die Hauptgebiete der Insel in einzelne Blöcke zu fassen. Von solch einer Blockgebietskarte ist es viel leichter die eigentliche Landkarte anzufertigen, als von der ersten Skizze.

Als kleine Blöcke habe ich den Strand und das Schiff eingezeichnet. Man kann also sagen, daß ein Blick um so mehr Räume enthält, je größer er ist. Zwischen den einzelnen Blöcken befinden sich einfache und doppelte Pfeile. Die einfachen Pfeile stellen die bereits besprochenen Verbindungen zwischen den einzelnen Räumen dar — in unserem Fall, Verbindungen zwischen den einzelnen Gebieten. Die Doppelpfeile sind ebenfalls Verbindungspfeile — mit der Ausnahme, daß sie Bedingungen stellen. So kann man, anhand der Doppelpfeile, aus der Karte lesen, daß man nur über den Geheimgang vom Vulkan zum Tempel gelangen kann. Des weiteren gelangt man vom Tempel nur über den Fluß zum großen See.

Der nächste Schritt besteht nun darin, die einzelnen Gebietsblöcke in einzelne Räume zu gliedern — es entsteht also die komplette Spielkarte.Dieser Ausbau kann je nach Wunsch gering oder sehr komplex sein.

Vielleicht interessiert es Sie, wie viele tatsächliche Räume die bekannten Adventures denn im einzelnen so haben.

Es ist sehr schwer, detaillierte Tips zu geben, wieviele Räume und wieviel Action man in ein gutes Adventure packen sollte. Als Faustregel könnte man vielleicht sagen, daß eine gute Relation zwischen Raumanzahl und Action etwa 1 zu 1 betragen sollte.

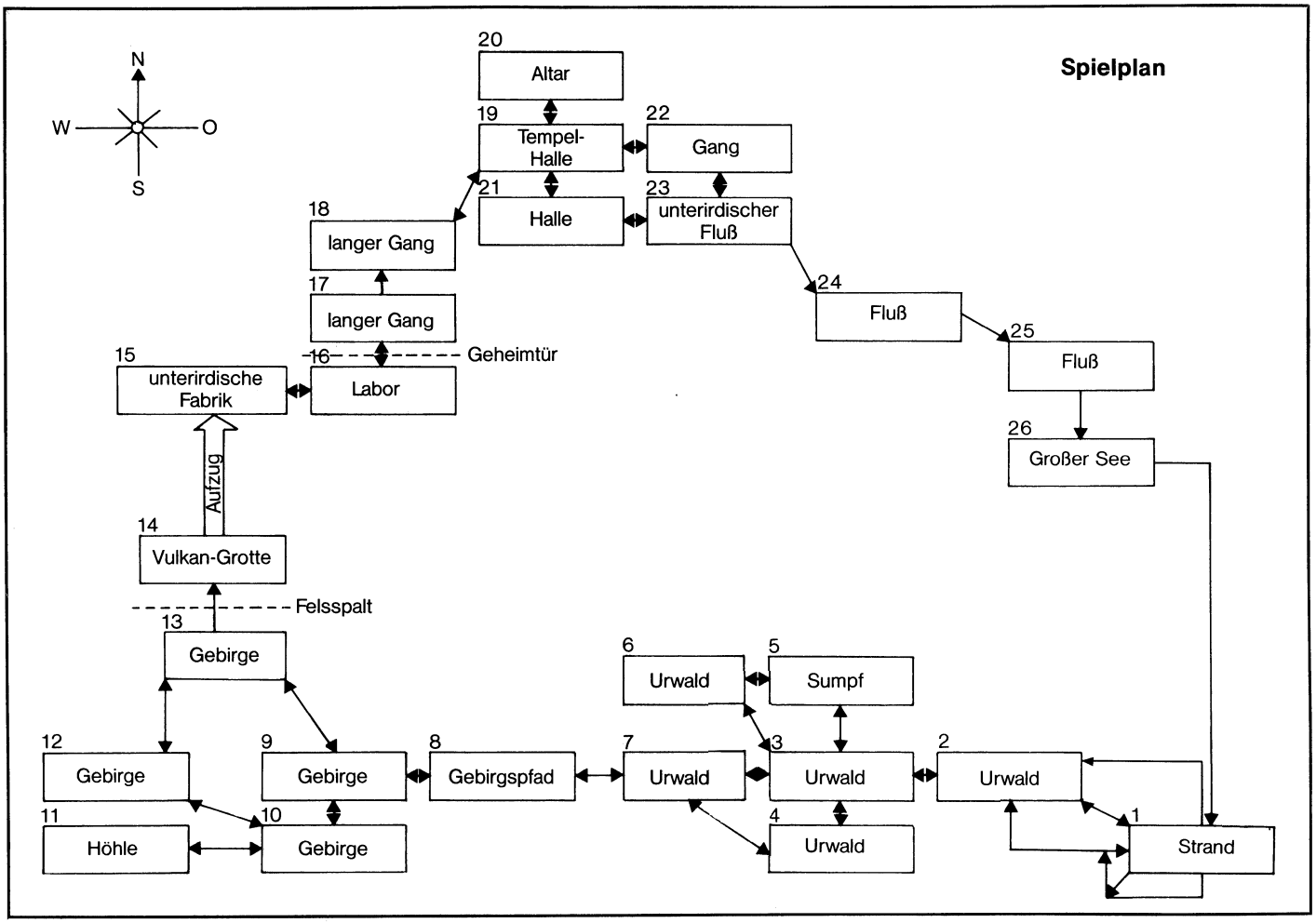
Um nicht allzu sehr vom Thema abzukommen, zurück zur Spielkarte.

Hier der komplette Spielplan, das heißt ein Spielplan, wie man ihn zu diesem Spiel gestalten könnte.

Ich habe mich bei diesem Plan nur auf einen kleinen Aufbau beschränkt, da ein Aufbau im Großen viel zu komplex zu erklären wäre (siehe Spielplan).

Der Plan liegt nun nahezu fertig vor uns. Aus ihm ist eindeutig ersichtlich, welche Räume existieren und wie sie miteinander verbunden sind. Wie ich bereits gesagt habe, ist dieser Plan nur wenig ausgebaut. Ein perfekter Ausbau kann erreicht werden, indem man den Wald vergrößert. Der Spieler könnte durch den Wald zu der hohen Mauer gelangen. Er kann zwar dort nichts anfangen, da er den Tempel nur durch den unterirdischen Geheimgang vom Gebirge aus erreicht, und später durch den unterirdischen Fluß wieder verläßt, aber wir haben in der Vorgeschichte des Spiels die sagenumwobene Mauer erwähnt. Also sollte sie auch im Spiel auftauchen. Des weiteren müßte die unterirdische Fabrik, sowie die gesamten Wandermöglichkeiten im Gebirge und der Tempelplan erweitert werden.

Nun wird es langsam Zeit, sich Gedanken über den Actionverlauf des Spiels zu machen. Woher kommen die Ideen zu



den Aufgaben, die der Spieler später lösen muß? Nun, ganz einfach! Innerhalb eines Kurses, wie diesem hier, ist es schwierig, Gedanken die parallel ablaufen, gleichzeitig im Kurs zu erklären, ohne Verwirrung zu stiften.

Tatsächlich arbeitet man beim Erstellen von Adventure-Spielen immer an mehreren Dingen gleichzeitig:

Während man die Landkarte zeichnet, kommen einem neue Ideen. Betrachten Sie doch die Karte einmal ganz genau! In Raum 11 finden Sie eine Höhle. Von dieser Höhle im Gebirge war bisher noch nie die Rede gewesen. Aber wie kommt sie denn nun in den Plan? Nun, ganz einfach: Während ich die Landkarte für das Gebirge zeichnete, dachte ich mir, daß es eigentlich monoton ist, wenn der Spieler mehrmals die Beschreibung »Sie befinden sich im Gebirge« erhält. So kam ich auf die Idee, daß sich im Gebirge zur Abwechslung auch eine Höhle befinden könnte. Wie wird der Spieler reagieren, wenn er die Lagebeschreibung »Sie befinden sich im Gebirge. Sie sehen einen Höhleneingang!« erhält? Ich versetzte mich also in die Lage des Spielers und überlegte, was ich machen würde, wenn ich ein Adventure spiele und diese Beschreibung erhalte. Nun, ich würde sicher aus Neugier in die Höhle gehen, so wie jeder neugierige Abenteurer es auch macht. Gut, der Spieler geht also in die Höhle. Was soll der Spieler in der Höhle vorfinden? Eine leere Höhle wäre schließlich zu langweilig. Ich beschließe deshalb, in die Höhle einen Gegenstand zu legen, der für den Fortlauf des Spiels sehr wichtig ist. Dieser Gegenstand könnte ein Schlüssel, ein Schwert oder ähnliches sein.

Dadurch, daß ich einen so wichtigen Gegenstand in die Höhle lege, stelle ich auch sicher, daß der Spieler in die Höhle gehen muß — früher oder später.

Selbstverständlich bekommt man bei einem Adventure nur dann etwas, wenn man es sich verdient hat. Der Spieler soll also nicht einfach in die Höhle gehen können, den Gegenstand nehmen und die Höhle dann wieder verlassen. Welche Aufgabe kann ich ihm denn stellen?

Primitive Lösungen wären, den Gegenstand im Boden der Höhle zu vergraben, oder einfach zu verstecken (der Spieler braucht dann eben eine Schaufel). Ich versuche also eine schwierigere Aufgabe zu finden, bei der der Spieler nur durch Nachdenken weiterkommen oder überleben kann.

Und da ist auch schon die Idee geboren: Wilde Bären hausen doch in Höhlen im Gebirge. Nun stelle ich mir die Spielszene einmal vor. Der Spieler entdeckt einen Höhleneingang und geht hinein. In der Höhle, die völlig verlassen ist, findet der Spieler einen interessanten Gegenstand. Er nimmt den Gegenstand. Plötzlich hört er ein gefährliches Brummen vor dem Höhleneingang. Kurz darauf betritt ein riesiger Bär mit lautem Gebrüll die Höhle und nähert sich dem Spieler. Dies ist der Moment, wo der Spieler seine grauen Zellen bemühen muß. Wie kommt er lebend aus der Höhle heraus? Einfach am Bären vorbeilaufen (wegrennen) ist sicher zu leicht. Den Bären mit einem Schwert erlegen, soll auch unmöglich sein, da der Bär zu stark ist. Angenommen, der Spieler hat ein Schwert und greift den Bären an, so soll er ihn verletzen können. Dadurch soll dieser aber höchstens noch wilder werden und ... game over.

Nun gut, ich habe viele Ideen wie der Spieler die Situation verlieren kann, aber er muß schließlich auch eine Chance haben zu entkommen. Wir haben ja bereits klargestellt, daß man den Spieler nie vor vollendete Tatsachen stellen sollte, wenn man ein gutes Adventure schreiben will. Die Idee: Bären haben Angst vor Feuer. Der Spieler muß also schnell etwas Holz vom Boden nehmen (das selbstverständlich dort liegt) und es mit seinem Feuerzeug (das er hoffentlich in seiner Ausrüstung hat) entzünden. Der Bär bekommt dann Angst vor der so entstandenen Fackel und flieht mit panischer Angst — der Spieler hat es geschafft!

Anhand dieses Beispiels können Sie erkennen, wie man vorgehen muß, um Action-Szenen zu gestalten.

Die Arbeitsschritte sind:

1. Ausdenken einer Gefahrensituation. Nicht davor zurückschrecken, wenn diese Gefahrensituation zunächst ausweglos erscheint.
2. Darüber nachdenken, durch welche Ideen der Spieler verliert.
3. Über eine Lösung nachdenken.

Vom 1. Schritt direkt zum 3. überzugehen, ist sehr schwer. Deshalb fügt man einen zweiten Schritt ein, bei dem man sich erst einmal Gedanken macht, wodurch der Spieler verlieren kann, also welche falsche Reaktionen des Spielers in Betracht gezogen werden können.

Dabei kommt man dann meistens auch auf die Lösung der jeweiligen Gefahrensituation.

Bedenken Sie dabei: Je länger Sie an dem Problem arbeiten und je schwieriger es für Sie ist eine vernünftige Lösung zu finden, desto interessanter und anspruchsvoller wird das Spiel für den Abenteurer später sein. Betonen möchte ich hierbei noch einmal den Ausdruck »vernünftige Lösung«. Damit ist gemeint, daß Sie keine utopischen Lösungen aushecken sollen, sondern Lösungen, die auch wirklich oder wenigstens zum Teil in der Realität vorkommen.

Eine utopische Lösung wäre, wenn man den Bären einfach mit Anbrüllen zur Flucht bewegen könnte.

Lassen Sie sich ruhig mehrere Tage Zeit, die Lösungen zu den einzelnen Aufgaben zu finden.

Ein anderes Beispiel für eine solche Action-Szene stand bereits in der Einleitung zu diesem Kurs. Es stammt aus dem Adventure »Gorden Saga«.

Oft bringt der Einbau von Action-Szenen auch eine Änderung der Spielkarte mit sich. Sie sehen, daß es beim Schreiben von Adventures nahezu unmöglich ist, einzelne Arbeitsschritte nacheinander durchzuführen — man muß vielmehr immer an mehreren gleichzeitig arbeiten, da sie alle eng miteinander verbunden sind.

Ich möchte jetzt nicht auf weitere Action-Szenen eingehen, die man in das Spiel einbauen kann. Es sollen statt dessen noch einige Erläuterungen zur Karte gemacht werden.

Vom Raum 1 (Strand) kann man in mehrere Richtungen gehen, die alle zum Urwald führen. Betrachten Sie nun einmal die Räume 4 und 7 im Wald und ihre Verbindungen zueinander. Sie werden feststellen, daß man von Raum 7 nach Südosten gehen muß, um zu Raum 4 zu gelangen etc. Es ist besonders wichtig, daß Sie auch die Himmelsrichtungen NO, SO, SW, NW in der Karte verwenden. Dies ist zwar später ein bißchen schwerer zu programmieren, als nur vier Richtungen, aber die Karten werden durch acht oder zehn Richtungen (mit rauf und runter) viel komplexer. Der Spieler wird somit gezwungen, Aufzeichnungen während des Spiels zu machen, und kann die Karte nicht gleich auswendig lernen, wenn sie nur wenig Räume hat.

Zwischen den Räumen 13 und 14 befindet sich eine gestrichelte Linie, mit der Bezeichnung Felsspalt. Diese gestrichelte Linie bedeutet, wie bereits besprochen, daß der Spieler nicht direkt von Raum 13 zu Raum 14 gelangen kann, sondern nur durch einen Felsspalt, den er finden muß. Dieser Spalt könnte durch einen Felsblock versperrt sein, den man erst wegrollen muß. Der Felsspalt kann als Tür zwischen Raum 13 und 14 angesehen werden. Zwischen den Räumen 16 und 17 befindet sich ebenfalls eine gestrichelte Linie mit der Bezeichnung Geheimtür; man gelangt also nur durch eine Geheimtür von Raum 16 nach Raum 17.

Zwischen Raum 14 und 15 befindet sich ein Doppelpfeil mit der Bezeichnung Aufzug. Man muß in Raum 14 (der Grotte) also in einen Aufzug steigen, diesen aktivieren, und gelangt dann zu Raum 15. In diesem Fall hätte man auch eine gestrichelte Linie statt eines Doppelpfeils verwenden können. Zwischen den Räumen 24 und 25 (dem Fluß) befindet sich ein Pfeil, der nur in eine Richtung weist. Dies ist logisch, da der Spieler nur in die Richtung kann, in der der Fluß treibt. Der Fluß ist so schnell, daß er nicht in der Lage ist, gegen die Strömung zu rudern.

Sicher haben Sie bereits den Sinn erkannt, die die Durchnumerierung der Räume in sich birgt. Sie können sich so leichter Notizen zu den einzelnen Räumen machen. Es empfiehlt sich zu jedem Raum eine kleine Notiz zu machen, was dort ist, und was dort passieren kann.

Des weiteren sollten Sie sich eine Tabelle anfertigen, die aussagt, wo sich die einzelnen Gegenstände befinden; oder zeichnen Sie die Gegenstände einfach in die Karte ein. Wichtig ist nur, daß Sie später beim Programmieren genügend Notizen angefertigt haben, auf die Sie dann zurückgreifen können.

Zeichnen Sie die endgültige Spielkarte sauber und deutlich. Damit wären wir auch schon am Ende des ersten Kursteils angelangt. Sie haben jetzt vieles über Adventures erfahren, wissen wodurch sich gute Adventures von schlechten unterscheiden, und wie man eine Adventurekarte entwirft und ein Spiel mit Action versieht. Es gibt natürlich noch einige weitere Tricks zum Schreiben und Ausdenken von Adventures. Es werden Ihnen sicher noch viele Ideen kommen, was man so alles machen kann. Sie werden schon bald Ihren eigenen Stil entwickeln.

4. Kapitel: Die Programmiertechnik

Wir haben bislang gelernt, wie man sich ein Abenteuerspiel ausdenkt und es ausarbeitet.

Jetzt wollen wir lernen, wie man solch ein ausgearbeitetes Spiel in ein Basic-Programm für den C 64 umsetzt.

Vielleicht haben Sie schon etwas von sogenannten Adventure-Generatoren gehört. Dies sind Programme, die es jedem, auch ohne jegliche Basic-Kenntnisse, ermöglichen, Adventure-Spiele zu erzeugen. Mit solchen Generatoren lassen sich oft relativ umfangreiche (mit vielen Räumen) Abenteuerspiele erstellen. Umfangreich heißt jedoch lange noch nicht anspruchsvoll.

Vom Kauf eines solchen Generators möchte ich Ihnen unbedingt abraten. Auch die allerbesten Generatoren schränken Ihre Kreativität in erheblichem Maße ein. Man erkennt auch meist von welchem Generator das Spiel erzeugt wurde, Sie sollen jedoch lernen, wie Sie Ihrem Adventure eine persönliche Note verleihen. Man soll erkennen, daß das Spiel von Ihnen programmiert wurde, und nicht von irgendeinem Programmgenerator.

Das Ziel dieses Kurses ist, Ihnen das Wissen zu vermitteln, das nötig ist, um anspruchsvolle Adventures zu schreiben, die den kommerziellen Spielen in keiner Weise an Qualität nachstehen. Die zahlreichen Beispiel-Programme (Listings) sind ausführlich dokumentiert. Ausreichende Basic-Kenntnisse und Erfahrung sind jedoch unbedingt erforderlich, da dieser Kurs kein Basic-Kurs ist. Selbstverständlich werde ich auf Kniffe und Programmiertricks ausführlich eingehen.

Arbeitsmittel und Methoden

Zum Programmieren gehen wir von folgender Hardware aus: Computer C 64 und Datasette (oder Diskettenlaufwerk). Ein Drucker und ein Diskettenlaufwerk wären zwar äußerst sinnvolle Ergänzungen, aber ich möchte vom Computerbesitzer ausgehen, der diese Erweiterungen nicht besitzt.

Für die Programme werden wir, früher oder später, alle 38 KByte RAM benötigen. Deshalb müssen wir auf Erweiterungen wie Simons Basic verzichten. Nicht zuletzt auch, weil solche Erweiterungen den Anwenderkreis stark einschränken.

Simons Basic wäre vielleicht noch vertretbar (abgesehen von den 8 KByte Basic-Speicher, die es belegt), da es schon weit verbreitet ist, aber andere Erweiterungen würden es nur demjenigen ermöglichen das Adventure zu spielen, der diese Erweiterung besitzt.

Äußerst sinnvoll hingegen sind Erweiterungen, die den Programmierkomfort unterstützen, der beim C 64, abgesehen vom Full-Screen-Editor, sehr gering ist.
Ich meine hiermit Zusatzbefehle wie:

DUMP zum Variablen auflisten
HELP zur Fehlerbeseitigung
TRACE zum Überprüfen des Programmablaufs sowie
KEY zur Funktionstastenbelegung etc...

Gemeint sind also Befehle, die nicht im Programm verwendet werden, sondern nur direkt, wie der Befehl LIST arbeiten.

Eine empfehlenswerte Erweiterung ist das mittlerweile weitverbreitete Turbotape — eine Schnellade-Routine, die die Ladegeschwindigkeit der Datasette um den Faktor 10 erhöht. Turbotape benötigt keinen Basic-Speicher. Durch die hohe Baudrate entfallen langwierige LOAD- und SAVE-Vorgänge. Eine andere sinnvolle Erweiterung ist das KFC-Supermodul. Es enthält Turbotape, sowie die oben erwähnten Befehle und noch einige mehr. Da es jedoch 8 KByte Speicher benötigt, kann es nur so lange eingesetzt werden, wie das Adventure die Länge von 30 KByte unterschreitet.

Reichen die 30 KByte schließlich nicht mehr aus, so muß das Modul abgeschaltet und somit auf die Hilfsbefehle verzichtet werden. Dadurch erklärt sich auch die Tatsache, daß die schwierigsten und kompliziertesten Programmteile zu Beginn geschrieben werden müssen. Um das Spiel später mit Grafik zu versehen, werden keine Trace- und Dump-Befehle mehr benötigt.

Eine weitere Schwierigkeit besteht darin, den Überblick über ein 38-KByte-Programm zu behalten, wenn man keinen Drucker besitzt. Ich kann Ihnen jedoch aus eigener Erfahrung versichern, daß ein Drucker bei einer guten Programmiertechnik nicht unbedingt erforderlich ist.

Sicherlich kann man den Überblick über ein langes Basic-Programm gewährleisten, wenn man es gut strukturiert. Da strukturierte Programme jedoch viel länger sind als nichtstrukturierte, müssen wir eine Zwischenlösung von strukturierter und »chaotischer« Programmierung suchen.

Ich habe mich für folgende Lösung entschieden:

— Das Programm wird in kleine Teilprogramme (Module) zerlegt.
— Diese Module werden ausführlich in ihrer Funktionsweise erklärt. Jedes Modul ist in sich abgeschlossen. Es kann individuell für jedes Programm verwendet werden. Das Modul wird im Normalfall mit GOSUB aufgerufen.
— Die einzelnen Module werden innerhalb des Programms vom Hauptmodul aufgerufen. Das Hauptmodul ist also der zentrale Punkt des Listings beziehungsweise des Programms.
— Die Module stehen immer in den selben Zeilennummern; zum Beispiel Befehlseingabemodul immer von Zeile 50000 bis maximal Zeile 50999 etc.

Die Module werden in den folgenden Abschnitten ausführlich behandelt. Ein Abschnitt baut auf dem anderen (vorhergehenden) auf. Deshalb ist es erforderlich, daß Sie die Module nicht nur einfach abtippen, sondern auch verstehen lernen. Sind Sie mit der Funktionsweise eines Moduls vertraut, so versuchen Sie damit zu experimentieren.

Im nächsten Kapitel wird ein wichtiges Modul behandelt — die Befehlseingabe. Es werden hierbei mehrere Möglichkeiten vorgestellt. Ziel eines jeden Kapitels soll sein, Ihnen das Wissen, das nötig ist, das jeweilige Modul zu programmieren, zu vermitteln. Damit Sie überprüfen können, ob Sie genug für das jeweils folgende Modul gelernt haben, werde ich Ihnen am Ende eines jeden Abschnittes eine Aufgabe stellen. Hierzu jedoch später.

Speichern Sie alle Beispielprogramme nach dem Eintippen stets auf Kassette/Diskette ab, damit Sie später darauf zurückgreifen können.

Bevor Sie nun beginnen, sollten Sie einen Blick in den Anhang dieses Kurses werfen, da dort mehrere Hilfsprogramme zu finden sind.

Während des Kurses werden wir öfter auf Programme aus diesem Anhang zu sprechen kommen. Also, Computer einschalten und los geht's...

String-Operationen

Gerade beim Programmieren von Abenteuerspielen ist es unbedingt notwendig, mit der Technik der Stringverarbeitung vertraut zu sein. Obwohl die eigentlichen Stringfunktionen im Basic leicht verständlich sind, haben Anfänger oft erhebliche Schwierigkeiten, sie sinnvoll einzusetzen. Dieser Fall tritt insbesondere dann auf, wenn mehrere Funktionen ineinander verschachtelt werden müssen.

Hier nun eine kleine Einführung in die Stringverarbeitung. Das Basic des Commodore 64 bietet folgende Stringfunktionen an:

ASC(X$): Geben Sie Ihrem Computer im Direktmodus einmal folgenden Befehl ein:
X$="A" : ? ASC(X$) <RETURN>.
Als Ergebnis erhalten wir 65. Im C 64-Handbuch auf Seite 135 bis 137 finden Sie eine Tabelle der ASCII- und CHR$-Codes. Diese Tabelle ordnet jedem Zeichen eine Zahl zu. Bei der Zahl 65 finden wir den Buchstaben A.
Die ASC-Funktion liefert uns also den Zahlenwert zu jedem Zeichen und zwar jeweils den Wert des 1. Zeichens des definierten Strings.

CHR$(X): CHR$ ist die Umkehrung der ASC-Funktion. CHR$(X) ergibt das Zeichen, das dem Zahlenwert X zugeordnet ist. Geben wir ? CHR$ (65) ein, so erhalten wir als Antwort das Zeichen A. Mit CHR$(X) können auch Sonderfunktionen wie Farben und Groß-/Kleinumschaltung ausgeführt werden. Die Tabelle im Handbuch gibt hier ausführlich Antwort.

LEN(X$): Diese Funktion ergibt die Anzahl der Zeichen, die ein String enthält.
X$ = "ABENTEUER"
? LEN(X$)
Das Ergebnis wäre hierbei 9, da das Wort »Abenteuer« 9 Buchstaben enthält. Achtung: Auch Leerzeichen im String werden mitgezählt.

LEFT$(X$,X): Diese Funktion ergibt die linken X-Zeichen von X$.
Beispiel: X$ = →ABENTEUER←
? LEFTS(X$,5) ergibt »ABENT«.

RIGHT$ (X$,X): Diese Funktion ergibt die rechten X-Zeichen von XS.

MID$(X$,S,X): Diese Funktion ergibt X-Zeichen von X$, beginnend mit dem Zeichen an Position S.
X$ = "ROTGELBGRUEN"
? MID$(X$,4,4) ergibt GELB.
Der dritte Patameter kann fortgelassen werden. MID$(X$,S) ergibt dann alle Zeichen ab Position S, in unserem Beispiel

also »GELBGRUEN«.

STR$(X): STR$(X) gibt uns die Möglichkeit, eine Zahl in einen String zu verwandeln.

```
X = 100+20+3
X$ = STR$(X)
? X$ ergibt 123
```

VAL(X$): Diese Funktion ist im wesentlichen die Umkehrung der STR$-Funktion.

```
VAL("100") ergibt 100
VAL("123ABC") ergibt 123
VAL("ABC123") ergibt 0, da das erste
Zeichen keine Zahl ist.
```

Dies waren nun auch schon alle Funktionen, die Ihnen nun verständlich sein sollten.
Nun aber einige Anwendungsbeipiele.

1. Prüfen, ob ein String in einem anderen enthalten ist. Eine solche Routine ist besonders dann nützlich, wenn man dem Spieler ermöglichen will, Befehle abzukürzen (zum Beispiel UNT statt UNTERSUCHE).
   Beispiel: Es soll überprüft werden, ob B$ in A$ enthalten ist.

```
10 A$ = "UNTERSUCHE"
20 B$ = "UNT"
30 IF B$=LEFT$(A$,LEN(B$)) THEN
   PRINT"B$ KOMMT IN A$ VOR!"
40 END
```

   Eine Programmdokumentation erübrigt sich hier. Falls Sie das Programm nicht verstehen, so studieren Sie nochmals ausführlich die einzelnen String-Befehle.
2. Ausdruck einer Zahlenkette.
   Geben Sie einmal folgendes Programm ein:

```
10 FOR I=1 TO 5
30 PRINT I;",";
40 NEXTI
50 END
```

Wenn Sie das Programm nun mit RUN starten, wird auf dem Bildschirm folgendes ausgedruckt: 1 , 2 , 3 , 4 , 5 ,

Uns stören hierbei die Zwischenräume (oder sollten uns zumindest stören). Deshalb soll das Programm so verändert werden, daß folgendes Ergebnis erreicht wird: 1,2,3,4,5,

Um dies zu erreichen, machen wir in dem Programm einen kleinen Umweg über den STR$(X)-Befehl.

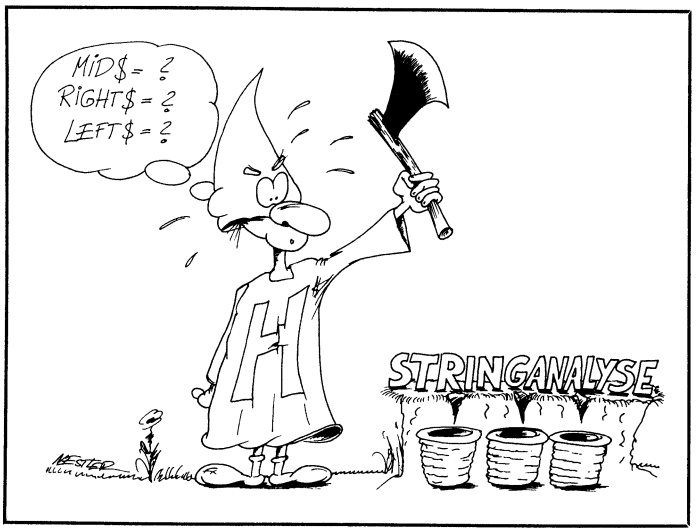
Wir ergänzen das Programm um Zeile

```
20 X$=MID$(STR$(I),2)
```

und ändern Zeile 30 um

```
30 PRINT X$;",";
```

Ich hoffe, Sie sind mir nicht böse, daß ich schon wieder keine ausführliche Dokumentation zu Zeile 20 liefere, aber Ziel des Kurses soll schließlich sein, Ihnen Programmieren beizubringen. Ich werde mich deshalb hauptsächlich auf die Dokumentation von kniffligen Problemen beschränken und Sie die einfacheren Aufgaben selbst ausarbeiten lassen. Aber nur keine Angst. Es wird bestimmt nicht zu schwer für Sie werden. Wenn Sie sich im Gebrauch von String-Befehlen nun einigermaßen sicher fühlen, dann können Sie sogleich mit dem nächsten Abschnitt fortfahren.



Die Befehlseingabe

In diesem Abschnitt werden wir einen Programmteil erarbeiten, der für jedes Text-Adventure unentbehrlich ist: die Befehlseingabe.

Darunter versteht man den Programmteil, bei dem der Computer einen Befehl vom Spieler erwartet. Bei den meisten Adventures erscheint dann auf dem Bildschirm die Frage »WAS NUN?«. Bereits daran, wie gut die Befehlseingabe ist, kann man über die Qualität des Spiels entscheiden. Ziel dieses Abschnitts soll sein, Ihnen soviel Wissen über Texteingaberoutinen zu vermitteln, daß Sie in der Lage sind, eine Befehlseingaberoutine zu schreiben, die professionellen Spielen in nichts nachsteht. Ziel ist es also, ein Befehlseingabe-Modul zu schreiben, das später universell eingesetzt werden kann.

Im Kapitel Arbeitsmittel und Methoden habe ich Ihnen bereits erklärt, daß jedes Modul immer in den selben Zeilennummern stehen soll, damit man auch bei längeren Programmen nicht den Überblick verliert.

Das Befehlseingabe-Modul wollen wir künftig immer in den Zeilen 50000 bis maximal 50999 unterbringen. Es wird folglich mit GOSUB 50000 aufgerufen.

a) Befehleingabe Modul 1
   Geben Sie nun einmal Listing 1 in den Computer ein:

```
10 GOSUB 50000                              <140>
20 PRINT BE$                                <088>
30 GOTO 10                                  <008>
50000 REM *** BEFEHLSEINGABE I ***          <203>
50010 POKE 198,0:BE$=""                     <141>
50020 INPUT"WAS NUN ";BE$                   <178>
50030 RETURN                                <191>
```

Listing 1.

Wenn Sie das Programm laufen lassen, so fragt der Computer »WAS NUN?« und wartet auf eine Befehlseingabe, etwa auf einen Befehl wie »NIMM SCHWERT« der mit RETURN abgeschlossen wird.

Dokumentation zum Listing:

10 Aufruf des Befehlseingabe-Moduls
20 Ausdruck des Befehls, den wir als BE$ bezeichnen wollen
30 Zurück zum Anfang
50000 POKE 198,0 setzt den Tastaturpuffer auf Null. Dadurch wird verhindert, daß die Befehlseingabe einfach übersprungen wird, weil noch Zeichen im Tastaturpuffer stehen. Außerdem wird der String, in dem der Befehl steht, gelöscht (BE$="")
50020 Mittels eines einfachen INPUT-Befehls wird der Befehlsstring BE$ eingelesen
50030 Ende des Moduls

Hiermit hätten wir auch schon unser erstes Modul.
Leider hat diese einfache Befehlseingabe einige Haken:

— Man kann mit dem Cursor über den ganzen Bildschirm fahren und beliebig »herumwüten«.
— Bei Eingabe von Komma erfolgt ein »?EXTRA IGNORED ERROR«.
— Professionelle Routinen fahren mit dem Programmablauf fort, wenn über einen längeren Zeitraum keine Eingabe erfolgt.

Dies ist bei unserer ersten Routine noch nicht der Fall.

Wir müssen eingestehen, daß der INPUT-Befehl anscheinend doch nicht das Wahre für eine professionelle Eingaberoutine ist.

Das Problem ist noch einmal neu anzupacken.

b) Befehlseingabe Modul 2
Geben Sie bitte das folgende Programm (Listing 2) ein und lassen Sie es nach SAVE laufen.

```
10 GOSUB 50000                                          <140>
20 PRINT:PRINT BE$:PRINT                                <254>
30 GOTO 10                                              <008>
50000 REM *** BEFEHLSEINGABE II ***                     <020>
50010 POKE 198,0:BE$="":PRINT"WAS NUN ? ";              <250>
50020 POKE 204,0                                        <176>
50030 GET X$:IF X$=""THEN 50030                         <036>
50040 IF PEEK(203)=1 OR LEN(BE$)>68 THEN P
      RINT" ":POKE 204,1:RETURN                         <239>
50050 I=ASC(X$):IF I<65 OR I>90 THEN IF I<
      >32 AND I<>20 AND I<>34 THEN 50030                <118>
50060 IF I=20 AND BE$=""THEN 50030                      <038>
50070 IF I=20 THEN POKE 204,1:PRINT"{LEFT,
      2SPACE,2LEFT}";:BE$=LEFT$(BE$,LEN(BE
      $)-1):GOTO 50020                                  <119>
50080 PRINT X$;:BE$=BE$+X$:GOTO 50030                   <214>
```

Listing 2.

Auf den ersten Blick scheint sich nicht viel geändert zu haben. Bei intensiverem Ausprobieren werden Sie jedoch einige Unterschiede zum vorhergehenden Programm bemerken:

— Die Cursortasten sind ausgeschaltet
— Es werden nur noch folgende Zeichen akzeptiert:
A bis Z und " und die Tasten SPACE, DEL, RETURN
— Mit DEL kann nur die Eingabe gelöscht werden — nicht jedoch die »WAS NUN?«-Frage.

Damit haben wir eine schon fast »idiotensichere« Routine, wie man sie in professionellen Spielen findet.

Dokumentation zum Listing:

50010 Tastaturpuffer und BE$ löschen. »WAS NUN?« fragen.

50020 POKE 204,0 zum Einschalten des Cursors.

50030 Auf Eingabe eines einzelnen Zeichens X$ warten.

50040 Jeder Taste ist ein bestimmter Wert zugeordnet (zum Beispiel RETURN = 1). Mittels PEEK (203) erfährt man, welche Taste gedrückt worden ist. Wird also die RETURN-Taste gedrückt oder die Länge des Befehls BE$ überschreitet die Länge von 68 Zeichen, so wird die Routine beendet: der Cursor wird gelöscht und gestoppt.

50050 Hier wird die Eingabemöglichkeit auf die oben erwähnten Zeichen und Tasten eingeschränkt (vgl. auch ASCII-Werte mit Commodore-Handbuch).

50060 Hier wird verhindert, daß mittels DEL mehr Zeichen gelöscht werden als vorher eingegeben worden sind.

50070 Prüfung, ob DEL-Taste gedrückt worden ist. Wenn Ja, dann Cursor ausschalten. Letztes Zeichen löschen. Vom Befehlsstring BE$ das letzte Zeichen wieder löschen.

50080 Das neue Zeichen X$ ausdrucken. Den Befehlsstring BE$ um das neue Zeichen X$ erweitern. Neues Zeichen von Tastatur abwarten.

Nun möchte ich Ihnen noch eine Routine (Listing 3) vorstellen. Erfolgt 30 Sekunden lang keine Eingabe, so endet die Routine und fährt im Programm fort — die Bedenkzeit des Spielers wird also eingeschränkt. Bei dieser Routine werden Sie feststellen, daß der Bildschirm in zwei Abschnitte unterteilt ist, wie es bei Adventures mit Grafik und Text vorkommt. Die oberen $^3/_4$ des Bildschirms sind als Grafik-Fenster, das untere Viertel als Texteingabe-Fenster definiert.

```
10 PRINT"{CLR,18DOWN}cccccccccccccccccccc
   cccccccccccccccccc"                                  <036>
20 GOSUB 50000                                          <150>
30 PRINT"{HOME}";BE$                                    <244>
40 GOTO 20                                              <019>
50000 TI$="000000":BE$="":POKE 198,0:POKE
      211,0:POKE 214,18:SYS 58732:PRINT"{D
      OWN,WHITE}> ";                                    <125>
50010 GET X$:IF PEEK(203)=1 THEN 50110                  <040>
50020 IF TI$>"000030"THEN 50110                         <041>
50030 IF X$=""THEN 50010                                <203>
50040 IF LEN(BE$)=0 AND ASC(X$)=20 THEN 50
      010                                               <090>
50050 I=ASC(X$):IF I<65 OR I>90 THEN IF I<
      >20 AND I<>32 AND I<>34 THEN 50010                <116>
50060 BE$=BE$+X$                                        <125>
50070 PRINT CHR$(20);X$;" ";                            <129>
50080 IF I=20 THEN BE$=LEFT$(BE$,LEN(BE$)-
      2):GOTO 50010                                     <148>
50090 TI$="000000":IF LEN(BE$)>76 THEN FOR
       I=1 TO 80:PRINT CHR$(20);:NEXT:GOTO
       50000                                            <066>
50100 GOTO 50010                                        <246>
50110 PRINT:PRINT"{UP,40SPACE}";                        <253>
50120 PRINT"{40SPACE}":RETURN                           <048>
```

Listing 3.

Auf eine ausführliche Dokumentation verzichte ich hier, da nur wenige neue Elemente im Listing auftauchen.

Dies sind: a) TI$ wird zum Einschränken der Abfragedauer eingesetzt. Wenn die Zeit von 30 Sekunden überschritten wird, so fährt das Programm automatisch fort (Zeile 50020).

b) Werfen Sie bitte einen Blick auf Zeile 50000.
Dort heißt es: POKE 214,18: POKE 211,0:SYS 58732
Durch diese POKE-Stellen kann man bestimmen, wo der Cursor oder das nächste PRINT erscheint.
POKE 214, Zeile (0-24)
POKE 211, Spalte (0-39)
dannach SYS 58732, um eine Betriebssystem-Routine aufzurufen, die den Cursor an der neuen Stelle positioniert.

Damit wären wir auch schon am Ende des ersten Abschnitts angelangt. Sie sollten nun gelernt haben, wie man mit den String-Funktionen umgeht und Befehlseingabe-Routinen schreibt. Bevor Sie mit dem nächsten Abschnitt fortfahren, sollten Sie ein wenig mit den vorgestellten Routinen experimentieren und sie nach eigenen Wünschen und Ideen umgestalten.

Schreiben Sie sich sodann ein Befehlseingabe-Modul (welches wie vereinbart in den Zeilen 50000 bis maximal 50999 liegt), das Sie dann in den folgenden Abschnitten verwenden können.

Das Modul muß durch GOSUB 50000 aufgerufen werden können und dann einen String BE$ liefern, der den Befehlssatz enthält.

Die Befehlszerlegung

Nachdem wir nun unser Befehlseingabe-Modul fertiggestellt haben, wollen wir uns einem neuen Modul »dem Befehlszerlegemodul« zuwenden und somit einen ersten Schritt zur Befehls-Analyse beginnen.

Was ist eigentlich der Unterschied zwischen Befehlszerlegung und Befehls-Analyse, werden Sie jetzt vielleicht fragen?

Ganz einfach. Die Befehlszerlegung (also unser neues Modul) besteht darin, einen Befehlssatz BE$ in kleine einzelne Befehle zu zerlegen. Dies ist nötig, da der Computer immer nur einen Befehl nach dem anderen und nicht einen

Mehrfachbefehl auf einmal bearbeiten kann. Die Befehls-Analyse besteht darin, die Einzelbefehle auszuführen. Das Programm reagiert auf den Befehl.

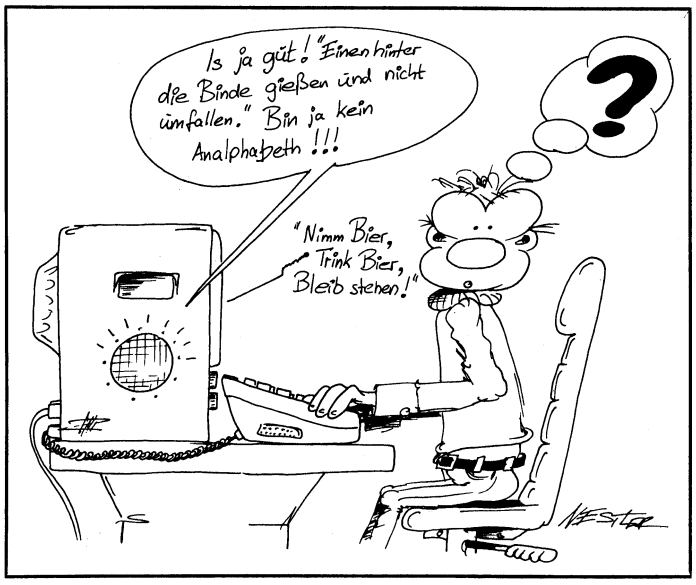
Vielleicht haben Sie schon einmal das englische Adventure »The Hobbit« gespielt oder zumindest davon gehört.

»The Hobbit« ist ein Adventure mit äußerst gelungener Befehls-Analyse. Der Wortschatz kennt kaum Grenzen.

Hier ein Beispiel, wie es beim Hobbit sein könnte:

SIE SEHEN: SCHWERT
SEIL
DIE FELSENTUER
MOEGLICHE RICHTUNGEN: SUEDEN
WAS NUN? Nimm das Schwert, das Seil und geh nach Sueden.
SIE NEHMEN DAS SCHWERT.
SIE NEHMEN DAS SEIL.
SIE GEHEN NACH SUEDEN.

Diesen Textausschnitt (den ich ins Deutsche übersetzt habe) wollen wir nun einmal näher betrachten.



Der Computer teilt dem Spieler mit, was er sieht und welche Richtungen möglich sind und erwartet sodann einen Befehl »WAS NUN?«.

Der Spieler gibt seinen Befehlssatz (bei uns BE$) ein.

Der Computer meldet sich jedoch nicht mit »SIE NEHMEN DAS SCHWERT, DAS SEIL UND GEHEN NACH SUEDEN.«, sondern gibt drei Meldungen aus — für jeden Teilbefehl eine.

Daran können wir erkennen, daß auch dieses professionelle Adventure Befehle nacheinander ausführt.

Unser Ziel soll nun sein, eine Befehlszerlegung und Befehls-Analyse zu programmieren, die der oben erwähnten professionellen in nichts nachsteht.

Bevor wir uns wieder zum Programmieren begeben, müssen wir einiges an Theorie durchkämmen.

Schauen wir uns doch einmal folgende Befehlssätze, die jeweils einen Befehl enthalten, an:

1. NIMM DAS SCHWERT.
2. GEH NACH NORDEN.
3. UNTERSUCHE DAS SCHWERT.
4. WIRF DAS SEIL.
5. OEFFNE DIE TRUHE.
6. LEGE DAS SCHWERT IN DIE TRUHE.
7. TOETE DAS MONSTER MIT DEM SCHWERT.
8. WARTE.

Dies sind typische Befehlssätze BE$, wie sie vom Spieler eingegeben werden.

Nun wollen wir die Befehlssätze auf das Nötigste komprimieren, das heißt, daß alle Worte, die für das Verständnis des Befehls unnötig sind, aussortiert werden.

Wir erhalten dann für die Tabelle:

1. NIMM SCHWERT
2. N
3. UNTERSUCHE SCHWERT
4. WIRF SEIL
5. OEFFNE TRUHE
6. LEGE SCHWERT TRUHE
7. TOETE MONSTER SCHWERT
8. WARTE

Diese Sätze sind zwar komprimiert, aber dennoch ist ihr Sinn eindeutig. Die Sätze 6 und 7 sind zwar extrem schlechtes Deutsch und für uns vielleicht zweideutig, aber für unser späteres Analyse-Programm gut verständlich.

Auffallend ist, daß unsere komprimierten Befehlssätze aus maximal drei Worten bestehen. Denken Sie sich doch einmal einige komplizierte Befehlssätze aus und sortieren Sie dann alle überflüssigen Worte aus. Der Befehl, der dann übrigbleibt, besteht garantiert aus nicht mehr als drei Worten. Diese Garantie gilt natürlich für Befehlssätze mit nur einem Befehl, es darf also kein »und« oder »,« im Satz vorkommen.

Befehlssätze mit mehreren Befehlen werden immer in Einbefehlssätze aufgeteilt.

Aus »NIMM DAS SCHWERT UND GEH NACH NORDEN« würde also »NIMM DAS SCHWERT« und »GEH NACH NORDEN«. Die beiden Sätze, die sich aus dieser Trennung ergeben, können dann wieder nach obigem Schema zerlegt werden.

Als nächstes wollen wir unsere komprimierten Befehlssätze auf folgende Elemente untersuchen:

VERBEN: Wörter wie NIMM, OEFFNE und Himmelsrichtungen (N, S). Der Befehl N ist eine Abkürzung für GEH NACH NORDEN und kann somit als Verb interpretiert werden.

OBJEKTE: Wörter wie TUER, TRUHE, BAUM, HAUS; also im wesentlichen Dinge, die man nicht transportieren kann. Außerdem Personen im weitesten Sinne (zum Beispiel MONSTER).

GEGENSTÄNDE: Dies sind Objekte, die man transportieren kann. Zum Beispiel SCHWERT, SCHLUESSEL, TRUHE, SEIL. Dinge wie die Truhe, können sowohl Gegenstände als auch Objekte sein; dazu jedoch später.

Ich werde folgende Abkürzungen verwenden:

ve für VERB
ob für OBJEKT
ge für GEGENSTAND

Wenn wir die zweite Tabelle auf die obigen Elemente überprüfen, so erhalten wir folgende neue Tabelle:

1. ve + ge
2. ve
3. ve + ge
4. ve + ge
5. ve + ob oder auch ve + ge (falls Truhe transportabel)
6. ve + ge + ob
7. ve + ob + ge
8. ve

Aus dieser Tabelle können wir folgendens erkennen:

— jeder Befehlssatz enthält ein und nur ein Verb
— das Verb steht immer an erster Stelle und ist somit das erste Wort des Befehlssatzes
— nach dem Verb können Objekte und Gegenstände aufgeführt sein.

Im allgemeinen sind folgende ve-ob-ge-Kombinationen möglich:

a) ve »WARTE«
b) ve + ge »NIMM SCHWERT«

c) ve + ob »OEFFNE TUER«
d) ve + ge + ob »LEGE SCHWERT TRUHE«
e) ve + ob + ge »TOETE MONSTER SCHWERT«
f) ve + ge + ge »VERKNOTE SEIL BRETT«

Wir wissen nun also, daß jeder Satz immer mit einem Verb beginnt. Das folgende zweite Wort kann dann ein Gegenstand oder ein Objekt sein. Das dritte Wort genauso. Schön und gut. Wir haben nun einiges an Grammatik-Theorie durchgekaut, und wissen wie man Sätze auf das Nötigste komprimieren kann. Wie programmiert man denn nun aber solch eine Befehlszerlegung?

Bevor wir mit dem Programmieren beginnen, wollen wir deshalb noch einige Gedankenexperimente durchführen:

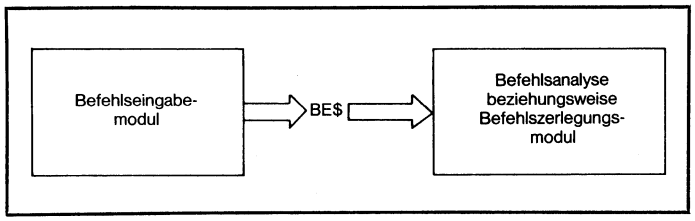
Greifen wir doch noch einmal auf den vorangegangenen Abschnitt zurück, in dem wir das Befehlseingabemodul behandelt haben.

Es ist Ihnen sicherlich bereits klar, daß dieses Modul immer vor dem Befehlszerlegungsmodul (das wir nun besprechen) aufgerufen werden muß, denn bevor man einen Befehlssatz zerlegen kann, muß man ja erst einmal einen haben. Wenn man das Befehlszerlegungsmodul aufruft, so läßt es den Spieler einen Befehl eingeben, der als BE$ gespeichert wird.

Zum Beispiel: BE$="NIMM DAS SCHWERT"

Dieser Befehlsstring BE$ wird dann in das Befehlszerlegemodul geschickt, wo es weiterbehandelt wird.

Das Schema ist folgendermaßen:



Das Befehlseingabemodul liefert also den Befehlsstring BE$ an das Befehlszerlegungsmodul. Was macht das Zerlegungsmodul nun mit dem String BE$?

Während das Befehlseingabemodul nur eine einzige Aufgabe hat, nämlich als eine komfortable INPUT-Routine zu fungieren, muß das Befehlszerlegemodul schon wesentlich mehr leisten.

In dem Zerlegungsmodul sind mehrere Module vereint, die folgende Aufgaben haben:

1. String-Zerlegung
2. String-Sortierung
3. String-Codierung

Diese Aufgabenbereiche wollen wir nun einmal im Einzelnen betrachten.

Wir wollen bei den folgenden Betrachtungen von einem Befehlsstring BE$ ausgehen, der so aussieht:

BE$="NIMM DAS SCHWERT".

Dieser Befehlsstring wird nun also vom Befehlseingabenmodul zum Befehlszerlegungsmodul geschickt.

Im Befehlszerlegungsmodul geschieht dann folgendens mit dem BE$:

1. Der erste Schritt besteht darin, den Befehlsstring BE$ in einzelne Worte aufzuschlüsseln.
   Jedem Wort des Befehlsstrings BE$ wird ein neuer String zugeordnet. Diese neuen Strings werden als BE$(1) bis BE$(X) bezeichnet, wobei X die Zahl der im Satz vorkommenden Worte darstellt.
   Für unser Beispiel würde dann gelten:
   BE$ = "NIMM DAS SCHWERT"
   BE$(1)="NIMM"
   BE$(2)="DAS"
   BE$(3)="SCHWERT"

2. Der zweite Schritt besteht anschließend darin, den Satz zu komprimieren, also alle überflüssigen Worte aussortieren. Auszusortierende Worte sind Worte, die zum Verständnis des Satzsinns überflüssig sind.
   Also Wörter wie DER, DIE, DAS, DEN, DEM, UEBER, UNTER, AUF VON, IN etc.
   Für unser Beispiel gilt hier:
   BE$ = "NIMM DAS SCHWERT"
   BE$(1)="NIMM"
   BE$(2)="SCHWERT"
   Eigentlich ist es falsch, dies als den zweiten Schritt zu bezeichnen, da der zweite Schritt parallel zum 1. Schritt abläuft. Es ist also nicht so, daß der Satz in BE$(1) bis BE$(X) zerlegt wird und dann die überflüssigen Wörter aussortiert werden und somit BE$(1) bis BE$(X) neu formatiert werden muß.
   Dieser parallele Ablauf sieht also so aus:
   Das Zerlegemodul holt sich ein Wort aus dem Befehlsstring und überprüft sofort, ob es überflüssig ist oder nicht. Wenn ja, dann wird es erst gar nicht in die Befehlswortkette BE$(1) bis BE$(X) aufgenommen.
   Für die folgenden Seiten und Abschnitte sind Kenntnisse über den Umgang mit dem DIM-Befehl notwendig.

3. Was nun folgt ist die String-Codierung.
   Wie das Wort Codierung bereits verrät, kommen jetzt Zahlen mit ins Spiel. Der Trick besteht darin, den gesamten Befehlssatz in eine Zahl umzuwandeln. Dies hat den Vorteil, daß viel Speicherplatz gespart wird. Damit jedoch der Überblick über die vielen Zahlen gewährleistet bleibt, ist es unbedingt erforderlich, ständig schriftliche Notizen zu machen (Tabellen etc.)
   Bevor der Computer Befehle analysieren kann, braucht er erst einmal ein Vokabular, einen Wortschatz, auf den er ständig zurückgreifen kann.
   Wie Sie bereits wissen, zeichnet sich ein gutes Adventure hauptsächlich durch einen umfangreichen Wortschatz aus. Das eigentliche Problem besteht nun darin, einen möglichst großen Wortschatz in möglichst wenig Speicher zu stecken. Es gibt zahlreiche Möglichkeiten, dieses Problem zu lösen. Leider jedoch auch viele schlechte.
   Sicherlich kennen auch Sie Adventures, bei denen die Aufgabe des Spielers hauptsächlich darin besteht, den spärlichen Wortschatz des Spiels zu erraten. Solche Spieler belohnen Befehlseingaben ständig mit Antworten wie »Ich kenne dieses Wort nicht!« etc. Andere Spiele bringen nicht einmal eine Fehlermeldung, sondern ignorieren den Befehl einfach. Durch solche Fehlermeldungen wird der Spieler ständig vom eigentlichen Adventure abgelenkt und verliert schnell die Lust am Weiterspielen.
   Im Rahmen dieses Kurses werde ich Ihnen deshalb ein Befehlsanalysesystem vorstellen, das kaum Schwächen hat und universell für jedes Adventure einsetzbar ist.
   Bei diesem System sind im Computer Wort-Tabellen gespeichert, die jedem Wort einer Gruppe eine Zahl zuordnen.
   Die verschiedenen Wortgruppen sind
   — Verben
   — Gegenstände
   — Objekte
   sowie — Personen —,
   die wir zunächst vernachlässigen wollen. Natürlich gibt es auch eine Tabelle, die alle Wörter enthält, die aussortiert werden müssen.
   Und so sehen diese Tabellen aus:

WORTGRUPPEN-TABELLEN

VERBEN			GEGENSTÄNDE		
NIMM	=	1	SCHWERT	=	1
VERLIERE	=	2	SCHLUESSEL	=	2
OEFFNE	=	3	SEIL	=	3
GIB	=	4	FACKEL	=	4
SAGE	=	5	ARMBRUST	=	5
INVENTUR	=	6	HELM	=	6
BEFESTIGE	=	7	SCHILD	=	7

OBJEKTE			PERSONEN		
FENSTER	=	1	GEIST	=	1
TUER	=	2	MONSTER	=	2
TRUHE	=	3	THORIN	=	3
KISTE	=	4	GOMMEL	=	4
FALLTUER	=	5	ORK	=	5

Die Beispieltabellen sind nur sehr kurz gehalten. In Wirklichkeit können sie beliebig lang sein, und je länger sie sind, desto größer ist der Wortschatz des Adventures.

Wir haben vier Grundtabellen: Verben, Gegenstände, Objekte und Personen. Die Tabellen ordnen jedem Wort, das in ihnen enthalten ist, eine Zahl zu. Diese Zahlen sind jedoch nicht willkürlich gewählt, sondern das erste Wort jeder Tabelle hat immer den Wert 1 und das letzte Wort immer den Wert X, wobei X die Anzahl der in der Tabelle vorkommenden Worte darstellt.

Wie die Tabelle programmiert wird, soll uns im Moment noch nicht interessieren. Wir wissen nur, daß wir verschiedene Tabellen haben, die bestimmten Worten Zahlen zuordnen. Sicher erinnern Sie sich daran, daß ich Ihnen empfohlen habe, während und vor dem Programmieren ständig schriftliche Notizen zu machen (Heft zu führen). Gerade bei diesen Tabellen ist es wichtig, daß sie in sauberer schriftlicher Form vorliegen. Sie werden während der Programmierung des Spiels ständig benötigt.

Gehen wir doch noch einmal von unserem Beispielsatz aus:

Wir hatten BE$ = "NIMM DAS SCHWERT"

Diesen Satz haben wir dann in einzelne Worte zerlegt. Es ergab sich:

BE$(1) = "NIMM"
BE$(2) = "DAS"
BE$(3) = "SCHWERT"

Als nächstes hatten wir alle unwichtige Worte aussortiert und erhielten so:

BE$(1) = "NIMM"
BE$(2) = "SCHWERT"

Die Zahlen in den Klammern hinter BE$ geben an, wo das Wort im Satz steht — BE$(1) ist also das erste Wort im Satz (das Wort mit dem der Satz beginnt). Denken wir noch einmal zurück an unsere kleine Grammatiklehre, in der wir folgendes festgestellt haben:

1. Das Verb ist immer das erste Wort im Satz (also BE$(1))
2. Die folgenden Worte sind Objekte, Gegenstände oder Personen

Nun ist es nicht mehr schwer, zu erraten, wie die Befehlscodierung funktioniert:

Wir nehmen unser erstes Wort BE$(1) und suchen dann in der Tabelle für die Verben (da das erste Wort immer ein Verb ist) welche Zahl ihm zugeordnet ist. Für das Wort BE$(1) = "NIMM" erhalten wir gemäß Tabelle den Wert 1.

Wir können somit sagen: VERBZAHL = 1

Nun kommen wir zu unserem zweiten Wort BE$(2). Da wir nicht wissen, ob BE$(2) ein Gegenstand, ein Objekt oder eine Person ist, müssen wir in allen drei Tabellen nach dem Wort suchen.

Als Ergebnis erhalten wir dann entweder eine Gegenstands-, eine Objekt- oder eine Personenzahl.

Für unser Wort BE$(2) = "SCHWERT" erhalten wir also die Gegenstandszahl 1, da BE$(2) in der Tabelle für Gegenstände steht.

Die gesamte Codierung für unseren Satz BE$ = "NIMM DAS SCHWERT" hat uns also folgende Werte geliefert:

VERBZAHL = 1
GEGENSTANDSZAHL = 1
OBJEKTZAHL = 0
PERSONENZAHL = 0

Damit haben wir nun eine Möglichkeit kennengelernt, mit der man einen Satz durch Zahlen ausdrücken kann, die natürlich in Tabellen mit Wörtern definiert sein müssen.

Bei unserem Beispiel haben wir als Wert für die Objektzahl und die Personenzahl jeweils 0 erhalten. Dies liegt daran, daß in unserem Ausgangssatz »NIMM DAS SCHWERT« weder eine Person noch ein Objekt vorkommen. Das Schwert ist ein Gegenstand; Objekte sind Dinge, die nicht transportiert werden können (wie wir bereits definiert hatten).

Allerdings hat dieses Analyse- beziehungsweise Codierungssystem noch einen Haken:

Was passiert, wenn in einem Satz zwei Gegenstände auftreten? Zum Beispiel:

BE$= »VERKNOTE DAS SEIL AM SCHWERT«.

In diesem Satz treten zwei Gegenstände, Seil und Schwert, auf. In unserer Gegenstands-Tabelle hat Schwert den Wert 1 und Seil den Wert 3. Was ist aber nun unsere Gegenstandszahl?

Um dieses Problem zu lösen, müssen wir noch einmal auf unsere Tabelle, die alle möglichen ve-ob-ge-Kombinationen darstellt, zurückgreifen (die Personen werden vorerst noch vernachlässigt).

Aus dieser Tabelle können wir folgendes lesen:

1. Ein Befehlssatz enthält maximal 1 Verb — also nur 1 Verbzahl.
2. Ein Befehlssatz enthält maximal 1 Objekt — also nur 1 Objektzahl.
3. Ein Befehlssatz kann bis zu maximal zwei Gegenstände enthalten — folglich brauchen wir auch zwei Gegenstandszahlen.
   Wir bezeichnen diese als 1. Gegenstandszahl
   und 2. Gegenstandszahl

Was ist nun mit den Personen los?

Lassen Sie uns dazu folgende Beispiele von dem Adventure »The Hobbit« hernehmen, in dem besonders viele Personen auftreten.

a) SAGE THORIN »GIB MIR DIE KARTE«.
b) SAGE ELROND »LIES DIE KARTE«.
c) SAGE BARD »ERSCHIESSE DEN DRACHEN«.

Satz a und b sind eindeutig. Es kommt nur eine Person vor, und somit wird zur Codierung auch nur eine Personenzahl benötigt. In unserem Beispiel c kommen jedoch zwei Personen vor — Bard und der Drache. Dennoch wird nur eine Personenzahl benötigt.

Dies liegt daran, daß vom Befehlssatz nur eine Person betroffen ist. Die Person ist Bard. Der Drache kommt wiederum im Befehlssatz vor, der für Bard gilt.

Punkt c ist damit ein Beispiel für einen indirekten Befehl. Während wir bisher nur dem Computer direkt Befehle gaben, erhält der Computer jetzt den Befehl, einer bestimmten Person etwas zu befehlen (sagen).

Der Befehl für die Person Bard »ERSCHIESSE DEN DRACHEN« wird im Programm nicht vom Modul behandelt, das wir gerade aufbauen, sondern wird an ein Modul weitergeleitet, das für Reaktionen von Personen verantwortlich ist. Dazu jedoch später.

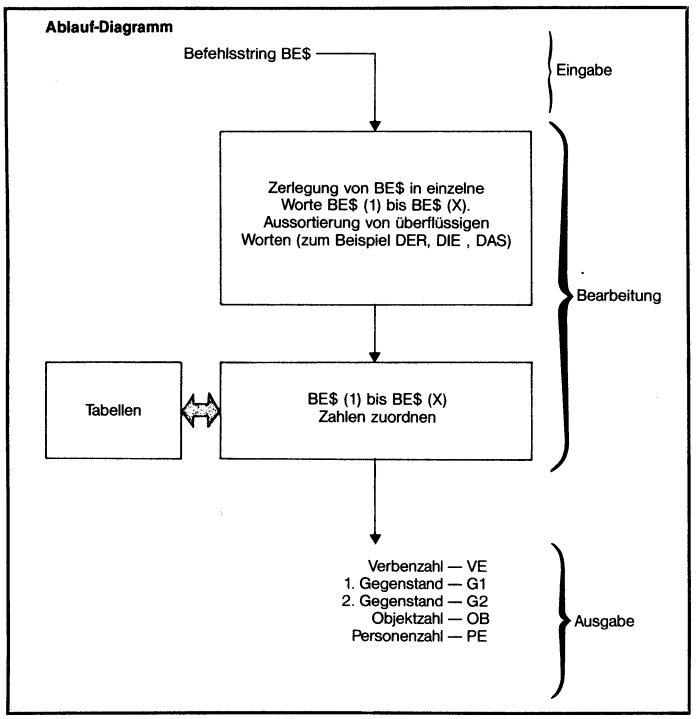
Wir wissen nun allerdings, daß im Modul, das für unsere direkten Befehle zuständig ist, nur eine Personenzahl benö-

tigt wird, da wir nie mehr als eine Person gleichzeitig ansprechen können.

Wir können also zusammenfassen, daß bei der Codierung eines Befehlssatzes folgende Werte ausgegeben werden:

VERBZAHL
1. GEGENSTANDSZAHL
2. GEGENSTANDSZAHL
OBJEKTZAHL
PERSONENZAHL

Hiermit sind nun auch schon die ersten Schritte auf dem Weg getan, der uns zum Programmieren der kompletten Befehlsanalyse führen wird. Im folgenden Diagramm ist das Ablaufschema unseres neuen Moduls (dem Befehlszerlegungs- und Codierungs-Modul) noch einmal in seinen Grundzügen zusammengefaßt.



Hier noch einige Beispiele:

Geben wir als Befehlsstring BE$=»OEFFNE DIE TUER« in das Codier-Modul, so erhalten wir als Ausgabe:

VERBZAHL = 3
1. GEGENSTANDSZAHL = 0
2. GEGENSTANDSZAHL = 0
OBJEKTZAHL = 2
PERSONENZAHL = 0
(alle Angaben gemäß unserer Beispieltabelle)

Für die Eingabe BE$=»VERLIERE DIE FACKEL« erhalten wir die Ausgabe:

VERBZAHL = 2
1. GEGENSTANDSZAHL = 4
2. GEGENSTANDSZAHL = 0
OBJEKTZAHL = 0
PERSONENZAHL = 0

Für die Eingabe BE$=»GIB DEM GEIST DEN SCHLUESSEL« erhalten wir die Ausgabe:

VERBZAHL = 4
1. GEGENSTANDSZAHL = 2
2. GEGENSTANDSZAHL = 0
OBJEKTZAHL = 0
PERSONENZAHL = 1

Für die Eingabe BE$=»BEFESTIGE DAS SEIL AM FENSTER« erhalten wir die Ausgabe:

VERBZAHL = 7
1. GEGENSTANDSZAHL = 3
2. GEGENSTANDSZAHL = 0
OBJEKTZAHL = 1
PERSONENZAHL = 0

Hiermit wäre die Therorie für unser neues Modul nahezu abgeschlossen. Allerdings müssen wir später noch Fälle betrachten und berücksichtigen, bei denen lange Sätze auftreten, die UND-Worte enthalten.

Programmierung des Zerlege-Chiffrier-Moduls

Bevor das Modul arbeiten kann, müssen die Tabellen programmiert werden. Für unsere Verb-Tabelle geht das folgendermaßen:

Die einzelnen Verben werden zunächst in DATA-Zeilen abgelegt.

Es gilt also:

..DATA NIMM, VERLIERE, OEFFNE, GIB, SAGE, INVENTUR, BEFESTIGE

Nun brauchen wir eine Variable, die die Anzahl der Verben beinhaltet. Für die Verb-Tabelle sei dies die Variable VZ (VZ = Anzahl der Verben — nicht zu verwechseln mit der Verbzahl VE).

Für unsere Verb-Tabelle, die aus sieben Worten besteht, gilt also:

VZ = 7

Für unsere Verb-Tabelle müssen wir nun ein Feld indizieren:

Das Feld für die Verb-Tabelle bestimmen wir als Stringfeld VE$. Es gilt also:

DIM VE$(VZ)

Falls Sie mit dem DIM-Befehl noch nicht genug vertraut sind, so sollten Sie gleich im Commodore-Handbuch ab Seite 98 nachlesen.

Nun müssen wir das Feld füllen:

```
FOR I=1 TO VZ : READ VE$(I) : NEXT I
```

Nach diesem Verfahren werden alle Tabellen programmiert. Für die Verb-Tabelle sieht das so aus:

```
VE$(1) = "NIMM"
VE$(2) = "VERLIERE"
VE$(3) = "OEFFNE"
VE$(4) = "GIB"
VE$(5) = "SAGE"
VE$(6) = "INVENTUR"
VE$(7) = "BEFESTIGE"
```

Wir nehmen nun einmal an, wir hätten ein Verb BE$(1)="OEFFNE".

Wenn wir wissen wollen, welche Verbzahl BE$(1) hat, so gehen wir folgendermaßen vor:

```
10 FOR I=1 TO VZ
20 IF BE$(1) = VE$(I) THEN VE=I
30 NEXT I
40 PRINT"VERBZAHL = ";VE
```

Wir können Zeile 20 auch so ändern, daß Abkürzungen akzeptiert werden.

```
20 IF BE$(1) = LEFT$(VE$(I),LEN(BE$(1)) THEN VE=I
```

Der Spieler kann nun also statt VERLIERE auch einfach nur V eingeben, und das Programm versteht trotzdem, was gemeint ist. Es werden jedoch nur Abkürzungen verstanden, die eindeutig sind. Will man das Verb SUCHE mit S abkürzen, so darf in der Verbtabelle neben dem Verb SUCHE kein anderes Verb mehr vorkommen, das mit S beginnt.

Ist dies jedoch der Fall, so wird stets das Verb erkannt, das in der Tabelle den höheren Wert hat. Dies ist jedoch nicht tragisch, da der Spieler im Laufe des Spiels schnell herausfinden wird, wie er Wörter abkürzen muß, damit sie richtig interpretiert werden.

Bitte geben Sie nun einmal Listing 4 ein:

```
10 INPUT"WAS NUN ";BE$:REM BEFEHL                    <004>
20 GOSUB 51000          :REM ZERLEGUNG               <019>
30 PRINT:PRINT"BE$=";CHR$(34);BE$;CHR$(34)
   :PRINT                                            <227>
40 FOR I=1 TO 10                                     <218>
50 :PRINT"BE$(";RIGHT$(STR$(I),LEN(STR$(I)
   )-1);")=";CHR$(34);BE$(I);CHR$(34)                <034>
60 NEXT:PRINT                                        <145>
70 GOTO 10                                           <048>
51000 REM BE$ IN BE$(1)-BE$(10) ZERLEGEN             <227>
51010 FOR I=1 TO 10:BE$(I)="":NEXT                   <236>
51020 WZ=1                                           <167>
51030 FOR I=1 TO LEN(BE$)                            <045>
51040 :IF MID$(BE$,I,1)=" "THEN WZ=WZ+1:GO
      TO 51060                                       <207>
51045 :IF WZ>10 THEN PRINT"EINGABE IST ZU
      LANG !":I=LEN(BE$)+1:GOTO 51060                <147>
51050 :BE$(WZ)=BE$(WZ)+MID$(BE$,I,1)                 <185>
51060 NEXT I                                         <006>
51070 RETURN                                         <211>
```

Listing 4.

Wenn Sie das Programm anschließend starten, so werden Sie um die Eingabe eines Befehls BE$ gebeten.

Wenn Sie einen Befehl eingeben und anschließend RETURN drücken, so wird auf dem Bildschirm ausgegeben, wie der Befehlsstring BE$ in BE$(1) bis BE$(10) zerlegt wird. Es können also nur Sätze eingegeben werden, die nicht mehr als zehn Worte enthalten (natürlich kann das leicht ausgebaut werden).

Geben Sie einmal verschiedene Sätze ein, und sehen Sie was passiert. Dieses Programm, in dem der Befehlsstring BE$ in einzelne Worte zerlegt wird, was ja der erste Schritt zur Befehlscodierung ist, hat jedoch noch einen kleinen Haken.

Wenn man zwischen den Worten im Satz mehrere SPACEs läßt, so werden diese SPACEs als Worte interpretiert. Diesen Fehler werden wir bald beheben.

Unser neues Modul soll immer ab Zeile 51000 stehen, also direkt nach dem Befehlseingabe-Modul.

Dokumentation zu Listing 4:

10 bis 70 sind nur zur Demonstration des Moduls 51000 gedacht, und müssen deshalb in ihrer Funktionsweise hier nicht näher besprochen werden.

51000 Beginn des Zerlege-Moduls

51010 Hier wird das Feld, das die einzelnen Wörter des Befehlssatzes enthält, gelöscht, damit später keine Irrtümer auftreten können.

51020 Hier wird die Wort-Zählvariable WZ auf 1 gesetzt.

51030 Diese Schleife durchläuft den Befehlsstring BE$ vom ersten bis zum letzten Zeichen.

51040 Prüfung, ob es sich beim jeweiligen Zeichen um ein Leerzeichen (SPACE) handelt. Wenn ja, so weiß das Programm, daß ein neues Wort beginnt. In diesem Fall wird die Wort-Zählvariable WS um 1 erhöht.

51045 Hier wird verhindert, daß die zulässige Wortzahl von 10 überschritten wird.

51050 Das aktuelle Wort BE$(WZ) wird aufgebaut.

51060 Sprung zum Beginn der Schleife I

51070 Ende des Moduls

Dieses Modul liefert nach Aufruf bis jetzt also die Werte BE$(1) bis BE$(WZ) — die einzelnen Worte von BE$ und WZ — die Variable die für die Anzahl der Wörter steht.

Die Funktionsweise dieses Moduls dürfte bis jetzt eigentlich leicht zu verstehen sein. Falls Sie Schwierigkeiten haben, so gehen Sie das bisher Gesagte noch einmal ausführlich durch.

Der nächste Schritt zur Fertigstellung des Moduls besteht nun, wie bereits theoretisch behandelt, darin, überflüssige Worte auszusortieren, beziehungsweise erst gar nicht in die Wortkette BE$(1) bis BE$(10) aufzunehmen.

Wir müssen unser erstes Programm also entsprechend Listing 5 verbessern:

```
5 GOSUB 60000:REM TABELLEN DEFFINIEREN               <183>
10 INPUT"WAS NUN ";BE$:REM BEFEHL                    <004>
20 GOSUB 51000          :REM ZERLEGUNG               <019>
30 PRINT:PRINT"BE$=";CHR$(34);BE$;CHR$(34)
   :PRINT                                            <227>
40 FOR I=1 TO 10                                     <218>
50 :PRINT"BE$(";RIGHT$(STR$(I),LEN(STR$(I)
   )-1);")=";CHR$(34);BE$(I);CHR$(34)                <034>
60 NEXT:PRINT"{DOWN}ANZAHL DER WOERTER (WZ
   ) :";WZ:PRINT                                     <162>
70 GOTO 10                                           <048>
51000 REM BE$ IN BE$(1)-BE$(10) ZERLEGEN             <227>
51010 FOR I=1 TO 10:BE$(I)="":NEXT                   <236>
51020 WZ=1                                           <167>
51030 FOR I=1 TO LEN(BE$)                            <045>
51040 :IF MID$(BE$,I,1)=" "THEN GOSUB 5110
      0:GOTO 51060                                   <100>
51045 :IF WZ>10 THEN PRINT"EINGABE IST ZU
      LANG !":I=LEN(BE$)+1:GOTO 51060                <147>
51050 :BE$(WZ)=BE$(WZ)+MID$(BE$,I,1)                 <185>
51060 NEXT I                                         <006>
51070 RETURN                                         <211>
51100 REM AU$ WOERTER AUSSORTIEREN                   <120>
51110 IC=0:FOR I1=1 TO AZ                            <050>
51120 :IF BE$(WZ)=AU$(I1)THEN IC=1                   <054>
51130 NEXT I1                                        <125>
51140 IF IC=0 THEN WZ=WZ+1:RETURN                    <226>
51150 BE$(WZ)="":RETURN                              <000>
60000 REM *** TABELLEN-DEFFINIEREN ***               <104>
60005 DATA DER,DIE,DAS,DEN, ,UEBER,UNTER,A
      UF,IN                                          <002>
60010 AZ=9:FOR I=1 TO AZ:READ AU$(I):NEXT            <209>
60015 RETURN                                         <231>
```

Listing 5.

Dokumentation zum Listing:

5 Hier wird das Unterprogramm, das die Tabellen definiert, aufgerufen.

10 bis 70 Sind wiederum nur für die Demonstration des Programmes nötig.

51000-51070 Sind analog zu Listing 4 — jedoch mit einer kleinen Änderung in Zeile 51040.

51040 Hier wird, wie auch schon im ersten Programm festgestellt, wann ein neues Wort beginnt (ein Leerzeichen SPACE wird gesucht in BE$). Angenommen wir haben ein Leerzeichen gefunden, so wurde im ersten Programm das aktuelle Wort BE$(1) bis BE$(10) übernommen und die Wort-Zählvariable WZ um 1 erhöht (um das nächste Wort zu bilden). Dadurch wurde jede durch SPACE getrennte Zeichenfolge als Wort anerkannt (so ergab sich auch der oben erwähnte Fehler, bei dem mehrere Leerzeichen hintereinander als Wort interpretiert wurden). Zur Korrektur dieses Fehlers und zur Aussortierung der überflüssigen Wörter wird deshalb jetzt ein kleines Unterprogramm ab Zeile 51100 aufgerufen, in dem entschieden wird, ob das aktuelle Wort BE$(WZ) mit in die Wortkette BE$(1) bis BE$(10) aufgenommen wird oder nicht.

60000-60015 Hier wird die Tabelle für die Wörter, die aussortiert werden sollen, definiert, nach dem Schema, das wir bereits besprochen haben. Das Stringfeld für die auszusortierenden Wörter sei AU$.
Die Anzahl der Wörter in der AU$-Tabelle ist AZ.

Und so sieht dieses Unterprogramm aus:

51100 Start des Unterprogrammes

51110 Die Check-Variable IC wird auf 0 gesetzt. Die Schleife I1 durchläuft sodann die AU$-Tabelle.

51120 Prüfung, ob es sich bei dem aktuellen Wort BE$(AW) um ein Wort handelt, das in der AU$-Tabelle vorkommt, und somit aussortiert werden muß. Ist dies der Fall, so wird die Check-Variable IC auf 1 gesetzt.

51130 Sprung zum Schleifenbeginn

51140 Nachdem die Schleife I1 durchlaufen ist, wird der Wert der Check-Variable IC geprüft. Ist IC=0, so kommt das aktuelle Wort nicht in der AU$-Tabelle vor und muß auch nicht aussortiert werden; es wird in die Wortkette BE$(1) bis BE$(10) übernommen, der Wortzähler WZ wird um 1 erhöht und das Unterprogramm ist beendet.

51150 Die Bedingung in Zeile 51140 wurde nicht erfüllt, also muß es sich bei dem aktuellen Wort BE$(AW) um ein Wort der AU$-Tabelle handeln, es muß aussortiert werden.
Dazu wird BE$(AW) gelöscht und der aktuelle Wortzähler WZ nicht erhöht. Die Check-Variable IC dient uns also zum Überprüfen von Bedingungen.

Starten Sie das Programm mit RUN. Da es durch die Zeilen 10 bis 70 dokumentiert ist, braucht der Programmablauf hier nicht näher erklärt zu werden. Geben Sie bitte mehrere Befehlssätze ein, und untersuchen die Ergebnisse. Wenn Sie allerdings wirre Buchstabenketten eingeben, so werden diese akzeptiert, solang sie nicht in der AU$-Tabelle vertreten sind. Wenn Sie die DATA-Zeile betrachten, in der die Worte stehen, die aussortiert werden sollen (die also die AU$-Tabelle bilden), so werden Sie dort auch ein Leerzeichen finden; es ist nötig, um den bereits besprochenen Fehler (mehrere Leerzeichen zwischen den Wörtern in BE$) zu beheben, Leerzeichen werden nun nicht mehr als Worte akzeptiert.

Als kleine Zwischenübung können Sie einmal versuchen die AU$-Tabelle zu erweitern. Sie müssen dazu nur die DATA-Zeilen erweitern und die AZ-Variable entsprechend anpassen.

Unser nächster Arbeitsschritt besteht nun darin, die Wort-Codierung zu programmieren.

Zunächst wieder einige Grundbetrachtungen:

Bisher haben wir immer nur Befehlssätze betrachtet, die nur einen Befehl enthielten, in denen also kein UND vorhanden war.

Für die folgenden Betrachtungen und Programme müssen jedoch auch Sätze, die mehrere Befehle enthalten, berücksichtigt werden.

Zu Beginn habe ich Ihnen bereits eine kleine Eingabe-Szene aus dem Adventure »The Hobbit« dargestellt. Daraus war zu erkennen, daß bei dem Analysesystem des Hobbits immer nur ein Befehl nach dem anderen ausgeführt werden kann. Dies ist daran zu erkennen, daß der Computer auf einen Befehlssatz wie »Nimm das Schwert und das Seil« nicht etwa »Sie nehmen das Schwert und das Seil« antwortet, sondern sich erst mit »Sie nehmen das Schwert« und danach mit »Sie nehmen das Seil« meldet.

Betrachten wir einmal den folgenden Befehlssatz, der mehrere Befehle enthält: BE$=»NIMM DAS SCHWERT UND DAS SEIL UND GEH NACH NORDEN«.

Nachdem das Modul den Satz zerlegt und überflüssige Wörter AU$ aussortiert hat, liegt folgendes Ergebnis vor:

BE$(1) = "NIMM"
BE$(2) = "SCHWERT"
BE$(3) = "UND"
BE$(4) = "SEIL"
BE$(5) = "UND" — bisher nicht berücksichtigt
BE$(6) = "NORDEN" — bisher nicht berücksichtigt

Wörter wie UND dürfen nicht aussortiert werden.

Bitte wundern Sie sich nicht, warum ich das Verb »GEH« aussortiert habe — Richtungsangaben sind Ausnahmefälle, die in einem der nächsten Abschnitte ausführlich behandelt werden.

Damit die folgenden Erklärungen nicht zu kompliziert werden, wollen wir BE$(5) und BE$(6) zunächst wieder streichen.

Wir betrachten also nur noch BE$(1) bis BE$(4), also einen Befehlssatz, der zwei Befehle enthält.

Da das Programm ein Wort nach dem anderen liest, brauchen wir einen Wortzähler, den wir mittels der Variablen WZ aufbauen wollen.

Am Anfang des Satzes, also vor Beginn des Lesevorgangs wird WZ auf 1 gesetzt.

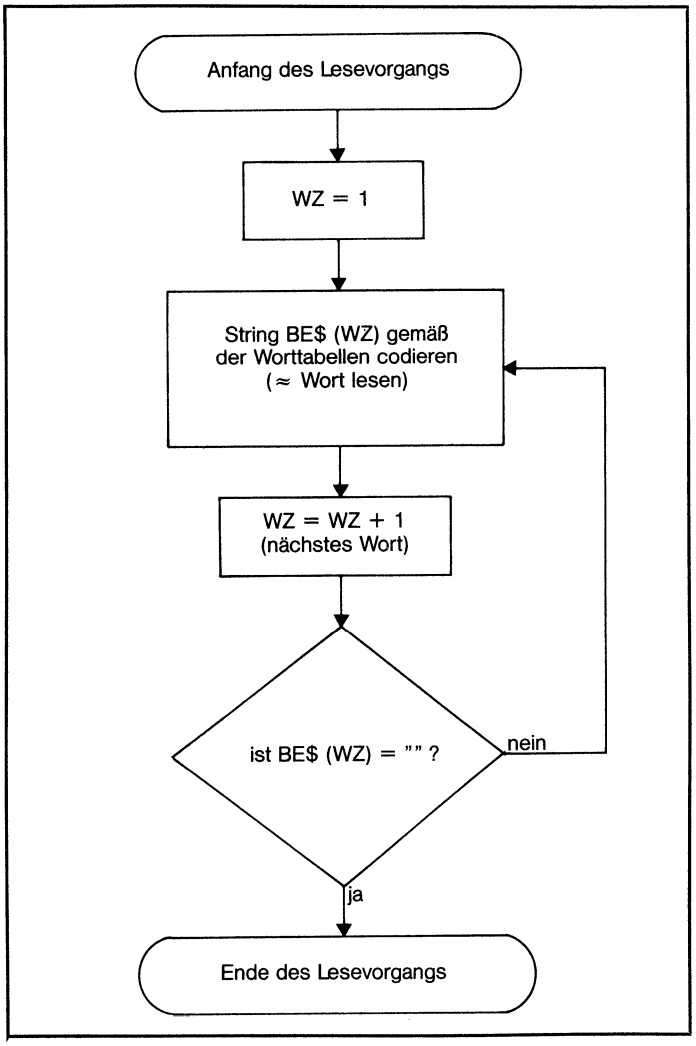
Nun liest der Computer das Wort BE$(WZ).

Nachdem er es gelesen und codiert hat, erhöht er den Wortzähler um 1, es gilt also WZ=WZ + 1.

Nun wird überprüft, ob das Wort BE$(WZ) existiert, das heißt, es wird überprüft, ob der String BE$(WZ) Zeichen enthält oder er " "(leer) ist.

Erweist sich der String BE$(WZ) als nicht leer, so wird der Lesevorgang fortgesetzt. Andernfalls wird er beendet.

Das Ablaufschema für das Lesen des Befehlssatzes sieht also folgendermaßen aus:

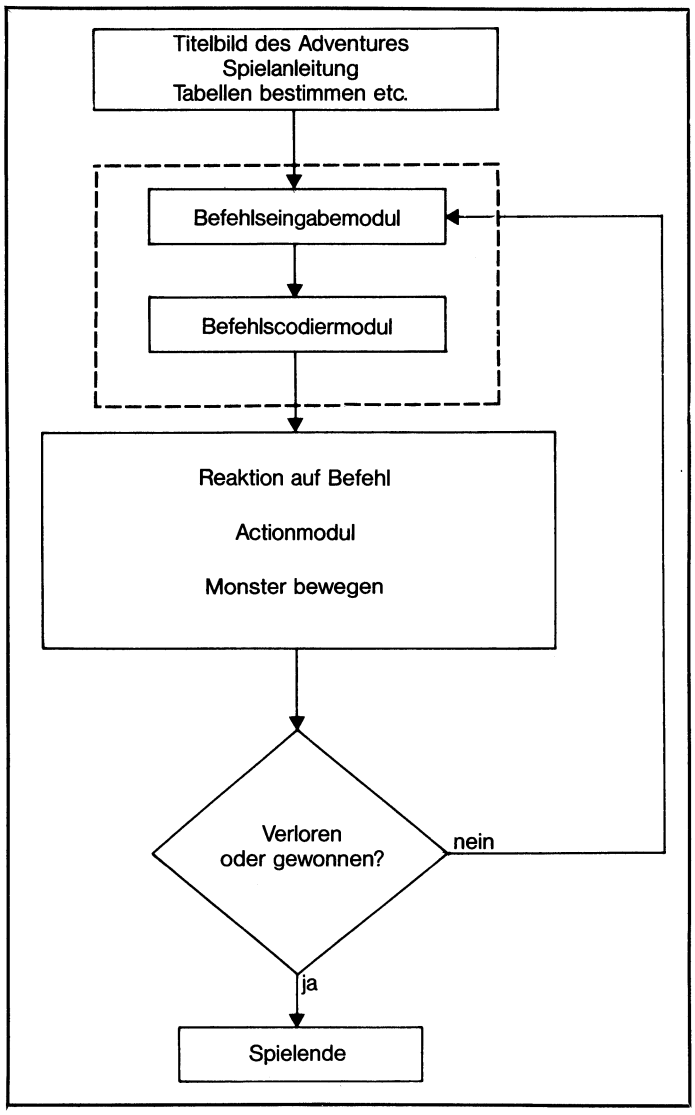


Achtung: Mit ein Wort LESEN ist das gleiche wie mit ein Wort CODIEREN gemeint.

Dieses Ablaufschema ist noch relativ einfach zu verstehen.

Für unsere Zwecke ist es im Moment jedoch nicht ausreichend. Es reicht höchstens für Programme die nur Zwei-Wort-Befehle verstehen.

Um dies besser verstehen zu können, müssen wir einen Blick auf das gesamte Ablaufschema eines Adventures werfen:



Das Befehlseingabemodul und das Befehlscodiermodul (worin natürlich auch Zerlegung und Aussortierung enthalten sind) sind mit einem Kästchen gestrichelt umrandet. Wir werden diese Module schon bald zu einem einzigen Modul zusammenfassen, das für jedes Adventure universell einsetzbar ist. In dem gestrichelten Kästchen ist unter anderem auch das Ablaufschema des Lesevorganges, das wir gerade zuvor behandelt haben, enthalten.

Das Gesamtablaufschema ist eigentlich relativ leicht zu verstehen. Am Anfang erscheint das Titelbild des Spiels und die Spielanleitung, und es werden Tabellen eingelesen, eventuell Sprites definiert etc. Danach beginnt das Spiel.

Dabei wird zunächst das Befehlseingabe- und Codiermodul aktiviert.

Daraufhin geht es zum Actionmodul. Dies ist das eigentliche Adventure, das heißt, das Actionmodul ist von Spiel zu Spiel verschieden.

Im Actionmodul wird auf Befehle reagiert, Lagebericht erstattet, und außerdem werden Personen, wie Monster gesteuert, bewegt, lebendig gemacht. Das Actionmodul stellt auch die Aufgaben und verwaltet die Spielkarte, das heißt es weiß, wo gerade welcher Gegenstand liegt, wo sich jede Person befindet, was sie gerade macht, welche Türen offen und welche geschlossen sind und noch vieles mehr. Der Umfang dieses Moduls bestimmt letztendlich die Qualität und Länge des Spiels.

Am Ende des Actionmoduls wird bestimmt (geprüft), ob das Spiel zu Ende ist (verloren oder gewonnen), oder ob es weiter geht, also zurück zum Befehlseingabe- und Codiermodul.

Sie sehen also, wie einfach das Grundprinzip eines Adventures eigentlich ist. Und dies ist ja schließlich das Ziel dieses Kurses. Sie sollen lernen sich gute Adventures auszudenken, und diese dann möglichst problemlos und ohne großen Aufwand in ein anderes Programm umzusetzen.

Nachdem das Befehlszerlegemodul den Befehlsstring sortiert und in einzelne Worte zerlegt hat, gibt es diese Worte BE$(1) bis BE$(X) an das Codiermodul weiter.

Der Sinn der vier dort vorkommenden Wörter lautet in Kurzform:

NIMM SCHWERT UND SEIL

Wie liest das Codiermodul diesen Satz nun eigentlich?

Eine erste Antwort auf diese Frage ist, wie bereits besprochen, »Wort für Wort«.

Welches Wort bearbeitet wird, bestimmt die Variable WZ (Wortzähler). Damit das Lesen beim ersten Wort beginnt, setzten wir die Wortzähl-Variable WZ am Anfang des Lese- beziehungsweise Codiermoduls auf 1.

Sodann wird das Wort BE$(WZ) gelesen, also codiert. Und das geht folgendermaßen:

Zuerst wird die Check-Variable IC und die Wortvariablen auf 0 gesetzt.

```
IC = 0 : VE=0 : G1=0 : G2=0 : OB=0 : PE=0
```

Nun wird überprüft, ob der String BE$(WZ) in der Verb-Tabelle enthalten ist.

```
FORI: 1 TO VZ : REM VZ = Anzahl der Verben
IF BE$(WZ) = VE$(I) THEN IC=1 : VE=I
NEXT I
```

Wird bei diesem Vorgang also ein Verb gefunden, so wird dieses Verb zur Verbzahl VE codiert und die Check-Variable IC auf 1 gesetzt. Auf die gleiche Art und Weise werden auch alle anderen Wort-Tabellen durchlaufen (Gegenstände, Objekte, Personen).

Es werden also alle Tabellen auf das Wort BE$(WZ) hin untersucht. Sie könnten nun einräumen, daß wir für das erste Wort doch eigentlich nur die Verb-Tabelle durchsuchen müßten, da das erste Wort eines Satzes ja immer nur ein Verb sein kann. Damit haben Sie recht. Ich schlage jedoch trotzdem die obige Methode vor. Damit das Modul jedoch möglichst schnell abläuft, kann man nach jedem Tabellendurchlauf überprüfen, ob sich die Check-Variable zum Wert 1 geändert hat und in diesem Falle (IC=1) die restlichen Tabellen überspringen. Dies ist möglich, da ein Wort, das in Tabelle A gefunden worden ist, auf keinen Fall ebenfalls in Tabelle B vertreten sein kann. Es gibt schließlich kein Wort, das gleichzeitig Verb und Person ist. Wenn alle Tabellen durchlaufen sind, wird die Check-Variable IC wieder auf ihren Zustand hin überprüft.

Erweist sich IC als 0, so steht fest, daß das Wort BE$(WZ) in keiner Tabelle vertreten ist. Natürlich auch nicht in der Tabelle der AU$-Worte (Tabelle der Worte, die überflüssig sind).

Ist dies der Fall, so bringt das Programm die Fehlermeldung

```
"Mein Wortschatz kennt das Wort ";BE$(WZ);" nicht ;".
```

Daraufhin wird die Befehlscodierung abgebrochen, und es ist folgendes möglich:

a) Es erfolgt ein Sprung zum Actionmodul. Dies hat zur Folge, daß alle Monster, Geister etc. wieder am Zug sind. Angenommen, der Spieler begegnet gerade einem Monster und gibt als Befehl »Ermorde das Monster« ein. Da der Computer jedoch leider das Wort »ermorde« nicht kennt, schlägt das Monster bei Fall a) voll zu — GAME OVER.

b) Es erfolgt ein Rücksprung zum Befehlseingabemodul.

Die Methode a) ist die, die am meisten vertreten ist.

Ich halte es jedoch für sinnvoller, Methode b) zu verwenden, da hier sichergestellt ist, daß der Spieler nicht auf Grund eines Mißverständnisses verliert.

Bei einem guten Adventure sollte der Spieler nur auf Grund von taktischen Fehlern verlieren und niemals auf Grund von Tippfehlern.

Wir wollen nun aber annehmen, daß keine Fehlermeldung auftritt und das Programm in seinem Ablauf fortfährt. Die Check-Variable IC ist also 1. Damit ist auch eine Wortzahl (Verbzahl VE, 1. Gegenstandszahl G1, etc.) gefunden worden. Bei dem ersten Wort, also wenn WZ = 1 ist, muß es sich im Normallfall stets um die Verbzahl VE handeln. Das erste Wort ist somit codiert. Nun wird die Wortzählvariable um 1 erhöht. Das nächste Wort wird bearbeitet. Zunächst wird überprüft, ob dieses Wort überhaupt existiert, also ob der String BE$(WZ) Zeichen enthält. Wenn nicht, so ist der Lesevorgang beendet und das Actionmodul ist dran. Wir nehmen jedoch an, daß das Wort BE$(WZ) existiert.

Gemäß unserem Ablaufschema wird der Lesevorgang nun für das neue Wort wiederholt.

Halt! Was passiert eigentlich, wenn es sich bei dem neuen String BE$(WZ) um »UND« handelt, also ein neuer Befehl innerhalb des Befehlssatzes folgt? Das heißt auch, daß nach dem »UND« ein neues Verb folgen wird, oder auch nicht, wie es bei »NIMM SCHWERT UND SEIL« der Fall ist.

Angenommen es folgt jedoch ein neues Verb (NIMM SCHWERT UND OEFFNE TUER), dann würde sich unsere Verbzahl ändern und der NIMM-Befehl verloren gehen.

Die Lösung dieses Problems lautet wie folgt:

Bevor der Lesevorgang für das neue Wort BE$(WZ) beginnt, wird überprüft, ob dieses Wort gleich »UND« ist. »UND« braucht nicht in einer Tabelle definiert zu werden. Wir fragen im Programm direkt

IF BE$(WZ) = "UND"...

Ist dies der Fall, so setzen wir die Variable UD auf 1.

Dann wird der Lesevorgang beendet. Es folgt ein Sprung zum Actionmodul. Im Actionmodul wird nun auf den ersten Befehl des Befehlssatzes reagiert. In unserem Fall meldet sich der Computer mit

»SIE NEHMEN DAS SCHWERT«

Nachdem dies geschehen ist, kehren wir (nach Schema) zurück zum Befehlseingabemodul.

Halt! Wir können doch nicht einfach einen neuen Befehlsstring BE$ vom Spieler holen, wo der alte BE$ doch noch nicht voll ausgewertet ist. Wir müssen in der ersten Zeile des Befehlseingabemoduls deshalb abfragen, welchen Zustand die Variable UD hat.

Hat sie den Wert 1, so wissen wir, daß der letzte Befehlsstring noch nicht voll ausgeführt ist.

Das Befehlseingabemodul muß also übersprungen werden und zwar direkt zum Lese-(Codier-)Modul.

Der Sprung muß jedoch so sein, daß der Wortzähler WZ nicht wieder wie zuvor auf 0 gesetzt wird, es soll schließlich an der Stelle im Satz weitergelesen werden, an der vorher unterbrochen wurde und nicht wieder vom ersten Wort des Satzes an.

Auch Verbzahl VE, Objektzahl OB und Personenzahl PE dürfen nicht gelöscht werden. Würde man bei unserem Beispielsatz »NIMM SCHWERT UND SEIL« nach der Befehlsausführung von NIMM SCHWERT die Verbzahl löschen, so würde der Computer nach Beendigung des zweiten Lesevorgangs nur wissen, daß ein Seil gemeint ist, jedoch nicht, was er damit machen soll. Falls jedoch ein neues Verb gefunden wird, so nimmt die Verbzahl einfach den Wert dieses neuen Verbs an und vergißt somit den alten Wert. Durch dieses Verfahren wird gewährleistet, daß alle UND-Zusammenhänge richtig interpretiert werden.

Dies ist ein besonders bemerkenswerter Vorteil unserer Befehlscodierung, den kaum ein anderes Adventure bietet.

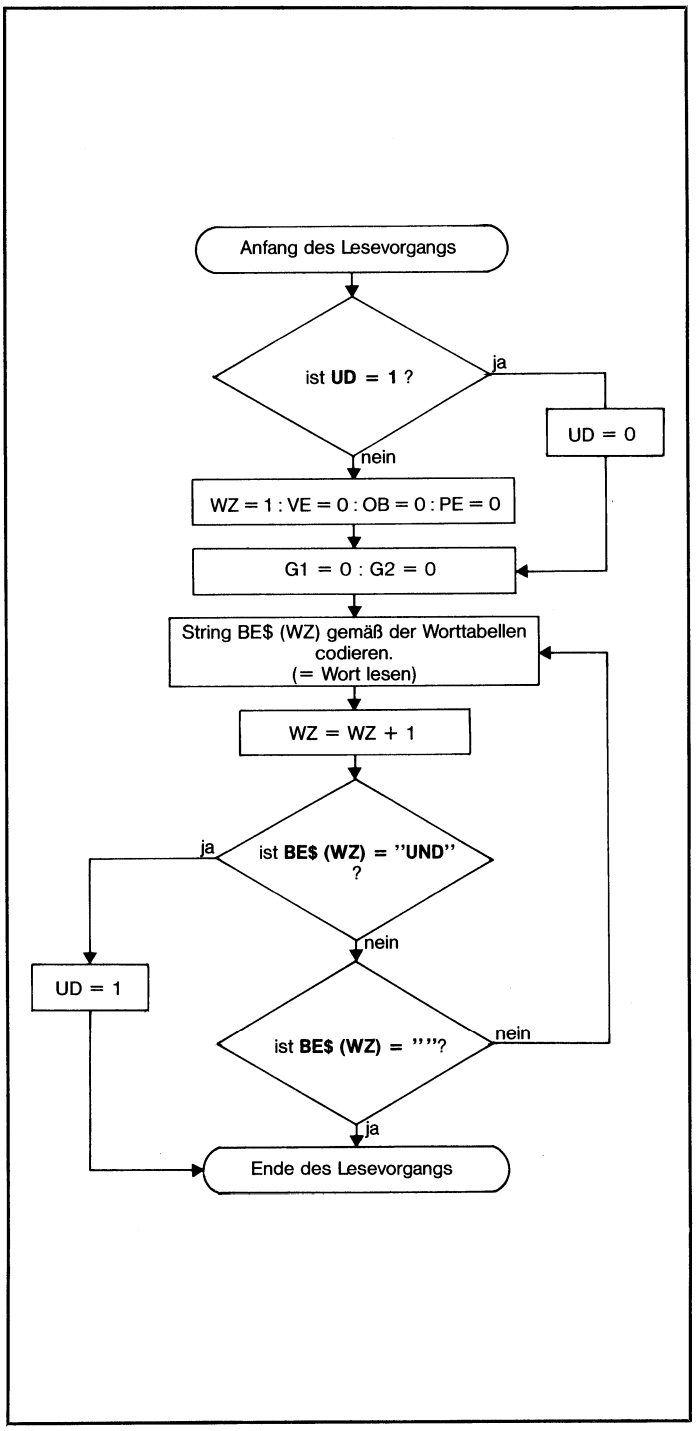
Die ersten Zeilen des Codiermoduls müssen folglich lauten:

1. Zeile: IF UD=1 THEN UD=0 : GOTO 3.Zeile
2. Zeile: WZ=1 : VE=0 : OB=0 : PE=0
3. Zeile: IC=0 : G1=0 : G2=0

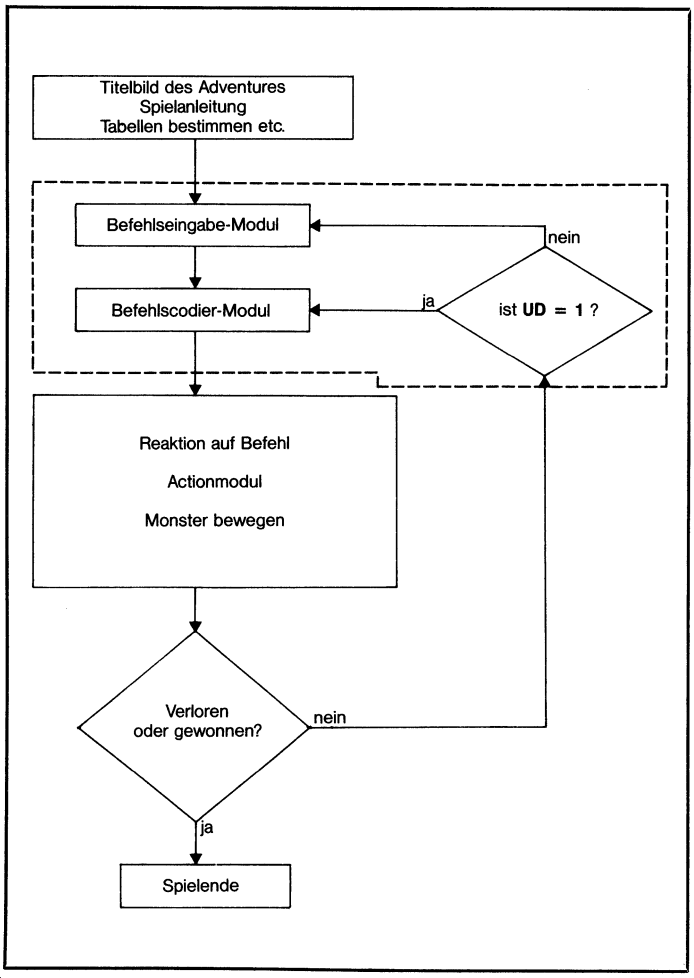
In der ersten Zeile wird UD wieder auf Null gesetzt, damit das Befehlseingabemodul nicht wieder übersprungen wird.

Findet sich im Laufe des zweiten Lesevorgangs jedoch wieder ein UND, so wird die UND-Variable UD einfach wieder auf 1 gesetzt.

Dies findet in dem nun verbesserten Ablaufschema seinen Ausdruck:



Durch diese Veränderung ändert sich natürlich auch in unserem Grundschema für Adventures etwas. Das neue Ablaufschema sieht deshalb folgendermaßen aus:



Damit wäre die Theorie für die Vervollständigung unseres neuen Moduls auch schon beendet. Unser neues Modul, das Befehlseingabe, -zerlegung, -sortierung und -codierung (der Lesevorgang) in sich vereint, steht im Programm wieder ab Zeile 50000. Dieses Modul können wir schon als halbwegs »intelligent« bezeichnen, da es immer mit GOSUB 50000 aufgerufen wird und dann selbständig entscheidet, woran es arbeiten muß, für die Befehlseingabe oder für die weitere Ausführung eines Befehlssatzes, der UND-Elemente enthält, also aus mehreren Befehlen besteht.

Diese Intelligenz macht es uns möglich, das Modul für jedes Abenteuer einzusetzen. Dabei muß höchstens der Wortschatz (die Worttabellen) abgeändert werden. In dem verbesserten Grundablaufschema für Abenteuerspiele können Sie erkennen, wie unser neues Modul eingesetzt werden muß.

Das Listing zum neuen Modul, werde ich nur noch leicht dokumentieren, da es schon ausführlich erklärt worden ist.

Während des Abtippens vom Listing sollten Sie ständig Blicke auf die Variablentabelle, die gleich folgt, werfen, damit Ihnen der Bezug zur Theorie deutlich wird.

Hier sind nun alle Variablen, die im Listing auftauchen:

AZ	Anzahl der Wörter, die in der Tabelle der auszusortierenden Wörter (der, die, das, etc.) stehen.
AU$(1) bis AU$(AZ)	Tabelle der auszusortierenden Worte.
VZ	Anzahl der Verben
VE$(1) bis VE$(VZ)	Tabelle der Verben
GZ	Anzahl der Gegenstände
GE$(1) bis GE$(GZ)	Tabelle der Gegenstände
OZ	Anzahl der Objekte
OB$(1) bis OB$(OZ)	Tabelle der Objekte
PZ	Anzahl der Personen
PE$(1) bis PE$(PZ)	Tabelle der Personen
UD	UND-Variable (wurde bereits erklärt)
BE$	String, der den kompletten Befehlssatz enthält.
BE$(1) bis BE$ (maximal 10)	Einzelne Worte des Befehlssatzes BE$, wobei überflüssige Worte bereits aussortiert sind.
WZ	Wortzählvariable, gibt an, welches Wort gerade bearbeitet wird.
IC	Checkvariable
I	Schleife
I1	Schleife, innerhalb der Schleife I

Alle Variablen die mit I beginnen sind entweder Check-Variablen oder Schleifen.

VE	Verbzahl (wird aus Verbtabelle ermittelt)
OB	Objektzahl (wird aus Objekttabelle ermittelt)
PE	Personenzahl (wird aus Personentabelle ermittelt)
G1, G2	1. und 2. Gegenstandszahl (werden aus Gegenstandstabelle ermittelt)

Dies wären auch schon alle auftretenden Variablen. Diese Variablen sind im ganzen Kurs beibehalten, damit Sie niemals unnötig verwirrt werden. Auf eventuelle Änderungen werde ich stets besonders hinweisen. Natürlich werde ich die Tabelle der Variablen im Laufe des Kurses ständig ergänzen. Ich empfehle Ihnen, bei allen Ihren Adventures immer die gleichen Variablen zu verwenden, das heißt, daß die Wortzählvariable WZ in jedem Adventure WZ lautet.

Durch dieses System ist es möglich, sehr schnell Adventures zu programmieren. Man hat ein eindeutiges Schema und kann sich beim Programmieren des Adventures voll und ganz auf das Abenteuer konzentrieren und muß nicht ständig über Programmierung und Variablenbelegung nachdenken. Man kann sich eine Karte anlegen, auf der alle Variablen, die in jedem Adventure verwendet werden, verzeichnet sind. Diese Tabelle hat man dann beim Programmieren als Hilfe immer neben sich liegen.

Ich würde mich freuen, wenn alle, die an diesem Kurs teilnehmen, auch diese Variablen übernehmen. Man könnte so vielleicht einen Standard für die Adventureprogrammierung festlegen, der sicherstellt, daß jeder Kursteilnehmer das Programm eines anderen Kursteilnehmers schnell verstehen und eventuell Fehler auch besser finden kann.

Wenn Sie schon einmal ein längeres Adventure aus einer Zeitschrift abgetippt haben, so wissen Sie sicher, wie chaotisch diese Listings oftmals sind und in welchem Maße eine Fehlersuche dadurch zu einem fast unmöglichen Unterfangen wird.

Geben Sie nun das Listing 6 der neuen Module ein, und versuchen Sie, es bereits beim Abtippen unter Zuhilfenahme der Variablentabelle zu verstehen.

```
5 GOSUB 60000: REM TABELLEN BESTIMMEN        <060>
10 REM ********* STEUERUNG ***********        <163>
30 GOSUB 50000                               <160>
40 REM REAGIEREN-AUFRUFEN                    <210>
60 REM ******* DEMONSTRATION *********       <082>
62 PRINT:PRINT"BE$ = ";BE$:PRINT             <143>
64 PRINT"VERBZAHL (VE) =";VE                 <122>
66 PRINT"1.GEGENSTANDSZAHL (G1) =";G1        <153>
68 PRINT"2.GEGENSTANDSZAHL (G2) =";G2        <158>
70 PRINT"OBJEKTZAHL (OB) =";OB               <252>
72 PRINT"PERSONENZAHL (PE) =";PE             <177>
80 GOTO 10                                   <058>
90 REM ******************************        <255>
50000 REM BEFEHL BE$ VOM SPIELER ERFRAGEN
      ------------------------------------
```

Listing 6.

```
      --                                          <241>
50010 IF UD=1 THEN 52000                          <194>
50020 INPUT"WAS NUN ";BE$                         <178>
50030 GOTO 51000                                  <176>
51000 REM BE$ IN BE$(1)-BE$(10) ZERLEGEN--
      ----------------------------------
      --                                          <235>
51010 FOR I=1 TO 10:BE$(I)="":NEXT                <236>
51020 WZ=1                                        <167>
51030 FOR I=1 TO LEN(BE$)                         <045>
51040 :IF MID$(BE$,I,1)=" "THEN GOSUB 5110
      0:GOTO 51060                                <100>
51045 :IF WZ>10 THEN PRINT"EINGABE IST ZU
      LANG !":I=LEN(BE$)+1:GOTO 51060             <147>
51050 :BE$(WZ)=BE$(WZ)+MID$(BE$,I,1)              <185>
51060 NEXT I                                      <006>
51070 GOTO 52000                                  <197>
51100 REM AU$ WOERTER AUSSORTIEREN                <120>
51110 IC=0:FOR I1=1 TO AZ                         <050>
51120 :IF BE$(WZ)=AU$(I1)THEN IC=1                <054>
51130 NEXT I1                                     <125>
51140 IF IC=0 THEN WZ=WZ+1:RETURN                 <226>
51150 BE$(WZ)="":RETURN                           <000>
52000 REM DIE BEFEHLSWOERTER UNTERSUCHEN U
      ND DARAUS EINEN SINN DEUTEN --------
      --                                          <118>
52005 IF UD=1 THEN UD=0:GOTO 52020                <213>
52010 WZ=1:VE=0:OB=0:PE=0                         <158>
52020 IC=0:G1=0:G2=0                              <150>
52025 REM *** VERB VE SUCHEN                      <161>
52030 FOR I=1 TO VZ                               <010>
52040 :IF BE$(WZ)=VE$(I)THEN VE=I:IC=1            <118>
52050 NEXT I:IF IC=1 THEN 52150                   <192>
52060 REM *** GEGENSTAND G1/G2 SUCHEN             <250>
52070 FOR I=1 TO GZ                               <035>
52080 :IF BE$(WZ)<>GE$(I)THEN 52090               <002>
52082 : IC=1                                      <230>
52084 : IF G1=0 THEN G1=I                         <120>
52086 : G2=I:IF G2=G1 THEN G2=0                   <089>
52090 NEXT I:IF IC=1 THEN 52150                   <232>
52100 REM *** OBJEKT OB SUCHEN                    <114>
52110 FOR I=1 TO OZ                               <083>
52120 :IF BE$(WZ)=OB$(I)THEN OB=I:IC=1            <178>
52130 NEXT I:IF IC=1 THEN 52150                   <016>
52132 REM *** PERSON PE SUCHEN                    <174>
52133 FOR I=1 TO PZ                               <107>
52134 :IF BE$(WZ)=PE$(I)THEN PE=I:IC=1            <200>
52135 NEXT I:IF IC=1 THEN 52150                   <021>
52137 IF BE$(WZ)="UND"THEN UD=1:IC=1              <085>
52140 IF IC=0 THEN PRINT"ICH KENNE ";BE$(W
      Z);" NICHT !":RETURN                        <255>
52150 WZ=WZ+1                                     <112>
52160 IF WZ>10 OR BE$(WZ)=""OR UD=1 THEN R
      ETURN                                       <141>
52170 IC=0:GOTO 52025                             <196>
60000 REM **********************                <155>
60001 REM * W O R T - T A B E L L E N *          <238>
60002 REM **********************                <157>
60010 REM *** AU$-WORT-TABELLE                    <186>
60020 DATA DER,DIE,DAS,DEN, ,UEBER,UNTER,A
      UF,IN,VON,VOM,IM,NACH,DURCH,MIT,DEM         <016>
60022 DATA AM                                     <113>
60030 AZ=17:DIM AU$(AZ):FOR I=1 TO AZ:READ
       AU$(I):NEXT                                <122>
60050 REM *** VERB-TABELLE VE$                    <157>
60060 DATA NIMM,VERLIERE,OEFFNE,GIB,SAGE,I
      NVENTUR,BEFESTIGE                           <078>
60070 VZ=7:DIM VE$(VZ):FOR I=1 TO VZ:READ
      VE$(I):NEXT                                 <186>
60090 REM *** GEGENSTANDS-TABELLE GE$             <186>
60100 DATA SCHWERT,SCHLUESSEL,SEIL,FACKEL,
      ARMBRUST,HELM,SCHILD                        <116>
60110 GZ=7:DIM GE$(7):FOR I=1 TO GZ:READ G
      E$(I):NEXT                                  <045>
60130 REM *** OBJEKT-TABELLE OB$                  <115
60140 DATA FENSTER,TUER,TRUHE,KISTE,FALLTU
      ER                                          <199
60150 OZ=5:DIM OB$(OZ):FOR I=1 TO OZ:READ
      OB$(I):NEXT                                 <223
60160 REM *** PERSONEN-TABELLE PE$                <065
60170 DATA GEIST,MONSTER,THORIN,GOMMEL,ORK        <077
60180 PZ=5:DIM PE$(PZ):FOR I=1 TO PZ:READ
      PE$(I):NEXT                                 <009
60200 RETURN                                      <161
```

Listing 6.

Dokumentation zu Listing 6:

5	Aufruf zum Definieren der Wort-Tabellen
10 bis 80	Steuerung des Adventures. Hier muß später auch das Actionmodul aufgerufen werden (vgl. auch letztes Grundschema).
50000	Anfang des Moduls
50000 bis 50030	Befehlseingabe
50010	Wenn UD=1 ist, der letzte Befehlsstring also mehrere Befehle enthielt, die noch nicht alle ausgeführt worden sind, so wird die Befehlseingabe übersprungen und der alte Befehlsstring wird weiterbehandelt.
51000 bis 51150	Zerlegung des Befehlssatzes BE$ in einzelne Worte BE$(1) bis BE$(maximal 10).
52000 bis 52170	Befehlscodierung (Lesevorgang)
52005	Ist UD=1, so steht fest, daß es sich um die Weiterbehandlung eines alten Befehlssatzes handelt. Deshalb darf der Wortzähler nicht auf 1 gesetzt werden, es muß bei dem Wort weitergelesen werden, bei dem schon vorher abgebrochen wurde. Damit Sätze wie »NIMM DAS SCHWERT UND DAS SEIL« verstanden werden, dürfen die Variablen VE, OB, PE ebenfalls **nicht** gelöscht werden.
52025 bis 52135	Hier werden alle Tabellen (Worttabellen) auf das aktuelle Wort BE$(WZ) hin untersucht. Wird das Wort in einer Verbtabelle gefunden, so erhält die Verbzahl VE ihren Wert etc. Wenn das Wort in irgendeiner Tabelle gefunden wird, so wird die Checkvariable IC auf 1 gesetzt. Nach jedem Tabellendurchlauf wird IC überprüft. Ist IC=1, so werden alle weiteren Tabellendurchläufe übersprungen, da ein Wort niemals in zwei Tabellen vertreten ist. Dieses Überspringen dient lediglich dazu, die Verarbeitungsgeschwindigkeit des Moduls zu erhöhen.
52137	Überprüfung, ob BE$(WZ) = "UND" ist. Wenn ja, so wird UD gleich 1 gesetzt.
52140	Wenn IC nun immer noch 0 ist, so steht fest, daß das Wort BE$(WZ) in keiner Tabelle beziehungsweise dem gesamten Wortschatz vertreten ist. Es erfolgt somit eine Fehlermeldung, die ausdrückt, welches Wort nicht erkannt wurde.
52150	Die Wortzählvariable wird auf das nächste Wort gesetzt.
52160	Überprüfung, ob der Befehlssatz die zulässige Wortzahl von maximal 10 Worten überschreitet, ob das neue Wort BE$(WZ) = "" ist oder ob UD = 1 ist. Trifft eine dieser Bedingungen zu, so wird der Codiervorgang a) beendet, wenn der Befehlssatz zuende ist (BE$(WZ)=""). b) unterbrochen, wenn ein Teilbefehl des Befehlssatzes BE$ ausgeführt werden muß, also wenn UD=1 ist.
52170	Die Check-Variable IC wird auf 0 gesetzt. Der Lesevorgang beginnt erneut.
Ab 60000	Definieren der Worttabellen

Bevor Sie nun im Kurs fortfahren, sollten Sie unser neues Modul bis ins letzte Detail verstanden haben.

Wenn Sie das Modul abgetippt haben, so starten Sie es nun bitte. Es erfolgt der Aufruf, einen Befehlssatz einzugeben »WAS NUN?«. Sie können nun einen Befehl eingeben, der aus Worten besteht, die im Wortschatz enthalten sind (also in Tabellen).

Wenn Sie anschließend RETURN drücken, so gibt das Programm die Zahlen aus, die aus der Codierung hervorgehen:

Verbzahl VE
1. Gegenstandszahl G1
2. Gegenstandszahl G2
Objektzahl OB
Personenzahl PE

Geben Sie nun einmal verschiedene Befehlssätze (auch Sätze die mehrere Befehle enthalten) ein, und untersuchen Sie die jeweiligen Ergebnisse.

Sie können auch einmal die Beispielssätze, die wir bereits theoretisch codiert haben, eingeben und die Ergebnisse mit den theoretisch gewonnenen vergleichen. Ergibt der Vergleich keine Unterschiede, so arbeitet unser Modul tadellos. Wenn nicht, so muß irgendwo ein Tippfehler vorliegen, den Sie anhand Ihres bisherigen Wissens jedoch leicht finden müßten. Ich gebe zu, daß es momentan etwas trocken und monoton zugeht, aber ich kann Ihnen getrost versichern, daß wir uns schon bald interessanteren Bereichen der Programmierung zuwenden werden.

Bevor ich jedoch das Geheimnis lüfte, was wir eigentlich mit den ganzen Verbzahlen, Objektzahlen etc. anfangen werden, müssen wir uns noch einem kleinen Problem zuwenden.

Betrachten Sie einmal die folgenden beiden Sätze:

1. VERLIER DAS SEIL.
2. VERLIERE DAS SEIL.

Für uns Deutsche ist hier zunächst eigentlich kein Unterschied zu sehen. Tatsache ist jedoch, daß das Verb im ersten Satz VERLIER und im zweiten Satz VERLIERE lautet. Trotzdem ist der Sinn des Satzes eindeutig. Der Befehlssinn des ersten Satzes ist also exakt der gleiche, wie der des zweiten, eigentlich korrekten Satzes. Doch wie kann man das dem Computer klarmachen?

Nun, dazu wollen wir die beiden Worte VERLIER und VERLIERE noch einmal näher betrachten. Genausogut könnten wir für diese Betrachtung auch das Beispiel SAG und SAGE hernehmen.

Das Problem ist einfach zu lösen.

Bisher lief die Verb-Untersuchung im Programm (Zeile 52025 bis 52050) ja so, daß wir überprüft haben, ob das aktuelle Befehlswort BE$(WZ) in der Verbtabelle enthalten ist. Dies sah so aus:

```
IF BE$(WZ) = VE$(I) THEN VE=I : IC = 1
```

Die Variable I durchläuft dabei die Werte 1 bis VZ, wobei VZ die Anzahl aller Verben darstellt.

Wir müssen diese Abfrage nun so umändern, daß nicht mehr überprüft wird, ob BE$(WZ) mit VE$(I) identisch ist, sondern nur ob BE$(WZ) im Verb VE$(I) enthalten ist.

Wie man überprüft, ob ein String in einem anderen String enthalten ist, haben Sie bereits im Kapitel Stringoperationen gelernt (lesen Sie dort gegebenenfalls nochmal nach).

Dies sieht zwar ziemlich kompliziert aus, ist bei näherem Betrachten jedoch ganz einfach.

Schreiben Sie jetzt im Programm die Zeile 52040, in der die Abfrage stattfindet, um:

```
52040 :IFBE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ)))
        THENVE=I:IC=1
```

Starten Sie nun das Programm erneut und überprüfen Sie, ob es ordungsgemäß abläuft.

Ist dies der Fall, so müßte für VERLIER und VERLIERE eine identische Verbzahl und keine Fehlermeldung herauskommen.

Wenn Sie einfach V eingeben, so erhalten Sie ebenfalls die Verbzahl, die Sie auch für VERLIERE erhalten. Jeder Verb-Befehl kann also jetzt beliebig abgekürzt werden.

Allerdings kann es bei Abkürzungen auch schnell zu Mißverständnissen kommen. Wenn in der Verbtabelle zum Beispiel zwei Verben enthalten sind, die mit den gleichen Buchstaben beginnen (zum Beispiel SAGE und SUCHE) und der Spieler als Befehlsabkürzung S eingibt, so kann das Programm nicht wissen ob SAGE oder SUCHE mit dieser Abkürzung gemeint ist. Es nimmt in einem solchen Fall immer die Verbzahl an, die die Größte ist. Diese Tatsache braucht uns jedoch nicht zu stören, da der Spieler schnell herausfinden wird, welche Befehle er wie abkürzen kann.

Das Befehlsanalyseprogramm, das uns nun zur Verfügung steht, ist ohne Zweifel schon sehr gut, aber wir wollen noch einen Schritt weitergehen.

Hierzu wollen wir einmal die folgenden beiden Verben betrachten: NIMM und NEHME.

Im Prinzip besteht zwischen den Befehlssätzen NIMM DAS SCHWERT und NEHME DAS SCHWERT kein Unterschied. Trotzdem gibt es kaum deutsche Adventures, die beide Worte akzeptieren. Bei unserem Befehlsanalysesystem soll dieses Problem jedoch endgültig gelöst werden.

Bevor wir uns jedoch mit der Lösung beschäftigen, wollen wir noch ein anderes Beispiel betrachten.

Nehmen wir einmal an, unser Spieler hat ein Brett und ein Seil. Er will das Seil am Brett festbinden.

Er könnte folgende Befehlssätze ausprobieren:

1. Binde das Seil am Brett fest.
2. Befestige das Seil am Brett.
3. Verknote das Seil am Brett.
4. Verbinde das Seil mit dem Brett.
5. Mache das Seil am Brett fest.
6. Binde das Seil ans Brett.

Zugegeben, einige Satzmöglichkeiten sind vielleicht extrem außergewöhnliche Beispiele, aber wir wollen einmal damit rechnen.

Diese Sätze sind auch gute Beispiele dafür, warum wir zwei Gegenstandsvariablen G1 und G2 für die Befehlsanalyse reserviert haben, denn in jedem der sechs Sätze kommen zwei Gegenstände vor: Seil und Brett.

Die Gegenstände, die in den Sätzen auftauchen, sind also immer identisch. Nun wollen wir einmal betrachten, welche Wörter Verben VE$ sind, und welche aussortiert werden müssen (AU$).

1. VE$: BINDE — AU$: das, am, fest
2. VE$: BEFESTIGE — AU$: das, am
3. VE$: VERKNOTE — AU$: das, am
4. VE$: VERBINDE — AU$: das, mit, dem
5. VE$: MACHE — AU$: das, am, fest
6. VE$: BINDE — AU$: das, ans

Wir haben jetzt also mehrere Verben, die alle den gleichen Sinn haben (in bezug auf unseren Satz) — eine Verbfamilie also.

Den Satz Nummer 5 wollen wir verbieten.

In ihm ist kein eindeutiges Verb vorhanden. »Mache« kann als Verb betrachtet werden, da zu »mache« immer noch ein Bezugswort gehört (zum Beispiel mache fest, wobei fest das Bezugswort ist). Das Wort »fest« kann aber wiederum nicht als Verbersatz dienen, da es in Satz 1 als ein auszusortierendes Wort betrachtet wird. Wir wollen Sätze mit »mache« also grundsätzlich verbieten. Immer wenn der Spieler also einen Befehlssatz eingibt, der das Wort »mache« enthält, wird er die Fehlermeldung »ICH KENNE DAS WORT MACHE NICHT« vom Programm erhalten.

Natürlich brauchen wir diese Fehlermeldung für das Wort »mache« nicht extra programmieren. Unser Befehlsanalyseprogramm bringt nämlich für jedes Wort, das nicht in einer der Worttabellen vertreten ist (einschließlich der Tabelle der

auszusortierenden Worte) automatisch eine Fehlermeldung. Unser Problem, das wir lösen wollen, ist also, daß die Befehlsanalyse sich nicht nur auf einzelne Verben, sondern auch auf ganze Verbfamilien bezieht. Die Verben NIMM und NEHME sind auch eine solche Verbfamilie.

Eine einfache Lösung des Problems wäre die folgende:

Wir erweitern die Verbtabelle VE$ einfach um das Verb »Nehme« (für das Beispiel der Wortfamilie Nimm, Nehme).

Das Verb »NEHME« bekäme in diesem Fall eine eigene Verbzahl zugeordnet. Die hätte zur Folge, daß die Verbzahlen für »NIMM« und »NEHME« unterschiedlich wären.

Diese Tatsache müßte man dann später im Actionmodul berücksichtigen. Wenn das Actionmodul später also feststellen will, ob der Spieler etwas nehmen will, dann würde die Abfrage hierzu folgendermaßen aussehen:

IF VE = A OR VE = B THEN:REM: SPIELER WILL ETWAS NEHMEN.

A ist hierbei die Verbzahl für das Wort »NIMM« und B die Verbzahl für das Wort »NEHME«.

Diese Lösung würde für unser erstes Beispiel zwar gut funktionieren, aber beim 2. Beispiel (Spieler will Seil mit Brett verbinden) wäre schon eine ziemlich lange IF-THEN-Abfrage nötig. Die erste Lösung frißt also zuviel vom eh' sehr kostbaren RAM-Speicherplatz.

Eine bessere Lösung wäre, wenn man für das Verb NIMM und das Verb NEHME die gleiche Verbzahl erhalten würde. Bei diesem Idealfall wäre die im Actionmodul nötige Abfrage schon wesentlich unkomplizierter:

IF VE = A THEN:REM: SPIELER WILL ETWAS NEHMEN.

Wir sind uns also einig, daß eine solche Lösung die Ideallösung ist. Wie programmiert man diese Ideallösung jedoch am einfachsten?

Eine Möglichkeit wäre, mehrere Alternativ-Verbtabellen zu erstellen. In der normalen Verbtabelle stände dann an erster Stelle das Verb NIMM und in der alternativen Verbtabelle das Alternativ-Verb zu NIMM, also NEHME. Wenn wir jedoch wieder unsere zweite Wortbeispieltabelle betrachten, bemerken wir, daß wir hier mehrere alternative Worttabellen benötigen würden, da unsere zweite Verbfamilie aus vier Mitgliedern (binde, befestige, verknote, verbinde) besteht. Diese Lösung wäre viel zu kompliziert zu realisieren und bräuchte auch zu viel Speicherplatz. Ich will Sie nun nicht länger auf die Folter spannen, und die Katze aus dem Sack lassen — die Ideallösung!

Und so sieht sie aus:

Das Wort beziehungsweise Verb »NIMM« steht ja bereits in unserer Verbtabelle in der DATA-Zeile 60060.

```
60060 DATA NIMM, VERLIERE, OEFFNE, GIB, SAGE,
      INVENTUR, BEFESTIGE
60060 DATA NIMM, NEHME1, VERLIERE, OEFFNE, GIB,
      SAGE, INVENTUR, BEFESTIGE
```

Wir haben das Verb »NEHME« also direkt hinter dem Verb »NIMM« eingefügt. Jedoch haben wir nicht einfach »NEHME« sondern »NEHME1« eingefügt — der Einser direkt nach dem Wort »NEHME« ist **kein** Druckfehler.

Was soll denn der Einser, werden Sie jetzt sicherlich mit Recht fragen. Stören tut er ja nicht, denn wenn wir das Programm starten und das Wort »Nehme« eingeben, so wird dies ja akzeptiert. Der Computer betrachtet »nehme« einfach als eine Abkürzung für das in seiner Verbtabelle an zweiter Stelle gespeicherte Verb »NEHME1«.

Oh, da sehe ich doch, daß mir soeben ein kleiner Fehler unterlaufen ist. Haben Sie ihn auch schon entdeckt? Nun, ich habe vergessen die VZ-Variable in Zeile 60070 umzuändern.

Denn da ich ein neues Verb in die Tabelle eingebaut habe, muß ich auch die VZ-Variable umändern, da VZ für die Anzahl der Verben in der Verbtabelle steht. Diese Änderung wollen wir jetzt schnell nachholen:

```
60070 VZ=8...
```

Solche Fehler sind sehr tückisch, da sie meist nur selten im Programm wirksam werden und daher nur schwer zu finden sind. Denken Sie also stets daran, daß Sie immer, wenn Sie irgendeiner Worttabelle eine weiteres Wort zufügen, auch die entsprechende Variable, die angibt wieviele Worte in dieser Tabelle stehen, korrigieren müssen.

Nun aber zurück zur Frage »Warum NEHME1?«.

Voraussetzung für das Funktionieren dieser Ergänzung ist natürlich, daß Sie Zeile 52040 bereits geändert haben (wie vor kurzem beschrieben). Können Sie sich noch an den VAL-Befehl erinnern, den wir beim Kapitel Stringoperationen besprochen haben?

Nach der Änderung der Verbtabelle sieht diese so aus:

```
VE$(1) = "NIMM"
VE$(2) = "NEHME1"
VE$(3)... etc.
```

Probieren Sie doch einmal aus, was VAL (VE$(2)) ergibt.

Geben Sie dazu im Direktmodus ? VAL ("NEHME") ein.

Nun, was glauben Sie was passiert, wenn man nach dieser Eingabe RETURN drückt?

Die richtige Antwort lautet Null, was Sie ja leicht überprüfen können.

Fall Sie als Antwort 1 erwartet haben, so beherrschen Sie den VAL-Befehl noch nicht. Geben Sie nun die folgende Zeile im Direktmodus ein:

```
? VAL(RIGHT$("NEHME1",1))
```

Nach Drücken von RETURN erhalten wir endlich das gewünschte Ergebnis. 1. Wenn wir für »NEHME1« den String »NIMM« einsetzen, erhalten wir als Ergebnis 0. Es ist also egal, ob wir »NIMM« oder »NIMM0« einsetzen.

Wir wollen also einmal folgendes festlegen:

Das Stammverb steht immer vor den alternativen Verben in der Tabelle. Die alternativen Verben folgen in der Tabelle direkt nach dem Stammverb und sind mit Zahlen versehen.

Betrachten wir nun nochmals die Verbfamilie BINDE, BEFESTIGE, VERKNOTE, VERBINDE.

In eine Verbtabelle eingebaut, würde diese Worttabelle so aussehen (wobei es immer egal ist, welches Mitglied der Familie als Stammverb gewählt wird):

```
VE$(1) = "BINDE"
VE$(2) = "BEFESTIGE1"
VE$(3) = "VERKNOTE2"
VE$(4) = "VERBINDE3"
```

Bei der Numerierung habe ich unsere bisherige Verbtabelle völlig außer Acht gelassen.

Wie ich bereits gesagt habe, besteht unser Ziel darin, für jedes Mitglied einer Verbfamilie die gleiche Verbzahl VE zu erhalten.

Ich lege nun fest, daß die Verbzahl, die man später für jedes Verb der Verbfamilie erhält, identisch ist mit der Verbzahl, die man für das Stammverb dieser Verbtabelle erhält.

Sicher wissen Sie noch, wie wir unsere Verbzahl bisher erhalten haben: Wir haben eine Schleife I, die VE$-Tabelle von 1 bis VZ (Anzahl der Verben) durchlaufen lassen. Dabei haben wir überprüft, ob das aktuelle Befehlswort BE$(WZ) gleich VE$(I) war. Stimmten beide Strings überein, so sagten wir einfach VE = I. Also Verbzahl = momentaner Stand der Schleife.

Im Programm sah dies so aus (in der ersten Version):

```
52030 FORI = 1TO VZ
52040 :IFBE$(WZ) = VE$(I) THEN VE=I
52050 NEXT I
```

Dieses Programm habe ich soweit wie möglich vereinfacht.

Wenn wir bei diesem Prgramm unsere obige Verbtabelle durchlaufen lassen, so erhalten wir im Moment noch für jedes Verb eine spezielle Verbzahl. Wie stellen wir es nun an, daß wir für jedes Wort einer Verbfamilie die gleiche Verbzahl VE erhalten?

Bisher war unsere Verbzahl VE nur vom jeweiligen Stand der Schleifenvariable I abhängig, bei dem sich BE$(WZ) mit VE$(I) als identisch erwies. Nun müssen wir einen Weg finden, bei dem sich VE in Abhängigkeit von der Schleifenvariablen I und von dem VAL-Wert des jeweiligen Verbs VES(I) ergibt.

Dazu wollen wir nochmals unsere neue Verbtabelle betrachten:

Für das Stammverb »BINDE« erhalten wir den VAL-Wert 0.

Für das erste Alternativverb »BEFESTIGE« ist der VAL-Wert gleich 1.

Für das zweite Alternativverb entsprechend 2.

Für das dritte Alternativverb schließlich 3.

Die Ziffer im String gibt also an, das wievielte Alternativverb eines Stammverbs der String ist.

Es wurde ja bereits festgelegt, daß die Verbzahl für alle Mitglieder einer Verbfamilie identisch sein soll mit dem Wert des Stammverbs dieser Familie, also dem Verb mit dem VAL-Wert 0.

Damit dürfte die Lösung des Problems eigentlich schon klar sein:

```
52030 FOR I = 1 TO VZ
52040 :IFBE$(WZ) = VE$(I) THEN VE = I-VAL
      (RIGHT$(VE$(I),1))
52050 NEXTI
```

Wir ziehen von dem I-Wert, bei dem die Identität festgestellt wurde, einfach den VAL-Wert ab. Wenn wir vom I-Wert, der sich für das Stammverb ergibt, den VAL-Wert des Stammverbs abziehen, so erhalten wir den Verbzahl-Wert VE des Stammverbs (da der VAL-Wert des Stammverbs ja immer 0 ist).

Wenn wir vom I-Wert, der sich für das erste Alternativverb ergibt, den VAL-Wert des ersten Alternativverbs abziehen, so erhalten wir wieder die selbe Verbzahl VE (da der VAL-Wert des ersten Stammverbs ja immer 1 ist). Für die weiteren Alternativverben gilt natürlich das Entsprechende.

Wir haben unsere angestrebte Ideallösung also gefunden. Und das Besondere ist, daß unsere neue Befehlsanalyse nur um ein paar Bytes länger geworden ist.

Wir dürfen jedoch nicht vergessen, die Zeile 52040 wieder so umzuändern, daß der Vergleich zwischen BE$(WZ) und VE$(I) wieder nur so erfolgt, daß überprüft wird, ob BE$(WZ) in VE$(I) enthalten ist. Denn nur dann werden Abkürzungen für Verben akzeptiert, und dies ist auch die Grundbedingung, die es uns erlaubt, die Strings der Alternativverben mit Nummern zu versehen.

Die endgültige Zeile 52040 lautet folglich:

```
52040 :IFBE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ)))
      THENVE=I-VAL(RIGHT$(VE$(I),1)) :IC=1
```

Damit haben wir es endlich geschafft: Unser Analysemodul ist fertig. Anschließend noch ein Listing, bei dem eine komplette Befehlseingabe mit unserer idealen Befehlsanalyse zu einem Unterprogramm, das ab Zeile 50000 steht, zusammengefaßt ist — das Programm ist soweit wie möglich komprimiert. Also ohne viele REM-Zeilen etc.

Es ist die Grundbasis, für alle folgenden Beispielprogramme. Dieses Modul kann für jedes Adventure verwendet werden, da es nichts programmspezifisches enthält. Änderungen können Sie leicht vornehmen, da das Modul jetzt ja ausführlich besprochen worden ist.

Das Modul kann natürlich nur dann arbeiten, wenn vorher im Programm die Worttabellen mit den entsprechenden Variablen definiert werden.

Zu beachten ist hierbei, daß die Verbtabellen immer die längstmöglichen Wörter enthalten. Also »VERLIERE« und nicht »VERLIER«, denn nur dann ist gewährleistet, daß Verben abgekürzt werden können.

Man könnte im Prinzip auch für die anderen Worttabellen, wie die Objekttabelle, Abkürzungen erlauben. Ich halte dies jedoch für überflüssig. Geben Sie nun bitte das Listing 7 ein und speichern Sie es auf Kassette oder Diskette ab, damit Sie in den folgenden Kapiteln darauf zurückgreifen können.

```
10 VZ=2:VE$(1)="N":VE$(2)="NW"                          <116>
50000 REM ***************************                  <058>
50010 REM *      BEFEHLSEINGABE       *                 <228>
50020 REM *      BEFEHLSZERLEGUNG     *                 <182>
50030 REM *      BEFEHLSCODIERUNG     *                 <173>
50040 REM ***************************                  <098>
50050 IF UD=1 THEN 50220                                <236>
50060 POKE 198,0:BE$="":PRINT"WAS NUN ? ";              <044>
50070 POKE 204,0                                        <226>
50080 GET X$:IF X$=""THEN 50080                         <091>
50090 IF PEEK(203)=1 OR LEN(BE$)>68 THEN P
      RINT" ":POKE 204,1:GOTO 50140                     <022>
50100 I=ASC(X$):IF I<65 OR I>90 THEN IF I<
      >32 AND I<>20 AND I<>34 THEN 50080                <173>
50110 IF I=20 AND BE$=""THEN 50080                      <093>
50120 IF I=20 THEN POKE 204,1:PRINT"{LEFT,
      2SPACE,2LEFT}";:BE$=LEFT$(BE$,LEN(BE
      $)-1):GOTO 50070                                  <174>
50130 PRINT X$;:BE$=BE$+X$:GOTO 50080                   <013>
50140 FOR I=1 TO 10:BE$(I)="":NEXT:WZ=1:FO
      R I=1 TO LEN(BE$)                                 <154>
50150 :IF MID$(BE$,I,1)=" "THEN GOSUB 5019
      0:GOTO 50180                                      <240>
50160 :IF WZ>10 THEN PRINT"EINGABE IST ZU
      LANG !":I=LEN(BE$)+1:GOTO 50180                   <028>
50170 :BE$(WZ)=BE$(WZ)+MID$(BE$,I,1)                    <069>
50180 NEXT I:GOTO 50220                                 <079>
50190 IC=0:FOR I1=1 TO AZ:IF BE$(WZ)=AU$(I
      1)THEN IC=1                                       <086>
50200 NEXT I1:IF IC=0 THEN WZ=WZ+1:RETURN               <105>
50210 BE$(WZ)="":RETURN                                 <081>
50220 IF UD=1 THEN UD=0:GOTO 50240                      <215>
50230 WZ=1:VE=0:OB=0:PE=0                               <163>
50240 IC=0:G1=0:G2=0                                    <155>
50250 FOR I=1 TO VZ:IF BE$(WZ)=VE$(I)THEN
      VE=I:IC=1                                         <114>
50251 IF LEN(BE$(WZ))<3 THEN 50260                      <229>
50255 IF BE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ))
      )THEN VE=I-VAL(RIGHT$(VE$(I),1)):IC=
      1                                                 <211>
50260 NEXT I:IF IC=1 THEN 50350                         <187>
50270 FOR I=1 TO GZ:IF BE$(WZ)<>GE$(I)THEN
       50300                                            <211>
50280 IC=1:IF G1=0 THEN G1=I                            <212>
50290 G2=I:IF G2=G1 THEN G2=0                           <020>
50300 NEXT I:IF IC=1 THEN 50350                         <227>
50310 FOR I=1 TO OZ:IF BE$(WZ)=OB$(I)THEN
      OB=I:IC=1                                         <147>
50320 NEXT I:IF IC=1 THEN 50350                         <247>
50330 FOR I=1 TO PZ:IF BE$(WZ)=PE$(I)THEN
      PE=I:IC=1                                         <176>
50340 NEXT I                                            <051>
50350 IF BE$(WZ)="UND"THEN UD=1:IC=1                    <083>
50360 IF IC=0 THEN PRINT"ICH KENNE ";BE$(W
      Z);" NICHT !":RETURN                              <004>
50370 WZ=WZ+1:IF WZ>10 OR BE$(WZ)=""OR UD=
      1 THEN RETURN                                     <177>
50380 IC=0:GOTO 50250                                   <189>
```

Listing 7

Bitte betrachten Sie einmal die folgenden Zeilen, die auch im Listing enthalten sind:

```
50250 FORI=1TOVZ:IFBE$(WZ)=VE$(I)THENVE=
      I:IC=1
50251 IFLEN(BE$(WZ))<3THEN50260
```

Die beiden IF-THEN-Abfragen sorgen dafür, daß die Abkürzungen für die Verben mindestens drei Buchstaben haben müssen.

Verben mit ein bis zwei Buchstaben werden jedoch akzeptiert, wenn sie in der Verbtabelle enthalten sind (zum Beispiel Richtungsangaben wie N, S, O, W, NW etc.).

5. Kapitel: Die Spielkarte

Als Resultat des letzten Kapitels steht uns nun ein komfortables Befehlsanalyse-System zur Verfügung.

Es ist keine Übertreibung, zu behaupten, daß wir dem kompletten Adventure nun schon sehr nahe sind. Da im letzten Abschnitt sehr viel Theorie enthalten war, die sich auch schon ein wenig auf den gesamten Ablauf eines Adventures bezog, haben wir die wesentliche Gedankenarbeit bereits geleistet — Sie haben gelernt Tabellen zu definieren und mit Strings zu arbeiten.

Für die folgenden Kapitel werde ich die Theorie deshalb etwas knapper halten. Nun aber gleich zum Thema — der Spielkarte.

Eine Spielkarte zu programmieren, ist das Problem, das jeden Adventure-Programmierer am meisten beschäftigt.

Wer sich ein Adventure-Listing aus einer Computerzeitschrift schon einmal näher angeschaut hat, wird wissen, wie verwirrend die Listings oft aussehen: Schier unendlich viele Variablen und GOTO/GOSUB-Anweisungen leisten ihren Beitrag zur Unübersichtlichtkeit, so daß es nahezu unmöglich ist, den Programmablauf vernünftig zu verfolgen. Eine weitere Tatsache ist, daß in keinem anderen Listing so viele Fehler auftauchen, wie bei Abenteuerspielen. Das liegt hauptsächlich daran, daß sich die meisten Adventure-Programmierer nach einiger Zeit in ihrem Listing nicht mehr zurecht finden, da sie zu »chaotisch« programmiert haben.

Unser System zur Programmierung der Spielkarte soll folgende Eigenschaften haben:

1. Sehr gute Übersichtlichkeit — Fehler können leicht behoben werden.
2. Möglichst geringer Speicherplatzverbrauch — denn besser das System zur Kartenprogrammierung ist, desto mehr Räume können untergebracht werden.
3. Es muß nur eine Variable verändert werden, um das Spiel an jeder beliebigen Stelle (Raum) starten zu lassen — wichtig zum späteren Testen des Spiels.

Im Prinzip gibt es verschiedene Grundmöglichkeiten. Alle haben jedoch ihre Mängel. Nach längerem Ausprobieren der einzelnen Möglichkeiten hat sich bei mir die Technik, die ich Ihnen nun vorstellen werde, bis jetzt tadellos bewährt.

Mit dem herkömmlichen Basic des C 64 ist diese Technik jedoch nicht realisierbar. Unser erster Schritt soll deshalb darin bestehen, das bescheidene Basic des C 64 um drei entscheidende Befehle zu erweitern, beziehungsweise drei bereits vorhandene Befehle zu modifizieren.

Bitte geben Sie das folgende Programm ein und speichern Sie es anschließend ab. Noch besser ist es, wenn Sie zuerst das Befehlseingabemodul, das Sie hoffentlich bereits abgetippt haben, laden, und es um das folgende Listing 8 ergänzen.

```
10 DATA 76,24,1,177,251,145,251,200,208,24
   9,230,252,202,208,244,96,120,160,0,169  <104>
12 DATA 160,132,251,133,252,162,32,32,11,1
   ,169,224,132,251,133,252,162,32,32,11   <025>
14 DATA 1,169,53,133,1,88,96:FOR I=264 TO
   310:READ X:POKE I,X:NEXT:SYS 264        <234>
16 FOR I=710 TO 730:READ X:POKE I,X:NEXT   <213>
18 DATA 208,3,76,29,168,32,192,2,32,19,166
   ,56,165,95,233,1,164,96,76,36,168       <165>
20 POKE 40996,197:POKE 40997,2:POKE 1,54   <087>
22 FOR I=43168 TO 43170:READ X:POKE I,X:NE
   XT                                      <174>
24 FOR I=704 TO 709:READ X:POKE I,X:NEXT   <230>
26 DATA 32,192,2,32,138,173,76,247,183     <114>
```

Listing 8

Nachdem das Programm gestartet wurde, stehen uns folgende neue, beziehungsweise erweitere Befehle zur Verfügung:
GOTO X, GOSUB X, RESTORE X.

Diese drei Befehle sind also so restauriert worden, daß sie auch in Abhängigkeit von einer Variablen und sogar von einem Formelausdruck funktionieren. Der ON GOTO- und der ON GOSUB-Befehl kann nun getrost auf den »Müll« geworfen werden.

Folgende Programmierung ist nun möglich:

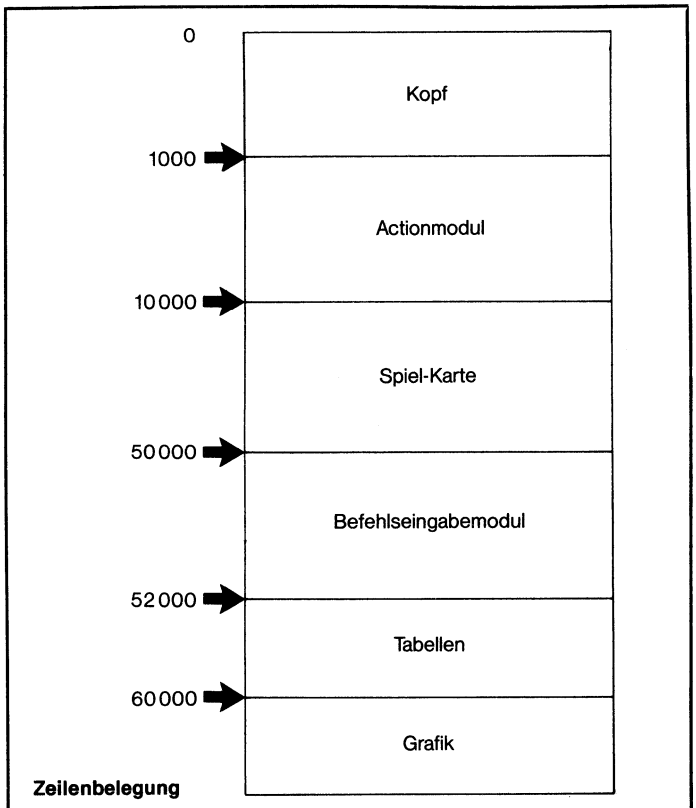
```
      10  rem demoprogramm
      20  X = 100
      30  GOTO X
      100 PRINT"SPITZE !"
      110 END

oder: 10  rem demoprogramm
      20  GOSUB 90+10
      30  END
      100 PRINT"SPITZE !" : RETURN

sowie: 10  rem demoprogramm
       20  INPUT"WERT"; X
       30  IF X<1 OR X>3 THEN GOTO 20
       40  RESTORE 100*3 : READX$ : PRINT X$ :
           GOTO 20
       100 DATA HALLO
       200 DATA WIE
       300 DATA GEHTS
```

Wer sich das letzte Beispielprogramm genau anschaut und nachdenkt, dem kommt vielleicht jetzt schon eine Idee, wie die Kartenprogrammierung funktionieren könnte.

Bevor wir uns nun mit der Kartenprogrammierung beschäftigen, müssen wir den folgenden Aspekt berücksichtigen:

Durch die drei neuen Befehle bietet sich eine vollkommen neue Programmiermöglichkeit, bei der beachtet werden muß beziehungsweise sollte, wie man die einzelnen Programmzeilen 0 bis 63999 belegt.

Betrachten Sie bitte einmal die folgende Skizze:



Zeilenbelegung

Damit wäre die Spielkarte nun auch schon zum ersten Teil programmiert.

Raum 1 steckt in den Zeilen 10100 bis 10120

Raum 2 steckt in den Zeilen 10200 bis 10220

entsprechend die anderen Räume.

Dieses Programm ist jedoch noch nicht lauffähig, da jegliche Steuerung fehlt. Die Steuerung ist Aufgabe des Actionmoduls, das in den Zeilen 1000 bis maximal 9999 steht (wie der letzten Skizze zu entnehmen).

Wir wollen zunächst eine primitive Steuerung simulieren. Geben Sie dazu bitte folgende weitere Zeilen ein:

```
1000 REM A C T I O N M O D U L
1010 INPUT"IN WELCHEN RAUM WOLLEN SIE
GEHEN?":ZN
1015 IF ZN<1 OR ZN>6 THENPRINT"DAS GEHT
NICHT":GOTO1010
1020 GOSUB 10000+ZN*100
1030 GOTO 1000
```

Nun ist das Programm lauffähig.

Wie auch bei den meisten folgenden Listings erübrigt sich hier eine Dokumentation. Interessant ist lediglich Zeile 1020.

Dort steht die Formel, mit der dann mittels GOSUB der aktuelle Raum erreicht werden kann.

Das vorliegende Actionmodul ist selbstverständlich noch extrem einfach. Im Moment kann man noch beliebig jeden Raum direkt anlaufen — falls man überhaupt von »Laufen« reden kann.

Löschen Sie bitte nun folgende Zeilen wieder: 1010, 1015, 1020 und 1030. Wichitg ist nur, daß Sie sich die Formel zur Errechnung des jeweiligen Raumes merken:

Raum = 10000+ZN*100.

ZN ist die jeweilige Zimmernummer, identisch mit den Nummern auf dem gezeichneten Spielplan. Der Raum beziehungsweise die Formel bestimmt die Zeilennummer, ab der der Raum im Programm steht.

Es wird nun höchste Zeit, daß wir unser Befehlseingabemodul in Bewegung setzen.

Das Actionmodul muß so geändert beziehungsweise gestaltet werden, daß man nicht mehr Räume direkt ansteuern, sondern nur noch von Raum zu Raum laufen kann.

Zunächst ein wenig Theorie:

Wir wollen dem Spieler unseres Adventures erlauben, direkte Himmelsrichtungen einzugeben, wenn er sich bewegen will.

Wenn er zum Beispiel nach Norden gehen will, so muß er einfach »N« eingeben.

Manche Adventures hingegen verlangen Eingaben wie »GEHE NORD«. Dies ist meines Erachtens nach jedoch unnötig, da es dem Spieler schnell auf die Nerven fallen kann, denn es ist wesentlich einfacher nur ein bis zwei Buchstaben zum Laufen einzugeben und nicht immer zwei Worte.

Vielleicht können Sie sich noch daran erinnern, daß ich bereits im Kapitel zur Programmierung unseres ersten Moduls festgelegt habe, daß Himmelsrichtungen wie gewöhnliche Verben behandelt werden können.

Deshalb wollen wir die zehn möglichen Himmelsrichtungen (inklusiv rauf und runter) nun in unserer Verbtabelle aufnehmen — beziehungsweise mit dem Programmieren der einzelnen Worttabellen beginnen.

Geben Sie bitte folgende Zeilen zum bisherigen Programm ein:

```
52000 REM T A B E L L E N
52005 RESTORE 52000
52010 REM VERBTABELLE ————————
52020 DATA N,S,W,O,NW,NO,SW,SO,RAUF,RUNTER
52100 VZ=10:DIMVE$(VZ):FOR I=1TOVZ : READ VE$(I)
: NEXT
53000 RETURN
```

Bitte legen Sie sich nun ein Blatt Papier an, auf dem Sie die einzelnen Tabellen, die bald kommen, ausführlich notieren — also den jeweiligen Wert zu jedem einzelnen Verb, Objekt, Gegenstand etc. ...

Nun müssen wir in den Kopf noch den Aufruf zum Unterprogramm der Tabellen einbauen. Dazu folgende Zeile eingeben:

30 GOSUB 52000 : REM TABELLEN DEFINIEREN

Wir erhalten somit also Verbzahlen VE für die einzelnen Himmelsrichtungen. Bevor wir jedoch die GEH-Routine im Actionmodul einbauen können, müssen wir die Spielkarte noch verbessern.

Bisher bestand die Spielkarte nur aus knappen Texten. Nun müssen noch Informationen eingebaut werden, die angeben, wie die Räume miteinander verbunden sind. Dies geht relativ einfach:

Geben Sie bitte wieder die Ergänzungszeilen aus Listing 9 ein:

```
0 REM ****************************          <081>
1 REM * ADVENTURE-PROGRAMMIERKURS *          <073>
2 REM *                          *          <229>
3 REM *     UEBUNGS-PROGRAMM     *          <145>
4 REM ****************************          <085>
9 REM BASIC-ERWEITERUNG                      <120>
10000 REM ------- SPIELKARTE -------          <048>
10100 REM RAUM 1 ---------                   <037>
10105 PRINT"RAUM NUMMER 1"                   <183>
10120 RETURN                                 <061>
10200 REM RAUM 2 ---------                   <138>
10205 PRINT"RAUM NUMMER 2"                   <028>
10220 RETURN                                 <161>
10300 REM RAUM 3 ---------                   <240>
10305 PRINT"RAUM NUMMER 3"                   <130>
10320 RETURN                                 <006>
10400 REM RAUM 4 ---------                   <085>
10405 PRINT"RAUM NUMMER 4"                   <231>
10420 RETURN                                 <106>
10500 REM RAUM 5 ---------                   <187>
10505 PRINT"RAUM NUMMER 5"                   <077>
10520 RETURN                                 <207>
10600 REM RAUM 6 ---------                   <032>
10605 PRINT"RAUM NUMMER 6"                   <178>
10620 RETURN                                 <051>
```

Listing 9

Wir haben in jedem »Raum« nun noch eine DATA-Zeile mit zehn Werten. Diese Werte geben an, ob und in welche Richtungen gelaufen werden kann. Jedesmal, wenn der Spieler einen Raum betritt, werden diese zehn Zahlen in das Feld RI(1) bis RI(10) eingelesen.

Dieses Feld ist so aufgebaut:

RI	1	2	3	4	5	6	7	8	9	10
	N	S	W	O	NW	NO	SW	SO	RAUF	RUNTER

Diese Skizze zeigt innerhalb welcher Programmzeilen welche Programmteile liegen.

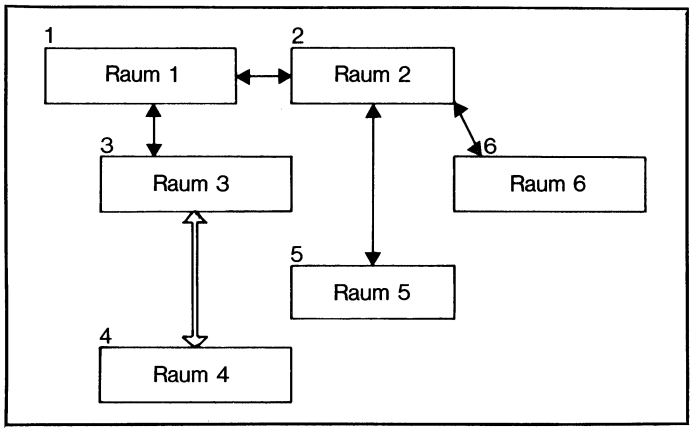
Bisher haben wir nur einige Zeilen des Kopfes (Befehlserweiterung um GOTO X etc.) sowie ab 50000 für das Befehlseingabemodul belegt.

Die Zeilen 10000 bis maximal 49999 sind der Spielkarte gewidmet.

Wir haben also 39999 Zeilen für die Programmierung der Spielkarte zur Verfügung. Man kann davon ausgehen, daß man für die Programmierung eines einzelnen Raumes der Spielkarte (Text und Action) allerhöchstens 100 Zeilen benötigt. In 39999 Zeilen lassen sich also maximal 400 Räume unterbringen. Ein vernünftiges Adventure in bezug auf Räume und Action wird jedoch bestimmt nicht 400 Räume haben, da dies mit 38 KByte RAM wohl kaum möglich sein

dürfte. Ich war trotzdem so großzügig und habe 39999 Zeilen für ein Maximum von 400 Räumen zur Verfügung gestellt (was natürlich nicht heißen soll, daß Ihre Spiele viele Räume haben müssen), denn es ist besser, wenn man sich zehnmal überschätzt als einmal unterschätzt hat, was die Menge und Verteilung der Programmzeilen anbelangt. Das vorgestellte System ist somit auch zukunftssicher (vielleicht gibt es doch einmal eine Basic-RAM-Erweiterung für den C 64, oder man setzt den neuen PC 128 ein).

Für die nächsten Überlegungen werden wir uns auf die folgende Spielkarte beziehen:



Dies ist ein Spielplan mit sechs Räumen. Wie man solche Pläne anfertigt, haben Sie bereits im Theorie-Teil dieses Kurses gelernt. Die einzelnen Räume sind von 1 bis 6 durchnumeriert.

Ich habe den Plan absichtlich so kurz gehalten, da er für unsere ersten Versuche völlig ausreicht, und ich Ihnen unnötige Abtipparbeit soweit wie möglich ersparen möchte.

Bitte ergänzen Sie Ihr bisheriges Programm (Basic-Erweiterungsroutine und Befehlseingabemodul) nun um die Zeilen von Listing 10.

```
10102 DATA 0,3,0,2,0,0,0,0,0,0          <145>
10202 DATA 0,5,1,0,0,0,0,6,0,0          <252>
10302 DATA 1,0,0,0,0,0,0,0,0,4          <090>
10402 DATA 0,0,0,0,0,0,0,0,3,0          <188>
10502 DATA 2,0,0,0,0,0,0,0,0,0          <032>
10602 DATA 0,0,0,0,2,0,0,0,0,0          <132>
```

Listing 10

Zum Verständnis soll folgendes Beispiel dienen:
Der Spieler steht im Raum A.

Im Feld RI(1) bis RI(10) ist festgelegt, welche Räume von Raum A unter welchen Richtungen erreicht werden können.

Nehmen wir einmal an, es gibt nur eine mögliche Richtung von Raum A aus, und zwar die Richtung Sueden (S), durch die man zu Raum B gelangt. Die DATA-Zeile für Raum A müßte in diesem Fall so aussehen:

DATA 0,B,0,0,0,0,0,0,0,0

Praktisch würde man für B natürlich den Wert des Raumes B einsetzen. Durch diese DATA-Zeile bedingt, muß das RI-Feld so aussehen:

RI(1)=0, RI(2)=B, RI(3) bis RI(10)=0

Wir nehmen nun an, daß der sich in Raum A befindliche Spieler in Richtung Norden (N) laufen will.

Auf die Frage »WAS NUN?« gibt er folglich »N« ein.

Das Befehlseingabemodul gibt sodann den Wert für das eingegebene Verb »N« aus. Also den Wert VE=1.

Nun tritt das Actionmodul in Kraft.

Zuerst überprüft es, ob der Verbzahlwert VE im Bereich zwischen 1 und 10 liegt — also, ob der Spieler sich bewegen will oder nicht (VE größer als 10). Das Actionmodul stellt fest, daß der Spieler sich bewegen will und fährt fort.

Es überprüft nun, ob die Richtung, in die sich der Spieler bewegen will, überhaupt existiert.

Es überprüft dazu, ob RI(VE) gleich Null ist oder nicht.

Ist RI(VE) = 0, so erfolgt die Fehlermeldung »Kein Weg in diese Richtung!«.

Ist RI(VE) jedoch ungleich Null (was in Raum A der Fall ist, wenn sich der Spieler nach Süden bewegen will), so fährt das Modul fort. Hätte unser Spieler also »S« statt »N« eingegeben, so wäre folgendes passiert.

Das Actionmodul hätte erst einmal die neue Zimmernummer registriert:

ZN = RI(VE)

Danach: GOSUB 10000 + ZN*100

Dann: neues Feld RI von Raum B einlesen.

Fertig.

Diese Theorie wird mittels den folgenden Ergänzungszeilen nun in die Praxis umgesetzt:

```
1010 GOSUB 50000 : REM BEFEHLSEINGABEMODUL
1100 REM GEHEN IN EIN NEUES ZIMMER
1105 IFVE<1 OR VE>10 THEN1200
1110 IF RI(VE)=0 THENPRINT"KEIN WEG IN DIESE
     RICHTUNG !":GOTO1200
1120 ZN = RI(VE) : PRINT "{CLR}"
1130 GOSUB 10000 + ZN*100
1140 RESTORE 10000 + ZN*100
1150 FOR I=1 TO 10 : READ RI(I) : NEXT
1200 GOTO 1000
```

Nun sind wir schon fast fertig.

Wir müssen im Kopf des Programmes nur noch eine Zeile einfügen, in der bestimmt wird, in welchem Zimmer der Spieler sich zu Beginn des Spiels befindet,

also: 100 ZN = 1 : GOTO 1130

Das Spiel beginnt somit in Raum 1. Um das Spiel in irgend einem anderen Raum beginnen zu lassen, muß man nur den Wert in Zeile 100 entsprechend ändern, was beim späteren Fehlersuchen und Testen des Spiels eine große Hilfe darstellt.

Hiermit ist unser Ziel erreicht — Sie wissen nun, wie man eine Spielkarte programmiert.

```
0 REM ****************************          <081>
1 REM * ADVENTURE-PROGRAMMIERKURS *          <073>
2 REM *                          *          <229>
3 REM *     UEBUNGS-PROGRAMM     *          <145>
4 REM ****************************          <085>
9 REM BASIC-ERWEITERUNG                      <120>
10 DATA 76,24,1,177,251,145,251,200,208,24
   9,230,252,202,208,244,96,120,160,0,169    <104>
12 DATA 160,132,251,133,252,162,32,32,11,1
   ,169,224,132,251,133,252,162,32,32,11     <025>
14 DATA 1,169,53,133,1,88,96:FOR I=264 TO
   310:READ X:POKE I,X:NEXT:SYS 264          <234>
16 FOR I=710 TO 730:READ X:POKE I,X:NEXT     <213>
18 DATA 208,3,76,29,168,32,192,2,32,19,166
   ,56,165,95,233,1,164,96,76,36,168         <165>
20 POKE 40996,197:POKE 40997,2:POKE 1,54     <087>
22 FOR I=43168 TO 43170:READ X:POKE I,X:NE
   XT                                        <174>
24 FOR I=704 TO 709:READ X:POKE I,X:NEXT     <230>
26 DATA 32,192,2,32,138,173,76,247,183       <114>
30 GOSUB 52000: REM TABELLEN DEFFINIEREN     <209>
100 ZN=1:GOTO 1130                           <119>
1000 REM A C T I O N M O D U L               <185>
1010 GOSUB 50000:REM BEFEHLSEINGABEMODUL     <165>
1100 REM GEHEN IN EIN NEUES ZIMMER           <013>
1105 IF VE<1 OR VE>10 THEN 1200              <038>
1110 IF RI(VE)=0 THEN PRINT"KEIN WEG IN DI
     ESE RICHTUNG !":GOTO 1200               <232>
1120 ZN=RI(VE):PRINT"{CLR}"                  <239>
1130 GOSUB 10000+ZN*100                      <123>
1140 RESTORE 10000+ZN*100                    <132>
1150 FOR I=1 TO 10:READ RI(I):NEXT           <230>
1200 GOTO 1000                               <254>
10000 REM ------- SPIELKARTE -------         <048>
10100 REM RAUM 1 ----------                  <037>
```

Listing 11

```
10102 DATA 0,3,0,2,0,0,0,0,0,0                <145>
10105 PRINT"RAUM NUMMER 1"                    <183>
10120 RETURN                                  <061>
10200 REM RAUM 2 ---------                    <138>
10202 DATA 0,5,1,0,0,0,0,6,0,0                <252>
10205 PRINT"RAUM NUMMER 2"                    <028>
10220 RETURN                                  <161>
10300 REM RAUM 3 ---------                    <240>
10302 DATA 1,0,0,0,0,0,0,0,0,4                <090>
10305 PRINT"RAUM NUMMER 3"                    <130>
10320 RETURN                                  <006>
10400 REM RAUM 4 ---------                    <085>
10402 DATA 0,0,0,0,0,0,0,0,3,0                <188>
10405 PRINT"RAUM NUMMER 4"                    <231>
10420 RETURN                                  <106>
10500 REM RAUM 5 ---------                    <187>
10502 DATA 2,0,0,0,0,0,0,0,0,0                <032>
10505 PRINT"RAUM NUMMER 5"                    <077>
10520 RETURN                                  <207>
10600 REM RAUM 6 ---------                    <032>
10602 DATA 0,0,0,0,2,0,0,0,0,0                <132>
10605 PRINT"RAUM NUMMER 6"                    <178>
10620 RETURN                                  <051>
50000 REM ****************************       <058>
50010 REM *      BEFEHLSEINGABE        *      <228>
50020 REM *      BEFEHLSZERLEGUNG      *      <182>
50030 REM *      BEFEHLSCODIERUNG      *      <173>
50040 REM ****************************       <098>
50050 IF UD=1 THEN 50220                      <236>
50060 POKE 198,0:BE$="":PRINT"WAS NUN ? ";    <044>
50070 POKE 204,0                              <226>
50080 GET X$:IF X$=""THEN 50080               <091>
50090 IF PEEK(203)=1 OR LEN(BE$)>68 THEN P
      RINT " ":POKE 204,1:GOTO 50140          <022>
50100 I=ASC(X$):IF I<65 OR I>90 THEN IF I<
      >32 AND I<>20 AND I<>34 THEN 50080      <173>
50110 IF I=20 AND BE$=""THEN 50080            <093>
50120 IF I=20 THEN POKE 204,1:PRINT"{LEFT,
      2SPACE,2LEFT}";:BE$=LEFT$(BE$,LEN(BE
      $)-1):GOTO 50070                        <174>
50130 PRINT X$;:BE$=BE$+X$:GOTO 50080         <013>
50140 FOR I=1 TO 10:BE$(I)="":NEXT:WZ=1:FO
      R I=1 TO LEN(BE$)                       <154>
50150 :IF MID$(BE$,I,1)=" "THEN GOSUB 5019
      0:GOTO 50180                            <240>
50160 :IF WZ>10 THEN PRINT"EINGABE IST ZU
      LANG !":I=LEN(BE$)+1:GOTO 50180         <028>
50170 :BE$(WZ)=BE$(WZ)+MID$(BE$,I,1)          <069>
50180 NEXT I:GOTO 50220                       <079>
50190 IC=0:FOR I1=1 TO AZ:IF BE$(WZ)=AU$(I
      1)THEN IC=1                             <086>
50200 NEXT I1:IF IC=0 THEN WZ=WZ+1:RETURN     <105>
50210 BE$(WZ)="":RETURN                       <081>
50220 IF UD=1 THEN UD=0:GOTO 50240            <215>
50230 WZ=1:VE=0:OB=0:PE=0                     <163>
50240 IC=0:G1=0:G2=0                          <155>
50250 FOR I=1 TO VZ:IF BE$(WZ)=VE$(I)THEN
      VE=I:IC=1                               <114>
50251 IF LEN(BE$(WZ))<3 THEN 50260            <229>
50255 IF BE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ))
      )THEN VE=I-VAL(RIGHT$(VE$(I),1)):IC=
      1                                       <211>
50260 NEXT I:IF IC=1 THEN 50350               <187>
50270 FOR I=1 TO GZ:IF BE$(WZ)<>GE$(I)THEN
       50300                                  <211>
50280 IC=1:IF G1=0 THEN G1=I                  <212>
50290 G2=I:IF G2=G1 THEN G2=0                 <020>
50300 NEXT I:IF IC=1 THEN 50350               <227>
50310 FOR I=1 TO OZ:IF BE$(WZ)=OB$(I)THEN
      OB=I:IC=1                               <147>
50320 NEXT I:IF IC=1 THEN 50350               <247>
50330 FOR I=1 TO PZ:IF BE$(WZ)=PE$(I)THEN
      PE=I:IC=1                               <176>
50340 NEXT I                                  <051>
50350 IF BE$(WZ)="UND"THEN UD=1:IC=1           <083>
50360 IF IC=0 THEN PRINT"ICH KENNE ";BE$(W
      Z);" NICHT !":RETURN                    <004>
50370 WZ=WZ+1:IF WZ>10 OR BE$(WZ)=""OR UD=
      1 THEN RETURN                           <177>
50380 IC=0:GOTO 50250                         <189>
52000 REM T A B E L L E N                     <161>
52005 RESTORE 52000                           <115>
52010 REM VERBTABELLE ---------               <065>
52020 DATA N,S,W,O,NW,NO,SW,SO,RAUF,RUNTER    <241>
52100 VZ=10:DIM VE$(VZ):FOR I=1 TO VZ:READ
       VE$(I):NEXT                            <163>
53000 RETURN                                  <101>
```

Listing 11

Die vorgestellte Spielkarte ist natürlich jedoch nur mit einem sehr spärlichen Raumbeschreibungstext ausgestattet. Dies können Sie aber leicht selbst verbessern — das nötige Wissen dazu haben Sie jetzt. Wenn Sie bisher mitgearbeitet haben, müßten Sie nun das Listing 11 im Computer haben.

Wir haben nun ein Adventure mit Räumen der primitivsten Art vorliegen. Alles was man bisher machen kann, ist herumlaufen.

Es ist nun sehr wünschenswert, daß man nach jeder Raumbeschreibung eine Angabe erhält, in welche Richtung man sich bewegen kann.

Wir schreiben deshalb nun eine kleine Routine, die nach jeder Raumbeschreibung »MÖGLICHE RICHTUNGEN:...« ausgibt.

Dies läßt sich ganz einfach realisieren, wenn man unser bisheriges Programm durch Listing 12 ergänzt:

```
1160 PRINT"{DOWN}MOEGLICHE RICHTUNGEN : "; <115>
1165 IC=0:FOR I=1 TO 10:IF RI(I)<>0 THEN P
     RINT VE$(I);",";:IC=1                  <161>
1170 NEXT I                                 <097>
1175 IF IC=0 THEN PRINT"KEINE."             <178>
1180 IF IC=1 THEN PRINT CHR$(20)            <084>
```

Listing 12

Eine Beschreibung erübrigt sich, da es sich um eine einfache Testschleife handelt, wie wir sie bereits des öfteren besprochen haben.

Im Prinzip ist die Programmierung der Spielkarte nun fast abgeschlossen. Wir brauchen nur noch zwei kleine Routinen. Eine davon sorgt dafür, daß bei jeder Raumbeschreibung ausgegeben wird, welche Gegenstände sich im jeweiligen Raum befinden.

Dazu müssen wir jedoch zunächst unseren Wortschatz mit einigen Gegenständen erweitern.

Die Gegenstandsnamen werden, wie bereits besprochen, in dem Feld GE$(1) bis GE$(GZ) abgelegt, wobei GZ die Anzahl der Gegenstände darstellt.

Wir brauchen jedoch noch ein weiteres Feld für die Gegenstände, in dem angegeben wird, in welchen Räumen sich diese befinden.

Dieses Feld sei GE(1) bis GE(GZ).

Wir ergänzen unser Programm nun um Listing 13.

```
52200 REM GEGENSTANDSTABELLE ---------      <003>
52210 DATA SCHWERT,1                        <189>
52211 DATA SEIL,2                           <204>
52212 DATA SCHLUESSEL,4                     <157>
52213 DATA DIAMANT,5                        <162>
52300 GZ=4:DIM GE$(GZ):DIM GE(GZ):FOR I=1
      TO GZ:READ GE$(I):READ GE(I):NEXT I   <098>
```

Listing 13

Bitte notieren Sie sich auch diese Tabelle auf Papier.

In jeder DATA-Zeile dieser Tabelle steht zuerst der Name des Gegenstands, und danach eine Zahl, die angibt, in welchem Raum sich der Gegenstand befindet (Raumzahl: vergleiche Karte).

```
1185 PRINT"ICH SEHE : ";                    <240>
1188 IC=0:FOR I=1 TO GZ:IF GE(I)=ZN THEN P
     RINT GE$(I);",";:IC=1                  <160>
1190 NEXT                                   <044>
1192 IF IC=0 THEN PRINT"NICHTS BESONDERES.
     "                                      <010>
1194 IF IC=1 THEN PRINT CHR$(20)            <098>
```

Listing 14

Nun wird die Routine eingefügt, die dafür sorgt, daß bei jeder Raumbeschreibung die sich im Raum befindlichen Gegenstände aufgezählt werden.

Tippen Sie bitte Listing 14 ein.

Wenn Sie nun wieder im Spielplan »herumlaufen« , so können Sie leicht feststellen, wo sich die einzelnen Gegenstände befinden.

Nun müssen wir uns den Objekten zuwenden (Türen, Fenster, Schränke etc.) Zunächst wird wieder die Tabelle für die Objekte definiert.

Die Objektnamen werden in OB$(1) bis OB$(OZ) untergebracht. Außerdem legen wir noch ein Feld OO(1) bis OO(OZ) an, in dem, analog zu den Gegenständen, gespeichert wird, wo sich die einzelnen Objekte befinden. Wir ergänzen durch Listing 15.

```
52400 REM OBJEKTTABELLE ---------          <088>
52410 DATA TRUHE,5                         <242>
52412 DATA SCHACHT,6                       <107>
52414 DATA EISENRING,6                     <019>
52416 DATA TUER,2                          <173>
52418 DATA TUER,5                          <178>
52500 OZ=5:DIM OB$(OZ):DIM OO(OZ):FOR I=1
      TO OZ:READ OB$(I):READ OO(I):NEXT I  <122>
```

Listing 15

Was nun folgt ist ganz klar: Es muß eine Routine geschrieben werden, die bei jeder Raumbeschreibung die Objekte aufzählt (ganz analog zu den Gegenständen). Da es stilistisch schöner ist, die Objekte in der Aufzählung den Gegenständen voranzustellen, fügen wir in die bereits geschriebene Routine für die Gegenstände einfach eine Schleife ein, die die Objekte ausgibt — falls im Raum welche vorhanden sind.

Es muß nur beachtet werden, daß in Zeile 1188 (Schleife für Gegenstände) die Checkvariable IC nicht mehr auf Null gesetzt werden darf, da die Meldung »Nichts besonderes« nur dann erfolgen soll, wenn sich weder Objekt noch Gegenstand im Raum befinden.

Ergänzen Sie bitte nun um Listing 16.

```
1186 IC=0:FOR I=1 TO OZ:IF OO(I)=ZN THEN P
     RINT OB$(I);",";:IC=1                 <189>
1187 NEXT                                  <041>
1188 FOR I=1 TO GZ:IF GE(I)=ZN THEN PRINT
     GE$(I);",";:IC=1                      <248>
```

Listing 16

Hiermit wären wir auch schon am Ende der Spielkarten-Programmierung angelangt. Sie haben dafür eine Programmiertechnik kennengelernt, die folgende besondere Merkmale hat:

— geringer Speicherplatzverbrauch
— leichte Korrekturmöglichkeit der Spielkarte
— vollkommen ausbaufreundlich

Das somit vorliegende Adventure ist also leicht modifizierbar.

Sie können jeden Raum mit beliebigem Text versehen, Gegenstände und Objekte auf der Spielkarte verteilen und Sie haben die Möglichkeit, das Spiel in jedem beliebigen Raum beginnen zu lassen (dazu muß nur der ZN-Wert in Zeile 100 verändert werden), was beim späteren Testen des Spiels sehr wichtig ist. Sie können auch jederzeit neue Gegenstände und Objekte in die Karte einfügen — dazu müssen nur die Tabellen, die ab Zeile 52000 liegen, ergänzt und die Variablen GZ, OZ etc. entsprechend angepaßt werden.

Wenn Sie bis jetzt richtig mitgearbeitet haben, so müssen Sie Listing 17 im Computer haben.

```
0 REM ****************************          <081>
1 REM * ADVENTURE-PROGRAMMIERKURS *         <073>
2 REM *                           *         <229>
3 REM *      UEBUNGS-PROGRAMM     *         <145>
4 REM ****************************          <085>
9 REM BASIC-ERWEITERUNG                     <120>
10 DATA 76,24,1,177,251,145,251,200,208,24
   9,230,252,202,208,244,96,120,160,0,169   <104>
12 DATA 160,132,251,133,252,162,32,32,11,1
   ,169,224,132,251,133,252,162,32,32,11    <025>
14 DATA 1,169,53,133,1,88,96:FOR I=264 TO
   310:READ X:POKE I,X:NEXT:SYS 264         <234>
16 FOR I=710 TO 730:READ X:POKE I,X:NEXT    <213>
18 DATA 208,3,76,29,168,32,192,2,32,19,166
   ,56,165,95,233,1,164,96,76,36,168        <165>
20 POKE 40996,197:POKE 40997,2:POKE 1,54    <087>
22 FOR I=43168 TO 43170:READ X:POKE I,X:NE
   XT                                       <174>
24 FOR I=704 TO 709:READ X:POKE I,X:NEXT    <230>
26 DATA 32,192,2,32,138,173,76,247,183      <114>
30 GOSUB 52000: REM TABELLEN DEFFINIEREN    <209>
100 ZN=1:GOTO 1130                          <119>
1000 REM A C T I O N M O D U L              <185>
1010 GOSUB 50000:REM BEFEHLSEINGABEMODUL    <165>
1100 REM GEHEN IN EIN NEUES ZIMMER          <013>
1105 IF VE<1 OR VE>10 THEN 1200             <038>
1110 IF RI(VE)=0 THEN PRINT"KEIN WEG IN DI
     ESE RICHTUNG !":GOTO 1200              <232>
1120 ZN=RI(VE):PRINT"{CLR}"                 <239>
1130 GOSUB 10000+ZN*100                     <123>
1140 RESTORE 10000+ZN*100                   <132>
1150 FOR I=1 TO 10:READ RI(I):NEXT          <230>
1160 PRINT"{DOWN}MOEGLICHE RICHTUNGEN : ";  <115>
1165 IC=0:FOR I=1 TO 10:IF RI(I)<>0 THEN P
     RINT VE$(I);",";:IC=1                  <161>
1170 NEXT I                                 <097>
1175 IF IC=0 THEN PRINT"KEINE."             <178>
1180 IF IC=1 THEN PRINT CHR$(20)            <084>
1185 PRINT"ICH SEHE : ";                    <240>
1186 IC=0:FOR I=1 TO OZ:IF OO(I)=ZN THEN P
     RINT OB$(I);",";:IC=1                  <189>
1187 NEXT                                   <041>
1188 FOR I=1 TO GZ:IF GE(I)=ZN THEN PRINT
     GE$(I);",";:IC=1                       <248>
1190 NEXT                                   <044>
1192 IC=0 THEN PRINT"NICHTS BESONDERES.
     "                                      <010>
1194 IF IC=1 THEN PRINT CHR$(20)            <098>
1200 GOTO 1000                              <254>
10000 REM ------- SPIELKARTE -------        <048>
10100 REM RAUM 1 ----------                 <037>
10102 DATA 0,3,0,2,0,0,0,0,0,0              <145>
10105 PRINT"RAUM NUMMER 1"                  <183>
10120 RETURN                                <061>
10200 REM RAUM 2 ----------                 <138>
10202 DATA 0,5,1,0,0,0,0,6,0,0              <252>
10205 PRINT"RAUM NUMMER 2"                  <028>
10220 RETURN                                <161>
10300 REM RAUM 3 ----------                 <240>
10302 DATA 1,0,0,0,0,0,0,0,0,4              <090>
10305 PRINT"RAUM NUMMER 3"                  <130>
10320 RETURN                                <006>
10400 REM RAUM 4 ----------                 <085>
10402 DATA 0,0,0,0,0,0,0,0,0,3,0            <188>
10405 PRINT"RAUM NUMMER 4"                  <231>
10420 RETURN                                <106>
10500 REM RAUM 5 ----------                 <187>
10502 DATA 2,0,0,0,0,0,0,0,0,0              <032>
10505 PRINT"RAUM NUMMER 5"                  <077>
10520 RETURN                                <207>
10600 REM RAUM 6 ----------                 <032>
10602 DATA 0,0,0,0,2,0,0,0,0,0              <132>
10605 PRINT"RAUM NUMMER 6"                  <178>
10620 RETURN                                <051>
50000 REM ****************************      <058>
50010 REM *      BEFEHLSEINGABE        *    <228>
50020 REM *      BEFEHLSZERLEGUNG      *    <182>
50030 REM *      BEFEHLSCODIERUNG      *    <173>
50040 REM ****************************      <098>
50050 IF UD=1 THEN 50220                    <236>
50060 POKE 198,0:BE$="":PRINT"WAS NUN ? ";  <044>
50070 POKE 204,0                            <226>
50080 GET X$:IF X$=""THEN 50080             <091>
50090 IF PEEK(203)=1 OR LEN(BE$)>68 THEN P
      RINT" ":POKE 204,1:GOTO 50140         <022>
50100 I=ASC(X$):IF I<65 OR I>90 THEN IF I<
      >32 AND I<>20 AND I<>34 THEN 50080    <173>
```

Listing 17

```
50110 IF I=20 AND BE$=""THEN 50080                    <093>
50120 IF I=20 THEN POKE 204,1:PRINT"{LEFT,
      2SPACE,2LEFT}";:BE$=LEFT$(BE$,LEN(BE
      $)-1):GOTO 50070                                <174>
50130 PRINT X$;:BE$=BE$+X$:GOTO 50080                 <013>
50140 FOR I=1 TO 10:BE$(I)="":NEXT:WZ=1:FO
      R I=1 TO LEN(BE$)                               <154>
50150 :IF MID$(BE$,I,1)=" "THEN GOSUB 5019
      0:GOTO 50180                                    <240>
50160 :IF WZ>10 THEN PRINT"EINGABE IST ZU
      LANG !":I=LEN(BE$)+1:GOTO 50180                 <028>
50170 :BE$(WZ)=BE$(WZ)+MID$(BE$,I,1)                  <069>
50180 NEXT I:GOTO 50220                               <079>
50190 IC=0:FOR I1=1 TO AZ:IF BE$(WZ)=AU$(I
      1)THEN IC=1                                     <086>
50200 NEXT I1:IF IC=0 THEN WZ=WZ+1:RETURN             <105>
50210 BE$(WZ)="":RETURN                               <081>
50220 IF UD=1 THEN UD=0:GOTO 50240                    <215>
50230 WZ=1:VE=0:OB=0:PE=0                             <163>
50240 IC=0:G1=0:G2=0                                  <155>
50250 FOR I=1 TO VZ:IF BE$(WZ)=VE$(I)THEN
      VE=I:IC=1                                       <114>
50251 IF LEN(BE$(WZ))<3 THEN 50260                    <229>
50255 IF BE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ))
      )THEN VE=I-VAL(RIGHT$(VE$(I),1)):IC=
      1                                               <211>
50260 NEXT I:IF IC=1 THEN 50350                       <187>
50270 FOR I=1 TO GZ:IF BE$(WZ)<>GE$(I)THEN
       50300                                          <211>
50280 IC=1:IF G1=0 THEN G1=I                          <212>
50290 G2=I:IF G2=G1 THEN G2=0                         <020>
50300 NEXT I:IF IC=1 THEN 50350                       <227>
50310 FOR I=1 TO OZ:IF BE$(WZ)=OB$(I)THEN
      OB=I:IC=1                                       <147>
50320 NEXT I:IF IC=1 THEN 50350                       <247>
50330 FOR I=1 TO PZ:IF BE$(WZ)=PE$(I)THEN
      PE=I:IC=1                                       <176>
50340 NEXT I                                          <051>
50350 IF BE$(WZ)="UND"THEN UD=1:IC=1                  <083>
50360 IF IC=0 THEN PRINT"ICH KENNE ";BE$(W
      Z);" NICHT !":RETURN                            <004>
50370 WZ=WZ+1:IF WZ>10 OR BE$(WZ)=""OR UD=
      1 THEN RETURN                                   <177>
50380 IC=0:GOTO 50250                                 <189>
52000 REM T A B E L L E N                             <161>
52005 RESTORE 52000                                   <115>
52010 REM VERBTABELLE ---------                       <065>
52020 DATA N,S,W,O,NW,NO,SW,SO,RAUF,RUNTER            <241>
52100 VZ=10:DIM VE$(VZ):FOR I=1 TO VZ:READ
       VE$(I):NEXT                                    <163>
52200 REM GEGENSTANDSTABELLE ---------                <003>
52210 DATA SCHWERT,1                                  <189>
52211 DATA SEIL,2                                     <204>
52212 DATA SCHLUESSEL,4                               <157>
52213 DATA DIAMANT,5                                  <162>
52300 GZ=4:DIM GE$(4):DIM GE(4):FOR I=1 TO
       GZ:READ GE$(I):READ GE(I):NEXT I               <136>
52400 REM OBJEKTTABELLE ---------                     <088>
52410 DATA TRUHE,5                                    <242>
52412 DATA SCHACHT,6                                  <107>
52414 DATA EISENRING,6                                <019>
52416 DATA TUER,2                                     <173>
52418 DATA TUER,5                                     <178>
52500 OZ=5:DIM OB$(OZ):DIM OO(OZ):FOR I=1
      TO OZ:READ OB$(I):READ OO(I):NEXT I             <122>
53000 RETURN                                          <101>
```

Listing 17 (Schluß)

Mit diesem Listing werden wir in den folgenden Abschnitten weiterarbeiten — vergessen Sie deshalb nicht, es richtig abzuspeichern!

Leider ist das vorliegende Adventure noch ohne jeglichen Witz.

Im folgenden Abschnitt lernen Sie, wie man das Actionmodul mit Routinen wie NIMM, VERLIERE, INVENTUR, SCHAU, OEFFNE etc. programmiert.

Bitte lesen Sie aber erst dann weiter, wenn Sie die Programmierung der Spielkarte in allen Einzelheiten gut verstanden haben.

Ausbau des Actionmoduls

Endlich ist es soweit — wir bringen Leben in unser bisher bescheidenes Adventure.

Im allgemeinen kann man die Actionprogrammierung in zwei Hauptgruppen aufteilen:

a) Allgemeine Action
   Dies sind Dinge, die der Spieler ständig machen kann — also überall auf der Spielkarte (zum Beispiel NIMM, VERLIERE, WARTE, SINGE, INVENTUR).

b) Raumspezifische Action
   Bei dieser Action handelt es sich um etwas, das nur in einem bestimmten Raum durchgeführt werden kann, zum Beispiel Öffnen einer Tür, einer Truhe oder Drücken eines Knopfes, der sich an der Zimmerwand befindet.

Wir wollen uns zunächst der allgemeinen Action zuwenden. Dazu müssen wir zunächst die Verbtabelle um einige Verben ergänzen, die im Rahmen der allgemeinen Action verwendet werden.

OEFFNE: Dieser Befehl dient zum Öffnen von Türen, Kisten etc.

SCHLIESSE: Umkehrung von OEFFNE

SCHAUE, UNTERSUCHE: Diese Verben haben Doppelfunktion.

Alleinstehend angewandt (»SCHAUE«) geben Sie die Raumbeschreibung wieder.

In Verbindung mit einem Objekt oder Gegenstand (»UNTERSUCHE TRUHE«) geben sie eine Beschreibung dieses Gegenstands beziehungsweise Objekts wieder.

NIMM, NEHME, HOLE: Sie dienen zum Nehmen von Gegenständen.

VERLIERE, LEGE, WIRF, WERFE: Umkehrung von NIMM.

INVENTUR: Mittels diesem Befehl erfährt der Spieler, welche Gegenstände er zur Zeit bei sich hat.

Lassen Sie uns nun die Verbtabelle ergänzen:

```
52030 DATA OEFFNE,SCHLIESSE,SCHAUE,
      UNTERSUCHE1,NIMM,NEHME1,HOLE2
52035 DATA VERLIERE,LEGE1,WIRF2,WERFE3,
      INVENTUR
```

Der VZ-Wert in Zeile 52100 muß entsprechend auf VZ=22 geändert werden.

Nach dieser Ergänzung sieht unsere Verbtabelle so aus:

1 bis 10	Himmelsrichtungen
11	OEFFNE
12	SCHLIESSE
13	SCHAUE,UNTERSUCHE
15	NIMM,NEHME,HOLE
18	VERLIERE,LEGE,WIRF,WERFE
22	INVENTUR

Sicher können Sie sich noch an den Abschnitt »Befehlsanalyse« erinnern, in dem der Begriff Alternativverb eingeführt worden ist.

Dadurch läßt sich auch erklären, warum die Verben nicht direkt durchnumeriert sind.

Die Tabelle sagt zum Beispiel aus, daß ganz egal, ob der Spieler nun NIMM, NEHME oder gar HOLE eingibt, die Verbzahl VE = 15 entsteht.

Entsprechendes gilt für die anderen Verbfamilien.

Lassen Sie uns nun die einzelnen Routinen in unser Adventure einbauen.

NIMM-Routine

Auf unserem Spielplan haben wir bereits einige Gegenstände verteilt (Schwert, Seil, Schlüssel, Diamant). Wir wollen unserem Spieler nun erlauben, diese Gegenstände auch einsammeln zu können.

Wir ergänzen das Actionmodul um Listing 18.

```
1200 REM                                          <067>
2000 REM REAKTION AUF BEFEHLE (ALLGEMEINE
     ACTION)                                      <108>
2100 REM NIMM ROUTINE                             <034>
2110 IF VE<>15 THEN 2200                          <161>
2120 IF GE(G1)<>ZN THEN PRINT"ICH SEHE DIE
     SEN GEGENSTAND HIER NICHT !"                 <016>
2125 IF GE(G1)=-1 THEN PRINT"SIE HABEN DIE
     SEN GEGENSTAND BEREITS !"                    <077>
2130 IF GE(G1)=ZN THEN GE(G1)=-1:PRINT"OK.
     "                                            <253>
2200 GOTO 1000                                    <234>
```

Listing 18

Bitte starten Sie das Adventure.

Geben Sie nacheinander folgende Befehle ein:

NIMM SCHWERT, S, N

Damit haben Sie in Raum 1 das Schwert genommen und sind nach S und dann, wieder mittels N, zurück nach Raum 1 gegangen — das Schwert wird in der Raumbeschreibung nun nicht mehr erwähnt. Wie funktioniert diese Routine?

Gehen wir einmal von der Befehlseingabe »NIMM SCHWERT« aus:

Das Befehlsanalysemodul liefert an das Actionmodul folgende Werte:

VE = 15 (Verbzahl)

G1 = 1 (1. Gegenstandszahl) vgl. auch Gegenstandstabelle.

Daraufhin läuft die NIMM-Routine so ab:

2110: Wenn die Verbzahl VE ungleich 15 ist, so wird die NIMM-Routine übersprungen. Mit so einer Abfrage beginnt jede Routine.

2120: Wenn der Gegenstand GE(G1) nicht im gleichen Zimmer wie der Spieler ist, erfolgt eine Fehlermeldung.

2125: Wenn der Spieler den Gegenstand bereits hat, erfolgt eine Fehlermeldung, denn man kann nichts nehmen, was man bei sich trägt.

2130: Wenn sich der Gegenstand im Zimmer befindet (GE(G1) = ZN), so nimmt der Spieler ihn (GE(G1) = —1).

Wir legen fest: Jeder Gegenstand, der im Besitz des Spielers ist, hat, beziehungsweise bekommt, den Wert —1.

INVENTUR-Routine

Mit dem Befehl INVENTUR erfährt der Spieler jederzeit, welche Gegenstände in seinem Besitz sind.

Bitte ergänzen Sie das Adventure um Listing 19.

```
2200 REM INVENTUR ROUTINE                         <208>
2205 IF VE<>22 THEN 2300                          <255>
2210 PRINT"ICH HABE: ";                           <224>
2220 IC=0:FOR I=1 TO GZ:IF GE(I)=-1 THEN P
     RINT GE$(I)",";:IC=1                         <165>
2225 NEXT                                         <059>
2230 IF IC=0 THEN PRINT"NICHTS."                  <050>
2235 IF IC=1 THEN PRINT"{LEFT,SPACE}"             <222>
2300 GOTO 1000                                    <078>
```

Listing 19

Die Routine besteht im Prinzip nur aus einer Schleife, die die Gegenstandstabelle durchläuft und überprüft, welche Gegenstände den Wert —1 haben — also im Besitz des Spielers sind.

Wenn Sie Ihr bisheriges Wissen einmal anwenden wollen, so können Sie einmal versuchen, die VERLIER-Routine, die ab Zeile 2300 stehen soll, selbst zu programmieren. Dies ist nicht schwer, da diese Routine fast analog zur NIMM-Routine läuft. Wer es selbst versuchen will, der sollte das Heft nun zunächst einmal weglegen.

Die **VERLIER-Routine** ist in Listing 20 abgebildet.

```
2300 REM VERLIER ROUTINE                          <210>
2305 IF VE<>18 THEN 2400                          <106>
2310 IF GE(G1)<>-1 THEN PRINT"ICH HABE DAS
      NICHT !"                                    <006>
2320 IF GE(G1)=-1 THEN GE(G1)=ZN:PRINT"OK.
     "                                            <188>
2400 GOTO 1000                                    <179>
```

Listing 20

Das Listing ist leicht zu verstehen. Wenn Sie Schwierigkeiten haben, so vergleichen Sie es bitte mit dem Listing der NIMM-Routine.

SCHAU-Routine

Der Befehl SCHAU alleine angewandt, gibt eine Raumbeschreibung aus, beziehungsweise wiederholt eine Raumbeschreibung.

In Verbindung mit einem Gegenstand oder einem Objekt, gibt er eine Beschreibung dieses Gegenstands beziehungsweise Objekts aus.

Bitte ergänzen Sie durch Listing 21.

```
2400 REM SCHAU - ROUTINE                          <191>
2405 IF VE<>13 THEN 2500                          <202>
2410 IF OB=0 AND G1=0 THEN PRINT"{CLR}":VE
     =0:GOTO 1130                                 <208>
2415 IF OB=1 AND OO(1)=ZN THEN PRINT"DIE T
     RUHE IST SEHR GROSS."                        <092>
2500 GOTO 1000                                    <023>
```

Listing 21

Auch dieses Listing ist leicht zu verstehen.

Der Sprung GOTO 1130 in Zeile 2410 bewirkt die Neuausgabe der Raumbeschreibung.

Mit der Ergänzung durch diese Routinen sind wir unserem Ziel — dem fertigen Adventure — schon ziemlich nahe gekommen.

Sie haben jetzt das Wissen, das nötig ist, um große Spiellandschaften zu programmieren, in denen man herumlaufen und Gegenstände transportieren kann.

Sie müssen jetzt noch lernen, wie man Aufgaben ins Spiel einbaut, die der Spieler lösen muß.

Dies lernen Sie alles im folgenden Abschnitt — die raumspezifische Action. Zunächst jedoch wieder ein Kontrollisting 22 zum Vergleich mit dem, was Sie momentan bei richtiger Mitarbeit vorliegen haben sollten.

```
0 REM ******************************          <081>
1 REM * ADVENTURE-PROGRAMMIERKURS *          <073>
2 REM *                           *          <229>
3 REM *     UEBUNGS-PROGRAMM      *          <145>
4 REM ******************************          <085>
9 REM BASIC-ERWEITERUNG                       <120>
10 DATA 76,24,1,177,251,145,251,200,208,24
   9,230,252,202,208,244,96,120,160,0,169     <104>
12 DATA 160,132,251,133,252,162,32,32,11,1
   ,169,224,132,251,133,252,162,32,32,11      <025>
14 DATA 1,169,53,133,1,88,96:FOR I=264 TO
   310:READ X:POKE I,X:NEXT:SYS 264           <234>
16 FOR I=710 TO 730:READ X:POKE I,X:NEXT      <213>
18 DATA 208,3,76,29,168,32,192,2,32,19,166
   ,56,165,95,233,1,164,96,76,36,168          <165>
20 POKE 40996,197:POKE 40997,2:POKE 1,54      <087>
22 FOR I=43168 TO 43170:READ X:POKE I,X:NE
   XT                                         <174>
24 FOR I=704 TO 709:READ X:POKE I,X:NEXT      <230>
26 DATA 32,192,2,32,138,173,76,247,183        <114>
30 GOSUB 52000: REM TABELLEN DEFFINIEREN      <209>
100 ZN=1:GOTO 1130                            <119>
1000 REM A C T I O N M O D U L                <185>
1010 GOSUB 50000:REM BEFEHLSEINGABEMODUL      <165>
1100 REM GEHEN IN EIN NEUES ZIMMER            <013>
1105 IF VE<1 OR VE>10 THEN 1200               <038>
1110 IF RI(VE)=0 THEN PRINT"KEIN WEG IN DI
```

Listing 22

Fortsetzung Listing 22

```
     ESE RICHTUNG !":GOTO 1200                <232>
1120 ZN=RI(VE):PRINT"{CLR}"                   <239>
1130 GOSUB 10000+ZN*100                       <123>
1140 RESTORE 10000+ZN*100                     <132>
1150 FOR I=1 TO 10:READ RI(I):NEXT            <230>
1160 PRINT"{DOWN}MOEGLICHE RICHTUNGEN : ";    <115>
1165 IC=0:FOR I=1 TO 10:IF RI(I)<>0 THEN P
     RINT VE$(I);",";:IC=1                    <161>
1170 NEXT I                                   <097>
1175 IF IC=0 THEN PRINT"KEINE."               <178>
1180 IF IC=1 THEN PRINT CHR$(20)              <084>
1185 PRINT"ICH SEHE : ";                      <240>
1186 IC=0:FOR I=1 TO OZ:IF OO(I)=ZN THEN P
     RINT OB$(I);",";:IC=1                    <189>
1187 NEXT                                     <041>
1188 FOR I=1 TO GZ:IF GE(I)=ZN THEN PRINT
     GE$(I);",";:IC=1                         <248>
1190 NEXT                                     <044>
1192 IF IC=0 THEN PRINT"NICHTS BESONDERES.
     "                                        <010>
1194 IF IC=1 THEN PRINT CHR$(20)              <098>
1200 REM                                      <067>
2000 REM REAKTION AUF BEFEHLE (ALLGEMEINE
     ACTION)                                  <108>
2100 REM NIMM ROUTINE                         <034>
2110 IF VE<>15 THEN 2200                      <161>
2120 IF GE(G1)<>ZN THEN PRINT"ICH SEHE DIE
     SEN GEGENSTAND HIER NICHT !"             <016>
2125 IF GE(G1)=-1 THEN PRINT"SIE HABEN DIE
     SEN GEGENSTAND BEREITS !"                <077>
2130 IF GE(G1)=ZN THEN GE(G1)=-1:PRINT"OK.
     "                                        <253>
2200 REM INVENTUR ROUTINE                     <208>
2205 IF VE<>22 THEN 2300                      <255>
2210 PRINT"ICH HABE: ";                       <224>
2220 IC=0:FOR I=1 TO GZ:IF GE(I)=-1 THEN P
     RINT GE$(I)",";:IC=1                     <165>
2225 NEXT                                     <059>
2230 IF IC=0 THEN PRINT"NICHTS."              <050>
2235 IF IC=1 THEN PRINT"{LEFT,SPACE}"         <222>
2300 REM VERLIER ROUTINE                      <210>
2305 IF VE<>18 THEN 2400                      <106>
2310 IF GE(G1)<>-1 THEN PRINT"ICH HABE DAS
      NICHT !"                                <006>
2320 IF GE(G1)=-1 THEN GE(G1)=ZN:PRINT"OK.
     "                                        <188>
2400 REM SCHAU - ROUTINE                      <191>
2405 IF VE<>13 THEN 2500                      <202>
2410 IF OB=0 AND G1=0 THEN PRINT"{CLR}":VE
     =0:GOTO 1130                             <208>
2415 IF OB=1 AND OO(1)=ZN THEN PRINT"DIE T
     RUHE IST SEHR GROSS."                    <092>
2500 GOTO 1000                                <023>
10000 REM ------- SPIELKARTE -------           <048>
10100 REM RAUM 1 ---------                    <037>
10102 DATA 0,3,0,2,0,0,0,0,0,0                <145>
10105 PRINT"RAUM NUMMER 1"                    <183>
10120 RETURN                                  <061>
10200 REM RAUM 2 ---------                    <138>
10202 DATA 0,5,1,0,0,0,0,6,0,0                <252>
10205 PRINT"RAUM NUMMER 2"                    <028>
10220 RETURN                                  <161>
10300 REM RAUM 3 ---------                    <240>
10302 DATA 1,0,0,0,0,0,0,0,0,4                <090>
10305 PRINT"RAUM NUMMER 3"                    <130>
10320 RETURN                                  <006>
10400 REM RAUM 4 ---------                    <085>
10402 DATA 0,0,0,0,0,0,0,0,3,0                <188>
10405 PRINT"RAUM NUMMER 4"                    <231>
10420 RETURN                                  <106>
10500 REM RAUM 5 ---------                    <187>
10502 DATA 2,0,0,0,0,0,0,0,0,0                <032>
10505 PRINT"RAUM NUMMER 5"                    <077>
10520 RETURN                                  <207>
10600 REM RAUM 6 ---------                    <032>
10602 DATA 0,0,0,0,2,0,0,0,0,0                <132>
10605 PRINT"RAUM NUMMER 6"                    <178>
10620 RETURN                                  <051>
50000 REM ****************************       <058>
50010 REM *      BEFEHLSEINGABE        *      <228>
50020 REM *      BEFEHLSZERLEGUNG      *      <182>
50030 REM *      BEFEHLSCODIERUNG      *      <173>
50040 REM ****************************       <098>
50050 IF UD=1 THEN 50220                      <236>
50060 POKE 198,0:BE$="":PRINT"WAS NUN ? ";    <044>
50070 POKE 204,0                              <226>
50080 GET X$:IF X$=""THEN 50080               <091>
50090 IF PEEK(203)=1 OR LEN(BE$)>68 THEN P
     RINT" ":POKE 204,1:GOTO 50140            <022>
50100 I=ASC(X$):IF I<65 OR I>90 THEN IF I<
     >32 AND I<>20 AND I<>34 THEN 50080       <173>
50110 IF I=20 AND BE$=""THEN 50080            <093>
50120 IF I=20 THEN POKE 204,1:PRINT"{LEFT,
     2SPACE,2LEFT}";:BE$=LEFT$(BE$,LEN(BE
     $)-1):GOTO 50070                         <174>
50130 PRINT X$;:BE$=BE$+X$:GOTO 50080         <013>
50140 FOR I=1 TO 10:BE$(I)="":NEXT:WZ=1:FO
     R I=1 TO LEN(BE$)                        <154>
50150 :IF MID$(BE$,I,1)=" "THEN GOSUB 5019
     0:GOTO 50180                             <240>
50160 :IF WZ>10 THEN PRINT"EINGABE IST ZU
     LANG !":I=LEN(BE$)+1:GOTO 50180          <028>
50170 :BE$(WZ)=BE$(WZ)+MID$(BE$,I,1)          <069>
50180 NEXT I:GOTO 50220                       <079>
50190 IC=0:FOR I1=1 TO AZ:IF BE$(WZ)=AU$(I
     1)THEN IC=1                              <086>
50200 NEXT I1:IF IC=0 THEN WZ=WZ+1:RETURN     <105>
50210 BE$(WZ)="":RETURN                       <081>
50220 IF UD=1 THEN UD=0:GOTO 50240            <215>
50230 WZ=1:VE=0:OB=0:PE=0                     <163>
50240 IC=0:G1=0:G2=0                          <155>
50250 FOR I=1 TO VZ:IF BE$(WZ)=VE$(I)THEN
     VE=I:IC=1                                <114>
50251 IF LEN(BE$(WZ))<3 THEN 50260            <229>
50255 IF BE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ))
     )THEN VE=I-VAL(RIGHT$(VE$(I),1)):IC=
     1                                        <211>
50260 NEXT I:IF IC=1 THEN 50350               <187>
50270 FOR I=1 TO GZ:IF BE$(WZ)<>GE$(I)THEN
      50300                                   <211>
50280 IC=1:IF G1=0 THEN G1=I                  <212>
50290 G2=I:IF G2=G1 THEN G2=0                 <020>
50300 NEXT I:IF IC=1 THEN 50350               <227>
50310 FOR I=1 TO OZ:IF BE$(WZ)=OB$(I)THEN
     OB=I:IC=1                                <147>
50320 NEXT I:IF IC=1 THEN 50350               <247>
50330 FOR I=1 TO PZ:IF BE$(WZ)=PE$(I)THEN
     PE=I:IC=1                                <176>
50340 NEXT I                                  <051>
50350 IF BE$(WZ)="UND"THEN UD=1:IC=1           <083>
50360 IF IC=0 THEN PRINT"ICH KENNE ";BE$(W
     Z);" NICHT !":RETURN                     <004>
50370 WZ=WZ+1:IF WZ>10 OR BE$(WZ)=""OR UD=
     1 THEN RETURN                            <177>
50380 IC=0:GOTO 50250                         <189>
52000 REM T A B E L L E N                     <161>
52005 RESTORE 52000                           <115>
52010 REM VERBTABELLE ---------               <065>
52020 DATA N,S,W,O,NW,NO,SW,SO,RAUF,RUNTER    <241>
52030 DATA OEFFNE,SCHLIESSE,SCHAUE,UNTERSU
     CHE1,NIMM,NEHME1,HOLE2                   <003>
52035 DATA VERLIERE,LEGE1,WIRF2,WERFE3,INV
     ENTUR                                    <126>
52100 VZ=22:DIM VE$(VZ):FOR I=1 TO VZ:READ
      VE$(I):NEXT                             <166>
52200 REM GEGENSTANDSTABELLE ---------        <003>
52210 DATA SCHWERT,1                          <189>
52211 DATA SEIL,2                             <204>
52212 DATA SCHLUESSEL,4                       <157>
52213 DATA DIAMANT,5                          <162>
52300 GZ=4:DIM GE$(4):DIM GE(4):FOR I=1 TO
      GZ:READ GE$(I):READ GE(I):NEXT I        <136>
52400 REM OBJEKTTABELLE ---------             <088>
52410 DATA TRUHE,5                            <242>
52412 DATA SCHACHT,6                          <107>
52414 DATA EISENRING,6                        <019>
52416 DATA TUER,2                             <173>
52418 DATA TUER,5                             <178>
52500 OZ=5:DIM OB$(OZ):DIM OO(OZ):FOR I=1
     TO OZ:READ OB$(I):READ OO(I):NEXT I      <122>
53000 RETURN                                  <101>
```

© 64'er

Listing 22 (Schluß)

Action in Räumen

Die raumspezifische Action bereitet meistens den größten Programmieraufwand, und bringt somit auch das größte Chaos in die anfangs gut strukturierten Listings Uns soll dies nicht passieren.

Wo bringt man die raumspezifische Action denn am besten im Listing beziehungsweise im Programm unter?

Die einfachste Lösung wäre es, die raumspezifische Action einfach an das Actionmodul anzuhängen. Man würde sich so für jeden Raum zirka 100 Zeilen reservieren (zum Beispiel 2500 bis 2599 für Raum 1) und vor jedem Raum eine IF-THEN-Abfrage stellen, die feststellt, ob der Raum übersprungen oder behandelt werden soll. Dieses Verfahren hat jedoch einen großen Verzögerungseffekt auf die Bearbeitungsgeschwindigkeit des Adventures zur Folge.

Um diesem und allen anderen Problemen auszuweichen, schreiben wir unsere raumspezifische Action direkt auf die Spielkarte. Dies geht ganz einfach:

Betrachten wir zunächst einmal erneut den Aufbau der Spielkarte und nehmen als Beispiel Raum 1.

Raum 1 liegt in den Zeilen 10100 bis maximal 10199.

Am Kopf des Raumes steht eine DATA-Zeile, die die möglichen Richtungen und ihre Zielorte zu diesem Raum enthält.

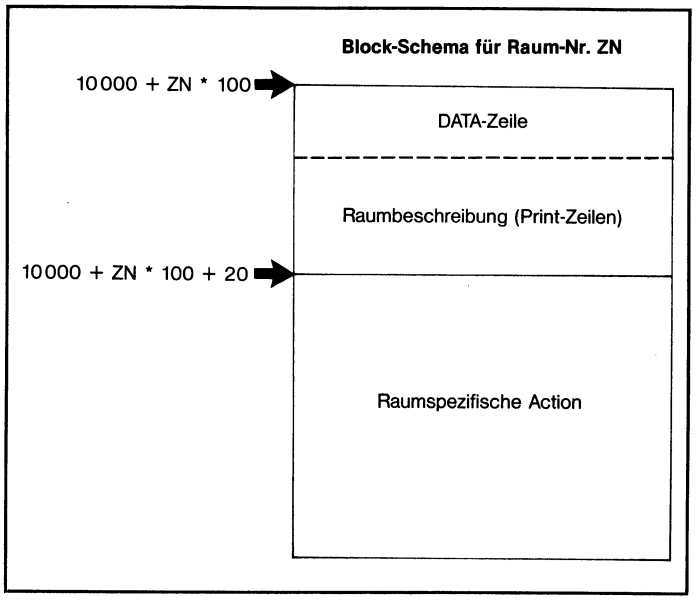
Die DATA-Zeile wird mittels
RESTORE 10000 + ZN*100
und einer READ-Schleife gelesen.

Danach wird die Raumbeschreibung mittels GOSUB 10000 + ZN*100 ausgegeben.

Die Raumbeschreibung steht dabei in PRINT-Zeilen von 10105 bis maximal 10119.

Was liegt nun näher, als die raumspezifische Action einfach in die Zeile von 10120 bis maximal 10199 zu legen?

Das allgemeine Ablaufschema für einen einzelnen Raum der Spielkarte samt Action sieht nun so aus:



Aus diesem Schema geht hervor, daß die raumspezifische Action zum Raum ZN (ZN = Zimmernummer der Karte) mittels
GOSUB 10000 + ZN*100 + 20
aufgerufen wird.

Um die Action des Raumes 1 aufzurufen, müßte man also
GOSUB 10000 + 1*100 + 20,
was identisch mit
GOSUB 10120 ist, eingeben.

Die raumspezifische Action wird immer am Ende des Actionmoduls aufgerufen. Ergänzen Sie Ihr Programm nun bitte um Listing 23.

```
2500 REM AUFRUF DER RAUMSPEZIFISCHEN ACTIO
     N                                    <126>
2510 GOSUB 10000+ZN*100+20                <240>
2600 GOTO 1000                            <124>
```

Listing 23

Damit ist eine übersichtliche Methode gegeben, unser Adventure mit jeder nur denkbaren Art von Action zu versehen. Der Vorteil der vorgestellten Methode liegt hauptsächlich darin, daß Fehler leicht beseitigt werden können, sofern sie im Spielverlauf auftreten.

Angenommen, wir sind in Raum 1 und irgendetwas funktioniert nicht, so muß man nur Zeile 10100 bis 10199 auflisten und kann sofort gezielt nach dem Fehler suchen.

Auf den folgenden Seiten finden Sie nun noch einige Beispiele zum Einbau und zur Programmierung von raumspezifischer Action.

Programmierung einer Tür

Unser Ziel soll nun sein, eine Tür zwischen den Räumen 2 und 5 zu programmieren.

Bisher kann man einfach von Raum 2 mittels der Eingabe »S« zu Raum 5 gelangen. Die Tür wird jedoch bereits in der Raumbeschreibung erwähnt, da sie in der Objekttabelle vertreten ist.

Zu bemerken ist hier, daß jede Tür immer zweimal in der Objekttabelle vertreten sein muß — denn man kann eine Tür immer von zwei Räumen aus sehen — den Räumen, die durch die Tür getrennt werden.

Vergleichen Sie bitte auch in Ihrem Listing die Zeilen 52416 und 52418. Der erste Schritt zur Programmierung der Tür besteht darin, den Verbindungsweg zwischen den beiden Räumen, zwischen die die Tür später soll, zunächst zu entfernen.

Für die Tür in unserem Beispiel (Raum 2 und 5) müssen wir die Richtungs-DATA-Zeilen der beiden Räume folgendermaßen ändern:

10202 DATA 0,0,1,0,0,0,0,6,0,0
10502 DATA 0,0,0,0,0,0,0,0,0,0

Damit ist die Verbindung zwischen Raum 2 und Raum 5 auch schon unterbrochen. Nun brauchen wir noch eine Variable, die den Zustand der Tür bestimmt.

Wir unterscheiden zwischen drei Zuständen:

1. Die Tür ist offen. Der Variablenwert zur Tür sei dann 0.
2. Die Tür ist zu und kann mittels »OEFFNE TUER« geöffnet werden. Der Variablenwert sei hierbei 1.
3. Die Tür ist zu und kann nur mit einem Hilfsmittel (zum Beispiel Schlüssel) geöffnet werden. Hierbei sei die Variable 2.

Unsere Türvariable soll TU(1) sein. Das Feld TU(1) bis TU(X) ist für die Zustände aller Dinge verantwortlich, die man öffnen und schließen kann.

Unsere Tür soll zunächst ohne Schlüssel geöffnet werden können. Die Türvariable TU(1) hat folglich den Wert 1.

Bitte ergänzen Sie das Listing nun um folgende Zeilen:

52900 REM ALLGEMEINE VARIABLEN
52910 TU(1) = 1 : REM TUER 2/5

Jetzt müssen wir die raumspezifische Action in den Räumen 2 und 5 zum Öffnen und Schließen der Tür programmieren.

Dies geht so:

10220 IF TU(1)=0 THEN RI(2)=5
10225 IF VE=11 AND OB=5 AND TU(1)=1
THENPRINT"OK.":TU(1)=0: RI(2)=5

```
10230  IF VE=12 AND OB=5 AND TU(1)=0
       THENPRINT"OK.":TU(1)=1: RI(2)=0
10250  RETURN
```

Damit unser Programm jedoch einwandfrei läuft, müssen wir im Actionmodul selbst erst noch eine kleine Änderung vornehmen: Bitte fügen Sie Zeile 1130 REM und 1155 GOSUB 10000+ZN*100 in das Programm ein. Eine Erklärung für diese Änderung folgt gleich. Nun die Erklärung zur Türprogrammierung in Raum 2:

10220 In dieser Zeile wird der Zustand der Tür überprüft. Wird dabei festgestellt, daß die Tür offen ist, so gilt RI(2)=5. Wie bereits festgelegt, ist RI(2) für die Richtung SUEDEN zuständig. Dadurch, daß die Tür offen ist, muß RI(2) also auf den Wert 5 gesetzt werden, was nichts anderes heißt, als daß man durch die Tür zu Raum 5 gelangt. Der ganze Trick der Türprogrammierung besteht darin, daß man zunächst den Verbindungsweg zwischen den Räumen entfernt, und im Falle einer offenen Tür einfach wieder herstellt. Diese Wiederherstellung erfolgt in dieser Zeile.

Hierduch läßt sich auch begründen, warum wir das Actionmodul ein wenig verändern mußten: Wenn der Spieler einen neuen Raum betritt, geschieht folgendes:

1. Der DATA-Zeiger wird mittels RESTORE 10000+ZN*100 auf die DATA-Zeile des neuen Raumes gesetzt.
2. Nun werden die Richtungsmöglichkeiten RI(1) bis RI(10) mittels READ-Befehl eingelesen.
3. Sodann wird mittels GOSUB 10000 +100*ZN die Raumbeschreibung aufgerufen, bei der gegebenenfalls (Tür ist offen) auch der Verbindungsweg wieder hergestellt wird.

Die Änderung des Actionmoduls bestand also lediglich in der Vertauschung der Arbeitsschritte 2 und 3 auf die jetzige Reihenfolge.

Nur bei der jetzigen Ordnung funktioniert das Programm einwandfrei.

10225 Hier wird geprüft, ob die Befehlseingabe des Spielers »OEFFNE TUER« lautet. Man schaut dazu einfach nach, ob VE=11 und OB=5 ist. Außerdem wird geprüft, ob die Tür zu ist, denn eine offene Tür kann man logischerweise nicht öffnen.

Stimmt alles, so wird TU(1) nun auf den Wert 1 (Tür ist offen) gesetzt, und der Verbindungsweg zu Raum 5 hergestellt: RI(2)=5.

10230 Analog zu 10225, nur das die Tür hier geschlossen wird.

Achtung: Sicher haben Sie sich bereits gewundert, wie die Abfrage, ob im Befehlssatz des Spielers die TUER vorkommt, erfolgt. Die Abfrage hierfür lautete einfach IF OB=5 THEN... Warum muß OB gleich 5 und nicht gleich 4 sein?

Lassen Sie uns einmal einen Blick auf die Objekttabelle werfen:

```
OB$(1)="TRUHE"        OO(1)=5
OB$(2)="SCHACHT"      OO(2)=6
OB$(3)="EISENRING"    OO(3)=6
OB$(4)="TUER"         OO(4)=2
OB$(5)="TUER"         OO(5)=5
```

Wie Sie bereits wissen, stammt der OB-Wert aus dem Befehlsanalysemodul. Sie erinnern sich sicher auch noch daran, wie das Modul den OB-Wert erhält:

Es durchläuft mit einer Schleife einfach die Objekttabelle:

```
FOR I=1 TO OZ : IFBE$(WZ) = OB$(I) THEN OB=I
```

Somit wird klar, warum der erhaltene OB-Wert 5 und nicht 4 ist. Wir müssen also darauf achten, daß für Objekte, die den gleichen Namen haben, immer der OB-Wert des Objekts ausgegeben wird, welches in der Tabelle zuletzt gefunden wird beziehungsweise, das in der Tabelle den höchsten Wert hat.

Nun müssen wir natürlich noch den Raum 5 mit der spezifischen Action versehen, die auch in Raum 2 enthalten ist (was das Öffnen und Schließen der Tür anbelangt).

Bitte ergänzen Sie mit Listing 24.

```
10520 IF TU(1)=0 THEN RI(1)=2                    <129>
10525 IF VE=11 AND OB=5 AND TU(1)=1 THEN P
      RINT"OK.":TU(1)=0:RI(1)=2                  <050>
10530 IF VE=12 AND OB=5 AND TU(1)=0 THEN P
      RINT"OK.":TU(1)=1:RI(1)=0                  <054>
10550 RETURN                                     <237>
```

Listing 24

Die Programmierung der Tür könnte nun eigentlich als abgeschlossen betrachtet werden. Allerdings ist es erstrebenswert, dem Spieler die Möglichkeit zu geben, mittels »GEH TUER« durch die Tür zu gelangen.

Bisher muß man nach dem Öffnen der Tür noch den Befehl »SCHAU« eingeben, um zu erfahren, welche weitere Richtung sich durch das Öffnen der Tür ergeben hat. Dazu müssen wir die Verbtabelle wieder um das Verb »GEH« erweitern:

```
52030 DATA GEHE,BETRETE1
```

und in Zeile 52100 den VZ-Wert auf 24 erhöhen.

Nun kann die raumspezifische Action in den Räumen 2 und 5 erweitert werden:

```
10240 IF VE=23 AND OB=5 THEN VE=2
10540 IF VE=23 AND OB=5 THEN VE=1
```

Nebenbei muß auch das Actionmodul noch so ergänzt werden, damit diese Routinen angenommen werden:

```
1110  IFRI(VE) = 0 THENPRINT"KEIN WEG IN DIESE
      RICHTUNG !": VE = 0:GOTO1200
1130  VE=0
2520  IFVE>0 AND VE<11 THEN1100
```

Wenn Sie wissen möchten, warum diese Änderungen unbedingt notwendig sind, dann spielen Sie das Adventure einmal vor und einmal nach der Änderung des Actionmoduls durch (die Tür-Szene).

Bisher kann die Tür noch problemlos geöffnet werden.

Nun wollen wir das Öffnen und Schließen der Tür von einem Schlüssel abhängig machen — die Tür kann nur dann noch geöffnet werden, wenn der Spieler im Besitz des Schlüssels ist.

Viele englische Adventures verlangen als Befehlssyntax zum Öffnen einer verschlossenen Tür mit einem Schlüssel etwa folgendes:

```
ENTRIEGLE TUER
OEFFNE TUER
GEH TUER
```

Im Deutschen ist es allerdings nicht besonders üblich »entriegle Tür« zu sagen. Wir wollen deshalb den ersten Schritt weglassen, und folgenden Ablauf vereinbaren:

— Wenn der Spieler die Tür mit »OEFFNE TUER« öffnen will, jedoch keinen passenden Schlüssel bei sich hat, so erhält er die Meldung »ICH HABE KEINEN PASSENDEN SCHLUESSEL«.

— Hat er einen Schlüssel bei sich und gibt als Befehl wieder »OEFFNE TUER« ein, so erhält er die Meldung »OK.«.

Es ist pure Haarspalterei, vom Spieler Eingaben wie zum Beispiel »OEFFNE DIE TUER MIT DEM SCHLUESSEL« beziehungsweise »ENTRIEGLE DIE TUER MIT DEM SCHLUESSEL« und dann »OEFFNE TUER« zu verlangen.

Wichtig ist nur die Tatsache, daß der Spieler einen Schlüssel gefunden haben muß, um durch die Tür zu gelangen.

Die Routine für die Schlüsselabfrage ist ganz einfach zu programmieren: Wir müssen vor der Tür-öffnen-Routine einfach eine Abfrage einbauen, die feststellt, ob GE(3) den Wert —1 hat, also, ob der Schlüssel im Besitz des Spielers ist.

Dazu sind nur zwei neue Ergänzungszeilen aus Listing 25 notwendig.

```
10224 IF VE=11 AND OB=5 AND GE(3)<>-1 THEN
       PRINT"ICH HABE KEINEN SCHLUESSEL.":
      GOTO 10230                          <123>
10524 IF VE=11 AND OB=5 AND GE(3)<>-1 THEN
       PRINT"ICH HABE KEINEN SCHLUESSEL.":
      GOTO 10230                          <169>
```

Listing 25

Wenn Sie Ihr Programm ergänzt haben, dann machen Sie gleich einmal ein Probespiel und probieren das Öffnen der Tür mit dem Schlüssel aus.

Spielen Sie auch einmal die folgende Variante durch:

1. Schlüssel holen
2. Tür in Raum 2 öffnen.
3. Schlüssel verlieren.
4. In Raum 5 gehen.
5. Tür schließen.

Sie sind nun in Raum 5 und können diesen nicht mehr verlassen, da das Türschloß zugeschnappt ist, und der Schlüssel sich in Raum 2 befindet. Man kann den Spieler also in eine Falle laufen lassen.

Interessant hierbei ist, daß kaum ein Spieler eine Tür wieder schließt, nachdem er sie erst einmal geöffnet hat. Die Programmierung dieses Effekts (die in der bisherigen Tür-Logik bereits enthalten ist) ist allein dadurch interessant, daß der Spieler überrascht wird.

Die Truhe

Wenn Sie sich in Raum 5 begeben, werden Sie feststellen, daß sich dort eine Truhe befindet.

Transportieren kann man die Truhe nicht, da wir bereits festgelegt haben, daß Objekte vom Spieler nicht transportiert werden können (eventuell, weil sie dazu zu schwer sind).

Lassen Sie uns zunächst noch eine Fehlermeldung in unser Adventure einbauen — wenn der Spieler versucht ein Objekt zu nehmen (Tür, etc.), so erhält er die Antwort »DAS GEHT UEBER MEINE KRAEFTE!«.

Diese Fehlermeldung läßt sich ganz einfach in die NIMM-Routine einbauen:

```
2115 IFOB<>0THEN PRINT"DAS GEHT UEBER
     MEINE KRAEFTE !":GOTO2200
```

Nun zur Truhe speziell:

In der Truhe können selbstverständlich Gegenstände liegen. Für die Gegenstände haben wir bisher folgendes definiert: Wenn GE(X) größer als Null ist, so liegt der Gegenstand in diesem Raum.

Wenn GE(X) den Wert —1 hat, so ist dieser Gegenstand im Besitz des Spielers.

Nun ergänzen wir mit folgender Bedingung:

Wenn GE(X) den Wert —2 hat, so liegt der Gegenstand in der Truhe.

Wir wollen unsere Gegenstandstabelle nun so umändern, daß das Schwert nicht mehr wie bisher in Raum 1 liegt, sondern sich zukünftig in der Truhe befindet, also:

```
52210 DATASCHWERT,—2
```

Nun liegt das Schwert in der Truhe.

Wie kommen wir im Verlauf des Spiels jedoch wieder zum Schwert?

Nun ganz einfach, wir müssen die Truhe öffnen. Dazu ist es erforderlich, in die raumspezifische Action von Raum 5 eine Routine einzubauen, die das OEFFNEN der TRUHE erlaubt. Im Prinzip können wir die Truhe wie eine Tür betrachten, denn auch bei der Truhe sind drei Zustandsformen möglich (offen, zu, verriegelt).

Wir benötigen also wieder eine Variable, die Auskunft über den jeweiligen Zustand der Truhe gibt. Diese Variable sei TU(2) und der Ausgangszustand der Truhe sei 1 (also Truhe ist zu und kann mittels OEFFNEN ohne irgendeinem Hilfsmittel a la Schlüssel geöffnet werden).

Wir ergänzen:

```
52920 TU(2)=1:REM TRUHE
```

Nun müssen wir die OEFFNE-Routine für Raum 5 programmieren. Dies geht ganz einfach:

```
10545 IF VE=11 AND OB=1 AND TU(2)=1
      THENPRINT"OK.":TU(2)=0
```

Was man öffnen kann, kann man auch schließen, also:

```
10546 IF VE=12 AND OB=1 AND TU(2)=0
      THENPRINT"OK.":TU(2)=1
```

Sie sehen schon, daß die meisten Routinen relativ einfach programmiert werden können, da wir zu Beginn viel Arbeit und Gedanken in die Programmierung des Befehlsanalysemoduls gesteckt haben. Diese Arbeit macht sich nun bezahlt, denn wir können nun jegliche Routine programmieren, und müssen dabei nur den Wortschatz entsprechend erweitern, in der Routine selbst nur VERBZAHL, OBJEKTZAHL etc. abfragen, und danach TUER-Variablen etc. entsprechend verändern.

So schön, so gut!

Wir können die Truhe nun öffnen und schließen, aber vom Schwert ist noch keine Spur zu sehen. Dazu müssen wir die Raumbeschreibungs-Routine erweitern. Die Gegenstände, die sich in der Truhe befinden, sollen künftig auch in der Raumbeschreibung erwähnt werden, unter der Bedingung einer geöffneten Truhe. Dazu genügt wieder eine einzige Abfrage:

```
1189 IFGE(I)=—2ANDTU(2)=0ANDZN=5THENPRINT
     GE$(I);",";:IC=1
```

Diese Abfrage steht innerhalb der Schleife, in der auch geprüft wird, ob sich im betreffenden Raum ein Gegenstand befindet.

In der Truhenabfrage muß also berücksichtigt werden, daß der Gegenstand nur dann in der Raumbeschreibung vorkommen darf, wenn der Spieler sich in Raum 5 (wo die Truhe steht) befindet, wenn die Truhe offen ist, und wenn in der Truhe Gegenstände liegen (also deren GE-Wert gleich —2 ist).

Wenn Sie nun in Raum 5 gehen und mittels »OEFFNE TRUHE« die Truhe öffnen, und danach den Befehl »SCHAU« eingeben, so wird das Schwert sichtbar. Wenn Sie die Truhe jedoch wieder schließen, so verschwindet das Schwert auch wieder aus der Raumbeschreibung.

Das Schwert ist nun sichtbar. Wenn Sie jedoch versuchen, sich das Schwert mit »NIMM SCHWERT« anzueignen, so werden Sie mit der Fehlermeldung »ICH SEHE DIESEN GEGENSTAND HIER NICHT!« enttäuscht. Unser Programm fragt nämlich nur, ob der Gegenstandswert des Gegenstands, den man nehmen will, der Zimmernummer ZN entspricht (also GE(G1) gleich ZN ist.).

Wir müssen nun eine Abfrage einbauen, die es dem Spieler erlaubt auch Gegenstände zu nehmen, die in der Truhe liegen, wenn diese offen ist, und wenn wir uns in dem Raum befinden, in dem die Truhe steht.

Diese Abfrage muß natürlich wieder in der raumspezifischen Action zu Raum 5 vorkommen. Wir benötigen wiederum nur eine Zeile:

```
10548 IFVE=15ANDGE(G1)=—2ANDTU(2)=0THEN
      PRINT"OK.":GE(G1)=—1
```

Damit können auch Gegenstände aus der Truhe herausgenommen werden.

Allerdings erfolgt immer noch die Fehlermeldung »ICH SEHE...«.

Um diesen Fehler zu beseitigen, stellen wir der Zeile, in der diese Fehlermeldung produziert wird, folgende Zeile voran:

```
2119 IFGE(G1)=—2ANDZN=5ANDTU(2)=0
THEN2125
```

Damit ist das Problem auch schon gelöst.

Wir sind jedoch noch nicht am Ende der Truhen-Programmierung angelangt. Aus einer Truhe, aus der man etwas herausnehmen kann, muß man auch etwas hineinlegen können.

Gerade solche Kleinigkeiten sind es letztendlich, die ein gutes Adventure ausmachen.

Um das »LEGEN« von Gegenständen in die Truhe möglich zu machen, müssen wir eine VERLIER-Routine erstellen, die auf Objekte bezogen ist.

Gemäß unserer Worttabelle besteht die VERLIER-Wortfamilie aus folgenden Mitgliedern: VERLIERE, LEGE, WIRF, WERFE.

Wir nehmen nun einmal an, der Spieler steht in Raum 5 und hat die Truhe bereits geöffnet. Wenn er den Befehl »VERLIER« und Gegenstand eingibt, so nimmt dieser Gegenstand den Wert ZN an (GE(G1)=ZN) und liegt nun in Raum 5.

Wir müssen nun eine Routine programmieren, die dem Gegenstand bei einer Eingabe wie »LEGE (GEGENSTAND) IN TRUHE« den Wert —2 verpaßt.

Dies bringen wir wieder in der raumspezifischen Action zu Raum 5 unter:

```
10550 IFVE=18ANDOB=1ANDTU(2)=0ANDGE(G1)
=—1THENPRINT"OK.":GE(G1)=—2
10590 RETURN
```

Zusätzlich muß noch eine Abfrage in die Verlier-Routine eingebaut werden, die dafür sorgt, daß die normale Verlier-Routine übersprungen wird, wenn der VERLIER-Befehl sich auf ein Objekt wie die Truhe bezieht:

```
2301 IFOB<>0THEN2400
```

Wenn der Spieler die Kiste beziehungsweise die Truhe untersucht, so erfährt er, daß sie sehr groß ist.

Der Spieler kommt nun vielleicht auf die Idee, in die Truhe zu gehen. Auch dies wollen wir ihm ermöglichen.

Dazu müssen wir die Truhe zunächst als Raum programmieren. Wir nehmen dazu Raum 7 — also ab Zeile 10700:

```
10700 REM IN DER TRUHE ————
10702 DATA 0,0,0,0,0,0,0,0,0,0
10705 PRINT"IN DER TRUHE."
10720 RETURN
```

Um in die Truhe zu gelangen, muß der Spieler in Raum 5 den Befehl »GEH TRUHE« eingeben, aber erst, nachdem er die Truhe geöffnet hat.

Dazu ergänzen wir die raumspezifische Action in Raum 5:

```
10560 IFVE=23ANDOB=1ANDTU(2)=0THENRI(1)=
      7:VE=1
```

Wie Sie aus dieser Zeile ersehen können, besteht der Trick des GEH-Befehls also einfach darin, in die Richtung RI(1) den Wert des Zielraums zu schreiben (für die Truhe also 7), und danach die Verbzahl VE auf den Wert 1 zu setzen, wodurch das GEHEN bewirkt wird.

Einmal in der Truhe, werden Sie enttäuscht feststellen, daß Sie das Schwert überhaupt nicht sehen können. Wenn Sie in der Truhe einen Gegenstand verlieren, so läßt sich dieser nur dann wieder nehmen, wenn Sie dazu in die Truhe gehen.

Dies liegt daran, daß wir zuvor bestimmt haben, daß jeder Gegenstand, der sich in der Truhe befindet, den Wert —2 bekommt. Da die Truhe nun aber selbst zu einem Raum geworden ist, müssen alle Gegenstände, die sich in ihr befinden den Wert 7 haben, da 7 der Raumwert der Truhe ist.

Wenn Sie den Fehler also beheben wollen, so müssen Sie lediglich alle —2-Werte in den Wert 7 umwandeln.

Damit die Truhe auch wieder verlassen werden kann, müssen wir die raumspezifische Action der Truhe programmieren:

Der Wortschatz wird zunächst um das Verb VERLASSE erweitert:

```
52045 DATA VERLASSE
52100 VZ=25.....(VZ-Wert anpassen!)
```

Nun die Action zum Verlassen der Truhe in Raum 7:

```
10720 IFVE=25ANDOB=1ANDTU(2)=0THENRI(1)=
      5:VE=1
```

Diese Abfrage ist lediglich die Umkehrung zur Abfrage in der die Truhe betreten wird. Sonderbar ist jedoch, daß auch abgefragt wird, ob die Truhe offen ist, wenn man sie verlassen will.

Schließlich läßt sich die Truhe doch nur dann betreten, wenn man sie geöffnet hat, und wenn man erst einmal in der Truhe ist, dann kann man sie von dort aus doch nicht schließen! Wozu also die Abfrage, ob die Truhe auch offen ist? Wer sollte sie denn schließen?

Damit kommen wir auch schon zum nächsten Abschnitt des Kurses:

Ein Gespenst geht um ...

Wie ich bereits zu Beginn des Kurses erwähnt habe, sind Adventures mit Nichtspielercharakteren — Spielfiguren, die vom Programm selbst gesteuert werden — besonders reizvoll.

Da die Programmierung solcher Spielfiguren oft für sehr schwer gehalten wird, treten nur in sehr wenigen Spielen solche Figuren auf.

Ein Meisterbeispiel für solche Adventures ist »Hobbit«. Tatsächlich ist es jedoch relativ einfach, Nichtspielercharaktere zu programmieren.

Das erste Problem besteht darin, einen Weg zu finden, wie der Nichtspielercharakter (in unserem Falle das Gespenst) in der Spielkarte umherlaufen kann. Eine Möglichkeit wäre es, das Gespenst per Random (zufallsgesteuert) von Raum zu Raum irren zu lassen. Diese Lösung erweist sich auf die Dauer jedoch als zu primitiv, da die Gefahr groß ist, daß das Gespenst sich in einer Sackgasse verfängt und dort sehr lange umherirrt. Dadurch trifft der Spieler nur äußerst selten auf das Gespenst, was nicht der Fall sein soll. Das Gespenst soll dem Spieler oft in die Quere geraten.

Eine weitere, primitive Lösung wäre es, das Gespenst erst gar nicht herumlaufen zu lassen, sondern es einfach per Zufall gesteuert plötzlich im Raum des Spielers auftauchen zu lassen.

Folgende Lösung hat sich in meinen Adventures bisher bestens bewährt: Zunächst wird das Gespenst einmal in die Personentabelle aufgenommen:

```
52600 REM PERSONENTABELLE ————
52610 DATA GESPENST
52700 PZ=1:DIMPE$(PZ):FORI=1TOPZ:READPE$(I):
      NEXT
```

Als nächster Schritt wird eine genaue Route festgelegt, die das Gespenst später ablaufen soll.

Für unser Adventure eignet sich die folgende Route:

1—3—4—3—1—2—5—2—6—2—1

Diese Zahlen beziehen sich selbstverständlich auf die einzelnen Räume. Das Gespenst beginnt also in Raum 1, geht dann nach Raum 3 etc.

Wichtig ist dabei nur, daß der letzte Raum der Kette wieder dem ersten Raum entspricht, denn wir wollen es auch unserem Gespenst nicht erlauben, durch Wände von Raum zu Raum zu gelangen.

Das Gespenst läuft also immer wieder die gleiche Route ab. Dies hört sich zwar primitiv an, aber ich kann Ihnen versichern, daß dem Spieler kaum auffallen wird, daß das Gespenst sich immer nach dem gleichen Schema fortbe-

wegt. Nun gut, bei unserem Mini-Adventure ist es vielleicht nicht sehr schwer, die Taktik des Gespenstes zu durchschauen, aber bei Spielen mit 100 und mehr Räumen, was natürlich mit längeren Routen verbunden ist, besteht kaum eine Chance, die Route eines Nichtspielercharakters herauszufinden — vorausgesetzt, man hat sie nicht selbst programmiert.

Ein weiterer Vorteil der Routenprogrammierung liegt darin, daß man die Gebiete der einzelnen Figuren gut begrenzen kann. Die Route wird nun durch das Feld PE(1) bis PE(11) festgelegt.

```
52920 DATA1,3,4,3,1,2,5,2,6,2,1
52935 DIMPE(11):FORI=1TO11:READPE(I):
      NEST:MO=1
```

Die Steuerung des Gespenstes soll innerhalb des Actionmoduls ab Zeile 3000 beginnen:

```
3000  REM STEUERUNG DES GESPENSTES
3001  PRINT"GESPENST=";PE(MO)
3010  IFMO=0THENRETURN
3020  MO=MO+1:IFMO=12THENMO=1
3025  IFPE(MO)<>ZNTHEN3100
3100  RETURN
```

Aufgerufen wird dieses Unterprogramm noch vor dem Aufruf der raumspezifischen Action in Zeile 2505:

```
2505  GOSUB 3000 : REM GESPENST
```

Zeile 3001 dient lediglich zum Verfolgen des Gespenstes. Sie kann später nach Belieben wieder entfernt werden.

Die Variable MO läuft von 1 bis 11 durch und beginnt dann wieder von vorne: PE(MO) ist somit der Raum, in dem sich das Gespenst gerade befindet. Ist MO=0, dann ist das Gespenst abgeschaltet (durch Zeile 3010).

Nach jedem Spielzug des Spielers wird die Variable MO um 1 erhöht — das Gespenst bewegt sich.

Damit der Spieler auch bemerkt, wenn das Gespenst den Raum betritt, müssen wir noch folgende Zeile einfügen:

```
3030 PRINT" EIN RIESIGES GESPENST ERSCHEINT !"
```

Bisher ist unser Gespenst ja noch ziemlich harmlos. Wir wollen ihm deshalb folgende Aufgabe stellen:

Wenn das Gespenst einen Raum betritt, in dem ein Gegenstand liegt, so soll es diesen mitnehmen. Es darf jedoch immer nur einen Gegenstand transportieren, kann allerdings den Gegenstand auch jederzeit gegen einen anderen austauschen. Geben Sie zunächst folgende Ergänzung ein — die Erklärung dazu folgt anschließend:

```
3100  IC=0:FORI=1TOGZ:IFGE(I)=
      PE(MO)THENIC=I
3105  NEXT
3110  IFIC=0THEM3150
3120  GE(GF)=PE(MO):GF=IC:GE(IC)=0
3150  RETURN
```

Auch diese Routine läßt sich leicht erklären.

Zunächst wird eine Schleife durchlaufen, in der geprüft wird, ob sich in dem Raum, in welchem sich das Gespenst gerade befindet, auch ein Gegenstand liegt. Trifft dies zu, so legt das Gespenst den Gegenstand, den es momentan bei sich trägt ab (GE(GF)=PE(MO)), und nimmt den neuen Gegenstand mit (GF=IC). Der Gegenstand, der im Besitz des Gespenstes ist, muß immer auf Null gesetzt werden (GE(IC)=0). Die Variable GF dient lediglich als Zwischenspeicher für die genommenen Gegenstände.

Das Gespenst kann nun also Gegenstände transportieren. Nun wollen wir noch einen weiteren Effekt einbauen:

```
3150  IFPE(MO)=5ANDZN=7ANDTU(2)=0THEN
      TU(2)=1:PRINT"JEMAND SCHLIESST DIE
      TRUHE"
3160  RETURN
```

Wenn das Gespenst nun Raum 5 betritt und der Spieler sich gleichzeitig in der Truhe befindet, so schließt das Gespenst die Truhe, und der Spieler ist gefangen.

Auf den letzten Seiten sollten Sie anhand der Beispiele gelernt haben, wie man ein Adventure mit Action versieht. Im Prinzip besteht jede Actionhandlung nur aus einfachen IF-THEN-Abfragen und einem anschließenden Spiel und Verändern von verschiedenen Variablen.

Sie sollten nun in der Lage sein, jede beliebige Actionszene zu programmieren. Es sollten hier nur die Prinzipien an einem einfachen Beispiel gezeigt werden. Welcher Art die Nichtspielcharakter sind, ihre Route und Fähigkeiten, liegt im Ermessen des jeweiligen Programmierers.

Als abschließende Übung zur Überprüfung Ihres Lernerfolgs empfehle ich diese Variante:

In Raum 6 befindet sich ein Schacht und ein Eisenring im Boden.

Programmieren Sie nun folgendes:

Durch den Schacht gelangt man zu Raum 8. Dazu muß man das Seil am Eisenring befestigen und anschließend hinabklettern. Wenn Sie dieses Problem gelöst haben, können Sie sich durchaus daran machen, eigene Probleme zu stellen,

```
0 REM ****************************          <081>
1 REM * ADVENTURE-PROGRAMMIERKURS *          <073>
2 REM *                           *          <229>
3 REM *     UEBUNGS-PROGRAMM      *          <145>
4 REM ****************************          <085>
9 REM BASIC-ERWEITERUNG                      <120>
10 DATA 76,24,1,177,251,145,251,200,208,24
   9,230,252,202,208,244,96,120,160,0,169    <104>
12 DATA 160,132,251,133,252,162,32,32,11,1
   ,169,224,132,251,133,252,162,32,32,11     <025>
14 DATA 1,169,53,133,1,88,96:FOR I=264 TO
   310:READ X:POKE I,X:NEXT:SYS 264          <234>
16 FOR I=710 TO 730:READ X:POKE I,X:NEXT     <213>
18 DATA 208,3,76,29,168,32,192,2,32,19,166
   ,56,165,95,233,1,164,96,76,36,168         <165>
20 POKE 40996,197:POKE 40997,2:POKE 1,54     <087>
22 FOR I=43168 TO 43170:READ X:POKE I,X:NE
   XT                                        <174>
24 FOR I=704 TO 709:READ X:POKE I,X:NEXT     <230>
26 DATA 32,192,2,32,138,173,76,247,183       <114>
30 GOSUB 52000: REM TABELLEN DEFFINIEREN     <209>
100 ZN=1:GOTO 1130                           <119>
1000 REM A C T I O N M O D U L               <185>
1010 GOSUB 50000: REM BEFEHLSEINGABE         <036>
1100 REM GEHEN IN EIN NEUES ZIMMER           <013>
1105 IF VE<1 OR VE>10 THEN 1200              <038>
1110 IF RI(VE)=0 THEN PRINT"KEIN WEG IN DI
     ESE RICHTUNG !":VE=0:GOTO 1200          <159>
1120 ZN=RI(VE):PRINT"{CLR}"                  <239>
1130 VE=0                                    <235>
1140 RESTORE 10000+ZN*100                    <132>
1150 FOR I=1 TO 10:READ RI(I):NEXT           <230>
1155 GOSUB 10000+ZN*100                      <148>
1160 PRINT"{DOWN}MOEGLICHE RICHTUNGEN : ";   <115>
1165 IC=0:FOR I=1 TO 10:IF RI(I)<>0 THEN P
     RINT VE$(I);",";:IC=1                   <161>
1170 NEXT I                                  <097>
1175 IF IC=0 THEN PRINT"KEINE."              <178>
1180 IF IC=1 THEN PRINT CHR$(20)             <084>
1185 PRINT"ICH SEHE : ";                     <240>
1186 IC=0:FOR I=1 TO OZ:IF OO(I)=ZN THEN P
     RINT OB$(I);",";:IC=1                   <189>
1187 NEXT                                    <041>
1188 FOR I=1 TO GZ:IF GE(I)=ZN THEN PRINT
     GE$(I);",";:IC=1                        <248>
1189 IF GE(I)=-2 AND TU(2)=0 AND ZN=5 THEN
      PRINT GE$(I);",";:IC=1                 <253>
1190 NEXT                                    <044>
1192 IF IC=0 THEN PRINT"NICHTS BESONDERES.
     "                                       <010>
1194 IF IC=1 THEN PRINT CHR$(20)             <098>
1200 REM                                     <067>
2000 REM REAKTION AUF BEFEHLE (ALLGEMEINE
     ACTION)                                 <108>
2100 REM NIMM ROUTINE                        <034>
2110 IF VE<>15 THEN 2200                     <161>
```

Listing 26

```
2115 IF OB<>0 THEN PRINT"DAS GEHT UEBER ME
     INE KRAEFTE !":GOTO 2200                <010>
2119 IF GE(G1)=-2 AND ZN=5 AND TU(2)=0 THE
     N 2125                                  <042>
2120 IF GE(G1)<>ZN THEN PRINT"ICH SEHE DIE
     SEN GEGENSTAND HIER NICHT !"            <016>
2125 IF GE(G1)=-1 THEN PRINT"SIE HABEN DIE
     SEN GEGENSTAND BEREITS !"               <077>
2130 IF GE(G1)=ZN THEN GE(G1)=-1:PRINT"OK.
     "                                       <253>
2200 REM INVENTUR ROUTINE                    <208>
2205 IF VE<>22 THEN 2300                     <255>
2210 PRINT"ICH HABE: ";                      <224>
2220 IC=0:FOR I=1 TO GZ:IF GE(I)=-1 THEN P
     RINT GE$(I)",";:IC=1                    <165>
2225 NEXT                                    <059>
2230 IF IC=0 THEN PRINT"NICHTS."             <050>
2235 IF IC=1 THEN PRINT"{LEFT,SPACE}"        <222>
2300 REM VERLIER ROUTINE                     <210>
2301 IF OB<>0 THEN 2400                      <034>
2305 IF VE<>18 THEN 2400                     <106>
2310 IF GE(G1)<>-1 THEN PRINT"ICH HABE DAS
      NICHT !"                               <006>
2320 IF GE(G1)=-1 THEN GE(G1)=ZN:PRINT"OK.
     "                                       <188>
2400 REM SCHAU - ROUTINE                     <191>
2405 IF VE<>13 THEN 2500                     <202>
2410 IF OB=0 AND G1=0 THEN PRINT"{CLR}":VE
     =0:GOTO 1130                            <208>
2415 IF OB=1 AND OO(1)=ZN THEN PRINT"DIE T
     RUHE IST SEHR GROSS."                   <092>
2500 REM AUFRUF DER RAUMSPEZIFISCHEN ACTIO
     N                                       <126>
2505 GOSUB 3000 : REM GESPENST               <084>
2510 GOSUB 10000+ZN*100+20                   <240>
2520 IF VE>0 AND VE<11 THEN 1100             <176>
2600 GOTO 1000                               <124>
3000 REM STEUERUNG DES GESPENSTES            <241>
3010 IF MO=0 THEN RETURN                     <011>
3020 MO=MO+1:IF MO=12 THEN MO=1              <056>
3025 IF PE(MO)<>ZN THEN 3100                 <096>
3030 PRINT"EIN RIESSIGES GESPENST ERSCHEIN
     T !"                                    <119>
 3100 IC=0:FOR I=1 TO GZ:IF GE(I)=PE(MO)THE
     N IC=I                                  <015>
3105 NEXT                                    <175>
3110 IF IC=0 THEN 3150                       <155>
3120 GE(GF)=PE(MO):GF=IC:GE(IC)=0            <100>
3150 IF PE(MO)=5 AND ZN=7 AND TU(2)=0 THEN
      TU(2)=1:PRINT"JEMAND SCHLIESST DIE T
     RUHE !"                                 <244>
3160 RETURN                                  <242>
10000 REM ------- SPIELKARTE -------          <048>
10100 REM RAUM 1 ----------                  <037>
10102 DATA 0,3,0,2,0,0,0,0,0,0               <145>
10105 PRINT"RAUM NUMMER 1"                   <183>
10120 RETURN                                 <061>
10200 REM RAUM 2 ----------                  <138>
10202 DATA 0,0,1,0,0,0,0,6,0,0               <247>
10205 PRINT"RAUM NUMMER 2"                   <028>
10220 IF TU(1)=0 THEN RI(2)=5                <087>
10224 IF VE=11 AND OB=5 AND GE(3)<>-1 THEN
       PRINT"ICH HABE KEINEN SCHLUESSEL.":
      GOTO 10230                             <123>
10225 IF VE=11 AND OB=5 AND TU(1)=1 THEN P
      RINT"OK.":TU(1)=0:RI(2)=5              <008>
10230 IF VE=12 AND OB=5 AND TU(1)=0 THEN P
      RINT"OK.":TU(1)=1:RI(2)=0              <009>
10240 IF VE=23 AND OB=5 THEN VE=2            <178>
10250 RETURN                                 <192>
10300 REM RAUM 3 ----------                  <240>
10302 DATA 1,0,0,0,0,0,0,0,0,4               <090>
10305 PRINT"RAUM NUMMER 3"                   <130>
10320 RETURN                                 <006>
10400 REM RAUM 4 ----------                  <085>
10402 DATA 0,0,0,0,0,0,0,0,3,0               <188>
10405 PRINT"RAUM NUMMER 4"                   <231>
10420 RETURN                                 <106>
10500 REM RAUM 5 ----------                  <187>
10502 DATA 0,0,0,0,0,0,0,0,0,0               <030>
10505 PRINT"RAUM NUMMER 5"                   <077>
10520 IF TU(1)=0 THEN RI(1)=2                <129>
10524 IF VE=11 AND OB=5 AND GE(3)<>-1 THEN
       PRINT"ICH HABE KEINEN SCHLUESSEL.":
      GOTO 10230                             <169>
10525 IF VE=11 AND OB=5 AND TU(1)=1 THEN P
      RINT"OK.":TU(1)=0:RI(1)=2              <050>
10530 IF VE=12 AND OB=5 AND TU(1)=0 THEN P
      RINT"OK.":TU(1)=1:RI(1)=0              <054>
10540 IF VE=23 AND OB=5 THEN VE=1            <222>
10545 IF VE=11 AND OB=1 AND TU(2)=1 THEN P
      RINT"OK.":TU(2)=0                      <009>
10546 IF VE=12 AND OB=1 AND TU(2)=0 THEN P
      RINT"OK.":TU(2)=1                      <011>
10548 IF VE=15 AND GE(G1)=-2 AND TU(2)=0 T
      HEN PRINT"OK.":GE(G1)=-1               <084>
10550 IF VE=18 AND OB=1 AND TU(2)=0 AND GE
      (G1)=-1 THEN PRINT"OK.":GE(G1)=-2      <124>
10560 IF VE=23 AND OB=1 AND TU(2)=0 THEN R
      I(1)=7:VE=1                            <235>
10590 RETURN                                 <021>
10600 REM RAUM 6 ----------                  <032>
10602 DATA 0,0,0,0,2,0,0,0,0,0               <132>
10605 PRINT"RAUM NUMMER 6"                   <178>
10620 RETURN                                 <051>
10700 REM IN DER TRUHE --------              <185>
10702 DATA 0,0,0,0,0,0,0,0,0,0               <230>
10705 PRINT"IN DER TRUHE."                   <255>
10720 IF VE=25 AND OB=1 AND TU(2)=0 THEN R
      I(1)=5:VE=1                            <139>
10730 RETURN                                 <161>
50000 REM ****************************       <058>
50010 REM *      BEFEHLSEINGABE        *     <228>
50020 REM *      BEFEHLSZERLEGUNG      *     <182>
50030 REM *      BEFEHLSCODIERUNG      *     <173>
50040 REM ****************************       <098>
50050 IF UD=1 THEN 50220                     <236>
50060 POKE 198,0:BE$="":PRINT"WAS NUN ? ";   <044>
50070 POKE 204,0                             <226>
50080 GET X$:IF X$=""THEN 50080              <091>
50090 IF PEEK(203)=1 OR LEN(BE$)>68 THEN P
      RINT" ":POKE 204,1:GOTO 50140          <022>
50100 I=ASC(X$):IF I<65 OR I>90 THEN IF I<
      >32 AND I<>20 AND I<>34 THEN 50080     <173>
50110 IF I=20 AND BE$=""THEN 50080           <093>
50120 IF I=20 THEN POKE 204,1:PRINT"{LEFT,
      2SPACE,2LEFT}";:BE$=LEFT$(BE$,LEN(BE
      $)-1):GOTO 50070                       <174>
50130 PRINT X$;:BE$=BE$+X$:GOTO 50080        <013>
50140 FOR I=1 TO 10:BE$(I)="":NEXT:WZ=1:FO
      R I=1 TO LEN(BE$)                      <154>
50150 :IF MID$(BE$,I,1)=" "THEN GOSUB 5019
      0:GOTO 50180                           <240>
50160 :IF WZ>10 THEN PRINT"EINGABE IST ZU
      LANG !":I=LEN(BE$)+1:GOTO 50180        <028>
50170 :BE$(WZ)=BE$(WZ)+MID$(BE$,I,1)         <069>
50180 NEXT I:GOTO 50220                      <079>
50190 IC=0:FOR I1=1 TO AZ:IF BE$(WZ)=AU$(I
      1)THEN IC=1                            <086>
50200 NEXT I1:IF IC=0 THEN WZ=WZ+1:RETURN    <105>
50210 BE$(WZ)="":RETURN                      <081>
50220 IF UD=1 THEN UD=0:GOTO 50240           <215>
50230 WZ=1:VE=0:OB=0:PE=0                    <163>
50240 IC=0:G1=0:G2=0                         <155>
50250 FOR I=1 TO VZ:IF BE$(WZ)=VE$(I)THEN
      VE=I:IC=1                              <114>
50251 IF LEN(BE$(WZ))<3 THEN 50260           <229>
50255 IF BE$(WZ)=LEFT$(VE$(I),LEN(BE$(WZ))
      )THEN VE=I-VAL(RIGHT$(VE$(I),1)):IC=
      1                                      <211>
50260 NEXT I:IF IC=1 THEN 50350              <187>
50270 FOR I=1 TO GZ:IF BE$(WZ)<>GE$(I)THEN
       50300                                 <211>
50280 IC=1:IF G1=0 THEN G1=I                 <212>
50290 G2=I:IF G2=G1 THEN G2=0                <020>
50300 NEXT I:IF IC=1 THEN 50350              <227>
50310 FOR I=1 TO OZ:IF BE$(WZ)=OB$(I)THEN
      OB=I:IC=1                              <147>
50320 NEXT I:IF IC=1 THEN 50350              <247>
50330 FOR I=1 TO PZ:IF BE$(WZ)=PE$(I)THEN
      PE=I:IC=1                              <176>
50340 NEXT I                                 <051>
50350 IF BE$(WZ)="UND"THEN UD=1:IC=1          <083>
50360 IF IC=0 THEN PRINT"ICH KENNE ";BE$(W
      Z);" NICHT !":RETURN                   <004>
50370 WZ=WZ+1:IF WZ>10 OR BE$(WZ)=""OR UD=
      1 THEN RETURN                          <177>
```

Listing 26 (Fortsetzung)

```
50380 IC=0:GOTO 50250                                  <189>
52000 REM T A B E L L E N                              <161>
52005 RESTORE 52000                                    <115>
52010 REM VERBTABELLE ---------                        <065>
52020 DATA N,S,W,O,NW,NO,SW,SO,RAUF,RUNTER             <241>
52030 DATA OEFFNE,SCHLIESSE,SCHAUE,UNTERSU
      CHE1,NIMM,NEHME1,HOLE2                           <003>
52035 DATA VERLIERE,LEGE1,WIRF2,WERFE3,INV
      ENTUR                                            <126>
52040 DATA GEHE,BETRETE1                               <023>
52045 DATA VERLASSE                                    <000>
52100 VZ=25:DIM VE$(VZ):FOR I=1 TO VZ:READ
       VE$(I):NEXT                                     <169>
52200 REM GEGENSTANDSTABELLE ---------                 <003>
52210 DATA SCHWERT,-2                                  <235>
52211 DATA SEIL,2                                      <204>
52212 DATA SCHLUESSEL,4                                <157>
52213 DATA DIAMANT,5                                   <162>
52300 GZ=4:DIM GE$(4):DIM GE(4):FOR I=1 TO
       GZ:READ GE$(I):READ GE(I):NEXT I                <136>
52400 REM OBJEKTTABELLE ---------                      <088>
52410 DATA TRUHE,5                                     <242>
52412 DATA SCHACHT,6                                   <107>
52414 DATA EISENRING,6                                 <019>
52416 DATA TUER,2                                      <173>
52418 DATA TUER,5                                      <178>
52500 OZ=5:DIM OB$(OZ):DIM OO(OZ):FOR I=1
      TO OZ:READ OB$(I):READ OO(I):NEXT I              <122>
52600 REM PERSONENTABELLE -------                      <114>
52610 DATA GESPENST                                    <059>
52700 PZ=1:DIM PE$(PZ):FOR I=1 TO PZ:READ
      PE$(I):NEXT                                      <175>
52900 REM ALLGEMEINE TABELLEN                          <027>
52910 TU(1)=1:REM TUER 2/5                             <041>
52920 TU(2)=1:REM TRUHE                                <230>
52930 DATA 1,3,4,3,1,2,5,2,6,2,1                       <249>
52935 DIM PE(11):FOR I=1 TO 11:READ PE(I):
      NEXT:MO=1                                        <181>
53000 RETURN                                           <101>
```

Listing 26 (Schluß)

und diese dann in Programmroutinen umzusetzen. Hier noch abschließend Listing 26, das Sie vorliegen haben sollten, wenn Sie bisher richtig mitgearbeitet haben.

Noch ein Tip zum Schluß

Versuchen Sie die Wahl der Bildschirmfarben so zu treffen, daß sie auch angenehm gelesen werden können.

Es empfiehlt sich in jedem Fall für Hintergrund und Rahmen die Farbe Schwarz zu wählen. Als Schriftfarbe wäre dann Hellgrau, Weiß oder Grün geeignet. Viele Adventures, wie man sie von Zeitschriften her kennt, sind in einer unangenehmen Farbkombination gehalten (zum Beispiel schwarze Schrift auf braunem Hintergrund etc).

Wie störend eine grelle Farbgebung für den Spieler sein kann, erfahren Sie spätestens dann, wenn auch Ihnen die Augen tränen — entscheiden Sie sich deshalb gründlich, bevor Sie eine endgültige Auswahl der Farben treffen. Dies sind zwar nur Kleinigkeiten, aber gerade diese Kleinigkeiten werden oft vom Programmierer übersehen. Dies führt nicht zuletzt auch zu einer Qualitätsminderung des Spiels. Bisher haben wir außerdem kaum von Grafik in Adventures gesprochen. Für Datasettenanwender bietet sich lediglich die normale Zeichengrafik an, da hochauflösende Grafik zuviel Speicherplatz benötigt und auch nur schwer mittels Datasette zu bearbeiten ist.

Die Grafik sollte immer das letzte sein, woran Sie arbeiten — sie soll also erst dann eingebaut werden, wenn das eigentliche Adventure schon komplett ist. Es empfiehlt sich, für die Grafiken eine Größe von zirka 10 mal 20 Zeichen zu wählen, da ganze Bildschirmseiten wiederum zuviel Speicherplatz kosten. Das folgende Programm stellt einen Maskengenerator dar, der es Ihnen ermöglicht, Bilder einfach zu erstellen.

Nachdem der Maskengenerator geladen ist, wird das Bild mittels Cursortasten auf den Bildschirm »gemalt«. Ist das Bild fertig, so wird es an jeder Ecke mit einem Klammeraffen-Zeichen versehen. Die vier Klammeraffen müssen ein Rechteck bilden, beziehungsweise die Eckpunkte eines Rechtecks begrenzen, in dem das Bild steht.

Sodann wird der Maskengenerator mittels RUN 60000 gestartet. Von der Größe des Bildes hängt die Bearbeitungszeit ab. Nach einer Weile, erscheint das Bild in Form von PRINT-Zeilen auf dem Bildschirm. Sie müssen nun nur noch die RETURN-Taste mehrmals drücken, um die PRINT-Zeilen zu übernehmen. Das Programm errechnet alle Steuerzeichen und Farben automatisch und setzt sie in PRINT-Zeilen um.

Das eigentlich mühselige Erstellen von Blockgrafiken wird so mit Listing 27 zum Kinderspiel.

```
60000 REM ****************************           <155>
60001 REM G R A F I K - D E S I G N E R          <012>
60002 REM ****************************           <157>
60003 DATA"{WHITE}","{RED}","{CYAN}","{PUR
      PLE}","{GREEN}","{BLUE}","{YELLOW}",
      "{ORANGE}","{BROWN}","{LIG.RED}","{G
      REY 1}","{GREY 2}","{LIG.GREEN}","{L
      IG.BLUE}","{GREY 3}"                             <020>
60004 DIM Z$(25):I=1024:DIM Y$(15):FOR I4=
      1 TO 15:READ Y$(I4):NEXT I4                      <100>
60005 IF PEEK(I)=0 THEN A=I+1:GOTO 60008               <151>
60006 IF I>2023 THEN 60011                             <059>
60007 I=I+1:GOTO 60005                                 <046>
60008 FOR I=A TO A+38:IF PEEK(I)=0 THEN B=
      I-1                                              <204>
60009 NEXT:FOR I=A-1+40 TO 2023 STEP 40:IF
       PEEK(I)=0 THEN C=I+1                            <241>
60010 NEXT                                             <214>
60011 AB=B-A:D=C+AB:IF A=0 OR B=0 OR C=0 O
      R D=0 THEN PRINT"?ILLEGAL SIZE{2SPAC
      E}ERROR":END                                     <251>
60012 FA=PEEK(53280)                                   <164>
60013 IF FA>15 THEN FA=FA-16:GOTO 60013                <182>
60014 FOR I1=A TO C STEP 40:Z$="?"+CHR$(34
      ):FOR I=I1 TO I1+AB:X=PEEK(I):CL=PEE
      K(I+54272)                                       <240>
60015 IF CL>15 THEN CL=CL-16:GOTO 60015                <210>
60016 IF RV=2 AND X<128 THEN RV=3                      <250>
60017 IF X>127 AND RV=0 THEN RV=1                      <244>
60018 IF RV=1 OR RV=2 THEN X=X-128                     <253>
60019 IF X>63 AND X<96 THEN X=X+32:GOTO 60
      023                                              <089>
60020 IF X>96 AND X<128 THEN X=X+64:GOTO 6
      0023                                             <145>
60021 IF X>0 AND X<32 THEN X=X+64:GOTO 600
      23                                               <029>
60022 IF X>=32 AND X<64 THEN X=X                       <056>
60023 IF FA=CL THEN 60027                              <090>
60024 FA=CL:IF FA=0 THEN Z$=Z$+"{BLACK}"               <043>
60025 FOR I2=1 TO 15:IF FA=I2 THEN Z$=Z$+Y
      $(I2)                                            <013>
60026 NEXT I2                                          <097>
60027 IF RV=1 THEN Z$=Z$+"{RVSON}":RV=2                <150>
60028 IF RV=3 THEN Z$=Z$+"{RVOFF}":RV=0                <023>
60029 Z$=Z$+CHR$(X):NEXT I:Z$=Z$+CHR$(34):
      Z$(AN)=Z$:Z$="":AN=AN+1:RV=0:NEXT I1             <154>
60030 INPUT"{CLR}AB WELCHER ZEILE ";ZN:IF
      ZN<0 OR ZN>63000 THEN 60030                      <159>
60031 PRINT"{CLR,2DOWN}":FOR I=0 TO(C-A)/4
      0:PRINT ZN+I;Z$(I):NEXT:PRINT"{HOME}
      ";:END                                           <178>
```

Listing 27. Grafik-Designer

Ich hoffe, daß es mir mit dem vorliegenden Kurs gelungen ist, Sie für Abenteuerspiele und deren Programmierung zu interessieren. Dazu wünsche ich Ihnen viele gute Ideen und viel Spaß.

(Michael Nickles/rg)