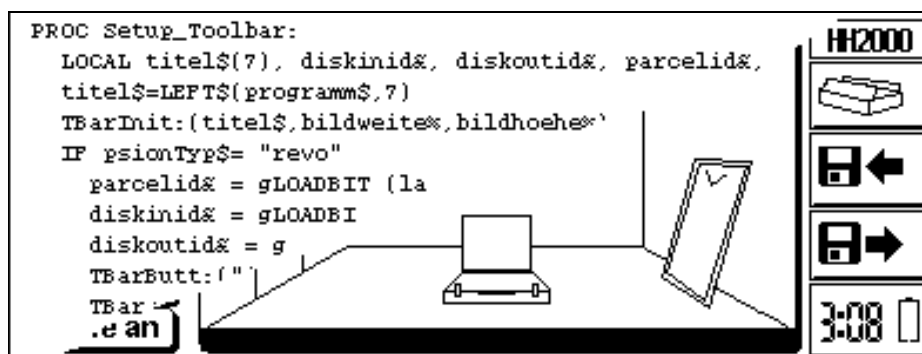


Der zweite Schritt:

Solide PSION-Programmierung unter OPL

- am Beispiel eines Adventure-Games –

Ulrich Krinzner



Der zweite Schritt:
Solide PSION-Programmierung unter OPL
- am Beispiel eines Adventure-Games –
von Ulrich Krinzner

© Ulrich Krinzner 2002

Das vorliegende Werk ist in allen seinen Teilen urheberrechtlich geschützt. Alle Rechte vorbehalten. Teilweise oder vollständige Reproduktionen - in welcher Form auch immer-, jedwede Veröffentlichung, Übersetzung sowie Speicherung in elektronischen Medien ist nur mit ausdrücklicher schriftlicher Genehmigung des Autors zulässig.

Die Warenzeichen aller erwähnten Erzeugnisse und Firmen werden ausdrücklich als solche anerkannt. Ihre Verwendung dient ausschließlich der Information des Lesers und keiner kommerziellen Absicht.

Hier vorgestellte Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind lediglich für Lehrzwecke bestimmt.

Das vorliegende Material wurde unter großer Sorgfalt zusammengestellt, trotzdem ist Fehlerfreiheit nicht zu gewährleisten. Der Autor kann für solche möglichen Fehler und deren Folgen weder juristische Verantwortung noch sonstige Haftungen übernehmen.

E-Mail: krinzner@snafu.de

2.Auflage, 7/2002

Inhaltsverzeichnis

Inhaltsverzeichnis	3
A - Vorwort	7
B - Ein Rahmen	11
Wie sich alles einordnet	12
Kopfteil zusammenstellen	12
<i>Die Applikationsanweisung</i>	<i>13</i>
<i>Externe Dateien einbinden</i>	<i>14</i>
<i>Toolbar einbinden</i>	<i>15</i>
<i>Die Initialisierung</i>	<i>16</i>
Die große Programm-Schleife	19
<i>Events</i>	<i>20</i>
<i>Was genau ist los?</i>	<i>21</i>
<i>Tastatureingaben</i>	<i>22</i>
<i>Stifteingaben</i>	<i>23</i>
<i>Das Menü</i>	<i>24</i>
Mehr	24
C - Das Programm planen	25
Was und wie?	26
Das Abenteuer beginnt	27
<i>Story & Co.</i>	<i>27</i>
<i>Der Ort</i>	<i>28</i>
<i>Inventar-Liste</i>	<i>29</i>
<i>Für den Spieler geheim: Der Lösungsweg</i>	<i>29</i>
<i>Kommando-Sachen</i>	<i>30</i>
<i>Die Bildschirmgestaltung</i>	<i>30</i>
Ein Grobplan	31
Teufel steckt im Detail	32
D - Das Spiel programmieren	35
Verändertes Rahmenprogramm	36
Erweiterte Initialisierung	39
Objekte verwalten	40
Spaziergang durch die Räume	42
<i>Gehrichtung ermitteln, Raumnummer ändern</i>	<i>42</i>
<i>Raum neu zeichnen</i>	<i>44</i>
Details erforschen	46
<i>Überblick durch "umsehen"</i>	<i>47</i>
<i>Gegenstände ansehen – "sieh an"</i>	<i>48</i>
<i>Inventar ansehen</i>	<i>49</i>
"Action"	49
<i>Öffnen von Objekten</i>	<i>49</i>

<i>Objekte nehmen</i>	50
<i>Wie man Gegenstände verwendet</i>	51
<i>Gegenstände verschieben</i>	52
<i>Höflich, aber sinnlos</i>	52
Spielstände sichern und wiederherstellen	53
<i>Spiel speichern</i>	53
<i>Spiel laden</i>	54
Hilfe!.....	54
Zu guter Letzt.....	55
Anhang 1 Quelltext: Rahmen.OPL	57
Anhang 2 Quelltext: HH2000.OPL	65
Anhang 3 Skizzen	109
Die Programm-Icons.....	111

A - Vorwort

Irgendwann hat man sich einmal einen Psion-Handheld der Serie 5xx oder auch ein netBook zugelegt, die alle mit dem gleichen Betriebssystem (EPOC32) arbeiten. Und eines schönen Tages entdeckt man, dass diese Geräte einen Schatz bergen: alle haben sie nämlich bereits die Programmiersprache OPL samt Editor und Übersetzer an Bord. Selbst der Revo ist OPL-fähig, den fehlenden Editor findet man im Internet.

Erfahrungsgemäß unternimmt man dann seine ersten zaghaften Programmiersuche aus beruflicher Notwendigkeit oder der puren Lust heraus. Die ersten Programm-Zeilen sind schnell und meist erfolgreich geschrieben, wie das bei allen BASIC-ähnlichen System so ist. Fortschritte lassen sich ebenfalls ganz gut realisieren. Und es geht weiter.

Man entdeckt, dass es außer dem Textbildschirm auch noch einen grafischen Bildschirm gibt – die eigenen Programme werden kühner. Aber es sind noch immer keine "richtigen", sondern "nur" OPO-Programme. Zunächst zielt noch kein eigenes Icon die Extras-Leiste und erleichtert den Aufruf der Programme. Und an einen echten Toolbar ist noch lange nicht zu denken.

Erste Stagnation tritt auf, vor einem liegt eine Qualitätsschwelle, die überschritten werden will, aber wie und wo ansetzen? Der Psion liefert so merkwürdige Musterprogramme mit, die wenig verständlich sind. Mit denen kann man zwar einen Toolbar zaubern und die Demo-Programme sind nach dem Übersetzen richtige Applikationen – aber wie funktioniert das? Oft ist man an dieser Stelle auch geneigt, den Leuten zu glauben, die behaupten: "BASIC, also auch OPL, ist Mist".

Ich selber habe als Einsteiger sehr viel in englischen Quellen lesen müssen und im Internet gestöbert, bis ich mir meinen eigenen Reim auf die Sachen machen konnte. Und meine dann folgenden Erfahrungen sagen: OPL ist wahrlich kein "Mist". Für alle Anwendungen, die keine extremen Geschwindigkeiten verlangen – auch "ganz komplizierte", ist OPL mehr als nur geeignet.

Mit dem vorliegenden Büchlein sollen es die Einsteiger beim Sprung in die nächste Ebene etwas einfacher haben als ich selbst. Für den engagierten Anwender wird das zwar nicht der letzte Sprung sein, aber eben der erste größere.

Vorausgesetzt werden Basiskenntnisse in OPL, ohne die ist diese Lektüre nicht zu empfehlen.

Damit die Themen nicht so staubtrocken bleiben, benutze ich ein spielerisches Beispiel, anhand dessen die Einzelheiten erklärt werden. Es ist ein kleines Game, das sich an den Vorbildern meiner "Haunted House"-Serie orientiert. Auch Leser, die keine Spielernaturen sind, werden sich mit dem Thema anfreunden können, denn das Mini-Adventure ist eben nur das Vehikel, mit dem das Wissen transportiert wird.

Folgende Themenblöcke werden behandelt:

1. Allgemeine Programm-Konstruktion (u.a. Programm-Rahmen, Events, Einbindung und Aktivierung eines Toolbars)
2. Planung eines Programmes
3. Umsetzung der Planung

Die einzelnen Schritte werden am Beispiel genügend ausführlich erläutert. In der Ausführung der Programmierung wurde mehr Wert auf Verständlichkeit als auf ausgefeilte Algorithmen gelegt. Der Einstieg in die Materie wird Ihnen deshalb nicht schwer fallen. Gebrauchen Sie im Zweifelsfall Ihren eigenen Verstand, dann haben Sie die nächste Stufe schon fast automatisch erklommen. Und eines gilt beim Programmieren und im "richtigen Leben" immer: "Man kann viele Sachen so, aber auch ganz anders machen!"

Ulrich Krinzner, Mai 2002

B - Ein Rahmen

Wie sich alles einordnet

Wer programmieren will und das Programm im Kopf schon fast fertig hat, möchte am liebsten gleich loslegen. "Richtige" Programme benötigen aber unabhängig davon eine gewisse "technische" Vorbereitung und einen Rahmen, der das Aussehen und die Funktionalität des Programms bestimmt.

Der Hersteller von Betriebssystem und OPL, Symbian, empfiehlt in einem umfangreichen "Style Guide", wie "richtige" Psion-Programme auszusehen haben. Das ist keine Gängelei der Entwickler, sondern der Anwender steht im Mittelpunkt: er soll sich immer schnell zurechtfinden, egal welches Programm er auf dem Psion benutzt. Zu den Empfehlungen gehört, dass der Entwickler einen Toolbar und Menüs programmiert. Darüber hinaus empfehle ich auch noch die Einbindung einer Fehlerroutine, die hoffentlich nie in Aktion treten muss.

Allein, wenn man nur diese drei Punkte berücksichtigen will, müssen eigene Programme einen bestimmten Aufbau haben. Die sehr groben Bestandteile sind:

1. ein Kopfteil für die verschiedensten Deklarationen;
2. Globalvariablen-Deklaration und -Initialisierung;
3. die Vorbereitung der Toolbarbenutzung;
4. Initialisierung;
5. ein Ausführungsteil, der aus einer unendlichen Schleife besteht.

Diese Bestandteile verteilen sich auf verschiedene Prozeduren, die in der Abbildung B1 noch einmal schematisch dargestellt sind .

Wir sehen uns im folgenden die Bestandteile detailliert an. Das zugehörige Demo-Programm heißt "Rahmen.opl". Ich werde Quelltext-Teile hier nur noch einmal zitieren, wenn es nötig ist. Die komplette und kommentierte Fassung finden Sie im Anhang - am besten legen Sie sich jeweils ein Lesezeichen an die betreffende Stelle ...

Kopfteil zusammenstellen

Im Kopfteil werden in dieser Reihenfolge ...

1. die Anweisungen für die Applikation zusammengestellt;
2. externe Dateien, auf die das Programm zurückgreift , mit INCLUDE eingebunden;
3. Konstanten mit CONST definiert.

Die Einzelheiten folgen auf dem Fuß.

Programm "Rahmen.opl"/Schema

```
REM Kopfbereich
  REM Applikationsanweisungen
  REM INCLUDE-Anweisungen
  REM CONST-Anweisungen
REM Kopfbereich Ende

PROC Begin:    REM eigentlicher Programmstart
  REM Globalvariablen deklarieren und initialisieren
  REM Toolbar laden
  TBarLink:("Main:")    REM Toolbar "linken"
  REM Programm kehrt nicht mehr hierher zurück!
ENDP

PROC TBarLink:(AppLink$) REM in Toolbar.opo
  REM Toolbar-Vorbereitungen
  @(AppLink$): REM Sprung zu Main:
ENDP

PROC Main:
  Init: REM Toolbar-Init & -Anzeige
  WHILE 1
    REM Schleife mit Eventabfrage
  ENDWH
ENDP
```

Abb. B1

Die Applikationsanweisung

Mit dem Anweisungsblock

```
APP Rahmen, &00004812
...
ENDA
```

entsteht nach dem Übersetzen ein "richtiges" Programm aus dem Quelltext – der OPL-Übersetzer sorgt dafür.

Die Liste zwischen APP und ENDA kann – muss aber nicht - einige der wenigen nutzbaren Anweisungen (CAPTION, ICON und FLAGS) enthalten, die ich mir hier erst einmal noch erspart habe.

Der Name hinter APP findet sich später als Programm-Titel in der Extras-Leiste wieder. Die Long-Integerzahl dahinter ist die UID, die Unique Identification Number, die Programme unverwechselbar macht. Der Sinn ist, dass Programme, die mit Dateien arbeiten (wie WORD), mit diesen Dateien zuverlässig verknüpft werden können. In der Praxis geht das dann so: Sie tippen im Systembildschirm z.B. auf eine WORD-Textdatei, wodurch zugleich die zugehörige Applikation - hier eben WORD - startet. Damit es da kein Durcheinander gibt, bedarf es dieser UID.

Für den privaten Bereich steht nach Vorgaben von Psion/Symbian der Bereich &01000000 bis &0FFFFFFF beliebig zur Verfügung. Sowie Sie mit Ihren Programmen an die Öffentlichkeit gehen wollen, besorgen Sie sich (kostenfrei) bei Symbian eine der weltweit nur einmal vorkommenden Nummern, die in den Bereich ab &10000000 fallen. Somit wird Ihr Programm nie mit anderen öffentlichen Programmen kollidieren können. Die derzeit gültige Symbian E-Mail-Adresse lautet:

uid@symbiandevnet.com

Sollte sich das geändert haben, suchen Sie die Information im Entwicklerbereich auf der Website <http://www.symbian.com/>. Geben Sie dort in die Suchmaske einen Begriff wie "EPOC UID" ein und Sie werden bestimmt ein hilfreiches Info-Dokument finden. Bei der Bestellung von UIDs geben Sie in der E-Mail an:

- im Betreff: "UID Request",
- Ihren Namen oder den Programm-Namen,
- Ihre E-Mail-Adresse,
- die Anzahl der benötigten UIDs.

Noch sind im Demo-Programm keine Icons definiert, deshalb hilft sich der Psion selbst und stellt aus seinem Fundus das Fragezeichen-Icon bereit. Damit präsentiert sich das Programm auch in der Extras-Leiste (Abb. B2).



Rahmen Abb. B2

Externe Dateien einbinden

Mit INCLUDE bindet man, falls erforderlich, zusätzliche Dateien ein, die bereits vor der Verwendung auf dem Gerät installiert sein müssen. Nicht jedes Programm braucht Zusatzdateien. Trotzdem bringt jeder Psion bereits einige davon im ROM mit, um fortgeschrittene Programmierung zu erleichtern. Es handelt sich dabei um professionell erstellte Programmschnipsel, die man wie Unterprogramme aufrufen kann. Manchmal ist es aber auch nur eine Definitionsdatei, eine "Header-Datei". Im Demo-Programm binden wir "CONST.OPH" ein, eine Datei, in der Symbian etliche System-Konstanten definiert und mit einem Wert versehen hat:

```
INCLUDE "CONST.OPH"
```


Die Verwendung dieser Konstanten ist zwar vorteilhaft, der Einsteiger geht jedoch meist lieber mit den dahinter steckenden Werten um. Ich habe die Benutzung auf ein Minimum eingeschränkt, wollte aber zur Anschauung nicht ganz darauf verzichten, Beispiel:

```
KTrue%      REM = vertritt den Wert -1 für logisch "WAHR"
KFalse%     REM = vertritt den Wert 0 für logisch "FALSCH"
```

Wer eigene Konstanten definieren will, macht das im Anschluss an die INCLUDE-Zeilen mit der CONST-Anweisung.

Alle Angaben müssen nur gemacht werden, wenn sie erforderlich sind. Soll aus dem Programm keine Applikation, sondern nur eine lauffähige OPO-Datei werden, und würde man weder System-Konstanten noch eigene Konstanten verwenden, könnte man sich diesen ganzen Vorspann sparen.

Toolbar einbinden

Dem Vorspann folgt im Demo-Programm die Prozedur *Begin:*. Weil sie die erste ist, startet OPL hier mit der Programmausführung.

Begin: dient nur einem einzigen Zweck: dem Laden des Toolbar-Moduls (*Toolbar.opo*) aus dem ROM des Gerätes. *Toolbar.opo* enthält alle Prozeduren, die den Umgang mit dem Toolbar unterstützen. Der nächste Schritt, der Aufruf von *TBarLink:()* in *Toolbar.opo*, ist trickreich und bedarf der Erläuterung:

Die Prozeduren in *Toolbar.opo* können nur funktionieren, wenn die dort verwendeten Globalvariablen vor der Benutzung deklariert und initialisiert wurden. Da wir als Programmierer vom Innenleben des "fremden" Moduls nichts wissen, kennen wir die zu deklarierenden Variablen nicht und können diese Arbeit nicht selber verrichten. Aber der Modulentwickler hat vorgesorgt und extra die *TBarLink:()*-Routine eingebaut, die diese Arbeit für uns übernimmt. Wir müssen sie nur noch aufrufen, aber das auf eine ganz besondere Art!

Üblicherweise kehren aufgerufene Unterprogramme am Ende wieder zu der aufrufenden Prozedur zurück. Sie wissen aber sicher, dass man in untergeordneten Prozeduren keine globalen Variablen für übergeordnete Prozeduren deklarieren kann. Deshalb muss man dafür sorgen, dass das Unterprogramm nicht zurückkehrt, sondern gewissermaßen einen Staffelstab übernimmt und nach getaner Arbeit weitergibt. Und so kehrt auch *TBarLink:()* nicht zu *Begin:* zurück, sondern springt zu der Prozedur *Main:* in unserem Programm.

Dass die Routine *TBarLink:()* zu *Main:* springen soll, weiß sie aus dem Parameter, dem "Staffelstab", der ihr mitgegeben wird. Verkürzt sieht der Vorgang so aus:

```

PROC Begin:
    ...
    TBarLink: ("Main")
    REM Tschüss, auf Nimmerwiedersehen!
ENDP

PROC TBarLink: (AppLink$)
    ...
    @(AppLink$): REM Sprung zu Main:
ENDP

```

Der Parameter wird also einfach durchgelangt und per @-Operator zum Sprung benutzt. Das Programm gelangt damit nie wieder zur Prozedur *Begin*: zurück - sie hat ihre Schuldigkeit getan. Die Aktivierung des Toolbars wird in *Main*: fortgesetzt.

Die Initialisierung

Inzwischen sind wir in *Main*., der Hauptroutine des Programmes gelandet. Bevor sie in die Schleife eintritt, werden die programmeigenen Variablen deklariert und, wo erforderlich, mit einem Wert versehen. Anschließend wird das Programm initialisiert. "Initialisieren" bedeutet in diesem Zusammenhang, dass weitere Vorbereitungen getroffen werden müssen, bevor das Programm so richtig loslegen kann.

Zu den Vorbereitungen zählt das weitere Einrichten des Bildschirms, z.B. mit Fenstern, Bildern oder Schrift, und auch der Aufbau des Toolbars. Um die Übersicht zu bewahren, wird die Initialisierung in eine eigene Routine, nämlich nach *Init*., verlegt.

Toolbar vorbereiten, die zweite

Der Hauptanteil von *Init*: beschäftigt sich mit dem Toolbar. Um dem Toolbar die richtige Größe und Position zuweisen zu können, wird zuerst der Gerätetyp festgestellt. Beachten Sie: die Psion-Clones mit abweichender Bildschirmgröße sind hier nicht berücksichtigt. Die Bildschirm-Abmaße für diese Geräte lauten: Geofox One: 640*320 Pixel, Osaris: 320*200 Pixel. Anschließend wird die eigentliche Toolbar-Initialisierung durch Aufruf von *Setup_Toolbar*:() vorgenommen.

Der Toolbar ist eigentlich nichts weiter als ein spezielles Fenster mit ein paar Buttons. Der Programmierer bestimmt innerhalb gewisser Grenzen das Aussehen dieses Fensters. Ihm stehen dabei weitere Prozeduren aus *Toolbar.opo* zur Seite, die er der Reihe nach benutzen muss:

- *TBarInit*:() Festlegung des Titels, der im Kopf angezeigt werden soll, und Festlegung der Toolbar-Position

- *TBarButt:()* Regelt, wie die Buttons gestaltet sind
- *TBarShow:* Befehl zum Anzeigen des Toolbars, mit *TbarHide:* wird der Toolbar bei Bedarf versteckt

Schlüsseln wir das Toolbar-Puzzle noch etwas weiter auf.

Titel und Position des Toolbars

Zur Vorbereitung werden zuerst Icon-Bitmaps und –Masken geladen. In diesem Falle haben wir sie aus dem ROM "geborgt", ohne Rücksicht darauf, welche Funktion sie eigentlich symbolisieren, später verwendet man dafür eigene Grafiken.

```
bild$="Z:\System\Apps\Data\Data.mbm"
bitmap1&=gLoadBit (bild$, 0, 3)
mask1&=gLoadBit (bild$, 0, 4)
```

Danach setzt *TbarInit:()* den Titel:

```
TBarInit: ("Demo", bildweite%, bildhoehe%)
```

Achten Sie darauf, dass der als Parameter mitgegebene String nicht mehr als 7 Zeichen lang ist. Ein längerer String erzeugt zwar keinen Fehler, aber der Titel könnte mitten in einem Buchstaben abgeschnitten sein – ein unschöner Anblick. Aus den beiden anderen Parametern, der Bildweite und Bildhöhe beim Programmstart, entnimmt *TbarInit:()* automatisch, wo der Toolbar angeordnet werden muss.

Die Buttons anlegen

Im Puzzlespiel kommen nun die Buttons an die Reihe. Für jeden einzelnen Button wird *TbarButt:()* aufgerufen, für seine Funktion und sein Aussehen sind die übergebenen Parameter zuständig. In allgemeiner Form sieht das so aus.

```
TbarButt: (shortcut$, pos%, text$, status%, icon&, mask&, flag%)
```

Die einzelnen Parameter bedeuten:

- shortcut\$

ist ein String mit dem Shortcut-Buchstaben; über die Tasten-Kombination <Strg>-<Shortcutbuchstabe> ruft man die gleiche Funktion auf, wie durch einen Tipp mit dem Stift auf den Button; Achtung, Groß- und Kleinschreibung wird ausnahmsweise einmal berücksichtigt!

- pos%

ist die Position des jeweiligen Buttons in der Leiste, gezählt von oben; die Zählung beginnt mit 1. Der Revo benutzt maximal drei, der Serie5xx vier und die großen, S7 und netBook, sechs Buttons.

- text\$

enthält den Button-Text

- status%

legt fest, ob der Button erhaben, flach oder eingedrückt dargestellt wird,
Werte: 0,1,2

- icon&

ist die Identitätsnummer (ID) einer geöffneten (geladenen) Bitmap-Datei. Ein Icon hat üblicherweise die Abmaße 24*24 Pixel. Ist kein Icon zur Hand oder soll absichtlich keines dargestellt werden, ist hier eine Long-Integer-Null (&0) einzusetzen. Am S5xx und am S7/netBook tritt dann an die Stelle des Icons ein Quadrat mit abgerundeten Ecken. Am Revo bleibt die Icon-Fläche automatisch leer, er benutzt auch bei den Standard-Applikationen keine Icons, trotzdem kann man welche verwenden.

- mask&

ist ebenfalls die ID einer geöffneten Bitmap-Datei, diese dient aber als Maske; ist keine Maske zur Hand, funktioniert behelfsmäßig auch das Icon-Bitmap als Maske. Ist icon&= &0, spielt der Wert von mask& keine Rolle – es wird das Rechteck bzw. am Revo nichts angezeigt.
Kurzerläuterung "Maske": Von dem eigentlichen Bitmap werden nur die Pixel am Bildschirm dargestellt, wo die Maske an der gleichen Stelle einen gesetzten Pixel besitzt. Die Maske muss genauso groß sein wie das eigentliche Bitmap.

- flag%

unterstützt das Aussehen der Buttons, wenn diese wie z.B. beim Taschenrechner-Programm des Psions benutzt werden: Jeweils einer der Buttons "Tisch" oder "Wiss" ist flach dargestellt und symbolisiert so, welche Art des Taschenrechners gerade aktiviert ist. Ich gehe in diesem Buch nicht auf diese Technik ein, der Wert von flag% wird einfach mit Null angegeben.

Im Demo-Programm wird das so ausgeführt:

```
IF psionTyp$= "revo"
  TBarButt: ("a",1,"Info",0,&0,&0,0)
  REM Beispiel f. zweizeilige Buttonbeschriftung:
  TBarButt: ("c",2,"Clear"+CHR$(10)+"Win 1",0,&0,&0,0)
  TBarButt: ("d",3,"Clear"+CHR$(10)+"Win 2",0,&0,&0,0)
ELSE
  TBarButt: ("a",1,"Info",0,bitmap1&,mask1&,0)
  TBarButt: ("c",2,"Clear"+CHR$(10)+"Win 1",0,&0,&0,0)
  TBarButt: ("d",3,"Clear"+CHR$(10)+"Win 2",0,&0,&0,0)
  TBarButt: ("b",4,"Ende",0,&0,&0,0)
  IF psionTyp$="netbook"      REM ... ein netBook/S7
    REM Buttons ohne Icon
    TBarButt: ("H",5,"Hilfe",0,&0,&0,0)
    TBarButt: ("r",6,"Rahmen"+CHR$(10)+"ein",0,&0,&0,0)
  ENDIF
ENDIF
```

Nach dem Einfügen der Buttons wird am unteren Ende des Toolbars automatisch eine Uhr eingeblendet. Um die müssen Sie sich nicht extra kümmern, das erledigen die internen Routinen von *Toolbar.opo* eigenständig.

Hinweis: Der Emulator reagiert wie ein S5mx, stellt also auch in der großen Bildschirmversion maximal nur vier Buttons dar. Deshalb reagiert er mit einem Fehler, wenn man die beim netBook und Serie 7 üblichen sechs Buttons unterbringen will.

Die Show beginnt

Zu guter Letzt folgt die Krönung der Zeremonie:

```
Show_Toolbar:
```

Diese Prozedur ruft wiederum *TbarShow*: und sorgt für die Darstellung des Toolbars am Bildschirm. Außerdem werden die Bildschirmparameter angepasst – das ist nur erforderlich, wenn man selber mit dem Bildschirm noch etwas machen will und die veränderten Abmaße wissen muss. *TbarShow*: könnte also durchaus auch direkt aufgerufen werden.

Obwohl inzwischen optisch präsent, ist der Toolbar noch immer nicht funktionsfähig. Es fehlt ein Programmstück, das einen Tipp auf die Buttons oder andere Bereiche erkennt und verarbeitet. Wir müssen dazu die Events des Computers auswerten. Dazu kommen wir gleich.

Zunächst einmal ist die Initialisierung erledigt und das Programm kehrt von *Init*: zu *Main*: zurück.

Die große Programm-Schleife

Gleich nach *Init*: tritt das Programm in eine unendliche Schleife ein. Um genau zu sein: es sind zwei Schleifen. Die äußere erfüllt lediglich die Aufgabe, nach einem Fehler und dessen Behandlung wieder in die innere Schleife einzutreten. Das verkürzte Prinzip der Schachtelschleife:

```
WHILE 1
  ONERR Fehler::
  WHILE 1
    REM Hier läuft das Hauptprogramm im Kreis;
    REM eine Abbruchstelle muss einprogrammiert sein!
  ENDWH
  Fehler::
  ONERR OFF
  REM Sprung zur Fehlerbehandlung
  Fehlermeldung:()
ENDWH
```

Durch diese Anordnung wird das Programm selbst bei einem Fehler fortgesetzt. Allerdings ist nie sicher, ob das Programm wirklich korrekt weiterläuft, aber der ganz große Crash bleibt zumeist aus.

Dass das Programm in eine Schleife gelegt wird, hat einen tieferen Sinn. Solange es frei laufen kann, kommt das Programm nämlich immer wieder an der Stelle vorbei, an dem die GETEVENT32-Anweisung steht. Hier hat unser Programm gewissermaßen das Ohr an den "Events" des Psions – erst dadurch kann der Spieler so richtig mit dem Programm "reden".

Events

Jede Veränderung am Computer, z.B. ein Tastendruck, löst einen "Event" aus. Es handelt sich dabei um eine Art Signal, das einerseits dem, der es wissen will, mitteilt: "He, jetzt ist was passiert!" und andererseits die Informationen bereitstellt, was "passiert" ist. Organisiert wird das ganze vom Betriebssystem, das Events teilweise auch selber auswertet, aber eben auch dem Programmierer Eingriffe in den Programmablauf gestattet.

Wenn es um Events am Psion geht, ist GETEVENT32 Ihr Verbündeter. Mit Hilfe dieser Anweisung erfahren Sie vom Psion alles über den Event: insbesondere, ob überhaupt einer stattgefunden hat und was ihn verursacht hat.

Uns interessiert am meisten, ob entweder eine Taste gedrückt oder der Stift benutzt wurde. Außerdem ist auch noch wichtig, ob "von außen" eine Aufforderung zum Beenden des Programmes gekommen ist (Schließen des Programmes über die "Liste der aktiven Anwendungen"). Wir werden uns nur um diese drei Event-Arten kümmern, es gibt jedoch noch mehr davon.

GETEVENT32 löst eine Abfrage an das System aus, deren Ergebnisse sich in einem Long-Integer-Array wiederfinden, das mindestens 16 Elemente haben muss. Im Beispiel heißt die Array-Variable a&(). Die vollständige Abfrage lautet dann also:

```
GETEVENT32 a&()
```

Die wichtigste Information steht gleich im ersten Element des Arrays, also a&(1). Dessen Wert wird mit einem IF .. ENDIF-Konstrukt auf die für uns interessanten drei Events hin "abgeklopft".

Unter Verwendung der Konstanten aus CONST.OPH liest sich das dann so:

```

IF a&(1)= KEvCommand&
    IF LEFT$(GETCMD$,1)="X" : Exit: : ENDIF
ELSEIF (a&(1) AND KEvNotKeyMask&)=0
    Handle_keyevent:(a&(1), a&(4))
ELSEIF (a&(1)= KEvPtr& OR (a&(1)<=10004 AND a&(1)>=10000))
    Handle_penevent:(a&(1),a&(3),a&(4),a&(6),a&(7))
ENDIF

```

Was genau ist los?

Die Erläuterung:

1. Wenn die Aussage

```
a&(1)= KEvCommand&    REM KEvCommand&= &404
```

WAHR ist, handelt es sich um ein "Kommandozeilen-Event". Uns interessiert der Fall nur, wenn es sich um eine Aufforderung zum Beenden des Programmes handelt. Das ist dann der Fall, wenn das erste Zeichen der Kommandozeile ein "X" ist. Man erfährt das durch Prüfen mit GETCMD\$. Ist die Bedingung erfüllt, wird das Programm über die Prozedur *Exit*: beendet. Ein solcher Kommandozeilen-Event wird durch den Beenden-Befehl in der "Liste der geöffneten Anwendungen" ausgelöst, dabei wird auch automatisch das "X" übergeben.

2. Wenn die Prüfbedingung

```
(a&(1) AND KEvNotKeyMask&)= 0    REM KEvNotKeyMask&= &400
```

WAHR ist, wurde eine Taste gedrückt. Die weitere Bearbeitung wird der Unterroutine *Handle_keyevent:()* überlassen. Als Parameter übergibt man die Werte des Tastaturcodes (steht in a&(1)) und der Modifiziertaste (steht in a&(4)).

3. Stift-Events ("Pen-Events") haben verschiedene Werte. Wird auf den Bildschirmbereich einschließlich Toolbar getippt, gilt:

```
a&(1)= KEvPtr&    REM KEvPtr&= &408
```

Die folgenden Werte sind der Funktionsleiste links vom Bildschirm zugeordnet:

```

a&(1)= 10000    REM Menü-Aufruf
a&(1)= 10001    REM Bearbeiten-Menü
a&(1)= 10002    REM IR senden/empfangen
a&(1)= 10003    REM Einzoomen
a&(1)= 10004    REM Auszoomen

```

Die zusammengefasste Prüfbedingung lautet damit:

```
a&(1) = KEvPtr& OR (a&(1) <= 10004 AND a&(1) >= 10000)
```

Alle Pen-Events werden an die Unteroutine *Handle_penevents()* weitergegeben – einschließlich wichtiger Parameter aus dem Array (s.a. Anmerkungen im Quelltext):

```
a&(1) = Event-Art  
a&(3) = Fenster-ID  
a&(4) = Stift-Kontaktart  
a&(6) = x-Koordinate im Fenster  
a&(7) = y-Koordinate im Fenster
```

Mit der GETEVENT32-Abfrage und der anschließenden Auswertung ist unsere Hauptschleife erst einmal fertig konstruiert. Erweiterungsmöglichkeiten lernen Sie später noch kennen. Verfolgen wir nun, was die Unterprogramme leisten.

Tastatureingaben

Wenn Sie anfangen, ein Programm zu schreiben, haben Sie meist eine Vorstellung, auf welche Tastendrücke das Programm reagieren soll. Dazu gehören einige Shortcut-Kombinationen (z.B. <Strg><E> zum Beenden des Programms), die Menütaste sowie die Tasten zum Bewegen des Cursors ("Pfeiltasten"). Die Buchstaben- und die Zifferntasten werden wir im Spiel nicht benötigen, aber vielleicht haben Sie eines Tages eine Verwendung dafür, deshalb habe ich sie in den Quelltext mit eingebaut. Sie machen sich im Moment lediglich mit einer Meldung bemerkbar.

Die Prozedur *Handle_keyevent()* verteilt aus Gründen der Übersichtlichkeit die einzelnen Events einfach nur weiter. Es gibt zwei Unter Routinen, die sich letztlich mit den Tastatur-Events beschäftigen:

1. *Handle_chars:(char&)* – Das ist die Sprungzentrale für die Shortcuts: Der per Tastatur gegebene Befehl wird an das zuständige Unterprogramm weitergeleitet. Dabei gibt es eine Besonderheit. Der Übergabe-Parameter nämlich, der den Tastenwert enthält, wird bereits in *Handle_keyevent()* speziell vorbereitet. Man muss dazu folgendes wissen:

Wird eine Taste normal gedrückt, bekommt man den ASCII-Wert des gedrückten Zeichens, für den Großbuchstaben "A" also die 65. Die Kombination aber von <Strg> und Buchstabentaste liefert die Ordnungszahl des Buchstabens im Alphabet, "A" also eine "1", "B" eine "2", usw. . . Das unangenehme: der kleine Buchstabe "a" ergibt ebenso eine "1" wie der Großbuchstabe.

Den Unterschied findet man nur heraus, wenn man auch den Zustand der Shift-Taste prüft. Wurde sie gedrückt, addiert man zur Ordnungszahl die Zahl 64, denn

der ASCII-Wert von "A" ist 65 (64 + 1). Wurde sie nicht gedrückt, wird 96 addiert, denn der ASCII-Wert von "a" ist 97 (96+1).

2. *Handle_hiKeyValues:(code&)* – Das ist die Sprungzentrale für die Menü-Taste und alle Tasten, die sich auf den Cursor auswirken.

Alle Tasten und Tastenkombinationen, die nicht in den drei Prozeduren berücksichtigt werden, fallen unter den Tisch. Beispielsweise werden weder Komma und Punkt noch die Klammern berücksichtigt. Sie müssen bereits in *Handle_keyevent:()* darauf Rücksicht nehmen, wenn Sie diese Zeichen verwenden wollen!

Stifteingaben

Die Routine, die alle Pen-Events verteilt, heißt *Handle_penevent:()*. Hier wird zuerst geprüft, ob der Stift in einem Fenster oder auf dem Toolbar gelandet ist.

In beiden Fällen wird der Parametersatz zuerst an *TBarOffer%:()* übergeben, um herauszufinden, ob auf den Toolbar getippt wurde. Im positiven Fall werden die Parameter verarbeitet und eine entsprechende Aktion hervorgerufen: entweder ...

- die Liste der offenen Anwendungen wird gezeigt oder
- die Uhranzeige wird verändert oder
- die zu einem Button gehörende selbstgeschriebene Prozedur wird aufgerufen. Der Namensaufbau für diese Prozeduren ist vorgeschrieben, der allgemeine Name lautet "Cmd*%:". Der Stern steht für den Buchstaben, der im Programm bei der Toolbar-Initialisierung für jeden Button mit *TbarButt:()* festgelegt wurde. Haben Sie dort einen Großbuchstaben eingesetzt, lautet der allgemeine Name "CmdS*%:"; OPL unterscheidet bei Prozedurnamen nicht zwischen Groß- und Kleinschreibung, über das eingeschobene "S" wird die Unterscheidung indirekt möglich.

Wurde anstatt auf den Toolbar auf eine andere Stelle des Bildschirms getippt, ist die Stunde der Verarbeitungsroutine *Pen_event:()* gekommen. Hier werden mit Hilfe der Parameter die Fenster-Id sowie die x- und y-Position des Stiftes übernommen. Anhand dieser Werte können weitere Entscheidungen getroffen werden. Das Demo-Programm zeichnet lediglich einen 5 Pixel breiten Punkt an der Stiftposition. Der Text am Bildschirm meldet, in welchem Fenster sich der Stift gerade befindet.

Trifft der Stift auf eines der Felder der Funktionsleiste, öffnet das Demo-Programm die Menü-Leiste oder gibt einfach nur eine Meldung aus.

Das Menü

Neben einem Toolbar gehört ein Menü in (fast) jede eigene Anwendung. Der Grund: Menüs sind dem Anwender vertraut, demzufolge erwartet er auch welche. Außerdem: Mit dem Menü ist die Funktion "Programm beenden" über den Shortcut <Strg><E> verbunden, diese Funktionalität wird vom Anwender ebenfalls vorausgesetzt. Alles andere verunsichert nur unnötig.

Üblicherweise ist das Menü entweder über die Taste "Menü" oder über die Funktionsleiste am linken Bildschirmrand zugänglich. So ist es auch in unserem Rahmen-Programm vorbereitet. Die eigentliche Menü-Prozedur heißt *Zeige_Menu*.

Die Konstruktion des Menüs bietet keine Besonderheiten. Lediglich die Verarbeitung der Shortcuts wird anders vorgenommen als das in Psions OPL-Handbuch beschrieben wird. Das Ergebnis der Menü-Auswahl wird nämlich an den "Verteiler" *Handle_chars()* übergeben, so kommt am Ende die eigentlich zuständige Prozedur zum Zuge.

Sie haben nun einen funktionierenden Rahmen zur Hand, den Sie für eigene Zwecke verwenden können. Wie der Rahmen prinzipiell arbeitet, wurde besprochen. Was die einzelnen Funktionen des Programmes leisten, erschließt sich Ihnen beim Experimentieren und Durchsehen des Quelltextes.

Etliche Unterprogramme wurden zunächst nur als Dummy angelegt und zeigen einfach nur eine Meldung. Hier können Sie jeweils Ihren eigenen Code einfügen.

Mehr

Wenn Sie weitere Anregungen für Rahmen benötigen, analysieren Sie die Muster von Alan Ritchie ("RMRevent", <http://www.rmrsoft.com>, unter: Software für das EPOC-System 5) oder auch die an Bord von Emulator und Psion befindlichen Demo-Applikationen TBARAPP und TOOLBAR, die auch als OPL-Quelltexte einzusehen sind.

Für ganz Neugierige sei auch das Programm REVTRAN von Mike Rudin empfohlen, mit dem sich in OPL geschriebene Applikationen und OPO-Programme rückübersetzen lassen (<http://www.cix.co.uk/~mrudin/>). Um dieses Programm gibt es eine Reihe von "moralischen" Diskussionen: Soll man in fremde Programme hineinschauen dürfen oder nicht? Nun – das Programm existiert, die moralische Institution sind nur Sie selber. Übrigens: Programmierer können sich gegen diese Analysemöglichkeit schützen (Prinzipien siehe unter: http://home.snafu.de/krinzner/antirev_d.htm).

C - Das Programm planen

Was und wie?

Noch bevor Sie die ersten Programmzeilen notieren, müssen Sie sich im Klaren darüber sein, was eigentlich programmiert werden soll. Dabei spielt es keine Rolle, ob Sie sich ein Business-Programm vornehmen oder ein Spiel.

Stellen Sie sich vor, Sie sollen für einen Auftraggeber ein Programm schreiben. Welche Fragen würden Sie an ihn richten, damit Sie diesen Auftrag erfüllen können? Es sind dieselben Fragen, die Sie für Ihre eigenen Programme auch an sich selber stellen und beantworten müssten. Verkürzt lauten sie: WAS und WIE?

Aus dem WAS? entnehmen Sie das zu lösende Problem. Sie können daraus auch gleich ableiten, ob OPL überhaupt das richtige Mittel für die Lösung ist. Beispielauftrag: "Programmieren Sie einen Währungsumrechner Euro/US-Dollar"

Aus dem WIE? erfahren Sie, welche Wünsche dahinter stecken. Meist haben Sie oder der Auftraggeber bereits eine Vorstellung davon, was das Programm leisten soll. Präzisieren Sie die Vorstellungen – am besten schriftlich in einer Programmbeschreibung, insbesondere, wenn ein Auftraggeber mit im Spiel ist. Das kann so aussehen:

Einzelheiten für ein Programm "Währungsumrechner":

- das Programm soll Währungskurse von Euro in US-Dollar und umgekehrt berechnen können;
- erforderlich sind Eingabemöglichkeiten für Euro und US\$ in Form einer Zahlenfeldeingabe;
- die Eingabe-Währung muss jederzeit schnell umschaltbar sein;
- gerechnet wird immer erst nach dem Druck auf einen speziellen Button;
- das Ergebnis soll in zwei getrennten Ausgabefeldern dargestellt werden, die Angabe soll nach dem Komma dreistellig sein;
- der Umrechnungsfaktor muss umstellbar sein und ständig angezeigt werden;
- der Währungsrechner soll ausschließlich auf einem Revo laufen.

Die Liste ist noch sehr grob und wird sich im Laufe der Arbeiten erweitern und verfeinern. Bleiben Sie im Kontakt mit Ihrem Auftraggeber und stimmen Sie Änderungen mit ihm ab. Nicht vergessen: Die Liste ist Ihre Arbeitsbasis und Ihre Rückversicherung im Streitfall!

Auf der Basis dieser Liste können Sie daran gehen, daraus eine Bedienoberfläche und eine Programmstruktur zu entwickeln. Den Rahmen haben Sie schon, Ihnen fehlen "nur noch" die eigenen Routinen.

Ich zeige Ihnen jetzt an einem Adventure-Game exemplarisch und ausführlich, was alles zu bedenken ist.

Das Abenteuer beginnt

In vielen Fällen hat es sich bewährt, vom Groben zum Feinen zu planen. "Top down" nennen die Profis diese Technik, die eine von mehreren Methoden für die Planung von Softwareprojekten ist.

Am Anfang steht immer ein leeres Blatt Papier. Es folgt der kreative Akt, in dem man die Vielzahl an Ideen niederschreibt, die man im Kopf mit sich herumträgt. Um die Ideen in die Tat umsetzen zu können, muss man sie ein wenig ordnen. Die tragende Geschichte sollte dabei schon ziemlich genau bekannt sein und man sollte auch wissen, welche Rätselnüsse man versteckt und welche Gegenstände beteiligt sind. Außerdem müssen auch erste Vorstellungen vorhanden sein, mit welchen Mitteln der Spieler das Ziel des Spieles erreichen kann.

Die Frage: "Kann ich das überhaupt technisch umsetzen?", beantworten Sie am besten erst nach der Kreativ-Phase, sonst schränken Sie Ihre Phantasie unnötig ein. Andererseits: bleiben Sie auf dem Teppich - als Besitzer eines Gerätes mit Monochrombildschirm müssen Sie keine farbigen Tagträume haben ...

Im folgenden finden Sie meine bereits geordneten und weit ausgearbeiteten Notizen zu dem geplanten Adventure. Falls Sie es nicht wissen sollten: Bei einem klassischen Adventure Game geht es darum, Gegenstände an einer Stelle zu finden und einzusammeln, um sie an einer anderen Stelle nutzbringend zu verwenden; das ganze ist kein Selbstzweck, sondern dient der Erlangung eines hochwertigen virtuellen Zugewinns (schöne Frauen, Reichtum, Macht, Kindergartenplatz ...) ;-)

Story & Co.

Die grundlegende Programmbeschreibung formuliert sich so:

"Ein Spieler soll sich durch Räume bewegen, in denen er Gegenstände beschafft, die ihm beim Erreichen des Ziels unterstützen. Sein Ziel ist es, ein Testament zu finden, das ihn – zumindest virtuell – reich macht. Der Weg dahin ist mit etlichen Stolpersteinen versehen."

Dazu erzähle ich diese Geschichte:

"Du bist der Erbe einer wunderschönen Villa. Dein Onkel hatte Dir das Erbe bereits angekündigt, doch nun, nach seinem Tode, ist das verflixte Testament einfach nicht aufzutreiben. Aber Du bist ganz sicher, es befindet sich im Haus des Onkels! So machst Du Dich also auf die Suche ..."

Parallel dazu habe ich die Gedanken zu den Örtlichkeiten und den Rätseln entwickelt. Dabei habe ich mir auch gleich überlegt, wo ich welche "Ostereier" verstecke. Herausgekommen sind die Beschreibung einer "Villa" mit Räumen, die Liste der Objekte und ihrer Verstecke, ein Lösungs-Szenarium sowie die Kommando-Liste für die Spielsteuerung.

Der Ort

Abbildung C1 zeigt den Grundriss der "Villa". Schon beim Entwurf ist darauf zu achten, dass es für den Spieler logisch zugeht, er muss in der Lage sein, sich den "Lageplan" des Spieles zu erarbeiten, sonst schmeißt er ganz schnell die Flinte ins Korn. Er soll aber schließlich die Chance haben, die Spiel-Arena als Sieger zu verlassen.

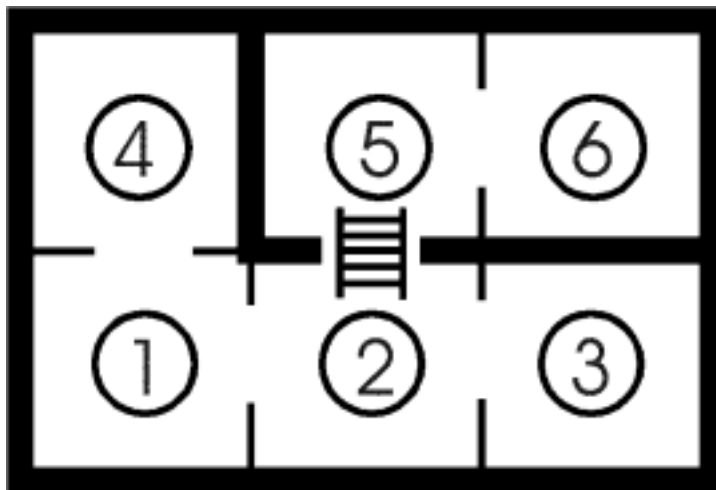


Abb. C1

Sie erkennen in der Skizze die Raumnummern, die wir bei der Programmierung zur Hilfe nehmen werden, die Tür-Durchbrüche und die zwei Geschosse der "Villa". Die Räume 1 .. 4 liegen im Erdgeschoss, 5 und 6 in der oberen Etage, in die man über eine Treppe kommt.

Der Spieler bekommt die Raumnummern nicht zu Gesicht, deshalb geben wir den Räumen gleich noch einen Namen:

RaumNr.	Name
1	Flur
2	Unterer Treppenabsatz
3	Wirtschaftsraum
4	Arbeitszimmer
5	Oberer Treppenabsatz
6	Abstellkammer

Inventar-Liste

Hier ist die Inventarliste, versteckte Gegenstände stehen in Klammern:

RaumNr.	Gegenstände
1	leer
2	Treppe
3	Wandschrank (Kästchen (Schlüssel))
4	Schreibtisch, Brieföffner, Schublade (Kugelschreiber)
5	Treppe
6	Bild (Koffer (Testament))

Für den Spieler geheim: Der Lösungsweg

Ein Großteil der Programmierung hängt vom Lösungsweg ab - so sieht er aus:

"Du startest im Raum 1. Gehe in Raum 4, öffne die Schreibtischschublade, nimm den dort verborgenen Kugelschreiber. Nimm auch den auf dem Schreibtisch liegenden Brieföffner. Gehe zu Raum 3. Öffne den Wandschrank, öffne das zum Vorschein kommende Kästchen. Du findest einen Schlüssel, nimm ihn. Gehe im Raum 2 die Treppe nach oben und öffne dort die Tür (das geht nur, wenn Du den Schlüssel aus

dem Kästchen genommen hast). Betrete Raum 6, verschiebe das Bild an der Wand. Zum Vorschein kommt ein Koffer. Öffne ihn mit dem Brieföffner, dadurch wird das Testament zugänglich. Das wird erst gültig, wenn man es mit dem Kuli aus der Schreibtischschublade unterschreibt."

Kommando-Sachen

Während ich den Lösungsweg notierte, habe ich automatisch Verben benutzt, die der Spieler als Kommandos wiederfinden sollte, damit er das Spiel steuern kann:

- umsehen (zeigt Raumbeschreibung)
- ansehen (beschreibt Gegenstände)
- öffnen (öffnet Türen und Gegenstände)
- nehmen (nimmt Gegenstände)
- bewegen (verschiebt Gegenstände)
- benutzen (verwendet Gegenstände, die man im Besitz haben muss)
- anklopfen (eine Scherzfunktion)

Die Bildschirmgestaltung

Nachdem die ersten Bausteine zusammengetragen waren, habe ich sie noch mit einem Entwurf für die Bildschirm-Oberfläche garniert. Abbildung C2 präsentiert das Ergebnis. Neben dem Lösungsweg liefert diese Skizze weitere Anhaltspunkte für die Ausführung der Programmierung.

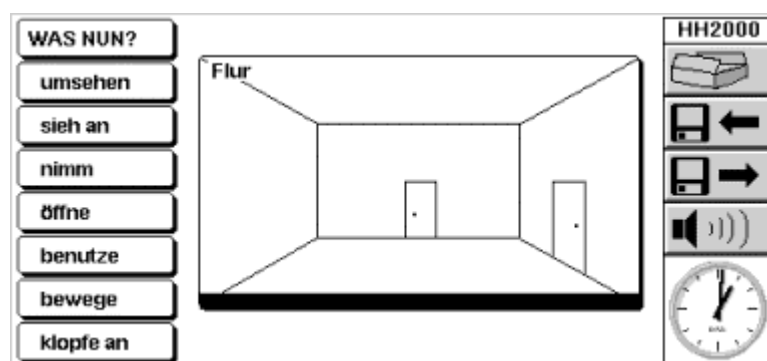


Abb. C2

Meine Vorstellung, wie die einzelnen Felder reagieren sollen:

Kommandos werden ausgeführt, wenn man mit dem Stift auf die Befehlsfelder links drückt. Einen Raumwechsel erreicht man durch einen Stift-Tipp in die Nähe von Türen. Andere wichtige Funktionen werden mit Hilfe des Toolbars ausgelöst oder per Menü aktiviert (Spielstand speichern und laden, Inventar ansehen, ...).

Folgende Menüs soll es geben:

Datei: Spielstand laden, Spielstand speichern, Spiel beenden

Ansicht: Inventar, Toolbar ein

Optionen: Ton an/aus, Tondauer

Extras: Hilfe, Warnung, Über HH2000

Ein Grobplan

Aus den oben genannten Vorstellungen lassen sich die ersten Forderungen an das Programm ableiten, insbesondere aus den Vorstellungen zur Bedienoberfläche. Wir brauchen:

1. einen Toolbar mit den Funktionen:
 - Inventar anzeigen,
 - Spielstand speichern,
 - Spielstand laden,
 - Ton an/aus, (ab S5xx)
 - Hilfe (nur für S7/netBook)
 - Über ... (nur für S7/netBook)
2. Menüs mit den oben beschriebenen Funktionen (Hinweis: die Punkte "Ton an/aus" und "Tondauer" ändern einen Piepston, der immer dann zu hören ist, wenn man einen Weg zu gehen versucht, der nicht begehbar ist.),
3. eine Kommandoleiste: jedes Kommando bekommt ein Fenster zugeordnet, das als Button fungiert,
4. ein Fenster für die Darstellung der Räumlichkeit; die Räume und deren Inhalte werden mit Grafikbefehlen gezeichnet; gezeichnet werden drei Wände, die Decke und der Fußboden – die vordere Wand ist durchsichtig, eine Tür wird durch einen Durchbruch im Türschwellen-Streifen gekennzeichnet; der Raumname soll in die linke obere Ecke,
5. Durch die Räume bewegt man sich, indem man mit dem Stift in die Nähe der Türen tippt, wahlweise können die Pfeiltasten benutzt werden.

Anschließend geht es um die "inneren Werte".

Das Spiel wird hauptsächlich über die Kommandos vorangetrieben. Jedes Kommando bekommt eine eigene Prozedur, auch die Toolbar-Kommandos. Wie verknüpft man aber nun die Kommandos mit den spielwichtigen Objekten? Bei der Erfindung von "Haunted House" ist mir dazu diese Variante eingefallen:

Man ordnet den Objekten eine Reihe von "Eigenschaften" zu, die ihren Aufenthaltsort und Zustand beschreiben sollen. Diese "Eigenschaften" werden dann durch die Kommando-Prozeduren ausgewertet oder auch verändert.

Ausgewertet wird immer dann, wenn der Spieler etwas wissen will:

- was befindet sich im Raum? ("umsehen"),
- welche Information gibt es über den Gegenstand? ("sieh an"),
- welches Objekt "besitzt" man? ("Inventar").

Alle anderen Kommandos verändern die "Eigenschaften" und führen zu einem neuen "Zustand" des Objektes.

Damit das Programm die Objekt-"Eigenschaften" verarbeiten kann, habe ich folgendes Prinzip verwendet:

- jedes Objekt wird durch eine Nummer identifiziert,
- die "Eigenschaften" der Objekte legt man über Integer-Zahlen fest, die in Array-Variablen gespeichert sind. Dadurch, dass die Objektnummern als Index für die Array-Variablen dienen, wird die Zuordnung zu den Gegenständen hergestellt.

Ein Beispiel verdeutlicht das:

Das Objekt "Kugelschreiber" hat in unserem Demo-Programm die Objektnummer "3". Ob ein Objekt gerade sichtbar ist oder nicht, regelt die Array-Variable *sichtbar%*(*i*). Ist der Index der Variablen "3", beschreibt ihr Wert den Zustand des "Kugelschreibers". Wenn also *sichtbar%(3) = KTrue%* ist, ist der "Kugelschreiber" tatsächlich sichtbar.

In diesem Sinne sind noch eine ganze Reihe von weiteren "Eigenschaften" festzulegen, die ich im nächsten Kapitel beschreibe.

Damit ist die Grobplanung abgeschlossen.

Teufel steckt im Detail

Der nächste Schritt besteht darin, die Grobplanung solange zu verfeinern, bis man die Programmieraufgabe im Detail beschreiben kann. Durch die Verfeinerung tastet man sich also Zug um Zug an die Lösung der gestellten Aufgabe heran. Für jedes Problemchen muss ein begehbarer Weg her!

Oft sind einzelne Probleme mit anderen Problemen komplex verknüpft, deshalb kann es gut sein, dass die erste Lösung nicht die endgültige ist.

Ich möchte Ihnen hier nicht die bekannten Theorien nahe bringen – die füllen allein schon ...zig Bücher. Es reicht, wenn Sie die Lösung Ihrer Aufgaben sinngemäß und mit wachem Verstand umsetzen ...

An einem einzelnen Beispiel stelle ich Ihnen jetzt vor, wie man vorgehen kann. Die gesamte Grobplanung umzusetzen, dafür reicht der Platz hier einfach nicht aus. Im nächsten Kapitel werde ich Ihnen jedoch noch ausführlich erläutern, wie sich meine Überlegungen im fertigen Programm niedergeschlagen haben.

Als Beispiel für den Verfeinerungsprozess dient mir das Kommando "umsehen".

Wie alle anderen Kommandos wird dafür eine eigene Prozedur bemüht, die ich einprägsam "*Umsehen*:" getauft habe. Sie wird über einen Pen-Event aufgerufen, und zwar immer dann, wenn der Spieler auf den richtigen Button tippt.

Der Verfeinerungsprozess besteht jetzt darin, herauszufinden, was in der Prozedur alles passieren muss, damit ein Text mit der Raumbeschreibung am Bildschirm gezeigt wird.

Die einfache erste Untergliederung, die mir spontan in den Sinn kam, war:

1. man muss den auszugebenden Text "besorgen"
2. der Text muss angezeigt werden

Das ist zwar richtig, aber immer noch sehr grob - beide Punkte müssen inhaltlich noch weiter untersetzt werden. Im Hinterkopf fließen bei dieser Planung die eigenen OPL-Erfahrungen mit ein, ohne dass man sofort auch die fertige Lösung parat haben muss. So kam ich auf diese Verfeinerung:

- Den Text holt man aus einem Text-Pool – einem Stringvariablen-Array. Durch die Einbeziehung der aktuellen Raumnummer sichert man, dass der Text am Ende auch der richtige ist. Idee zur Ausführung: man baut eine auf die Raumnummer bezogene IF-ELSEIF-ENDIF-Abfrage auf oder arbeitet mit dem VECTOR-Befehl.
- Wenn Text als String vorliegt, kann die Zeichenkette bis zu 255 Zeichen lang sein. Damit passt der Text mit Sicherheit nicht auf den Bildschirm und muss zerlegt werden. Da der Textbildschirm durch andere Fenster und Grafiken verdeckt wird, muss die Anzeige in einem eigenen Grafikenfenster oder mit Hilfe von DIALOG-Befehlen dargestellt werden.

Mit diesen Ideen könnte die Prozedur *Umsehen*: in "Pseudo-Code" so aussehen:

```
PROC Umsehen:
  REM UP: = "Unterprogramm"
  UP: Hole den zum Raum passenden Text
  UP: Teile den Text in bildschirmgerechte Zeilen
  UP: Stelle den Text am Bildschirm dar
ENDP
```

Umsehen: verteilt definierte Arbeitsschritte wegen der Übersichtlichkeit und einer möglichen Mehrfachnutzung an weitere Unterprogramme, von denen wir nicht gleich wissen müssen, wie sie intern aussehen. Die Entwicklung der einzelnen Programmteile bleibt dem Arbeitsgang "Programmieren" vorbehalten, der hier notierte "Pseudo-Code" beschreibt die Teilaufgabe nur.

Die schließlich in HH2000 verwendete Prozedur hat nach dem Ausprogrammieren dann auch die gerade formulierte Struktur:

```
PROC Umsehen:
  Teile_Satz: (@$ ("Raumbeschreibung") : (raum%))
  Textausgabe:
ENDP
```

Für den "Hausgebrauch" sollte dieser Grad an Verfeinerung reichen, auch wenn noch nicht alle Details herausgearbeitet worden sind. Ganz anders ist das, wenn an einem Projekt mehrere Leute professionell arbeiten. In solchen Fällen muss die Aufgliederung noch weit detaillierter ausfallen. Aber das gehört dann auch in ein anderes Manuskript ...

D - Das Spiel programmieren

Verändertes Rahmenprogramm

Im vorigen Kapitel wurden die Rahmenbedingungen für das Spiel erläutert. Mit diesem Wissen habe ich das Programm durchgeplant, das Ergebnis finden Sie mit allen Einzelheiten in diesem Kapitel.

Obwohl sich die Beschreibungen hier stark auf das Spiel konzentrieren, können Sie etliche Tipps für eigene Programme entnehmen, z.B.:

- Laufwerke erkennen und verwenden,
- Initialisierungsdateien anlegen und benutzen,
- umfangreiche Deklaration von Variablen managen,
- Toolbar mit Grafik verwenden,
- Datenbanken anlegen sowie Werte speichern und laden,
- eine Hilfedatei verwenden,
- Beispiele für individuelle Pen-Event-Routinen.

Das Gesamtprogramm "HH2000.opl" finden Sie im Anhang. Sie nutzen die nachfolgenden Zeilen am besten, wenn Sie parallel die Programmzeilen mitverfolgen und analysieren.

Wenn Sie nun den Quelltext durchgehen, fallen Ihnen sicherlich die Änderungen und Ergänzungen am Programm-Rahmen auf. Das sind die Details:

Programm-Kopf

- Die Icons für die Applikation wurden ergänzt.
- "System.OXH", die Header-Datei für die Einbindung von "System.OPX", wurde im INCLUDE-Teil eingefügt (ist für die Benutzung der Hilfe-Funktion erforderlich).

PROC Begin:

- Der Parameter für *TBarLink:()* heißt nicht länger "Main:", sondern "*Declare_and_init*". In der nun angesprungenen Prozedur *Declare_and_init*: liegen alle unsere eigenen Deklarationen und Initialisierungen der verwendeten Globalvariablen. Erst am Ende dieser Prozedur wird zu *Main*: gesprungen. Der einfache Grund: Bequemlichkeit. Der Umfang der deklarierten Variablen hat beträchtlich zugenommen, so dass ein aufgeblasenes Gebilde entsteht, dass beim Arbeiten am Quelltext durch notwendiges Hin- und Herrollen nur stört. Die Verlagerung ans Quelltextende schafft Abhilfe.

Diese Bequemlichkeit ist auch der Grund dafür, dass selten gebrauchte Routinen an das Ende des Quelltextes gerückt wurden.

PROC Main:

- Wie gerade erläutert, wurde die Variablendeklaration an das Quelltextende verbannt.
- Die spieleigenen Programmteile wurden hinter der Event-Analyse eingeordnet. Der erste Teil davon wird immer dann wirksam, wenn ein anderer Raum betreten wird oder sich im Raum etwas verändert - ich komme noch darauf zu sprechen. Der zweite Teil leitet das Finale ein, wenn das Spiel gewonnen ist; die Variable *gewonnen%* muss ungleich Null sein, damit das Finale abläuft.

PROC Handle_hiKeyEvent:()

- Aktiv werden nur noch die Menü-Taste und die Pfeil-Tasten verarbeitet.
- Mit Hilfe der Pfeil-Tasten wandert man zwischen den Räumen hin und her, entsprechend wird die Raumnummer verändert. Das Zusammenspiel mit der Prozedur *Corr:()* klären wir noch auf.

PROC Handle_chars:()

- Das Prinzip der Verarbeitung von Tastendrücken blieb erhalten. Angepasst wurden lediglich die Buchstaben und die Sprungziele.

PROC Handle_penevent:()

- Wenn es sich jetzt um einen Pen-Event handelt, der vom Bildschirmbereich außerhalb des Toolbars stammt, wird nur noch das Abheben des Stiftes berücksichtigt: Üblicherweise lösen auch das Aufsetzen und das Bewegen des Stiftes einen Event aus. Die Beschränkung auf das Abheben hat einen Vorteil. Beispiel: Sie tippen versehentlich auf einen Button, der das Programm beendet ... Sie retten die Situation, indem Sie den Stift einfach zur Seite in einen anderen Bereich ziehen und dann erst abheben!

```
PROC Handle_penevent: (key&, a3&, a4&, a6&, a7&)
  REM Behandelt alle Pen-Events
  IF key&=KEvPtr&
    IF TBarOffer%: (a3&, a4&, a6&, a7&)
      RETURN
    ELSEIF a4&= KEvPtrPenUp&      REM Stift hat abgehoben
      Pen_event: (a3&, a6&, a7&)
    ENDIF
  ELSEIF
    REM etc.
  ENDIF
ENDP
```

PROC Penevent:()

- Die Prozedur prüft nun, in welchem der etlichen geöffneten Fenster sich der Stift befindet und gibt entsprechend dem Ergebnis die Verarbeitung an Unterrouتين ab. Die Einzelheiten bespreche ich später.

PROC Zeige_Menu: und PROC Cmdxx%:

- Leichte Anpassungen an die Gegebenheiten des Spiels

PROC Spiel_laden:, PROC Spiel_speichern: und PROC Hilfe:

- Die Prozeduren sind nun dem Dummy-Status enthoben.

PROC Init:

- Es mussten einige neue Elemente hinzugefügt werden, diesem Thema widme ich einen eigenen Abschnitt.

PROC Setup_Toolbar:

- Die Toolbars werden mit Icon-Elementen versehen. Abweichend von Symbians Design-Richtlinien nehmen die Icons die gesamte Button-Fläche ein. Dahinter steckt die Idee, bei möglichen Sprachversionen des Programms einfach nur die Bilder sprechen zu lassen. Die Grafik-Vorlagen und Abmessungen finden Sie im Anhang bei den Quelltexten.

PROC Toolbar:, PROC Show_Toolbar: und PROC Hide_Toolbar:

- Weil es für das Spiel keinen Grund gibt, den Toolbar auszublenden, wurden diese Prozeduren entfernt.

Sie kennen nun in groben Zügen die Veränderungen, die das Rahmen-Programm erfahren hat. Ich gehe nun auf die erforderlichen Initialisierungen genauer ein.

Erweiterte Initialisierung

Mit der neuen Prozedur *Declare_und_init*: werden nur die Global-Variablen initialisiert. Darüber hinaus sind weitere Vorbereitungen zu treffen, die alle in der Prozedur *Init*: erledigt werden. Und das passiert:

1. Es werden zwei Verzeichnisse auf dem Laufwerk C: angelegt:

- C:\HH2000, hier hinterlegt man eine Initialisierungsdatei ("hh2000.ini"), die beim Programmstart ausgelesen wird, um die zuletzt gemachten Voreinstellungen des Programmes zu übernehmen.

- C:\HH2000\bin, in diesem Verzeichnis landen die Spielstandsdateien

Der TRAP-Befehl sorgt dafür, dass eine Fehlermeldung unterdrückt wird, wenn die Verzeichnisse bereits existieren.

2. Einen Teil der Toolbar-Vorbereitungen kennen Sie bereits, ...

... nämlich die Feststellung des Gerätetyps. Neu hinzu kommt die Ermittlung des Laufwerkes, auf dem die Applikation installiert ist. Die aufgerufene Routine *Laufwerk:()* macht sich zunutze, dass alle zum Spiel gehörigen Original-Dateien im gleichen Verzeichnis wie die Programmdatei "HH2000.app" liegen.

Unser Programm funktioniert nun auch dann korrekt, wenn es auf D: oder E: installiert wird. Auf dem Revo können die Dateien natürlich nur auf Laufwerk C: liegen.

Nach dieser Vorbereitung wird der Toolbar in schon bekannter Art initialisiert. Jetzt kommt die Laufwerkbestimmung zum Tragen, denn die Icons werden auf dem richtigen Laufwerk gesucht und gefunden.

3. Über die Prozedur *Load_ini*: wird die Initialisierungsdatei geladen.

Bei uns enthält sie lediglich die Information, ob die Piepser ein- oder ausgeschaltet sein sollen. Reguliert wird die Piepser-Funktion über die Variable *ton%*.

Beim ersten Spielstart existiert die ini-Datei noch nicht und wird deshalb automatisch mit Hilfe der Prozedur *save_ini*: angelegt. Dadurch entsteht eine schlichte Datenbank, die nur einen einzigen Wert enthält. Beim ersten Mal wird eine "-1" (*KTrue%*) dort abgelegt, der Wert kann aber über das Menü jederzeit auf Null (*KFalse%*) umgestellt werden. Die zugehörige Prozedur heißt *Ton_an_aus*:. Sie ruft bei jeder Veränderung wieder *Save_ini*: auf. *Save_ini*: löscht zuerst die alte Datenbank und legt sie dann mit dem neuen Wert wieder an.

Save_ini: wird auch von der Prozedur *Exit*: kurz vor der Programm-Beendigung noch einmal aufgerufen, um die aktuelle Einstellung aufzubewahren. Hinweis: Eigentlich wäre das nicht notwendig, da die Arbeit bereits *Ton_an_aus*: übernommen hat. Der Aufruf von *Save_ini*: steht in diesem Falle mehr als

beispielhafte Begründung für die Existenz von *Exit*., denn hier könnten auch weitere Aktionen untergebracht werden. Ich denke da z.B. an die automatische Abspeicherung des aktuellen Spielstandes.

4. Die Fenster, die zur Spielsteuerung über die ...

... Kommandos (also die Verben) dienen, werden angelegt und auf dem Bildschirm platziert. Zuständig ist die Routine *Setup_commandWin*.. Wichtig: die Fenster-Id's werden in dem Array *cmdWin%()* abgelegt. Diese "Fensternummern" helfen später bei der Identifizierung der Pen-Events.

5. Das Spielfenster, die "Bühne" wird ...

...durch die Prozedur *Setup_gameWin*.. eingerichtet. In Wirklichkeit handelt es sich sogar um zwei Fenster. Das leer bleibende Blindfenster (*blindWin%*) dient nachher als "Bühnenvorhang", denn immer, wenn das eigentliche Spielfenster (*gameWin%*) neu gezeichnet wird, wird zuerst das Blindfenster als Sichtschutz vor das Spielfenster gestellt. Das vermeidet, dass der Benutzer das Löschen und den Bildneuaufbau als holprig empfindet.

6. Zum Schluss wird der Spielbereich zum ersten Mal gezeichnet.

Benutzt wird dabei die Prozedur *Zeichne_Raum*., die über verschiedene Variablen-Werte weiß, wie der Raum aussieht und welche Türen zu malen sind. Über das Unterprogramm *Setze_Titel*.. findet der Raumtitel den Weg auf den Bildschirm.

Nach abgeschlossener Initialisierung läuft unser Programm endlich in die große endlose Schleife ein - das Abenteuer für den Spieler nimmt seinen Lauf.

Objekte verwalten

"Objekte", das sind die Gegenstände im Spiel, sind zentrale Elemente - um sie dreht sich so ziemlich alles. Sie sollen auf dem Bildschirm dargestellt werden, der Spieler muss ihre Eigenschaften kennenlernen und sie manipulieren können. Damit all das möglich wird, beschreibt man Objekte programmintern mit geeigneten Variablen, die über ihre Eigenschaften Auskunft geben sollen. Insbesondere die numerischen Variablen helfen, die Behandlungsroutinen einfach zu fassen. Zur Erinnerung – ich sprach schon im vorigen Kapitel darüber:

Das Prinzip: Alle Objekte bekommen eine individuelle Nummer, mit der sie zu identifizieren sind. Die Eigenschaften der Objekte werden durch numerische Array-Variablen beschrieben, wobei der Index dieser Variablen die Zuordnung zum Objekt herstellt.

Beispiel: Der Kugelschreiber hat die Objektnummer 3, deshalb beschreibt die Eigenschaftsvariable `sichtbar%(3)` durch ihren Wert, ob der Kugelschreiber gerade sichtbar oder unsichtbar ist.

Die folgende Liste benennt die anderen benutzten Variablen und beschreibt ihre Funktion am Beispiel des Objektes "Kugelschreiber" (Variable `kuli%`):

- Jedem Gegenstand ist eine Objektvariable zugeordnet, z.B.: `kuli%= 3`. Auf diesem Wege kann man das Objekt ansprechen, ohne sich seine Nummer merken zu müssen. Prinzipiell kann man auch mit der blanken Objekt-Nummer arbeiten.
- `objekt$()` enthält den Namen des Gegenstandes, diese Variable benötigt man u.a. zur Textausgabe. Beispiel: `objekt$(kuli%)= "Kugelschreiber"`
- `uart$()` ist der unbestimmte Artikel des Gegenstandes; zusammen mit `objekt$()` wird daraus z.B. `uart$(kuli%) + objekt$(kuli%) --> "ein Kugelschreiber"`; findet Verwendung u.a. bei der Anzeige der gesammelten Gegenstände, dem "Inventar".
- `sichtbar%()` bekommt den Wert WAHR (`KTrue%`), wenn der Gegenstand theoretisch sichtbar ist (nicht immer wird auch alles am Bildschirm dargestellt) oder bereits in der eigenen Tasche steckt. Umgekehrt formuliert: das Objekt ist nicht verdeckt, versteckt oder durch Benutzung "im Äther" verschwunden. Ist der Gegenstand nicht "sichtbar", hat die Variable den Wert FALSCH (`KFalse%`). Beispiel: `sichtbar%(kuli%)= KFalse%` bedeutet, dass der Kugelschreiber zu Anfang nicht sichtbar ist, denn er ist in der Schublade versteckt. Das Öffnen der Schublade macht ihn sichtbar. Nur wenn er sichtbar ist, ist auch seine Beschreibung zugänglich und er kann genommen und benutzt werden.
- `imRaum%()` enthält den Aufenthaltsort, also die aktuelle Raumnummer des Gegenstandes; ist der Wert Null, wurde das Objekt weggenommen oder anderweitig entfernt. Beispiel: Der Kugelschreiber ist beim Programmstart im Raum 4: `imRaum%(kuli%)= 4`
- `nehmbar%()`, `genommen%()`: beschreibt genau das, was die Namen der Variablen ausdrücken; mögliche Werte sind `KTrue%` und `KFalse%`. Beispiele: `nehmbar%(kuli%)= KTrue%` und `genommen%(kuli%)= KFalse%`. Obwohl der Kugelschreiber zu Programmbeginn nicht sichtbar ist, so kann er doch prinzipiell genommen werden.
- `nehmbarMax%`: gibt die maximale Anzahl der nehmbaren Objekte an. Der Sinn: Die Objektnummern sind so aufgestellt, dass die nehmbaren Objekte die kleinsten Nummern haben, dann folgen die "fest montierten". Wird nun die Liste der bereits gesammelten Gegenstände abgerufen, werden die Objekte in Reihenfolge ihrer Nummern in aufsteigender Reihenfolge auf den Zustand "genommen" untersucht. Weil nur "nehmbare" Objekte auch genommen werden können, kann die Suche nach der Objektnummer nach `nehmbarMax%` abgebrochen werden. Insbesondere bei großen Listen stellt diese Beschränkung einen Geschwindigkeitszuwachs bei der Abarbeitung der zugehörigen Routine dar. Bei uns werden beispielsweise anstatt aller 14 Objekte lediglich die ersten fünf getestet.

- *oeffnungsStatus%()*: neben den Eigenschaften AUF und ZU wird hier hinterlegt, ob für ein geschlossenes Objekt (Tür, Koffer, Schrank, Schublade, ...) ein "Schlüssel" erforderlich ist; mögliche Werte: 1= zu, 2= zu, braucht Schlüssel zum Öffnen, 4= öffnen.
- *wirt%()*, *gast%()*: helfen bei versteckten Gegenständen. Der "Wirt" ist selber ein Objekt, das ein anderes bei sich aufnimmt, nämlich den "Gast", z.B. : Der Wirt des Kugelschreibers ist die Schublade, *wirt%(kuli%)= schublade%*, und anders herum: der Gast der Schublade ist der Kugelschreiber, *gast%(schublade%)= kuli%*.
- *benutzt%()*, kennzeichnet benutzte Objekte. Benutzte Objekte besitzen eine andere Beschreibung, das ist der Hauptgrund für die Einführung dieser Variablen. Wegen der einheitlichen Systematik werden selbst Objekte, die nach erfolgreicher Verwendung in der Versenkung verschwinden, auf den Status "benutzt" gesetzt, z.B.: *benutzt%(kuli%)= KTrue%*.

Wenn nötig, werden die Variablen in der Prozedur *Declare_and_init*: mit einem Erstwert versehen – man nennt das "initialisieren". Nicht initialisierte numerische Variablen erhalten allein durch die Deklaration den Wert Null (das entspricht *KFalse%*), Stringvariablen enthalten als Wert "Nichts" (sogen. "Leerstrings").

Spaziergang durch die Räume

In HH2000 bewegt sich der Spieler durch Räume. Dazu tippt er entweder mit dem Stift in einen bestimmten Bereich der Spielszene oder er benutzt die Pfeiltasten.

Gehrichtung ermitteln, Raumnummer ändern

Das Raumnummern -System macht es durch die Auswertung von Pen- oder Tastatur-Events ziemlich leicht, zu einer anderen Raumnummer zu wechseln. Die entscheidenden Prozeduren sind: *Handle_hiKeyValues:()* für die Pfeiltasten und *Check_gameWin:()* für die Berührung des Spielfensters mit dem Stift. Die Stiftkoordinaten kommen von *Pen_event:()* und werden in *Check_gameWin:()* auf die gewünschte Bewegungsrichtung analysiert. Danach machen beide Prozeduren das gleiche: sie rufen mit einem Richtungsparameter die Korrekturfunktion *Corr%:()* auf.

Der dort bestimmte Wert wird zu einer neuen Raumnummer verrechnet. Das Prinzip:

```
raum% = raum% + Corr%:(richtungsparameter)
```

Die Richtungsparameter wurden in Abstimmung mit einem anderen Mechanismus festgelegt. Wenn Sie sich die aufgerufene Funktion *Corr%():* ansehen, fällt Ihnen sicherlich die Global-Variable *wege%()* auf, die über eine bitweise AND-Verknüpfung mit dem Richtungsparameter *dir%* verbunden ist. *Corr%():* liefert nur dann einen Wert ungleich Null zurück, wenn das in *wege%()* gesetzte Bit mit dem *dir%*-Bit übereinstimmt. Dahinter steckt folgendes:

Wege%() enthält die zahlenmäßige Nachbildung der Wände unserer Räume. Jeder Wand wird in *wege%()* ein Bit zugeordnet. Ist das Bit gesetzt, ist die Wand durchlässig - in der Spielfeld-Darstellung befindet sich an dieser Stelle zumeist eine Tür. Ein nicht gesetztes Bit bedeutet: hier geht's nicht weiter, hier ist eine Wand oder eine verschlossene Tür. Für die Bit-Belegung gilt:

Richtung Nord:	zuständig ist Bit 0, Wert in <i>wege%()</i> = 1
Richtung Süd:	zuständig ist Bit 1, Wert in <i>wege%()</i> = 2
Richtung West:	zuständig ist Bit 2, Wert in <i>wege%()</i> = 4
Richtung Ost:	zuständig ist Bit 3, Wert in <i>wege%()</i> = 8

Beispiele:

1. Raum 1 hat freie Wege in Richtung Nord und Ost, die gesetzten Bits addieren sich so:

```
wege%(1) = 8 + 0 + 0 + 1 = 9
```

Wollen Sie sich hier nach Norden bewegen, rechnet die Funktion *Corr%():*

```
dir% = 1
dir% AND wege%(1) ergibt: 1 AND 9 = 1 = WAHR
```

Der Rückgabewert ist folglich: *verDiff%* = 3, es folgt deshalb in *Check_gameWin():* bzw. *Handle_hiKeyValues():*

```
raum% = 1
raum% = raum% + Corr%:(1) = 1 + 3 = 4
```

Die neue Raumnummer ist daher 4.

2. Raum 5 hat einen freien Weg nach Süden, der Weg nach Osten besitzt zwar in der Darstellung eine Tür, die ist aber verschlossen:

```
wege%(5) = 0 + 0 + 2 + 0 = 2
```

Der Weg nach Osten ist damit versperrt, der Beweis:

```
dir% = 8
dir% AND wege%(5) ergibt : 8 AND 2 = 0 = FALSCH
```

Das sorgt einerseits für einen Piepser, andererseits für den Rückgabewert Null, weil der ELSE-Zweig in *Corr%()* durchlaufen wird.

```
raum% = 5
raum% = raum% AND Corr%(5) = 5 + 0 = 5
```

Die Raumnummer ändert sich nicht.

Beachte: Die Korrekturwerte *verDiff%* und *horDiff%* sind auf die Anzahl der Räume und deren vertikale und horizontale Aufteilung abgestimmt und müssen bei anderer Raumanordnung angepasst werden.

Was fängt das Programm nun mit der geänderten Raumnummer an? Sehen wir dazu in der Prozedur *Main*: um.

Raum neu zeichnen

Die IF-Bremse

Das Programm erkennt die Veränderung der Raumnummer am Vergleich von *raum%* und *alterRaum%* und tritt in den IF-Zweig ein.

Hier wird gleich zu Beginn die aktuelle Raumnummer in den "Merker" *alterRaum%* übertragen, damit beim nächsten Schleifendurchlauf die Schritte nicht unnötig abgearbeitet werden. Ein Flackern des Bildschirms wäre die Folge, weil ohne die IF-"Barriere" bei jedem Event die Bildschirmdarstellung regeneriert werden würde.

Vorhang auf und zu

Aus ähnlichem Grund wird der dann folgende Neuaufbau des Spielfensters vor dem Spieler verdeckt – man lässt vorher einen Sichtvorhang herunter und zieht ihn wieder herauf, wenn der Vorgang beendet ist. Technisch gesprochen verschiebt man einmal das Blindfenster und dann das Spielfenster in den Vordergrund. Zur Beschleunigung der Arbeit schaltet man auch noch das Bildschirm-Update aus und wieder an:

```
gORDER blindWin%,1
gUPDATE OFF
...
```

```
gUPDATE ON
gORDER gameWin%,1
```

Dazwischen liegen im wesentlichen zwei getrennte Vorgänge:

1. der unmöblierte Raum wird gezeichnet,
2. das Inventar einschließlich sichtbarer beweglicher Objekte wird gezeichnet.

So geht's:

Der Raum nimmt Kontur an

Die Prozedur *ZeichneRaum*: stellt den Raum anhand des Wertes der Variablen *raum%* und weiterer raumspezifischer Variablenwerte selbsttätig dar.

Das grundsätzliche Aussehen der Räume steht fest: drei sichtbare Wände, Fußboden und Decke. Variabel dagegen ist die Anordnung der Türen. Um sie automatisch zeichnen zu lassen, wird jeder Tür in den Räumen eine Variable zugeordnet - ist sie gesetzt, wird eine Tür gezeichnet.

Ein ähnliches System haben wir oben mit der Variablen *wege%()* für die "offenen Wände" benutzt. Um zu zeigen, dass es auch noch etwas anders geht, habe ich jeder Himmelsrichtung ein eigenes Variablen-Array spendiert. Das verschwendet zwar etwas mehr Speicherplatz, ist aber beim Programmieren viel einsichtiger.

TuerN%(), *tuerS%()*, *tuerW%()* und *tuerO%()* – so heißen die vier benutzten Variablen-Arrays. Der Großbuchstabe im Variablen-Namen steht für die Himmelsrichtung. Die Zuordnung der Türen zu den Räumen erfolgt über den Index der Variablen – als Index fungiert die Raumnummer *raum%*.

Beispiel: Der Raum 6 soll eine Tür in Richtung West haben, die Variablen lauten also:

```
raum%= 6
tuerN%(raum%)= KFalse%
tuerS%(raum%)= KFalse%
tuerW%(raum%)= KTrue%
tuerO%(raum%)= KFalse%
```

Die Zuordnung von Raum und Türen wird in der Initialisierungsphase festgelegt (Prozedur *Declare_and_init*:).

Nachdem Wände und Türen gezeichnet wurden, wird durch die Prozedur *ZeichneRaum*: auch noch der Name des Raumes ausgeschrieben (Prozedur *SetzeTitel*:), der mit Hilfe der Funktion *RaumTitel*:() "besorgt" wird. Damit ist die Aufbereitung des leeren Raums abgeschlossen.

Räume möblieren

Jeder Raum hat so seine Besonderheiten. Manche grafischen Bestandteile lassen sich mit der universellen Raum-Zeichenroutine nicht einbinden, zum Beispiel die Treppe

in Raum 2. Außerdem gibt es eine Anzahl von raumspezifischen Variablen, die gesetzt werden müssen. Für alle diese Anforderungen bekommt jeder Raum eine individuelle Prozedur, die von *Main*: aus aufgerufen wird.

Der Name der individuellen Prozedur wird automatisch aus dem Buchstaben "R" und der in einen String umgewandelten Raumnummer zusammengestellt und der Stringvariablen *jump\$* zugewiesen. Schließlich ruft man die Prozedur noch mit dem @-Operator auf:

```
jump$= "R" + NUM$(raum%,3)
@ (jump$) :
```

Sehen wir uns am Beispiel des Raums 4 an, welchen Beitrag eine solche Raum-Prozedur leistet:

In diesem Raum steht ein Schreibtisch, der gezeichnet werden muss. Das lässt sich mit einfachen grafischen Anweisungen erreichen. Neben den festen Objekten, die der Spieler nicht nehmen kann, befinden sich auch einige bewegliche Gegenstände in diesem Raum. Dazu gehört der Brieföffner. Solange er noch nicht weggenommen wurde, muss er gezeichnet werden. Eine einfache IF-Anweisung entscheidet darüber:

```
IF imRaum$(briefoeffner%) ...
```

Den Abschluss der Raum-Prozedur bildet eine Liste, die noch einmal die "festgenagelten" Objekte aufzählt (Array *fixo%()*). Eine zugehörige Variable sagt, wie viele solcher Objekte es hier gibt (*fixomax%*). Diese Konstruktion dient später der Prozedur *Umsehen*:, denn anstatt die gesamte Objektliste durchsuchen zu müssen, werden nur die festen Objekte gelistet, die im Array *fixo%()* stehen. Ich komme bei der Prozedur *Umsehen*: darauf zurück.

Die Raum-Prozedur ist damit beendet und das Programm setzt seinen Lauf in *Main*: fort.

Der Rest von *Main*: besteht aus der Nachfrage, ob das Spiel schon beendet ist (*gewonnen%= KTrue%*). Nur in diesem Falle wird das Finale aufgerufen. Ansonsten beginnt die große unendliche Schleife ihre Arbeit von vorn: das Programm wartet also darauf, dass der Spieler neue Befehle eingibt ...

Sie wissen nun, wie man sich durch die Räume fortbewegt, solange die Türen offen stehen. Wir kommen nun zu den Kommandos, mit denen der Spielverlauf bestritten wird.

Details erforschen

Die Bildschirme, insbesondere von Revo und S5xx sind recht klein, so dass viele Objekte grafisch nicht so detailliert dargestellt werden können. Die Information über

Einzelheiten bezieht man deshalb über extra Kommandos, die man mit den Kommando-Buttons in der linken Bildschirmfläche auslöst. "Umsehen" liefert dabei die Informationen über den Raum und zählt die hier sichtbaren Gegenstände auf. Wer sich über die einzelnen Gegenstände informieren will, wählt das Kommando "sieh an". Mit dem "Inventar"-Button, der Bestandteil des Toolbars ist, erfährt man, welche Gegenstände man bereits eingeheimst hat. Gehen wir in die Details.

Überblick durch "umsehen"

Die Funktion "umsehen" ist klar umrissen, zusätzliche Angaben braucht das Programm nicht. "Umsehen" ist die Aufforderung an das Programm, zu erzählen, was es im Raum zu sehen gibt. Die Prozedur arbeitet recht einfach. Sie wird über diese Zwischenstationen aufgerufen:

Main: > *Handle_penevent:()* > *Pen_event:(window&=1)* > *Umsehen:*

Umsehen: selber untergliedert sich in drei Teile:

- Vorbereitung der Raumbeschreibung (*Teile_Satz:()*)
- Zusammenstellung der im Raum sichtbaren Gegenstände (*zeige_Objekte:*)
- Ausgabe des Textes (*Textausgabe:*)

Dem ganzen liegt ein eigenes Textausgabekonzept zugrunde. Der Aufwand besteht darin, vorhandenen Text korrekt in einzelne Textzeilen zu zergliedern, um sie dann mit dem Dialog-Befehl dTEXT darstellen zu können.

Die Textaufgliederung wird in der Prozedur *Teile_Satz:(s\$)* vorgenommen, wobei der Text selber über den String-Parameter bereitgestellt wird. Damit ergibt sich automatisch eine Begrenzung der Einzelsatzlänge auf 255 Zeichen. In der Praxis reicht das allemal. Der String wird mit einem hier noch öfter genutzten Verfahren "besorgt":

Über den @-Operator wird eine Prozedur aufgerufen, die mit Hilfe der VECTOR-Anweisung einen ganz bestimmten Text zurückliefert. Welcher Text das ist, bestimmt der mitgegebene Parameter - in diesem Falle ist es die Raumnummer *raum%*.

Berücksichtigt man die Bildschirmgröße von Revo und S5xx, stehen maximal 7 Zeilen für die Textausgabe zur Verfügung. Jede Zeile sollte nicht mehr als 50 Zeichen enthalten. Zur Orientierung: in jede passen höchstens 40 große "W".

Teile_Satz:() nimmt nun den Text, entfernt ein eventuell endständiges Komma und fügt gewissermaßen als Ende-Marke ein Leerzeichen an. Die Einzelheiten entnehmen Sie bitte den Kommentaren im Quelltext.

Ergebnis des Verfahrens: die sieben ehemals leeren Textzeilen, gebildet durch das Text-Array *textzeile\$()*, werden mit dem sauber getrennten Text belegt. Dabei bleiben mit Sicherheit ein paar Zeilen frei, die dann für die Zusammenstellung der Objekte im Raum genutzt werden können.

Die Vorarbeit dazu leistet die Prozedur *Zeige_Objekte:*. Sie stellt einen String zusammen, der zuerst die feststehenden sichtbaren Objekte aufführt und dann die beweglichen sichtbaren Objekte ergänzt.

Achtung bei der Bewertung der Variablen *sichtbar%()*. Hat sie den Wert *KTrue%*, kann es sich auch um einen bereits genommenen Gegenstand handeln. Hier sollen aber nur die Gegenstände, die sich im Raum befinden, gelistet werden. Deshalb wird zusätzlich geprüft, ob sich der Gegenstand im Raum befindet (Prozedur *Anwesenheit%()*). Der zusammengestellte String wird mit *Teile_Satz()*: auf freie Zeilen verteilt.

Die Textdarstellung ist nun komplett vorbereitet, die Ausgabe erfolgt mit der Prozedur *Textausgabe:*. Diese sorgt auch dafür, dass das Text-Array *textzeile\$()* nach der Anzeige wieder "geputzt", also geleert wird.

Gegenstände ansehen – "sieh an"

Will man Objekte beschreiben, muss man zuerst sagen, um welches Objekt es sich handelt. Programmtechnisch besteht der Aufwand darin, die möglichen Objekte für den Spieler in einer Liste anzubieten und die Auswahl zu ermöglichen.

Die Prozedur *Ansehen:* spielt dabei lediglich die Rolle eines Arbeitsverteilers. Erstes aufgerufenes Unterprogramm ist *Objektauswahl%()*, das die Nummer des Objektes zuliefert, das angesehen werden soll.

Um die Übersicht zu wahren, wurde *Objektauswahl%()* in etliche weitere Unterprogramme aufgesplittet, die folgende Funktionen besitzen:

- *Objekte_erzeugen:* stellt die Liste aller Gegenstände im Raum und die des Inventars in einer eigenen Liste mit Nummer und Namen zusammen (*objListe%()* und *objListe\$()*).
- *Objektliste_xxx_anzeigen:()* zeigt die vorbereitete Liste in Abhängigkeit vom Gerätetyp auf dem Bildschirm dar.
- *Objektliste_abfragen%:* wartet in einer eigenen GETEVENT32-Schleife auf die Auswahl durch den Spieler. Dabei werden nur die Fenster, in denen die Objekte namentlich abgebildet sind und die Esc-Taste berücksichtigt. Der Auswahlprozess liefert auf jeden Fall eine Zahl zurück, deren Wert kann allerdings auch Null sein.
- *Objektliste_Anz_schliessen:* löscht die Darstellung der Objektliste.

Das Ergebnis von *Objektauswahl%():* ist die Nummer des ausgewählten Gegenstandes.

Anhand der Entscheidungskriterien "benutzt", "genommen" oder "weder benutzt noch genommen" wird die Beschreibung des Objektes aus den Prozeduren *Eigenschaft1():*, *Eigenschaft2():* oder *Eigenschaft3():* bezogen. Mit Hilfe der Satzzergliederung *Teile_Satz():* wird die Textdarstellung vorbereitet und mit *Textausgabe:* vollzogen.

Die einzelnen Prozeduren sind umfangreich kommentiert. Sollten jetzt hier noch Fragen verblieben sein, finden Sie dort die Antworten.

Inventar ansehen

Das Listen des Inventars ist vergleichsweise so einfach zu programmieren wie "umsehen". Es müssen lediglich alle nehmbaren Objekte mit der Eigenschaft *genommen%()= KTrue%* zusammengestellt und angezeigt werden.

Die Prozedur *Inventar:*, die das organisiert, stellt einen entsprechenden String zusammen und bildet ihn mit Hilfe von *Teile_Satz():* und *Textausgabe:* auf dem Bildschirm ab.

Inventar: kann auf drei Wegen aktiviert werden:

- über das Menü
- den Toolbar und
- per Tastatur-Shortcut.

"Action"

Öffnen von Objekten

Für das Kommando "sieh an" haben wir bereits herausgefunden, wie man den Gegenstand, den das Kommando betrifft, dem Spieler durch eine Auswahl anbietet und ermittelt. Diese Verfahrensweise benötigen wir auch für das Kommando "öffne".

Die Prozedur *Oeffnen:* ruft deshalb zuerst auch wieder *Objektauswahl%():* auf, bevor es die Arbeit endgültig verteilt. Welcher Zweig der IF-Entscheidung dabei durchlaufen wird, hängt vom aktuellen Zustand des Objektes ab. Nehmen wir das Beispiel einer Tür:

1. Steht sie sperrangelweit auf, ist ihr Zustand "offen" (*oeffnungsStatus%()= 4*). Auf das Kommando "öffne" müssen wir also dem Spieler mit der Antwort "Ist schon

offen" informieren. Das ist kurz und schmerzlos in einer Zeile unterzubringen – es muss also nicht erst ein Unterprogramm bemüht werden.

2. Erfüllt die Tür ihre natürliche Funktion und ist deshalb im Zustand "geschlossen" (*oeffnungsStatus%()* = 1), bewirkt das Kommando "öffne", dass sie anstandslos aufschwingt. Zuständig ist das Unterprogramm *O_ist_zu:()*.
3. Die Tür kann nicht nur "geschlossen" sein, sondern darüber hinaus den Zustand "verschlossen" (*oeffnungsStatus%()* = 2) annehmen. In diesem Falle funktioniert "öffne" nur, wenn man bereits den richtigen Schlüssel besitzt. Aus Gründen der Übersicht wird die Behandlung dieses Falles an das Unterprogramm *O_ist_verschlossen:()* übertragen.

Die Unterprogramme weisen noch ein paar Besonderheiten auf:

- *O_ist_zu:()*: Manche bewegliche Objekte lassen sich erst öffnen, wenn man sie genommen hat. Das ist z.B. der Fall, wenn man den Schlüssel haben will, der sich in dem Kästchen versteckt, das im Wandschrank gefunden wird. Sollte das Objekt (hier das Kästchen) noch einen anderen Gegenstand verbergen, muss der natürlich jetzt sichtbar werden. Die Prozedur bewerkstelligt das an dieser Stelle mit Hilfe der Variablen *gast%*. Wenn der Wirt (Kästchen) einen Gast (Schlüssel) beherbergt, wird der Gast sichtbar geschaltet: *sichtbar%(gast%(obj%)) = KTrue%*. Da sich durch "öffne" optische Veränderung am Bildschirm ergeben könnten, setzt man zum Schluss noch: *geaendert% = KTrue%*, die wichtige Variable *oeffnungsstatus%()* wurde bereits beim Eintritt in den Else-Zweig auf den Zustand "geöffnet" geändert.
- *O_ist_verschlossen:()*: Zu jedem verschlossenen Objekt gehört ein ganz individueller Schlüssel. Jedes Öffnen ist also ebenfalls ein ganz individueller Vorgang. Um den Überblick zu behalten, hat es sich deshalb bewährt, diese Prozedur ebenfalls nur als Aufgabenverwalter arbeiten zu lassen, der die Einzelfälle an eigenständige Unterprogramme verteilt. In unserem Beispielprogramm sind das die Prozeduren *O_tuer5:* und *O_koffer:*. Nur, wenn der richtige "Schlüssel" vorhanden ist, wird der Öffnungsvorgang durchgeführt und alle erforderlichen Variablen gesetzt. Am Ende steht wieder das *geaendert% = KTrue%*, damit optische Veränderungen wirksam werden können. Im Beispiel mit dem Koffer wurde dazu die Raum-Prozedur *PROC R6:* entsprechend angepasst: Wenn nun der Koffer geöffnet wird, wird wegen des geänderten Wertes von *oeffnungsStatus%()* automatisch auch optisch der Deckel aufgeklappt!

Objekte nehmen

"Nehmen" hat programmiertechnisch viele Ähnlichkeiten mit "öffnen". Der wesentliche Unterschied besteht darin, dass als Folge des "Nehmens" andere Variablen als bei "öffnen" verändert werden.

Die Hauptroutine *PROC Nehmen*: ist wieder als Arbeitsverteiler aufgebaut. Nachdem feststeht, welches Objekt genommen werden soll (*objektauswahl%()*), werden hier drei Fälle unterschieden:

1. Das Objekt wurde bereits genommen (*genommen%() = KTrue%*). In diesem Falle wird nur die Mitteilung "Hast du schon!" vorbereitet.
2. Das Objekt ist an diesem Ort noch da und kann genommen werden, weil *nehmbar%() = KTrue%* ist. Das führt dann in das Unterprogramm *O_ist_nehmbar:()*.
3. Das Objekt kann nicht mitgenommen werden, weil es durch *nehmbar%() = KFalse%* fest verankert ist. Das trifft z.B. auf den Schreibtisch zu – wer möchte den schon gern mit sich herumschleppen, selbst wenn man ihn "nehmen" könnte ... Für solche Fälle reicht es, eine einfache Zufallsantwort über *Quatsch_nimm\$*: zusammenzustellen, denn Variablen müssen nicht geändert werden.

Das Unterprogramm *O_ist_nehmbar:()*, mit dem tatsächlich "genommen" wird, unterscheidet nun noch einmal zwischen Objekten, die einer Spezialbehandlung bedürfen, und solchen, die sich universell abhandeln lassen.

Die Spezialbehandlungen sind mit zwei Muster-Objekten nur angedeutet, um das Prinzip zu zeigen. Die Objekte wurden aus diesem Grunde auch nur lokal deklariert, um Fehler beim Übersetzen zu vermeiden. Wenn Sie selber solche Objekte einfügen wollen, gliedern Sie diese von Anfang an richtig in die globale Deklaration und in der richtigen Reihenfolge ein ("nehmbare" Objekte an den Anfang, s.o.).

In vielen Fällen kommen Sie mit dem normalen "nehmen" aus. Das Programm benutzt in dem Falle eine weitere Unteroutine: *N_allgemein:()*. Hier werden nun die entscheidenden Variablen geändert: Das Objekt ist ab sofort nicht mehr nehmbar (*nehmbar%() = KFalse%*), dafür aber genommen (*genommen%() = KTrue%*). Durch das "Nehmen" verschwindet es auch aus dem Raum (*imRaum%() = KFalse%*). Denken wir an das Beispiel Schlüssel / Kästchen, müssen wir hier noch die gegenseitigen Bindungen lösen: Der Wirt verliert den Gast und der Gast verabschiedet sich vom Wirt. In Variablen ausgedrückt, heißt das: *gast%(wirt%(obj%)) = KFalse%* und *wirt%(obj%) = KFalse%*.

Wie man Gegenstände verwendet

Auch hier ist die zentrale Unterprozedur wieder der Arbeitsverteiler – sie heißt *PROC Benutzen*:

Wie schon zuvor in den anderen Kommando-Prozeduren wird zuerst das Objekt benannt, das man "benutzen" möchte. Meist ist das "Benutzen" für jedes Objekt an ganz bestimmte individuelle Umstände geknüpft. Was beim "Nehmen" noch als Option angedeutet wurde, ist hier der Regelfall: Das "Benutzen" erfordert für jedes

Objekt eine Sonderbehandlung, die in einer jeweils eigenen Unterroutine vorgenommen wird.

Wie ausgeprägt spezifisch diese Routinen sind, zeigen Ihnen *PROC B_schlüssel*., *PROC B_briefoeffner*. und *PROC B_kuli*:. Allen gemein jedoch ist, dass die Variable *benutzt%()* auf *KTrue%* gesetzt wird.

Wenn man in einem Spiel sehr viele Gegenstände verwendet, ist es günstig, sie nach erfolgreicher Benutzung sang- und klanglos wieder verschwinden zu lassen. Setzen Sie dazu einfach *sichtbar%()* und *genommen%()* auf *KFalse%*. Das würde eigentlich schon reichen, da die Gegenstände danach in keiner Auswahl- oder Anzeigeliste mehr auftauchen können. Der Vollständigkeit halber setzen Sie aber auch noch *benutzt%()* auf *KTrue%* - gelegentlich zahlt sich solche Erbsenzählerei in der Praxis beim Testen doch aus ...

Gegenstände verschieben

Die "schwierigen" Kommandos haben Sie bereits kennengelernt. Die Interpretation der *PROC Bewegen*. wird Ihnen daher nicht mehr schwer fallen.

Das Ziel von "bewegen" ist es zumeist, versteckte Objekte ans Tageslicht zu befördern oder verdeckte Türen etc. freizugeben. Deshalb müssen die Variablen der verdeckten Objekte immer mit verändert werden.

In unserem Beispiel wird durch "bewegen" des Bildes der Koffer sichtbar. Man beachte: Beide Gegenstände sind nicht "nehmbar", deshalb kommen hier auch die Variablen *wirt%()* und *gast%()* nicht zum Einsatz.

Höflich, aber sinnlos

Um den Spieler ein wenig zu verwirren, erfindet man Funktionen, die nie ernsthaft zum Zuge kommen. So ist es in unserem Beispiel mit dem "Anklopfen". Die *PROC Anklopfen*. stellt lediglich eine Antwort zu Verfügung, bewirkt aber das ganze Spiel hindurch absolut nichts.

Eine solche Funktion ist natürlich nicht unbedingt Pflicht, bei der richtigen Wahl der Antworten entspannt es aber den Spieler vielleicht ein wenig, der gerade mit der Lösung nicht so recht voran kommt ...

Vielleicht haben Sie auch gar keinen Platz mehr, um ihn an einen solchen Spaß-Button zu verschwenden. Und das ist mit das Schöne für den Programmierer: Er darf das alles selbst entscheiden und seiner Fantasie freien Lauf lassen.

Spielstände sichern und wiederherstellen

In Fall von HH2000 reichen zwei einfache Routinen, die den Spielstand in eine einfache Datenbank sichern und wieder aus ihr ablesen. Die Datenbank benötigt nur ein einziges Feld, denn es müssen ausschließlich Integer-Variablen abgelegt oder geladen werden. Es handelt sich dabei um all diese Variablen, mit denen im Spiel die Objektzustände beschrieben werden.

Spiel speichern

Das Ablegen der Daten in die Datenbank erledigt das Unterprogramm *Spiel_speichern*: Eingeleitet wird der Vorgang durch die Abfrage des Dateinamens, denn jeder Spielstand kann in einer eigenen Datenbank abgelegt werden. Trägt der Spieler in den Dialog einen bereits bestehenden Namen ein, wird die Datei ohne Rückfrage gelöscht und neu angelegt. Wem das nicht gefällt, der kann an dieser Stelle einen Abbruch mit nachfolgender Neuabfrage einfügen.

Der Parameter *f%* sorgt beim dFILE-Befehl dafür, dass nach Abschluss der Eingabe automatisch die Dateinamen-Endung "h2k" vergeben wird.

```
f%= 1 + 128
dINIT
  dFILE file$, "", f%
  ...
DIALOG
```

Das klappt allerdings nur, wenn die Vorgaben von Pfad und Laufwerk nicht verändert werden!

Das eigentliche Abspeichern ist völlig unkompliziert, Sie müssen nur wissen, welche Variablen abgelegt werden sollen.

Spiel laden

Einen Spielstand laden ist einfach nur die Umkehrung des Speicherprozesses, die zugehörige Routine heißt *Spiel_laden*:. Das Laden wird durch die Abfrage des Datenbank-Namens eingeleitet. Der Spieler hat es recht einfach: er muss nur aus einer Liste auswählen. Wieder bestimmt *f%*, wie der zugehörige Dialog funktioniert:

```
f%= 16
dINIT
    dFILE file$, "", f%
    ...
DIALOG
```

Die Liste zeigt alle Dateien im Verzeichnis an und filtert nicht nach der Datei-Endung aus. Das ist nicht weiter schlimm, weil ja das vorgegebene Verzeichnis für die Spielstandsdateien reserviert ist. Wechselt man aber das Verzeichnis, stehen auch Dateien mit anderen Endungen (oder auch ohne) zur Auswahl zur Verfügung. Solche Dateien weist die Lade-Routine kurzerhand ab, weil sie die Endung überprüft.

Die eigentliche Laderoutine wurde aus Übersichtsgründen in die Prozedur *Laden*:(*)* ausgelagert und ist unspektakulär. Alles, worauf Sie achten müssen, ist: Die Reihenfolge des Ladens muss mit der Reihenfolge beim Abspeichern übereinstimmen. Aber das ist ja wohl klar.

Hilfe!

Über die Prozedur *Hilfe*: ruft der Spieler eine externe Hilfe-Datenbank auf. Eine solche Datenbank erstellen Sie mit der Psion-Applikation DATEN. Dort schreiben Sie nach Herzenslust hinein, was der Spieler wissen soll. Es macht sich gut, wenn Sie nicht nur im Spiel selbst, sondern auch hier Ihre Kontaktangaben hinterlegen.

Wenn die Hilfe-Datenbank fertig ist, brauchen Sie diese nur noch so umzubenennen, dass sie die Endung "hlp" bekommt, also z.B. "hilfe.hlp". Damit wird ihr Inhalt automatisch vor Zugriffen des Benutzers geschützt, weil die aktiven Funktionen der geöffneten Datenbank in diesem Zustand nicht zugänglich sind.

Zum Öffnen der Hilfedatei aus dem Spiel heraus werden zwei Funktionen aus der Datei "SYSTEM.OPX" verwendet, die uns durch "includieren" der zugehörigen Header-Datei "SYSTEM.OXH" zugänglich ist.

Durch den Aufbau der Prozedur *Hilfe*: werden drei Zustände der Hilfe-Datei unterschieden und entsprechende Reaktionen ausgelöst:

1. Die Hilfedatei existiert nicht: Es wird eine Meldung am Bildschirm ausgegeben, das Programm läuft aber ungestört weiter.
2. Die Hilfedatei ist bereits geöffnet und liegt im "Hintergrund", also für den Anwender verdeckt: Die Funktion

```
SetForegroundByThread&: (thread&, 0)
```

holt die verdeckte Datei in den Vordergrund, macht sie also dem Benutzer einsehbar. Mit der "Zurück"-Funktion im Toolbar der Hilfedatei schiebt man diese wieder in den Hintergrund, ohne sie zu schließen.

3. Die Hilfedatei ist nicht geöffnet: Die Funktion

`SetForegroundByThread&: ()` kann nicht aktiv werden, weil ja keine Datei geöffnet ist. Der daraus resultierende Fehler führt dann in der letzten Zeile der Prozedur *Hilfe*: zum Öffnen der Hilfedatei:

```
thread&= RunApp&: ("Data", file$, "", 0)
```

Weitere Informationen zur Anwendung von Funktionen der OPX-Dateien findet man in den Dokumentationen zum OPL-SDK, das man von Symbian (www.symbian.com) kostenlos downloaden kann.

Zu guter Letzt

Zu einem gewonnenen Spiel gehört als "Belohnung" ein wie auch immer geartetes Finale. Die Prozedur *Finale*: ist der eigentliche Einstiegspunkt. Als Demo habe ich ein Beispiel hinzugefügt, das zeigt, wie man eine zerbröselnde Schrift machen kann (*Demo_finale*:). Sie müssen nur das "REM" im Quelltext entfernen, dann funktioniert.

Wer den Aufwand nicht scheut, kann hier auch kleine grafische Animationen anhängen, die sich mit Sprites oder auch anders erzeugen lassen. Sie können an dieser Stelle Ihrer Fantasie freien Lauf lassen und verwenden, was in Ihren Fähigkeiten steckt.

Damit bin ich mit meinen Ausführungen fertig. Ihr Part beginnt aber wohl erst jetzt so richtig ... Ich wünsche Ihnen viel Spaß und Freude beim Experimentieren!

Anhang 1

Quelltext: Rahmen.OPL

REM Programm Rahmen.opl

REM Achtung! Besonders lange Zeile mussten für den Druck
REM umgebrochen werden. Die Einrückungen sind so deutlich, dass
REM diese Tatsache nicht noch einmal extra markiert wurde.

APP Rahmen,&00004812

REM einfachste Applikation, Icons fehlen noch
ENDA

INCLUDE "CONST.OPH"

REM benutzt werden folgende Konstanten aus CONST.OPH:

REM CONST KEvNotKeyMask& = &400
REM CONST KEvCommand& = \$404
REM CONST KEvPtr& = &408
REM CONST KFontArialBold11& = 268435952
REM CONST KKmodShift% = 2 Shift-Taste gedrueckt
REM CONST KKmodControl% = 4 Strg-Taste gedrueckt

PROC Begin:

GLOBAL bildweite% REM Bildweite
GLOBAL bildhoehe% REM Bildhoehe

bildweite%= gWIDTH
bildhoehe%= gHEIGHT
LOADM "Z:\System\Opl\Toolbar.opo"
TBarLink:("Main")
ENDP

PROC Main:

REM kommt von TBarLink:() / Toolbar.opo
GLOBAL programm\$(20) REM Programmname
GLOBAL version\$(20) REM Programm-Versionsnummer
GLOBAL a&(16) REM Feld f.Ablage d.Event-Variablen
GLOBAL minit% REM Merker für letzte Menueposition
GLOBAL toolbar% REM Merker fuer Toolbar an/aus
GLOBAL psionTyp\$(7) REM Handheld-Geraetetyp
GLOBAL basicWin%, extraWin%
LOCAL jump\$(6)

basicWin%= gIDENTITY
programm\$= "HH2000"
version\$ = "v3.2"

Init:

WHILE 1

ONERR Fehler::

CLS : PRINT "<Strg><E> beendet das Programm!"

WHILE 1

GETEVENT32 a&()

REM nicht ausgewertet werden diese Events:

REM &401 Applikation in den Vordergrund

REM &402 Applikation in den Hintergrund

REM &403 Psion wurde eingeschaltet

```

IF a&(1)= KEvCommand&
    REM nur "X"= Programm schließen wird beruecksich-
    REM tigt, weil Programm keine Dokumente verwendet.
    IF LEFT$(GETCMD$,1)="X" : Exit: : ENDIF

ELSEIF (a&(1) AND KEvNotKeyMask&)=0
    REM Es ist ein Tastatur-Event.
    REM a&() enthaelt an wesentlichen Infos jetzt:
    REM a&(1) Tasten-Code
    REM a&(4) Benutzte Modifizierer-Taste(n):
    REM          Shift=2, Strg=4, Caps=16, Fn=32
    Handle_keyevent:(a&(1), a&(4))

ELSEIF (a&(1)=KEvPtr& OR (a&(1)<=10004 AND
                                a&(1)>=10000))
    REM Es ist ein Pen-Event
    REM a&() enthaelt an wesentlicheninfos jetzt:
    REM a&(1) Event-Art (v. Bildsch.o.Funktionsleiste)
    REM a&(3) Fenster-Id, in dem der Stift agiert
    REM a&(4) Pen:abgesetzt=0, abgehoben=1,
    REM          auf Flaeche=6
    REM a&(6) x-Koordinate im Fenster
    REM a&(7) y-Koordinate im Fenster
    Handle_penevent:(a&(1),a&(3),a&(4),a&(6),a&(7))

ENDIF

REM Hier folgen die Spiel-eigenen Routinen

ENDWH
Fehler::
ONERR OFF
Fehlerbehandlung:(programm$,version$)
ENDWH
ENDP

PROC Handle_keyevent:(key&, modifier&)
    REM Verteilung des Tastatur_Events auf Subroutinen
    IF modifier& AND KKmodControl% REM Strg gedrueckt
        IF modifier& AND KKmodShift% REM Shift gedrueckt
            Handle_chars:(key&+64)
        ELSE
            Handle_chars:(key&+96)
        ENDIF
    ELSE
        IF key&>4000
            Handle_hiKeyValues:(key&)
        ELSEIF key& < 58 AND key& > 47
            giPRINT CHR$(key&)+" : Ziffern nicht unterstuetzt!",0
        ELSEIF key&<=90 AND key&>=65
            giPRINT CHR$(key&)+" :Grossbuchst. nicht unterstuetzt",0
        ELSEIF key&<=122 AND key&>=97
            giPRINT CHR$(key&)+" :Kleinbuchst. nicht unterstuetzt",0
        ENDIF
    ENDIF

```

```

        ENDIF
    ENDP

PROC Handle_hiKeyValues:(code&)
    REM Tastaturwerte > 4000 abarbeiten
    REM (Cursortasten und Menue)
    IF code&=4150
        Zeige_Menu:
    ELSEIF code&=4106
        giPRINT "Cursor nach unten, nicht unterstuetzt",0
    ELSEIF code&=4105
        giPRINT "Cursor nach oben, nicht unterstuetzt",0
    ELSEIF code&=4104
        giPRINT "Cursor nach rechts, nicht unterstuetzt",0
    ELSEIF code&=4103
        giPRINT "Cursor nach links, nicht unterstuetzt",0
    ELSEIF code&=4098
        giPRINT "Pos 1, nicht unterstuetzt",0
    ELSEIF code&=4099
        giPRINT "Ende, nicht unterstuetzt",0
    ELSEIF code&=4100
        giPRINT "Bild auf, nicht unterstuetzt",0
    ELSEIF code&=4101
        giPRINT "Bild ab, nicht unterstuetzt",0
    ENDIF
ENDP

PROC Handle_chars:(char&)
    REM Selektive Verarbeitung d. Tastatur-Eingabe
    IF char&= ASC("a") : Programm_Info:
    ELSEIF char&= ASC("c") : Clear_win1:
    ELSEIF char&= ASC("d") : Clear_win2:
    ELSEIF char&= ASC("e") : Exit:
    ELSEIF char&= ASC("l") : Spiel_laden:
    ELSEIF char&= ASC("n") : Kein_Bilderrahmen:
    ELSEIF char&= ASC("r") : Bilderrahmen:
    ELSEIF char&= ASC("s") : Spiel_speichern:
    ELSEIF char&= ASC("t") : Toolbar:
    ELSEIF char&= ASC("H") : Hilfe:
    ENDIF
ENDP

PROC Pen_event:(window&,x&,y&)
    AT 2,3 : PRINT " "
    AT 2,3 : PRINT "Win:",window&
    gUSE window&
    gSETPENWIDTH 5
    gAT x&,y&
    gLINEBY 0,0
    gSETPENWIDTH 1
ENDP

```

```

PROC Handle_penevent: (key&, a3&, a4&, a6&, a7&)
    REM Behandelt alle Pen-Events
    IF key&=KEvPtr&
        IF TBarOffer%: (a3&, a4&, a6&, a7&)
            RETURN
        ELSE
            REM Spaeter wird nur noch Pen up beruecksichtigt!:
            REM ELSEIF a4&=1

                Pen_event: (a3&, a6&, a7&)
            ENDIF
        ELSEIF key&=10000      REM Menue
            Zeige_Menu:
        ELSEIF key&=10001
            giPRINT "Cut/Copy/Paste, nicht unterstUtzt",0
        ELSEIF key&=10002
            giPRINT "InfraRot, nicht unterstUtzt",0
        ELSEIF key&=10003
            giPRINT "Zoom in, nicht unterstUtzt",0
        ELSEIF key&=10004
            giPRINT "Zoom out, nicht unterstUtzt",0
        ELSE
            ENDIF
    ENDP

PROC Zeige_Menu:
    LOCAL key&

    mINIT
    mCARD "Datei", "Speichern", %s, "Laden", -%l, "Beenden", %e
    mCARD "Ansicht", "CLS Win1", %c, "CLS Win2", %d, "Toolbar ein",
        %t OR $2800*(-toolbar%)
    mCARD "Optionen", "Rahmen ein", %r, "Rahmen aus", %n
    mCARD "Extras", "Hilfe", -%H, "Über HH2000", %a
    key&=MENU(minit%)

    Handle_chars: (key&)
ENDP

PROC Cmda%:
    Programm_Info:
ENDP

PROC Cmdc%:
    Clear_win1:
ENDP

PROC Cmdd%:
    Clear_win2:
ENDP

```

```

PROC Cmdb%:
    Exit:
ENDP

PROC Cmdr%:
    Bilderrahmen:
ENDP

PROC CmdSH%:
    Hilfe:
ENDP

PROC Clear_win1:
    CLS
    PRINT "<Strg><E> beendet das Programm!"
ENDP

PROC Clear_win2:
    gUSE extraWin%
    gCLS
    gBORDER 3
ENDP

PROC Spiel_speichern:
    giPRINT "Speichern noch nicht unterstützt!",0
ENDP

PROC Spiel_laden:
    giPRINT "Laden noch nicht unterstützt!",0
ENDP

PROC Bilderrahmen:
    gUSE basicWin%
    gBORDER 3
ENDP

PROC Kein_Bilderrahmen:
    gUSE basicWin%
    gGMode 2    REM invertieren
    gBORDER 3
    gGMode 0    REM wieder normal zeichnen
ENDP

PROC Programm_Info:
    dINIT "Über "+programm$+" "+version$
    dTEXT "","Das ist ein Programm von ...", 2
    dTEXT "","Hacker Arno",2
    dTEXT "","Email: arno@gmx.net",2
    dTEXT "","Web : http://www.thefreespace.com/arno/",2
    dBUTTONS "Weiter",13
    DIALOG
ENDP

PROC Hilfe:

```



```

        giPRINT "HILFE noch nicht unterstützt!",0
ENDP

```

```

PROC Exit:
    REM hier koennten noch irgendwelche Dinge vor der
    REM Beendigung des Programmes erledigt werden
    STOP
ENDP

```

```

PROC Fehlerbehandlung: (p$,v$)
    DINIT "Fehlermeldung" REM ,2
    dTEXT "","Sorry, das sollte nicht mehr passieren!"
    dTEXT "","Bitte eine eMail an: autor@hotmail.com"
    dTEXT "","mit möglichst genauer Fehlerbeschreibung."
    dTEXT "","Dazu hätte ich gern auch noch die nachstehende"
    dTEXT "","Information: -danke für die Hilfe!-"
    dTEXT "","",$800
    dTEXT "","p$+v$+","+" Es gab einen "
    dTEXT "","ERRX$+","
    dTEXT "","und zwar : "+ERR$(ERR)
    dBUTTONS "Weiter",13 OR $300
    DIALOG
ENDP

```

```

PROC Init:
    IF bildweite%= 480 AND bildhoehe%= 160
        psionTyp$= "revo"
    ELSEIF bildweite%= 640
        IF bildhoehe%= 240
            psionTyp$= "s5"
        ELSEIF bildhoehe%= 480
            psionTyp$= "netbook"
        ENDIF
    ENDIF

    toolbar%= KTrue%
    Setup_toolbar:
    REM Demo-Fenster
    extraWin%= gCREATE (200,50,100,100,1,1)
    gBORDER 3
ENDP

```

```

PROC Toolbar:
    REM Toolbar an/aus im Wechsel
    IF toolbar%= KTrue%
        Hide_Toolbar:
        toolbar%= KFalse%
    ELSE
        Show_Toolbar:
        toolbar%= KTrue%
    ENDIF
ENDP

```

```

PROC Show_Toolbar:
    TBarShow:
    IF psionTyp$= "s5" OR psionTyp$= "netbook"
        bildweite%=640-72
    ELSEIF psionTyp$="revo"
        bildweite%=480-54
    ENDIF
ENDP

PROC Hide_Toolbar:
    TBarHide:
    IF psionTyp$= "s5" OR psionTyp$= "netbook"
        bildweite%=640
    ELSEIF psionTyp$="revo"
        bildweite%=480
    ENDIF
ENDP

PROC Setup_Toolbar:
    LOCAL bild$(128),bitmap1&,mask1&
    bild$="Z:\System\Apps\Data\Data.mbm"
    bitmap1&=gLoadBit(bild$,0,3)
    mask1&=gLoadBit(bild$,0,4)

    TBarInit:("Demo",bildweite%,bildhoehe%)
    IF psionTyp$= "revo"
        TBarButt:("a",1,"Info",0,&0,&0,0)
        TBarButt:("c",2,"Clear"+CHR$(10)+"Win 1",0,&0,&0,0)
        TBarButt:("d",3,"Clear"+CHR$(10)+"Win 2",0,&0,&0,0)
    ELSE
        REM es ist ein S5,oder ...
        TBarButt:("a",1,"Info",0,bitmap1&,mask1&,0)
        TBarButt:("c",2,"Clear"+CHR$(10)+"Win 1",0,&0,&0,0)
        TBarButt:("d",3,"Clear"+CHR$(10)+"Win 2",0,&0,&0,0)
        TBarButt:("b",4,"Ende",0,&0,&0,0)
        REM wegen Emulator auskommentiert:
        REM IF psionTyp$="netbook" REM ... ein netBook/S7
        REM REM Buttons ohne Icon
        REM TBarButt:("H",5,"Hilfe",0,&0,&0,0)
        REM TBarButt:("r",6,"Rahmen"+CHR$(10)+"ein",0,&0,&0,0)
        REM ENDIF
    ENDIF

    Show_Toolbar:
ENDP

```

Anhang 2
Quelltext: HH2000.OPL

```

REM Programm HH2000.opl

REM Achtung!
REM Besonders lange Zeile mussten für den Druck umgebrochen
REM werden. Die Einrückungen sind so deutlich, dass diese
REM Tatsache nicht noch einmal extra markiert wurde.

APP hh2000,&00004712
    ICON "icon24.mbm"
    ICON "mask24.mbm"
    ICON "icon32.mbm"
    ICON "mask32.mbm"
    ICON "icon48.mbm"
    ICON "mask48.mbm"
    REM FLAGS 1 oder 2, nicht setzen
ENDAPP

INCLUDE "CONST.OPH"
INCLUDE "SYSTEM.OXH"

REM benutzt werden folgende Konstanten aus CONST.OPH:
REM CONST KEvNotKeyMask&          = &400
REM CONST KEvCommand&            = $404
REM CONST KEvPtr&                 = &408
REM CONST KEvPtrPenUp&            = 1      Stift abgehoben
REM CONST KFontArialBold11&       = 268435952
REM CONST KKmodShift%             = 2      Shift-Taste gedrueckt
REM CONST KKmodControl%           = 4      Strg-Taste gedrueckt
REM KTrue%                        = -1      Boolean WAHR / TRUE
REM KFalsse%                      = 0      Boolean FALSCH / FALSE

PROC Begin:
    GLOBAL bildweite%, bildhoehe%
    bildweite%= gWIDTH
    bildhoehe%= gHEIGHT
    LOADM "Z:\System\Opl\Toolbar.opo"
    TBarLink:("Declare_and_init")
ENDP

```

```

PROC Main:
  LOCAL jump$(6)
  programm$ = "HH2000"
  version$ = "v4.1"
  Init:
  WHILE 1
    ONERR Fehler::
    WHILE 1
      GETEVENT32 a&()
      IF a&(1) = KEvCommand&
        REM nur "X" = Programm schließen wird beruecks.:
        IF LEFT$(GETCMD$,1) = "X" : Exit: : ENDIF
      ELSEIF (a&(1) AND KEvNotKeyMask&) = 0
        Handle_keyevent: (a&(1), a&(4))
      ELSEIF (a&(1) = KEvPtr& OR (a&(1) <= 10004 AND
                                                                    a&(1) >= 10000))
        Handle_penevent: (a&(1), a&(3), a&(4), a&(6), a&(7))
      ENDIF

      IF alterRaum% <> raum% OR geaendert%
        alterRaum% = raum%
        geaendert% = KFalse%
        gORDER blindWin%, 1
        gUPDATE OFF
        ZeichneRaum:
        REM siehe nach ob Sonder- & Grafikinfos vorhanden
        jump$ = "r" + NUM$(raum%, 3)
        @(jump$):
        gUPDATE ON
        gORDER gameWin%, 1
      ENDIF
      IF gewonnen%
        Finale:
        gewonnen% = KFalse%
      ENDIF
    ENDWH
    Fehler::
    ONERR OFF
    Fehlermeldung: (programm$, version$)
  ENDWH
ENDP

```

```

REM Tastatur-Event und Folgeprozeduren
*****

PROC Handle_keyevent:(key&, modifier&)
    REM Verteilung des Tastatur_Events auf Subroutinen
    IF modifier& AND KKmodControl% REM Strg gedrueckt
        IF modifier& AND KKmodShift% REM Shift gedrueckt
            Handle_chars:(key&+64)
        ELSE
            Handle_chars:(key&+96)
        ENDIF
    ELSE
        IF key&>4000
            Handle_hiKeyValues:(key&)
        ENDIF
    ENDIF
ENDP

PROC Handle_hiKeyValues:(code&)
    REM Tastaturwerte > 4000 abarbeiten (Cursortasten & Menue)
    IF code&=4150
        Zeige_Menu:
    ELSEIF code&=4106
        REM Cursor nach unten
        raum% = raum% + Corr%:(2)
    ELSEIF code&=4105
        REM Cursor nach oben
        raum% = raum% + Corr%:(1)
    ELSEIF code&=4104
        REM Cursor nach rechts
        raum% = raum% + Corr%:(8)
    ELSEIF code&=4103
        REM Cursor nach links
        raum% = raum% + Corr%:(4)
    ENDIF
ENDP

PROC Handle_chars:(char&)
    REM Selektive Verarbeitung d. Tastatur-Eingabe
    IF char&= ASC("a") : Programm_Info:
    ELSEIF char&= ASC("b") : Ton_an_aus:
    ELSEIF char&= ASC("d") : Tondauer:
    ELSEIF char&= ASC("e") : Exit:
    ELSEIF char&= ASC("i") : Inventar:
    ELSEIF char&= ASC("r") : Spiel_laden:
    ELSEIF char&= ASC("s") : Spiel_speichern:
    ELSEIF char&= ASC("w") : Warnung:
    ELSEIF char&= ASC("H") : Hilfe:
    ENDIF
ENDP

```

```

REM Pen-Event und Folgeprozeduren
*****

PROC Handle_penevent: (key&, a3&, a4&, a6&, a7&)
    REM Behandelt alle Pen-Events
    IF key&=KEvPtr&
        IF TBarOffer%: (a3&, a4&, a6&, a7&)
            RETURN
        ELSEIF a4&= KEvPtrPenUp& REM Pen up
            Pen_event: (a3&, a6&, a7&)
        ENDIF
    ELSEIF key&=10000      REM Menue
        Zeige_Menu:
    ELSEIF key&=10001
        giPRINT "Cut/Copy/Paste, nicht unterstuetzt",0
    ELSEIF key&=10002
        giPRINT "InfraRot, nicht unterstuetzt",0
    ELSEIF key&=10003
        giPRINT "Zoom in, nicht unterstuetzt",0
    ELSEIF key&=10004
        giPRINT "Zoom out, nicht unterstuetzt",0
    ELSE
        ENDIF
ENDP

PROC Pen_event: (window&, x&, y&)
    IF window&=gameWin%
        Check_GameWin: (x&, y&)
    ELSEIF window&=cmdWin% (1)
        Umsehen:
    ELSEIF window&=cmdWin% (2)
        Ansehen:
    ELSEIF window&=cmdWin% (3)
        Nehmen:
    ELSEIF window&=cmdWin% (4)
        Oeffnen:
    ELSEIF window&=cmdWin% (5)
        Benutzen:
    ELSEIF window&=cmdWin% (6)
        Bewegen:
    ELSEIF window&=cmdWin% (7)
        Anklopfen:
    REM ELSEIF ... hier koennten weitere Fenster geprueft werden
    ENDIF
ENDP

```

```

PROC Check_gameWin: (x&,y&)
  IF x& < 90
    raum% = raum% + Corr%:(4)    REM cursor left
  ELSEIF X& < 180
    IF y&<115
      raum% = raum% + Corr%:(1)  REM cursor up
    ELSEIF y&<160
      raum% = raum% + Corr%:(2)  REM cursor down
    ENDIF
  ELSEIF x& < 280
    raum% = raum% + Corr%:(8)    REM cursor right
  ENDIF
ENDP

PROC Corr%:(dir%)
  REM berechnet den Sprungwert fuer den neu zu betretenden Raum,
  REM wenn das Betreten ueberhaupt erlaubt ist. Uebergeben wird
  REM ein Verrechnungswert fuer die gewuenschte Laufrichtung
  LOCAL horDiff%, verDiff%
  horDiff%=1
  verDiff%=3
  IF (dir% AND wege%(raum%)) = 1
    RETURN verDiff%             REM Nord
  ELSEIF (dir% AND wege%(raum%)) = 2
    RETURN -verDiff%            REM Sued
  ELSEIF (dir% AND wege%(raum%)) = 4
    RETURN -horDiff%            REM West
  ELSEIF (dir% AND wege%(raum%)) = 8
    RETURN horDiff%             REM Ost
  ELSE
    IF ton% : BEEP beepLen%,freq% : ENDIF : RETURN 0
  ENDIF
ENDP

PROC Umsehen:
  REM Raumbeschreibung
  Teile_Satz: (@$("Raumbeschreibung")):(raum%)
  Zeige_Objekte: REM   addiert d. sichtb. Obj. im Raum hinzu
  Textausgabe:
ENDP

```



```

PROC Zeige_Objekte:
    REM fuegt der Raumbeschreibung die sichtbaren Objekte hinzu
    LOCAL a$(255), pos%, lastpos%, n%
    a$="Du siehst:"
    REM gefundene Objekte in einem Gesamtstring unterbringen:
    REM zuerst die fixen Objekte aus dem Raum:
    IF fixomax%
        n%=1
        DO
            IF sichtbar%(fixo%(n%))
                a$=a$+uart$(fixo%(n%))+objekt$(fixo%(n%))+", "
            ENDIF
            n%=n%+1
        UNTIL n%>fixomax%
    ENDIF
    REM zweitens: im Raum befindl. sichtbare & nehmbare Objekte:
    REM die Liste der nehmbaren Objekte wird durchforstet:
    n%=1
    DO
        IF sichtbar%(n%)
            IF Anwesenheit%:(n%)
                a$=a$+uart$(n%)+objekt$(n%)+", "
            ENDIF
        ENDIF
        n%=n%+1
    UNTIL n%>nehmbarMax%

    IF a$= "Du siehst:" REM keine zu zeigenden Objekte da, daher:
        a$= a$ + " sonst nichts weiter."
    ENDIF

    Teile_Satz:(a$)

ENDP

PROC Ansehen:
    LOCAL kopf$(11) REM Titel fuer Kopfzeile
    LOCAL o% REM Objektnummer
    kopf$="Sieh an ..."
    o%=Objektauswahl%:(kopf$)
    IF o%
        REM hier zuerst Sonderfaelle bearbeiten
        REM dann die allgemeinen:
        IF benutzt%(o%)
            Teile_Satz:@$("Eigenschaft3"):(o%)
        ELSEIF genommen%(o%)
            Teile_Satz:@$("Eigenschaft2"):(o%)
        ELSE
            Teile_Satz:@$("Eigenschaft1"):(o%)
        ENDIF
        Textausgabe:
    ENDIF
ENDP

```

```

PROC Nehmen:
    LOCAL kopf$(11)                REM Titel fuer Kopfzeile
    LOCAL o%                       REM Objektnummer
    kopf$="Nimm ..."
    o%=Objektauswahl%:(kopf$)
    IF o%
        IF genommen%(o%)
            Teile_Satz:("Hast du schon!")
        ELSEIF nehmbar%(o%)
            O_ist_nehmbar:(o%)
        ELSE
            Teile_Satz:(Quatsch_nimm$:)
        ENDIF
        Textausgabe:
    ENDIF
ENDP

PROC O_ist_nehmbar:(obj%)
    LOCAL pflaster%
    LOCAL handschuh%
    REM Die ersten beiden Objekte sind fiktive Demo-Objekte!
    REM Es handelt sich im Ernstfall um Objekte, die mit der
    REM allgemeinen Routine "N_allgemein:" nicht hinreichend
    REM verarbeitet werden koennen.
    IF obj% = handschuh%
        N_handschuh:
    ELSEIF obj%=pflaster%
        N_pflaster:
    ELSE
        N_allgemein:(obj%)          REM alle unspezifischen
    ENDIF
    geaendert%= KTrue%
ENDP

PROC N_allgemein:(obj%)
    nehmbar%(obj%) = KFalse%
    genommen%(obj%) = KTrue%
    imRaum%(obj%) = KFalse%
    IF wirt%(obj%)      REM Wenn es einen Wirt fuer o% gibt ...
        REM Der Gast verlaesst den Wirt, der Wirt ist ohne Gast:
        gast%(wirt%(obj%)) = KFalse% REM Achtung Reihenfolge !!!
        wirt%(obj%) = KFalse%
    ENDIF
    IF obj% = kuli%
        benutzt%(schublade%) = KTrue%
    ENDIF
    IF obj%=schluessel%
        benutzt%(kaestchen%) = KTrue%
        sichtbar%(kaestchen%) = KFalse%
        genommen%(kaestchen%) = KFalse%
        nehmbar%(kaestchen%) = KFalse%
    ENDIF
    Teile_Satz:("OK. Ein Gegenstand mehr in der Tasche.")
ENDP

```

```

PROC N_pflaster:
    REM Nur Muster! Test auf spezif.Bedgg., die erfuehlt sein
    REM muessen damit das Objekt genommen werden kann.
    Teile_Satz:("Irgendetwas")
ENDP

PROC N_handschuh:
    REM Nur Muster! Test auf spezif.Bedgg., die erfuehlt sein
    REM muessen damit das Objekt genommen werden kann.
    Teile_Satz:("Irgendetwas")
ENDP

PROC Benutzen:
    LOCAL kopf$(11)                REM Titel fuer Kopfzeile
    LOCAL o%                       REM Objektnummer
    kopf$="Benutze ..."
    o%=Objektauswahl%:(kopf$)
    IF o%
        IF o%= kuli%
            B_kuli:
        ELSEIF o%= schluessel%
            B_schluessel:
        ELSEIF o%= briefoeffner%
            B_briefoeffner:
        ELSE
            Teile_Satz:("Das ist nicht erforderlich und geht
                        deshalb auch nicht!")
        ENDIF
        Textausgabe:
    ENDIF
ENDP

PROC B_schluessel:
    IF raum%= 5
        IF oeffnungsstatus%(tuer5%)= 4      REM offen,
            Teile_satz:("Wozu offene Türen aufschließen? Absurd!")
        ELSE
            Teile_satz:("Das Richtige zum richtigen Zeitpunkt.
                        Die verschlossene Tür ist nun auf!")
            wege%(raum%)=wege%(raum%) OR 8
            geaendert%= KTrue%
            benutzt%(tuer5%)= KTrue%
            oeffnungsstatus%(tuer5%)=4
            REM der benutzte Schluessel verschwindet:
            benutzt%(schluessel%)= KTrue%
            genommen%(schluessel%)= KFalse%
            sichtbar%(schluessel%)= KFalse%
        ENDIF
    ELSE
        Teile_satz:("Was willst Du hier mit dem Schlüssel machen?
                    Dein Monogramm in die Wand kratzen? Untersteh' Dich!")
    ENDIF
ENDP

```

```

PROC B_briefoeffner:
  IF raum%=6
    IF sichtbar%(koffer%)
      Teile_satz:("Du machst Dich wie ein gewöhnlicher
        Gauner daran, mit dem Brieföffner das
        Kofferschloß zu knacken - und es klappt!")
      benutzt%(briefoeffner%)= KTrue%
      sichtbar%(briefoeffner%)= KFalse%
      genommen%(briefoeffner%)= KFalse%
      benutzt%(koffer%)= KTrue%
      geaendert%= KTrue%
    ELSE
      Teile_satz:("Das könnte durchaus hier noch etwas
        werden, aber im Moment sind die
        Bedingungen dafür noch nicht reif.")
    ENDIF
  ELSE
    Teile_satz:("Hier kannst Du damit wirklich nichts
      anfangen")
  ENDIF
ENDP

PROC B_kuli:
  IF benutzt%(kuli%)
    Teile_Satz:("Du hast das Testament bereits einmal
      gegengezeichnet und bist damit Sieger des Spieles.
      Schätze, Du willst nur das Finale noch einmal
      sehen? Also dann ...")
    gewonnen%= KTrue%
  ELSE
    IF genommen%(testament%)
      Teile_Satz:("Du nimmst den Kuli und zeichnest das
        Testament mit schwungvoller Schrift gegen.
        Jetzt gehört alles Dir! Du hast das Spiel
        gewonnen! Gratuliere!")
      gewonnen%= KTrue%
      benutzt%(kuli%)= KTrue%
      benutzt%(testament%)= KTrue%
      geaendert%= KTrue%
    ELSE
      Teile_Satz:("Das, was Du mit dem Kugelschreiber
        verzieren möchtest, hast Du offenbar
        nicht in Deinem Besitz!")
    ENDIF
  ENDIF
ENDP

```

```

PROC Oeffnen:
    LOCAL kopf$(11)          REM Titel fuer Kopfzeile
    LOCAL o%                 REM Objektnummer
    kopf$="Öffne ..."
    o%=Objektauswahl%:(kopf$)
    IF o%

        IF oeffnungsStatus%(o%)=4
            REM Objekt ist bereits offen
            Teile_Satz:("Ist doch bereits offen!")
        ELSEIF oeffnungsStatus%(o%)=2
            REM Objekt ist zu, Schluessel erforderlich
            O_ist_verschlossen:(o%)
        ELSEIF oeffnungsStatus%(o%)=1
            REM Objekt ist geschlossen
            O_ist_zu:(o%)
        ELSE
            REM alles andere laesst sich nicht oeffnen
            Teile_Satz:(Quatsch_oeffnen$:)
        ENDIF
        Textausgabe:
    ENDIF
ENDP

PROC O_ist_verschlossen:(obj%)
    REM Es handelt sich um Spezialfaelle → Einzelbehandlung,
    REM denn jedes Objekt braucht eigenen 'Schluessel' zum Oeffnen
    IF obj%=tuer5%
        O_tuer5:
    ELSEIF obj%=koffer%
        O_koffer:
    ENDIF
ENDP

PROC O_tuer5:
    IF genommen%(schluessel%)
        Teile_Satz:("Gut, dass du den Schlüssel gefunden hast! Du
            steckst ihn ins Schloß, drehst ihn herum und -
            voila - die Tür ist auf!")
        wege%(raum%)=wege%(raum%) OR 8
        geaendert%= KTrue%
        benutzt%(tuer5%)= KTrue%
        oeffnungsstatus%(tuer5%)=4
        REM der benutzte Schluessel verschwindet:
        benutzt%(schluessel%)= KTrue%
        genommen%(schluessel%)= KFalse%
        sichtbar%(schluessel%)= KFalse%
    ELSE
        Teile_satz:("Wenn du den passenden Schlüssel hättest, wäre
            die Tür schon auf! Mach dich am besten gleich
            auf die Suche.")
    ENDIF
ENDP

```

```

PROC O_koffer:
  IF genommen%(briefoeffner%)
    Teile_Satz: ("Gut, dass Du den Brieföffner mitgenommen
                  hast. Nun dient er als 'Schlüssel' für den
                  Koffer. Der Koffer ist nun auf.")
    geaendert%= KTrue%
    benutzt%(koffer%)= KTrue%
    oeffnungsstatus%(koffer%)= 4
    REM Benutzer Briefoeffner verschwindet in der Versenkung:
    benutzt%(briefoeffner%)= KTrue%
    genommen%(briefoeffner%)= KFalse%
    sichtbar%(briefoeffner%)= KFalse%
    IF gast%(koffer%)
      sichtbar%(testament%)= KTrue%
      Teile_Satz: ("Du findest ein Testament.")
    ENDIF

  ELSE
    Teile_satz: ("Irgendetwas müsstest Du haben, um das
                 Kofferschloss zum Aufgehen zu überreden ...")
  ENDIF
ENDP

PROC O_ist_zu: (obj%)
  IF nehmbaar%(obj%)
    Teile_Satz: ("Erst nehmen, dann kann man vielleicht
                 mehr damit machen!")
  ELSE
    oeffnungsstatus%(obj%)=4
    Teile_Satz: ("OK. Ist jetzt offen!")
    IF gast%(obj%)
      sichtbar%(gast%(obj%))= KTrue%
      Teile_Satz: ("Du findest " +
                   uart$(gast%(obj%))
                   +objekt$(gast%(obj%)))
    ENDIF
    REM wenn das Programm grafisch ausgepraegter wird,
    REM muss bei optisch wirksamen Aenderungen der
    REM Raum neu gezeichnet werden, das wird
    REM durch setzen von "gaendert%" erreicht
    geaendert%= KTrue%
  ENDIF
ENDP

```

```

PROC Bewegen:
    LOCAL kopf$(11)          REM Titel fuer Kopfzeile
    LOCAL o%                 REM Objektnummer
    kopf$="Bewege ..."
    o%=Objektauswahl%:(kopf$)
    IF o%
        IF o%= bild%
            BW_bild:(o%)
        ELSE
            Teile_Satz:("Nö - das geht einfach nicht")
        ENDIF
        Textausgabe:
    ENDIF
ENDP

PROC BW_bild:(obj%)
    IF benutzt%(bild%)
        Teile_Satz:("Nicht nötig, das Bild noch ein weiteres Mal
                    herumzuschubsen. Das hat es nicht verdient!")
    ELSE
        benutzt%(bild%) = KTrue%
        sichtbar%(koffer%)= KTrue%
        geaendert%= KTrue%
        Teile_Satz:("Du schiebst das Bild ein wenig zur Seite und
                    siehe da ... ein Koffer kommt zum Vorschein!")
    ENDIF
ENDP

PROC Anklopfen:
    LOCAL kopf$(15)          REM Titel fuer Kopfzeile
    LOCAL o%                 REM Objektnummer
    kopf$="Klopfe an ..."
    o%=Objektauswahl%:(kopf$)
    IF o%
        Teile_Satz:(Quatsch_anklopfen%)
        Textausgabe:
    ENDIF
ENDP

PROC Anwesenheit%:(o%)
    REM ist das Objekt im aktuellen Raum?
    IF raum%= imRaum%(o%)
        RETURN KTrue%
    ENDIF
ENDP

```

```
REM *** Objektlistenhandling *****
```

```
PROC Objektauswahl%:(k$)
  GLOBAL lastobj%
  GLOBAL lastobjMax%
  GLOBAL objListe$(19,20)
  GLOBAL objListe%(19)
  GLOBAL objWin%(19)
  GLOBAL objBaseWin%
  LOCAL obj%

  lastobjMax%= 18
  Objektliste_erzeugen:

  IF psionTyp%= "revo"
    Objektliste_revo_anzeigen:(k$)
  ELSEIF psionTyp%= "s5"
    Objektliste_s5_anzeigen:(k$)
  ELSEIF psionTyp%= "netbook"
    Objektliste_nb_anzeigen:(k$)
  ENDIF

  obj%= Objektliste_abfragen%:
  Objektliste_Anz_schliessen:

  RETURN obj%
ENDP
```



```

PROC Objektliste_erzeugen:
  LOCAL n%                                REM Zaehlvariable
  lastobj%=0                             REM Zaehler der Objektliste auf Null

  REM zuerst alle "nehmbaren" Objekte in die Liste:
  n%=1
  DO
    IF sichtbar%(n%)
      IF genommen%(n%) OR Anwesenheit%:(n%)
        lastobj%=lastobj%+1
        objListe$(lastobj%)=objekt$(n%)
        objListe%(lastobj%)=n%
      ENDIF
    ENDIF
    n%=n%+1
  UNTIL n%>nehmbarMax% OR lastobj%>lastobjMax%

  REM dann alle sichtb. Obj. aus d.Obj.liste des Raumes fixo%(),
  REM bei fixomax% Null --> kein (verwendbares) Objekt im Raum:
  IF fixomax%
    n%=1
    WHILE n%<= fixomax%
      IF sichtbar%(fixo%(n%))
        lastobj%=lastobj%+1
        objListe%(lastobj%)=fixo%(n%)
        objListe$(lastobj%)=objekt$(fixo%(n%))
      ENDIF
      n%=n%+1
    ENDWH
  ENDIF

  REM Kniff: fuegt "Waende" als letztes Objekt an, damit
  REM existiert immer mindestens ein Objekt, ansonsten waere
  REM extra Aufwand erforderlich, um den Status "kein Objekt"
  REM irgendwie abzufangen
  lastobj%=lastobj%+1
  objListe%(lastobj%)=waende%
  objListe$(lastobj%)=objekt$(waende%)

ENDP

```

```

PROC Objektliste_revo_anzeigen:(k$)
    LOCAL bwx%      REM Basisfenster x-Position
    LOCAL bwy%      REM Basisfenster y-Position
    LOCAL bwb%      REM Basisfenster Breite
    LOCAL bwh%      REM Basisfenster Hoehe
    LOCAL owx%      REM Objektfenster x-Position
    LOCAL owy%      REM Objektfenster y-Position
    LOCAL owb%      REM Objektfenster Breite
    LOCAL owh%      REM Objektfenster Hoehe
    LOCAL n%        REM Zaehlvariable

    bwx%= 49      :   bwy%= 1
    bwb%= 392     :   bwh%= 159
    objBaseWin%= gCREATE(bwx%,bwy%,bwb%, bwh%,1)
    gBORDER 1
    gFONT KFontArialBold11&
    gAT 0,19      :   gLINEBY gWIDTH,0
    gAT 10,15     :   gPRINT k$

    owb%= 130     :   owh%= 20
    n%=1
    WHILE n%<= lastobj%
        IF n%<8
            owx%=50
            owy%= owh% * (n%)
        ELSEIF n%<15
            owx%=180
            owy%= owh% * (n%-7)
        ELSEIF n%<22
            owx%=310
            owy%= owh% * (n%-14)
        ENDIF
        objWin%(n%)= gCREATE(owx%,owy%,owb%, owh%,1)
        gBORDER 1
        gFONT KFontArialBold11&
        gAT 8,15
        gPRINT objListe$(n%)
        n%= n%+1
    ENDWH
ENDP

```

```

PROC Objektliste_s5_anzeigen: (k$)
    LOCAL bwx%    REM Basisfenster x-Position
    LOCAL bwy%    REM Basisfenster y-Position
    LOCAL bwb%    REM Basisfenster Breite
    LOCAL bwh%    REM Basisfenster Hoehe
    LOCAL owx%    REM Objektfenster x-Position
    LOCAL owy%    REM Objektfenster y-Position
    LOCAL owb%    REM Objektfenster Breite
    LOCAL owh%    REM Objektfenster Hoehe
    LOCAL n%      REM Zaehlvariable

    bwx%= 189 :   bwy%= 12
    bwb%= 263 :   bwh%= 210
    objBaseWin%= gCREATE(bwx%,bwy%,bwb%, bwh%,1)
    gBORDER 1
    gFONT KFontArialBold11&
    gAT 0,19 :   gLINEBY gWIDTH,0
    gAT 10,15 :  gPRINT k$

    owb%= 130 :   owh%= 20
    n%=1
    WHILE n%<= lastobj%
        IF n%<10
            owx%=190
            owy%= owh% + owh% * n%
        ELSEIF n%<19
            owx%= 320
            owy%= owh% + owh% * (n%-9)
        ENDIF
        objWin%(n%)= gCREATE(owx%,owy%,owb%, owh%,1)
        gBORDER 1
        gFONT KFontArialBold11&
        gAT 8,15
        gPRINT objListe$(n%)
        n%= n%+1
    ENDWH
ENDP

```

```

PROC Objektliste_nb_anzeigen:(k$)
  LOCAL bwx%      REM Basisfenster x-Position
  LOCAL bwy%      REM Basisfenster y-Position
  LOCAL bwb%      REM Basisfenster Breite
  LOCAL bwh%      REM Basisfenster Hoehe
  LOCAL owx%      REM Objektfenster x-Position
  LOCAL owy%      REM Objektfenster y-Position
  LOCAL owb%      REM Objektfenster Breite
  LOCAL owh%      REM Objektfenster Hoehe
  LOCAL hkorrr%   REM vert. Positionskorrektur
  LOCAL n%        REM Zaehlvariable

  hkorrr%=120

  bwx%= 189 :   bwy%= 12 + hkorrr%
  bwb%= 263 :   bwh%= 210
  objBaseWin%= gCREATE(bwx%,bwy%,bwb%, bwh%,1)
  gBORDER 1
  gFONT KFontArialBold11&
  gAT 0,19 :   gLINEBY gWIDTH,0
  gAT 10,15 :  gPRINT k$

  owb%= 130 :   owh%= 20

  n%=1
  WHILE n%<= lastobj%
    IF n%<10
      owx%=190
      owy%= owh% + hkorrr% + owh% * (n%)
    ELSEIF n%<19
      owx%=320
      owy%= owh% + hkorrr% + owh% * (n%-9)
    ENDIF
    objWin%(n%)= gCREATE(owx%,owy%,owb%, owh%,1)
    gBORDER 1
    gFONT KFontArialBold11&
    gAT 8,15
    gPRINT objListe$(n%)
    n%= n%+1
  ENDWH
ENDP

```

```

PROC Objektliste_abfragen:
    REM Abfr.f.Objektauswahl, ausser ESC-Taste nur Stiftabfrage.
    REM Rueckgriff auf globale Variablen!
    LOCAL n%           REM Zaehlvariable
    LOCAL num%         REM auserwaehlte Objektnummer
    LOCAL e%(16)       REM Feld f.d.GETEVENT32-Variablen
    LOCAL event%       REM Event: Art
    LOCAL winId%       REM Event: in welchem Fenster
    LOCAL mod%         REM Event: Modifikator

    DO
        GETEVENT32 e%()
        event%=e%(1)
        winId%=e%(3)
        mod%=e%(4)
        num%=0
        IF (event% AND $400)=0           REM Tastaturevent
            IF event%=27                 REM Esc-Taste
                BREAK                   REM num% bleibt Null
            ENDIF
        ELSEIF event%=$408 and mod%=1    REM Pen hat abgehoben
            n%=1
            DO
                IF winId%=objWin%(n%)
                    num%=objListe%(n%)
                    BREAK
                ENDIF
                n%=n%+1
            UNTIL n%>lastobj%
            BREAK
        ENDIF
    UNTIL 0
    RETURN num%
ENDP

PROC Objektliste_Anz_schliessen:
    LOCAL n%
    n%=1
    WHILE n%<= lastobj%
        gCLOSE objWin%(n%)
        n%=n%+1
    ENDWH
    gCLOSE objBaseWin%
    gUSE gameWin%
ENDP

```

```

PROC Quatsch_oeffnen$:
    REM gibt eine Zufallsantwort fuer nicht zu oeffnende Objekte
    LOCAL random%
    randomize minute+second
    random%=1+INT(14*RND)
    VECTOR random%
        a1, a2, a3, a4, a5, a6,a7,a8, a9, a10, a11, a12, a13, a14
    ENDV
    a1:: RETURN "Das geht beim besten Willen nicht!"
    a2:: RETURN "Nee. Laß dir mal was anderes einfallen!"
    a3:: RETURN "Genau das geht hier nicht!"
    a4:: RETURN "Zu dumm, das klappt hier nicht!"
    a5:: RETURN "Guter Versuch. Ist aber nicht möglich."
    a6:: RETURN "Du könntest auch 'was Sinnvolles tun!"
    a7:: RETURN "Denk mal vorher nach, bevor du das wieder
        ausprobierst."
    a8:: RETURN "Manchmal klappt's. Hier aber nicht."
    a9:: RETURN "Das ist's, was du möchtest. Es geht aber nicht."
    a10:: RETURN "Dein schlechter Tag heute. Es geht nicht."
    a11:: RETURN "Du denkst also wirklich, so'was könnte
        funktionieren?"
    a12:: RETURN "Allen Ernstes? Vergiß' es!"
    a13:: RETURN "Das kannst du doch nicht wirklich wollen! Es
        geht nämlich nicht!"
    a14:: RETURN "Würdest du das auch im richtigen Leben
        versuchen? Mach' was anderes!"
ENDP

PROC Quatsch_nimm$:
    REM gibt eine Zufallsantwort fuer nicht zu oeffnende Objekte
    LOCAL random%
    randomize minute+second
    random%=1+INT(14*RND)
    VECTOR random%
        a1, a2, a3, a4, a5,a6,a7, a8, a9, a10, a11, a12, a13, a14
    ENDV
    a1:: RETURN "Also DAS geht beim besten Willen nicht!"
    a2:: RETURN "Nö. In diesem Falle mal nicht!"
    a3:: RETURN "Also komm, das lass mal an seinem Platz!"
    a4:: RETURN "Dumm gelaufen, denn das klappt hier nicht!"
    a5:: RETURN "Guter Versuch. Ist aber nicht möglich."
    a6:: RETURN "Raff, raff, raff. Wie wär's mit etwas mehr
        Bescheidenheit!"
    a7:: RETURN "Ähhhh .. nö! Diesmal nicht!"
    a8:: RETURN "Manchmal klappt's. Aber diesmal ist's
        vergeblich."
    a9:: RETURN "Selbst, wenn du es wirklich möchtest, es geht
        nicht."
    a10:: RETURN "Dein schlechter Tag heute. Es geht nicht."
    a11:: RETURN "Du denkst wirklich, man kann das nehmen? Haha!"
    a12:: RETURN "Allen Ernstes? Vergiß' es!"
    a13:: RETURN "Das willst du doch nicht wirklich an dich
        bringen? Es geht nämlich nicht!"
    a14:: RETURN "Das geht schlichtweg mal nicht!"

```

ENDP

PROC Quatsch_anklopfen\$:

```
REM gibt eine Zufallsantwort fuer "Anklopfen"
LOCAL random%
randomize minute+second
random%=1+INT(3*RND)
VECTOR random%
    a1, a2, a3
ENDV
a1:: RETURN "Klopf, klopf ..."
a2:: RETURN "Meinst Du, das ist hier die richtige Methode?"
a3:: RETURN "Du denkst also allen Ernstes, das bringt Dich
weiter?"
```

ENDP

REM *** Raum-spezifische Teilprogramme und
Funktionen*****

PROC ZeichneRaum:

```
REM Zeichnet den aktuellen Raum inkl. Tueren

gCLS : gBORDER 3

REM Die Waende:
gAT 1,1 : gLINETO 74,42 : gBOX 127,72
gAT 201,114 : gLINETO 275,155
gAT 1,155 : gLINETO 74,113
gAT 201,42 : gLINETO 275,1
gAT 1,148 : gFILL 274,10,0

REM Die Tueren:
IF tuerW%(raum%) REM linke Tuer
    gAT 35,135 : gLINETO 35,78
    gLINETO 55,78 : gLINETO 55,125
    gAT 41,105 : gBOX 2,2
ENDIF
IF tuerO%(raum%) REM rechte Tuer
    gAT 241,135 : gLINETO 241,78
    gLINETO 221,78 : gLINETO 221,125
    gAT 235,105 : gBOX 2,2
ENDIF
IF tuerS%(raum%) REM vordere Tuer = Loch in der Mauer
    gAT 105,146 : gFILL 70,12,1
ENDIF
IF tuerN%(raum%) REM hintere Tuer
    gAT 129,78 : gBOX 20,36
    gAT 134,98 : gBOX 2,2
ENDIF
SetzeTitel:
ENDP
```

```

PROC SetzeTitel:
    REM schreibt die Bezeichnung des aktuellen Raumes ins Fenster
    LOCAL a$(24)
    gfont KFontArialBold11&
    gAT 6,4
    gFILL 240,12,1    REM weiss fuellen
    a$=RaumTitel$:(raum%)
    gAT 8,14
    gPRINT a$
ENDP

PROC RaumTitel$:(r%)
    REM liefert Raumnamen zurueck
    VECTOR r%
        rr1, rr2, rr3, rr4, rr5, rr6
    ENDV
    rr1:: RETURN "Flur"
    rr2:: RETURN "Unterer Treppenabsatz"
    rr3:: RETURN "Wirtschaftsraum"
    rr4:: RETURN "Arbeitszimmer"
    rr5:: RETURN "Oberer Treppenabsatz"
    rr6:: RETURN "Abstellkammer"
ENDP

PROC Raumbeschreibung$:(raumnummer%)
    VECTOR raumnummer%
        raum1, raum2, raum3, raum4, raum5, raum6
    ENDV
    raum1:: RETURN "Lauter langweilige Wände in diesem Flur."
    raum2:: RETURN "Der Treppenaufgang zur nächsten Etage."
    raum3:: RETURN "Der Wirtschaftsraum ist weitgehend leer
        geräumt."
    raum4:: RETURN "Das einst viel benutzte Arbeitszimmer. Der
        Onkel rauchte Zigarren, die Tapete
        hat darunter gelitten."
    raum5:: RETURN "Nur der obere Treppenabsatz, nichts weiter.
        Die hölzernen Stufen führen ins Erdgeschoß ..."
    raum6:: RETURN "Eigentlich die Gerümpelkammer. Meist versteckt
        sich in solchen Kammern etwas Geheimnisvolles!"
ENDP

PROC R1:
    fixomax%=0
ENDP

```



```

PROC R2:
    REM gGMode 0
    REM Treppe
    gAT 114,38 : gBOX 48,5 : gLINETO 102,15
    gAT 161,38 : gLINETO 173,15
    gAT 114,41 : gBOX 48,7
    gLINETO 100,15: gLINETO 175,15: gLINETO 162,41
    gAT 113,46 : gBOX 50,8
    gAT 112,52 : gBOX 52,8
    gAT 111,58 : gBOX 54,8
    gAT 110,64 : gBOX 56,9
    gAT 109,71 : gBOX 58,9
    gAT 108,78 : gBOX 60,9
    gAT 107,85 : gBOX 62,9
    gAT 106,93 : gFILL 64,3,0
    gAT 105,94 : gBOX 66,10
    gAT 104,103 : gFILL 68,4,0
    gAT 103,104 : gBOX 70,10
    gAT 102,114 : gFILL 72,6,0
    gAT 100,117 : gFILL 76,3,0
    gAT 101,115 : gFILL 74,4,0
    gAT 101,119 : gBOX 75,8
    gAT 102,106 : gLINETO 173,106

    fixomax%=1
    fixo%(1)=treppe2%
ENDP

PROC R3:
    REM Schrank
    gAT 118,73 : gBOX 38,17
    gAT 120,75 : gBOX 16,13
    gAT 138,75 : gBOX 16,13
    gAT 131,80 : gBOX 2,2
    gAT 141,80 : gBOX 2,2

    fixomax%=1
    fixo%(1)=schrank%
ENDP

```

```

PROC R4:
    REM Schreibtisch
    gAT 110,112 : gFILL 63,3,1      REM Hintergrund löschen
    gAT 108,98
    gLINEBY 9,-4 : gLINEBY 48,0 : gLINEBY 9,4 : gLINEBY -66,0
    gFILL 67,2,0
    gAT 110,99 : gBOX 21,26
    gAT 112,101 : gBOX 17,11
    gAT 115,104 : gLINEBY 11,0
    gAT 153,99 : gBOX 21,26
    gAT 155,101 : gBOX 17,11
    gAT 158,104 : gLINEBY 11,0
    gAT 135,99 : gBOX 14,15
    gAT 129, 124 : gLINEBY 6,-11
    gAT 153, 124 : gLINEBY -6,-11

    IF imRaum%(briefoeffner%)
        REM Briefoeffner
        gAT 130,96 : gLINEBY 15,0
    ENDIF

    fixomax%=2
    fixo%(1)=schreibtisch%
    fixo%(2)=schublade%
ENDP

PROC R5:
    gAT 106,157 : gLINETO 123,140
    gAT 174,157 : gLINETO 157,140
    gAT 123,140 : gBOX 35,6
    gAT 123,145 : gBOX 35,2
    gAT 120,147 : gBOX 41,2
    gAT 121,148 : gBOX 39,6
    gAT 120,153 : gBOX 41,2
    gAT 118,155 : gBOX 45,2
    gAT 119,157 : gBOX 43,3
    fixomax%=2
    fixo%(1)=treppe5%
    fixo%(2)=tuer5%
ENDP

```

```

PROC R6:
  REM Bilder an der Wand
  gAT 210,110 : gFILL 20,25,1
  trapez: (222,76,208,126,228,139,246,77)
  trapez: (224,78,210,124,226,136,244,79)
  gAT 227,81 : gLINETO 229,87 : gLINETO 236,82
  IF sichtbar%(koffer%)
    REM Koffer
    gAT 115,126 : gBOX 46,12 : gBOX 46,6
    gLINEBY 8,-8 : gLINEBY 30,0 : gLINEBY 8,8
    gAT 120,129 : gFILL 3,6,1 : gBOX 3,6
    gAT 153,129 : gFILL 3,6,1 : gBOX 3,6
    gAT 130,129 : gFILL 16,4,0
  ENDIF
  fixomax%=1
  IF sichtbar%(koffer%)
    IF oeffnungsStatus%(koffer%)=4 REM geoeffnet
      REM Koffer geoeffnet
      gAT 115,132 : gBOX 46,6
      gLINEBY 8,-8 : gLINEBY 30,0 : gLINEBY 8,8
      gAT 120,129 : gFILL 3,6,1 : gBOX 3,6
      gAT 153,129 : gFILL 3,6,1 : gBOX 3,6
      gAT 130,132 : gFILL 16,2,0
      gAT 123,100 : gFILL 30,24,1 : gBOX 30,24
    ELSE
      REM Koffer geschlossen
      gAT 115,126 : gBOX 46,12 : gBOX 46,6
      gLINEBY 8,-8 : gLINEBY 30,0 : gLINEBY 8,8
      gAT 120,129 : gFILL 3,6,1 : gBOX 3,6
      gAT 153,129 : gFILL 3,6,1 : gBOX 3,6
      gAT 130,129 : gFILL 16,4,0
    ENDIF
    fixomax%= 2
  ENDIF
  fixo%(1)=bild%
  fixo%(2)= koffer%
ENDP

PROC trapez: (x1%,y1%,x2%,y2%,x3%,y3%,x4%,y4%)
  REM Zeichnet geschlossenes Viereck
  gAT x1%,y1%
  gLINETO x2%,y2%
  gLINETO x3%,y3%
  gLINETO x4%,y4%
  gLINETO x1%,y1%
ENDP

```

```

REM *** Eigenschaften der Objekte
*****
REM Eigenschaft1= im Grundzustand, E2= wenn Objekt genommen,
REM E3= wenn Objekt benutzt

PROC Eigenschaft1$(o%)

    VECTOR o%
    REM Begriffe in Reihenfolge d. Initialisierung d. Objekte!
    testament, schluessel, kuli, briefoeffner, kaestchen
    schreibtisch, schublade, treppe2, schrank, treppe5
    tuer5, bild, koffer, waende
    ENDV

    testament:: RETURN "Das Testament macht Dich reich.
                        Außerdem erbst Du die Villa. Aber erst
                        wenn Du das Testament gegengezeichnet
                        hast!"
    schluessel:: RETURN "Ein Schlüssel mit einem sooooo langen
                        Bart."
    kuli:: RETURN "Ein simpler Kugelschreiber mit voller
                        Mine"
    briefoeffner:: RETURN "Ein recht robuster Brieföffner."
    kaestchen::
        IF oeffnungsStatus%(kaestchen%)= 4
            RETURN "Ein niedliches Kästchen aus Holz. Es ist
                    geöffnet. Ein Schlüssel blickt Dich fragend an."
        ELSE
            RETURN "Ein niedliches Kästchen aus Holz."
        ENDIF
    schreibtisch:: RETURN "Der Schreibtisch ist antik. Leider
                        haben ihn schon die Holzwürmer besucht. Er
                        hat eine Schublade."
    schublade::
        IF oeffnungsStatus%(schublade%)= 4
            RETURN "Die Schublade hat einen Griff und ist weit
                    herausgezogen. Es liegt ein Kugelschreiber drin."
        ELSE
            RETURN "Die Schublade hat einen Griff. Man möchte
                    glatt dran ziehen!"
        ENDIF
    treppe2:: RETURN "Die Treppe führt in den ersten Stock
                        des Hauses."
    schrank::
        IF oeffnungsStatus%(schrank%)= 4
            RETURN "Der Hängeschränk steht sperrangelweit auf."
        ELSE
            RETURN "Der Hängeschränk hängt lässig in der Gegend
                    herum und erfreut sich bester Gesundheit."
        ENDIF
    treppe5:: RETURN "Die Treppe führt Dich wieder hinunter
                        in das Erdgeschoss."

    tuer5:: RETURN "Die Tür ist verschlossen. Meist hilft

```

```

        es, wenn man den passenden Schlüssel dazu hat."
bild::      RETURN "Das Bild steht leider mit dem Gesicht
              zur Wand, so dass man nicht sehen kann, wie
              glanzvoll es ist."
koffer::
    IF oeffnungsStatus%(koffer%)= 4
        IF nehmbar%(testament%)
            RETURN "Du hast den Koffer gewaltsam geöffnet. Da
                    liegt das Testament drinnen!"
        ELSE
            RETURN "Du hast den Koffer gewaltsam geöffnet. Außer
                    etwas Kleinkram ist nichts wirklich wichtiges
                    darin."
        ENDIF
    ELSE
        RETURN "Der Koffer ist verschlossen. Hast Du einen
                Schlüssel oder was denkst Du, wie Du ihn auf
                bekommst?"
    ENDIF
waende:: RETURN Waende$:
RETURN "Häh?"
ENDP

PROC Eigenschaft2$(o%)
    REM Betrifft nur genommene Objekte, daher ist die VECTOR-Liste
    REM kuerzer. Begriffe in Reihenfolge der Initialisierung der
    REM Objekte!
    VECTOR o%
        testament, schluessel, kuli, briefoeffner, kaestchen
    ENDV
    testament::
        IF genommen%(kuli%)
            RETURN "Du hältst das gesuchte Stück nun in Deinen
                    etwas zittrigen Händen. Du musst es
                    gegenzeichnen, damit es rechtlich gültig wird!"
        ELSE
            RETURN "Du hältst das gesuchte Stück nun in Deinen
                    etwas zittrigen Händen. Du brauchst etwas zum
                    Schreiben, um es gegenzeichnen zu können!"
        ENDIF
    schluessel:: RETURN "Ein Schlüssel. Es wird sich wohl ein
                        passendes Schlüsselloch dafür finden lassen ..."
    kuli::      RETURN "Ein simpler Kugelschreiber mit voller
                        Mine. Damit muss man einfach schreiben ...
                        oder unterschreiben."
    briefoeffner:: RETURN "Ein recht robuster Brieföffner - fast
                        schon eine Art Brechstange."
    kaestchen::
        IF oeffnungsStatus%(kaestchen%)= 4
            RETURN "Ein niedliches offenes Kästchen aus Holz, in
                    dem ein Schlüssel liegt."
        ELSE
            RETURN "Ein geschlossenes Kästchen aus Holz."
        ENDIF

```

```

        RETURN "Häh?"
    ENDP

PROC Eigenschaft3$(o%)
    REM Liefert den Text für benutzte Objekte; die meisten
    REM benutzten "nehmbaren" verschwinden nach Gebrauch aus
    REM dem Inventar!

    VECTOR o%
        REM Begriffe in Reihenfolge der Initialisierung der
        REM Objekte!
        testament, schluessel, kuli,    briefoeffner, kaestchen
        schreibtisch, schublade, treppe2, schrank, treppe5
        tuer5, bild, koffer, waende
    ENDV

    testament:: RETURN "Du hast das Testament gegengezeichnet
                        und bist nun rechtlich Besitzer eines
                        Vermögens und der Villa. Gratuliere!"
    schluessel:: RETURN ""
    kuli::      RETURN "Der Kugelschreiber hat seine
                        Schuldigkeit getan."
    briefoeffner:: RETURN ""
    kaestchen:: RETURN "Ein niedliches Kästchen aus Holz. Es
                        ist geöffnet. Vor kurzem lag da noch ein
                        Schlüssel drin."
    schreibtisch:: RETURN ""
    schublade:: RETURN "Die Schublade hat einen Griff und ist
                        weit herausgezogen. Der Kugelschreiber, der
                        hier lag, ist weg."
    treppe2::   RETURN ""
    schrank::   RETURN ""
    treppe5::   RETURN ""
    tuer5::     RETURN "Da war jemand ganz pfiffig und hat die
                        Tür mit einem Schlüssel geöffnet."
    bild::      RETURN ""
    koffer::
        IF genommen%(testament%)
            RETURN "Der Koffer ist offen und zeigt Dir freimütig
                    alles, was er enthält: nichts"
        ELSE
            RETURN "Der Koffer ist offen und zeigt Dir freimütig
                    alles, was er enthält: ein Testament"
        ENDIF
    waende::    RETURN Waende$:
    RETURN "Häh?"
ENDP

PROC Waende$:
    REM gibt eine Zufallsantwort fuer die Ansicht der Waende
    LOCAL random%
    randomize minute+second
    random%=1+INT(7*RND)
    VECTOR random%
        a1, a2, a3, a4, a5,a6,a7

```

```

ENDV
a1:: RETURN "Pass auf, sie schauen zurück!"
a2:: RETURN "Wenn das mal keine Wände sind!"
a3:: RETURN "Eine sieht aus wie die andere."
a4:: RETURN "Hier ist nun wirklich nichts besonderes zu
          sehen."
a5:: RETURN "Wie üblich: vier Wände bilden einen Raum."
a6:: RETURN "Könnte mal wieder renoviert werden ..."
a7:: RETURN "Wände ... was erwartest du?"
ENDP

REM *** Textverarbeitungs-Routinen
*****

PROC Textausgabe:
  REM Bildschirmausgabe von Meldetexten aller Art
  REM Zeilenbegrenzung b.Bedarf in den UNTIL-Zeilen anpassen
  REM Feldanzahl-Deklaration von textzeile$() nicht vergessen
  REM Max. 40 Zeichen pro Zeile (40*"W" füllt den Bildschirm!)
  LOCAL n%

  dINIT "",2      REM Titel nicht anzeigen
  dPOSITION 0,0
  n%=1
  DO
    IF textzeile$(n%)<>""
      dTEXT "",textzeile$(n%)
    ENDIF
    n%=n%+1
  UNTIL n%>7
  dBUTTONS "Enter oder Esc",13+$100
  DIALOG
  n%=1
  DO
    textzeile$(n%)=""
    n%=n%+1
  UNTIL n%>7
ENDP

```

```

PROC Teile_Satz:(s$)
    REM zerlegt TextString v.max.255 Zeichen in max.6 50er Zeilen
    REM String kann mit Komma, Leerzeichen oder beliebig enden.
    REM Es duerfen bereits Zeilen belegt sein.
    REM (Zur Ausgabe Textausgabe-Routine extern aufrufen!)
    REM Passt der zu verarbeitende String nicht in die Zeilen,
    REM wird der Wert der lokalen Variable overrun%
    REM auf -1 gesetzt und zurueckgegeben an d.aufrufende Progr.,
    REM das den Fehler selber abfangen muss.
    REM Das wird hier nicht genutzt!
    REM Fuer 7 Zeilen zu max. 50 Zeichen muessen global deklariert
sein:
    REM textzeile$(7,50)
    LOCAL satz$(255)          REM f.Satz-Uebernahme & Manipulation
    LOCAL maxZeilenLaenge%    REM zur Uebergabe an Unterprogramm
    LOCAL maxZeilen%          REM max. Zeilenanzahl
    LOCAL z%                  REM Zaehlvariable
    LOCAL pos%                REM Zeiger auf Trennstelle

    maxZeilenLaenge%= 50
    maxZeilen%= 7
    satz$= s$

    REM Komma entfernen (falls vorh.) & Leerzeichen anfüegen:
    IF RIGHT$(satz$,1)=","
        satz$=LEFT$(satz$,LEN(satz$)-1)
        satz$=satz$+" "
    ELSEIF RIGHT$(satz$,LEN(satz$))<>" "
        satz$=satz$+" "
    ENDIF

    REM erste freie textzeile$() suchen:
    z%=1
    DO
        IF textzeile$(z%)=""
            BREAK
        ENDIF
        z%=z%+1
    UNTIL z%>maxZeilen%

    IF z%<=maxZeilen%      REM nur, wenn freie Zeilen da sind!
        REM Text auf freie Zeilen aufteilen:
        WHILE LEN(satz$)>50 AND z%<maxZeilen%
            pos%= Finde_Leerzeichen:(satz$,maxZeilenLaenge%)
            textzeile$(z%)= LEFT$(satz$,pos%-1)
            satz$= RIGHT$(satz$, LEN(satz$)-pos%)
            z%= z%+1
        ENDWH

        REM Resttext in naechste/letzte Zeile uebernehmen;
        REM war der Text laenger, als die 7 zeilen aufnehmen
        REM konnten, wird er einfach abgehackt,
        REM ein Programmfehler wird vermieden.
    
```



```

        IF LEN(satz$)<=50
            REM sauberer Abschluss:
            REM die naechsten 2 Zeilen in eine Zeile schreiben!!!
            textzeile$(z%)=
                LEFT$(satz$,LEN(satz$)-1) REM Leerz.entfernen
        ELSE
            REM Text wird abgehackt:
            textzeile$(z%)= LEFT$(satz$,maxZeilenLaenge%)
        ENDIF
    ENDIF
ENDP

PROC Finde_Leerzeichen:(satz$,maxLen%)
    LOCAL pos%
    LOCAL lastpos%
    pos%=1 : lastpos%= pos%
    WHILE pos%<maxLen%
        IF MID$(satz$,pos%,1)=" "
            lastpos%= pos%
        ENDIF
        pos%= pos%+1
    ENDWH
    RETURN lastpos%
ENDP

REM Menue- und Toolbarfunktionen
*****

PROC Zeige_Menu:
    LOCAL key&

    mINIT
    REM die naechsten beiden Zeilen in eine Zeile schreiben!!!
    mCARD "Datei","Spielstand speichern",%s,
        "Spielstand laden",-%r,"Beenden",%e
    REM die naechsten beiden Zeilen in eine Zeile schreiben!!!
    mCARD "Ansicht","Inventar",%i
    mCARD "Optionen","Ton an/aus",%b,"Tondauer",%d
    mCARD "Extras","Hilfe",-%H,"Warnung",%w,"Über HH2000",%a
    key%=MENU(minit%)

    Handle_chars:(key&)

ENDP

PROC Cmdl%:
    Inventar:
ENDP

PROC Cmds%:
    Spiel_speichern:
ENDP

PROC Cmdl%:

```

```

        Spiel_laden:
ENDP

PROC Cmdb%:
    Ton_an_aus:
ENDP

PROC Cmda%:
    Programm_Info:
ENDP

PROC CmdSH%:
    Hilfe:
ENDP

PROC Spiel_laden:
    LOCAL file$(255)      REM Name des Datenfiles
    LOCAL f%              REM Aussehen der Editbox f. Dateinamen
    LOCAL kennung$(128)   REM Hilfsvariable f. Filekennungsabfrage
    REM zur Erinnerung: dataPath$="C:\HH2000\"

    file$=dataPath$ + "bin\"
    f%= 16

    dINIT "Spielstand laden"
    dTEXT "", "Datei auswählen:", 2
    dFILE file$, "", f%
    dBUTTONS "Abbrechen", 27, "Laden", 13
    IF DIALOG
        IF RIGHT$(file$, 3) = "h2k"
            Laden: (file$)
            IF ton% : BEEP beepLen%, freq% : ENDIF
            giPRINT "Spiel geladen", 0
            geaendert% = KTrue%
        ELSE
            IF ton% : BEEP 10, 333 : ENDIF
            giPRINT "Das ist keine gültige Spielstandsdatei!", 0
            ENDIF
        ENDIF
    ENDIF
ENDP

```

```

PROC Spiel_speichern:
    LOCAL n%          REM Zaehlvariable
    LOCAL file$(255)  REM Name des Datenfiles
    LOCAL f%          REM Aussehen der Editbox f. Dateinamen
    LOCAL l%          REM Hilfsvariable f. Stringlaenge
    REM zur Erinnerung: dataPath$="C:\HH2000\"
    f%= 1 + 128      REM Editbox + Wildcards verwenden
    file$=dataPath$ + "bin\*.h2k"
    REM Endung ".h2k" wird automatisch uebernommen, aber nur wenn
    REM der Nutzer weder Verzeichnis noch Laufwerk veraendert!
    dINIT "Spielstand speichern"
        dTEXT " ", "Bezeichnung eingeben:", 2
        dFILE file$, "", f%
        dBUTTONS "Abbrechen", 27, "Speichern", 13
    IF DIALOG
        BUSY "Speichern ...", 0
        IF EXIST (file$)
            DELETE (file$)
        ENDIF
        CREATE file$ + " FIELDS wert TO Variablen", A, w%
        REM Zustand der Objekte
        n%=objekteMax%
        WHILE n%>0
            INSERT : A.w%= sichtbar%(n%) : PUT
            INSERT : A.w%= nehmbaar%(n%) : PUT
            INSERT : A.w%= genommen%(n%) : PUT
            INSERT : A.w%= oeffnungsStatus%(n%) : PUT
            INSERT : A.w%= wirt%(n%) : PUT
            INSERT : A.w%= gast%(n%) : PUT
            INSERT : A.w%= benutzt%(n%) : PUT
            INSERT : A.w%= imRaum%(n%) : PUT
            n%=n%-1
        ENDWH
        REM raumspezifische Daten (offene Tueren/Waende)
        n%=raumMax%
        WHILE n%>0
            INSERT : A.w%= tuerW%(n%) : PUT
            INSERT : A.w%= tuerO%(n%) : PUT
            INSERT : A.w%= tuerN%(n%) : PUT
            INSERT : A.w%= tuerS%(n%) : PUT
            INSERT : A.w%= wege%(n%) : PUT
            n%=n%-1
        ENDWH
        REM Raumnummer
        INSERT : A.w%= raum% : PUT
        REM Spielende erreicht
        INSERT : A.w%= gewonnen% : PUT
        CLOSE
        BUSY OFF : BUSY "Komprimiere ...", 0
        COMPACT file$
        BUSY OFF
        IF ton% : BEEP beepLen%, freq% : ENDIF
        giPRINT "Spiel gespeichert ...", 0
    ENDIF
ENDP

```

```

PROC Laden: (db$)
  LOCAL n%
  OPEN db$ + " SELECT wert FROM Variablen",A,w%
  FIRST

  REM Objektzustand
  n%=objekteMax%
  WHILE n%>0
    sichtbar%(n%)= A.w% : NEXT
    nehmbar%(n%)= A.w% : NEXT
    genommen%(n%)= A.w% : NEXT
    oeffnungsStatus%(n%)= A.w% : NEXT
    wirt%(n%)= A.w% : NEXT
    gast%(n%)= A.w% : NEXT
    benutzt%(n%)= A.w% : NEXT
    imRaum%(n%)= A.w% : NEXT
    n%=n%-1
  ENDWH

  REM raumspezifische Daten
  n%=raumMax%
  WHILE n%>0
    tuerW%(n%)= A.w% : NEXT
    tuerO%(n%)= A.w% : NEXT
    tuerN%(n%)= A.w% : NEXT
    tuerS%(n%)= A.w% : NEXT
    wege%(n%)= A.w% : NEXT
    n%=n%-1
  ENDWH
  raum%= A.w% : NEXT
  gewonnen%= A.w%
CLOSE
ENDP

PROC Inventar:
  REM Inventar listen
  LOCAL z$(255), n%
  REM alle genommenen in einen String bringen:
  n%=1
  z$="Du hast ..."
  DO
    IF genommen%(n%)
      z%=z$+uart$(n%)+objekt$(n%)+", "
    ENDIF
    n%=n%+1
  UNTIL n%>nehmbarMax%
  IF z%= "Du hast ..."
    z%=z$+"nichts weiter!"
  ENDIF
  Teile_Satz: (z$)
  Textausgabe:
ENDP

```

```

PROC Ton_an_aus:
  IF ton%
    ton%= KFalse%
    gIPRINT "Piepser sind jetzt AUS",0
  ELSE
    ton%= KTrue%
    gIPRINT "Piepser sind jetzt AN",0
  ENDIF
  Save_ini:
ENDP

PROC Tondauer:
  LOCAL long&
  long&=beepLen%
  dINIT "Länge der Piepser ändern"
  dTEXT "", "32= ca.1 Sekunde",2
  dText "", "Standard=1, max. Wert=3840",2
  dLONG long&, "Neuer Wert:",1,3840
  dBUTTONS "Abbruch",27,"Ändern",13
  DIALOG
  beepLen%=long& AND &FFF
ENDP

PROC Hilfe:
  LOCAL file$(255)
  file$=laufwerk$+progPath$+"hilfe.hlp"
  IF EXIST (file$)
    ONERR Hilfedatei_nicht_offen::

    REM geoeffnete Hilfe-Datei in d.Vordergrund:
    SetForegroundByThread:(thread&,0)

  ELSE
    gIPRINT "Hilfedatei nicht vorhanden.",0
  ENDIF

  RETURN

  Hilfedatei_nicht_offen::
  ONERR OFF
  thread&=RunApp&:("Data",file$,"",0)
ENDP

PROC Warnung:
  Teile_Satz:("Dieses Programm erfordert den intensiven Gebrauch
    des Stiftes. Es besteht Gefahr des
Zerkratzens
    des Bildschirms! Der Autor übernimmt keine
    Haftung für irgendwelche Schäden, die durch
    Nutzung des vorliegenden Programmes
    entstehen!")

```

```

        Textausgabe:
ENDP

```

```

PROC Programm_Info:
    dINIT "Über "+programm$+" "+version$
    dTEXT "","Das ist ein Programm von ...", 2
    dTEXT "","Hacker Arno",2
    dTEXT "","Email: arno@gmx.net",2
    dTEXT "","Web : http://www.thefreespace.com/arno/",2
    dBUTTONS "Weiter",13
    DIALOG
ENDP

```

```

PROC Exit:
    Save_ini:
    STOP
ENDP

```

```

PROC Fehlermeldung: (p$,v$)
    DINIT "Fehlermeldung"REM ,2
    dTEXT "","Sorry, das sollte nicht mehr passieren!"
    dTEXT "","Bitte eMail an: autor@hotmail.com"
    dTEXT "","mit möglichst genauer Fehlerbeschreibung."
    dTEXT "","Dazu hätte ich gern auch noch die nachstehende"
    dTEXT "","Information: -danke für die Hilfe!-"
    dTEXT "","", $800
    dTEXT "","p$+v$+", "+" Es gab einen "
    dTEXT "","ERRX$+", "
    dTEXT "","und zwar : "+ERR$(ERR)
    dBUTTONS "Weiter",13 OR $300
    DIALOG
ENDP

```

```

REM *** Programm-Initialisierung
*****

```

```

PROC Load_ini:
    IF EXIST (dataPath$+"hh2000.ini")
        OPEN dataPath$+"hh2000.ini",A,wert%
        ton%= A.wert%
        CLOSE
    ELSE
        REM nicht vorhanden, also erzeuge die ini-Datei
        ton%= KTrue%
        Save_ini:
    ENDIF
ENDP

```

```

PROC Save_ini:
    TRAP DELETE dataPath$+"hh2000.ini"
    CREATE dataPath$+"hh2000.ini",A,wert%
    A.wert%=ton%
    APPEND
    CLOSE
ENDP

```

```

PROC Init:
    progPath$="\System\Apps\HH2000\"
    dataPath$="C:\HH2000\"
    TRAP MKDIR dataPath$
    TRAP MKDIR dataPath$ + "bin\"
    IF bildweite%= 480 AND bildhoehe%= 160
        psionTyp$= "revo"
    ELSEIF bildweite%= 640
        IF bildhoehe%= 240
            psionTyp$= "s5"
        ELSEIF bildhoehe%= 480
            psionTyp$= "netbook"
        ENDIF
    ENDIF
    ENDIF
    IF NOT Laufwerk: (progPath$+"HH2000.app")
        REM dient nur der Feststellung des
        REM Installations-Laufwerkes (laufwerk$),
        REM in dem auch die Icon-Dateien fuer den Toolbar liegen
        ALERT ("Programmverzeichnis nicht gefunden.")
        STOP
    ENDIF

    Setup_Toolbar:

    Load_ini:
    Setup_commandWin:
    Setup_gameWin:
    ZeichneRaum:
ENDP

PROC Laufwerk: (f$)
    REM Auf welchem Laufwerk ist das Programm installiert?
    laufwerk$="C:"
    IF EXIST (laufwerk$+f$)
        RETURN KTrue%
    ELSE
        laufwerk$="D:"
        IF EXIST (laufwerk$+f$)
            RETURN KTrue%
        ELSE
            laufwerk$="E:"
            IF EXIST (laufwerk$+f$)
                RETURN KTrue%
            ELSE
                RETURN KFalse%
            ENDIF
        ENDIF
    ENDIF
    ENDIF
ENDP

```

```

REM Toolbar-Setup und -Initialisierung
*****

PROC Setup_Toolbar:
    LOCAL titel$(7)
    LOCAL diskid&                                REM IconId fuer Toolbar
    LOCAL diskoutid&                             REM IconId fuer Toolbar
    LOCAL parcelid&                             REM IconId fuer Toolbar
    LOCAL speakerid&                             REM IconId fuer Toolbar

    titel$=LEFT$(programm$,7)
    TBarInit:(titel$,bildweite%,bildhoehe%)

    IF psionTyp$="revo"
        parcelid&=gLOADBIT(laufwerk$+progPath$+"parcel_revo.mbm")
        diskid&=gLOADBIT(laufwerk$+progPath$+"diskin_revo.mbm")

        diskoutid&=gLOADBIT(laufwerk$+progPath$+"diskout_revo.mbm")
        TBarButt:("i",1,"",0,parcelid&,parcelid&,0)
        TBarButt:("s",2,"",0,diskid&,diskid&,0)
        TBarButt:("l",3,"",0,diskoutid&,diskoutid&,0)

    ELSE
        REM es ist ein S5,oder ...
        parcelid&=gLOADBIT(laufwerk$+progPath$+"parcel_s5.mbm")
        diskid&=gLOADBIT(laufwerk$+progPath$+"diskin_s5.mbm")
        diskoutid&=gLOADBIT(laufwerk$+progPath$+"diskout_s5.mbm")
        speakerid&=gLOADBIT(laufwerk$+progPath$+"speaker_s5.mbm")
        TBarButt:("i",1,"",0,parcelid&,parcelid&,0)
        TBarButt:("s",2,"",0,diskid&,diskid&,0)
        TBarButt:("l",3,"",0,diskoutid&,diskoutid&,0)
        TBarButt:("b",4,"",0,speakerid&,speakerid&,0)
        REM IF psionTyp$="netbook"          REM ... ein netBook/S7
        REM TBarButt:("H",5,"Hilfe",0,&0,&0,0)
        REM TBarButt:("a",6,"Über...",0,&0,&0,0)
        REM ENDIF
    ENDIF

    TBarShow:

ENDP

```



```

PROC Setup_gameWin:
    REM Beispielfenster
    LOCAL linkerRand%
    LOCAL obererRand%
    IF psionTyp$= "revo"
        linkerRand%= INT((480-278)/2)
        obererRand%= INT((160-159)/2)
    ELSEIF psionTyp$= "s5"
        linkerRand%= INT((640-278)/2)
        obererRand%= INT((240-159)/2)
    ELSEIF psionTyp$= "netbook"
        linkerRand%= INT((640-278)/2)
        obererRand%= INT((480-159)/2)
    ENDIF
    blindWin%=gCREATE (linkerRand%,obererRand%,278,159,1,1)
    gBORDER 3
    gameWin%=gCREATE(linkerRand%,obererRand%,278,159,1,1)
    gBORDER 3
ENDP

PROC Setup_commandWin:
    LOCAL x%, y%          REM Zeichenkoordinaten
    LOCAL xText%, yText%  REM Textkoordinaten
    LOCAL breite%, hoehe% REM Fensterbreite & -Hoehe
    LOCAL step%           REM vert. Ortsdifferenz b. Zeichnen
    LOCAL n%
    IF psionTyp$= "revo"
        x%=1 : y%=1 : step%=19
        breite%=90 : hoehe%=20
        xText%=10 : yText%=13
    ELSEIF psionTyp$= "s5"
        x%=1 : y%=1 : step%=27
        breite%=100 : hoehe%=26
        xText%=7 : yText%=17
    ELSEIF psionTyp$= "netbook"
        x%=1 : y%=120 : step%=27
        breite%=100 : hoehe%=26
        xText%=7 : yText%=17
    ENDIF
    wasnunWin%= gCREATE (x%,y%,breite%,hoehe%,1)
    gBorder 3
    gFONT KFontArialBold11&
    gAT xText%, yText% : gPRINT "WAS NUN?"
    y%=y%+5 : xText%=xText%+7
    n%=1
    DO
        y%=y%+step%
        cmdWin%(n%)=gCREATE (x%,y%,breite%,hoehe%,1)
        gBorder 3
        gFONT KFontArialBold11&
        gAT xText%, yText%
        gPRINT kommando$(n%)
        n%=n%+1
    UNTIL n%>7

```

```

ENDP

PROC Finale:
    Teile_Satz:("Hier sollte das Finale kommen ...")
    Textausgabe:
    REM      Demo_finale:
ENDP

PROC Demo_finale:
    LOCAL fenster%, fn%, fx%, fy%, fb%, fh%, t$(255)
    fb%= 110      : fh%= 30
    fx%= INT(bildweite%/2) - INT(fb%/2)
    fy%= INT(bildhoehe%/2) - INT(fh%/2)

    fenster%=gCREATE(fx%,fy%,fb%,fh%,1,1)
    gBORDER 3

    t$= "Gewonnen!"
    fx%=7      : fy%= 20
    gAT fx%, fy% : gPRINT t$
    PAUSE -40

    fn%=-4
    gTMODE 1
    DO
        PAUSE -5
        gAT fx%+fn%,fy% : gPRINT t$
        fn%=fn%+1
    UNTIL fn%>0
    gTMODE 0

    fx%=35 : fy%= 20
    gAT fx%, fy%
    gPRINT "Bye!"
    PAUSE -40
    gCLOSE fenster%
    gUSE gameWin%
ENDP

```

```

REM *** Globale Deklarationen und Variablen-initialisierung
PROC Declare_and_init:
    REM kommt von TBarLink:() / Toolbar.opo
    REM Deklaration der Variablen fuer den Programmrahmen *****
    GLOBAL programm$(20)      REM Programmname
    GLOBAL version$(20)       REM Programm-Versionsnummer
    GLOBAL a$(16)             REM Feld f. Ablage d.Event-Variabl.
    GLOBAL minit%             REM Merker für letzte Menueposition
    GLOBAL psionTyp$(7)       REM Handheld-Geraetetyp
    GLOBAL progPath$(255)     REM Programm-Pfad
    GLOBAL dataPath$(255)    REM Daten-Pfad
    GLOBAL laufwerk$(3)       REM Laufwerksbuchstabe, z.B. "C:\"
    GLOBAL thread&           REM Thread-Var. f. Aufr. Hilfedatei

    REM Deklaration der Variablen fuer das eigene Programm *****
    GLOBAL blindWin%          REM Id des Blindvorhangs
    GLOBAL gameWin%          REM Id des "Spielfeldes"
    GLOBAL cmdWin$(7)         REM Id d.Kommando(Verben)-Fenster
    GLOBAL wasnunWin%         REM Id WasNun?-Fenster
    GLOBAL diskinid&          REM IconId fuer Toolbar
    GLOBAL diskoutid&         REM IconId fuer Toolbar
    GLOBAL parcelid&          REM IconId fuer Toolbar
    GLOBAL speakerid&         REM IconId fuer Toolbar
    GLOBAL ton%               REM Merker fuer Ton an/aus
    GLOBAL beepLen%,freq%     REM Laenge der Piepser & Frequenz

    REM Raeume & Raumeigenschaften *****
    GLOBAL raum%              REM aktuelle Raumnummer
    GLOBAL raumMax%           REM max. Anz.der Raeume
    GLOBAL alterRaum%         REM Merker f.zuletzt betret. Raum
    GLOBAL geaendert%         REM Kennung,wenn Raum wg.Veraend.
                                REM neu gezeichnet werden soll
    GLOBAL tuerW%(6)          REM sichtb. Tuer linke Wand (West)
    GLOBAL tuerO%(6)          REM sichtb. Tuer rechte Wand (Ost)
    GLOBAL tuerN%(6)          REM sichtb. Tuer hint. Wand (Nord)
    GLOBAL tuerS%(6)          REM sichtb. Tuer vord. Wand (Sued)
                                REM zulaessige Werte fuer door..%:
                                REM Werte: KTrue%=-1=vorhanden,
                                REM KFalse%=0=nicht vorhanden
    GLOBAL wege%(6)           REM offene Tueren, 1=N 2=S 4=W 8=E
    GLOBAL fixo%(9)           REM Liste d. festen Objekte im Raum
    GLOBAL fixomax%           REM max. Anz. d.festen Obj.im Raum

    REM Objekte und deren Eigenschaften *****
    GLOBAL schreibtisch%,schublade%,kuli%,briefoeffner%
    GLOBAL treppe2%,treppe5%,schrank%,kaestchen%,schluessel%
    GLOBAL tuer5%,bild%,koffer%,testament%,waende%
    GLOBAL objekt$(14,20)     REM Namen der Objekte
    GLOBAL uart$(14,7)        REM unbest. Artikel des Objektes
    GLOBAL sichtbar%(14)     REM
    GLOBAL nehmbar%(14)       REM
    GLOBAL nehmbarMax%        REM max. Anz.d.nehmbaren Objekte
    GLOBAL genommen%(14)     REM
    GLOBAL oeffnungsStatus%(14) REM

```

```

GLOBAL wirt%(14)      REM Wirt d.Obj. ist Obj.Nr.xx
GLOBAL gast%(14)      REM Gast d.Obj. ist Obj.Nr.yy
GLOBAL benutzt%(14)   REM
GLOBAL imRaum%(14)    REM
GLOBAL objekteMax%    REM max.Anz.d.Obj.f.Speich.&Laden

REM Textausgabe *****

GLOBAL textzeile$(7,50) REM Textzeilen-Array f.Ausgabefenster

REM Action *****

GLOBAL kommando$(7,10) REM Name des Kommandos/Verbs,
                        REM 7 Strings a 10 Zeichen
GLOBAL gewonnen%      REM Gewonnen-Flag
LOCAL n%

REM Variableninitialisierung*****

REM Beschreibung der Tueren in den Raeumen:
tuerW%(1)=KFalse% : tuerO%(1)=KTrue%
tuerN%(1)=KTrue%  : tuerS%(1)=KFalse%
tuerW%(2)=KTrue%  : tuerO%(2)=KTrue%
tuerN%(2)=KFalse% : tuerS%(2)=KFalse%
tuerW%(3)=KTrue%  : tuerO%(3)=KFalse%
tuerN%(3)=KFalse% : tuerS%(3)=KFalse%
tuerW%(4)=KFalse% : tuerO%(4)=KFalse%
tuerN%(4)=KFalse% : tuerS%(4)=KTrue%
tuerW%(5)=KFalse% : tuerO%(5)=KTrue%
tuerN%(5)=KFalse% : tuerS%(5)=KTrue%
tuerW%(6)=KTrue%  : tuerO%(6)=KFalse%
tuerN%(6)=KFalse% : tuerS%(6)=KFalse%

REM Beschreibung der freien Wege:
wege%(1)= 1 + 0 + 0 + 8
wege%(2)= 1 + 0 + 4 + 8
wege%(3)= 0 + 0 + 4 + 0
wege%(4)= 0 + 2 + 0 + 0
wege%(5)= 0 + 2 + 0 + 0      REM Ost-Tuer ist verschlossen!
wege%(6)= 0 + 0 + 4 + 0

REM Verben 1..7
n%=0
n%=n%+1 : kommando$(n%)="umsehen"      REM Verb 1
n%=n%+1 : kommando$(n%)="sieh an"       REM Verb 2
n%=n%+1 : kommando$(n%)="nimm"          REM Verb 3
n%=n%+1 : kommando$(n%)="öffne"         REM Verb 4
n%=n%+1 : kommando$(n%)="benutze"       REM Verb 5
n%=n%+1 : kommando$(n%)="bewege"        REM Verb 6
n%=n%+1 : kommando$(n%)="klopfe an"     REM Verb 7

REM Nummern f. Objekte
testament%= 1
schluessel%= 2
kuli%= 3

```

```

briefoeffner%= 4
kaestchen%= 5
schreibtisch%= 6
schublade%= 7
treppe2%= 8
schrank%= 9
treppe5%= 10
tuer5%= 11
bild%= 12
koffer%= 13
waende%= 14
objekteMax%= 14

REM Objektnamen, unbestimmter Artikel, Aufenthaltsort
objekt$(testament%)= "Testament"
    uart$(testament%)= " ein " : imRaum%(testament%)= 6
objekt$(schluessel%)= "Schlüssel"
    uart$(schluessel%)= " einen " : imRaum%(schluessel%)= 3
objekt$(kuli%)= "Kugelschreiber"
    uart$(kuli%)= " einen " : imRaum%(kuli%)= 4
objekt$(briefoeffner%)= "Brieföffner"
    uart$(briefoeffner%)= " einen " : imRaum%(briefoeffner%)= 4
objekt$(kaestchen%)= "Kästchen"
    uart$(kaestchen%)= " ein " : imRaum%(kaestchen%)= 3
objekt$(schreibtisch%)= "Schreibtisch"
    uart$(schreibtisch%)= " einen " : imRaum%(schreibtisch%)= 4
objekt$(schublade%)= "Schublade"
    uart$(schublade%)= " eine " : imRaum%(schublade%)= 4
objekt$(treppe2%)= "Treppe"
    uart$(treppe2%)= " eine " : imRaum%(treppe2%)= 2
objekt$(schrank%)= "Hängeschränk"
    uart$(schrank%)= " einen " : imRaum%(schrank%)= 3
objekt$(treppe5%)= "Treppe"
    uart$(treppe5%)= " eine " : imRaum%(treppe5%)= 5
objekt$(tuer5%)= "Tür"
    uart$(tuer5%)= " eine " : imRaum%(tuer5%)= 5
objekt$(bild%)= "Bild"
    uart$(bild%)= " ein " : imRaum%(bild%)= 6
objekt$(koffer%)= "Koffer"
    uart$(koffer%)= " einen " : imRaum%(koffer%)= 6
REM Sonderfall:
objekt$(waende%)= "Wände" : uart$(waende%)= " "

REM Objekteigenschaften
REM alle "nicht initialisierten" Werte besitzen den Wert Null
sichtbar%(testament%)= kFalse%
sichtbar%(schluessel%)= kFalse%
sichtbar%(kuli%)= kFalse%
sichtbar%(briefoeffner%)= kTrue%
sichtbar%(schreibtisch%)= kTrue%
sichtbar%(schublade%)= kTrue%
sichtbar%(treppe2%)= kTrue%
sichtbar%(schrank%)= kTrue%
sichtbar%(kaestchen%)= kFalse%
sichtbar%(treppe5%)= kTrue%

```

```

sichtbar%(tuer5%)= kTrue%
sichtbar%(bild%)= kTrue%
sichtbar%(koffer%)= kFalse%

nehmbarMax%=5
nehmbar%(testament%)= KTrue%
nehmbar%(schluessel%)= KTrue%
nehmbar%(kuli%)= KTrue%
nehmbar%(briefoeffner%)= KTrue%
nehmbar%(kaestchen%)= KTrue%

REM Oeffnungszustand: 0= nicht zu oeffnen, 1= geschlossen,
REM 4= offen 2= geschlossen, nur mit passendem
REM "Schluessel" (Schuessel, Briefoeffner) zu oeffnen,
oeffnungsStatus%(schublade%)= 1
oeffnungsStatus%(schrank%)= 1
oeffnungsStatus%(tuer5%)= 2
oeffnungsStatus%(koffer%)= 2
oeffnungsStatus%(kaestchen%)= 1

gast%(koffer%)= testament% : wirt%(testament%)= koffer%
gast%(schublade%)= kuli% : wirt%(kuli%)= schublade%
gast%(schrank%)= kaestchen% : wirt%(kaestchen%)= schrank%
gast%(kaestchen%)= schluessel%
wirt%(schluessel%)= kaestchen%

beepLen%= 1
freq%= 333
raum%=1
alterRaum%=raum%

Main:
ENDP

```

Anhang 3

Skizzen

Die Programm-Icons

