Free!

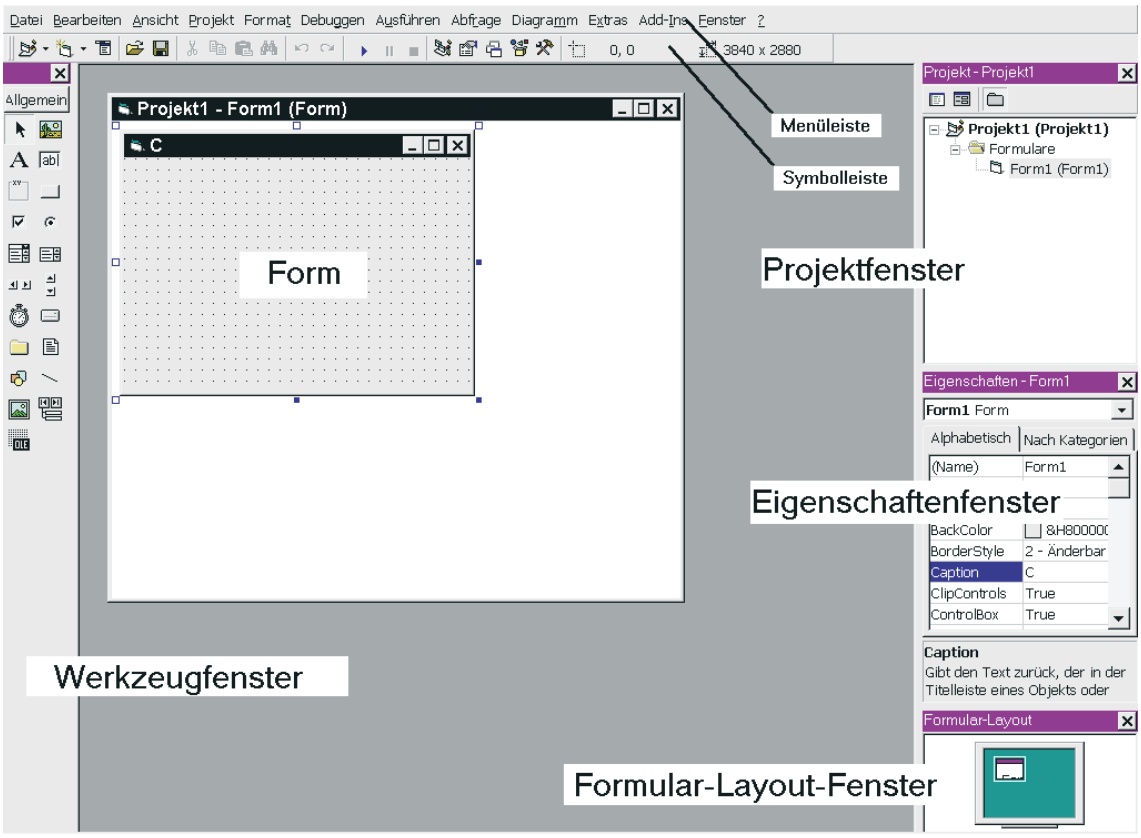
KnowWare

Special!

Einfach & verständlich

Visual Basic

für Einsteiger



Peter Hatje

www.KnowWare.de

KnowWare Special!

Acrobat Reader: Wie ...

F5/F6 öffnet/schließt die Ansicht **Lesezeichen**

Strg+F sucht

Im Menü Ansicht stellst du ein, wie die Datei gezeigt wird

STRG+0 = Ganze Seite **STRG+1** = Originalgrösse **STRG+2** = Fensterbreite

Im selben Menü kannst du folgendes einstellen:: **Einzelne Seite**, **Fortlaufend** oder **Fortlaufend - Doppelseiten** .. Probiere es aus, um die Unterschiede zu sehen.

Navigation

Pfeil Links/Rechts: eine Seite vor/zurück

Alt+ Pfeil Links/Rechts: Wie im Browser: Vorwärts/Zurück

Strg+**+** vergrößert und **Strg**+**–** verkleinert

Bestellung und Vertrieb für den Buchhandel

Bonner Pressevertrieb, Postfach 3920, D-49029 Osnabrück
Tel.: +49 (0)541 33145-20
Fax: +49 (0)541 33145-33
bestellung@knowware.de
www.knowware.de/bestellen

Autoren gesucht

Der KnowWare-Verlag sucht ständig neue Autoren. Hast du ein Thema, daß dir unter den Fingern brennt? - ein Thema, das du anderen Leuten leicht verständlich erklären kannst?

Schicke uns einfach ein paar Beispielseiten und ein vorläufiges Inhaltsverzeichnis an folgende Adresse:

lektorat@knowware.de

Wir werden uns deinen Vorschlag ansehen und dir so schnell wie möglich eine Antwort senden.

www.knowware.de

Vorbemerkung

Du hast dich entschieden, eigene Programme schreiben zu wollen? Ich finde das gut. Du befindest dich damit in einer guten Tradition, denn es ist noch nicht allzu lange her (allerdings bei dem Tempo, dass die Computerentwicklung vorlegt, sind es Jahrhunderte), dass Amateure – und da vornehmlich Jugendliche und auch Kinder – *auf Teufel komm raus* programmierten. Es war die Zeit des Commodore 64, kurz C64 genannt. Er stand in fast jedem Kinderzimmer, hatte eine weltweite Verbreitung und müsste sich vor heutigen PCs im Prinzip nicht verstecken. Alles, was PCs heute leisten, konnte der C64 auch – natürlich sehr viel langsamer. Und in seinen Arbeitsspeicher würde nicht einmal ein heutiger Maustreiber passen.

Er hatte aber einen großen Vorteil: er war billig und nach dem Einschalten (Hochbooten gab es nicht) stand sofort die Programmiermöglichkeit in Basic (**B**eginners **A**llpurpose **S**ymbolic **I**nstruction **C**ode) zur Verfügung. Das wurde von den Besitzern kräftig benutzt.

Es gibt eine ganze Reihe von Programmiersprachen. Für jeden besonderen Zweck wurde eine andere entwickelt. Grundsätzlich haben aber alle dieselbe Aufgabe: sie sollen es dem Menschen ermöglichen, sich einer Maschine verständlich zu machen. Auch wenn es manchmal nicht so aussieht: ein Computer ist dumm. Die Maschine führt brav und zuverlässig Programme aus, die in einer Form geschrieben sind, die beide, Mensch und Computer, verstehen.

Basic hat den Vorteil, dass es leicht zu erlernen ist, verhältnismäßig unkompliziert aufgebaut ist und trotzdem zu ganz passablen Ergebnissen führt.

Gerade diese Vorzüge ließen Profis die Nase rümpfen. Wenn Schüler programmieren konnten, wurde an ihrem Image der geheimnisumwitterten Kenner von Computern gekratzt. Folglich wurde Basic madig gemacht. Man muss allerdings zugeben, dass irrsinnig viel Schrott programmiert wurde.

Dieses Vorurteil gegenüber Basic wird heute immer noch gepflegt, auch wenn heimlich große Teile von Anwendungen nach dem Visual Basic – Prinzip entwickelt werden.

Die Umgebung um den Heimcomputer C64 entwickelte sich zu einer richtigen Szene. Es gab eine 'hauseigene' Zeitschrift, es gab jede Menge Zubehör in jedem Kaufhaus (bis hin zu Schnittstellen für Robotermodelle aus dem Spielwarenbereich), es gab Spiele in Hülle und Fülle und es gab junge Leute, die ihre Programme der Öffentlichkeit zur Verfügung stellten. Das war so etwas wie die heutige Shareware-Szene, allerdings öffentlicher. Programmlistings waren jedem zugänglich und Programmierkniffe und –tricks wurden jedem zugänglich gemacht. Sehr demokratisch, das Ganze.

Dann kam der PC; die Firma Commodore verschlief die Entwicklung, der C64 verschwand und merkwürdigerweise endete damit auch das private Programmieren.

Heute haben wir Programmiermöglichkeiten, von denen man damals nur träumen konnte und daher ist es umso unverständlicher, dass sich keine neuen Fan-Clubs bilden.

Vielleicht gelingt das ja über diesen Weg.

Manuskript und Programme sind mit Jugendlichen durchgearbeitet worden. Dabei wurde die Bandbreite von kompletter Unkenntnis, auch mit Windows, bis zu einigen Erfahrungen mit Basic berücksichtigt. Daher werden einige Leser das Eine oder Andere als zu ausführlich beschrieben betrachten, andere vielleicht noch mehr Grundkenntnisse benötigen.

> Ich bin ein Opfer der neuen Rechtschreibung, will sagen, ich muss ganz neu lernen. Ich hoffe, es wird mir verziehen, wenn noch nicht alles einwandfrei richtig ist.

Das Programm

Visual Basic ist ein umfassendes System zur Entwicklung von Anwendungen unter Windows. Es gibt verschiedene Ausgaben oder Editionen und verschiedene Entwicklungen oder Versionen. Derzeit liegt die Version Visual Basic 6.0 vor. Eine Edition wäre z.B. die Visual Basic Learning Edition von Visual Basic 6.0. Darauf aufbauend gibt es die Professional Edition.

Erwerb

Die Learning Edition liegt einem Buch über Visual Basic 6.0 bei. Zitat aus der Readme-Datei:

> Das Verzeichnis \VB6 enthält die Ablaufmodell-Edition von Visual Basic 6. Diese Visual Basic-Edition ist gegenüber anderen Visual Basic-Editionen in folgenden Punkten eingeschränkt:
>
> *Sie können keine ausführbaren Dateien (.exe) erstellen.*
>
> *Sie können die meisten Beispiele des Buches nachvollziehen, die Programmierung von ActiveX-Steuerelementen ist allerdings nicht möglich.*
>
> *Sie erhalten keinen Support.*
>
> *Es gibt keine Online-Hilfe.*

Auf vielen CDs von Computerzeitschriften findet man eine kostenlose CCE-Version von Visual Basic 5.0. Sie ist allerdings in Englisch. Zudem kann man durch Stöbern von Zeitschriften-CDs eine Version 4.0 finden und installieren. Schüler und Studenten können sich kostengünstig Lern-Editionen besorgen. Die Beschaffung ist zwar etwas umständlich, aber die Vertreiber von Software helfen sicherlich gerne weiter. Kostengründe können es also nicht sein, wenn Visual Basic nicht eingesetzt wird.

Dieses Heft ist so aufgebaut, dass ab Version 4.0 alle Programme entwickelt werden können. Auf die Einschränkungen der Billig-Ausgaben wird Rücksicht genommen. Alle Einschränkungen haben für Anfänger keine Bedeutung. Auch wenn davon geträumt wird, eine Internet-Anbindung zu programmieren, müssen zunächst einmal grundlegende Kenntnisse für den Alltag vorhanden sein. In der Schule wird einem auch nicht im 1. Schuljahr das Lösen von quadratischen Gleichungen beigebracht.

Visual Basic entwickelt sich immer weiter und lässt kaum Lösungsmöglichkeiten aus. Mit hoch spezialisierten Programmiersprachen, die beispielsweise für komplexe Grafikoperationen eingesetzt werden, kann Visual Basic natürlich nicht konkurrieren, für alltägliche Grafiklösungen reicht es aber allemal – wie wir sehen werden.

Installation

Hast du deine Version von Visual Basic erworben, muss das Programm auf der Festplatte installiert werden. Zunächst solltest du dir überlegen, in welchem Verzeichnis du Visual Basic installieren willst. Da ich nicht weiß, wie du deine Dateiordner organisiert hast, schlage ich vor, die Vorgaben des Installationsprogramms zu übernehmen und das Programm aus dem Startmenü heraus zu starten.

Wer sich gut mit Windows auskennt, kann sich ja einen eigenen Ordner für Visual Basic anlegen, wobei die Pfade möglichst kurz gehalten werden sollten, damit später beim Einsetzen von Pfaden keine ellenlangen Dateinamen eingesetzt werden müssen.

Du startest die Installation, indem du die CD einlegst, START | AUSFÜHREN | DURCHSUCHEN wählst und im CD-Laufwerk VB6\SETUP.EXE aktivierst. Durchweg kannst du einfach auf WEITER klicken, die Lizenzbedingungen bestätigen und den gewünschten Pfad übernehmen oder einen eigenen anlegen. Zwischendrin wird der Rechner einmal neu gestartet, wobei du die CD nicht aus dem Laufwerk entfernen darfst.

Schließlich erscheint die Meldung, dass die Installation erfolgreich abgeschlossen wurde.

Im Verlauf dieses Heftes werden verschiedene Programme entwickelt, die irgendwo gespeichert werden müssen. Daher solltest du dir wegen der besseren Übersichtlichkeit einen Ordner anlegen, in dem du die Dateien speicherst. Da das Register VORHANDEN im Startdialog des Programms als Standard auf den Ordner VB98 verweist, passt hier ein Ordner PROGRAMME hin.

Starten

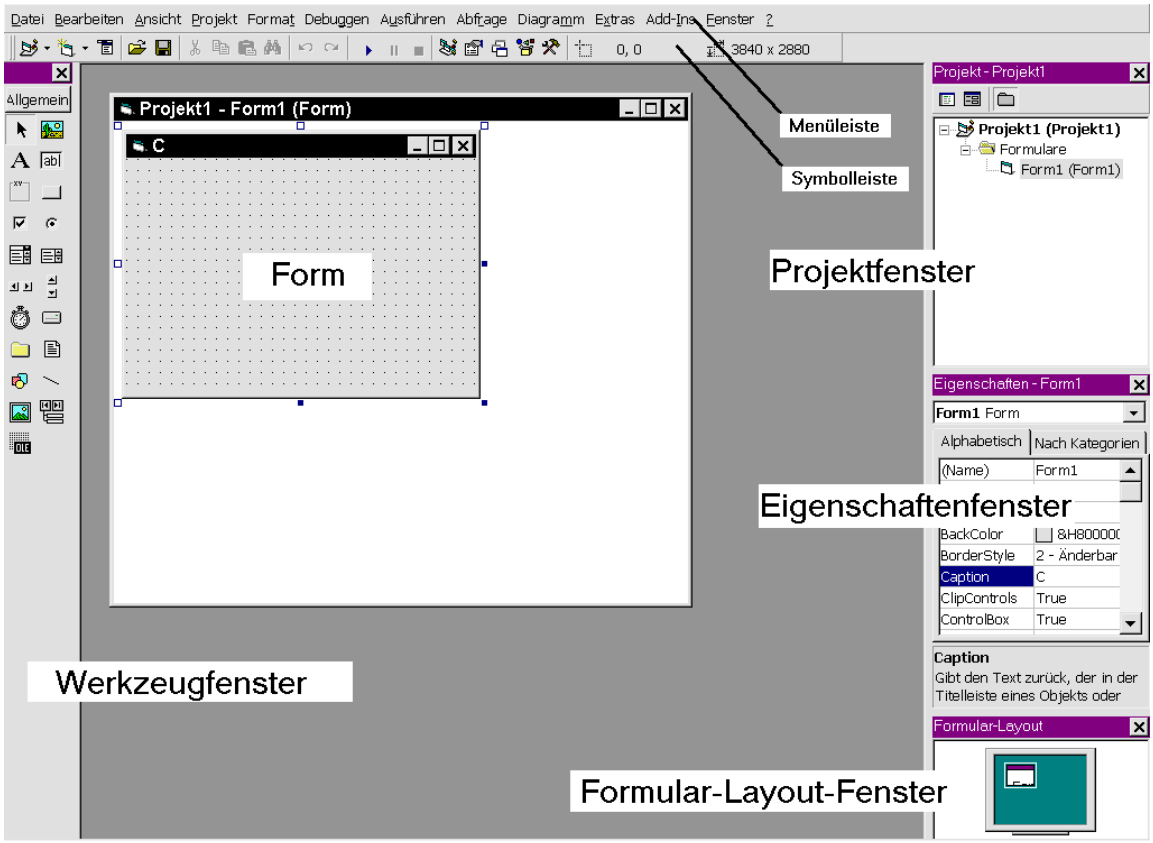
Visual Basic kann auf unterschiedliche Weise gestartet werden. Der üblichste Weg ist wohl: STARTMENÜ | PROGRAMME | MICROSOFT VISUAL BASIC 6.0 | MICROSOFT VISUAL BASIC 6.0.

Inmitten von vielen Fenstern wartet das Dialogfeld zur Auswahl der Projekte auf dich. Vorgewählt ist STANDARD-EXE im Register NEU. Außerdem gibt es die Register VORHANDEN und AKTUELL. Im Register VORHANDEN werden Verzeichnisse geöffnet, in denen sich Projekte befinden; im Register AKTUELL sind alle schon einmal geöffneten Programme abgelegt. Hier findest du deine neu entwickelten Programme wieder, nachdem du sie gespeichert hast.

Bei dir sieht die Sache vermutlich so ähnlich aus wie in der nebenstehenden Abbildung. Hier markierst du STANDARD.EXE und wählst ÖFFNEN, worauf das Programm loslegt.

Einrichten

Jetzt hast du Visual Basic in aller Pracht vor dir. Als erstes solltest du das PROJEKT-Fenster verkleinern, indem du mit der Maus an den unteren Rand fährst (der Mauszeiger ändert seine Form) und den Fensterrand nach oben ziehst; dadurch vergrößert sich das EIGENSCHAFTEN-Fenster – und das brauchen wir ständig. Also sollten wir es auch unbedingt etwas breiter machen. Das FORMULAR-LAYOUT-Fenster dagegen klickst du weg – das brauchen wir vorläufig nicht.



Die FORM ist die Bedienoberfläche des späteren Programms. Je nach den Bedürfnissen kann seine Größe einfach verändert werden, indem du die sogenannten ANFASSER, also die kleinen Kästchen an der FORM, mit der Maus greifst, d.h. anklickst und dann bei gehaltener Maustaste an ihnen ziehst, wodurch sich die FORM vergrößert bzw. verkleinert. Die FORM ist im FORMULAR-DESIGNER-Fenster angelegt – dessen Titelleiste zeigt also ganz logisch den Namen PROJEKT1 – FORM1 (FORM) an.

Es gibt noch eine ganze Reihe individueller Einstellmöglichkeiten, die du vornehmlich unter EXTRAS | OPTIONEN findest.

Beschreibung

Sehen wir uns nun die einzelnen Fenster an – mit ihrer Hilfe gestaltet du das Programm, das du entwickeln möchtest.

Projekt-Fenster

Im PROJEKT-FENSTER werden alle Zutaten zu einem Projekt gesammelt. Dazu gehört standardmäßig eine Form. Es können noch andere Formen oder Module dazukommen.

In der Symbolleiste des Projekt-Fensters gibt es drei Schaltflächen:
CODE ANZEIGEN,
OBJEKT ANZEIGEN,
ORDNER WECHSELN.

Über diese Schaltflächen kannst du vom Objekt (das ist hier die Form mit dem Namen FORM1) zum CODE-Fenster wechseln. Das Code-Fenster zeigt dir natürlich noch nichts an, denn wir haben ja noch keinen Code. Genaueres über das Cofe-Fenster erfährst du auf Seite 14.

> Den Begriff *Code* kennst du sicher. Damit wird meistens Geheimschrift oder auch Chiffre verbunden. So ähnlich sieht der Code für Visual Basic auch für einen Laien aus. Dieses Heft soll dafür sorgen, dass Visual Basic-Code für dich keine Geheimschrift mehr ist.

Eigenschaften-Fenster

Menschen gibt es in vielen Formen: groß oder klein, dick oder dünn, mit verschiedenen Namen usw. All das sind *Eigenschaften*. Jedes Objekt in Visual Basic hat natürlich auch seine Eigenschaften, wobei eine Linie weniger Eigenschaften hat als ein Kreis. Eine Linie hat einen Start- und einen Endpunkt, eine bestimmte Dicke, unterschiedliche Farben. Ein Kreis hat einen bestimmten Radius, kann mit Farben oder Mustern ausgefüllt sein usw.

Diese und andere Eigenschaften findest du im EIGENSCHAFTEN-Fenster. Oben im Fenster siehst du die Objektliste, in der im Moment nur die Form zu finden ist. Darunter sind zwei Register, in denen du nach Geschmack die Anzeige der Eigenschaften einstellen kannst. Die linke Spalte führt die EIGENSCHAFTEN auf (leider nur in Englisch) und die rechte Spalte die dazugehörigen WERTE. Diese Werte können direkt im Eigenschaften-Fenster verändert werden.

1. Du klickst auf BACKCOLOR, also auf die Hintergrundfarbe,
2. ... dann auf den Pfeil in der Werteliste,
3. ... dann auf PALETTE
4. ... und endlich auf eine Farbe

   Visual Basic führt die Veränderungen jeweils sofort aus – fertig!
5. Nun wählst du AUSFÜHREN | STARTEN oder drückst F5 bzw. die entsprechende Schaltfläche in der Symbolleiste.

   > Fährst du mit dem Mauszeiger über eine Schaltfläche, erscheint nach einem kurzen Augenblick ein sogenannter Tooltip.

   Was du jetzt siehst, ist die Bedienoberfläche während der Laufzeit, d.h. das Programm läuft jetzt. Arbeitest du an der Bedienoberfläche, bist du also in der Entwicklungsphase.
6. Du aktivierst AUSFÜHREN | BEENDEN bzw. drückst auf die SCHLIEßEN-Schaltfläche der Form oder die entsprechende Schaltfläche in der Symbolleiste.

Mit der dritten Schaltfläche in diesem Zusammenhang kannst du den Programmablauf unterbrechen, um Überprüfungen des Programms vorzunehmen.

Die einzelnen Eigenschaften werden im Laufe der Programmentwicklung besprochen. Im übrigen ist unter dem EIGENSCHAFTEN-Fenster eine Kurzbeschreibung zu sehen, ansonsten gilt: Lernen durch Ausprobieren.

Werkzeug-Fenster

Das Werkzeug-Fenster beinhaltet die sogenannten Werkzeuge oder *Tools*. Damit werden vorgefertigte Steuerelemente für die Bedienoberfläche zur Verfügung gestellt.

Auch hier erschient wieder ein Tooltip mit der (leider) englischen Bezeichnung.

	Tooltip	Deutsch
	ZEIGER	Zeiger
	PICTUREBOX	Bildfeld
	LABEL	Bezeichnungsfeld
	TEXTBOX	Textfeld
	FRAME	Rahmen
	COMMAND-BUTTON	Befehlsschaltfläche
	CHECKBOX	Kontrollkästchen
	OPTION-BUTTON	Optionsfeld
	COMBOBOX	Kombinationsfeld
	LISTBOX	Dateiliste
	HSCROLLBAR	Horizontale Bildlaufleiste
	VSCROLLBAR	Vertikale Bildlaufleiste
	TIMER	Zeitgeber
	DRIVELIST-BOX	Laufwerksfeld
	DIRLISTBOX	Verzeichnislistenfeld
	FILELISTBOX	Dateilistenfeld
	SHAPES	Figuren
	LINE	Linie
	IMAGE	Bildanzeigefeld
	DATA	Datensteuerelement
	OLE	Datenverknüpfung über OLE

Dieses Fenster kann durch unzählige Werkzeuge für jeden erdenklichen Zweck ergänzt werden. Das machst du so: du wählst DATEI | NEUES PROJEKT, verneinst die Abfrage nach Speichern und wählst VB ABLAUFMODELL-EDITION-STEUERELEMENTE. Nun werden zusätzliche Steuerelemente geladen, die wir in diesem Heft aber nicht verwenden werden.

Über DATEI | NEUES PROJEKT – und Verneinung der Abfrage nach Speichern – aktivierst du die STANDARD.EXE und kehrst so wieder in den Ausgangszustand zurück.

Wie bekommst du nun die Schaltflächen auf die Form? Wie immer bei Visual Basic gibt es mehrere Möglichkeiten. Du kannst:

- eine Schaltfläche anklicken, den Mauszeiger in die Form setzen, die linke Maustaste drücken und bei gedrückter Maustaste ein Rechteck aufziehen. Fertig!
- die Schaltfläche im Werkzeugfenster doppelklicken. Fertig!

Alle Schaltflächen haben Anfasser, mit denen du die Größe der Schaltfläche verändern kannst; die Schaltfläche kann bei gedrückter linker Maustaste auch verschoben werden.

Es kann jeweils nur eine Schaltfläche aktiv sein – welche, das ist an den Anfassern zu erkennen. Im Eigenschaften-Fenster werden die jeweiligen Eigenschaften für die Schaltfläche automatisch angezeigt. Willst du die Eigenschaften für eine Schaltfläche ändern, kannst du sie in der Form anklicken und sie aktivieren oder in der Auswahlliste des Eigenschaften-Fensters anwählen.

Form-Fenster

Das FORM-Fenster wird meistens nur Form oder *Formular* genannt. Hier findet die Gestaltung deiner Anwendung statt. Auf der Form, oder man sagt auch in der Form, werden die Objekte angeordnet, die in ihrer Kombination die Bedienoberfläche ergeben. Beim Start des Programms ist dann nur noch die Form mit ihren Elementen aktiv und wartet darauf, dass etwas geschieht.

In der offiziellen Version von Visual Basic 6 gibt es die Möglichkeit, eine sogenannte EXE-Datei zu erstellen. Wenn du genau hinsiehst, kannst du im Datei-Menü ein abgeblendetes Menü namens PROJEKT1-EXE ERSTELLEN erkennen. Damit wird ein ausführbares Programm entwickelt, dass du auch auf Rechnern ausführen kannst, auf denen kein Visual Basic installiert ist. Diese Möglichkeit ist auf dieser Testversion leider nicht vorhanden.

Die Form ist in das FORMULAR-DESIGNER-Fenster eingebettet. In der Titelleiste der Form siehst du die Beschriftung (Caption) der Form. Die änderst du, indem du im Eigenschaften-Fenster die Eigenschaft Caption wählst und einen anderen Wert eingibst. Etwas verwirrend ist, dass die Form auch den Namen Form1 hat. Das kannst du im Eigenschaften-Fenster bei der Eigenschaft Name sehen. Den namen kannst du natürlich ändern. Unter diesem namen wird die Form im Programm angesprochen.

Wie du siehst, übernimmt Visual Basic deine Eingaben sofort in die Titelleiste der Form. Hier siehst du also auch, wie einfach es ist, Eigenschaften zu benutzen: du musst nur in der Eigenschaftenliste eine Eigenschaft auswählen und in der Werteliste einen neuen Wert eintragen.

Die Eigenschaft CAPTION gibt es für die meisten Steuerelemente. Übersetzt heißt das in etwa Überschrift, Beschriftung.

Die Form hat ein Punkteraster, das während der Laufzeit verschwindet. Es ist nur dazu da, die Anordnung der Steuerelemente ziemlich genau vornehmen zu können.

Willst du die Rastereinstellungen verändern, tust du das über EXTRAS | OPTIONEN | ALLGEMEIN. Wie du siehst, kannst du hier jede Menge Einstellungen vornehmen. Solange du dich nicht gut genug auskennst, solltest du die Standard-Einstellungen beibehalten.

Hier folgt nun eine Liste der Eigenschaften:

Eigenschaft	Beschreibung
NAME	Name der Form
BORDERSTYLE	Rahmentyp der Form
BACKCOLOR	Hintergrundfarbe
FORECOLOR	Schriftfarbe
CONTROLBOX	Systemmenüfeld (in der Titelleiste links), sichtbar, unsichtbar.
FONT	Schriftarten in Formularen.
ICON	Legt das Symbol fest, das angezeigt wird, wenn das Formulars minimiert wird.
MAXBUTTON	Legt fest, ob die Schaltfläche Maximieren beim Programmablauf angezeigt wird oder nicht.
MINBUTTON	Legt fest, ob die Schaltfläche Minimieren beim Programmablauf angezeigt wird oder nicht.
STARTUPPOSITION	Legt die Position des Formulars fest.
WINDOWSTATE	Legt fest, ob das Formular maximiert, minimiert oder normal angezeigt wird.

Basic

Es gibt Konzepte, die in jedem Basic-Dialekt gelten, also nicht nur in Visual Basic. Einige davon sollen an dieser Stelle besprochen werden.

Code

Ein Code besteht zu einem großen Teil aus Anweisungen, die man auch Befehle nennt. Der Computer wird angewiesen, bestimmte Aktionen vorzunehmen. Anweisungen werden nacheinander ausgeführt. Das üblichste Beispiel ist immer wieder die Anweisung **`Print "Hallo Welt!"`**, die dazu führt, dass der Computer 'Hallo Welt' auf die Form schreibt. `Print` ist der Befehl., **`"Hallo Welt!"`** ein Argument. Der Computer 'versteht' den Code aber nur, wenn er ohne Fehler ist – jeder Schreibfehler wird missverstanden. Deshalb muss man bei den Anweisungen genau aufpassen. Das Argument, eine Variable, kann so fehlerhaft sein wie es will. Alles, was innerhalb der Anführungszeichen steht, wird genauso wiedergegeben, also auch mit sämtlichen Schreibfehlern. Zum Glück hilft Visual Basic nach Kräften, Fehler zu vermeiden.

Variablen

Jede Programmiersprache lebt von der Vielfalt der möglichen Variablen. Eine Variable ist ein Wert mit veränderlicher Größe, ein Platzhalter, in den Werte eingesetzt werden können – gewissermaßen ein Schubfach mit einem Namen in einem Schrank, in das man etwas hineinlegt, was man dann bei Bedarf wieder herausholt.

Sehm wir uns das obige Beispiel genauer an:

```
1. Satz = "Hallo Welt!"
2. Print Satz
```

In Zeile 1 legst du den Wert **`"Hallo Welt!"`** in ein Schubfach mit dem Namen **`Satz`**. In Zeile 2 gibst du folgende Anweisung: *Schreibe den Inhalt des Schubfaches 'Satz'* oder auf computerisch: *schreibe die Variable 'Satz'*. Änderst du den Inhalt, wird der alte Inhalt schlicht überschrieben.

```
1. Satz = "Hallo Welt!"
2. Print Satz
3. Satz = "Guten Tag, schöne
   Welt!"
4. Print Satz
```

Der Computer würde jetzt beide Inhalte untereinander auf den Bildschirm schreiben. Wie das gemacht wird, dazu kommen wir später.

Für fast jeden Zweck gibt es unterschiedliche Variablentypen. Damit wird auf den jeweiligen Inhalt einer Variablen Rücksicht genommen. Es ist nicht egal, ob der Inhalt, bzw. der Wert einer Variablen eine Zahl, ein Wort oder ein Geldbetrag ist. Die Unterschiede hängen vornehmlich mit dem Speicherbedarf und damit mit der Rechengeschwindigkeit zusammen.

Der Einsatz der Variablen wird jeweils in den Programmprojekten besprochen.

Variablentyp	Inhalt (Wert)	Wertebereich
BYTE	Ganzzahl	0 - 255
INTEGER	Ganzzahl	-32.768 – 32.767
LONG	Fließkomma	-2.147.483.648 – 2.147.483.647
SINGLE	Fließkomma	riesengroße Werte
DOUBLE	Fließkomma	noch größere Werte
CURENCY	Währung	-922.337.203.685.477, 5808 - + wie bei -
DATE	Datum/Zeit	1. Januar 100 – 31. Dezember 9999
STRING	Text	Ca. 65.500 Zeichen
VARIANT	alle	variabel
BOOLEAN	Boole	True oder False (-1 oder 0)

Variablen deklarieren

Variablen benötigen einen Namen, und dieser Name muss grundsätzlich mit einem Buchstaben beginnen. Die Zeichen können aus Buchstaben, Ziffern und dem Unterstrich (_) bestehen. Damit sind eindeutige Variablennamen möglich, z.B. **`Zeugnis_Mathe`**, **`Einnahme_Verkauf`**, **`Einnahme1`** o.ä.

Es ist ein guter Brauch, Variablen zu deklarieren, d.h. bei Visual Basic anzumelden. Dazu wird die Anweisung **`Dim`** verwendet. z.B.:

```
Dim Satz As String
```

Damit bekommt die Variable den Namen **`Satz`** und Visual Basic wird mitgeteilt, dass der Wert vom Typ *Zeichenkette* (STRING*)* sein muss. Visual Basic wird zukünftig mit einer Fehlermeldung meckern, wenn der Wert der Variablen nicht stimmt – z.B. wenn du eine Zahl benutzt.

Willst du sichern, dass du nicht vergisst, Variablen zu deklarieren, kannst du Visual Basic dazu veranlassen, dich energisch daran zu erinnern. Dazu wählst du EXTRAS | OPTIONEN und das Registerblatt EDITOR. Dort aktivierst du die Option VARIABLENDEKLARATION ERFORDERLICH, wählst DATEI | NEUES PROJEKT – das Speichern verneinst du – und wählst endlich STANDARD.EXE. Der Neustart ist nötig, damit die Anweisung OPTION EXPLICIT aktiviert wird, die dafür da ist, die Variablendeklaration zu erzwingen.

Der Wert dieser Anweisung liegt vor allem darin, dass Tippfehler in Variablennamen von Visual Basic bemerkt werden. Nehmen wir an, du hast die Variable **`Satz`** deklariert. Im Verlauf der Programmierung willst du die Variable benutzen und schreibst irrtümlich z.B. **`Sazz`**. Für den Computer ist das eine neue Variable, die nicht deklariert ist – weswegen er sich entsprechend meldet. Das ist zwar manchmal lästig, bei umfangreicheren Programmen aber sehr hilfreich. Außerdem zeigt dir Visual Basic auch gleich an, um welche Variable es sich handelt – nämlich die farblich unterlegte.

Geltungsbereich

Wichtig sind auch Geltungsbereich und -dauer der Variablen. Das erläutern wir später konkret an Beispielen.

Variablenwerte

Das Gleichheitszeichen hat keine mathematische Bedeutung, sondern ist ein Zuweisungszeichen:

```
Dim Satz As String
Dim Zahl As Integer
Dim Grosse_Zahl As Double
Satz = "Hallo Welt, mir geht es gut!"
Zahl = 123
Grosse_Zahl = 23632.143
```

Es wird der Variablen ein Wert zugewiesen, wobei eine Zeichenkette (String) auch ein Wert ist.

> Zeichenketten müssen grundsätzlich in Anführungszeichen gesetzt werden. Dezimalzahlen müssen grundsätzlich mit einem Punkt (.) und nicht mit einem Komma (,) geschrieben werden.

Konstanten

Es tauchen in Programmen immer wieder Zahlenwerte auf, die sich nicht verändern, die also konstant sind. Beispiele sind der Mehrwertsteuersatz von 16% oder die Zahl Pi bei mathematischen Berechnungen. Pi hat mit genügender Genauigkeit den Wert 3, 1415926546. Solche Werte werden in benutzerdefinierten Konstanten mit der Anweisung CONST angelegt.

```
Const Pi As Single = 3.1415926546
```

Im Gegensatz zu Variablen lässt sich der Wert einer Konstanten also nicht ändern – ansonsten gelten für Konstanten gleiche Bedingungen wie für Variablen.

Visual Basic stellt eine Unzahl von systemdefinierten Konstanten zur Verfügung. Diese Konstanten liegen fertig vor und müssen nicht festgelegt werden. Wir werden sie in den Programmen verwenden.

Operatoren

Standardoperatoren

Die Standardoperatoren kennst du aus der Schule. Es sind die Zeichen für Plus, Minus, Mal und geteilt durch (+, -, *, /). Diese Zeichen kannst du in Visual Basic genauso einsetzen, wie in der Schule. Wenn du ausrechnen willst, wieviel 236 * 14 / 536 ist, nimmst du deinen Taschenrechner oder Visual Basic. Du doppelklickst auf die Form, worauf sich das Codefenster öffnet. Hier sind grundsätzlich die erste und die letzte Zeile vorgegeben:

```
Private Sub Form_Load()
End Sub
```

Du setzt selber die folgenden Zwischenzeilen dort ein, wo der Textcursor bereits blinkt:

```
2. Show
3. Print 236 * 14 / 536
```

> Wie du siehst, habe ich vor jede Zeile eine Nummer gesetzt. Das dient nur der Orientierung, darf aber keinesfalls bei der eigentlichen Programmierung Anwendung finden.
> Im übrigen befassen wir uns auf Seite 14 genauer mir dem Aufbau von Prozeduren.

Zeile 2 ist die Anweisung für ANZEIGEN; Zeile 3 die Anweisung, die Werte auszurechnen und in die Form zu schreiben.

Drückst du nun auf F5 oder wählst AUSFÜHREN | STARTEN, erscheint in der Form der Wert 6,16417910447761.

Abschließend drückst du auf die SCHLIEßEN-Schaltfläche der Form oder auf ALT + F4 bzw. wählst AUSFÜHREN | BEENDEN.

Vergleichsoperatoren

Wie der Name schon sagt, vergleichen diese Operatoren zwei Werte miteinander. Ist ein Wert kleiner als ein anderer Wert, ist er größer oder ist er gleich? Manchmal ist es wichtig zu wissen, ob ein Wert ungleich einem anderen ist.

In Programmen finden meistens in diesem Zusammenhang Fragestellungen statt; wenn Wert1 größer ist als Wert2, dann ....

Im Computer wird die Frage immer nur daraufhin überprüft, ob das Ergebnis wahr (TRUE) oder falsch (FALSE) ist. Die Werte können auch Wörter sein. `Ist Erwin < Klaus` bezieht sich nicht die Körpergröße, sondern das Programm würde prüfen, ob Erwin mit dem Anfangsbuchstaben E im Alphabet vor Klaus mit dem Anfangsbuchstaben K liegt. In diesem Fall wäre das Ergebnis wahr (true)

Operator	Beispiel	Bedeutung
=	`Wert1 = Wert2`	Wert1 ist gleich Wert2
<>	`Wert1 <> Wert2`	Wert1 ist ungleich Wert2
>	`Wert1 > Wert2`	Wert1 ist größer als Wert2
<	`Wert1 < Wert2`	Wert1 ist kleiner als Wert2
>=	`Wert1 <= Wert2`	Wert1 ist größer als oder gleich Wert2
<=	`Wert1 <= Wert2`	Wert1 ist kleiner als oder gleich Wert2

Logische Operatoren

Manchmal sollen Programmabläufe davon abhängig sein, dass mehrere Werte gleichzeitig zutreffen. Wenn Wert1 und Wert2 und Wert3 oder Wert4 zutreffen, dann .... Das kann Visual Basic natürlich auch. Dafür werden logische Operatoren verwendet. Sie verknüpfen die Werte. Nach der Verknüpfung wird wieder geprüft, ob die Gesamtaussage zutrifft, wahr ist oder nicht zutrifft also falsch ist.

Operator	Beispiel	Bedeutung
AND	`Wert1 AND Wert2`	beide Werte sind wahr, treffen zu
OR	`Wert1 OR Wert2`	ein Wert von beiden ist wahr, trifft zu
NOT	`NOT Wert1`	der Wert1 trifft nicht zu

Es gibt noch mehr logische Operatoren, die wir aus Platzgründen hier nicht erwähnen.

Funktionen

Es tauchte der Begriff Anweisung auf. Ein Basic-Programm besteht größtenteils aus Anweisungen, also Befehlen, die der Computer ausführt. Es gibt aber auch Funktionen.

Numerische Funktionen

Eine häufig verwendete Funktion ist **Len** – vom englischen Wort *Length*. Sie ermittelt die Länge einer Zeichenkette: du doppelklickst in die Form, löschst eventuellen Code und setzt den Code für PROGRAMM2 ein:

```
1. Private Sub Form_Load()
2.    Show
3.    Dim Zeichenkette As String
4.    Dim länge As Integer
5.    Zeichenkette = "Es ist ein
   schöner Tag!"
6.    länge = Len(Zeichenkette)
7.    Print länge
8. End Sub
```

Schließlich drückst du auf F5.

In Zeile 6 wird die Funktion **Len** eingesetzt, und der Funktionswert wird der Variablen **länge** übergeben. Als Ergebnis erhältst du 23. Hier werden auch die Leerzeichen gezählt.

Funktion	Beispiel	Bedeutung
SQR	`Sqr(25)`	es wird die Quadratwurzel ermittelt
INT	`Int(25.1234)`	es wird der nächstkleinere ganzzahlige Wert ermittelt (bei positiven Zahlen werden einfach die Nachkommastellen abgeschnitten)
FIX	`Fix(25.1234)`	entfernt die Nachkommastellen
SIN	`Sin(90)`	Der Sinus eines Winkels wird ermittelt

Zeichenkettenfunktionen

Grundsätzlich gilt, dass alles, was zwischen Anführungszeichen (Gänsefüßchen) steht, eine Zeichenkette ist. **"Erwin ist ein guter Schüler."** oder **"1 Tor hat zum Sieg gereicht."** ist eine Zeichenkette. Du kannst alles in Anführungszeichen einschließen, auch Sonderzeichen; z.B. **"$%231+"**. Visual Basic interessiert nur die Länge der Zeichenkette.

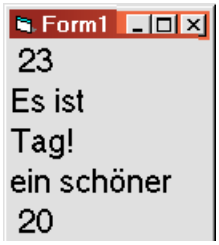
Zeichenketten kann man manipulieren. Dafür stehen etliche Funktionen zur Verfügung.

Funktion	Beispiel	Bedeutung
LEFT	`Zeichenkette = Left("Es ist ein schöner Tag!",6)`	gibt die Zeichen bis zur 6. Position von links aus
RIGHT	`Zeichenkette = Right("Es ist ein schöner Tag", 6)`	gibt die Zeichen bis zur 6. Position von rechts aus
MID	`Zeichenkette = Mid("Es ist ein schöner Tag!", 8, 3)`	gibt 3 Zeichen von der 8. Position von links aus
INSTR	`Zeichenkette = Instr(1, "Es ist ein schöner Tag!", "Tag")`	gibt die Position von links an, an der der Suchtext beginnt

Als Beispiel setzt du nun in den Code aus PROGRAMM2 zwischen den Zeilen 7 und 8 folgende Zeilen ein und erhältst so unser PROGRAMM3:

```
Print Left(Zeichenkette,6)
Print Right(Zeichenkette,4)
Print Mid(Zeichenkette,8,11)
Print Instr(1, „Es ist ein schöner Tag!“,“Tag“)
```

Vergiss nicht den abschließenden Druck auf F5.



In der Abbildung hast du die Zeichenkettenmanipulationen untereinander. Im Anhang ist eine Prozedur *Umkehren* vorhanden, die dir zeigt, wie man mit den Zeichenketten-Funktionen eine Zeichenkette von hinten nach vorn umkehren kann.

Visual Basic

Visual Basic umfasst die Grundlagen von Basic sowie umfangreiche Neuerungen. Mit dem klassischen Basic ist Visual Basic überhaupt nicht mehr zu vergleichen, obwohl im Rahmen dieses Heftes die Unterschiede noch nicht allzu deutlich zu erkennen sind. Das klassische Basic war sozusagen ein Programmwurm, der vorn anfing und bis zum Ende abgearbeitet wurde. Visual Basic dagegen ist ereignis- und objektorientiert und setzt eigentlich die Anwendung der Maus voraus.

Was heißt objektorientiert? Visual Basic versteht seine Elemente (Fenster, Schaltflächen, Listen usw.) als *Objekte*, die Eigenschaften haben und denen man bestimmte Methoden und Ereignisse zuordnen kann.

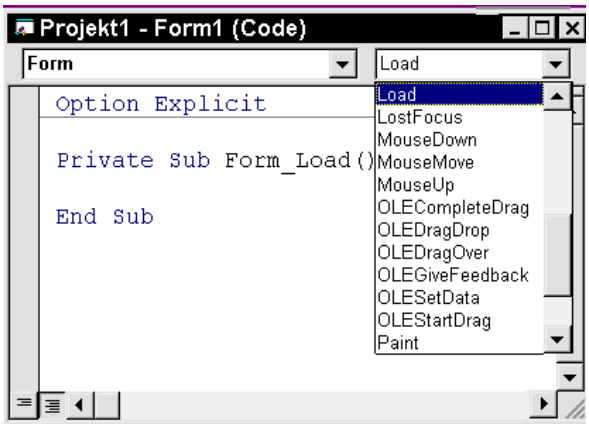
Was heißt ereignisorientiert? Programme unter Visual Basic warten auf Ereignisse. Du kannst ein Programm starten – und es passiert nichts. Klickst du eine Schaltfläche an, geschieht irgend etwas, aber was, das muss programmiert werden.

Eigenschaften

Eigenschaften wurden schon besprochen. Sie sind so wichtig, dass es dafür ein eigenes Fenster gibt. Es gibt aber auch Eigenschaften, die nicht zur Entwicklungszeit eingestellt werden können, sondern im Programm aufgebaut werden müssen.

Ereignisse

Jedes Objekt hat unterschiedlich viel Möglichkeiten, auf Ereignisse zu reagieren. Ein häufiges Ereignis ist z.B. der Klick mit der Maus auf eine Schaltfläche. Die möglichen Ereignisse werden von Visual Basic in einer Liste dargestellt, die du mit einem Klick auf den kleinen Pfeil rechts neben LOAD öffnest.



Hier siehst du eine lange Liste mit allen Ereignissen, die dem Objekt FORM zugeordnet sind. Unmittelbar ist das Ereignis LOAD markiert. Markiert werden alle eingesetzten Ereignisse. Zum Glück braucht man meist nur wenige.

Methoden

Methoden sehen den Eigenschaften ähnlich, bedeuten aber die Anweisungen an ein Objekt, sich auf eine bestimmte Weise zu verhalten, z.B. eine bestimmte Position auf der Form einzunehmen oder einer Liste neue Elemente hinzuzufügen.

Hilfe-Dateien zu diesem Heft findest du auf der Homepage. Suche die Seite mit der Heftbeschreibung. Dort gibt es einen Link zu der Datei, die du kostenlos downloaden kannst.

Prozeduren

Um all das leisten zu können, sind Visual Basic-Programme in sogenannte Prozeduren eingeteilt. Sehen wir uns das an zwei Beispielen an:

```
Private Sub Form_Load()
    (Anweisungen)
End Sub
```

Eine Prozedur wird durch das Schlüsselwort **Sub** eingeleitet. Das Schlüsselwort **Private** zeigt an, dass diese Prozedur von anderen Prozeduren aufgerufen werden kann. Alles, was zwischen **Sub** und **End Sub** steht, wird vom Computer abgearbeitet, worauf die Prozedur dann wieder verlassen wird. Die oben genannte Sub verweist auf das Objekt Form und durch einen Unterstrich getrennt auf das Ereignis **Load**. Alles, was zwischen der Startzeile und der Endzeile eingetragen wird, geschieht beim Laden der Form, also im allgemeinen, wenn man das Programm startet.

```
Private Sub Form_Click()
    (Anweisungen)
End Sub
```

Wird das Ereignis Click gewählt, kann Code in die Prozedur geschrieben werden, der ausgeführt wird, wenn auf die Form geklickt wird.

Der Code innerhalb einer Ereignis-Prozedur ist durch die beiden Zeilen von allen anderen Prozeduren abgeschottet. Auch Variablen in einer Prozedur sind grundsätzlich nur aktiviert, wenn die Prozedur aufgerufen wird – und ihre Werte sind wieder verschwunden, wenn die Prozedur verlassen wird. Daher kann man auch in Prozeduren die gleichen Variablennamen verwenden, denn eine Prozedur kennt die andere nicht.

Funktionen

In diesem Zusammenhang sind selbstentwickelte Funktionen gemeint, nicht die Funktionen, die als Basic-Funktionen bekannt sind – und die wir auf Seite 13 besprachen – wie z.B. die Funktion SIN(WINKEL).

Über diese mathematische Funktion wird der Sinus eines Winkels berechnet.

Funktionen dieser Art sind in Basic eingebaut. Wir wollen jetzt aber über Funktionen sprechen, die wie Prozeduren aufgebaut sind, aber einen Wert an das Programm zurückgeben. Wenn in einem Programm immer wieder Mehrwertsteuern berechnet werden, legt man für diesen Zweck eine Funktion an und übergibt dieser die entsprechenden Beträge. Die Funktion gibt den berechneten Betrag zurück.

```
Private Function
MehrwertSteuer(Betrag)
   Berechnungen
   MehrwertSteuer =
   ErgebnisBerechnungen
End Function
```

Selbstentwickelte Funktionen verhalten sich also im Grunde genauso wie die integrierten Funktionen. Ein Beispiel für die Verwendung von Funktionen ist im Anhang auf Seite 55 beschrieben.

Das erste Programm

Die Voraussetzungen für einen ersten Versuch sind geschaffen. Visual Basic stellt dir eine Form und ein Werkzeugfenster zur Verfügung, und die grundlegenden Kenntnisse hast du. Fehlt nur noch eine Idee.

Ich habe bei den Programmiervorschlägen weniger an verwertbare Programme gedacht, als vielmehr an Programmiertechniken. Wir wollen möglichst viele Techniken kennen lernen – das bedeutet aber, dass ein eleganterer Code verloren geht. Mit zunehmenden Kenntnissen kannst du deine eigenen Vorstellungen entwickeln.

Ratsam ist, sich optisch zunächst an das Windows-Aussehen zu halten. Individuelle Gestaltungen können nicht über unsaubere Programmierung hinwegtäuschen. Schlechte Beispiele findest du zuhauf im Freeware-und Shareware-Bereich.

Bedienoberfläche

Als erstes Beispiel soll ein Würfelspiel programmiert werden. Der Computer würfelt, und der Anwender soll die Augenzahl raten. Das Ergebnis wird angezeigt und ausgewertet. Für dieses Programm benötigst du ein Bildfeld, sieben Formen für die Augen, sechs Befehlsschaltflächen für den Tipp, eine Befehlsschaltfläche fürs Beenden des Programms, drei Bezeichnungsfelder für die Anzeigen der Treffer und zwei Bezeichnungsfelder für Beschriftungen.

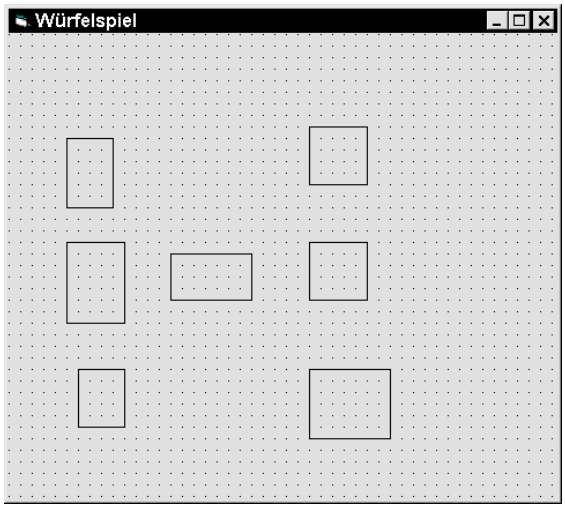
Form.		
HEIGHT	Höhe	ca. 5200
WIDTH	Breite	ca. 5800
CAPTION	Beschriftung	**Würfelspiel**

Den Wert einer Eigenschaft veränderst du ganz einfach, indem du sie anklickst und einen neuen Wert eingibst, wodurch der alte Wert überschrieben wird. Ein Druck auf EINGABE setzt die Eigenschaft sofort um. Du kannst also bei Breite oder Höhe der Form die Wirkung sofort sehen und, falls nötig, einen anderen Wert eingeben.

Willst du eine Eigenschaft in der Liste ansteuern, machst du das so: du drückst STRG+UMSCHALT und dann den Anfangsbuchstaben der gewünschten Eigenschaft. Sind mehrere Eigenschaften mit gleichem Anfangsbuchstaben vorhanden, tippst du einfach noch einmal den selben Buchstaben.

PictureBox		
NAME	Name	picWürfel
WIDTH	Breite	ca. 3000
HEIGHT	Höhe	ca. 3000
TOP	Abstand vom oberen Rand	ca. 350
LEFT	Abstand vom linken Rand	ca. 350

1. m Werkzeugfenster doppelklickst du auf die PICTUREBOX ... 
2. ...und gibst im EIGENSCHAFTEN-Fenster die obigen Werte ein
3. Dann klickst du auf SHAPE – Vorsicht, nur ein einfacher Klick, da du die einzusetzende Objekt andernfalls nicht bearbeiten kannst! ..... 
4. ... ziehst ein Kästchen im Bildfenster auf und platzierst es links oben ...
5. ... und wiederholst diesen Prozess, bis du insgesamt sieben Kästchen wie in der Abbildung angeordnet hast.



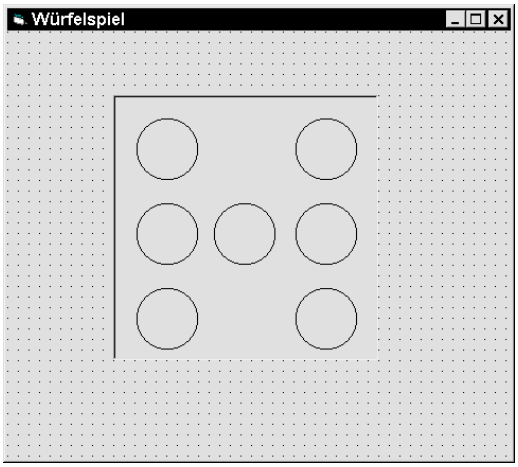
Dass sieben 'Augen' benötigt werden, hat seinen Grund, den du noch erfahren wirst.

Im EIGENSCHAFTEN-Fenster ist die Eigenschaft SHAPE (Figur) markiert. Klickst du auf den keinen Pfeil im der Werteliste, öffnet sich eine weitere Auswahlliste mit den möglichen Figuroptionen. Dort findest du unter der Ziffer 3 die Option KREIS.

Nun kann man die Kästchen einzeln in Kreise verwandeln – das lässt sich aber auch einfacher erledigen – du hältst STRG, während du mit der Maus die Kästchen nacheinander anklickst, worauf alle hierdurch markierte Objekte Anfasser erhalten. Anders augedrückt gruppierst du die Objekte. Klickst du dann die Option KREIS an, verwandeln sich alle Kästchen auf einen Schlag in Kreise. Vergiss nicht die Werte einzugeben:

Shape		
Height	Höhe	700
Width	Breite	700

1. Klickst du in das Bildfeld, wird die Gruppierung aufgehoben
2. Du ordnest die Kreise der Augen wie bei einer Sechs an, indem du sie jeweils anklickst und korrekt platzierst.
3. Das siebte Auge setzt du wie in der Abbildung genau in die Mitte.



4. Du ordnest die 'Augen' so an, dass die Durchnumerierung von Shape1 bis Shape6 von links nach rechts erfolgt und die Figur in derMitte Shape7 ist. Dann passt du das Bildfeld so an, dass ein optisch ausgewogener Eindruck entsteht.

1		2
3	7	4
5		6

5. Nun doppelklickst du auf COMMANDBUTTON
6. Die Schaltfläche platzierst du oben rechts neben dem Bildfeld – wie das bei mir aussieht, siehst du in der Darstellung des Endergebnisses.
7. Im Eigenschaftenfenster machst du folgende Angaben:

   CommandButton

NAME	Name	**cmdSchließen**
CAPTION	Beschriftung	**Schließen**

8. Dann doppelklickst du auf die neue Schaltfläche, die inzwischen ihren Namen cmdSchließen erhalten hat.
   Es öffnet sich das Code-Fenster mit dem Prozedurrahmen für das Ereignis CLICK.
   Visual Basic vermutet klugerweise das Wahrscheinlichste, und das ist bei einem Befehlsbutton nun einmal ein Klick.
9. Dort, wo der Cursor blinkt, schreibst du **end**.
   Damit Program-Listings übersichtlich bleiben, rückt man im allgemeinen Befehlszeilen um einen Tabstop ein. Also TAB und **end** schreiben.
   Visual Basic wandelt dein **end** in **End** um und färbt es blau ein – so teilt dir das Programm mit, dass der Befehl korrekt geschrieben und von Visual Basic erkannt wurde. Die Sache sieht also so aus:

```
Option Explicit
Private Sub cmdSchliEßen_Click()
    End
End Sub
```

Abschließend drückst du wie immer F5, worauf das Programm startet. Mit einem Klick auf die Befehlsschaltfläche kannst du es wieder beenden.

Du hast gesehen, wie einfach es sein kann, dem Computer Befehle zu erteilen. Leider geht es nicht immer so unproblematisch. Immerhin kennst du schon die erste Anweisung, nämlich **End**.

Der nächste Punkt in unerer Programmierung ist eine Geschmacksfrage. Es geht um die farbliche Ausgestaltung des Würfelspiels – wir wollen das graue Einerlei etwas auffrischen. Für diese Aufgabe benutzt du selbstverständlich wieder das EIGENSCHAFTEN-Fenster.

1. Du schließt das CODE-Fenster bw. klickst in den sichtbaren Teil der Form, so dass diese wieder im Vordergrund steht.
2. Dann klickst du einen Kreis an, ...
3. ... öffnest, falls notwendig, die AUSWAHLLISTE über den Pfeil...
4. und machst folgende Angaben:

Shapes1 - 7		
FILLSTYLE	Füllstil	ausgefüllt
FILLCOLOR	Füllfarbe	aussuchen
BORDERWIDTH	Randstärke	3
BORDERCOLOR	Randfarbe	aussuchen

Probiere an einem SHAPE alle möglichen Einstellungen aus. Wenn du dich entschieden hast, werden die Augen wieder gruppiert und die Einstellungen erneut vorgenommen..

5. Bei der Gelegenheit kannst du dem Bildfeld gleich eine passende Hintergrundfarbe geben: du klickst in eine freie Stelle des Bildfeldes bzw. wählst in der AUSWAHLLISTE oben im EIGENSCHAFTEN-Fenster den Punkt PICWÜRFEL, wodurch das Bildfeld aktiviert wird.

picWürfel		
BackColor	Hintergrundfarbe	aussuchen

Je nach Geschmack sieht der Würfel jetzt schön bunt aus. Fehlen noch die Befehlsschaltflächen für den Tipp des Spielers.

6 Also doppelklickst du sechs Mal auf COMMANDBUTTON im WERKZEUG-Fenster, worauf sechs Buttons übereinander in der Mitte der Form angelegt werden.

7 Diese Buttons ordnest du auf der rechten Seite der Form an, unter der SCHLIESSEN-Schaltfläche, und gruppierst sie.

8 Im EIGENSCHAFTEN-Fenster wählst du WIDTH und trägst **`372`** ein.
Visual Basic legt standardmäßig die Höhe eines COMMANDBUTTON mit 372 fest, so dass bei derselben Breite eine quadratische Schaltfläche entsteht.

9 Dann machst du folgende Angaben:

Command1 – Command6		
NAME	Name	**`cmdEins, cmdZwei, cmdDrei, cmdVier, cmdFünf, cmdSechs`**
CAPTION	Beschriftung	**`1,2,3,4,5,6`**

Es erleichtert die Übersicht, wenn man sich angewöhnt, jedem Namen eine Kennung beizufügen, z.B. CMDEINS oder PICWÜRFEL. Für die FORM und das PROJEKT behältst du allerdings die Standardbezeichnungen – hier gibt es kaum Identifikationsprobleme.

10 Nun klickst du drei Mal LABEL an und ziehst die drei Bezeichnungsfelder auf der Form auf.

11 Im EIGENSCHAFTEN-Fenster machst du folgende Angaben:

Label1		
NAME	Name	**`lblVerloren`**
CAPTION	Beschriftung	**`00`**
BORDER-STYLE	Rahmenstil	Fest Einfach
FONT (Größe)	Schrift	14
AUTOSIZE	Automatische Anpassung	**`True`** (Wahr)
ALIGNMENT	Ausrichtung der Anzeige	Zentriert
Label2		
NAME	Name	**`lblGewonnen`**
CAPTION	Beschriftung	**`00`**
BORDER STYLE	Rahmenstil	Fest Einfach
FONT (Größe)	Schrift	14
AUTOSIZE	Automatische Größenanpassung	True (Wahr)
ALIGNMENT	Ausrichtung der Anzeige	Zentriert

Label3		
NAME	Name	**lblAnzeige**
CAPTION	Beschriftung	Anzeige
BORDER-STYLE	Rahmenstil	Fest Einfach
FONT (Größe)	Schrift	18
ALIGNMENT	Ausrichtung des Textes	Zentriert

Zur Einstellung der Eigenschaften gibt es nicht viel zu sagen. Du wählst die jeweiligen Eigenschaften, die dazugehörigen Werte tippst du ein, bzw. wählst sie aus der Auswahlliste, die sich bei einigen Eigenschaften öffnen lässt.

12 Die Bezeichnungsfelder LBLVERLOREN und LBLGEWONNEN sollen so groß sein, dass eine zweistellige Zahl genau hineinpasst. Deshalb wird als Wert die CAPTION 00 gesetzt und AUTOSIZE auf TRUE gestellt.

13 Naturgemäß beginnt die Anzeige mit einer einstelligen Zahl zu zählen. Visual Basic würde die Größe des Bezeichnungsfeldes entsprechend anpassen. Um das zu verhindern, lassen wir Visual Basic die optimale Größe mit **AutoSize = True** anpassen und schalten dann die AUTOSIZE wieder auf FALSE.

14 Wenn du jetzt noch die Eigenschaft Alignment, also die Ausrichtung der Anzeige im Feld auf zentriert stellst, wird die einstellige Anzeige mittig dargestellt.

> Für einen Programmablauf sind solche Feinheiten unwichtig; für eine sorgfältige Präsentation eines Programms aber unerlässlich.

15 Die Eigenschaftswerte **True** und **False** haben hier etwa die Bedeutung von eingeschaltet, ausgeschaltet.

16 Das Anzeigefeld LBLANZEIGE passt du in der Größe so in die Form ein, dass es möglichst professionell aussieht.

17 Durch den Wert Zentriert der Eigenschaft Alignment werden die Anzeigen immer mittig dargestellt.

18 Selbstverständlich kannst du dir noch Schriftfarben oder andere Schrifttypen aussuchen.

19 Zum Schluss werden noch die Bezeichnungsfelder für die Beschriftung der Trefferzähler angeordnet:

Label1		
CAPTION	Beschriftung	**Verloren**
AUTOSIZE	Automatische Anpassung	**True**
Label2		
CAPTION	Beschriftung	**Gewonnen**
AUTOSIZE	Automatische Anpassung	**True**

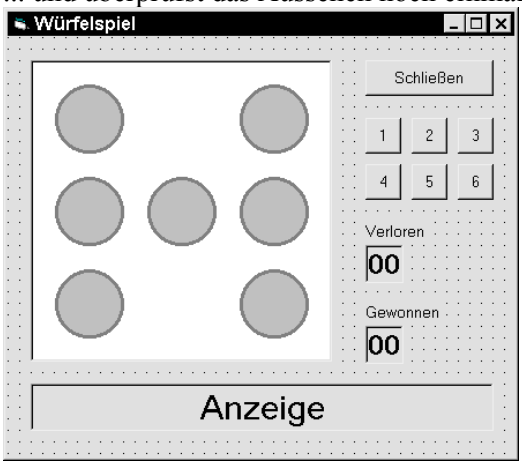
> Bei Steuerelementen, die im Programm nicht wieder angesprochen werden, wie hier z.B. bei reinen Beschriftungen, lasse ich die Namen bestehen. Es gibt keinen Sinn, sich Namen auszudenken, die gar nicht verwendet werden. Anders ist das, wenn im Laufe des Programms die Werte geändert werden.

20 Nun richtest du alle Objekte aus, ...

21 ... passt die Größe der Form an, ...

22 ... drückst F5 ...

23 ... und überprüfst das Aussehen noch einmal.



So ähnlich wie in der Abbildung könnte deine Oberfläche aussehen.

24 Damit das Kunstwerk nicht verloren geht, muss es gespeichert werden. Also klickst du auf die entsprechende Schaltfläche.

25 Es öffnet sich ein Dialogfenster, in dem du dir einen Ordner für dein Projekt wählen kannst.

26 Als Dateiname gibst du WÜRFELN ein. Die Form wird unter dem Namen WÜRFELN. gespeichert – wie du siehst, heißt die Dateiendung für Visual Basic .FRM.

> Während der Arbeit an einem Projekt sind Pannen unvermeidlich – sei es, dass der Computer abstürzt, sei es, dass er sich nur aufhängt. So etwas ist meist mit Datenverlust verbunden. Daher rate ich dazu, ein Projekt bei Arbeitsbeginn unter einem eigenen Namen zu speichern und während der Arbeit immer wieder mal die Schaltfläche für Speichern zu klicken. Auf diese Art wird der aktuelle Stand gesichert.

Zufall

Bevor wir dem Spiel Leben einprogrammieren, wäre ein Exkurs über den Zufall einzufügen.

Computer faszinieren immer wieder mit ihrem sogenannten Zufallsgenerator – anscheinend sind sie in der Lage, zufällige Ereignisse zu erzeugen, sei es die Auswahl eines Buchstabens aus dem Alfabet oder einer Zahl aus einer Zahlenreihe.

Natürlich gibt es beim Computern keine wirklichen Zufälle – das sieht nur so aus.

1. Du startest erneut Visual Basic. Das führt dazu, dass du nun zwei Versionen des Programms im Arbeitsspeicher hast
   Für diesen Versuch legen wir keinen Wert auf die Namensgebung usw, da wir das Programm nicht speichern.
2. Du wählst STANDARD.EXE ...
3. ... drückst im WERKZEUG-Fenster auf den COMMANDBUTTON ...
4. ... doppelklickst auf Command1
5. ... und setzt folgende Eigenschaften ein:

```
1.Sub Private Command1_Click()
2.     Dim zufall As Single
3.     zufall = Rnd
4.     Print zufall
5.End Sub
```

Die Zeilen 3 – 4 sind Anweisungen, die ausgeführt werden, wenn auf die Befehls-schaltfläche geklickt wird.

- Zeile 2: Die Variable mit dem Namen ZUFALL wird als Variablentyp SINGLE deklariert.
- Zeile 3: Dieser Variablen wird der Wert zugewiesen, der sich aus der Funktion **Rnd** ergibt. Diese Funktion liefert einen Wert zwischen 0 und 0,999999 und weist ihn der Variablen ZUFALL zu..
- Zeile 4: Das Programm wird angewiesen, die Variable *zufall* auf die Form zu schreiben.

Abgesehen davon, dass du mit den Zahlen nicht viel anfangen kannst, wirst du feststellen, dass bei jedem Neustart immer wieder dieselben Zahlen angezeigt werden – also war das doch nichts mit dem Zufall?! Der Grund ist ein komplizierter mathematischer Transformations-prozess. Prinzipiell benötigt die RND-Funktion einen Startwert, der ohne Eingriff immer gleich ist – daher die gleichen Zahlen. Ändert man den Startwert, wird die Sache interessanter.

Mit der Anweisung **Randomize** wird jeweils ein neuer Startwert erzeugt. Setzt du hinter Zeile 1 die Anweisung **Randomize** ein und startest das Programm erneut, wirst du feststellen, dass die Zahlenfolgen jetzt jedesmal anders aussehen.

Die nächste Funktion, die wir benötigen, um zu anwendbaren Zufallszahlen zu kommen, ist **Int**. Vereinfacht ausgedrückt liefert diese Funktion einen Wert, bei dem die Nachkommastellen abgeschnitten sind. Aus **0,1234567** wird durch **Int(0,1234567)** der Wert **0**. Das würde zueiner ganzen Reihe von Nullen führen. Werden die erzeugten Zahlen aber z.B. mit 10 multipliziert, dann verschiebt sich das Komma. Aus
0,23*10 wird **2,3**; durch **Int(2,3)** dann 2. So bekommt man ganzzahlige Zahlen.

Bei einem Würfel gibt es nur sechs mögliche Zahlen. Also müsste es eine Formel geben, die den Zahlenumfang berücksichtigt. Gibt es:

```
zufall = Int(rnd * Obergrenze) + Untergrenze
```

- **Zeile 3: zufall = Int(Rnd * 6) + 1**

Jetzt klappt's! Wie gesagt, ein echter schicksalshafter Zufall ist das nicht, aber es scheint so.

Beende dieses Visual Basic, ohne zu speichern, und aktiviere dein Würfelprogramm.

Programmcode

Der Spieler soll eine Zahl anklicken, worauf der Computer eine Zufallszahl wählt. In einer Auswertung werden die Zahlen verglichen, und es wird geprüft, ob ein Treffer vorliegt oder nicht. Die Treffer und Nichttreffer werden hochgezählt. Im Anzeigefeld wird angezeigt, ob getroffen wurde (gewonnen) oder nicht (verloren).

Natürlich müssen Werte in Variablen gespeichert werden, z.B. der vom Spieler gewählte Wert. Wir nennen die Variable RATEZAHL und den vom Computer erzeugten Wert ZUFALLSZAHL.

Die RATEZAHL muss in jeder Prozedur immer wieder deklariert werden – sie bekommt nur jeweils einen anderen Wert. Das ist umständlich. Wenn eine Variable im gesamten Projekt gültig sein soll, deklariert man sie im Allgemeinteil, der vor den Prozeduren liegt. Für den Zahlenbereich des Würfels genügt der Variablentyp BYTE.

> Einige Worte zum Code: alles, was auf den nächsten Seiten im Codefenster eingegeben wird, ist Bestandteil des Gesamtcodes für unser Projekt.
>
> Leerzeilen im Code drücken aus, dass hier eine neue Prozedur beginnt, notwendig sind sie aber nicht.

Schaltflächen für den Tipp

Du doppelklickst auf CMDEINS. Unmittelbar hat Virtual Basic folgende Zeilen eingesetzt – die Zeilen 3 und 5 geben den Rahmen für die Prozedur ab, die wirksam wird, wenn auf CMDEINS geklickt wird.:

```
1. Option Explicit
2.
3. Private Sub cmdEins_Click()
4.
5. End Sub
```

Der Textcursor blinkt in der Zeile 2. Hier gibst du ein:

```
2. Dim Ratezahl As Byte
```

Der Code lautet jetzt also:

```
1. Option Explicit
2. Dim Ratezahl As Byte
3. Private Sub cmdEins_Click()
```

Nun machst du folgende Eingaben:

```
4. Ratezahl = 1
```

In Zeile 4 wird der Variablen RATEZAHL der Wert 1, entsprechend der Befehlsschaltfläche zugewiesen.

```
5.   lblAnzeige.Caption =
  Rate-
  zahl
```

Die Zeile 5 dient nur für Testzwecke und wird später wieder gelöscht.

```
6. End Sub
```

Die Anweisung lautet: Gib der Eigenschaft CAPTION des Objektes **lblAnzeige** den Wert aus der Variablen **Ratezahl**.

Bevor du das Programm startest, programmierst du noch die Zeilen 8 und 9, indem du vorher auf CMDZWEI doppelklickst. Im CODE-fenster öffnet sich die entsprechende Prozedur mit den üblichen vorgebenenen Zeilen:

```
7. Private Sub cmdZwei_Click()
```

und

```
9. End Sub
```

Beim Cursor gibst du die Zeilen 8 und 9 ein:

```
8. Ratezahl = 2
9. lblAnzeige.Caption =
Ratezahl
```

Hierdurch ändert sich der Wert von end sub natürlichauf 10:

```
10. End Sub
```

Abschließend drückst du auf F5.

Willst du nachprüfen, ob die Sache funktioniert, klickst du auf CMDEINS und CMDZWEI – in LBLANZEIGE müsste die Zahl 1 bzw. 2 erscheinen.

Die Zeilen 5 und 9 kannst du wieder löschen: du setzt den Mauszeiger vor die zu löschende Zeile, markierst sie mit einem Klick und drückst auf ENTF.

Nun gibst du für alle Schaltflächen die entsprechende Variable RATEZAHL ein – du wiederholst also die notwendigen Eingaben für die Buttons CMD3 bis CMD6, wobei du natürlich jeweils die Zahl des aktuellen Buttons einsetzt

Die Zufallszahl des Computers

Der Computer erhält die Aufgabe, nach einem Würfeln durch den Spieler selber eine Zahl zu würfeln. So eine Prozedur hat Visual Basic natürlich nicht in seinem Programm – also müssen wir sie selbst anlegen unter dem Namen COMPUTER. Diese Prozedur wird mehrmals verwendet – also setzt du im Allgemeinteil am Anfang unseres Code, nach der Zeile

```
dim Ratezahl As Byte,
```

folgende Zeile ein.

```
Dim Zufallszahl As Byte
```

Dann gibst du am Ende des CODE-Fensters

```
1. Private sub Computer
```

ein, den Rest erledigt Visual Basic, worauf der Textcursor in der nächsten Leerzeile steht.

Der Computer soll in dieser Prozedur seine Zahl erzeugen und der Variablen ZUFALLSZAHL zuweisen. In Zeile 2 setzt du die Anweisung Randomize ein, und in Zeile 3 wird die auf Seite 20 geübte Funktion für die Zufallszahl eingetragen – das Ergebnis ist:

```
2. Randomize
3. Zufallszahl Int(Rnd * 6)+1
4.End Sub  (vom Programm vorgegeben)
```

Um zu sehen, was der Computer erzeugt, kannst du zwischen Zeile 3 und 4 eine Leerzeile einfügen und folgendes eintragen:

```
LblAnzeige.Caption=Zufallszahl
```

Dann drückst du auf F5.

Was nun? Nichts passiert. Das ist ein gutes Beispiel für die objektorientierte Programmierweise von Visual Basic: solange die Prozedur COMPUTER nicht aufgerufen wird, schlummert sie nur still vor sich hin.

Zum Glück ist es ganz einfach, die Prozedur zum Leben zu erwecken. Man ruft sie mit ihrem Namen auf. Der Computer soll seine Zahl erzeugen, wenn der Spieler eine Zahl mit der Befehlsschaltfläche gewählt hat; also gehört der Aufruf in die jeweiligen Click-Prozeduren.

Der Code sieht so aus:

```
1.Private Sub cmdFünf_Click()
2.    Ratezahl = 5
3.    computer
4.End Sub
```

Dieses Beispielprogramm zeigt dir das Listing für die Prozedur CMDFÜNF_CLICK.

Die zwischen Zeile 3 und Zeile 4 eingefügte Anweisung in der Prozedur Computer kannst du nun wieder löschen.

Wenn du das alles erledigt hast, können wir uns Gedanken darüber machen, wie ein Computer Werte miteinander vergleicht. Hast du in der Schule im Matheunterricht aufgepasst, ist die Sache nicht besonders kompliziert. Man braucht nur die mathematischen Zeichen '**=**', '**<**' '**>**' und '**<>**', also gleich, kleiner als, größer als und ungleich. Visual Basic kann damit umgehen – wie, das soll uns hier nicht weiter interessieren.

Die Prozedur Auswertung anlegen

Im Abschnitt „Kontrollstrukturen“ auf Seite 27 wird ausführlich über Möglichkeiten der Kontrolle des Programablaufs eingegangen. Hier genügt ein kurzer Abriss, damit das Programm vervollständigt werden kann.

In der Prozedur AUSWERTUNG wird der Computer angewiesen, zu überprüfen, ob 'seine' Zahl mit der Spielerzahl übereinstimmt oder nicht – computerisch ausgedrückt: wenn Zufallszahl = Ratezahl, dann gewonnen, sonst verloren. Eingeenglischt heißt das: If Zufallszahl = Ratezahl, then gewonnen, else verloren; abgekürzt:

```
If Zufallszahl = Ratezahl Then
gewonnen
If Zufallszahl <> Ratezahl
Then verloren
```

Diese Schreibweise würde genügen, kann aber übersichtlicher geschrieben werden:

```
If Zufallszahl = Ratezahl Then
   Gewonnen
Else
   Verloren
End If
```

Gewonnen und verloren sind Variablen, in denen die Treffer hochgezählt werden sollen. Es müssen also erst einmal die Variablen deklariert werden.

Hier taucht allerdings ein Problem auf: Visual Basic setzt den Wert der Variablen bei jedem Aufruf der Prozedur wieder auf Null. Wir würden also nicht hochzählen können. Es gibt für dieses Problem das Schlüsselwort **Static**. Eine mit **Static** deklarierte Variable behält ihren aktuellen Wert so lange, bis das Programm neu gestartet wird.

Zeile 2 und Zeile 3 in der folgenden Prozedur zeigen, wie die beiden Variablen deklariert werden.

```
1.Private Sub Auswertung()
2.Static gewonnen As Byte
3.Static verloren As Byte
4.   If Zufallszahl = Ratezahl
     then
5.   gewonnen = gewonnen + 1
6.   lblAnzeige.Caption =
     "Gewonnen!"
7.   lblGewonnen.Caption =
     gewonnen
8. Else
9.   verloren = verloren + 1
10.  lblAnzeige.Caption =
     Verloren!"
11.   lblVerloren.Caption =
     verloren
12. End If
13.End Sub
```

Etwas merkwürdig kommt dir sicherlich der Code in Zeile 5 und in Zeile 9 vor. Das Gleichheitszeichen ist kein mathematisches Zeichen, sondern ein Zuweisungszeichen. Der Variablen GEWONNEN wird ein Wert zugewiesen. In diesem Fall ist das der ursprüngliche Wert der Variablen, um 1 erhöht (+ 1). Der Startwert der Variablen ist 0; dem ursprünglichen Wert 0 wird der um 1 erhöhte Wert zugewiesen – und das ist 0 + 1. Beim nächsten Mal wird der alte Wert 1 wiederum um 1 erhöht. So wird dem Computer das Zählen beigebracht. Den Wert der Variablen zeigen wir mit Zeile 7 bzw. Zeile 11 in den entsprechenden Anzeigefeldern mit der Eigenschaft CAPTION an. Zeile 6 und Zeile 10 zeigen die Zeichenketten zwischen den Anführungszeichen an.

Abschließend drückst du wie gewohnt F5 – und vergiss bitte nicht, das Spiel zu speichern.

Die Anzeige der Augenzahl

Grundsätzlich kann unser Spiel jetzt laufen. Die Augenzahl-Anzeige im Bildfeld haben wir zwar noch nicht, du kannst aber immerhin schon spielen. Allerdings ist es vorläufig noch ein geheimnis, welche Zahl der Computer gewürfelt hat. Das könnte man in einem Anzeigefeld darstellen, aber wozu haben wir das Bildfeld?!

Bisher zeigt der Würfel sieben Augen. Das siebte Auge in der Mitte brauchst du für die Eins, die Drei und die Fünf. Damit vor dem Würfeln des Computers eine Starteinstellung zu sehen ist, muss für eine Sechs das SHAPE7 verschwinden – es darf aber nicht gelöscht werden, denn es wird ja gebraucht. Also muss es unsichtbar werden.

1. Du klickst SHAPE7 an ...
2. ... und drückst STRG + UMSCHALT + V

   Im EIGENSCHAFTEN-Fenster ist die Eigenschaft VISIBLE –also sichtbar – markiert. Der Wert ist TRUE, also Wahr. Das Objekt ist sichtbar.
3. Nun stellst du den Wert um auf **Visible = False** ...
4. ... und drückst auf F5

Es sollte jetzt eine Sechs erscheinen.

Achte darauf, dass du die Eigenschaft für das SHAPE veränderst und nicht versehentlich ein anderes Objekt, z.B. die FORM, aktiviert hast. Oben im EIGENSCHAFTEN-Fenster steht in der Anzeige, welches Objekt gerade 'dran' ist. Es kann dir sonst passieren, dass plötzlich die Form unsichtbar gemacht wird.

Du ahnst sicher schon, wie es weitergeht. Es wird für jede Augenanzeige das jeweilige Shape unsichtbar gesetzt. Da kommt einige Tipparbeit auf dich zu, aber über BEARBEITEN | KOPIEREN und BEARBEITEN | EINFÜGEN kannst du den Arbeitsaufwand verkürzen.

Du legst die Prozeduren 1-6 mit der folgenden Prozedur an:

```
1.Private Sub Eins()
2.    Shape1.Visible = False
3.    Shape2.Visible = False
4.    Shape3.Visible = False
5.    Shape4.Visible = False
6.    Shape5.Visible = False
7.    Shape6.Visible = False
8.    Shape7.Visible = True
9.End Sub
```

Diese Prozedur zeigt dir, dass bis auf SHAPE7 alle Shapes auf unsichtbar gesetzt sind, so dass beim Aufruf von PROZEDUR EINS nur das mittlere Auge sichtbar sein dürfte. Um das zu überprüfen, legen wir eine Befehlsschaltfläche an, doppelklicken auf sie und versehen sie mit folgender Prozedur:

```
Private Sub Command1_Click()
   Eins
End Sub
```

In diese Test-Prozedur trägst du den Aufruf der Prozedur Eins ein, dann drückst du auf F5 und klickst auf die Befehlsschaltfläche. Eigentlich müsste jetzt die Augenzahl Eins auf dem Würfel zu sehen sein. Stimmt's?

Wie du dich vielleicht erinnerst, hatte ich darauf hingewiesen, dass es wichtig ist, die richtige Reihenfolge der Shapes einzuhalten. Hast du alles beachtet, dürfte es keine Probleme geben.

1. Du markierst den Code in der Prozedur Eins – und zwar nur den Code, also die Zeilen 2-8 in der Prozedur oben auf der Seite!
2. Dann drückst du STRG + C, wodurch der markierte Teil kopiert wird, ...
3. setzt den Cursor in die Leerzeile in Prozedur Zwei ...
4. und drückst STRG + V

Die Sache sieht so aus:

```
Private Sub Zwei()
   Shape1.Visible = True
   Shape2.Visible = False
   Shape3.Visible = False
   Shape4.Visible = False
   Shape5.Visible = False
   Shape6.Visible = True
   Shape7.Visible = False
End Sub
```

Es müssen die beiden Augen links oben und rechts unten, also Shape1 und Shape6 sichtbar (`Visible = True`), alle anderen unsichtbar sein. Die Eins im Code wird durch Zwei ersetzt. Dann drückst du auf F5 und klickst die Befehlsschaltfläche an. Es sollten zwei Augen zu sehen sein.

So machst du das jetzt für jede Augenzahl – und testest immer wieder mit der Prozedur auf Seite 22:

```
If Zufallszahl = Ratezahl Then
   Gewonnen
Else
   Verloren
End If
```

Wenn du für alle Prozeduren den Code geschrieben hast, löscht du die Prozedur `Command1_Click`. Nun fehlt nur noch die Anbindung an diese Prozeduren. Der Compter erzeugt eine Zahl und diese Zahl soll an die Augenanzeige-Prozeduren weitergeleitet werden, d.h. mit Wenn-Dann müssten wir weiterkommen. Wenn Zufallszahl = 1, dann Prozedur Eins usw. Es gibt natürlich verschiedene Wege, so ein Problem zu lösen, aber für diesen Zweck ist der Weg der Wenn-Dann (If-Then)-Entscheidung der übersichtlichste.

Prozedur Würfelanzeige anlegen

Als nächstes definieren wir die Prozedur WÜRFELANZEIGE mit folgender Prozedur:

```
Private Sub Würfelanzeige()
    If Zufallszahl =1 Then Eins
    If Zufallszahl =2 Then Zwei
    If Zufallszahl =3 Then Drei
    If Zufallszahl =4 Then Vier
    If Zufallszahl =5 Then Fünf
    If Zufallszahl =6 Then
Sechs
End Sub
```

Diese Prozedur leuchtet unmittelbar ein – es muss diese Prozedur nur noch aufrufen.

Willst du die optimale Platzierung für den Prozeduraufruf finden, gehst du gedanklich das Programm durch und prüfst, was in welcher Reihenfolge erfolgt. Wird auf eine Zahl-Schaltfläche geklickt, wird das Click-Ereignis ausgelöst. Darin wird die Variable gespeichert und zur Computer-Prozedur verzweigt, in der wiederum eine Zufallszahl erzeugt wird. Es ist sinnvoll, die Würfelanzeige hier aufzurufen. Der Würfel wird angezeigt, der Computer kehrt zur Prozedur zurück und verzweigt zur AUSWERTUNG. Die wird logischerweise dort vorgenommen und das Ergebnis angezeigt. Damit ist dieser Teil abgearbeitet, und das Programm wartet auf eine erneute Eingabe. Dann beginnt das gleiche Spiel noch einmal – bis du auf die Schließen-Schaltfläche klickst.

```
1.Private Sub computer()
2.    Randomize
3.    Zufallszahl = Int(Rnd * 6)+1
4.    Würfelanzeige
5.    Auswertung
6.End Sub
```

Zeile 4 und Zeile 5 in dieser Prozedur rufen die jeweilig aktuellen Prozeduren auf. Dieses Unterteilen eines Programms in viele kleine Prozeduren hat den großen Vorteil, dass bei Änderungen immer nur ein kleines Häppchen durchsucht und geändert werden muss. Du hast gesehen, dass es kein Problem ist, eine selbstdefinierte Prozedur anzulegen und in eine Ereignisprozedur einzufügen. Willst du eine Prozedur zu Testzwecken kurzfristig abschalten, brauchst du nur ein sogenanntes Kommentarzeichen davor zu setzen. Das Apostroph ist ein solches Kommentarzeichen. So veranlasst z.B. `'Auswertung` Visual Basic, den Aufruf zu überlesen. Ein Kommentarzeichen wird – wie der Name schon sagt – für Kommentare in einem Programm eingesetzt.

> Du fragst dich, wozu Kommentare gut sind? Ich garantiere dir, dass du nach einigen Monaten bei deinen Programmen nicht immer mehr genau weißt, welchen Sinn eine bestimmte Programmieridee hatte. Wenn du das Programm mit Kommentaren versehen hast, hast du deine eigenen Hinweise als Gedächtnisstütze.
>
> Möglich wäre auch, dass du ein Programm nachträglich um irgendeine Prozedur ergänzen willst.

Sehen wir uns ein Beispiel für eine solche Kommentarzeile an:

```
1.Option Explicit
2.Dim Ratezahl As Byte
3.Dim Zufallszahl As Byte

4.'MessageBox bei Spielende

5.Private Sub cmdEins_Click()
6.Ratezahl = 1
    ...
```

Zeile 4 in dieser Prozedur ist eine Kommentarzeile, die ich mittendrin angelegt habe, als mir die Idee kam, dass man vor Beenden des Programms noch eine Sicherheitsabfrage einbauen könnte. Damit ich es nicht vergesse, habe ich diese Kommentarzeile eingefügt. Wenn die Angelegenheit erledigt ist, wird sie einfach gelöscht.

Wenn du fremde Programme liest, kannst du oft nicht nachvollziehen, was der Entwickler des Programms beabsichtigt hat, denn du weißt ja, dass es bei Visual Basic oft mehrere Lösungsmöglichkeiten für ein Problem gibt. In solchen Fällen helfen Kommentare dem Leser sehr gut weiter. Kommentarzeilen sind meistgrün eingefärbt, so dass sie schnell zu erkennen sind.

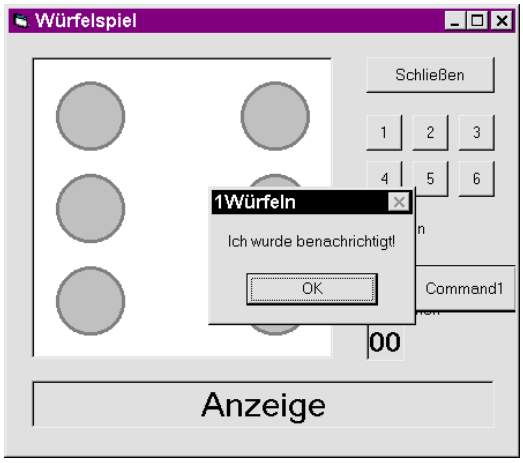
Mitteilungsfenster (Messagebox)

Visual Basic stellt dir einfach zu programmierende Dialogfelder zur Verfügung, die du in gewissen Grenzen nach deinen Wünschen ausgestalten kannst.

1. Du erstellst eine provisorische Befehlsschaltfläche und doppelklickst auf sie
2. Im Codefenster gibst du folgende Prozedur ein:

```
Private Sub Command1_Click()
  MsgBox "Ich werde benach-
  richtigt!"
End Sub
```

3. Dann doppelklickst du auf die Befehlsschaltfläche,...
4. ... fügst die Zeile 2 des obigen Programmms ein, ...
5. ... drückst auf F5 ...
6. und klickst auf die Schaltfläche.



Die Abbildung zeigt dir das Ergebnis der kleinen Zeile in der obigen Prozedur. Dieses Fenster wird selbständig von Windows und nicht etwa von nicht von Visual Basic angelegt. Seine Platzierung kannst du nicht beeinflussen, und auch die Ausgestaltung bleibt bis auf wenige Ausnahmen Windows überlassen. Aber ganz so hilflos bist du natürlich nicht.

Die MessageBox hat eine Titelleiste, die, falls du nichts anderes angibst, den Namen des Projekts anzeigt. Es gibt Platz für eine Nachricht, und offensichtlich gibt es die Möglichkeit, Befehlsschaltflächen zu verwenden.

Du schließt das Mitteilungsfenster mit einem Klick auf OK,, worauf das Mitteilungsfenster geschlossen wird und das Programm wartet, als wäre nichts geschehen.

Dir ist sicher schon öfter aufgefallen, dass sogenannte QuickInfos eingeblendet werden, wenn du den Mauszeiger auf ein Objekt bewegst, z.B. im WERKZEUG-Fenster, aber auch beim Code-Schreiben. Visual Basic will dir helfen, indem das Programm häufig eine Auswahlliste aller denkbaren Möglichkeiten einblendet. Schreibst du **`Dim Variable As`**, blendet Visual Basic eine Liste ein, die du mit der ➪-Taste ansteuern kannst. Tippst du z.B. den ersten Buchstaben von INTEGER ein, zeigt dir Visual Basic die Liste ab dem Buchstaben i. Hast du den Wert ausgewählt, kannst du ihn mit der LEERTASTE einfügen.

Schreibst du den Begriff MSGBOX, siehst du eine längere Zeile, die anzeigt, was alles in einer MSGBOX eingetragen werden kann. Häufig sind diese QuickInfos verwirrend, manchmal aber sehr hilfreich, vor allem, wenn es um die richtige Reihenfolge der Parameter geht.

Message-Box	
Typ	**Beschreibung**
0	SCHALTFLÄCHE OK
1	SCHALTFLÄCHEN OK, ABBRECHEN
2	SCHALTFLÄCHEN BEENDEN, WIEDERHOLEN, IGNORIEREN
3	SCHALTFLÄCHEN JA, NEIN, ABBRECHEN
4	SCHALTFLÄCHEN JA, NEIN
5	SCHALTFLÄCHEN WIEDERHOLEN, ABBRECHEN
16	STOP-SYMBOL
32	FRAGEZEICHEN-SYMBOL
48	AUSRUFEZEICHEN-SYMBOL
64	INFO-SYMBOL

Die übliche Reihenfolge der Parameter ist MELDUNG, TYP, TITEL. Die Meldung oder meinetwegen auch Nachricht ist das, was dem Anwender mitgeteilt werden soll, der Typ ist aus der Tabelle zu entnehmen und der Titel wird in der Titelleiste des Meldungsfensters angezeigt.

Sehen wir uns das an einer vorläufigen Prozedur an, indem wir auf COMMAND1 doppelklicken und folgenden Code einfügen:

```
1.Dim Titel As String
2.    Dim Meldung As String
3.    Titel = "Nachfrage"
4.    Meldung = "Willst du das
  Spiel wirklich beenden?"
5.    MsgBox Meldung, 3 + 32,
  Titel
```

In dieser Prozedur werden zuerst die Variablen als Zeichenketten oder STRINGS deklariert, dann den Variablen die Werte zugewiesen und in Zeile 5 die MSGBOX-Anweisung mit dem Typ 3 der Typentabelle formuliert. Dadurch werden die Befehlsschaltflächen Ja, Nein und Abbrechen dargestellt.

Zum gewählten Typ für die Darstellung der Befehlsschaltflächen kann noch der Typenwert für Symbole addiert werden. Der Wert 3 + 32 kann als Summe angegeben werden (35) oder auch als Term und zeigt die Befehlsschaltflächen JA, NEIN und ABBRECHEN sowie ein Fragezeichen an. So eine Kombination eignet sich selbstverständlich, um auf eine Frage hinzuweisen. So weit klappt ja alles, aber es muss auch darauf reagiert werden, wenn eine Schaltfläche angeklickt wird. Die Click-Ereignisse für Message-Boxes sind vordefiniert, siehe Tabelle. Der entsprechende Wert wird in einer Variablen gespeichert und abgefragt.

Typ	Beschreibung
1	OK wurde angeklickt
2	Abbrechen wurde angeklickt oder [ESC]
3	Abbrechen wurde angeklickt
4	Wiederholen wurde angeklickt
5	Ignorieren wurde angeklickt
6	Ja wurde angeklickt
7	Nein wurde angeklickt

Für diesen Zweck wird die Message-Box als Funktion eingesetzt. Es wird also nicht nur eine Mitteilung gemacht (Message-Box als Anweisung), sondern eine Weiterverarbeitung der Klicks vorgenommen.

Du löschst die Befehlsschaltfläche sowie die gesamte eben formulierte Prozedur

Weiter geht es mit einem Doppelklick auf cmdSchließen.

Der einzugebende Code lautet:

```
1.Private Sub
  cmdSchließen_Click()
2.Dim Meldung As String
3.Dim Titel As String
4.Dim Typ As Byte
5.Dim Schaltfläche As Byte
6.Meldung = " Willst du das
  Spiel wirklich beenden?"
7.Titel = "Sicherheitsanfrage"
8.Typ = 3 + 32 + 0
9.Schaltfläche =
  MsgBox(Meldung, Typ, Titel)
10.If Schaltfläche = 6 Then
11.    End
12.ElseIf Schaltfläche = 7
  Then
13.    MsgBox "Dann eben
  nicht!", vbOKOnly
14.End If
15.End Sub
```

Innerhalb der Prozedur CMDSCHLIEßEN muss die Abfrage eingerichtet werden. Nachdem die Variablen deklariert wurden und der Typ festgelegt ist, wird der Variablen SCHALTFLÄCHE in Zeile 9 der Wert der Message-Box übergeben. In Zeile 10–13 werden die Schaltflächen-Klicks ausgewertet, wobei in Zeile 12 eine weitere Message-Box als Anweisung eingefügt ist. Es wurde anstatt des Wertes 0 für den Typ (SCHALTFLÄCHE OK) eine Visual Basic-Konstante eingesetzt. Davon gibt es eine ganze Menge, die genauso wie der Zahlenwert verwendet werden können. Auch für diejenigen, die kein Englisch können, ist die Konstante manchmal verständlicher als der reine Zahlenwert

Kontrollstrukturen

In jedem Programm müssen irgendwann Entscheidungen getroffen oder Programmabläufe wiederholt werden. Für solche Operationen stellt Visual Basic eine ganze Reihe von Lösungsmöglichkeiten zur Verfügung, für jeden besonderen Zweck.

GOTO

Aus dem Wort ergibt sich schon der Sinn dieser Anweisung: Gehe zu. Dieser Befehl ist eine unbedingte Anweisung. Er stammt noch aus alten Basic-Zeiten, als Programme mit Zeilennummern versehen waren. Goto 210 bedeutetet: gehe zu Zeile 210 und mache dort weiter. Heute wird dieser Befehl nur noch selten verwendet, meistens bei der Fehlerbearbeitung. Fehlerbehandlung ist in sehr vielen Prozeduren angebracht. Dringend erforderlich ist sie, wenn Benutzereingaben vorkommen, denn Eingabefehler sind nicht selten. Sehen wir uns das an einem Beispiel an:

```
1.Private Sub
  IrgendeineProzedur()
2.  On Error Goto
  Fehlerbehandlung
  3.  Dim ....
  4.  Code
  5....
  6....
  7.  Exit Sub
  8.Fehlerbehandlung:
  9.  Dim Fehler As String
  10.  Fehler = "Es ist ein
    Fehler aufgetreten!" +
    Chr$(13) + Chr$(10)
  11.  Fehler = Fehler +
    "Fehlernummer: " +
    Str(Err.Number) + Chr$(13) +
    Chr$(10)
  12.  Fehler = Fehler + "Text:
    " + Err.Description
  13.  MsgBox Fehler
  14.End Sub
```

On Error *Goto* Fehlerbehandlung

Innerhalb einer Prozedur wird eine Sprungmarke gesetzt. Du gibst ihr einen Namen und setzt einen Doppelpunkt dahinter.

Fehlerbehandlung:

Danach folgen die Anweisungen, die bei Auftreten eines Fehlers ausgeführt werden sollen. Meistens wird eine MessageBox mit dem ermittelten Fehler ausgegeben. In einem unserer Programme wird eine Fehlerbehandlungsroutine programmiert.

Fehlercode (Err.Number)	**Beschreibung (Err. Description)**
5	Unzulässiger Prozeduraufruf
7	nicht genügend Speicher
13	Typen unverträglich
53	Datei nicht gefunden
55	Datei bereits geöffnet
76	Pfad nicht gefunden
321	Ungültiges Dateiformat
31037	Fehler beim Laden einer Datei

Nach Aufruf der Prozedur 'merkt' sich Visual Basic, dass eine Fehlerroutine angelegt ist, (Zeile 2: bei Fehler gehe zu Fehlerbehandlung) und fährt mit dem Bearbeiten des Codes fort. Tritt kein Problem auf, wird in Zeile 7 mit der Anweisung EXIT SUB – Verlasse die Prozedur – die Prozedur beendet. Tritt ein Fehler auf, wird zur Sprungmarke gesprungen und der Fehlerbehandlungsteil abgearbeitet. In diesem Fall wird in einem Mitteilungsfenster der Fehler beschrieben, wobei man wissen muss, dass es unzählige Fehlernummern mit den entsprechenden Beschreibungen gibt, für jeden Sonderfall eine eigene Fehlernummer.

Ab Zeile 10 wird die Ausgabe für das Mitteilungsfenster geschrieben. Näheres dazu findest du im Abschnitt Mitteilungsfenster auf Seite 26.

Entscheidungen

Im einfachsten Fall ist eine Entscheidung eine Ja/Nein-Entscheidung. "Willst du das Programm fortsetzen? Ja, Nein?". Wenn Nein, dann Ende des Programms, wenn ja, dann Neustart.

If-Strukturen

Wenn mehrere Auswahlmöglichkeiten zur Entscheidung anstehen, wird meistens die IF/THEN-Struktur verwendet. Die einfachste Anwendung ist die Form

```
If Bedingung Then Anweisung
```

Sehen wir uns das genauer an. Die Bedingung kann ein Vergleich sein, etwa:

```
If A = 13 Then
Hintergrundfarbe = rot
```

Trifft die Bedingung zu, ist also wahr, wird die Anweisung ausgeführt, sonst nicht. Du kannst aber auch anweisen, was sonst geschehen soll:

```
If A = 13 Then
  Hintergrundfarbe = rot
Else
  Hintergrundfarbe = grün
End If
```

Trifft die Bedingung zu, wird die Hintergrundfarbe rot werden, sonst (Else) wird sie grün.

Auch bei einfachen Entscheidungen bietet sich die Blockstruktur an. Sie ist übersichtlicher.

Vielfach besteht die Wahl zwischen mehreren Bedingungen. Natürlich kann für jede Bedingung eine IF/THEN-Abfrage geschrieben werden, aber es geht einfacher. Die Abfrage wird um ELSEIF erweitert, was soviel heisst wie *SonstWenn*.

Wenn A = 13 ist, dann Hintergrundfarbe = rot, sonst wenn A = 20 ist, dann Hintergrundfarbe = blau, sonst wenn A = 30 ist, dann Hintergrundfarbe = gelb, sonst, wenn keine Bedingung zutrifft, dann Hintergrundfarbe = grün:

```
If A = 13 Then
  Hintergrundfarbe = rot
ElseIf A = 20 Then
  Hintergrundfarbe = blau
ElseIf A = 30 Then
  Hintergrundfarbe = gelb
Else
  Hintergrundfarbe = grün
End If
```

Ein beliebter Fehler ist, die Anweisung **End If** am Ende der Struktur zu vergessen. Visual Basic weist dich mit einer Fehlermeldung darauf hin.

Select/Case-Struktur

Eine Variable kann unterschiedliche Werte haben. Soll die Variable auf ihren Wert geprüft werden, welchen Wert sie hat, setzt man die SELECT/CASE-Anweisung ein. SELECT bedeutet *Auswahl* und CASE *im Fall*. Durchsuchst du z.B. eine Adressenliste nach Namen, wird in der SELECT/CASE-Anweisung jede Eingabe in ein Textfeld und jede Auswahl aus einer Liste geprüft. Trifft eine Auswahl zu, wird die Anweisung nach der Auswahl ausgeführt, andernfalls wird ähnlich wie bei IF/THEN die CASE ELSE-Anweisung aktiviert.

```
Select Case Name
  Case "Erwin„
    Bezeichnungsfeld.Caption =
"Erwin ist mein Freund."
  Case "Ursula"
    Bezeichnungsfeld.Caption =
"Ich mag Ursula nicht. Die ist
zickig!"
  Case "Torsten"
    Bezeichnungsfeld.Caption =
"Torsten ist ein guter
Fussballspieler."
  Case Else
    Bezeichnungsfeld.Caption =
"Ich habe leider keine
Freunde."
End Select
```

Es können Zahlen und Zahlenbereiche geprüft werden. Visual Basic prüft, ob ein Zahlenwert kleiner als 5 oder größer als 5 ist.

Einen Zahlenbereich zwischen 50 und 90 prüft man mit der Anweisung Case 50 To 90.

```
Select Case Zahlenwert
  Case < 5
'Visual Basic setzt automatisch
Case Is < 5 ein
    Print "Das ist sehr wenig!"
  Case > 5
'Visual Basic setzt automatisch
Case Is > 5 ein
    Print "Das ist etwas mehr!"
  Case Else
    Print "Das ist Nichts!"
End Select
```

Schleifen

Häufig müssen Anweisungen mehrmals ausgeführt werden, z.B. wenn eine Liste nach einem Suchbegriff durchsucht werden muss. Visual Basic hat mehrere Möglichkeiten, um solche Abläufe zu automatisieren.

Der Aufbau einer Schleife sieht so aus: es wird mit Startwerten begonnen, dann werden die notwendigen Anweisungen ausgeführt, und die Schleife beginnt wieder von vorn. Visual Basic muss nur wissen, wo die Schleife beginnt, wo sie endet und wann sie beendet werden soll.

For-Next-Schleife

Die Schleife wird mit dem Schlüsselwort **For** eingeleitet und mit **Next** beendet.

```
For Zähler = Startwert To
Endwert Step Schrittweite
Anweisungen
Next
```

(Die ersten beiden Zeilen gehören in eine Zeile.)

Als Zähler wird eine Variable eingesetzt, in der die Anzahl der Durchläufe der Schleife gespeichert werden. Es hat sich eingebürgert, dafür den Buchstaben i zu verwenden. Du kannst aber auch jede andere Bezeichnung benutzen.

```
For i 1 To 10 Step 2
  Print i, „Visual Basic"
Next
```

Dieser Code gibt die Anweisung, mit dem Wert 1 zu beginnen und mit 10 zu enden. Dabei wird die Schrittweite 2 gezählt, also 1, 3, 5, 9, 11.

Führst du diese Prozedur aus, setzt Visual Basic eine Liste auf den Bildschirm, die folgende Punkte enthält:

1 VISUAL BASIC

3 VISUAL BASIC

5 VISUAL BASIC

7 VISUAL BASIC

9 VISUAL BASIC

Da 11 größer ist als 10, wird die Schleife bei 9 beendet. Dabei besagt diese Anweisung folgendes: schreibe den Wert I und die Zeichenkette VISUAL BASIC auf den Bildschirm. Die Prozedur kommt beim Schlüsselwort **Next** an, was bewirkt, dass Visual Basic wieder an den Anfang der Schleife springt und dort prüft, ob der Endwert schon erreicht ist, um die Anweisungen gegebenenfalls nochmals auszuführen. Wird keine Schrittweite angegeben, zählt das Programm ganz normal hoch.

Soll rückwärts gezählt werden, ist eine Schrittweite -1 nötig

```
For i = 10 To 0 Step -1
Print i „Visual Basic"
Next
```

Für den Startwert und den Endwert können auch Variablen benutzt oder berechnete numerische Werte eingesetzt werden. Eine Variable wie

```
For Zählen = 3 * Wert + 20 To
100/10
```

ist ziemlich unsinnig, aber möglich.

While-Wend-Schleifen

FOR-NEXT-Schleifen haben einen Nachteil: man muss vorher wissen, wie oft die Schleife durchlaufen werden soll. Mit einer WHILE-WEND-Schleife kann man das Schleifenende von einer Bedingung abhängig machen. Ist die Bedingung erfüllt, wird die Schleife verlassen.

```
While Bedingung
  Anweisungen
Wend
```

WHILE heisst so viel wie während, also solange. Solange die Bedingung erfüllt ist, werden die Anweisungen ausgeführt.

```
While
  I < 10
Print i
i = i + 1
Wend
```

In dieser Prozedur hat die Variable i den Wert 0. Zu Beginn der Schleife wird geprüft, ob die Variable kleiner als 10 (I < 10) ist. Wenn ja, d.h. wenn die Aussage wahr ist, dann wird i geschrieben und der Wert um 1 erhöht. Mit der Anweisung WEND, also zurückkehren, wird wieder an den Anfang der Schleife gesprungen und erneut geprüft. Diese Schleife ist etwas aus der Mode gekommen, weil es heute leistungsfähigere Exemplare dieser Art gibt.

Do-Loop-Schleifen

Während man die FOR-NEXT-Schleife als Zählschleife bezeichnet, kann die DO-LOOP-Schleife als Bedingungsschleife bezeichnet werden. Die Bedingungen können am Ende oder auch am Anfang der Schleife getestet werden.

```
1.Do While Bedingung
2.Do Until Bedingung
    Anweisungen
  Loop
```

Beim Testen der Abbruchbedingung hat man die Möglichkeit, entweder Zeile 1 oder Zeile 2 einzusetzen, je nachdem, welchen Zweck man erreichen will. **`Do While`** bedeutet, dass die Schleife so lange ausgeführt wird, *wie* die Bedingung erfüllt ist. **`Do Until`** bedeutet, dass die Schleife so lange ausgeführt, *bis* die Bedingung erfüllt ist. Ob du die While- oder Until-Version bevorzugst, ist Geschmackssache. Es muss immer nur die Bedingung entsprechend geschrieben sein.

```
Do
Anweisungen
1.Loop While Bedingung
Loop Until Bedingung
```

Ob du allerdings am Ende oder am Anfang einer Schleife testest, ist nicht egal, denn der Test am Ende einer Schleife bedeutet, dass die Schleife erst einmal durchlaufen werden muss.

Wenn in einer Datei ein Name gesucht werden soll, ist es natürlich sinnvoll, den Abbruch zu Beginn zu testen, denn wenn der Name gar nicht vorhanden ist, sucht die Prozedur endlos weiter. Irgendwann hängt sie sich dann auf.

```
  Do Until „Klaus“ Or
Dateiende
  Eingaben in ein Textfeld
Loop
```

Diese Schleife wird durchlaufen, bis entweder ‘Klaus’ gefunden wird oder das Dateiende erreicht ist.

Wichtig ist, dass du dir immer ganz klar über den Schleifenablauf bist, damit keine unliebsamen Überraschungen passieren – entweder dass die Schleife gar nicht durchlaufen wird, weil die Abbruchbedingungen schon vorher erfüllt sind, oder dass die Schleife nie aufhört, weil die Abbruchbedingung nicht erreicht wird.

Dateiverwaltung

Im Abschnitt Grafik auf Seite 40 werden Dateiverwaltungs-Operationen verwendet. Es werden Daten gespeichert, bzw. geladen. Im Rahmen dieses Heftes kann nicht auf den sehr umfangreichen Bereich der Dateiverwaltung eingegangen werden. Es soll nur das Öffnen und Schließen einer Datei beschrieben werden.

Grundsätzlich gibt es die Schritte:	
OPEN-ANWEISUNG	eine Datei wird geöffnet
INPUT-ANWEISUNG	Daten werden gelesen
PRINT-ANWEISUNG	Daten werden geschrieben
CLOSE-ANWEISUNG	eine Datei wird geschlossen

Datei lesen

Es können gleichzeitig mehrere Dateien geöffnet sein. Daher bekommt jede Datei eine Nummer. Nun kannst du dir die Nummern merken oder Visual Basic nach der jeweils freien Nummer suchen lassen. In Zeile 4 der folgenden Prozedur steht die Anweisung für Visual Basic, nach der freien Nummer zu suchen:

```
DateiNr = FreeFile.
```

In der Variablen Dateiname wird der Pfad zu der Datei gespeichert. Je nachdem, wo du die Datei ablegst, musst du den Pfad ändern. Ist keine Datei vorhanden, erfolgt eine Fehlermeldung.

```
Private Sub cmdÖffnen_Click()
1. Dim DateiNr As Integer
2. Dim DateiName As String
3. Dim Temp As String
4. DateiNr = FreeFile
5. DateiName = "C:\Zahlen.txt"
6. Open DateiName For Input As
DateiNr
7.   Input #DateiNr, a, b, c
8.   txtZahl1.Text = a
9.   txtZahl2.Text = b
10.  txtZahl3.Text = c
11.Close
End Sub
```

In Zeile 6 wird die Datei zum Lesen geöffnet. Wer etwas Englisch kann, versteht die Anweisung sofort. Eingedeutscht heißt das: Öffne die Datei mit dem beschriebenen Dateinamen zum Lesen mit der nächsten freien Dateinummer.

In Zeile 7 werden die in den Variablen A, B und C abgelegten Daten eingelesen und in den Textfeldern hezeigt. In Zeile 11 wird die Datei wieder geschlossen – vergiss nicht den CLOSE-Befehl!

Hier haben wir definierte Variablen verwendet. Ist die Größe einer Datei nicht bekannt, muss das Programm sie bis zu ihrem Ende lesen.

Die Anweisung dafür lautet wie folgt:

```
1. Do While Not EOF(DateiNr)
2.   Line Input #DateiNr, Temp
3.   Text1.Text = Text1.Text &
     Temp & vbCrLf
4. Loop
```

In einer Do-Schleife wird die Datei so lange gelesen, wie EOF (**E**nd **O**f **F**ile = Ende der Datei) nicht erreicht ist. Dann werden die einzelnen Zeilen der Datei mit der Anweisung LINE INPUT gelesen und in einer Zwischenvariablen TEMP gespeichert. In Zeile 3 wird in einem Textfeld jede einzelne Zeile dargestellt und mit einem Steuerzeichen für den Zeilenabschluss versehen. Das entspricht einer EINGABE nach jeder Zeile und ist eine Konstante für die Anweisungen CHR(10) & CHR(13), die wir auf Seite 38 verwenden.

Datei speichern

In Zeile 5 der folgenden Prozedur wird die benannte Datei zum Speichern bzw. Schreiben geöffnet. Die Dateinummer wurde mit FREEFILE ermittelt. In Zeile 6 folgen dem Schreibbefehl PRINT nach der Dateinummer die Variablen, wonach die Datei geschlossen wird. Das ist – wie gesagt – nur ein ganz kleiner Abriss zur Dateiverwaltung. Für mehr ist leider kein Platz.

```
Private Sub cmdSpeichern_
Click()
1. Dim DateiNr As Integer
2. Dim DateiName As String
3. DateiName = "C:\Zahlen.txt"
4. DateiNr = FreeFile
5. Open DateiName For Output As
DateiNr
6. Print #DateiNr, a, b, c
7. Close
End Sub
```

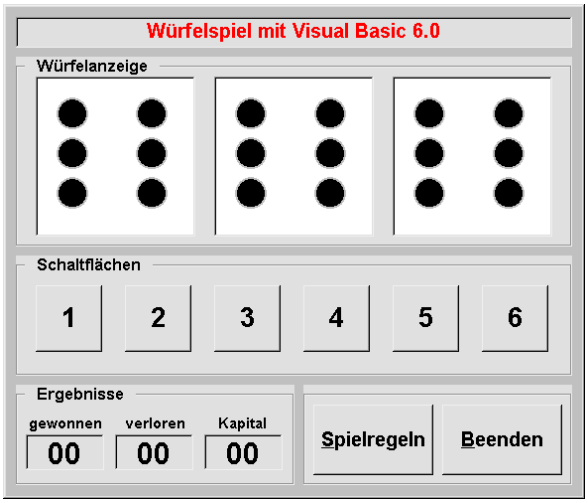
Zweites Programmprojekt

Auch bei dieser Prozedur steht nicht die Klasse des Programms im Vordergrund, sondern die Programmiertechniken. Grundlage ist unser Würfelspiel. Diesmal sollen drei Würfel eingesetzt werden. Dafür muss die Form natürlich größer aufgezogen werden.

Du startest Visual Basic und wählst wiederum die STANDARD.EXE

Oberfläche

Einen Vorschlag zur Oberfläche siehst du im Bild. Die Kodierung erarbeiten wir anschließend.



Für diese Oberfläche werden Rahmenfelder oder FRAMES eingesetzt. Diese haben programmtechnisch keinen Nutzen – sie dienen schlicht zu Gestaltungszwecken. Mit ihrer Hilfe lassen sich Gruppierungen von Objekten gestalterisch vornehmen. In unserem Beispiel fügen wir die Befehlsschaltflächen für die Zahlen oder die drei Bildfelder für die Würfelanzeigen in Rahmenfelder ein.

Form.		
HEIGHT	Höhe	`ca. 5950`
WIDTH	Breite	`7040`
CAPTION	Beschriftung	""
CONTROLBOX	System-Menüfeld	`False`
BORDERSTYLE	Rahmenform	`Fester Dialog`
BACKCOLOR	Hintergrund-farbe	auswählen

Wie gesagt wird nur ein optischer Effekt erzeugt. Auf das Programm hat das keinen Einfluss, und wie du siehst, gibt es auch keine besonderen Eigenschaften. Die Eigenschaft CAPTION spielt die größte Rolle – damit hat es sich auch schon. Übrigens kannst du hier einen Schönheitstrick nutzen – damit die CAPTION nicht so sehr an der Rahmenlinie klebt, kannst du vor und nach dem Wert für sie ein Leerzeichen eingeben.

Hast du die Form entsprechend der Tabelle eingerichtet und startest sie mit F5, stellst du fest, dass sie keine Titelleiste hat: wenn du den Wert für CONTROLBOX auf FALSE setzt und BORDER-STYLE auf FESTER DIALOG, erhältst du eine Form, die nur aus einer einfachen Fläche besteht. Das bedeutet natürlich auch, dass die Schalt-flächen für MINIMIEREN, MAXIMIEREN und SCHLIEßEN fehlen.

Einen Titel soll das Projekt natürlich haben. Also setzen wir ein Bezeichnungsfeld oder LABEL ein.

Label1		
CAPTION	Beschriftung	`Würfelspiel mit Visual Basic 6.0`
BORDER-STYLE	Randform	Fest Einfach
ALIGNMENT	Ausrichtung	Zentriert
FONT	Schrift	aussuchen
FORECOLOR	Schriftfarbe	aus Palette aussuchen

Unsere Arbeitsschritte sehen also so aus:

- Zunächst speichern wir unser Projekt über DATEI | PROJEKT SPEICHERN.
- Dann erstellen wir die Rahmen über DREIWÜRFEL.FRM ...
- ... und die Würfel über DREIWÜRFEL.VBP ...

und ziehen endlich das Rahmenfeld auf.

Soll ein Rahmenfeld sich wie ein Behälter für die anderen Objekte verhalten, müssen Objekte *im*(!) Rahmenfeld aufgezogen werden. Setzt man sie von außen hinein, z.B. über Doppelklick auf das Werkzeug und Drag & Drop, entsteht das Problem, dass die angeordneten Objekte beim Verschieben des Rahmenfeldes an ihrem Platz liegen bleiben. Ziehst du die Objekte aber im Rahmenfeld auf, werden sie mitverschoben.

1. Zunächst ziehst du den ersten Rahmen auf:

Frame1		
CAPTION	Beschriftung	**`Würfelanzeige`** (Leerzeichen nicht vergessen!)

2. Nun ziehst du im FRAME1 drei Bildfelder auf:

Picture 1 – 3		
NAME	Name	**`Würfel1, Würfel2, Würfel3`**
BACKCOLOR	Hintergrundfarbe	aus Palette aussuchen

3. Dann ordnest du im ersten Bildfeld SHAPE1 bis SHAPE7 an, so wie wir das im vorhergehenden Programm 'Würfeln' getan haben, ...
4. und wiederholst den Prozess in den anderen Bildfeldern.
5. Die Namen für die Figuren sollen einen Bezug zum jeweiligen Würfel haben: im ersten Würfel bekommt SHAPE1 den Namen **`Auge1_1`**, SHAPE2 den Namen **`Auge2_1`** usw. Im zweiten Würfel heißen die Namen dann **`Auge1_2`**, **`Auge2_2`** usw. und im dritten Würfel **`Auge1_3`**, **`Auge2_3`** usw. Das ganze sieht so aus:

```
Auge1_1   Auge1_2   Auge1_3
Auge2_1   Auge2_2   Auge2_3
Auge3_1   Auge3_2   Auge3_3
Auge4_1   Auge4_2   Auge4_3
Auge5_1   Auge5_2   Auge5_3
Auge6_1   Auge6_2   Auge6_3
Auge7_1   Auge7_2   Auge7_3
```

6. Nun legst du das zweite Rahmenfeld an, ...

Frame2		
CAPTION	Beschriftung	**`Schaltflächen`**

7. ...ziehst sechs Befehlsschaltflächen auf, ...
8. ... setzt CAPTIONS wie angezeigt ...
9. ... und gibst ihnen die Namen **`cmdEins`**, **`cmdZwei`**, **`cmdDrei`**, **`cmdVier`**, **`cmdFünf`**, **`cmdSechs`**
10. Fehlt noch das dritte Rahmenfeld:

Frame3		
CAPTION	Beschriftung	**`Ergebnisse`**

11. Hier setzt du drei Bezeichnungsfelder ein:

Labels		
NAME	Name	**`lblGewonnen lblVerloren lblKapital`**
CAPTION	Beschriftung	**`00`**
FONT	Schrift	aussuchen
ALIGNMENT	Ausrichtung	zentriert
BORDER-STYLE	Randform	Fest Einfach
FORECOLOR	Schriftfarbe	aussuchen

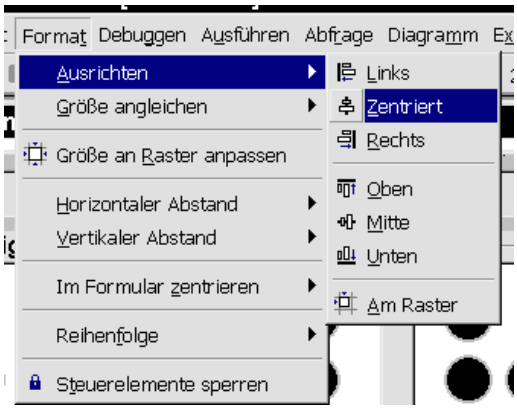
12. Für die Beschriftung der Bezeichnungsfelder legst du wiederum drei Bezeichnungsfelder an und setzt als CAPTIONS: **`'gewonnen'`**, **`'verloren'`**, **`'Kapital'`**.
13. Felt noch noch ein viertes Rahmenfeld für die Befehlsschaltflächen **`Spielregeln`** und **`Beenden:`**

Befehlsschaltflächen		
NAME	Name	**`cmdSpielregeln`**
CAPTION	Beschriftung	**`&Spielregeln`**
NAME	Name	**`cmdBeenden`**
CAPTION	Beschriftung	**`&Beenden`**

Wenn du den Wert der Eigenschaft Caption einträgst, vergiss nicht das sogenannte 'Kaufmännisch Und' (&). Dieses Zeichen veranlasst Windows-Programme dazu, den Buchstaben, vor dem es steht, zu unterstreichen. Damit kannst du das Klick-Ereignis mit der Tastenkombination ALT + (unterstrichener) BUCHSTABE auslösen. Das Klick-Ereignis **Beenden** wird also mit ALT + B ausgelöst.

Willst du einzelne Objekte aufeinander ausrichten, kannst du das über Drag & Drop erledigen. Genauer geht das, wenn du die Werte LEFT, TOP, HEIGHT, WIDTH unmittelbar änderst. Du musst dabei nicht immer den alten Wert löschen, sondern kannst so lange neue Werte eingeben, bis du zufrieden bist; der alte Wert wird überschrieben.

Sollen Objekte auf einer Linie ausgerichtet werden, empfiehlt sich FORMAT | AUSRICHTEN:



Vergiss nicht, dass du bei gehaltener STRG-Taste mehrere Objekte gleichzeig markieren kannst.

Vermutlich musst du die Sache etliche Male ausprobieren, bis du die Wirkungsweise verstanden hast. Ausgerichtet wird immer an dem Objekt, das die dunkleren Anfasser hat – meistens also an dem Objekt, das zuletzt angeklickt wurde. Du musst dir also vorher über die Reihenfolge klar sein.

Code

Die Oberfläche steht nun also – fehlt noch der gesamte Code, der sie ans Laufen bringt.

1. Du doppelklickst auf auf CMDBEENDEN ...
2. ... und trägst im Prozedurrahmen CMDBEENDEN_CLICK die Anweisung **End** ein.
3. Es müssen für jeden Würfel Zufallszahlen erzeugt werden.
4. Also doppelklickst du in eine freie Stelle der Form ...
5. ... und gibst im Prozedurrahmen FORM_LOAD die Anweisung **Randomize** ein – damit wird beim Start des Programms, also wenn die Form in den Arbeitsspeicher geladen wird, der 'Zufallsgenerator' aktiviert.
6. Nun legst du die Prozedur **Private Sub Computer** an.
7. Die folgende Prozedur zeigt den Code, mit dem in einer Schleife für jeden Würfel Zufallszahlen erzeugt werden:

```
1 Private Sub Computer()
2 Dim i As Byte
3 For i = 1 To 3
4 Zufallszahl(i) = Int(Rnd *
  6) + 1
5 Next i
6 End Sub
```

Bevor der Code erläutert wird, müssen wir den Begriff ARRAY erklären.

In Programmen tauchen vielfach Gruppen von Variablen auf, die gleichartig sind. In unserem Fall taucht für jeden Würfel die Variable ZUFALLSZAHL auf, die grundsätzlich keine Unterschiede aufweist. Also wird hier ein *Variablenfeld* angelegt, in dem jede gleichartige Variable einen Index zur Unterscheidung bekommt. So ein Variablenfeld nennt man ein ARRAY. Der Index ist in Klammern an die Variable angefügt; und über ihn werden die Variablen angesprochen.

7. Hier sieht die Sache so aus:

```
Zufallszahl(1)
Zufallszahl(2)
Zufallszahl(3)
```

8. Wie deklariert man nun ein Array? Ähnlich wie einfache Variablen:

```
Dim Zufallszahl(3)
```

9. Da die Variable auch in anderen Prozeduren benutzt wird, deklarierst du sie im Allgemeinteil am Anfang des Code, also hinter **Option Explicit.**
10. FOR-NEXT-Schleifen sahen wir uns im Abschnitt „Kontrollstrukturen“ auf Seite 28 an. Innerhalb der Schleife werden drei

Zufallszahlen erzeugt und über den Index jeweils der Variablen **Zufallszahl()** zugeordnet. Es können auch zwei oder drei gleiche Zahlen erzeugt werden. Soll der Spieler die Zahlen des Computers sehen, kannst du die Werte im sogenannte Direktfenster anzeigen lassen.

11. Zwischen Zeile 4 und Zeile 5 in der obigen Prozedur gibst du folgende Zeile ein:
```
Debug.Print Zufallszahl(i)
```

12. In der Prozedur FORM_LOAD gibst du den Aufruf der Prozedur COMPUTER ein:
```
Private Sub Form_Load()
Randomize
Computer
End Sub
```
Nun werden nach jedem Programmstart im Direktfenster die drei vom Computer erzeugten Zahlen angezeigt. Mit der DEBUG.PRINT-Methode kannst du in diesem Fenster jede Variable anzeigen und damit ganz einfach überprüfen, welche Werte Variablen haben.

13. Manchmal ist das Direktfenster unübersichtlich. Für diese Fälle lege ich mir ein Bezeichnungsfeld an und lasse die Variable über die Eigenschaft CAPTION anzeigen.

14. Hast du dich überzeugt, dass das Programm bisher einwandfrei läuft, löscht du die Anweisung **Debug.Print** wieder – den Aufruf COMPUTER dagegen lässt du stehen.

15. Jetzt werden diese Zahlen auf die Würfelanzeige übertragen – hier mit der Methode SELECT-CASE. Das Prinzip siehst du im Abschnitt „Kontrollstrukturen“:
```
Private Sub Anzeige1()
    Select Case Zufallszahl(1)
        Case 1
            Eins1
        Case 2
            Zwei1
        Case 3
            Drei1
        Case 4
            Vier1
        Case 5
            Fünf1
        Case 6
            Sechs1
    End Select
End Sub
```

16. Entsprechend der Zufallszahl steuert das Programm die Prozedur für die Augenanzeige an. Dort werden die Augen mit der bekannten Eigenschaft **Visible = True/False** dargestellt:
```
Private Sub Eins1()
    Auge1_1.Visible = False
    Auge2_1.Visible = False
    Auge3_1.Visible = False
    Auge4_1.Visible = False
    Auge5_1.Visible = False
    Auge6_1.Visible = False
    Auge7_1.Visible = True
End Sub
Private Sub Zwei1()
    Auge1_1.Visible = True
    Auge2_1.Visible = False
    Auge3_1.Visible = False
    Auge4_1.Visible = False
    Auge5_1.Visible = False
    Auge6_1.Visible = True
    Auge7_1.Visible = False
End Sub
Private Sub Drei1()
    Auge1_1.Visible = True
    Auge2_1.Visible = False
    Auge3_1.Visible = False
    Auge4_1.Visible = False
    Auge5_1.Visible = False
    Auge6_1.Visible = True
    Auge7_1.Visible = True
End Sub
Private Sub Vier1()
    Auge1_1.Visible = True
    Auge2_1.Visible = True
    Auge3_1.Visible = False
    Auge4_1.Visible = False
    Auge5_1.Visible = True
    Auge6_1.Visible = True
    Auge7_1.Visible = False
End Sub
Private Sub Fünf1()
    Auge1_1.Visible = True
    Auge2_1.Visible = True
    Auge3_1.Visible = False
    Auge4_1.Visible = False
    Auge5_1.Visible = True
    Auge6_1.Visible = True
    Auge7_1.Visible = True
End Sub
Private Sub Sechs1()
```

```
    Auge1_1.Visible = True
    Auge2_1.Visible = True
    Auge3_1.Visible = True
    Auge4_1.Visible = True
    Auge5_1.Visible = True
    Auge6_1.Visible = True
    Auge7_1.Visible = False
End Sub
```

17. Diese Prozeduren müssen für Anzeige2 und Anzeige3 wiederholt werden, wie du aus den beiden folgenden Prozeduren ersiehst:

```
Private Sub Anzeige2
  Select Case Zufallszahl(2)
    Case1
      Eins2
      usw.
End Select


Private Sub Anzeige3
  Select Case Zufallszahl(3)
    Case1
    Eins3
    usw....
End Select
```

18. Danach programmierst du die Anzeigen für die Augen der beiden anderen Würfel.

19. In der Click-Prozedur wird eine Variable SPIELER gebraucht. Diese wollen wir im Allgemeinteil deklarieren, um sie allen Prozeduren zugänglich zu machen:

```
Dim Spieler As Byte
```

20. Nun doppelklickst du auf die Befehlsschaltfläche CMDEINS und gibst folgendes ein:

```
Private Sub cmdEins_Click()
    Spieler = 1
    Computer
End Sub
```

21. Das wiederholst du für CMDZWEI ...

```
Private Sub cmdZwei_Click()
    Spieler = 2
    Computer
End Sub
```

22. und so weiter bis CMDSECHS.

In den Prozeduren werden der Variablen die unterschiedlichen Werte zugewiesen und danach die Prozedur COMPUTER aufgerufen. Dort werden die Zufallszahlen erzeugt und mit den Prozeduren ANZEIGE1, ANZEIGE2 und ANZEIGE3 angezeigt.

23. Bis auf die Auswertung der Eingaben müsste alles planmäßig funktionieren. Tippfehler in den Listings werden allerdings nicht als Fehler angezeigt, sondern verfälschen nur die Anzeigen. Daher solltest du mehrfach überprüfen, ob sich keine Tippfehler eingeschlichen haben – mit folgendem Code:

```
1  Private Sub Auswertung()
2    Static gewonnen As Byte
3    Static verloren As Byte
4    Static Kapital As Integer
5    Dim i As Byte
6    For i = 1 To 3
7      If Spieler =
   Zufallszahl(i) Then
8      gewonnen = gewonnen + 1
9      lblGewonnen.Caption =
   gewonnen
10     Kapital = Kapital + 1
11     lblKapital.Caption =
   Kapital
12   End If
13   Next
14   If Spieler <>
   Zufallszahl(1) And _
15    Spieler <> Zufallszahl(2)
   And _
16    Spieler <> Zufallszahl(3)
   Then
17    verloren = verloren + 1
18    lblVerloren.Caption =
   verloren
19    Kapital = Kapital - 1
20     blKapital.Caption =
   Kapital
21   End If
22 End Sub
```

Bis zur Zeile 13 bietet der Code nichts Neues. Innerhalb der Schleife wird geprüft, ob die Variable Spieler den gleichen Wert wie die Variable Zufallszahl hat.

24. Es war die Absicht, dass dem Spieler bei gleichen Würfelanzeigen mehrfach das Kapital erhöht wird. Damit er eine größere

Chance gegen den Computer hat, sollen die Verluste nicht mehrfach zählen:

```
1  For i = 1 To 3
2    If Spieler = Zufall
     zahl(i) Then
3      gewonnen = gewonnen + 1
4      lblGewonnen.Caption =
       gewonnen
5      Kapital = Kapital + 1
6      lblKapital.Caption =
       Kapital
7    ElseIf Spieler <> Zufalls-
     zahl(1) And _
8        Spieler <> Zufall
         zahl(2) And _
9          Spieler <> Zufalls-
           zahl(3) Then
10     verloren = verloren + 1
11     lblVerloren.Caption =
       verloren
12     Kapital = Kapital - 1
13     lblKapital.Caption =
       Kapital
14   End If
15 Next
```

25 In dieser Prozedur wird durch eine andere Programmiertechnik erreicht, dass die Nichttreffer mehrfach hochgezählt werden. Die folgende Prozedur zeigt eine andere Codierweise:

```
1  For i = 1 To 3
2    If Spieler = Zufalls-
     zahl(i) Then
3      gewonnen = gewonnen + 1
4    l  blGewonnen.Caption =
     gewonnen
5      Kapital = Kapital + 1
6      lblKapital.Caption =
       Kapital
7    ElseIf Spieler <> Zufalls-
     zahl(i) Then
8      verloren = verloren + 1
9      lblVerloren.Caption =
       verloren
10     Kapital = Kapital - 1
11     lblKapital.Caption =
       Kapital
12   End If
13Next
```

Für welche Variante du dich entscheidest, bleibt dir überlassen. Man könnte an dieser Stelle beispielsweise eine Auswahlmöglichkeit für Schwierigkeitsgrade einbauen.

26 Zeile 14 – 16 in der Prozedur zur Auswertung, Seite 37, und Zeile 7 – 9 in der ersten der beiden eben angeführten Prozeduren zur Schleifenabfrage auf Seite 38 haben eine Besonderheit. Der Code gehört in eine zusammenhängende Zeile. Die Zeile wird dadurch ziemlich lang und kann nur noch über die Bildlaufleisten eingesehen werden. Visual Basic erkennt den Unterstrich als Trennzeichen einer Zeile und verbindet intern die Zeilenteile wieder. Wichtig ist ein Leerzeichen vor dem Unterstrich. Wenn du die Zeilenteile im Codefenster etwas einrückst, ist besser erkennbar, welche Teile zusammen gehören

27 Zum Abschluss wird die Spielanleitung programmiert. Den Text überlasse ich dir. Eine Möglichkeit ist, eine Mitteilungsbox anzulegen:

```
Private Sub
cmdSpielregeln_Click()
    Dim Titel As String
    Dim Meldung As String
    Titel = "Spielanleitung"
    Meldung = Meldung + "Der
Computer würfelt drei
Zufallszahlen." + Chr$(10) +
Chr$(13)
    Meldung = Meldung + "Du
kannst gegen den Computer
würfeln, " + Chr$(10) +
Chr$(13)
    Meldung = Meldung + "usw."
    MsgBox Meldung, vbOKOnly,
Titel
End Sub
```

Größere Gestaltungsmöglichkeiten hast du, wenn du eine zweite Form verwendest. Dazu musst du wissen, dass es zwei Möglichkeiten gibt, ein Formular zu laden. Über die LOAD-Anweisung wird ein Formular nur in den Speicher geladen, aber nicht angezeigt – mit der SHOW-Methode dagegen wird die Form geladen und angezeigt.

Der umgekehrte Weg, nämlich eine Form wieder los zu werden, verwendet analog zwei Möglichkeiten. Soll eine Form komplett aus dem Arbeitsspeicher entfernt werden, wird sie mit der UNLOAD-Anweisung entladen, ansonsten versteckt man sie nur mit der HIDE-Methode.

Du aktivierst PROJEKT | FORMULAR HINZUFÜGEN und wählst ein Formular.

Wie du siehst, hat Visual Basic verschiedene Dialog-Formulare in seinem Angebot, die auch schon mit Code versehen sind. Die solltest du alle einmal ausprobieren. Im Projektfenster ist die neue Form als FORM2 angezeigt. Wenn Programme klein sind und nur wenige Projektteile enthalten, belasse ich es meistens bei den vorgegebenen Namen. Hier könntest du aber z.B. FRMANLEITUNG als Name eingeben.

Du doppelklickst auf CMDSPIELREGELN und gibst folgenden Code ein:

```
Private Sub
cmdSpielregeln_Click()
    frmAnleitung.Show
End Sub
```

Dann drückst du auf F5 und dann auf SPIELREGELN.

Die Form FRMANLEITUNG kannst du mit deinen Kenntnissen so gestalten, wie du es möchtest. Entweder setzt du die Print-Methode für den Text ein, oder du legst dir ein Textfeld an, oder du verwendest ein Bildfeld oder ... In jedem Fall muss eine Prozedur geschrieben werden, mit der das Fenster wieder geschlossen wird. Also doppelklickst du auf FRMANLEITUNG, wählst ein CLICK-Ereignis und gibst folgenden Code ein:

```
Private Sub Form_Click()
    frmAnleitung.Hide
End Sub
```

Die Spielanleitung wird nur versteckt. Dadurch steht sie dem Spieler immer wieder sehr schnell zur Verfügung. Eine Befehlsschaltfläche mit dem CLICK-Ereignis ist vielleicht offensichtlicher. Zumindest solltest du darauf hinweisen, dass die Form durch einen Klick geschlossen werden kann, z.B. als CAPTION.

Du wählst DATEI | FORM FRMANLEITUNG SPEICHERN UNTER ... und gibst als Name **Anleitung** ein

Zum Abschluss setzen wir noch ein Mitteilungsfenster ein für den Fall, dass das Kapital weniger als Null, bzw. größer als 25 wird. Der Wert soll nur ein Lob für eine gute Leistung sein; du kannst selber festlegen, ab wann gutes Würfeln gelobt werden soll.

Die Abfrage fügst du in unseren Code zur Fehlerüberprüfung auf Seite 37 zwischen Zeile 5 und Zeile 6 ein:

```
If Kapital < 0 Then Alarm
If Kapital > 25 Then Loben
```

Du legst die Prozedur ALARM an:

```
1. Private Sub Alarm()
2.   Dim Titel As String
3.   Dim Meldung As String
4.   Dim Schaltfläche As Byte
5.   Titel = "ALarm"
6.   Meldung = "Du bist leider
       pleite!"
7.   Schaltfläche =
       MsgBox(Meldung, 1 + 16,
       Titel)
8.   If Schaltfläche = 1 Then
9.     End
10.  End If
11. End Sub
```

In der Prozedur ALARM wird ein Mitteilungsfenster als Funktion ausgegeben. In Zeile 7 wird der Variablen SCHALTFLÄCHE ein Wert des Mitteilungsfensters zurückgegeben, der in Zeile 8 abgefragt wird. Wird der Wert 1 zurückgegeben, also die Schaltfläche OK angeklickt, soll das Programm beendet werden. Alle anderen Eingaben haben keine Wirkung. Folglich wird bei Abbrechen zum Spielprogramm zurückgekehrt.

Nun legst du die Prozedur LOBEN an:

```
1.Private Sub Loben()
2.  'blinken
3.  Dim i As Byte
4.  For i = 0 To 15
5.    Form1.BackColor =
  QBColor(i)
6.    Frame1.BackColor =
  QBColor(i)
7.    Frame2.BackColor =
  QBColor(i)
8.    Frame3.BackColor =
  QBColor(i)
9.    Frame4.BackColor =
  QBColor(i)
10.    verz 0.1
11.  Next
12.  Form1.BackColor =
  &H8000000F
13.  Frame1.BackColor =
  &H8000000F
14.  Frame2.BackColor =
  &H8000000F
15.  Frame3.BackColor =
  &H8000000F
16.  Frame4.BackColor =
  &H8000000F
17.End Sub
```

In der Schleife, Zeile 4, werden die Hintergrundfarben der Form sowie der Rahmen fünfzehn Mal mit den Standardfarben versehen. Damit es nicht zu schnell geht, habe ich bei jedem Schleifendurchlauf eine Verzögerung eingebaut:

```
1.Private Sub verz(zeit!)
2.  Dim zeit1 As Variant
3.  zeit1 = Timer
4.  Do
5.    DoEvents
6.  Loop Until (Timer - zeit1
  >zeit)
7.End Sub
```

Danach werden die Farben wieder auf ihren Ursprungswert gesetzt.

Die Prozedur VERZ(ZEIT!)ist ziemlich verzwickt, weswegen ich dir empfehle, sie einfach zu übernehmen – dem unbekannten Autor sei gedankt!

Grafik

Unter Windows muss man sich die gesamte Bildschirmdarstellung als Grafik vorstellen. Auch Text-Ausgaben sind eine Form von Grafik.

Visual Basic bietet grundsätzlich zwei Möglichkeiten, Grafik einzusetzen:

1. Grafik-Steuerelemente mit ihren Eigenschaften und Methoden aus der Werkzeugsammlung
2. GDI-Funktionen (Graphical Device Interface)

Bis auf bestimmte Spezialfälle, wie z.B. Polygone, stehen für die gängige Grafik-Programmierung genügend Visual Basic-Komponenten zur Verfügung.

Grundlagen

Grafikoperationen sind gleichermaßen im Fenster, also in der FORM, auf dem Drucker oder in einem Bildfeld möglich. Für alle drei Komponenten gelten gleiche Methoden.

Methoden

Grafikausgaben beziehen sich grundsätzlich auf ein bestimmtes Objekt, also nicht auf den gesamten Bildschirm. So kann eine Ausgabe mit der Methode PRINT auf einem Drucker erfolgen, aber auch in einem Bildfeld oder in der Form.

Methoden	Beschreibungen
LINE	zeichnet eine Linie oder ein Rechteck
CIRCLE	zeichnet einen Kreis, eine Ellipse oder einen Bogen
PSET	setzt einen Punkt
POINT	ermittelt die Farbe eines Punktes
PRINT	gibt eine Zeichenfolge aus
CLS	löscht den Inhalt des Objektes

1. Du startest Visual Basic, ...
2. gibst der FORM den NAMEN **frmHaupt** ...
3. ... und speicherst sie als GRAFIK1
4. Dann ziehst du in der Form ein großes BILDFELD auf, ...
5. ... und setzt als NAME **picGrafik1**
6. Jetzt erstellst du eine BEFEHLSSCHALTFLÄCHE.
7. Hier setzt du als NAME **cmdSchließen** ...

   ... und als CAPTION `Ende`**.**
8. Du erstellst noch eine Befehlsschaltfläche, ...
9. ... setzt als NAME **cmdZeichnen** ...
10. .... und als CAPTION **Zeichnen**.
11. Dann doppelklickst du auf CMDZEICHNEN ... Der Code lautet:

```
Private Sub
cmdZeichnen_Click()
  picGrafik1.Line (100, 100)-
  (500, 500)
End Sub
```

12. .... drückst auf F5
13. .... und endlich auf ZEICHNEN.

    Nach F5 wird eine Linie von links oben nach rechts unten gezeichnet.
14. Nun doppelklickst du auf CMDSCHLIEßEN, ...
15. setzt **End**,...
16. .... beendest das Programm und vergrößerst das Bildfeld ...
17. .... und drückst wiederum **F5**

Die Linie befindet sich innerhalb des Bildfeldes an derselben Position.

Es taucht die Frage auf, ob die Linie von links oben nach rechts unten oder umgekehrt gezeichnet wird – bei der Arbeitsgeschwindigkeit des Computers kann man nicht erkennen, in welche Richtung gezeichnet wurde. Dazu benötigen wir einige grundlegende Kenntnisse.

Eigenschaften

Schüler kennen ein beliebtes Spiel, das vornehmlich im langweiligen Unterricht gespielt wird: »Schiffe versenken«. Verwendet wird ein kariertes Papier – aus guten Gründen meist aus dem Mathe-Heft.

Jedes Kästchen hat durch die Angaben für die Zeile und die Spalte einen festgelegten Platz. Das 'x' befindet sich in Zeile '4' und Spalte 'd'. Mit der Angabe **4/d** wird der Ort bestimmt, an dem sich eindeutig das 'x' befindet.

	a	b	c	d	e	f	g	h	i	j
1										
2										
3										
4				**x**						
5										
6										
7										
8										
9										
10										

So etwa musst du dir den gesamten Bildschirm, eine FORM oder – wie hier – ein Bildfeld eingeteilt vorstellen, mit dem Unterschied, dass die Einteilungen viel feiner vorgenommen werden können und dass keine Buchstaben verwendet werden.

Visual Basic verwendet standardmäßig eine besondere Maßeinheit: *Twips*. Ein Kästchen ist ein *Twip*. Der Vorteil dieser Maßeinheit liegt darin, dass sie systemunabhängig ist, also auch maßstabsgetreu zeichnet, wenn eine andere Bildschirmauflösung verwendet wird. Das Koordinatensystem ist standardgemäß aufgebaut, also beginnen die Zeilen und Spalten links oben mit dem Wert 0; **`0/0`** ist die linke obere Ecke.

In der Mathematik werden solche Koordinaten mit den Buchstaben x und y bezeichnet. Die Koordinaten bewegen sich auf der y-Achse von oben nach unten und auf der x-Achse von links nach rechts. Diese Darstellung ist für uns etwas ungewohnt. Wir kennen z.B. bei grafischen Darstellungen eher den umgekehrten Weg, dass sich die y-Achse von unten nach oben bewegt. Wie alles in Visual Basic lässt sich auch dieses ändern. Dazu später mehr.

In unserer Prozedur CMDZEICHNEN_CLICK hatten wir folgende Anweisung formuliert:

```
picGrafik1.Line
(100,100) – (500,500).
```

PICGRAFIK1 ist das Bildfeld, und nach einem Punkt als Trennzeichen wird LINE als Methode festgelegt. Dann folgen die Zahlenpaare **`(100,100)`** als Startpunkt und **`(500,500)`** als Endpunkt. Zu Übungszwecken kannst du auch andere Zahlen verwenden.

Wichtig ist, dass die Linie bei **`(x1,y1)`** beginnt und bei **`(x2,y2)`** aufhört.

Nun klickst du das Steuerelement LINE im WERKZEUG-Fenster an und legst die neue Linie irgendwo im Bildfeld ab, indem du den Mauszeiger ins Bildfeld setzt, die linke Maustaste drückst und ziehst.

Im EIGENSCHAFTEN-Fenster findest du u.a. die Eigenschaften X1, X2, Y1 und Y2. Daneben stehen in der Werteliste die entsprechenden Angaben.

Klickst du einen Anfasser der Linie an, kannst du ihre Lage durch Ziehen verändern. Lässt du die Maustaste los, werden die neuen Koordinatenwerte angezeigt. Probiere ein bisschen herum, damit du ein Gefühl dafür bekommst, wie die Lage der Linie im Bildfeld beschrieben wird.

Von den wenigen Eigenschaften einer Linie sind für uns BORDERCOLOR, BORDERSTYLE und BORDERWIDTH interessant. Damit können die *Farbe*, die *Darstellung* und die *Stärke* der Linie eingestellt werden.

Dazu eine kleine Prozedur:

```
Private Sub Form_Load()
    frmHaupt.Height = 7000
    'Me.Height = 7000
    Me.Width = 6000
    picGrafik1.Height = 5200
    picGrafik1.Width = 5200
End Sub
```

Diese Prozedur hat als Startanweisung die Größendarstellung der Form. Die Höhe der Form hat einen Wert von 7000 Twips. Mit dem Schlüsselwort ME kann man den Namen eines Objektes ersetzen, für das gerade Code geschrieben wird – ME bedeutet soviel wie 'ich' oder 'meine'.

Nun geben wir die Abmessungen des Bildfeldes PICGRAFIK1 an. Damit wird sichergestellt, dass sowohl die Form als auch das Bildfeld die passenden Abmessungen für die folgenden Operationen haben.

Die Aufgabe ist, bei jedem Klick auf die ZEICHNEN-Befehlsschaltfläche eine Linie mit zufälligen Anfangs- und Endpunkten zu

zeichnen. Dazu werden die Punkte X1, X2, Y1 und Y2 zufällig über die RND-Funktion erzeugt und dann der Methode LINE als Werte zugewiesen:

```
1.Private Sub cmdZeichnen_
Click()
2.   Randomize
3.   Dim x1 As Single,
     x2 As      Single
4.   Dim y1 As Single,
     y2 As      Single

5.   x1 = Int(Rnd * 4000)
6.   x2 = Int(Rnd * 4000)
7.   y1 = Int(Rnd * 4000)
8.   y2 = Int(Rnd * 4000)

9.   picGrafik1.Line
     (x1, y1)-(x2, y2)
10.End Sub
```

Die Eigenschaft *BorderColor*, also die Farbe der Linie, steht uns nur zur Entwicklungszeit zur Verfügung, zur Laufzeit wird die Eigenschaft FORECOLOR verwendet. Die QBCOLOR-Funktion beinhaltet nur ein Spektrum von 16 Standardfarben, die mit den Werten von 0 – 15 beschrieben werden; da sie sich sehr einfach einsetzen lässt, wird sie gerne für einfache Farbgestaltungen verwendet.

`Farbe = QBColor(10)` für Hellgrün merkt man sich eher als `Farbe = RGB(255,0,0)` für Rot. Allerdings kann man mit der QBCOLOR-Funktion nur 16 Farbwerte einstellen, mit der RGB-Funktion dagegen unzählige Abstufungen, da die drei Farbanteile jeweils einen Wert zwischen 0 und 255 annehmen können. In der Prozedur CMDZEICHNEN_CLICK werden folgende Befehlszeilen hinzugefügt:

```
Dim Farbe as Integer
Farbe = Int(Rnd * 15)
picGrafik1.ForeColor =
QBColor(Farbe)
```

Damit die Linien dicker aussehen, setzt du noch folgende Zeile ein:

```
picGrafik1.DrawWidth = 3
```

Dadurch wird die Zeichenstärke der Linie eingestellt. Probiere auch größere Werte aus.

Farben (RGB-Funktion)

Farbe	RGB-Wert	Rot-anteil	Grün anteil	Blau-anteil
Schwarz	&H00	0	0	0
Blau	&HFF0000	0	0	255
Grün	&HFF00	0	255	0
Cyan (Türkis)	&HFFFF00	0	255	255
Rot	&HFF	255	0	0
Magenta (Violett)	&HFF00FF	255	0	255
Gelb	&HFFFF	255	255	0
Weiß	&HFFFFFF	255	255	255

Farben (QBColor-Funktion)

Farbe	Wert	Farbe	Wert
SCHWARZ	0	Grau	8
BLAU	1	Hellgrau	9
GRÜN	2	Hellgrün	10
TÜRKIS	3	HellTürkis	11
ROT	4	Hellrot	12
VIOLETT	5	Hellviolett	13
BRAUN	6	Gelb	14
WEIß	7	Hellweiß	15

Eine Entschuldigung: die Farben der Linien können ganz einfach erreicht werden, indem man folgenden Code schreibt:

```
picGrafik1.Line
(x1, y1)-(x2, y2),
QBColor(Farbe)
```

Für die Variable Farbe wird die Zahl der gewünschten Farbe eingesetzt, also z.B. `QBColor (10)`. Mir ging es nur darum, dass die Farbgebung deutlich wird.

Die Methode LINE kann noch mehr als nur Striche in das Bildfeld zeichnen. Die Anfangs- und Endkoordinaten bestimmen nicht nur Anfang und Ende einer Linie, sondern auch die gegenüberliegenden Ecken eines Rechtecks

```
picGrafik1.Line
(x1, y1)-(x2, y2),
QBColor(Farbe), B
```

Dieser Code erzeugt Rechtecke mit den Anfangskoordinaten als linke obere Ecke und den Endkoordinaten als rechte untere Ecke in der gewünschten Farbe – das abschließende **B** ist wichtig, da es das Rechteck aktiviert. Drückst du auf F5, wird das Ergebnis angezeigt.

Der nächste Code erzeugt ein ausgefülltes Rechteck:

```
picGrafik1.Line
(x1, y1)-(x2, y2),
QBColor(Farbe), BF
```

Vergiss auch hier nicht die letzten beiden Code-Buchstaben!

Nun wählst du DATEI | SPEICHERN VON GRAFIK1.FRM UNTER ... und gibst als Name GRAFIK2ein.

Dann wählst du DATEI | PROJEKT SPEICHERN UNTER ... und gibst wiederum den Namen GRAFIK2 an

Schließlich löschst du den Code in CMDZEICHNEN_CLICK mit Ausnahme der Variablendeklarationen

Lässt du den Computer Berechnungen ausführen, erzielst du hübsche überraschende Effekte:

```
1. Private Sub
   cmdZeichnen_Click()
2.   Dim i As Integer
3.   Dim x1 As Single,
     x2 As Single
4.   Dim y1 As Single,
     y2 As Single
5.   Dim Faktor As Integer

6.   x1 = 100: x2 = 2000
7.   y1 = 100: y2 = 2000
8.   Grafik1.DrawWidth = 10
9.   Faktor = 7.5

10.  For i = 0 To 254
11.  picGrafik1.Line
     (x1 + Faktor * i,
     y1 + Faktor * i)- _
12.  (x2 + Faktor * i,
     y2 + Faktor * i),
     RGB(i, 0, 0), B
13.  Next
14.End Sub
```

In einer Schleife werden nacheinander 255 Rechtecke gezeichnet. Die Zeilen 11 und 12 gehören hintereinander. Der Startpunkt **x1** errechnet sich aus der Variablen x1 mit dem Wert **100** plus dem Faktor mit dem Wert **7.5 * i** . **i** hat zu Beginn der Schleife den Wert **0**, d.h. der Startpunkt **x1 = x1 + 7.5 * 0 = 107.5**. Der Startpunkt für **y1** und die Endpunkte errechnen sich entsprechend.

Für die Farbe wurde die RGB-Funktion eingesetzt. Der Farbwert ändert sich bei jedem Schleifendurchlauf von **RGB(0,0,0)** bis **RGB(254,0,0)**. So entsteht der Effekt eines Farbtunnels.

Eine andere häufig eingesetzte Methode ist CIRCLE. Mit dieser Methode wird ein Kreis, eine Ellipse oder auch nur ein Kreisbogen gezeichnet. Die neuen Kenntnisse über das Koordinatensystem gelten auch hier.

Du löschst den aktuellen Code und setzt den nachfolgenden ein:

```
1.Private Sub
  cmdZeichnen_Click()
2.   Dim x1 As Single,
     y1 As Single
3.   Dim Radius As Single

4.   x1 = 500: y1 = 500
5.   Radius = 300
6.   picGrafik1.Circle (x1,
   y1),
     Radius, QBColor(12)
7.End Sub
```

Dann wählst du DATEI | SPEICHERN VON GRAFIK2.FRM UNTER ... mit dem Namen GRAFIK3 sowie DATEI | PROJEKT SPEICHERN UNTER ... mit dem Namen GRAFIK3

Der Einsatz der Methode entspricht dem Vorgang bei LINE. Allerdings benötigt man nur den Mittelpunkt des Kreises im Koordinatensystem und den Radius des Kreises

in der obigen Prozedur. Für exakte mathematische Berechnungen müssten Kenntnisse über Bogenmaß und Gradmaß vorhanden sein. Das führt uns hier zu weit.

Die Spielereien, die mit der RND-Funktion beim Thema LINE gemacht wurden, kannst du dir hier selber programmieren.

Als Eigenschaften kommen im allgemeinen nur FILLCOLOR als Farbe für das Ausfüllen des Kreises und FILLSTYLE in Frage als Möglichkeit, verschiedene Hintergrundmuster zu benutzen. Willst du die Wirkung sehen, benutzt du das Werkzeug SHAPE: du ziehst mit der Maus eine Figur im Bildfeld oder in der Form auf. Im EIGENSCHAFTEN-Fenster kannst du der Eigenschaft verschiedene Werte zuweisen, u.a. auch den Wert *3-Kreis*. Die Eigenschaft FILLSTYLE lässt dir jede Menge Möglichkeiten. Klicke alle Möglichkeiten an, um die Darstellung beurteilen zu können. Während der Laufzeit kannst du natürlich auch bestimmte vom Programm vorgegebene FILLSTYLES zuweisen, so für vertikale Linien **`Shape1.FillStyle = 3`**

```
1. Private Sub
   cmdZeichnen_Click()
2.   Dim x1 As Single,
     y1 As Single
3.   Dim Radius As Single

4.   x1 = 500: y1 = 500
5.   Radius = 300

6.   picGrafik1.FillColor =
     QBColor(8)
7.   picGrafik1.FillStyle = 0
8.   picGrafik1.DrawWidth = 3
9.   picGrafik1.Circle (x1,
   y1),
     Radius, QBColor(12)
10.End Sub
```

Zeile 6 in in dieser Prozedur bestimmt die Füllfarbe, Zeile 7 die Darstellung, Zeile 8 die Stärke der Kreislinie und Zeile 9 zeichnet die Figur mit der entsprechenden Linienfarbe.

Du speicherst wie üblich alles unter GRAFIK4 und löschst den Code in der Prozedur CMDZEICHEN_CLICK.

Die CIRCLE-Methode hat verschiedene Eingabeparameter, wie z.B. (x1, y1) oder Radius. Diese Parameter erwartet Visual Basic in einer bestimmten Reihenfolge. Zuerst also den *Mittelpunkt*, dann den *Radius*, dann die *Farbe*. Danach erwartet Visual Basic Werte für den *Startwert* eines Kreisbogens, seinen *Endwert* und das Verhältnis der *Querachse* des Kreises zur *Längsachse*. Sagst du, dass in einem Kreis alle Achsen gleich lang sind, hast du recht – das ist aber bei einer Ellipse nicht der Fall. Das Verhältnis von Querachse zur Längsachse nennt man Aspekt.

Das Zeichnen der Ellipse ist, wie du in der nachfolgenden Prozedur sehen wirst, kein Problem. Visual Basic erwartet aber wie gesagt auch den Start- und Endwert für einen Kreisbogen. Werden bestimmte Parameter nicht angegeben, weil man sie nicht benötigt, genügt es, keine Angabe zu machen und nur ein Komma einzugeben. Das Komma steht gewissermaßen als Platzhalter für **`'kein Wert'`**. Mit der nachfolgenden Prozedur werden drei Ellipsen gezeichnet, die unterschiedliche Verhältnisse der Achsen zueinander haben.

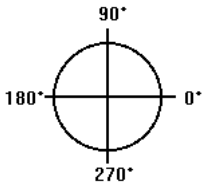
```
1. Private Sub cmdZeichnen_Click()
2.   Dim x1 As Single,
     y1 As Single
3.   Dim Radius As Single
4.   x1 = 2500: y1 = 2500
5.   Radius = 1000
6.   picGrafik1.Circle
     (x1, y1),
     Radius,
     QBColor(4), , , 0.5
7.   picGrafik1.Circle
     (x1, y1),
     Radius,
     QBColor(5), , , 2
8.   picGrafik1.Circle
     (x1, y1),   Radius,
     QBColor(6), , , 3
9. End Sub
```

Du löschst den Code, ohne zu speichern, da wir nur eine kleine Demonstration erstellt haben.

In den erweiterten Versionen von Visual Basic gibt es ein Steuerelement, mit dem man Daten grafisch darstellen kann, z.B. als *Säule* oder als sogenannte *Tortendiagamm*. So ein Tortendiagramm wollen wir jetzt selber

herstellen. Dazu benötigen wir mathematischeVorkenntnisse. Ich hatte schon erwähnt, dass zwei unterschiedliche Maßangaben für einen Kreisbogen vorhanden sind: das *Gradmaß* – ein Vollkreis hat 360°, ein Rechter Winkel 90° – und das *Bogenmaß*. Wir messen normalerweise Winkel in Grad, also benötigen wir ein Formel, die den Computer zwingt, so zu rechnen, wie wir das wollen.

Ohne in Einzelheiten zu gehen, nur so viel: die Kreiszahl π (Pi) hat einen abgerundeten Wert von 3,14159. 1° hat also den Wert π/180. Wollen wir einen Kreisbogen berechnen, der einen Winkel von 45° umschließt, muss der Computer `45 * (π / 180)` berechnen.



Berücksichtigt werden muss noch, wie der Computer die Winkelangaben sieht. Wo liegt für ihn 0°? Stelle dir dazu das Ziffernblatt einer Uhr vor. Bei 3-Uhr liegt 0°, und es wird gegen den Uhrzeigersinn weitergezählt, so dass oben 90°, bei 9 Uhr 180° und bei 6 Uhr 270° liegt.Wollen wir einen Kreisbogen von 0° bis 67° zeichnen, heißt die Anweisung also:

```
picGrafik1.Circle (x1, y1),
Radius, QBColor(15),
0 * (3.14159 / 180),
67 * (3.14159 / 180)
```

Du löschst den Code in CMDZEICHEN_CLICK bis auf die **Dim**-Zeilen und gibst diese Zeile ein.

Wir legen eine Variable **bogen** an, die uns den jeweiligen Wert berechnet – sie steht in Zeile 5 der nachfolgenden Prozedur. Es muss also nur die gewählte Gradzahl mit dem Faktor BOGEN multipliziert werden:

```
1. Private Sub cmdZeichnen_Click()
2.   Dim x1 As Single,
     y1 As Single
3.   Dim Radius As Single
4.   Dim bogen As Single
5.   bogen = 3.14159 / 180
6.   x1 = 3000: y1 = 3000
7.   Radius = 1500
8.   picGrafik1.Circle
     (x1, y1), Radius, _
9.     QBColor(15), 0 * bogen,
       67 * bogen
10.End Sub
```

Der nächste Schritt auf dem Weg zum Kuchendiagramm ist sehr einfach. Die Methode CIRCLE fügt eine Linie vom Start- bzw. Endpunkt des Kreisbogens zum Mittelpunkt ein, wenn die Werte negativ sind. Also setzt du ein Minus vor die 0 und vor 67.

Enttäuscht? Nun ja – offensichtlich gibt es für Visual Basic kein **`minus 0`**. Das Problem bekommst du mit hinreichender Genauigkeit gelöst, wenn du anstatt 0 den Wert 0.000001 einsetzt. Das millionstel Grad ist nicht bemerkbar, der Computer wird aber überlistet – denn die Zahl `-0.000001` gibt es natürlich.

Füge noch folgende Zeilen in den Code ein:

```
picGrafik1.FillStyle = 0 '(das
ist der Wert für 'ausgefüllt')
picGrafik1.FillColor =
QBColor(5)
```

Das Tortenstück ist fertig, das Prinzip bekannt. Wenn man jetzt bedenkt, dass ein Kreis 360° hat, kann man natürlich jeden Anteil des Kreises berechnen und zeichnen lassen. Nehmen wir an, dass wir drei Werte haben, nämlich 67, 53, 90, die anteilmäßig als Tortenstücke dargestellt werden sollen.

```
1.picGrafik1.FillStyle = 0
2.picGrafik1.FillColor =
  QBColor(5)
3.picGrafik1.Circle (x1, y1),
  Radius, QBColor(15),
  -0.00001 * bogen,
  -67 * bogen
4.picGrafik1.FillColor =
  QBColor(4)
5.picGrafik1.Circle (x1, y1),
  Radius, QBColor(15),
  -67 * bogen, -120 * bogen
6.picGrafik1.FillColor =
  QBColor(3)
7.picGrafik1.Circle (x1, y1),
  Radius, QBColor(15),
  -120 * bogen, -210 * bogen
```

Verwendest du diese Prozedur, bekommst du zwar schöne Tortenstückchen – aber die richtige Lösung ist das noch nicht. Um zu einer passablen Lösung zu kommen, müssen wir einige Vorüberlegungen anstellen.

```
1.Private Sub cmdZeichnen_Click()
2.Dim x1 As Single, y1 As Single
3.Dim Radius As Single
4.Dim Bogen As Single
5.Dim a As Single, b As Single, c As Single
6.Dim ges As Single
7.Dim ProzAntA As Single, ProzAntB As Single, ProzAntC As Single
8.Dim WinA As Single, WinB As Single, WinC As Single
9.a = 67: b = 53: c = 90
10.ges = a + b + c
11.ProzAntA = a * 100 / ges
12.ProzAntB = b * 100 / ges
13.ProzAntC = c * 100 / ges
14.WinA = 360 * ProzAntA / 100
15.WinB = 360 * ProzAntB / 100
16.WinC = 360 * ProzAntC / 100
17.Bogen = 3.14159 / 180
18.x1 = 3000: y1 = 3000
19.Radius = 1500
20.picGrafik1.FillStyle = 0
21.picGrafik1.FillColor = QBColor(5)
22.picGrafik1.Circle (x1, y1), Radius, QBColor(15), -0.00001 *
  Bogen, -WinA * Bogen
23.picGrafik1.FillColor = QBColor(4)
24.picGrafik1.Circle (x1, y1), Radius, QBColor(15), -WinA * Bogen,
  -(WinA + WinB) * Bogen
25.picGrafik1.FillColor = QBColor(3)
26.picGrafik1.Circle (x1, y1), Radius, QBColor(15), -(WinA + WinB)
  * Bogen, -(360) * Bogen
27.End Sub
```

Jeder Wert ist ein prozentualer Anteil der Gesamtsumme. Also berechnen wir die Gesamtsumme und speichern sie in der Variablen GES in der Zeile 10 der obigen Prozedur. Dann werden die jeweiligen prozentualen Anteile der Werte an der Gesamtsumme berechnet und gespeichert (Zeilen 11–13). Jetzt brauchen wir noch die Winkel für die Werte. Hier müssen wir davon ausgehen, dass die Bezugsgröße der Vollkreis mit 360° ist. In Zeilen 14–16 werden die Berechnungen vorgenommen.

Es wird spannend (Zeile 22). Der errechnete Winkel soll von 0 bis zu seinem Wert WinA gehen. Mit den Minuszeichen bekommen wir bekanntlich die Radien.

Wir drücken ganz nach alter Gewohnheit F5. Hast du alles sauber programmiert, müsste jetzt ein Tortenstückchen zu sehen sein, das dem Wert von 67° im Vollkreis entspricht.

Damit das nächste Stück an das bestehende anschließt, dürfen wir natürlich nicht wieder 0 als Startwert nehmen, sondern den Endwert des bestehenden Kreisbogens (Zeile 24). Der neue Endwert errechnet sich aus dem Kreisbogen, bzw. **`Winkel a+b`**. In Zeile 26 werden die Werte für das letzte Stück eingesetzt; der Endwert ist logischerweise 360.

Kleine Ungenauigkeiten nehmen wir in Kauf – wer es genauer haben will, muss mit Winkelfunktionen arbeiten. Wer es ganz professionell haben will, gibt Geld aus und beschafft sich ein Steuerelement für Diagramme.

Der nächste Schritt besteht darin, eigene Werte in Textfelder einzugeben. Wir löschen den Code in der Prozedur FORM_LOAD und gestalten die Größe der Form und des Bildfeldes mit eigenen Werten. Das Bildfeld sollte etwa eine Höhe und Breite von 4000 Twips haben, die Form etwa 5800 in Höhe und Breite.

Nun wollen wir drei Textfelder einrichten.

drei Textfelder	
TABINDEX	0, 1, 2
TEXT =	""
NAME	txtZahl1, txtZahl2, txtZahl3

1. Im Allgemeinteil deklarieren wir die Variablen a, b und c als Single, damit sie in jeder Prozedur des Projekts gelten.
2. Nun löschen wir die Variablendeklaration für a, b und c in der Prozedur CMDZEICHNEN_CLICK ...
3. dann die Zeile 9 der obigen Prozedur mit den Werten für a, b und c ...
4. und tragen schließlich folgenden Code in die Prozedur ein:
   ```
   x1 = picGrafik1.Width / 2: y1 = picGrafik1.Height / 2
   ```
   Die Werte für **x1** und **y1** ergeben sich aus der Hälfte der Breite und Höhe des Bildfeldes; damit ist der Mittelpunkt des Kreises genau in der Mitte.
5. In der Prozedur **txtZahl1_Change**, die so aussieht:
   ```
   Private Sub txtZahl1_Change()
       a = Val(txtZahl1.Text)
   End Sub
   ```
   wird der Wert für die Variable *a* eingegeben. Da Visual Basic alle Eingaben in ein Textfeld als Zeichenketten oder STRINGS betrachtet, muss der Wert mit der VAL-Funktion in einen Zahlenwert umgewandelt werden, denn mit einer Zeichenkette können wir nicht weiter rechnen.

   Über die Maus oder TAB werden die Textfelder angesteuert; deswegen sind die Tabindices in der entsprechenden Reihenfolge angelegt worden. Für die anderen beiden Textfelder müssen identische Codes geschrieben werden.
6. Damit man die Eingaben auch mit EINGABE abschließen kann und das nächste Textfeld angesprungen wird, tragen wir in die Prozedur Key Press folgenden Code ein:
   ```
   Private Sub txtZahl1_KeyPress(KeyAscii As Integer)
       If KeyAscii = 13 Then
           SendKeys "{TAB}"
       End If
   End Sub
   ```
   Damit wird geprüft, ob die Eingabetaste (**KeyAscii = 13**) gedrückt wurde. Wenn ja, wird der Befehl SENDKEYS verwendet, der in diesem Fall das gleiche bewirkt wie die TAB-Taste –SENDKEYS bedeutet 'Sende die Taste', es also wird der Tastendruck simuliert, der in den geschweiften Klammern steht, etwa {F5} oder {Esc}.

   Fehlen auf deiner Tastatur die Zeichen für die geschweiften Klammern, erzeugst du sie mit ALT + 0123 für '{' und ALT + 0125 für '}'; du musst den Zahlenblock benutzen!
7. Damit man weiß, welche Farbe zu welchem Wert gehört, werden die Eigenschaften wie in der folgenden Prozedur gesetzt:
   ```
   8 Private Sub Form_Load()
   9    TxtZahl1.BackColor = QBColor(5)
   10   TxtZahl1.ForeColor = QBColor(15)
   11   TxtZahl1.FontSize = 12
   12   TxtZahl2.BackColor = QBColor(4)
   13   TxtZahl2.ForeColor = QBColor(15)
   14   TxtZahl2.FontSize = 12
   15   TxtZahl3.BackColor = QBColor(3)
   16   TxtZahl3.ForeColor = QBColor(15)
   17 TxtZahl3.FontSize = 12
   18End Sub
   ```
   Wichtig: Eingabefehler, wie z.B. Buchstaben werden nicht abgefangen, ebenso keine Nichts-Eingaben.

8 Verwende zunächst Zahlenwerte, bei denen du kalkulieren kannst, ob alle Berechnungen einwandfrei ausgeführt werden., z.B. 100, 100, 100 oder 200, 100, 100.

9 Schließlich drückst du wieder F5.

Diagramme haben den Zweck, Werte anschaulich darzustellen. Manchmal erzeugt man mit sogenannte Säulendiagrammen den besseren Eindruck. Wir wollen dem Anwender die Möglichkeit geben, zwischen einer Säulen- und einer Kreisdarstellung zu wählen.

1. Zunächst speicherst du alles unter GRAFIK5.
2. Als neuen Namen für das Objekt CMDZEICHNEN wählst du CMDKREIS ...
3. ... und gibst für CMDKREIS.CAPTION **Kreisdiagramm** an.
4. Dann änderst du die Prozedur CMDZEICHNEN_CLICK in PRIVATE SUB KREISDIGRAMM() mit folgendem Code:

```
Private Sub KreisDiagramm()
Dim x1 As Single, y1 As Single
    Dim Radius As Single
    Dim Bogen As Single
     :::
     ::: usw.
     :::
End Sub
```

5. Der Wert der Form.CAPTION ist **Grafische Darstellung von Werten**

Befehlsschaltfläche		
NAME	Name	**cmdSäule**
CAPTION	Beschriftung	**Säulen-diagramm**

5. Du doppelklickst auf cmdKreis und gibst folgenden Code ein:

```
Private Sub cmdKreis_Click()
    picGrafik1.Cls
    KreisDiagramm
End Sub
```

In dieser Prozedur wird mit der Methode CLS der Inhalt des Bildfeldes gelöscht und dann angewiesen, die Prozedur KREISDIAGRAMM auszuführen, die unseren Code für das Zeichnen von Kreisdiagrammen enthält.

6. Endlich drückst du auf F5

   Hast du alles richtig gemacht, müsste das Programm ablaufen wie vorher.

7. Nun doppelklickst du auf CMDSÄULE und gibst folgenden Code ein:

```
Private Sub cmdSäule_Click()
    picGrafik1.Cls
End Sub
```

Mit diesem Code wird das Bildfeld gelöscht.

8. Dann drückst du ... genau: F5
9. Mit dem nachfolgenden Code wird ein sogenannter Dummy eingesetzt:

```
Private Sub SäulenDiagramm:
  picGrafik1.Print
  "Hier sollen Säulen hin."
End Sub
```

10. .... und der übliche Druck auf F5

    Klappt alles, wenn du ein Kreisdiagramm erzeugst und dann auf die Befehlsschaltfläche Säulendiagramm klickst?

11. Du doppelklickst auf cmdSäule, ...
12. .... ergänzt den Code wie folgt:

```
Private Sub cmdSäule_Click()
  picGrafik1.Cls
  SäulenDiagramm
End Sub
```

13. .... und drückst wieder einmal F5

Der Code in dieser Prozedur ist ein Dummy. Dummies kennst du als Puppen, die u.a. bei Auto-Crash-Tests eingesetzt werden. Mit diesem Dummy probierst du aus, ob bis hierher alles klappt.

Es ist auf diese Art einfacher, Programmierfehler einzugrenzen und zu korrigieren.

Nebenbei: du wirst in Visual Basic-Zeitschriften oder Lehrbüchern immer wieder lesen, dass jedes Projekt vorher genau skizziert werden soll – und wenn du die Programmbeschreibung liest, denkst du, du hast ein Genie vor dir: es ist alles im vorhinein bedacht worden. Niemand weiß vorher ganz genau, welche Prozeduren entwickelt, welche Variablen deklariert werden müssen usw. Ein Unsinn ist auch, dem Leser vorzugaukeln, dass die ausgefeilte Bedienoberfläche zuerst entwickelt wurde. Der grobe Rahmen muss stehen, Feinheiten und neue Ideen ergeben sich unterwegs. Für Testläufe sind Dummies also eine feine Sache.

Nun zu unseren Säulen. Der erste Vorschlag ist, einfach die *Line*-Methode mit dem Parameter **B,** bzw. **BF** zu verwenden .

```
1.Private Sub SäulenDiagramm()
2.  picGrafik1.Line
    (100, 100)-(500, a),
    QBColor(5), BF
3.  picGrafik1.Line
    (600, 100)-(1000, b),
    QBColor(4), BF
4.  picGrafik1.Line
    (1100, 100)-(1500, c),
    QBColor(3), BF
5.End Sub
```

Die Prozedur ist selbsterklärend, hat aber eindeutig große Nachteile. Zunächst einmal 'hängen' die Säulen – das liegt daran, dass der Koordinaten-Nullpunkt in der linken oberen Ecke liegt, zum anderen passen die Säulen bei großen Werten nicht mehr in das Bildfeld.

Wir müssen einen kleinen Exkurs über 'virtuelle Koordinatensysteme' einlegen. Visual Basic ermöglicht mit der Methode SCALE die Einrichtung eigene Koordinatensysteme, wobei auch andere Maßeinheiten eingesetzt werden können – so können z.B. anstatt *Twips* auch *cm* verwendet werden. Für uns ist interessant, dass wir den Koordinatenursprung verlegen können.

Das Bildfeld hat über die Eigenschaften HEIGHT und WIDTH eine bestimmte Anzahl von *Twips*. Wir können aber auch sagen, dass das Bildfeld z.B. 1000 * 1000 Einheiten messen soll. Das Bildfeld behält seine *Twips*, aber virtuell - also künstlich - können wir so tun, als gingen uns die *Twips* nichts an. Wir können alle Methoden und Operationen auf der Grundlage 1000 * 1000 durchführen. Das ist für Skalierungen sehr praktisch, weil komplizierte Umrechnungen entfallen. Unsere virtuelle Skala oder das Säulendiagramm werden durch Visual Basic vom künstlichen zum normalen Zustand umgearbeitet.

```
1.Private Sub SäulenDiagramm()
2.  picGrafik1.Scale (0, picGrafik1.Height)-(picGrafik1.Width, 0)
3.  picGrafik1.DrawWidth = 2
4.  picGrafik1.ForeColor = QBColor(0)
5.  'waagerechte Skala
6.  picGrafik1.Line (250, 250)-(picGrafik1.Width - 500, 250)
7.  'senkrechte Skala
8.  picGrafik1.Line (250, 250)-(250, picGrafik1.Height - 500)
9.  'Säule a
10.  picGrafik1.Line (300, 250)-(800, a+250), QBColor(5), BF
11.  'Säule b
12.  picGrafik1.Line (850, 250)-(1350, b+250), QBColor(4), BF
13.  'Säule c
14.  picGrafik1.Line (1400, 250)-(1850, c+250), QBColor(3), BF
15.End Sub
```

Für unseren Zweck legen wir die Null-Punkte der Koordinaten mit der Befehlszeile 2 in der obigen Prozedur in die linke untere Ecke und geben ein virtuelles Koordinatensystem von `picGrafik1.Height * picGrafik1.Width` vor, also die Höhe und Breite des Bildfeldes. Später werden wir eine andere Skalierung ausprobieren.

Der Code löst unser Problem zunächst einmal. Mit seiner Hilfe wird gewissermaßen die y-Achse in ihrer Richtung umgekehrt.

Zu einem Säulendiagramm gehört eine Skala. Zeile 6 und Zeile 8 zeichnen die Linien. Dann werden Rechtecke mit den Werten für die Variablen a, b und c gezeichnet, wobei zu den Werten der Abstand der unteren Skala zum Bildfeldrand addiert werden muss.. Die Koordinaten sind so abgemessen, dass sie nicht an den Rändern liegen. Welche Abstände man zwischen den Säulen vornimmt, bzw. welche Breite die Säulen haben sollen, ist Geschmackssache. Soll die Skala noch eine Einteilung und Beschriftungen bekommen, müssen die Abstände größer als hier sein. Verwende erst einmal die angegebenen Werte und probiere eigene Einstellungen.

Du kannst jetzt zwischen dem Kreis- und dem Säulendiagramm hin- und herschalten, neue

Werte eingeben, auch während der Laufzeit die Werte verändern usw.

Damit die leidige Fehlermeldung und damit der Programmabbruch bei leeren Textfeldern vermieden wird, setzen wir in die Prozeduren SÄULENDIAGRAMM und KREISDIAGRAMM folgende Zeile ein:

```
On Error Resume Next
```

Das bedeutet etwa: bei einem Fehler einfach weitermachen. Dieses Anweisung ist gleichermaßen praktisch wie gefährlich, denn damit wird natürlich die Fehlersuche, das sogenannte DEBUGGING, sehr erschwert. Für einfache, überschaubare Programme kann man die Anweisung allerdings bedenkenlos verwenden.

Nun speicherst du alles unter GRAFIK6.

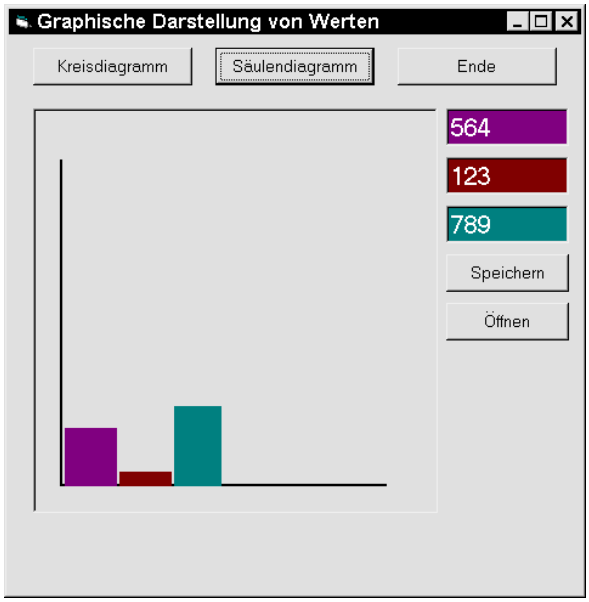
Zum Abschluss sollen noch Prozeduren für das Speichern und Laden der Werte eingefügt werden. Die Bedeutung der einzelnen Anweisungen sind im Abschnitt Dateiverwaltung beschrieben – wir wiederholen die beiden Prozeduren noch einmal:

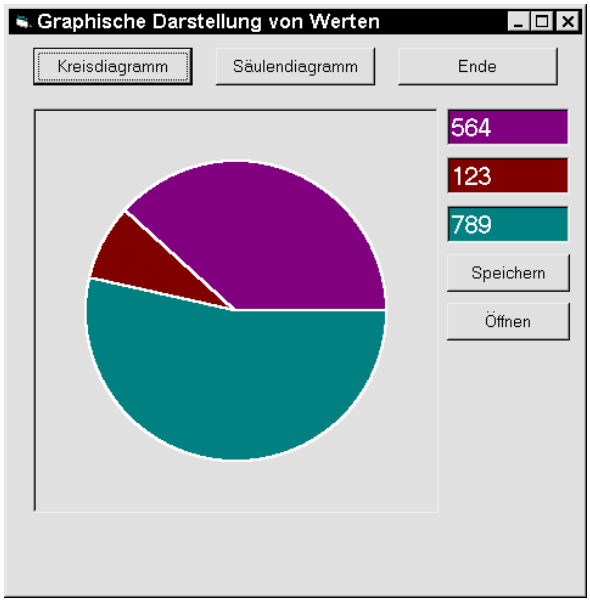
```
1. Private Sub
   cmdSpeichern_Click()
2.   Dim DateiNr As Integer
3.   Dim DateiName As String
4.   DateiName =
       "C:\Coden\Zahlen.txt"
5.   DateiNr = FreeFile
6.   Open DateiName For Output
     As DateiNr
7.     Print #DateiNr, a, b, c
8.   Close
9. End Sub
```

```
1. Private Sub
   cmdÖffnen_Click()
2.   Dim DateiNr As Integer
3.   Dim DateiName As String
4.   Dim Temp As String
5.   DateiNr = FreeFile
6.   DateiName =
   "C:\Coden\Zahlen.txt"
7.   Open DateiName For Input
   As DateiNr
8.    Input #DateiNr, a, b, c
9.    txtZahl1.Text = a
10.   txtZahl2.Text = b
11.   txtZahl3.Text = c
12. Close
13. End Sub
```

- Denke bei der Variablen DateiName daran, deinen eigenen Pfad zu verwenden.

Die beiden Diagramme haben folgendes Aussehen:





Simulationen

Computer sind durch ihre Rechengeschwindigkeit vorzüglich in der Lage, komplizierte Zusammenhänge schnell und übersichtlich darzustellen. Daher werden sie ja auch für sogenannte Hochrechnungen verwendet. Du kennst das sicherlich aus den Wahlberichterstattungen – aus wenigen ausgewählten Daten wird auf das Endergebnis hochgerechnet – und zwar mit erstaunlicher Genauigkeit.

Epidemie im Aquarium

Im folgenden Programm soll der Computer die Lebensqualität in einem Aquarium untersuchen.

Angenommen, in einem Aquarium leben 1000 Wasserflöhe. Diese Tiere vermehren sich mit einer bestimmten Vermehrungsrate, die aus der Biologie bekannt ist. Der Nachwuchs vermehrt sich, dessen Nachwuchs vermehrt sich usw. Ältere Tiere sterben. Auch die Lebenserwartung ist bekannt, man weiß, wie alt Wasserflöhe werden.

Nun kann man sich denken, dass die Wasserflöhe Abfallprodukte erzeugen, die das System Aquarium belasten. Der Lebensraum wird also zusehends vergiftet.

Wie wirkt sich dieser Sachverhalt auf den Lebensraum aus und wie auf die Lebewesenanzahl?

Der Computer soll in einer Tabelle die Entwicklung darstellen.

Grundannahmen

Es müssen einige Grundannahmen gemacht werden. Wir unterstellen eine Vermehrungsrate von 0, 3 Punkten und eine Sterberate von 0, 2. Diese Werte haben sich in der Biologie als annähernd realistisch herausgestellt. Im übrigen geht es um das Prinzip, nicht um neue wissenschaftliche Erkenntnisse. Als Vergiftungsrate nehmen wir 0, 05 an; dadurch wird der Vergiftungsfaktor 0, 00005 (das hängt von der Wasserqualität ab).

Die Gleichungen habe ich aus Biologiebüchern übernommen und computergerecht umgesetzt.

Es gibt also folgende Variablen:

WASSERFLÖHE	= 1000
VERMEHRUNGSRATE	= 0, 3
STERBERATE	= 0, 2
ABFALLPRODUKTE	= 0 (zu Beginn des Systems sind ja noch keine vorhanden)
VERGIFTUNGSFAKTOR	= 0,00005
VERGIFTUNGSRATE	= 0, 05

Mathematisch ausgedrückt müssen Gleichungssysteme programmiert werden. Für Mathematiker: die Grundlage sind Differentialgleichungen.

1. `Sterberate = Sterberate + Vergiftungsfaktor * Abfallprodukte`
2. `Wasserflöhe = Wasserflöhe + (Vermehrungsrate - Sterberate) * Wasserflöhe`
3. `Abfallprodukte = Abfallprodukte + Vergiftungsrate * Wasserflöhe`

Die Gleichungen müssen in dieser Reihenfolge geschrieben werden, weil die errechneten Werte in der nächsten Gleichung eingesetzt werden.

Code

1. In der FORM gibst du etwa **`6300 * 6300 Twips`** ein und als CAPTION **`Wasserflöhe`**.
2. Dann erstellst du eine BEFEHLSSCHALTFLÄCHE:

NAME	Name	**`cmdStart`**
CAPTION	Beschriftung	**`&Start`**

3. Du deklarierst die Variablen im Allgemeinteil des nachfolgenden Code:

```
1 Option Explicit
2 Dim Wasserflöhe As Single
3 Dim Vermehrungsrate As Single
4 Dim Sterberate As Single
5 Dim Abfallprodukte As Single
6 Dim Vergiftungsfaktor As Single
```

```
7 Dim Vergiftungsrate As
  Single
8 Dim Generationen as Integer
```

4. In der Prozedur FORM_LOAD weist du den Variablen die nachfolgenden Werte zu:

```
1 Private Sub Form_Load()
2 Wasserflöhe = 1000
3 Vermehrungsrate = 0.3
4 Sterberate = 0.2
5 Abfallprodukte = 0
6 Vergiftungsfaktor = 0.00005
7 Vergiftungsrate = 0.05
8 End Sub
```

5. Dann doppelklickst du auf Start und setzt folgenden Code ein:

```
1 Private Sub cmdStart_Click()
2   Print "Generationen", "Anzahl der", "Sterberate",
  "Abfallprodukte"
3   Print , "Wasserflöhe"
4   Print Generationen, Wasserflöhe, Sterberate, Abfallprodukte
5   Do While Int(Wasserflöhe) > 0
6     Generationen = Generationen + 1
7     Sterberate = Sterberate + Vergiftungsfaktor * Abfallprodukte
8     Wasserflöhe = Wasserflöhe + (Vermehrungsrate - Sterberate) *
  Wasserflöhe
9     Abfallprodukte = Abfallprodukte + Vergiftungsrate *
  Wasserflöhe
10    Print Generationen, Int(Wasserflöhe), Int(10 * Sterberate) /
  10, Int(Abfallprodukte)
11  Loop
12End Sub
```

6. In einer sehr unprofessionellen Form werden die Werte durch die PRINT-Methode einfach auf der Form dargestellt. Das hat aber den Vorteil, dass man erst einmal sehen kann, ob die Berechnungen stimmen.
7. Zeile 2 und 3 sind für die Titel der Listen da. In Zeile 3 musst du darauf achten, dass das Komma nicht übersehen wird. Es ist gewissermaßen ein Tabulator und schreibt den Wert **`Wasserflöhe`** an die nächste Tabulatorstelle. In Zeile 4 werden die Startwerte der Variablen geschrieben, weil in der Schleife schon die nächste Berechnung erfolgt.
8. Zwischen Zeilen 5 und 11 wird eine Schleife so lange wiederholt, wie der Wert der Variablen **`Wasserflöhe`** größer als Null ist. Dabei wird die Zahl der Generationen in Zeile 6 hochgezählt.
9. Die Zeilen 7–9 sind die Berechnungsformeln, die du nicht unbedingt verstehen musst.
10. In Zeile 10 werden die errechneten Werte auf die Form geschrieben, wobei du die Anweisung
    `Int(10 * Sterberate) / 10`
    beachten solltest. Dadurch wird eine Darstellung der Dezimalzahl auf eine Stelle nach dem Komma beschränkt, wobei das Auf- und Abrunden mathematisch einwandfrei ausgeführt wird.

    `Int(100 * Sterberate) / 100`
    ergibt eine zweistellige Dezimalzahl.
11. Nun drückst du auf F5 ...
12. .... und klickst auf CMDSTART.

Die Form sollte die folgenden Zahlenreihen aufweisen:

Generationen	Anzahl der Wasserflöhe	Sterberate	Abfallprodukte
0	1000	0,2	0
1	1100	0,2	55
2	1206	0,2	115
3	1317	0,2	181
4	1425	0,2	252
5	1525	0,2	328
6	1606	0,2	409
7	1659	0,2	492
8	1673	0,2	575
9	1639	0,3	657
10	1551	0,3	735
11	1411	0,3	805
12	1227	0,4	867
13	1014	0,4	918
14	791	0,5	957
15	579	0,5	986
16	395	0,6	1006
17	250	0,6	1018
18	145	0,7	1026
19	77	0,7	1030
20	37	0,8	1031
21	15	0,8	1032
22	5	0,9	1033
23	1	0,9	1033
24	0	1	1033

Interpretation

Zunächst vermehren sich die Wasserflöhe, die Sterberate bleibt annähernd gleich, aber die Abfallprodukte nehmen zu. In der 10. Generation kippt das System bis hin zum totalen Zusammenbruch in der 24. Generation.

Du setzt den Wert der Variablen VERGIFTUNGS-FAKTOR auf 0.00002. und drückst auf F5

Damit hast du die Lebensqualität der Wasserflöhe verbessert; es könnte die Wasserqualität sein, aber auch das Futter. Immerhin bricht das System erst in der 32. Generation zusammen.

Viel aussagekräftiger als diese Zahlenreihen sind grafische Darstellungen. Dafür reicht leider der Platz nicht.

Anhang

Suffixe für Steuerelemente

Empfehlungen für Namen (Steuerelemente)	
cmd	Befehlsschaltfläche (Command)
frm	Form
pic	Bildfeld (Picture)
lbl	Bezeichnungsfeld (Label)
fra	Rahmenfeld (Frame)
txt	Textfeld (Textbox)
shp	Figurenelement (Shape)

Text mit Zeichenketten-Funktionen umkehren

1. Du erstellst die Befehlsschaltfläche CMDENDE, ...
2. ... dann die Befehlsschaltfläche CMDUMWANDELN.
3. Dann folgt das Textfeld TXTEINGABE ...
4. ... und das Textfeld TXTAUSGABE.

   Der Code lautet:

```
I.    Umwandeln Option Explicit
II.   Private Sub
      cmdEnde_Click()
III.    End
IV.   End Sub

V.    Private Sub
      cmdUmwandeln_Click()
VI.     Dim ursprung As String
VII.    Dim temp As String
VIII.   Dim i As Integer
IX.     ursprung =
      txtEingabe.Text
X.      temp = ursprung
XI.     For i = 1 To
      Len(ursprung)
XII.    Mid$(ursprung, i, 1) =
      Mid$ _
        (temp, (Len(temp) - i) + 1,
      1)
XIII. Next i
XIV.  txtAusgabe.Text =
      ursprung
```

5. End Sub
6. Dann drückst au auf F5...
7. ... gibst den Text ein und siehst UMWANDELN.

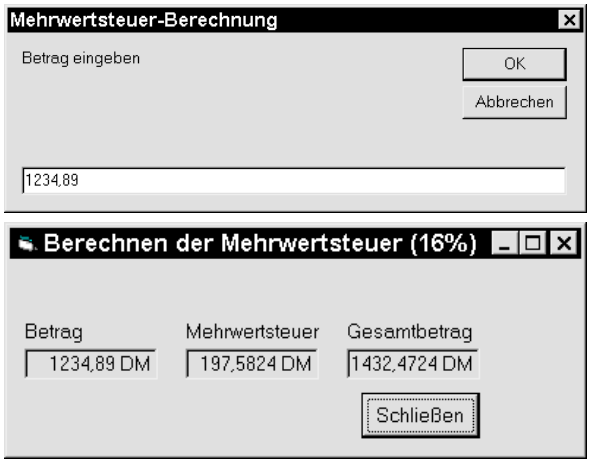
Du kannst dir das Programm für deine Anwendungen abkupfern. Es geht hier nicht darum, dass es bessere Lösungen gibt, sondern um die Zeichenkettenverarbeitung.

Funktion

Die Formel für die Berechnung der Mehrwertsteuer ist **bei einem MWS-Satz von 16%**

Steuer = Betrag * 16 / 100

Du legst die Bezeichnungsfelder wie im Bild an.



Die Namen sind: LBLBETRAG, LBLMWS, LBLGESAMTBETRAG

Dann erstellst du die Befehlsschaltfläche CMDENDE

Der Code lautet:

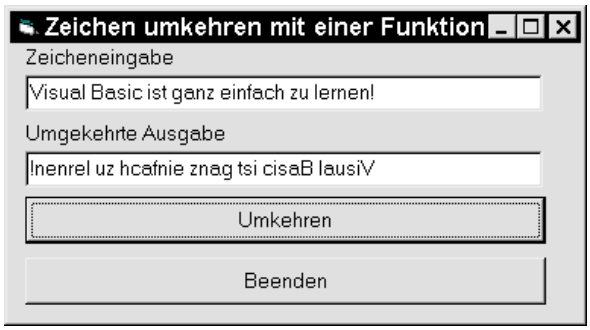
```
Option Explicit
Dim Steuer As Currency

Private Sub Form_Load()
    Dim Betrag As Currency
    Betrag=InputBox("Betrag ein-
    geben", "Mehrwertsteuer-
    Berechnung")
    MehrwertSteuer (Betrag)
    lblBetrag.Caption = Betrag &
    " DM "
    lblMws.Caption = Steuer &
    " DM "
    lblGesamtbetrag.Caption =
    Betrag
    + Steuer & " DM "
End Sub

Private Function
MehrwertSteuer(Betrag) As Currency
    Steuer = Betrag * 16 / 100
End Function

Private Sub cmdEnde_Click()
    End
End Sub
```

Zeichenkette umkehren mit Funktion

- Textfeld txtEingabe
- Textfeld txtAusgabe
- Befehlsschaltfläche cmdUmkehren
- Befehlsschaltfläche cmdEnde
- Bezeichnungsfelder
- cmdUmkehren.Default = True (damit kann auch durch EINGABE 'umgekehrt' werden.

Der Code lautet:

```
1. Private Sub
   cmdUmkehren_Click()
2.   Dim Text As String
3.   Text = txtEingabe.Text
4.   Rückwärts (Text)
5.   txtAusgabe.Text =
   Rückwärts(Text)
6. End Sub
```



In Zeile 3 wird der Inhalt des EINGABE-Textfensters in der Variablen TEXT abgelegt. Dann wird die Funktion aufgerufen und dabei die Variable als Wert übergeben. In der nachfolgenden wird die Variable in der For-Next-Schleife, Zeilen 4 - 6 verarbeitet, indem sie von hinten nach vorn eingelesen wird. Die einzelnen Zeichen werden in einer Hilfsvariablen *Temp* gespeichert, deren Wert als Funktionsergebnis zurückgegeben und im Ausgabefenster angezeigt wird.

```
1. Private Function
   Rückwärts(a) As String
2.   Dim i As Integer
3.   Dim Temp As String
4.   For i = Len(a) To 1 Step-1
5.     Temp = Temp + Mid(a, i,
   1)
6.   Next
7.   Rückwärts = Temp
8. End Function
```

Eingabe-Dialogfeld (Input-Box)

Für einfache Eingaben in Programme stellt Visual Basic ein Eingabe-Dialogfeld zur Verfügung. Die Input-Box zeigt ein Fenster mit einer Eingabeaufforderung an und erwartet in einem Textfeld eine Eingabe. Der Text, es kann auch eine Zahl sein, wird an das Programm zurückgegeben und ausgewertet.

Ich verwende die Input-Box fast ausschließlich für das schnelle testen von Eingaben, bevor ich eine andere Methode programmiere.

```
1.Private Sub Form_Load()
2.Dim Frage As String, Titel As
  String
3.Dim Vorgabe As Integer
4.Dim Bewertung As Integer
5.Frage = "Wie alt bist du?"
6.Titel = "Altersabfrage"
7.Vorgabe = 100
8.Bewertung = InputBox(Frage,
  Titel, Vorgabe, 200, 200)
9.Select Case Bewertung
10.  Case Is < 20
11.    lblBewertung.Caption =
  "Ganz schön jung!"
12.  Case 21 To 29
13.    lblBewertung.Caption = "Du
  bist ein Zwanziger"  Case 30
  To 39
14.    lblBewertung.Caption = "Du
  bist ein Dreissiger"
15.  Case Is > 40
16.    lblBewertung.Caption =
  "Ganz schön alt!"
17.End Select
18.End Sub
```

In diesem Listing werden die Variablen deklariert und mit Werten belegt. Dann wird der Variablen Bewertung das Ergebnis der Input-Box-Funktion übergeben und in einer SELECT/CASE-Anweisung überprüft. Das Ergebnis der Auswertung wird in einem Anzeigefeld als CAPTION angezeigt.

Fehler werden in diesem Beispiel nicht abgefangen. Solltest du also Buchstaben eingeben, steigt das Programm mit einer Fehlermeldung aus, denn die Variable Bewertung wurde als Integer deklariert.

Willst du Texteingaben überprüfen, z.B. Namen, musst du das Programm nur etwas umschreiben. So könntest du dir einen kleinen Schutzcode für ein Programm schreiben, der erst nach der Eingabe eines richtigen Codewortes den Start des Programms ermöglicht.