DELPHI 04

Standard-Komponenten

[4] KOMPONENTEN IM PROGRAMM

Im Folgenden sollen häufig benutzte Steuerkomponenten von DELPHI kurz beschrieben werden. Wichtige Eigenschaften bei allen Komponenten sind beispielsweise *Align*, *Position*, *Color*, *Enabled*, *Font*, *ReadOnly* und *Visible*. Diese definieren Ausrichtung, Position, Farbe, Modalität, Zeichensatz, Zugriff und Sichtbarkeit der Komponenten. Weitere Eigenschaften werden fallweise genauer beschrieben. In den nachfolgenden Ausführungen wird, der Sprachvereinfachung wegen, nicht immer genau zwischen Objektklasse (z.B. *TForm*) und Objektinstanz (*Form*) unterschieden.

[4.01] Das Formular *Form*

Das Objekt *Form* dient als Behälter (Container) aller anderen Komponenten. Von seinen vielen Properties und Events sollen nur *KeyPreview* und *OnKeyPress* exemplarisch erklärt werden. Wenn die Eigenschaft *KeyPreview := True* ist, dann wird jedes Tastaturereignis zuerst zum Formular geleitet und nachher erst zum sendenden Objekt. Wenn *KeyPreview := False* ist, dann erfolgt die Nachricht direkt zum Objekt.

In unserem Beispiel soll der Eingabeabschluss in den beiden Editierfeldern *Edit1* und *Edit2* mit der <Enter>-Taste (Code 13) möglich sein, worauf der Inhalt des aktuellen Feldes einer Stringvariablen zugewiesen, der Inhalt des anderen Feldes gelöscht und dann sofort zu diesem gesprungen wird. Bei der Formularerzeugung muss zuerst mit einem Doppelklick auf die Formularfläche von *Form1* die Schablone der *OnCreate*-Ereignisbehandlungsroutine *Form1.FormCreate* in der Programmunit generiert werden. Diese dient zur Initialisierung von Eigenschaften und Variablen am Programmanfang. Sodann wird die Schablone für die *OnKeyPress*-Routine *Form1.FormKeyPress* programmiert. Zuletzt müssen die *OnKeyPress*-Routinen der Editierfelder im Objektinspektor ebenfalls mit *Form1.FormKeyPress* benannt werden. Dadurch wird jedes Tastaturereignis, welches in den Editierfeldern stattfindet, im darunter liegenden Formular behandelt.

```
var X,Y: String;

procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
{ Behandlung von Tastaturereignissen }
begin
  if (Sender = Edit1) and (Key = #13) then begin
    X := Edit1.Text;
    Edit2.Text := '';
    Edit2.SetFocus;
  end;
  if (Sender = Edit2) and (Key = #13) then begin
    Y := Edit2.Text;
    Edit1.Text := '';
    Edit1.SetFocus;
  end;
end;
```

Die verschiedenen Komponenten eines Formulars können auch als ein einfach indiziertes Array angesprochen werden. DELPHI stellt dafür das Objekt *Controls* aus der Klasse *TControl* zur Verfügung. *ControlCount* liefert dabei die Anzahl der aktuellen Formularkomponenten. Damit können beispielsweise folgende Anweisungen sehr elegant geschrieben werden:

```
var I : Integer;

With Form1 do begin
  For I := 0 to ControlCount-1 do Controls[I].Enabled := False;
  For I := 0 to ControlCount-1 do ListBox1.Items.Add(Controls[I].Name);
end;
```

Ein häufiges Problem liegt darin, dass die Formulargröße eines Programmes nicht zur jeweiligen Bildschirmauflösung des Anwenders passt. Ist das Formular beispielsweise in einer Auflösung von 1024 x 768 Bildpunkten programmiert, dann wird es für eine Auflösung von 800 x 600 zu groß sein. Mit folgenden Anweisungen, welche in einer *FormCreate*- oder *FormActivate*-Routine am Programmanfang stehen, ist es möglich, das Formular an den Bildschirm anzupassen.

```
procedure FitForm(F: TForm);
begin
  with F do begin
    if (Screen.Width <> 1024) then ScaleBy(Screen.Width,1024);
    if (Font.PixelsPerInch <> 120) then ScaleBy(120,Font.PixelsPerInch);
    WindowState := wsMaximized;
  end;
end;
```

Der wesentliche Befehl dabei ist *ScaleBy(Z,N)*, durch welchen das Formularobjekt und alle seine untergeordneten Komponenten im gewünschten Verhältnis *N : Z* skaliert werden. Ist in der Systemsteuerung von Windows die Option Schriftgrad auf kleine Fonts (96) eingestellt, so kann dies mit *Form1.Font.PixelsPerInch* abgefragt und dann gegebenenfalls das ganze Formular noch einmal entsprechend skaliert werden, wenn die Programmierung mit großen Fonts (120) erfolgte.

[4.02] Die Verwendung zusätzlicher Formulare

Will man in einem Projekt (*Project1*) zwei oder mehrere Formulare verwenden, dann muss zunächst ein Formular (Hauptformular *Form1*) mit der zugehörigen Unit (*Unit1*) angelegt werden. Dieses erscheint immer beim Programmstart. Ein zweites Formular (Nebenformular *Form2*) wird beim Entwurf in der Entwicklungsumgebung mittels Menüpunkt <*Datei / Neues Formular*> hinzugefügt. Dabei legt DELPHI automatisch eine neue Unit (*Unit2*) an und bindet diese auch in die Projektdatei ein. Mit Hilfe des Menüpunktes <*Ansicht / Projekt Quelltext*> kann der Projektcode direkt eingesehen werden.

```
program Project1;

uses Forms,
     Unit1 in 'Unit1.pas'  {Form1},
     Unit2 in 'Unit2.pas'  {Form2};

{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end;
```

Damit auf das Nebenformular vom Hauptformular auch zugegriffen werden kann, muss vom Programmierer entweder im Interfaceteil oder im Implementierungsteil der Hauptunit (*Unit1*) die Nebenunit (*Unit2*) mit einer entsprechenden *Uses*-Anweisung eingebunden werden (*uses Unit2*).

Der Aufruf des Nebenformulares aus dem Hauptformular kann mit Hilfe des *OnClick*-Ereignisses eines entsprechenden Schaltknopfes (*Form1.Button1*) erfolgen.

```
procedure TForm1.Button1Click(Sender: TObject);
// Öffnen von Form2
begin
    Form2.Show;           // die Methode Show öffnet das zweite Formular
//  Form2.ShowModal;      // alternative Öffnung des zweiten Formulars
end;
```

Wird ein zweites Formular mit *Show* aufgerufen, dann kann parallel dazu auch auf das erste Formular zugegriffen werden. Das ist mit der alternativen Methode *ShowModal* nicht möglich!

Um vom Nebenformular in das Hauptformular zurückzukehren, kann ein *OnClick*-Ereignis eines entsprechenden Schaltknopfes (*Form2.Button1*) verwendet werden. Das geschieht aber in *Unit2*.

```
procedure TForm2.Button1Click(Sender: TObject);
// Schließen von Form2
begin
  Form2.Close;            // die Methode Close schließt das Formular
end;
```

Natürlich können auf diese Weise auch mehrere Formulare mit ihren zugehörigen Units erzeugt werden. In der Entwicklungsumgebung bietet der Menüpunkt <*Projekt / Optionen ...*> die Möglichkeit, alle bereits erstellten Formulare eines Projektes aufzulisten. Mit der Taste <*Strg F12*> kann zwischen den einzelnen Units bzw. ihren Formularen gewechselt werden.

Dem interessierten Leser steht das Programm "***fenster***" auf der Begleit-CD zur Verfügung. Dort können mit der Maus die verschiedenen Formen der Fenstermanipulationen ausprobiert werden (Öffnen des Fenstermenüs, Verschieben, Verkleinern, Vergrößern, zum Symbol schrumpfen, auf Screengröße ausdehnen oder überhaupt schließen). Wird dabei das Formularfenster kleiner als seine Originalgröße, dann werden automatisch Bildlaufleisten am Fensterrand erzeugt. Das alles demonstriert sehr eindrucksvoll die Funktionalität von Fensterobjekten (eben ***Windows***)!

[4.03] Der Schaltknopf *Button*

Ein Schaltknopf *Button* soll meistens auf ein *OnClick*-Ereignis mit der Maus reagieren. Dazu wird zuerst die Schablone der entsprechenden Ereignisbehandlungsroutine *Form.ButtonClick* erzeugt. In diese wird dann der gewünschte Programmcode geschrieben. Für den Anwender ist neben den anderen Standardeigenschaften die Texteigenschaft *Caption* von besonderer Wichtigkeit. Durch sie kann der Schaltknopf aussagekräftig beschriftet werden.

[4.04] Die Wechselschalter *CheckBox*

Eine *CheckBox* ist ein einfacher Ein/Aus-Schalter. Die logische Eigenschaft *Checked* (True/False) entscheidet darüber, ob die *CheckBox* eingeschaltet (ON) oder ausgeschaltet (OFF) ist.

```
if CheckBox1.Checked then ...         //  wenn Schalter auf ON steht, dann ...
if not CheckBox1.Checked then ...     //  wenn Schalter auf OFF steht, dann ...
```

[4.05] Das Anzeigefeld *Label*

Mit einem Anzeigefeld *Label* kann ein bestimmter Text an der gewünschten Bildschirmposition ausgegeben werden. Zwei wichtige Eigenschaften sind *Caption* und *WordWrap*. Erstere enthält den Text und Zweitere erlaubt innerhalb des Textes einen Zeilenumbruch.

```
var S: String;

S := ' Bitte eine Zahl eintasten ' + #13#10 + ' und mit <Enter> abschließen ';
Form1.Label1.Caption := S;
```

[4.06] Die Rahmenfelder *Bevel* und *Panel*

Die ***Bevel***-Komponente ist ein Rahmenobjekt zur schönen Abgrenzung von Formularteilen. Unter der Bezeichnung *Bevel* versteht man allgemein einen abgeschrägten Rand.

Das Ausgabefeld ***Panel*** dient als eine optisch begrenzte Behälterfläche (Container) für visuelle Objekte des Formulars, welche auf dem Panel platziert werden. Der *Caption*-Text ist auf eine Zeile beschränkt. 3D-Eigenschaften wie *BevelInner, BevelOuter, BevelWidth* bestimmen die Randform.

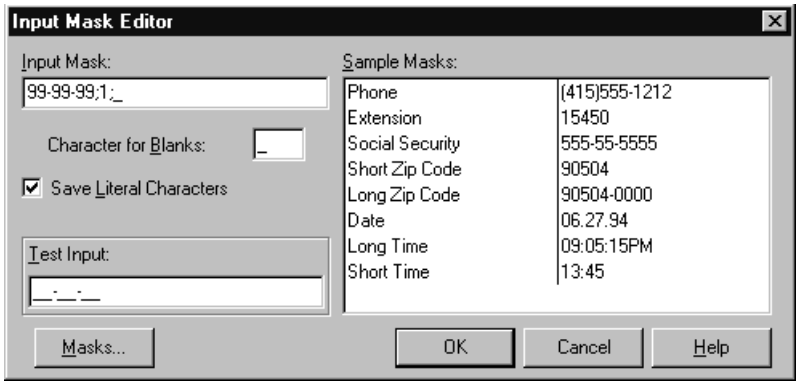
[4.07] Das einzeilige Editierfeld *Edit*

Editierfelder dienen zur Eingabe, Ausgabe und zum Editieren von einzeiligen Texten. Auf ihren Textinhalt kann mit Hilfe von Stringvariablen ***S*** zugegriffen werden.

```
Form1.Edit1.Text  :=  S;          // schreibt in den Text von Edit1 die Stringvariable S
S  :=  Form1.Edit1.Text;          // liest in die Stringvariable S den Text von Edit1
```

[4.08] Die Eingabekomponente *MaskEdit*

Die zusätzliche Komponente *MaskEdit* stellt eine sehr nützliche Erweiterung der *Edit*-Komponente dar: Sie dient der formatierten Dateneingabe. Über die Eigenschaft *EditMask* wird ein interner Maskeneditor aufgerufen, mit dessen Hilfe das Format (Maske) der eingegebenen Daten festgelegt werden kann. Die eingegebenen Daten befinden sich in der Eigenschaft *EditText*, die vom Typ *String* ist. Die Methode *Clear* löscht dieses Textfeld. Jede Eingabemaske (*EditMask*) besteht aus drei Stringfeldern, welche durch Semikolons getrennt sind. Das erste Feld enthält die gewünschte Anzahl an Steuerzeichen (L, l, A, a, C, c, 0, 9 .....). Das zweite Feld gibt an, ob bestimmte Sonderzeichen (Literale, wie beispielsweise Bindestriche) aus der Eingabemaske auch im Datentext gespeichert werden sollen. Es kann nur 0 (nein) oder 1 (ja) sein. Das dritte Maskenfeld gibt jenes Zeichen an, welches bei der Dateneingabe anstelle des Leerzeichens (Blank) angezeigt werden soll. Die nachfolgende Abbildung demonstriert den Maskeneditor.



Wichtige Steuerzeichen:

>	Alle nachfolgenden Zeichen werden in Großbuchstaben umgewandelt.
<	Alle nachfolgenden Zeichen werden in Kleinbuchstaben umgewandelt.
\	Das nachfolgende Zeichen wird als Literal im Datentext angezeigt und gespeichert.
A	An dieser Stelle muss ein alphanumerisches Zeichen eingegeben werden.
a	Wie 'A', jedoch ist hier keine Eingabe erforderlich.
L	An dieser Stelle muss ein Buchstabe (A ... Z, a ...z) eingegeben werden.
l	Wie 'L', jedoch ist hier keine Eingabe erforderlich.
0	An dieser Stelle muss eine Ziffer (0 ... 9) eingegeben werden.
9	Wie '0', jedoch ist hier keine Eingabe erforderlich.
#	An dieser Stelle kann nur eine Ziffer, '+' oder '-' eingegeben werden.
:	Trennzeichen für Stunden, Minuten und Sekunden bei Zeiteingaben.
/	Trennzeichen für Tag, Monat und Jahr bei Datumseingaben.

Ein Programmbeispiel:

Die Eingabemaske muss natürlich nicht unbedingt mit Hilfe des Maskeneditors im Objektinspektor beim Programmentwurf erstellt werden, sondern kann auch direkt im Programm erzeugt werden. Nachfolgende Anweisungen erzeugen eine Maske, welche eine Dateneingabe von nur 4 Ziffern im Editierfeld *EditText* einer Formularkomponente *MaskEdit1* erzwingt.

```
Help := '';                              // Hilfsstring Help als Leerstring initialisieren
For  i  := 1 to 4  do  Help := Help + '0';   // Help mit 4 Steuerzeichen '0' belegen
Help := Help  + ';1;_';                  // Literale speichern und Blanks mit '_' anzeigen
Form1.MaskEdit1.EditMask := Help;        // die Eigenschaft EditMask setzen

S := Form1.MaskEdit1.EditText;           // dem String S den eingegebenen Datentext zuweisen
```

[4.09] Die verschiedenen *String*-Typen

Der Variablentyp *String* ist eine Zeichenkette, welche aus Zeichen (*Char*) des erweiterten ANSI-Schriftsatzes besteht. Konstante Zeichenketten müssen immer von Hochkommas begrenzt werden (z.B. *S := 'Berger'*). Der Zugriff auf die einzelnen Zeichen der Kette erfolgt über einen Index (z.B. liefert *S[4]* das Zeichen '*g*' und *S[2] := 'u'* macht aus '*Berger*' einen '*Burger*'). Es gibt eine Vielzahl von systemdefinierten Prozeduren und Funktionen, welche der Verarbeitung von Zeichenketten dienen. Nach ihrem internen Speicherformat kann man drei Arten von *Strings* unterscheiden.

(a) Kurze Strings

Kurze Strings enthalten maximal 255 Zeichen, deren Index von 0 bis 255 geht. Im ersten Byte mit dem Index 0 ist aber kein ANSI-Code gespeichert, sondern die aktuelle Länge des Strings. Es gibt zwei Möglichkeiten, kurze Strings zu deklarieren:

```
var  S: ShortString;          // Deklaration ohne Längenangabe
     S: String[N];            // Deklaration mit Längenangabe (N)
```

(b) Lange Strings

Lange Strings sind Zeiger (Pointer), die eine 32-Bit-Adresse enthalten. Diese Adresse zeigt auf einen Speicherbereich, der folgendermaßen gegliedert ist: Die ersten vier Byte enthalten die aktuelle Länge des Strings, die zweiten vier Byte einen internen Referenzzähler und dann folgen die ANSI-Codes der einzelnen Textzeichen. Weil für die Stringlänge vier Byte vorgesehen sind, können solche Strings theoretisch bis zu vier Gigabyte lang sein.

```
var  S: String;
     C: Char;

begin
   S := 'Herbert';
   C := S[5];
   S := ' ';
   C := S[5];
end;
```

In diesem Beispiel führt der zweite Zugriff auf das fünfte Zeichen zu einem Programmabbruch, d.h., ein fehlerhafter Zugriff erfolgt, weil vorher der String leer gesetzt wurde, was der Länge Null entspricht.

(c) Nullterminierte Strings

Diese Strings sind ebenfalls Zeiger, welche auf einen Speicherbereich verweisen. Der Bereich enthält keine Längenangabe des Strings, sondern nur die ANSI-Codes der Textzeichen. Im letzten Byte dieser Zeichenkette steht aber immer der Wert 0. So wird das Ende des Strings angezeigt.

Nullterminierte Strings werden mit dem Bezeichner *PChar* deklariert. Mit Hilfe entsprechender Typumwandlungen ist es einfach möglich, lange Strings in nullterminierte Strings umzuwandeln und auch umgekehrt.

```
var S: PChar;
    T: String;

begin
   S := 'Herbert';
   T := String(S);          // Erste Typumwandlung
   ShowMessage(T);
   T := 'Susanne';
   S := PChar(T);           // Zweite Typumwandlung
   ShowMessage(S);
end;
```

Nullterminierte Strings wurden deswegen in DELPHI eingeführt, um Kompatibilität zur Sprache ***C*** zu gewährleisten, weil die Programmiersprache ***C*** nur solche Strings kennt und viele Codeteile des Betriebssystems WINDOWS in dieser Sprache programmiert sind.

[4.10] Die Objektklassen *TStrings* und *TStringList*

Die abstrakte Objektklasse *TStrings* ist eine wichtige Klasse der VCL (Visual Component Library) von Delphi. Davon direkt abgeleitet ist die Objektklasse *TStringList*, welche nützliche Methoden zur Verwaltung von Stringlisten zur Verfügung stellt. Viele visuelle Komponenten (Memo, RichEdit, Listbox) verwenden intern solche Stringlisten, welche beim Programmentwurf mit Hilfe eines internen Stringlisten-Editors bearbeitet werden können. In der Klassenhierarchie ist *TStrings* ein direkter Nachfahre von der noch allgemeineren Klasse *TPersistent*.

Will man eine Stringliste *MyList* programmieren, dann kann diese zuerst im öffentlichen Teil der Typdeklaration von *TForm1* als Objekt der Basisklasse *TStrings* definiert werden. Es ist aber durchaus möglich, *MyList* auch als Objekt von *TStringList* zu deklarieren. Auch kann *MyList* mit Hilfe einer einfachen Variablendefinition im Implementierungsteil der Programmunit deklariert werden. Diese letztgenannte Möglichkeit wird durch folgende Anweisung realisiert:

```
var  MyList: TStringList;
```

In der *OnCreate*-Routine des Formulares wird sodann beim Programmstart die eigentliche Listenstruktur im Speicher dynamisch angelegt:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   MyList := TStringList.Create;
end;
```

Beim Schließen des Formulares sollte der Ordnung halber die erzeugte Liste wieder aus dem Speicher entfernt werden, was in der *OnDestroy*-Routine des Formulares geschieht.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
   MyList.Free;
end;
```

Nachdem diese Rahmenbedingungen geschaffen worden sind, stehen eine Vielzahl von Verwaltungsroutinen zur Verfügung. Jedes Listenelement von *MyList* stellt einen String dar, der durch einen Index angesprochen werden kann. Dieser beginnt bei *Null* und endet bei *MyList.Count-1*.

```
ANZ := MyList.Count;                 // weist ANZ die Anzahl der Listenelemente zu

S := MyList[N];                      // weist dem String S das N-te Listenelement zu
MyList[N] := S;                      // weist dem N-ten Listenelement den String S zu
MyList.Add(S);                       // fügt den String S an das Ende der Liste hinzu
```

```
MyList.Insert(N,S);                 // fügt ein neues Listenelement an der N-ten Position ein
MyList.Delete(N);                   // löscht das N-te Listenelement
MyList.Clear;                       // löscht die ganze Liste

N := MyList.IndexOf(S);             // sucht  in der Liste den String S und übergibt bei Erfolg
                                    // den Index des gefundenen Listenelementes, ansonsten -1
S := MyList.Text;                   // liefert alle Listenelemente in einem einzigen String, wo sie
                                    // durch EndOfLine-Markierungen (#13#10) getrennt sind.

MyList.LoadFromFile(FileName);      // ladet eine Textdatei (mit FileName) in die Liste
MyList.SaveToFile(FileName);        // speichert die Liste auf eine Textdatei (mit FileName)

Memo1.Lines.Clear;                  // löscht den Memoinhalt und
Memo1.Lines.Assign(MyList);         // kopiert die ganze Liste in das Memo
MyList.Clear;                       // löscht die Liste und
MyList.Assign(Memo1.Lines);         // kopiert den ganzen Memoinhalt in die Liste
```

[4.11] Die mehrzeiligen Editierfelder *Memo* und *RichEdit*

Memofelder dienen zur Eingabe, Ausgabe und zum Editieren von mehrzeiligen Texten. Die Textzeilen (*Lines*) eines Memofeldes sind nichts anderes als indizierte Einträge einer Stringliste, wobei der ersten Zeile der Index 0 zugeordnet ist. Bei der Einrichtung der Komponente mittels Objektinspektor kann beim Entwurf eines Programmes der interne Stringlisten-Editor verwendet werden, um zeilenweise Einträge in die Komponente zu schreiben. In einem Memofeld gelten alle Eigenschaften und Methoden der Objektklasse *TStringList*, welche weiter oben sehr ausführlich beschrieben wurden. So sind beispielsweise folgende Anweisungen möglich:

```
var  S: String;
   n,i: Integer;

with Form1.Memo1 do begin
   Lines[i] := S;              // String S in die i-te Textzeile schreiben
    S := Lines[i];             // die i-te Textzeile in den String S auslesen
   Lines.Clear;                // Inhalt des Memos löschen
   n := Lines.Count;           // liest die Zeilenanzahl des Memos in die Variable n
   Lines.Add(S);               // fügt den String S als neue Memozeile hinzu
   Lines.Delete(i);            // löscht die i-te Memozeile
   Lines.Insert(i,S);          // fügt den String S an die i-te Zeile ein
end;
```

In einem Memofeld kann mit der Maus oder der Tastatur, so wie in jeder Textverarbeitung auch, ein Textblock markiert werden. Dieser Bereich kann mit Hilfe von *SelText* direkt auf eine Stringvariable ausgelesen werden. Außerdem geben die Integervariablen *SelStart* und *SelLength* die Anfangsposition und die Zeichenanzahl des markierten Bereiches innerhalb des gesamten Memotextes an. *SelectAll* markiert den gesamten Memotext. Ist kein Textbereich markiert, dann liefert *SelStart* den Index des Textzeichens an der aktuellen Cursorposition und der Befehl *SelText := S* fügt dort den String *S* in den Memotext ein. Umgekehrt setzt die Anweisung *SelStart := N* den Mauscursor auf das N-te Zeichen im Memofeld.

```
with Form1.Memo1do begin
   SelStart := 150;                 // Setzt den Cursor auf das 150-te Textzeichen
   SelText := 'Herbert';            // und fügt dort den String 'Herbert' ein.
end;
```

Zum Laden und Speichern von Textdateien in ein Memofeld werden sehr mächtige Routinen zur Verfügung gestellt: *Memo1.Lines.LoadFromFile(FName)* bzw. *Memo1.Lines.SaveTo(FName)*, wo der String *FName* den Dateinamen bezeichnet.

Ein Memo wird im Normalfall dazu benutzt, um eine Stringliste visuell darzustellen. Will man beispielsweise mit der Tastenkombination <Strg> <Enter> einen Seitenvorschub für den Drucker (Zeichencode 12) an die aktuelle Cursorposition (*SelStart*) in den Memotext einfügen, dann kann eine Tastatur-Ereignisbehandlung *OnKeyUp* folgendermaßen programmiert werden:

```
procedure TForm1.Memo1KeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if (Shift = [ssCtrl]) and (Key = 13) then Memo1.SelText := #12;
end;
```

Die Eigenschaft *Text* liefert nicht einzelne Zeilen, sondern den gesamten Textinhalt des Memofeldes. Mit der Eigenschaft *WordWrap* wird ein automatischer Zeilenumbruch am rechten Rand erzwungen. *ReadOnly* ermöglicht nur das Lesen, aber nicht das Schreiben eines Memotextes zur Laufzeit. Mit der Eigenschaft *Scrollbars* können horizontale und vertikale Bildlaufleisten eingeblendet werden. Die Methode *Undo* versucht, die letzte Aktion rückgängig zu machen.

Auch die Technik von "*Drag and Drop*" ist im Memofeld implementiert und schließlich sind auch die in WINDOWS üblichen Interaktionen mit der Zwischenablage (*ClipBoard*) möglich. Die Methode *CutToClipBoard* (Taste <*Strg X*>) schneidet einen markierten Textblock aus und transportiert ihn in die Zwischenablage, *CopyToClipBoard* (Taste <*Strg C*>) kopiert den markierten Block und *PasteFromClipBoard* (Taste <*Strg V*>) fügt einen Text aus der Zwischenablage an die aktuelle Position in das Memofeld ein. Für diese Aktionen gibt es sogar ein eigenes Kontextmenü.

Wie aus all diesen Erklärungen ersichtlich ist, wird mit der *Memo*-Komponente eine Funktionalität zur Verfügung gestellt, welche den Editor-Funktionen von *Notepad* von WINDOWS entspricht.

Die Komponente "*RichEdit*"

RichEdit ist prinzipiell ein erweitertes Memofeld mit der Möglichkeit, den Text zu formatieren. Um reinen ANSI-Text ohne Formatierung zu verwenden, muss die Eigenschaft *PlainText* auf *True* gesetzt werden. Dann ist *RichEdit* nichts anderes als ein Memofeld.

Zur Textformatierung verwendet *RichEdit* das so genannte **RTF**-Format (Rich Text Format). Die eigentlichen Befehle zur Textauszeichnung sind im Text eingebettet, werden aber nicht angezeigt. Sichtbar sind nur ihre Auswirkungen auf den dargestellten Text. Für die Formatierung sind die Eigenschaften *DefAttributes* und *SelAttributes* vom allgemeinen Typ *TTextAttributes* zuständig. Die erste beinhaltet alle Standard-Einstellungen für den gesamten Text, die zweite enthält die Schriftmerkmale für einen bereits markierten Text oder den neu einzugebenden Text.

Der allgemeine Typ *TTextAttributes* besitzt alle wichtigen Merkmale zur Textformatierung: Name des Zeichensatzes (*Name*), Schriftgröße (*Size*), Schriftauszeichnung (*Style*: fett, kursiv, unterstrichen ...) und Schriftfarbe (*Color*). Erwähnenswert ist, dass *Style* den Datentyp *Set* (Menge) aufweist. Damit ist *TTextAttributes* fast gleich in seinen Inhalten wie der allgemeine Schrifttyp *TFont*. Daher ist es auch möglich, Instanzen beider Objektklassen einander zuzuweisen. So können beispielsweise die aktuellen Textattribute in einem *Font*-Objekt abgespeichert werden und umgekehrt kann mit Hilfe einer *FontDialog*-Komponente eine Schrift ausgewählt werden.

```
with Form1 do begin
   FontDialog1.Font.Assign(RichEdit1.SelAttributes);
   if FontDialog1.Execute  then RichEdit1.SelAttributes.Assign(FontDialog1.Font);
   RichEdit1.SelAttributes.Style := RichEdit1.SelAttributes.Style + [fsBold];
   RichEdit1.SelAttributes.Size  := RichEdit1.SelAttributes.Size + 2;
end;
```

Zwei weitere, wichtige Eigenschaften von *Memo*- und *RichEdit*-Komponenten sind *CaretPos.X* und *CaretPos.Y*. Die Erste liefert die Spalte und die Zweite liefert die Zeile der aktuellen Cursorposition im Text.

Eine spezielle Methode von *RichEdit* ist *FindText*. Dabei wird ein Textbereich ab einer Startposition (MyStart) und mit einer bestimmten Länge (MyLength) nach einem Suchtext (MyText) durchsucht. Mit Hilfe eines vierten Parameters kann festgelegt werden, ob die Suche abhängig von Groß- oder Kleinschrift (*stMatchCase*) und nur nach ganzen Worten (*stWholeWord*) erfolgen soll. Im Erfolgsfall wird die Position der Fundstelle zurückgeliefert, ansonsten der Wert -1. Zusätzlich kann noch der gefundene Text markiert werden.

```
var  MyText:  String;
     MyStart, MyLength:  Integer;
     Found: Integer;

begin
   MyText := 'Herbert';
   MyStart := 0;
   MyLength := Length(RichEdit1.Text) - 1;
   Found := RichEdit1.FindText(MyText, MyStart, MyLength, [stWholeWord]);
   if Found <> -1 then begin
      RichEdit1.SelStart := Found;
      RichEdit1.SelLength := Length(MyText);
   end;
end;
```

Im Buchteil [Delphi 05] ist ein menügesteuerter Texteditor "***textdemo***" programmiert, der als zentrales Bestandteil eine *RichEdit*-Komponente enthält. Für weitere Informationen sei der Leser auf dieses Projekt verwiesen.

[4.12] Die Auswahlfelder *ListBox* und *ComboBox*

In einer *ListBox* ist eine Auflistung von Einträgen (*Items*) enthalten, von der man mittels Maus oder Tastatur einen oder mehrere auswählen kann. Eine *ComboBox* ist eine Kombination aus einem zusätzlichen *Eingabefeld* und einer *ListBox*. Im Eingabefeld kann der ausgewählte Listeneintrag angezeigt bzw. eingegeben werden. Ähnlich einem *Memo* sind auch *List*- und *ComboBoxen* als Stringlisten organisiert. Die einzelnen Einträge werden hier jedoch *Items* genannt und der Zugriff auf sie erfolgt über einen *ItemIndex*, der bei Null beginnt. Die Eigenschaft *Items.Count* liefert die Anzahl aller Listeneinträge der Box. *TopIndex* gibt den Index jenes Eintrages an, der an der Spitze der Box angezeigt wird. Beim Entwurf der Komponente kann der interne Stringlisten-Editor verwendet werden, um zeilenweise Einträge in die Komponente zu schreiben.

```
var  n, i: Integer;
     S: String;

with Form1.ListBox1 do begin
     n := ItemIndex;                  // liest den Index des ausgewählten Eintrages
     S := Items[ItemIndex];           // liest den ausgewählten Eintrag in den String S
     Items.Add(S);                    // fügt den String S als neuen Eintrag hinzu
     Items.Delete(i);                 // löscht den i-ten Eintrag
     Items.Clear;                     // löscht den gesamten Inhalt der Box
end;
```

[4.13] Die Wahlschalter *RadioButton* und *RadioGroup*

Eine *RadioGroup* besteht aus mehreren einfachen *RadioButtons*, die nichts anderes als alternative Auswahlschalter sind. Intern ist eine *RadioGroup* ähnlich wie eine *ListBox* als Stringliste organisiert, deren Einträge *Items* genannt werden. Die ganzzahlige Eigenschaftsvariable *ItemIndex* bestimmt, welcher Eintrag ausgewählt wurde (-1 = keine Auswahl, 0 = erster Eintrag, 1 = zweiter Eintrag usw.). Bei der Einrichtung der Komponente mittels Objektinspektor kann der interne Stringlisten-Editor verwendet werden, um zeilenweise Einträge in die Komponente zu schreiben.

```
var  n, i :  Integer;

with Form1.RadioGroup1 do begin
   Columns := 5;                  // platziert die Radiobuttons in 5 Spalten
   For i := 1 to 5 do             // beschriftet die 5 Radiobuttons mit den Ziffern 1,2 ... 5
      Items. Add(chr(48+i));
    n := ItemIndex;               // ermittelt den gewählten RadioButton (-1,0,1,2 ... )
   ItemIndex := n;                // schaltet den n-ten RadioButton ein
end;
```

[4.14] Die Menükomponente *MainMenu*

Die Standardkomponente *MainMenu* ist zur Laufzeit unsichtbar. Es ist deshalb unwichtig, wo sie zur Entwurfszeit am Formular platziert wird. Beim Entwurf steht ein eigener Menü-Editor zur Verfügung, der durch Doppelklick auf die Komponente aufgerufen wird. Die dabei eingerichteten Menüeinträge sind indizierte Objekte der Klasse *TMenuItem* mit Eigenschaften und Ereignissen.

Die *Checked*-Eigenschaft setzt ein Häkchen vor den gewählten Menüeintrag (*True/False*). *Enabled* aktiviert bzw. deaktiviert den Menüeintrag, *Visible* macht ihn sichtbar bzw. unsichtbar. *ShortCut* dient der Verknüpfung eines Eintrages mit einer Tastenkombination. Ein "**&**" vor einem Buchstaben in der Beschriftung (*Caption*) eines Menüeintrages erlaubt die Auswahl dieses Eintrages durch gleichzeitiges Drücken der <*Alt*>-Taste und der Buchstabentaste. Ein einzelnes "**-**" in der Beschriftung bewirkt eine optische Trennung der Menüeinträge. Durch einen Doppelklick auf den eingerichteten und ausgewählten Menüeintrag im Formular kann beim Entwurf die Programmschablone einer entsprechenden *OnClick*-Ereignisbehandlungsroutine erzeugt werden. Diese wird dann zur Laufzeit bei Anwahl des Menüeintrages ausgeführt. Sie könnte beispielsweise für einen Menüeintrag *Beenden* folgendermaßen aussehen:

```
procedure TForm1.BeendenClick(Sender: Object);
begin
   Application.Terminate;
end;
```

Um ein Menü einzurichten, muss man sich erst an das Wechselspiel zwischen Menü-Editor und Objekt-Inspektor gewöhnen. Ein neuer Menüeintrag wird im Fenster des Menü-Editors mit einem leeren Menükästchen dargestellt. Durch einen Klick darauf wechselt man automatisch zum Objekt-Inspektor. Dort wird zunächst in der Caption-Eigenschaft der Menüeintrag beschriftet. Die Betätigung der <Enter>-Taste überträgt diese Beschriftung automatisch auf den Namen des aktuellen Eintrages, wobei aber eine zusätzliche Kennziffer hinzugefügt wird. Lautet beispielsweise die Caption eines Menüeintrages *Speichern*, dann wird sein Name zu *Speichern**1***. Die Kennziffer wird immer dann um Eins erhöht, wenn eine Caption eingegeben wird, die bereits vorhanden ist. Dadurch ist sichergestellt, dass es keine doppelten Namen für Menüeinträge gibt. Das Drücken der <Enter>-Taste bewirkt außerdem die Anlage eines neuen leeren Menükästchens für den nächsten Menüeintrag. Das Gesagte gilt sowohl für die Haupteinträge in der horizontalen Menüleiste als auch für die Nebeneinträge in den vertikalen PullDown-Menüs.

Ein Klick mit der rechten Maustaste auf einen bereits fertig eingerichteten Menüeintrag öffnet ein lokales Hilfsfenster des Menü-Editors, wodurch zusätzliche Auswahlen zur Verfügung stehen. Damit kann zu dem aktuellen Menüeintrag ein Untermenü (*SubMenu*) erzeugt, der Eintrag gelöscht oder an seine Stelle ein neuer Eintrag eingefügt werden.

[4.15] Die *Dialog*-Objekte

Weil viele Routineaufgaben in sehr vielen Anwendungen vorkommen, verfügt WINDOWS über einige Standarddialoge, die jede Anwendung in gleicher Weise benutzen kann. In DELPHI sind daraus leicht handhabbare Komponenten geworden.

Im Gegensatz zu den Steuerelementen ist ein solcher Dialog zur Entwurfszeit nur durch ein kleines Icon symbolisiert, während er erst zur Laufzeit seine Funktionalität anbietet. Sämtliche Dialogobjekte befinden sich in der Registerseite *Dialog* der Komponentenpalette.

TOpenDialog	dient der Dateiauswahl zum Laden von Dateien.
TSaveDialog	dient der Dateiauswahl zum Speichern von Dateien.
TColorDialog	ermöglicht bequeme Farbauswahlen.
TFontDialog	ermöglicht den Zugriff auf die von Windows verwendeten Schrifttypen.
TPrintDialog	stellt verschiedene Druckoptionen zur Verfügung.
TPrinterSetUp	ermöglicht die Einstellung des Druckers.
TFindDialog	dient dem bequemen Finden von Suchbegriffen in einem Text.
TReplaceDialog	dient dem bequemen Finden und Ersetzen von Begriffen in einem Text.

Alle Dialog-Komponenten besitzen bestimmte Properties, die für den Dialog mehr oder minder wichtig sind. Beispielsweise enthält *FileName* den Dateinamen bei den Dateiauswahl-Dialogen. Bestimmte Properties sind vor dem Dialog einzugeben, andere werden wieder nach dem Dialog abgefragt. Der Aufruf eines Dialogs erfolgt immer mit der logischen Funktion ***Execute***, die dann den Wert *True* liefert, wenn im Dialog <*OK*> bzw. <*Öffnen*> gedrückt wird. Bei <*Abbrechen*> liefert die Funktion den Wert *False*.

[4.16] Spezielle *Listboxen* zur Dateiauswahl

Auf der Registerseite *Win 3.1* der Komponentenpalette findet man mehrere spezielle Listboxen, welche dem bequemen Zugriff auf Dateien dienen. Zuerst soll eine ***DriveComboBox*** eingerichtet werden. Diese aufklappbare Liste bietet die zur Verfügung stehenden Laufwerke an. Das gewählte Laufwerk kann über die Property *Directory* abgefragt werden. Dann wird eine ***DirectoryListBox*** am Formular installiert, welche die Verzeichnisse und optional deren Unterverzeichnisse anzeigt. Damit die *DriveComboBox* automatisch mit der *DirectoryListBox* verknüpft wird, muss die Eigenschaft *DirList* der *DriveCombobox* mittels Objektinspektor auf die *DirectoryListBox* gesetzt werden. Nun wird eine ***FilterComboBox*** eingerichtet, mit deren Eigenschaft *Filter* die Anzeige der Dateien beschränkt werden kann. Sowohl in der FilterComboBox als auch in der DirectoryListBox wird die Eigenschaft *FileList* auf die Komponente ***FileListBox*** gesetzt, welche als Nächstes am Formular zu erzeugen ist. Dort kommt es schließlich zur Anzeige der Dateien in der Form, dass jeder gefundenen Datei ein Item der Box entspricht. Wird ein solches mit Mausklick ausgewählt, dann liefert die Eigenschaft *FileName* den Namen und die Eigenschaft *Directory* den Verzeichnispfad der ausgewählten Datei. Mit *FileTyp* kann sogar das Dateiattribut spezifiziert werden.

Beispielsweise liest der Befehl *S := Form1.FileListBox1.FileName* den Dateinamen auf die Stringvariable *S*. Handelt es sich dabei um eine Textdatei, so kann sie mit Hilfe der entsprechenden Datei-Verwaltungsroutinen zeilenweise in ein Memofeld transferiert werden. Handelt es sich um eine Grafikdatei vom Typ Bitmap (*.bmp) oder MetaFile (*.wmf), dann kann sie mit Hilfe der Grafik-Laderoutine der Komponente *Image.Picture* in ein Image des Formulares transferiert werden. Das Programm "***scan***" im Abschnitt [8.06] demonstriert den Einsatz dieser Komponenten.

[4.17] Die Zeitkomponente *Timer*

Auf der Registerseite *System* der Komponentenpalette befindet sich der *Timer*. Dieser kann im Programm über das *OnTimer*-Ereignis maximal 18,2-mal pro Sekunde aufgerufen werden. Der Zeitabstand der Timer-Aufrufe wird durch die Eigenschaft *Interval* bestimmt, welche in Milli-Sekunden anzugeben ist, d.h., 1000 entspricht einer Sekunde. Die vordefinierte Variable *Time* vom Typ *TDateTime* enthält die Systemzeit. Diese kann durch die vordefinierte Procedure *DecodeTime* in Stunde, Minute, Sekunde und Millisekunde zerlegt werden oder man wandelt sie ganz einfach mittels *TimeToStr* in eine entsprechende Zeichenkette um.

```
procedure TForm1.Timer1Timer(Sender: TObject);      // Zeitanzeige im eingestellten Intervall;
begin                                               // bei Intervall 1000 ergibt das eine Digitaluhr.
    Label1.Caption := TimeToStr(Time);
end;

function Zeit: Integer;                             // wandelt die ermittelte Zeit in Millisekunden um;
var Hour, Min, Sec, MSec : Word;                    // ein zweimaliger Aufruf dieser Funktion ermöglicht
    Z : Integer;                                    // die Zeitmessung zwischen zwei Ereignissen.
begin
    DecodeTime(Time,Hour,Min,Sec,MSec);
    Z := (Sec+60*Min+3600*Hour)*1000 + MSec;
    Result := Z;
end;
```

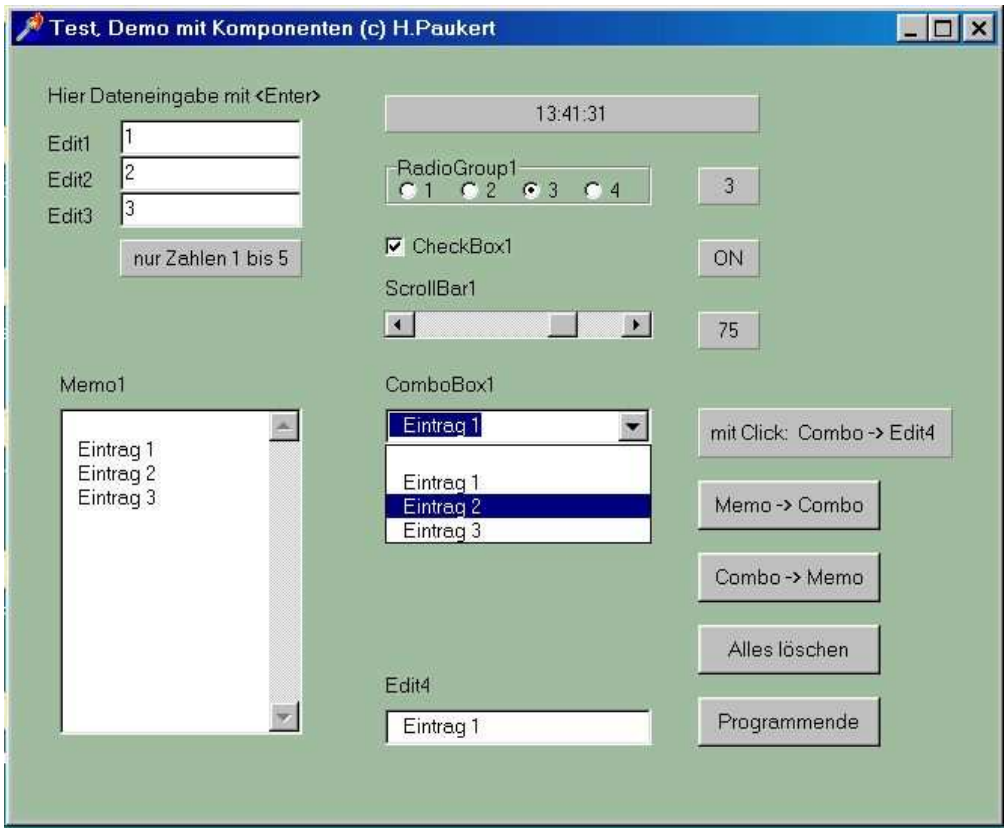
[4.18] Der *Scrollbar* als Schieberegler

Eine simple Bildlaufleiste *Scrollbar* kann auch als Schieberegler zur Einstellung von diskreten Werten dienen. Als Beispiel soll ein horizontaler *Scrollbar* zur Schieberegelung ganzzahliger Werte von 0 bis 100 und daneben ein *Panel* zur Ausgabe dieser Werte dienen. Dazu kann in der *OnCreate*-Ereignisbehandlungsroutine des Formulares der entsprechende *Scrollbar* eingerichtet werden. Mit Hilfe der *OnScroll*-Ereignisbehandlungsroutine des *Scrollbars* wird dann der eingestellte Wert am *Panel* angezeigt. Der zu skalierende *Wert* sei eine globale Integervariable.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with ScrollBar1 do begin
      Min := 0;                                 // unterer Skalenwert
      Max := 100;                               // oberer Skalenwert
      Position := 50;                           // aktuelle Skalenposition
      Kind := sbHoricontal;                     // horizontale Ausrichtung
      LargeChange := 10;                        // Grobabstufung beim Klicken auf die Leiste
      SmallChange := 1;                         // Feinabstufung beim Klicken auf Randpfeile
  end;
   Panel1.Caption := IntToStr(Scrollbar1.Position);          // Anzeige am Panel
end;

procedure TForm1.Scrollbar1Scroll(Sender : TObject; ScrollCode: TScrollCode;
                                  var ScrollPos: Integer);
var Wert: Integer;
begin
  Wert := ScrollPos;                            // der Variablen Wert die Schieberposition
  Panel1.Caption := IntToStr(Wert);             // zuweisen und am Panel ausgeben
end;
```

Natürlich muss die Einrichtung des *Scrollbars* nicht im Programm, also zur Laufzeit erfolgen, sondern kann auch beim Programmentwurf mittels Objektinspektor durchgeführt werden. Aber das gilt ja für alle Komponenten.

Das Programm "***test***" demonstriert die Verwendung von verschiedenen Steuerkomponenten. In den Editierfeldern *Edit1*, *Edit2* und *Edit3* erfolgt der Eingabeabschluss mit der <Enter>-Taste, worauf sofort der Inhalt des anderen Feldes gelöscht und zu diesem gesprungen wird. Dabei können nur ganze Zahlen von 1 bis 5 eingegeben werden. Weiters befinden sich im Formular eine Gruppe von *RadioButtons* und eine *CheckBox*, sowie ein *Timer*-Objekt zur digitalen Anzeige der Uhrzeit. Ein einfacher *Scrollbar* wird als Schieberegler zur Einstellung von Zahlenwerten verwendet. Mit Hilfe von entsprechenden Schaltknöpfen können die Inhalte von einem *Memo* in eine *Combobox*, von der *ComboBox* zurück in das *Memo* und vom Auswahlfeld der *ComboBox* in das Editierfeld *Edit4* transferiert werden.




```
unit test_u;
// Test, Arbeiten mit Komponenten (c) H.Paukert

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Memo1: TMemo;
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Panel4: TPanel;
    Panel5: TPanel;
    Panel6: TPanel;
    Timer1: TTimer;
    ScrollBar1: TScrollBar;
    RadioGroup1: TRadioGroup;
    CheckBox1: TCheckBox;
    ComboBox1: TComboBox;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
```

```
    procedure FormCreate(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure Timer1Timer(Sender: TObject);
    procedure ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
                               var ScrollPos: Integer);
    procedure RadioGroup1Click(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure ComboBox1KeyUp(Sender: TObject; var Key: Word;
                             Shift: TShiftState);
    procedure ComboBox1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    private { Private declarations }
    public  { Public declarations }
  end;

var Form1: TForm1;

implementation
{$R *.DFM}

var X,Y,Z: Integer;

function IntegerInput(E: TEdit; Min,Max: Integer): Integer;
{ Hilfsroutine zur gesicherten Zahleneingabe im Editierfeld E }
var N , Code : Integer;
    Error    : Boolean;
    S        : String;
begin
  Error := False;
  S := E.Text;
  Val(S,N,Code);
  if (Code<>0) or (N<Min) or (N>Max) then Error := True;
  if Error then begin
     E.Text := '';
     E.SetFocus;
     Result := -1;
  end
  else Result := N;
end;

procedure TForm1.FormCreate(Sender: TObject);
{ Initialisierungen zum Programmbeginn }
begin
  Color := RGB(160,190,160);
  KeyPreview := True;                  // Tastaturnachrichten zum Formular
  with ScrollBar1 do begin             // Einrichtung des Scrollbars
       Min := 0;
       Max := 100;
       Position := 50;
       Kind := sbHorizontal;
       LargeChange := 10;
       SmallChange := 1;
  end;
  Panel3.Caption := IntToStr(ScrollBar1.Position);
  RadioGroup1.ItemIndex := 0;          // Initialisierung der Radiogroup
  CheckBox1.Checked := False;          // Initialisierung der Checkbox
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
{ Zentrale Verarbeitungen der Tastaturereignisse im Formular }
begin
  if (Sender=Edit1) and (Key=#13) then begin
    X := IntegerInput(Edit1,1,5);
    if X <> -1 then Edit2.SetFocus;
  end;
```

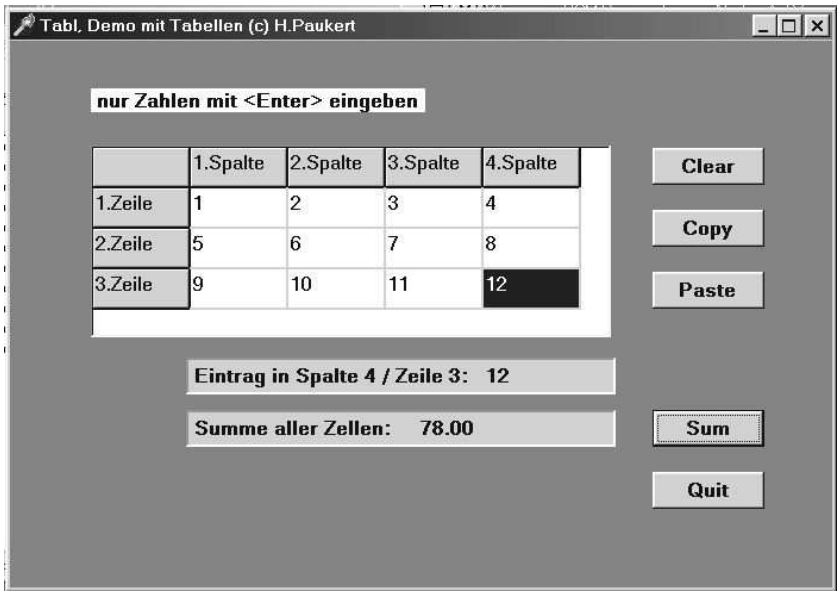
```
  if (Sender=Edit2) and (Key=#13) then begin
    Y := IntegerInput(Edit2,1,5);
    if Y <> -1 then Edit3.SetFocus;
  end;
  if (Sender=Edit3) and (Key=#13) then begin
    Z := IntegerInput(Edit3,1,5);
    if Z <> -1 then Edit1.SetFocus;
  end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
{ Erzeugung einer digitalen Zeitanzeige mit Hilfe des Timers }
begin
  Panel5.Caption := TimeToStr(Time);
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
                                  var ScrollPos: Integer);
{ Verwendung des Scrollbars als Schieberegler }
{ die Reglerposition wird der zentralen Variablen Z zugewiesen }
begin
  Z := ScrollPos;
  Panel3.caption := IntToStr(Z);
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
{ Anzeigen des gedrückten Radiobuttons }
begin
  Panel2.Caption := IntToStr(RadioGroup1.ItemIndex+1);
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
{ Anzeigen des Schaltzustandes der Checkbox }
begin
  if CheckBox1.Checked then Panel4.Caption := 'ON'
                       else Panel4.Caption := 'OFF';
end;

procedure TForm1.ComboBox1KeyUp(Sender: TObject; var Key: Word;
                                Shift: TShiftState);
{ Dateneingabe in der Combobox mit Enter-Taste }
begin
  if Key = 13 then begin
     ComboBox1.Items.Add(ComboBox1.Text);
     ComboBox1.Text := '';
  end;
  ComboBox1.DroppedDown := True;
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
{ Selektiver Datentransfer von der Combobox zu Edit4 }
begin
  Edit4.Text := ComboBox1.Items[ComboBox1.ItemIndex];
end;

procedure TForm1.Button1Click(Sender: TObject);
{ Datentransfer von Memo zur Combobox }
begin
  ComboBox1.Items := Memo1.Lines;
end;

procedure TForm1.Button2Click(Sender: TObject);
{ Globaler Datentransfer von der Combobox zum Memo }
begin
  Memo1.Lines := ComboBox1.Items;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
{ Alle Objektfelder löschen bzw. initialisieren }
begin
  Memo1.Clear;
  ComboBox1.Clear;
  Edit1.Text := ''; Edit2.Text := ''; Edit3.Text := ''; Edit4.Text := '';
  Panel2.Caption := ''; Panel3.Caption := ''; Panel4.Caption := '';
  ScrollBar1.Position := 0;
  RadioGroup1.ItemIndex := 0;
  CheckBox1.Checked := False;
end;

procedure TForm1.Button4Click(Sender: TObject);
{ Programm beenden }
begin
   Application.Terminate;
end;

end.
```

[4.19] Das Tabellengitter *StringGrid*

StringGrid dient zur Eingabe, Editieren und Ausgabe von Textdaten in Tabellenform. Diese zusätzliche Komponente stellt eine zweidimensional-indizierte Tabelle mit *ColCount* Spalten *(x)* und *RowCount* Zeilen *(y)* zur Verfügung. Die einzelnen Tabelleneinträge können wie in Editierfeldern eingegeben werden, und es wird mittels *Cells[x,y]* auf sie zugegriffen. Die Einträge sind einfache Zeichenketten. Damit in der Tabelle auch editiert werden kann, müssen unbedingt die beiden Optionen *[goEdit]* und *[goTabs]* auf *True* gesetzt werden. Die Indizierung beginnt immer bei Null, was einer Randzeile bzw. Randspalte entspricht. Mit der Eigenschaft *Scrollbars* können Bildlaufleisten gesetzt und die Tabelle entsprechend gescrollt werden. Will man die Randelemente der Tabelle beim Scrollen fixieren, anders einfärben und von der Edition ausschließen, so erfolgt dies mit den Eigenschaften *FixedRows*, *FixedCols* und *FixedColor*. Die Integer-Variablen *Row* und *Col* liefern die Indizes der Zeile und der Spalte jener Zelle, welche in der Tabelle aktuell ausgewählt wurde, beispielsweise durch ein Maus- oder ein Tastaturereignis. In den entsprechenden Ereignisbehandlungsroutinen (*OnMouseClick* oder *OnKeyUp*) könnte dann der ausgewählte Zelleneintrag einer Stringvariablen *S* zugewiesen werden (*S := StringGrid1.Cells[Col,Row]*).

Das Programm "***tabl***" demonstriert die Erstellung und Verarbeitung von Tabellen. In der Routine *TableInit* wird zunächst eine 5 x 4-Tabelle initialisiert. Es werden die Tabellenränder beschriftet und alle Einträge auf 0 gesetzt. Dann kann der nicht fixierte Tabellenbereich (d.h. alles ohne die Ränder) in ein reellzahliges 4 x 3-Speicherarray kopiert und wieder zurückgeholt werden. Als Zusätze sind noch eine sichere Zahleneingabe und die Summierung aller Einträge programmiert.



```
type TMatrix = Array[1..4,1..3] of Real;  // Daten-Matrix
var  MAT : TMatrix;


procedure TableInit(TB: TStringGrid);
{ Tabellenränder und Einträge initialisieren }
var x, y: Integer;
begin
  With TB do begin
    For x := 1 to 4 do Cells[x,0] := IntToStr(x)+ '.Spalte';
    For y := 1 to 3 do Cells[0,y] := IntToStr(y)+ '.Zeile';
    For y := 1 to 3 do
      For x := 1 to 4 do
        Cells[x,y] := FloatToStr(0);
  end;
end;

procedure TableToMatrix(TB: TStringGrid; var M: TMatrix);
{ Tabelle ins Speicherarray M kopieren }
var x, y, Error : Integer;
          Zahl  : Real;
          S     : String;
begin
   For y := 1 to 3 do begin
      For x := 1 to 4 do begin
        S  :=  TB.Cells[x,y];
        val(S,Zahl,Error);
        M[x,y] := Zahl;
      end;
   end;
end;

procedure MatrixToTable(TB: TStringGrid; var M : TMatrix);
{ Speicherarray M in die Tabelle kopieren }
var x, y : Integer;
    Zahl : Real;
    S    : String;
begin
   For y := 1 to 3 do begin
      For x := 1 to 4 do begin
        Zahl := M[x,y];
        Str(Zahl:8:2,S);
        TB.Cells[x,y] := S;
      end;
   end;
end;

function TableSum(TB: TStringGrid): Real;
{ Alle Zellen der Tabelle summieren und die Summe ausgeben }
var x, y, Error : Integer;
    Zahl, Sum   : Real;
          S     : String;
begin
  Sum := 0;
  For y := 1 to 3 do begin
      For x := 1 to 4 do begin
        S  :=  TB.Cells[x,y];
        val(S,Zahl,Error);
        Sum := Sum + Zahl;
      end;
  end;
  Result := Sum;
end;

procedure TForm1.StringGrid1KeyPress(Sender: TObject; var Key: Char);
{ Sichere Dateneingabe mit <Enter> }
var S : String;
    Z : Real;
    x,y,Err : Integer;
```

```
begin
  with StringGrid1 do begin
    if Key = #13 then begin
      x := Col; y := Row;
      S := Cells[x,y];
      val(S,Z,Err);
      if Err > 0 then Cells[x,y] := '';
      if Err = 0 then begin
         x := x + 1;
         if x > 4 then begin x := 1; y := y + 1; end;
         if y > 3 then begin y := 1; end;
      end;
      Col := x; Row := y;
    end;
  end;
end;
```

[4.20] Die Objektklassen *TPoint* und *TRect*

Beide Strukturen werden sehr häufig benutzt. *TPoint* ist ein Record, dessen zwei Felder vom Typ Integer sind und als Koordinaten eines Punktes Verwendung finden. Vor allem bei der Grafikprogrammierung leisten *TPoint*-Objekte nützliche Dienste.

```
type  TPoint = record
         X: Integer;
         Y: Integer;
      end;
```

TRect ist ein so genannter ***varianter Record***, d.h., er kann wahlweise in zwei Arten verwendet werden. Einmal als Record mit vier Integer-Feldern, welche die Koordinaten der Eckpunkte eines Rechtecks angeben, oder als Record mit nur zwei Feldern, welche vom Typ *TPoint* sind und den linken oberen bzw. den rechten unteren Eckpunkt des Rechtecks angeben. In beiden Fällen ist die Speichergröße des Records gleich, nämlich bestehend aus vier ganzzahligen Werten, die jedoch in den beiden Varianten verschieden strukturiert sind. Ein solcher varianter Record ist in DELPHI folgendermaßen bereits vordefiniert:

```
type TRect = record
       case Integer of
         0: (Left, Top, Rigth, Bottom: Integer);
         1: (TopLeft, BottomRight: TPoint);
       end;

var  x1,y1,x2,y2 : Integer;
     A, B : TPoint;
     Rect : TRect;

with Form1 do begin
  A.X := x1;  A.Y := y1;         // Punkterzeugung durch Koordinatenzuweisungen
  B.X := x2;  B.Y := y2;
  Rect.Left  := A.X; Rect.Top := A.Y;     // 1. Variante zur Rechteckserzeugung
  Rect.Right := B.X; Rect.Bottom := B.Y;
  Rect.TopLeft     := A;                  // 2. Variante zur Rechteckserzeugung
  Rect.BottomRight := B;
  Canvas.Brush.Color := clRed;   // Weist der Zeichenbürste eine Farbe zu
  Canvas.FrameRect(Rect);        // Zeichnet einen Rahmen in der Bürstenfarbe
  Canvas.FillRect(Rect);        // Füllt das Rechteck mit der Bürstenfarbe
end;
```

DELPHI erkennt aus der Programmcodierung automatisch, um welche Variante es sich handelt und kann diese fehlerlos weiterverarbeiten. Objekte vom Typ *TRect* sind in DELPHI sehr häufig. So sind die Positionsangaben einer Komponente und ihre Größe im Formular *TRect*-Strukturen. Die Methode *BoundsRect* liefert das, die Komponente begrenzende Rechteck zurück, beispielsweise *Rect := Image1.BoundsRect*. Der Befehl *Rect := Bounds(Left,Top,Width,Heigth)* erzeugt allgemein ein Rechteck mit den angegebenen Abmessungen.

[4.21] Die Objekte *Application, Screen* und *Printer*

Das Objekt ***Application*** wird von jedem DELPHI-Programm verwendet. Die Methode *CreateForm* erzeugt ein Formular. Die Methode *Run* startet die Programmausführung, *Terminate* beendet das Programm. Die Methode *ProcessMessage* gibt Rechenzeit für andere Anwendungen frei.

Das Objekt ***Screen*** stellt sämtliche wichtigen Grafikparameter des Bildschirmes zur Verfügung: *Width* (Breite des Bildschirmes in Pixeln), *Height* (Höhe des Bildschirmes in Pixeln), *Fonts* (unterstützte Schriftarten), *PixelsPerInch* (Anzahl der Bildpunkte pro Zoll). Alle diese Screen-Eigenschaften stehen nur zur Laufzeit zur Verfügung und können dann nur ausgelesen werden. Im folgenden Beispiel wird ein Formular exakt auf dem Bildschirm zentriert.

```
Form1.Left := (Screen.Width - Form1.Width) div 2;
Form1.Top := (Screen.Height - Form1.Height) div 2;
```

Wird die Unit ***Printers*** eingebunden, kann man Grafikausgaben auf *Printer.Canvas* mit den gleichen Methoden wie auf die Zeichenfläche anderer Objekte programmieren. Alle Grafikbefehle müssen dabei von *Printer.BeginDoc* und *Printer.EndDoc* eingeschlossen sein. Die Eigenschaften *Printer.PageWidth* und *Printer.PageHeight* geben die Abmessungen eines Druckerblattes an. Im folgenden Beispiel wird ein Quadrat in der linken oberen Papierecke gedruckt.

```
Printer.BeginDoc;
  Printer.Canvas.Rectangle(0,0,200,200);
Printer.EndDoc;
```

Zum Ausdrucken von Texten wird der Drucker ganz einfach zum Textfile erklärt: *AssignPrn(F)* mit *F: TextFile*. Sodann kann mittels *Writeln(F,S)* die Stringvariable S ausgedruckt werden. Es können aber auch Steueranweisungen gesendet werden, wie *Print.Canvas.Font.Size := 14*, was die Einstellung einer Schriftgröße bewirkt. Nicht zu vergessen ist, dass mit *CloseFile(F)* die Verbindung zum Drucker wieder geschlossen werden muss. Etwaige Druckereinstellungen können vorher mit einem Druckerdialog vorgenommen werden (*Print.Dialog* im Dialogregister der Komponentenpalette). Das ganze aktuelle Formular mitsamt seinen Komponenten kann mit der Anweisung *Form1.Print* ausgedruckt werden (mit *PrintScale := poProportional*).

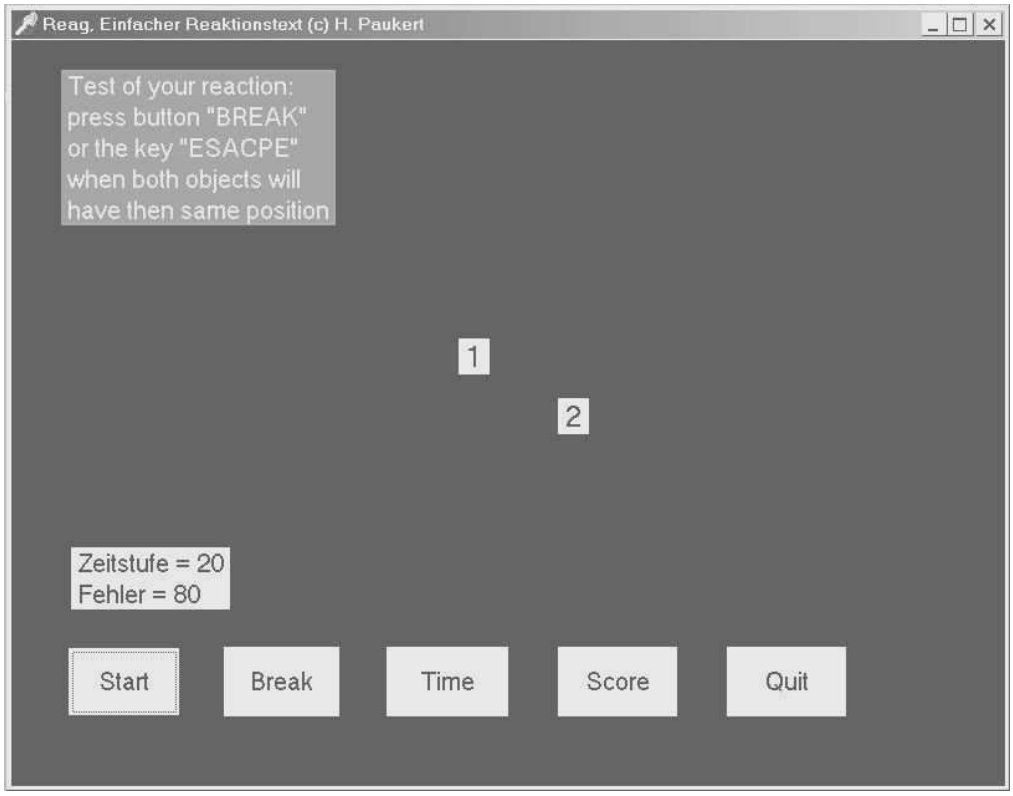
[4.22] *Warte*-Schleifen und *Escape*-Unterbrechungen

In diesem Abschnitt sollen zwei Aufgaben beschrieben und gelöst werden, welche als Teilprobleme in Programmprojekten häufig anzutreffen sind. **Erstens**, wie kann in einer Wiederholungsschleife eine Pause mit bestimmter Zeitdauer programmiert werden? **Zweitens**, wie kann eine Wiederholungsschleife durch Betätigung der <*Escape*>-Taste abgebrochen werden? Zur Lösung der gestellten Aufgaben werden drei systemdefinierte Routinen verwendet:

(1) Die Systemfunktion ***GetTickCount*** liefert jene Zeit in Millisekunden, welche seit Einschalten des Computers vergangen ist.
(2) Die Systemfunktion ***GetAsyncKeyState(vk_code)*** liefert einen Wert <> 0, wenn die Taste mit dem angegebenen virtuellen Tastaturcode *vk_code* gedrückt worden ist.
(3) Ein Aufruf der Methode ***Application.ProcessMessages*** ermöglicht es, genau jene Botschaften abzuarbeiten, welche sich aktuell in der Windows-Botschaften-Warteschlange befinden, z.B. eine Nachricht über ein Maus-Ereignis an einem Schaltknopf. Dadurch wird immer kurzfistig der laufende Prozess unterbrochen.

Das nachfolgende Programm "***reakt***" ist ein einfacher Reaktionstest, bei dem zwei Labelobjekte im Formular sich horizontal gegeneinander bewegen und bei Erreichen des Randes ihre Richtung wieder umkehren. Der Anwender soll nun entweder mit Hilfe der <*Escape*>-Taste oder mit einem Mausklick auf den Schaltknopf *Break* die Bewegung der Objekte dann unterbrechen, wenn sie sich möglichst an der gleichen horizontalen Position befinden.

Ihre gegenseitige Entfernung im Moment der Unterbrechung ist ein Maß für die Reaktionsfähigkeit des Anwenders. Zusätzlich ist es möglich, mit Hilfe der Routine *Pause* die Geschwindigkeit der Bewegung einzustellen.

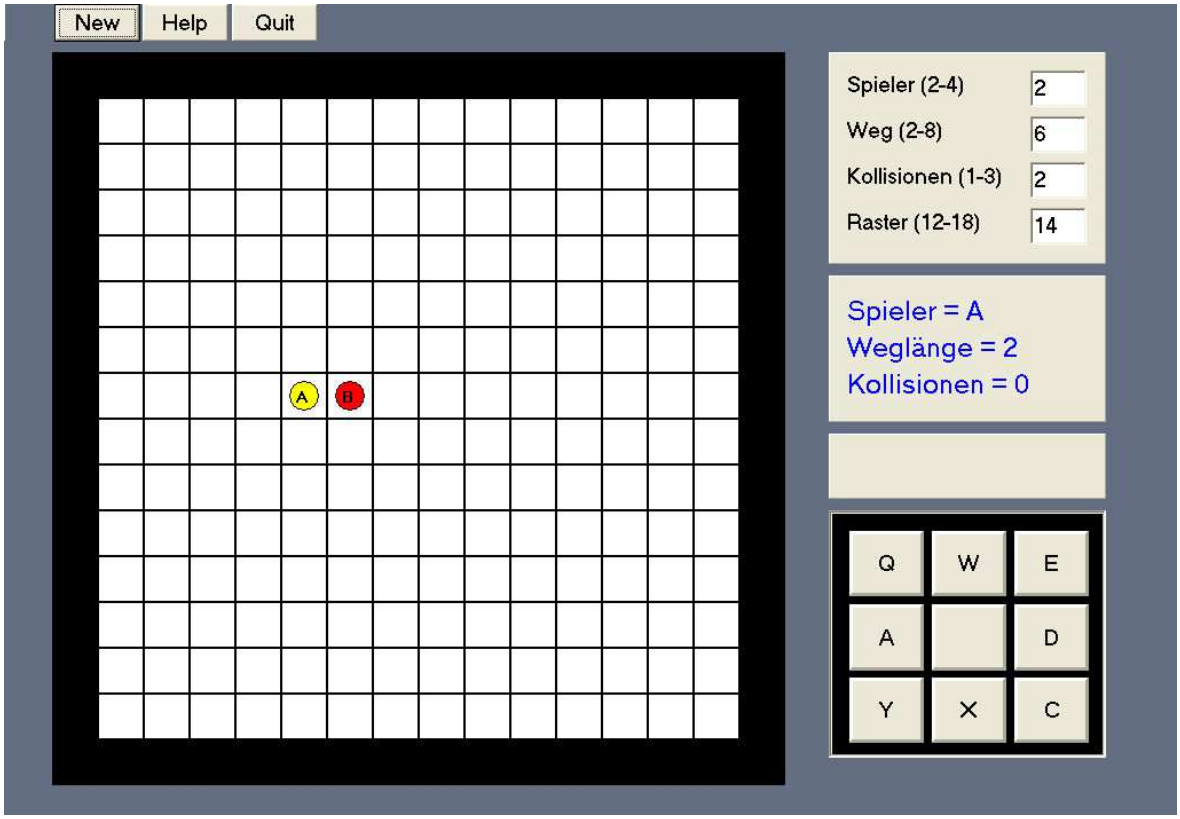



```
unit reakt_u;
// REAKT, Einfacher Reaktionstest(c) H. Paukert

interface

uses Windows, Messages, SysUtils, Classes, Graphics,
     Controls, Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Label1 : TLabel;
    Label2 : TLabel;
    Label3 : TLabel;
    Label4 : TLabel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    private { Private declarations }
    public  { Public declarations }
  end;

var Form1: TForm1;

implementation
{$R *.DFM}
```

```
var XA,XE  : Integer;      //  X-Grenzen
    X      : Integer;      //  X-Koordinate
    StepX  : Integer;      //  Schrittweite
    Zeit   : Integer;      //  Wartezeit (MSEC)
    Fehler : Integer;      //  Fehler
    Anzahl : Integer;      //  Versuchszähler
    Treffer: Integer;      //  Trefferzähler
    Summe  : Real;         //  gewichtete Fehlersumme
    Score  : Real;         //  gewichteter mittlerer Fehler
    Break_Flag : Boolean;  //  Break-Flag


procedure FitForm(F :TForm);
// Anpassung des Formulares an die Monitorauflösung
begin
  with F do begin
    if (Screen.Width<>1024) then ScaleBy(Screen.Width,1024);
    if (Font.PixelsPerInch<>120) then ScaleBy(120,Font.PixelsPerInch);
    WindowState := wsMaximized;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
// Initialisierungen
begin
   Position := poScreenCenter;
   Label4.Caption := ' Test of your reaction:  ' + #13 +
                     ' press button "BREAK"    ' + #13 +
                     ' or the key "ESACPE"     ' + #13 +
                     ' when both objects will  ' + #13 +
                     ' have then same position ';
   StepX := 10;
   Zeit  := 20;
   XA := 60; XE := Form1.ClientWidth - XA;
   Label1.Left := XA;
   Label2.Left := XE;
   Label3.Color:= clBlack;
   Break_Flag := False;
   Anzahl := 0; Treffer := 0; Summe := 0; Score := 0;
end;

procedure TrefferAusgabe;
// Ausgabe des Fehlers
var S0,S1 : String;
begin
  Fehler := abs(Form1.Label2.Left - Form1.Label1.Left);
  str(Zeit,S0); S0 := Trim(S0)+ ' ';
  str(Fehler,S1); S1 := Trim(S1) + ' ';
  Form1.Label3.Color   := clWhite;
  Form1.Label3.Caption := ' Zeitstufe = ' + S0 + #13 + ' Fehler = ' + S1;
  Anzahl := Anzahl + 1;
  if Fehler = 0 then Treffer := Treffer + 1;
  Summe := Summe + Fehler * Zeit / 20;
end;

procedure Pause(Zeit: Integer);
// Pause in MilliSekunden
var Zeit1: Integer;
begin
  zeit1 := GetTickCount;
  repeat
    Application.ProcessMessages;
  until (GetTickCount - Zeit1 > Zeit);
end;

procedure TForm1.Button1Click(Sender: TObject);
// X-Koordinaten der Labels verändern (horizontale Bewegung)
begin
  Memo1.Visible := False;
  Button2.SetFocus;
  Label3.Color := clBlack;
```

```
  Break_Flag := False;
  Fehler := 0;
  StepX  := 10;
  X := 0;
  Label1.Left := XA;
  Label2.Left := XE;
  repeat
    Label1.Left := XA + X;
    Label2.Left := XE - X;
    Pause(Zeit);
    if Break_Flag or (GetAsyncKeyState(VK_ESCAPE)<>0) then begin
       Break_Flag := True;
       Beep;
       TrefferAusgabe;
       Exit;
    end;
    X := X + StepX;
    if X >= (XE - XA) then StepX := - StepX;
    if X <= 0 then StepX := - StepX;
  until False;
end;

procedure TForm1.Button2Click(Sender: TObject);
// Break-Flag setzen
begin
  Break_Flag := True;
  Button1.SetFocus;
end;

procedure TForm1.Button3Click(Sender: TObject);
// Wartezeit eingeben
var S : String;
begin
  if Not Break_Flag then Exit;
  S := InputBox('Wartezeit in MSEC','','100');
  Zeit := StrToInt(S);
  Button1.SetFocus;
end;

procedure TForm1.Button4Click(Sender: TObject);
// Mittleren Score anzeigen
var S : String;
begin
  if Not Break_Flag then Exit;
  Score := Summe / Anzahl;
  Str(Score:12:2,S);
  S := Trim(S);
  with Memo1 do begin
     Clear;
     Visible := True;
     Lines.Add(' ');
     Lines.Add(' Anzahl  = ' + IntToStr(Anzahl));
     Lines.Add(' Treffer = ' + IntToStr(Treffer));
     Lines.Add(' ');
     Lines.Add(' tempo-gewichteter');
     Lines.Add(' mittlerer Fehler:');
     Lines.Add(' ');
     Lines.Add(' Score = ' + S);
     Lines.Add(' ');
  end;
  Anzahl := 0; Treffer := 0; Summe := 0; Score := 0;
end;

procedure TForm1.Button5Click(Sender: TObject);
// Programm beenden
begin
  if Not Break_Flag then Exit;
  Application.Terminate;
end;

end.
```

[4.23] Das einfache Strategiespiel „*Spuren*“



Spielplan:

Zuerst müssen die Anzahl der Spieler (2 bis 4), die maximale Weglänge (2 bis 8), die Höchstzahl an Kollisionen (1 bis 3), und die Spielfeldgröße (12 bis 24) eingegeben werden. Die Eingaben sind nur wirksam, wenn danach der Schalter <**New**> betätigt wird.

Der Zufall wählt für jeden Spieler eine Weglänge aus.

Die Bewegungsrichtung seines Spielsteines bestimmt jeder Spieler mit einem Mausklick auf die entsprechenden Schaltflächen. Dabei sind auch Diagonalschritte möglich. Die Bewegung kann auch mit Hilfe der Tasten Q, W, E, A, D, Y, X, C gesteuert werden.

Nach der Eingabe der Bewegungsrichtung wird automatisch die entsprechende Bewegung des jeweiligen Spielsteins ausgeführt.

Jede Kollision mit einer Wegspur oder dem Rand soll vermieden werden. Wird die Höchstanzahl der Kollision erreicht, so scheidet der Spieler aus.

Ziel des Spieles ist es, durch strategisch angelegte Wegspuren die Gegner einzukesseln, so dass sie bewegungsunfähig sind. Gewonnen hat der Spieler, der als Letzter noch seinen Spielstein bewegen kann. Dieser muss dann mittels herbeigeführter Kollisionen ebenfalls bewegungsunfähig gemacht werden. Sodann kann mit Hilfe von Schalter <New> ein neues Spiel begonnen werden.

Der Schalter <**Help**> zeigt den Hilfetext an. Der Schalter <**Quit**> beendet das Programm.

```
unit spuren_u;
// SPUREN, ein grafisches Strategiespiel (c) Herbert Paukert

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TForm13 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Memo1: TMemo;
    Image1: TImage;
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Panel4: TPanel;
    Panel5: TPanel;
    Panel6: TPanel;
    Panel7: TPanel;
    Panel8: TPanel;
    Panel9: TPanel;
    Panel10: TPanel;
    Panel11: TPanel;
    Panel12: TPanel;
    Panel13: TPanel;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Panel5Click(Sender: TObject);
    procedure Panel10Click(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
                          Shift: TShiftState);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
  private { Private-Deklarationen }
  public  { Public-Deklarationen }
end;

var  Form13: TForm13;

implementation
{$R *.DFM}

var   a: Integer = 14;
      b: Integer = 14;
      Farbe: Array[1..4] of TColor = (clYellow,clRed,clBlue,clGreen);

type  TMatrix = Array[1..26,1..26] of Integer;
      Feld = Array[1..4] of Integer;
      Position = Array[1..4] of TPoint;
```

```
var   Matrix: TMatrix;
      XMax,YMax: Integer;
      Rand : Integer;
      l,e,r: Integer;
      WegMax,Kollmax : Integer;
      Weg,Richt,Aus: Integer;
      Anz,N: Integer;
      Posi : Position;
      Koll : Feld;

procedure FitForm(F :TForm);
// Anpassung des Formulares an die Monitorauflösung
const SW: Integer = 1024;
      SH: Integer = 768;
      FS: Integer = 96;
      FL: Integer = 120;
      AR: Real    = 0.625;
var   X,Y,K: Integer;
      Z: Real;
begin
  with F do begin
    X := Screen.Width;
    Y := Screen.Height;
    K := Font.PixelsPerInch;
    Scaled := True;
    Z := Y/X;
    if (Z > AR)  then ScaleBy(X,SW) else ScaleBy(Y,SH);
    if (K <> FL) then ScaleBy(FL,K);
    WindowState := wsMaximized;
  end;
end;

procedure Wait(Zeit: Integer);
//  Unterbricht das Programm um 'Zeit' MilliSekunden
var Zeit0 : Integer;
begin
  Zeit0 := GetTickCount;             // Internen Zeitzähler aufrufen
  repeat
     Application.ProcessMessages;     // ermöglicht Systembotschaften
  until (GetTickCount - Zeit0) > Zeit;
end;

procedure InitFrame;
// Initialisiert das Spielfeld
var S: String;
    Z,Code: Integer;
begin
  S := Form13.Edit5.Text;
  Val(S,Z,Code);
  if (Code<>0) or (Z<12) or (Z>18) then begin
     Form13.Edit5.Text := '14';
     Z := 14;
  end;
  a := Z + 2;
  b := Z + 2;
end;

procedure InitForm;
// Initialisiert das Spielfeld
begin
  with Form13 do begin
    e := YMax div a;
    XMax := a * e;
    YMax := b * e;
    r := e div 3;
    Rand := Image1.Left + XMax + 40;
```

```
   Image1.Left   := Screen.Width div 24;
   Image1.Top    := Screen.Width div 24;
   Image1.Width  := XMax;
   Image1.Height := YMax;
   Memo1.Left    := Image1.Left;
   Memo1.Top     := Image1.Top;
   Memo1.Width   := Image1.Width;
   Panel11.Left  := Rand ;
   Panel11.Top   := Image1.Top;
   Panel12.Left  := Rand ;
   Panel12.Top   := Image1.Top + Panel11.Height + 10;
   Panel13.Left  := Rand ;
   Panel13.Top   := Panel12.Top + Panel12.Height + 10;
   Panel10.Left  := Rand;
   Panel10.Top   := Panel13.Top + Panel13.Height + 10;
  end;
end;

procedure InitMatrix;
// Initialisiert die Matrix
var i,k : integer;
begin
  For i := 1 to a do
    For k := 1 to b do Matrix[i,k] := 0;      //  0 = freier Weg
  For i := 1 to a do begin
    Matrix[i,1] := 1; Matrix[i,b] := 1;       //  1 = Randhindernisse
  end;
  For k := 1 to b do begin
    Matrix[1,k] := 1; Matrix[a,k] := 1;
  end;
end;

procedure InitPicture;
// Initialisiert das Bild
var i,k: Integer;
   P,Q: TPoint;
   Col: TColor;
begin
  With Form13.Image1.Canvas do begin
   Font.Color := clBlack;
   Font.Size  := 10;
   Font.Style := [fsbold];
   Pen.Width := 1;
   Pen.Color := clBlack;
   Brush.Color := clWhite;
   Brush.Style := bsSolid;
   Rectangle(0,0,XMax,YMax);
   Brush.Style := bsClear;
   For i := 1 to a do begin
      MoveTo(i*e,0); LineTo(i*e,YMax);
   end;
   For i := 1 to b do begin
      MoveTo(0,i*e); LineTo(XMax,i*e);
   end;
   For i := 1 to a do
     For k := 1 to b do begin
       P.X := (i-1)*e; P.Y := (k-1)*e;
       Q.X := i*e; Q.Y := k*e;
       if (Matrix[i,k] = 0) then Col := clWhite;
       if (Matrix[i,k] > 0) then Col := clBlack;
       Brush.Color := Col;
       Brush.Style := bsSolid;
       Rectangle(P.X,P.Y,Q.X,Q.Y);
       Brush.Style := bsClear;
     end;
  end;
end;
```

```
procedure MarkField(i,k,N: Integer);
// Markiert das Matrixfeld (i,k) des Spielers N
var M,P: TPoint;
begin
  M.X := (i-1)*e + e div 2; M.Y := (k-1)*e + e div 2;
  With Form13.Image1.Canvas do begin
    Brush.Color := Farbe[N];
    Brush.Style := bsSolid;
    Ellipse(M.X-r,M.Y-r,M.X+r,M.Y+r);
    Brush.Style := bsClear;
    P.X := (i-1)*e + e div 3;
    P.Y := (k-1)*e + e div 3;
    TextOut(P.X,P.Y,Chr(64+N));
  end;
end;

procedure PaintField(i,k: Integer; Col: TColor);
// Markiert das Matrixfeld (i,k)
var P,Q: TPoint;
begin
  P.X := (i-1)*e; P.Y := (k-1)*e;
  Q.X := i*e; Q.Y := k*e;
  With Form13.Image1.Canvas do begin
    Brush.Color := Col;
    Brush.Style := bsSolid;
    Rectangle(P.X,P.Y,Q.X,Q.Y);
    Brush.Style := bsClear;
  end;
end;

procedure ShowData(N: Integer);
// Anzeige der Werte des N-ten Spielers
begin
  Weg := Random(WegMax) + 1;
  Form13.Label4.Caption := 'Spieler = ' + Chr(64+N);
  Form13.Label5.Caption := 'Weglänge = ' + IntToStr(Weg);
  Form13.Label6.Caption := 'Kollisionen = ' + IntToStr(Koll[N]);
end;

procedure InitPlay;
// Weitere Initialisierungen
var I,X,Y,Z,Code : Integer;
    S: String;
begin
  S := Form13.Edit1.Text;
  Val(S,Z,Code);
  if (Code<>0) or (Z<2) or (Z>4) then begin
     Form13.Edit1.Text := '2';
     Z := 2;
  end;
  Anz := Z;
  S := Form13.Edit2.Text;
  Val(S,Z,Code);
  if (Code<>0) or (Z<2) or (Z>8) then begin
     Form13.Edit2.Text := '8';
     Z := 8;
  end;
  WegMax := Z;
  S := Form13.Edit3.Text;
  Val(S,Z,Code);
  if (Code<>0) or (Z<1) or (Z>3) then begin
     Form13.Edit3.Text := '3';
     Z := 3;
  end;
  KollMax := Z;
  Form13.Label7.Caption := 'Aus für ';
  Form13.Label7.Visible := False;
```

```
For I := 1 to Anz do begin
   Koll[I] := 0;
   Posi[I].X := a div 3 + I; Posi[I].Y := b div 2;
   X  := Posi[I].X; Y  := Posi[I].Y;
   MarkField(X,Y,I);
   Matrix[X,Y] := 2;
  end;
  N := 1;
  ShowData(N);
  Aus := 0;
  Richt := 40;
  Form13.Panel1.Color := clBtnFace;
  Form13.Panel2.Color := clBtnFace;
  Form13.Panel3.Color := clBtnFace;
  Form13.Panel4.Color := clBtnFace;
  Form13.Panel6.Color := clBtnFace;
  Form13.Panel7.Color := clBtnFace;
  Form13.Panel8.Color := clBtnFace;
  Form13.Panel9.Color := clBtnFace;
  Form13.Button2.SetFocus;
end;

procedure TForm13.FormCreate(Sender: TObject);
// Erste Initialisierungen des Formulars
begin
  Color := RGB(100,110,130);
  FitForm(Form13);
end;

procedure TForm13.FormActivate(Sender: TObject);
// Zweite Initialisierungen des Formulars
begin
  Randomize;
  Memo1.Visible := False;
  XMax := Round(5*Screen.Width/6);
  YMax := Round(5*Screen.Height/6);
  InitFrame;
  InitForm;
  InitMatrix;
  InitPicture;
  InitPlay;
end;

procedure Move(N: Integer);
// Spielstein bewegen
var  X,Y,
     X0,Y0,
     SX,SY,Z : Integer;
begin
  if Koll[N] >= KollMax then Exit;
  if (Richt = 35) or (Richt = 97)  then begin SX := -1; SY :=  1; end;
  if (Richt = 40) or (Richt = 98)  then begin SX :=  0; SY :=  1; end;
  if (Richt = 34) or (Richt = 99)  then begin SX :=  1; SY :=  1; end;
  if (Richt = 37) or (Richt = 100) then begin SX := -1; SY :=  0; end;
  if (Richt = 39) or (Richt = 102) then begin SX :=  1; SY :=  0; end;
  if (Richt = 36) or (Richt = 103) then begin SX := -1; SY := -1; end;
  if (Richt = 38) or (Richt = 104) then begin SX :=  0; SY := -1; end;
  if (Richt = 33) or (Richt = 105) then begin SX :=  1; SY := -1; end;
  X := Posi[N].X;
  Y := Posi[N].Y;
  Z := 0;
  repeat
    X0 := X;
    Y0 := Y;
    X  := X + SX;
    Y  := Y + SY;
```

```
    if (Matrix[X,Y] > 0) then begin
      Koll[N] := Koll[N] + 1;
      ShowMessage('Achtung - Kollision');
      Form13.Label6.Caption := 'Kollisionen = ' + IntToStr(Koll[N]);
      X := X0;
      Y := Y0;
      if Koll[N] >= KollMax then begin
        ShowMessage('Spieler ' + Chr(64+N) + ' ausgeschieden');
        Form13.Label7.Visible := True;
        Form13.Label7.Caption := Form13.Label7.Caption + Chr(64+N) + ',';
        Aus := Aus + 1;
        if Aus = Anz then begin
          Form13.Label4.Caption := '';
          Form13.Label6.Caption := '';
          Form13.Label5.Caption := 'Sieger ist Spieler ' +  Chr(64 + N);
          Exit;
        end;
      end;
      Break;
    end;
    if (Matrix[X,Y] = 0) then begin
      PaintField(X0,Y0,clGray);
      Wait(200);
      MarkField(X,Y,N);
      Matrix[X,Y] := 2;
    end;
    Z := Z + 1;
  until (Z = Weg);
  Posi[N].X := X;
  Posi[N].Y := Y;
end;

procedure Play(var N: Integer);
// Spiel mit Spieler N
begin
  Move(N);
  if Aus = Anz then Exit;
  N := N + 1;
  if N > Anz then N := 1;
  While (Koll[N] >= KollMax) do begin
    N := N + 1;
    if N > Anz then N := 1;
  end;
  ShowData(N);
end;

procedure TForm13.Panel5Click(Sender: TObject);
// Go on
begin
  Exit;
  Play(N);
  if Aus = Anz then Exit;
end;

procedure TForm13.Panel10Click(Sender: TObject);
begin
  Panel1.Color := clBtnFace;
  Panel2.Color := clBtnFace;
  Panel3.Color := clBtnFace;
  Panel4.Color := clBtnFace;
  Panel6.Color := clBtnFace;
  Panel7.Color := clBtnFace;
  Panel8.Color := clBtnFace;
  Panel9.Color := clBtnFace;
  if Sender = Panel1 then begin Panel1.Color := clRed; Richt := 36; end;
  if Sender = Panel2 then begin Panel2.Color := clRed; Richt := 38; end;
  if Sender = Panel3 then begin Panel3.Color := clRed; Richt := 33; end;
```

```
  if Sender = Panel4 then begin Panel4.Color := clRed; Richt := 37; end;
  if Sender = Panel6 then begin Panel6.Color := clRed; Richt := 39; end;
  if Sender = Panel7 then begin Panel7.Color := clRed; Richt := 35; end;
  if Sender = Panel8 then begin Panel8.Color := clRed; Richt := 40; end;
  if Sender = Panel9 then begin Panel9.Color := clRed; Richt := 34; end;
  Play(N);
  if Aus = Anz then Exit;
end;

procedure TForm13.FormKeyDown(Sender: TObject; var Key: Word;
                              Shift: TShiftState);
begin
  Panel1.Color := clBtnFace;
  Panel2.Color := clBtnFace;
  Panel3.Color := clBtnFace;
  Panel4.Color := clBtnFace;
  Panel6.Color := clBtnFace;
  Panel7.Color := clBtnFace;
  Panel8.Color := clBtnFace;
  Panel9.Color := clBtnFace;
  if UpCase(chr(Key)) = 'Q' then begin Panel1.Color := clRed; Richt := 36; Play(N); end;
  if UpCase(chr(Key)) = 'W' then begin Panel2.Color := clRed; Richt := 38; Play(N); end;
  if UpCase(chr(Key)) = 'E' then begin Panel3.Color := clRed; Richt := 33; Play(N); end;
  if UpCase(chr(Key)) = 'A' then begin Panel4.Color := clRed; Richt := 37; Play(N); end;
  if UpCase(chr(Key)) = 'D' then begin Panel6.Color := clRed; Richt := 39; Play(N); end;
  if UpCase(chr(Key)) = 'Y' then begin Panel7.Color := clRed; Richt := 35; Play(N); end;
  if UpCase(chr(Key)) = 'X' then begin Panel8.Color := clRed; Richt := 40; Play(N); end;
  if UpCase(chr(Key)) = 'C' then begin Panel9.Color := clRed; Richt := 34; Play(N); end;
  if Aus = Anz then Exit;
end;

procedure TForm13.Button2Click(Sender: TObject);
// Neues Spiel
begin
  if Memo1.Visible then Exit;
  InitFrame;
  InitForm;
  InitMatrix;
  InitPicture;
  InitPlay;
end;

procedure TForm13.Button3Click(Sender: TObject);
// Hilfetext ein- und ausblenden
begin
  Memo1.Visible := NOT Memo1.Visible;
end;

procedure TForm13.Button4Click(Sender: TObject);
// Programm beenden
begin
  Form13.Close;
end;

end.
```