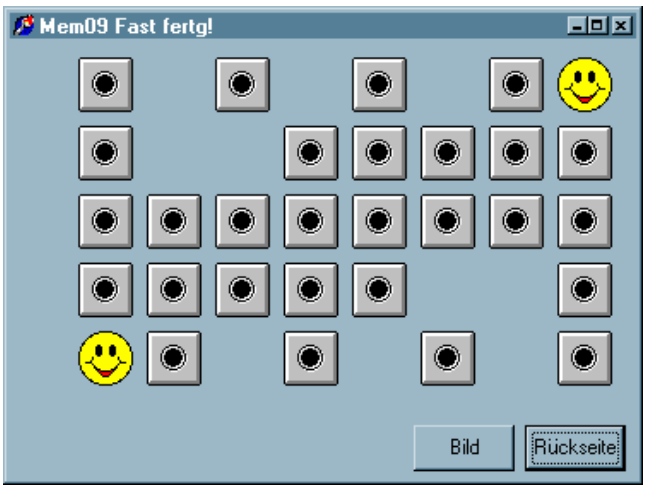
Zur Laufzeit verwaltete Objekte

Am Beispiel des bekannten Memory-Spieles

Warum Spiele? fragen Sie.
Ich antworte: um die Kunst der Erfindung zu vervollkommnen. (Leibniz)

Mit seinen einfachen und unzweideutigen Regeln ist ein Spiel immer so etwas wie die Insel der Ordnung im unübersichtlichen Chaos unseres Erlebens.(Huxley)

So verwundert nicht, dass die wundervollsten Spielzeuge unseres Jahrhunderts, die Computer, von Anfang an mit Spielen zu tun hatten.
(Mc Corduck)



Die Zitate sind aus dem Arbeitsheft „Strategiespiele“ von R. Baumann entnommen. Baumann gibt eine Reihe von sehr guten didaktischen Begründungen für die Modellierung von Spielen im Informatikunterricht.

Gerade mit den modernen Programmierwerkzeugen wie Visual Basic und Delphi lassen sich sehr schnell mit wenig Aufwand Benutzeroberflächen für Spiele realisieren, die der Erwartungshaltung der Schüler gerecht werden. Insbesondere am Beispiel von Strategiespielen können grundlegende Algorithmen der Informatik erarbeitet werden.

Wegen der Einfachheit eignet sich das bekannte Memory-Spiel m. E. hervorragend zur Einführung in die objektorientierte und ereignisgesteuerte Programmierung. Da Memory kein Strategiespiel ist, sind keine größeren algorithmischen Probleme vorhanden. Man kann sich voll auf das Wesentliche konzentrieren. Durch Verwendung von Standardikonen werden keine besonderen Grafikkenntnisse erforderlich.

Im Folgenden soll, ausgehend vom einfachsten Fall einer Spielkarte als TImag-Komponente, sukzessive über zur Laufzeit generierten Karten-Komponenten bis zu einer gewissen Stufe ein Memory-Spiel mit Delphi implementiert werden.

Zur Laufzeit verwaltete Objekte

1. Der einfachste Fall (Mem1)

Eine Spielkarte wird durch eine TImage-Komponente abgebildet. Über die Ereignisprozeduren Onklick der Buttons „Bildseite“ und „Rückseite“ wird über die Methode „LoadFromFile“ das jeweilige Bild geladen und angezeigt.


```
unit UMem01;

interface

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

type
  TForm1 = class(TForm)
    BBildseite:  TButton;
    BRueckseite: TButton;
    Karte: TImage;
    procedure BRueckseiteClick(Sender: TObject);
    procedure BBildClick(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.BRueckseiteClick(Sender: TObject);
begin
  Karte.Picture.LoadFromFile('../ico/icon0.ico');
end;

procedure TForm1.BBildseiteClick(Sender: TObject);
begin
  Karte.Picture.LoadFromFile('../ico/icon1.ico');
end;


end.
```

2. Die TImage-Komponente wird zur Laufzeit erzeugt (Mem2)
- Instanzierung

Die TImage-Komponente soll nun zur Laufzeit mit dem Fenster Form1 erzeugt werden. Wir vereinbaren dazu eine Variable Karte vom Typ TKarte innerhalb des „private“-Bereiches von Form1. Form1 hat damit einen Zeiger auf ein Unterobjekt vom Typ TKarte.

Mit der Konstruktormethode Create von TImage wird ein Exemplar von TKarte erzeugt (instanziert). Diese Konstruktormethode rufen wir mit der Ereignismethode FormCreate von Form1 auf. Der Parameter der Konstruktormethode ist ein Zeiger auf den Eigentümer (Owner), also Form1 oder allgemeiner self. Über die Objektvariable Karte können nun die Attribute des so erzeugten Objekts verändert werden.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Karte:=TImage.Create(Self); {Constructor}
  Karte.Parent:= Self; {das Objekt gehört zu Form1}
  Karte.Left:=30;
  Karte.Top:= 90;
  Karte.AutoSize:=True;
  Karte.Picture.LoadFromFile('../ico/icon0.ico');
end;
```

Parent sollte immer die erste Eigenschaft einer neuen Komponente sein, der Sie einen Wert zuweisen.
Diese Eigenschaft ist nur für TwinControls definiert und gibt an, zu welcher Komponente die Komponente gehört. Normalerweise ist dies das Formular also Form1 oder allgemeiner self. Parent und Owner können auch verschieden sein.

```
procedure TForm1.BRueckseiteClick(Sender: TObject);
begin
  Karte.Picture.LoadFromFile('../ico/icon0.ico');
end;

procedure TForm1.BBildseiteClick(Sender: TObject);
begin
  Karte.Picture.LoadFromFile('../ico/icon1.ico');
end;
```

3. Die TKarte als Nachkomme von TImage (Mem3)
- Vererbung

Eine konsequente objektorientierte Modellierung macht die Definition eines Objektes (Komponente) TKarte als Nachkomme von TImage erforderlich. Sie soll zunächst die beiden Methoden ZeigeBildseite und ZeigeRückseite enthalten. Damit nicht bei jedem Wechsel das anzuzeigende Bild aus der Datei gelesen wird, enthält TKarte die beiden TPicture-Objekte Bildseite und Rückseite.

Schon beim Instanzieren sollen die Attribute von TKarte auf die Defaultwerte gesetzt werden. Dazu muss man die von TImage geerbte Konstruktormethode ergänzen (überschreiben).

```
TKarte = class(TImage)
  Constructor Create(Owner:TWinControl; x,y: Integer; ICO: String);
  public
    procedure ZeigeBildseite;
    procedure ZeigeRueckseite;
  private
    Bildseite:  TPicture;
    Rueckseite: TPicture;
end;


TForm1 = class(TForm)
    ......
  private
    Karte1, Karte2, Karte3: TKarte;

end;
```

Der neue Konstruktor muss die von TImage geerbte Konstruktormethode mit inherited create(owner) aufrufen, da man ja nur die hinzugefügten Attribute setzen möchte.

```
constructor TKarte.Create(Owner:TWinControl; x,y:integer; ico:String);
begin
  inherited Create(Owner);
  Parent:= Owner;
  Top:=y; Left:=x;
  AutoSize:= True;
  Rueckseite:=TPicture.Create;  {ohne Constructor RunTime-Error}
  Rueckseite.LoadFromFile(IcoPath+'Icon0.ico');
  Bildseite:=TPicture.Create;
  Bildseite.LoadFromFile(IcoPath+ico+'.ico');
  Picture:=Bildseite;
end;

procedure TKarte.ZeigeBildseite;
begin
  Picture:=Bildseite;
end;

procedure TKarte.ZeigeRueckseite;
begin
  Picture:=Rueckseite;
end;
```

Mit dem Konstruktor von TKarte können jetzt sehr einfach mehrere Instanzen (Exemplare) von TKarte erzeugt werden, die über Karte1, Karte2 usw. adressiert werden.

```
procedure TForm1.BRueckseiteClick(Sender: TObject);
begin
  Karte1.ZeigeRueckseite;
  Karte2.ZeigeRueckseite;
end;

procedure TForm1.BBildseiteClick(Sender: TObject);
begin
  Karte1.ZeigeBildseite;
  Karte2.ZeigeBildseite;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Karte1:=TKarte.Create(Self, 40,40,'Icon1');
  Karte2:=TKarte.Create(Self, 80,40,'Icon2');
  Karte3:=TKarte.Create(Self,120,40,'Icon3');
end;
```

4. Alle Karten sollen auf ein Button-Ereignis reagieren (Mem4)

- ein Ereignis wird an alle Karten gesendet

Zur bequemen Adressierung der einzelnen Karten verwenden wir jetzt statt drei Objektvariabler ein Array.

```
TForm1 = class(TForm)
    BBildseite:  TButton;
    BRueckseite: TButton;
    procedure BRueckseiteClick (Sender: TObject);
    procedure BBildseiteClick  (Sender: TObject);
    procedure FormCreate       (Sender: TObject);
  private
    Kartenzahl: Integer;
    KartenFeld: Array[1..2*IcoZahl] of TKarte;
    { Private-Deklarationen }
  public
...


procedure TForm1.FormCreate(Sender: TObject);
begin
  KartenFeld[1]:=TKarte.Create(Self, 40,40, 'Icon1');
  KartenFeld[2]:=TKarte.Create(Self, 80,40, 'Icon2');
  KartenFeld[3]:=TKarte.Create(Self,120,40, 'Icon3');
  KartenFeld[4]:=TKarte.Create(Self,160,40, 'Icon4');
  Kartenzahl:=4;
end;


Mit einer einfachen FOR-Schleife kann man jetzt beispielsweise die Botschaft
,Zeige Rückseite' an alle Karten senden.



procedure TForm1.BRueckseiteClick(Sender: TObject);
var i: integer;
begin
  for i:=1 to Kartenzahl do
    Kartenfeld[i].ZeigeRueckseite;
end;
```

5. Eine Karte soll sich beim Anklicken umdrehen (Mem5)

Zunächst implementieren wir die Methode Umdrehen, die man mit dem Button Umdrehen testen kann. Dazu benötigt man ein Zustandsattribut. Der Einfachheit halber wird das Kartenobjekt um die Eigenschaft IsBild erweitert. Diese Eigenschaft muss vom Konstruktor initialisiert und von den Methoden ZeigeBild und ZeigeRückseite aktualisiert werden.

```
type
  TKarte = class(TImage)
  Constructor Create(Owner:TComponent; x,y: Integer; ICO: String);
  public
    procedure Umdrehen(Sender: TObject);
    procedure ZeigeBildseite;
    procedure ZeigeRueckseite;
  private
    isbild:     Boolean;
    Bildseite:  TPicture;
    Rueckseite: TPicture;
  end;
{ Public-Deklarationen }
  end;
```


```
procedure TKarte.Umdrehen;
begin
  if IsBild then
     ZeigeRueckseite
  else
     ZeigeBildseite;
end;
```

Die geerbte Methode TKarte.Click soll die Methode Umdrehen aufrufen. Damit dies auch tatsächlich geschieht, muss im Konstruktor der geerbte Methodenzeiger OnClick von TKarte auf die Ereignisprozedur TKarte.Umdrehen gesetzt werden. Wegen der Typenkompatibilität muss die Methode Umdrehen den Parameter Sender vom TObjekt enthalten.

```
constructor TKarte.Create(Owner:TComponent; x,y:integer; Ico:String);
begin
  inherited Create(Owner);
  Parent:= Form1;
  Top:=y; Left:=x;
  AutoSize:= True;
  Rueckseite:=TPicture.Create;
  Rueckseite.LoadFromFile(IcoPath+'Icon0.ico');
  Bild:=TPicture.Create;
  Bild.LoadFromFile(IcoPath+Ico);
  Picture:=Bild;
  IsBild:= True;
  OnClick:= Umdrehen;
end;
```

6. Jetzt erzeugen wir viele Karten (Mem6)

Die nachstehende Ereignisprozedur ist selbst-erklärend.


```
procedure TForm1.FormCreate(Sender: TObject);
var Zeile, Spalte, IconNr, KartenNr: integer;
    IconName: string;
begin
  IconNr:=0; KartenNr:=0;
  for Zeile:=1 to 4 do
     for Spalte:=1 to 8 do begin
       if IconNr >= IconZahl  then IconNr:=0;
       inc(IconNr);
       IconName:='Icon'+IntToStr(IconNr);
       inc(KartenNr);
       KartenFeld[KartenNr]:=
                  TKarte.Create(Self,Spalte*40-20,Zeile*40-20,IconName)
     end;
  Kartenzahl:= KartenNr;
end;
```

7. Jetzt mit Destruktor zum Abräumen (Mem7)

Um eine Karte zu entfernen, rufen wir den geerbten Destruktor free auf. Free überprüft im Gegensatz zu destroy die Zulässigkeit der Aktion. Der Destruktor entfernt die Komponente aus der Form und gibt den Speicherplatz frei.

Mit einem Klick auf „Abräumen“ werden alle aufgedeckten Karten entfernt.


```
procedure TForm1.BAbraeumenClick(Sender: TObject);
var i: integer;
begin
  for i:=1 to Kartenzahl do
    if (Kartenfeld[i]<>NIL) AND Kartenfeld[i].IsBild then begin
       Kartenfeld[i].Destroy;
       Kartenfeld[i]:= NIL; end;
end;
```

Es ist leider nicht so, dass Delphi auch den Speicherbereich der in einem Objekt enthaltenen Objekte automatisch freigibt. TKarte enthält zwei Pictureobjekte, deren Destruktoren explizit aufgerufen werden müssen.

Da beim Abräumen der Aufruf mehrerer Destruktormethoden erforderlich ist, erfordert ein guter Programmierstil für TKarte eine eigene Destruktormethode.

```
TKarte = class(TImage)
  constructor Create(Owner:TWinControl; x,y: Integer; ICO: String);
  destructor Destroy; override;
.......

destructor TKarte.Destroy;
begin
  Rueckseite.free;
  Bild.free;
  inherited free;
end;
```

Zum Üben und Weiterarbeiten

Aufgabe 1:
- mem08: Abräumen unter Memory-Bedingungen

Speichern Sie mem07 in einem neuen Ordner mem08. Ändern Sie den Unit –und Projektnamen entsprechend.

Nur wenn zwei Karten aufgedeckt sind sollen diese bei:
- gleichen Bild abgeräumt
- ansonsten umdreht werden

Implementieren Sie dazu in TForm1.BAbraeumenClick die lokalen Funktionen
```
function zweiaufgedeckt: boolean;
function gleicheBilder:  boolean;
```

Um feststellen zu können ob zwei Karten das gleiche Bild haben, muss TKarte noch das Attribut „BildName“ vom Typ String enthalten. Mit der Konstruktormethode von TKarte wird dann mit `Bildname:= ico` der Dateiname gespeichert.

Aufgabe 2:
- mem09: Fast fertig….

Mit einem Click auf eine Karte soll jetzt die in Aufgabe 1 entwickelte Methode ausgeführt werden. Verfahren Sie wie in Memory05.

Aufgabe 3:
- mem10: Mischen

Implementieren Sie ein Button-Ereignis zum Mischen der Spielkarten. Sicher gibt es hier mehrere Möglichkeiten, z.B. durch Austauschen der Bilder oder durch Austauschen der Koordinaten, oder durch Laden in einer anderen Reihenfolge.

Aufgabe 4:
Im Hinblick auf die Verwendung von Destruktoren ist sicher im Unterricht eine Betrachtung der Speicherverwaltung erforderlich. Mit GetHeapStatus kann verdeutlicht werden, dass jede neue Instanz einer Karte auf dem Heap einen gewissen Speicherplatz beansprucht, der beim Aufruf von Destruktormethoden wieder frei werden muss. Wie viele Byte jede Instanz tatsächlich beansprucht, lässt sich nur schwer nachvollziehen. Die Speicherverwaltung unter Delphi ist komplizierter als unter Turbo Pascal. Es kommt aber nur darauf an, kontrollieren zu können, ob die Destruktormethoden die die Speicherbereiche wieder vollständig freigeben.

```
procedure TForm1.ShowHeap;
var H: THeapStatus;
    s: string;
begin
    H:= GetHeapStatus;
    s:= 'Heap in Byte: ';
    Statusbar.SimpleText:= S + IntToStr(H.TotalAllocated);
end;
```

Aktualisieren Sie die ShowHeap nach jedem Konstrutor bzw. Destruktoraufruf die Statuszeile.

Aufgabe 5:
Das Array Kartenfeld wird eigentlich nicht benötigt. Delphi verwaltet alle Komponenten einer Komponente in einer Liste. Mit dem Konstruktor und dem Destruktor einer Komponente wird die Komponenten-Liste des Eigentümers aktualisiert. Deswegen muss dem Konstruktor als Parameter der Eigentümer übergeben werden. Über die Eigenschaft ComponentCount erhält man die Anzahl der Komponenten einer Komponente. Eine Komponente kann mit Components[i] adressiert werden. Nach einer vorangegangenen Typ-Prüfung mit dem „is“-Operator, kann mit „as“ eine Eigenschaft oder Methode einer Komponente angesprochen werden.

```
procedure TForm1.BBildseiteClick(Sender: TObject);
var i: integer;
begin
  for i:=0 to Componentcount-1 do
    if Components[i] is TKarte then  (Components[i] as TKarte).ZeigeBildseite;
end;
```

Damit wird das Array Kartenfeld überflüssig! Die Karten werden einfach wie folgt instanziert:

```
procedure TForm1.FormCreate(Sender: TObject);
var Zeile, Spalte, IconNr, KartenNr: integer;
    IconName: string;
begin
  IconNr:=0; KartenNr:=0;
  for Zeile:=1 to 4 do
     for Spalte:=1 to 8 do begin
       if IconNr >= IconZahl  then IconNr:=0;
       inc(IconNr);
       IconName:='Icon'+IntToStr(IconNr);
       inc(KartenNr);
       TKarte.Create(Self,Spalte*40-20,Zeile*40-20,IconName)
     end;
  Kartenzahl:= KartenNr;
end;
```

Zum Weiterarbeiten:

- Das Objekt TKarte mit den dazu gehörenden Methoden soll in eine Unit ausgelagert werden.
- Implementierung einer Komponente Spielfeld als Nachkomme von Form1.
- Implementierung eines Punktekontos
- Implementierung einer Zeitsteuerung
- Implementierung von Sound
- Spielen im Netz

Literatur

Richard Kaiser, Object Pascal mit Delphi, Springer 1997
Elmar Warken, Delphi 3, Entw. Leistungsf. Anwendung. u. Komponenten, Addison Wesley, 1997
Rüdiger Baumann, Arbeitshefte Informatik, Strategiespiele, Klett 1994
Rüdiger Baumann, Didaktik der Informatik, Klett 1996
M. Zentgraf, Einf. i.d. Programmierung in Visual Basic, Hausarbeit im VI. LWB-Kurs Informatik