AUFGABE 3: WIE STEHT'S UM DEN IQ

Lösungsidee:

Meine Basisobjekte sollen aus Linien und Kreisen bestehen. Bei Linien wir der Anfangs und Endpunkt gespeichert, bei Kreisen Mittelpunkt und Radis. Ein Objekt kann maximal 15 Linien oder Kreise besitzen. Dazu kommen noch die Parameter, welche Symmetrien das Objekt besitzt. Gespeichert habe ich hierbei die Symmetrie an der Horizontalen und Vertikalen, sowie nach wie viel Grad das Objekt wieder deckungsgleich erscheint.

Mein Programm kennt die 3 Transformationen drehen, spiegeln und hinzufügen/entfernen. Im Programm soll das erstellen der IQ Tests so ablaufen: Der Benutzer erstellt Bild A und C indem er die Basisobjekte auf einer Zeichenfläche positioniert. Dabei kann er Position, Größe, Winkel und Spiegelungen frei wählen. Um die erwünsche Transformation festzulegen, bestimmt der Benutzer eine Abfolge aus drehen, spiegeln und hinzufügen/entfernen. Wenn er nun den IQ test erstellt, wird Bild A zu Bild B transformiert und C zu irgendeinem der 5 Auswahlmöglichkeiten. Um die restlichen 4 zu erzeugen, wird die Transformationsliste leicht abgeändert. Damit die Abänderung nicht zu leicht war und 5 gleiche Bilder entstehen, wird nach dem erstellen eines random Bildes immer getestet, ob es mit dem Original übereinstimmt und wenn nötig wir die Transformation neu variiert.

Beim ermitteln ob die ein entstandenes Bild richtig oder falsch ist, habe ich mir die Frage gestellt, ob es sinnvoll ist, die Transformationen „intelligent" herzuleiten oder einfach à la Brute force alle Kombinationen durchzutesten. Bei einer Rotationsmöglichkeit von 360° wollte ich mir und meinem Rechner nicht antun. Deshalb hab ich mir überlegt, wie das der Rechner auch anders machen könnte. Es sollte die Transformationen direkt an den Objekten feststellen. Dadurch entstand auf vielen kleinen Skizzenblättern viele Regeln. Irgendwie konnte ich sie dann aber nicht unter einen Hut bringen. Aus diesem Grund habe ich mich entschlossen, die Rotationseinschränkung aus der intelligenteren Methode zu verwenden um meine Brute Force Attacke zu optimieren.

Programmdokumentation:

Basisobjekte:

Die Basisobjektdatenbank liegt in beiden Programmen als `objlib` Array vor. In ihr befinden sich für jedes einzelne Basisobjekt die Angaben, ob es horizontal symmetrisch (hor=true → symetrisch) und vertikal symmetrisch ist und nach wie viel Grad es sich wiederholt (winekl=1° → wenn ich des Objekt um 1 Grad drehe, sieht es immer noch so aus wie vorher, trifft für den Kreis zu). Die einzelnen Teile werden in dem festen Array `teile` gespeichert. Es sind deshalb nur 15 Kreise oder Linien in einem Objekt möglich. Die Typendefinitionen um die Objektinformationen aufzunehmen sieht so aus.

```
type
Ttypen = (kreis, linie);
Tobj = record
    x1, y1 : shortint;
    case typ : Ttypen of
    kreis : (r: byte);
    linie : (x2, y2: shortint);
    END;
Tobjekt = record
    len: byte;
    teile : array[0..15] of Tobj;
    hor, vert: boolean;
    winkel : word;
    END;
var objlib : array of Tobjekt;
```

Zusammengesetzten Bilder:

Um im IQ-Test Generator die einzelnen Objekte der kleinen Bildchen zu speichern habe ich die Typen `Telement` und `Tbild` eingeführt. Im Programm selbst existiert ein Array der die 8 Tbilder aufnimmt. Die Variable `mark` ist dafür zuständig, dass beim Zeichen das Markierte Objekt rot gezeichnet wird. Der Array of `Telement` nimmt die Objekte auf, die dem Bild zu sehen sein sollen. Dabei speichert er Typ (= Nummer des Objekts in der `objlib` Variable), `x` und `y` – Offset vom Ursprung, die `Groesse` den Dreh-`winkel` und in `hor` und `vert` ob ein Objekt horizontal oder vertikal gespiegelt ist.

```
Telement = record
    typ : byte;
    x, y : integer;
    groesse : real;
    winkel : word;
    hor, vert: boolean;
    END;

Tbild = record
    mark : word;
    element: array of Telement;
    END;
```

Transformations- Anweisungen:

Um eine Transformationsabfolge für das Programm zu realisieren habe ich die den Datenrecord `Ttransformation` verwendet.

```
Ttrans = (spiegeln, drehen, hinzu);
Ttransformation = record
     case      typ : Ttrans of
      spiegeln : (hor, vert : boolean);
      drehen : (winkel : word);
      hinzu : (loeschen : boolean; art, typennr, x, y : shortint; gr : real);
     END;
TARRAYTrans = array of Ttransformation;
var transformation, fake : Tarraytrans;
```

Die Variable `transformation` speichert die vom Benutzer erwünsche Transformation. Die Variable `fake` wir zum erstellen von falschen Transformationsbeschreibungen verwendet. Ich habe den Umweg über den Typ TARRAYTrans verwendet, um die komplette Transformation besser mit Subroutinen behandeln zu können. Einzelne Objekte des Array besitzen verschiedene Informationen, je nach dem ob sie eine Spiegelung, Rotation oder ein Hinzufügen/Löschen ausdrücken sollen.

Transformation von Bildern

Die Transformation übernimmt die `Procedure transformpic`. Sie verlangt dafür die Angabe des zu verändernden Bildes und die Transformationsbeschreibung. Dann führt sie für das angegebne Bild alle Transformationen nacheinander durch. Die Transformation selbst übernimmt die `Procedure objedit`. Beim Hinzufügen wird ein Objekt ausgewählt und an das ende des `bild.element` Array gehängt. Damit die 2 Objekte sich nicht verdecken, sollten entweder Größe oder Position verändert werden. Beim Löschen wird einfach das Objekt aus dem `bild.element` Array entfernt. Rotation bedeutet nicht nur das der Winkel der einzelnen Objekte verdreht wird. Hinzu kommt noch, dass die x und y Werte der einzelnen Basisobjekte mitrotieren. Bei der Rotation ist zu beachten, dass ein einfach gespiegeltes Objekt sich um den negativen Winkel drehen muss, damit die Transformation richtig erscheint. Das Spiegeln beseht ebenso wie das Drehen nicht nur im Spiegeln der einzelnen Bildelemente sonder im Vorzeichenwechsel der x und y Komponenten.

Erstellen von falschen Bildern

Um falsche Bilder zu erstellen verwende ich als Transformationsarray nicht den Standardarray sondern die abgeänderte Kopie `fake`. Um eine Variation zu erhalten, verwende ich die Funktion `edittrans`. Sie verändert Rotationswinkel um vielfache von 90°, ändert Spiegelungen oder die Parameter beim hinzufügen oder Löschen von Objekten. Außerdem kann sie neue Transformationen hinzufügen. Eine veränderte Transformation reicht aber noch lange nicht für eine anders aussehendes Bild aus. Deshalb habe ich die Funktion `comparebild` implementiert, welche wenn 2 Bilder gleich sind true wird und false, wenn sie sich unterscheiden. In Kombination laufen diese Prozedur so ab: Es wird so lange die `fake` Transformation verändert und damit ein Bild erstellt bis dadurch ein neues Bild entstanden ist (`comparebild` = false).

Vergleichen zweier Bilder

Die Funktion `comparebild` übernimmt das vergleichen zweiter Bilder. Sie wird bei zwei verschiedenen Programmabschnitten verwendet und besitzt deshalb auch leicht unterschiedliche Funktionsweisen. Wenn der ohne Parameter auf false gesetzt ist, dann ist sie beim Erstellen von Falschbildern verwendet worden. Bei diesen ist auch schon eine Veränderung erkennbar, wenn sich die Anzahl der Objekte unterscheidet. Wenn allerdings bei der Brute Force Attack die Bildergleichheit untersucht werden soll, ist die Objektanzahlgleichheit nicht mehr relevant, da die Hinzu/Entfernen Transformationen ja nicht beim Brute Force durchgeführt werden. Die Suche nach identischen Objekten funktioniert bei beiden Varianten gleich.

```
falsche := 0;
WENN Bild 1 Objekte enthält dann
  SCHLEIFE (i) für alle Bildelemente von Bild 1
    falsch := false;
    k := -1;
    Wiederhole
      Erhöhen von k
      Wenn k großer als die Anzahl der Bildelemente von Bild 2 ist dann
        falsch := true;
        Erhöhen von falsche
        k := 0;
    BIS falsch ODER (Größe, Typ, x und y Wert {auf 1.5 Pixel genau) stimmt bei Objekt i vom Bild
                     1 mit denen von Objekt k Bild 2 überein)
      UND (
        (Winkel, Horizontal und Vertikalspiegelung stimmen überein)
        ODER
        (Bildobjekt ist einfach symmetrisch und Winkel bis auf eine Differenz von der Hälfte des
          Symmetriewinkels identisch und mindestens eine Spiegelung vorhanden)
      )
Wenn falsche maximal so groß wie die Objektdifferenz ist dann Richtig zurückgeben
SONST Falsch zurückgeben
```

Beim Vergleich der Repeat-Schleife ist der erste Teil (bis zum zweiten oder) gleich. Die Schleife wird abgebrochen, wenn sich zwei Objekte in allen Parametern (fast) identisch sind. Der Ausdruck nach dem oder existiert, um einen Sonderfall (den ich nie richtig nachstellen konnte) zu behandeln. Mir ist aufgefallen, dass Objekte, die nur eine Symmetrieachse besitzen, nach einer Rotation um die Hälfte ihres Symmetriewinkels identisch zum gespiegelten (natürlich an der anderen Achse) Objekt sind.

Die Entscheidung ob True oder False funktioniert so: Die Variable `falsche` entspricht der Anzahl von Objekten aus Bild 1 die kein Pendant in Bild 2 besitzen. Ist die Objektanzahl von Bild 1 um x größer als die von Bild 2 dann dürfen natürlich auch maximal x zuweisbare Bilder vorkommen.

Wie kommen die Bilder auf das Canvas?

Das Programm besitzt 8 verschiedene Images. Die Bilder A, B; C werden mit `figa, figb, figc` angesprochen. Die 5 anderen befinden sich im array `fig` und werden zur Laufzeit erstellt. Um das Bemalen ihrer Zeichenflächen zu erleichtern, habe ich die Function `drawpic(inp:Tbild; var output:TImage)` erstellt. Sie liest die Objekte aus dem Bildarray aus und zeichnet sie nacheinander auf dem Canvas des angegebenen Bildes. Das Objekt, mit dem Index aus der `mark` Variable wird rot gezeichnet, der Rest schwarz. In einem TBild werden nur eine Abfolge von Elementen mit Typ, Größe, x und y Koordinate, Winkel und Spiegelung gespeichert. Um ein Bild zu erzeugen muss das Programm so vorgehen: Die Einzelnen Bauteile eines Objektes entnimmt das Programm der `objlib` Variable.
Die x und y Koordinaten der einzelnen Elemente müssen um den angegebenen Winkel gedreht werden, passend vergrößert/verkleinert und horizontal oder vertikal gespiegelt werden. Daraus ergibt sich diese Formel für x und y Werte.

```
x := vert * round(groesse * cos(winkel) * x1 - groesse * sin(winkel) * y1)
y := hor * round(groesse * sin(winkel) * x1 + groesse * cos(winkel) * y1)
```

Das ist die Umrechnung für einen Punkt. Bei Kreis(der durch ein Quadrat definiert wird) und Linien werden immer 2 Punke benötigt. Wobei man die beiden Punkte des Quadrats um den Kreis über Mittelpunkt und Radius berechnet. Die vom Benutzer einstellbare x und y-Verschiebung muss auch noch hinzuaddiert werden. Wenn das für jedes Element der Objekte und alle Objekte des Bildes durchgeführt wurde, ist das Bild fertig.

Die intelligente Variante:

Bei dieser Variante versucht das Programm die Entscheidung richtig oder falsch nicht durch einfaches Rumprobieren herauszufinden, sondern ermittelt die Transformation zwischen jeweils 2 von 4 Bildern und vergleicht diese. Bei der Transformationsermittlung geht es so vor:

Zuerst wird getestet ob die Veränderung der Objektanzahl von Bild 1 auf Bild 2 gleich der von Bild 3 auf Bild x ist. Ist dies nicht der Fall kann das Ergebnis schon nicht mehr richtig sein und das Programm bricht ab. Danach testet das Programm ob die Größenverhältnisse der einzelnen Objekte übereinstimmen. Auch hier kann bei einem Fehler das entstandene Bild nur falsch sein.

Jetzt ermittelt das Programm für Bildgruppe 1 (Bild 1 und 2) und Bildgruppe (Bild 3 und x) die Art der Rotations- und Spiegelungstransformationen. Es müssen jeweils 4 Berechnungen für jede Bildgruppe durchgeführt werden. Ich möchte das Vorgehen anhand der Objektgruppe 1 näher erläutern.

Das Programm beginnt mit den Spiegelungen, die es Anhand der Betrachtung der Betrachtung der Objekte, also ohne Beachtung der Veränderung von irgendwelchen x und y-Werten feststellen kann. Es sollen dabei Folgende Parameter entstehen: hors1 und verts1 geben an ob anhand ein Objekte eine Spiegelung überhaupt feststellbar war, und hor1 und vert1 um welche art es sich gehandelt hat. Um diese Parameter zu bestimmen wird für alle Objekte eines Bildes getestet, ob sie horizontal oder vertikal symmetrisch sind. Ist dies nicht der Fall, kann ermittelt werden, ob das Objekt horizontal oder vertikal gespiegelt wurde und `hors1` oder `verts1` auf `true` gesetzt werden. Im Quelltext sieht das dann so aus.

```
hors1 := False;
verts1 := False;
Schleife für alle Objekte
  Wenn der Objekttyp nicht horizontal symmetrisch ist dann
    Wenn sich die Spiegelung nicht unterscheidet dann hor1 := 1
    Sonst hor1 := -1;
    hors1 := True;
  Wenn der Objekttyp nicht vertikal symmetrisch ist dann
    Wenn sich die Spiegelung nicht unterscheidet dann vert1 := 1
    Sonst vert1 := -1;
    verts1 := True;
```

Spiegelung drückt sich aber nicht nur durch die Objekte aus sondern auch durch deren x und y-Koordinaten. Der nächste Programmteil soll aus x und y Veränderungen die Spiegelung der Bilder ermitteln. Hierzu betrachtet er auch alle Elemente und testet ob die Koordinaten betragsmäßig gleich sind. Wenn ja kann an ihnen festgestellt werden, dass sie vielleicht (!!!) gespiegelt wurden. hora1 wird 1 wenn keine Spiegelung vorliegt, 0 wenn man nichts erkennen kann und -1 wenn gespiegelt wurde. Implementiert habe ich das so:

```
hora1 := 0;
verta1 := 0;
Schleife für alle Objekte
  Wenn die Koordinaten betragsgleich sind dann
    Wenn x Koordinate <> 0 dann
      verta1 := x-Koordinate Objekt Bild1 Div x-Koordinate Objekt Bild2
    Wenn y Koordinate <> 0 dann
      hora1 := y-Koordinate Objekt Bild1 Div y-Koordinate Objekt Bild2
```

Ebenso wie bei der Spiegelung kann man die Rotation nur durch die Betrachtung der Objekte ermitteln. Dabei geht man auch alle Objekt durch und vergleicht ihre Rotation. Da sich aber die Rotationen bei den meisten Objekten wegen ihrer Symmetrie nicht auf 360° genau vorhersagen lassen, muss man bei mehreren Objekten etwas Geschick anwenden, um die sinnvollste Lösung zu ermitteln. Es soll bei der Berechnung der Rotationswinkel und die Symmetriewinkel (Winkel für eine Deckungsgleichheit) ermittelt werden.

```
winkel1 := 0
symwink1 := 1
Schleife für alle Objekte
  Wenn das Objekt (Bild1) einmal gespiegelt wurde dann
```

```
          dummyw1 := Symmetriewinkel des Objektes - Winkel des Objektes
        Wenn nicht (2 mal oder 0 mal)
          dummyw1 := Winkel des Objektes
        Wenn das Objekt (Bild2) einmal gespiegelt wurde
          dummyw2 := Symmetriewinkel des Objektes - Winkel des Objektes
        Wenn nicht (2 mal oder 0 mal)
          dummyw2 := Winkel des Objektes
(*A*)   Winkel1 soll der minimale Winkel sein, für den die Gleichung
             winkel+x*Symmetriewinkel = Winkel des Objektes + y*Symmetriewinkel des Objektes
             erfüllt ist.
        Wenn Symmetriewinkel = 1 Dann Symmetriewinkel := Symmetriewinkel des Objektes
        Wenn nicht
          Symmetriewinkel := kgV(Symetriewinkel, Symetriewinkel des Objektes)
```

Den Programmschritt bei A) habe ich in die Funktion `winkelmoeg` ausgelagert. Sie setzt die Werte x und y zuerst auf 0 und erhöht dann, bei der Gleichungsseite mit dem kleineren Wert, das x oder y, bis die Gleichung erfüllt ist. Die Funktion gibt dann als Ausgabe den Wert einer Gleichungsseite aus. Durch die Verwendung das kgV erziehe ich das der Symmetriewinkel immer passend vergrößert wird. Sind die Zwei Objekte, z.B. ein Viereck und ein Dreieck, so entsteht aus kgV(120, 90) der Wert 360. Was bedeutet das ich dadurch die Rotation des Bildes exakt bestimmen kann.

Auch durch die Veränderung der x und y-Koordinaten kann eine Bilddrehung berechnet werden. Dabei geht das Programm so vor:

```
rot1 := 360;
Schleife für alle Bildelemente
  Wenn x oder y Abstand von 0 grösser als 10 ist dann
    rot1 := (arctan2(bild[2].element[i].x, bild[2].element[i].y) -
      arctan2(bild[1].element[i].x, bild[1].element[i].y)) / PI * 180
```

Durch die entstandenen Parameter kann man meist (vielleicht auch immer, wenn man noch etwas optimiert) die Gleichheit der Transformationsregeln feststellen. In meinem Fall mache ich das so:

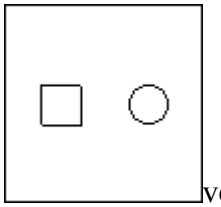
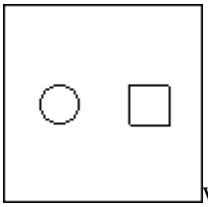
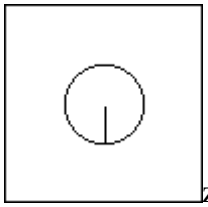
```
Wenn nicht 100%ig horizontal oder Vertikalspiegelung übereinstimmt (Koordinatenermittelt)
  Wenn mit winkelmoeg die winkel1 und winkel2 durch symwink1 und symwink2 gleichwertig gemacht
    werden können (Objektermittelt) dann:
    Wenn wenigstens eine Spiegelung unsicher ist (Objektermittelt) dann
      Wenn im ersten Bild keine Spiegelung feststellbar ist (Koordinatenermittelt) dann
        Wenn Koordinatenermittelte Rotation zutrifft dann Richtig
      Wenn überhaupt keine Spiegelung feststellbar ist dann Richtig
    SONST WENN ALLE Spiegelungen sicher feststellbar sind (Objektermittelt) dann
      WENN Spiegelungen übereinstimmen dann Richtig
    SONST WENN eine Spiegelung unsicher und dafür die andere übereinstimmt dann Richtig
```

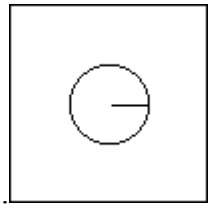
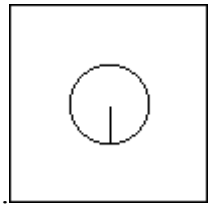
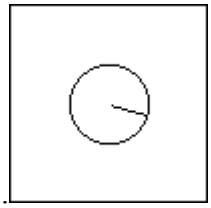
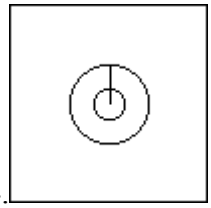
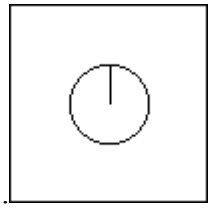
Brute Force:

Die Brute Force Variante beginnt genau so wie die zuvor beschriebene „Intelligente“ Variante. Nur dass die Parameterberechnung nur die Rotation anhand der Objekte verwendet und den Rest nicht berechnet. Die Werte `winkel` und `symwink` aus dieser Berechnung werden verwendet, um die Brute Force Versuche zu minimieren. Die verschiedenen Versuch habe ich eine rekursive Funktion gepackt, die den wert true zurückliefert, wenn einmal ein richtiges Ergebnis auftaucht.

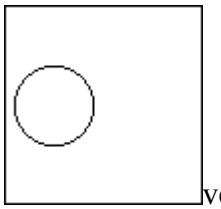
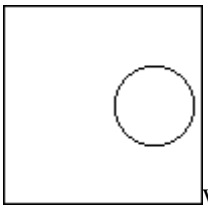
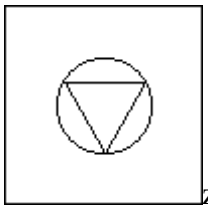
```
weitermachen := True;
False zurückgeben
Wenn die Rekursionstiefe < 3 dann
  Bild 1 auf Bild 9 kopieren
  Bild 9 mit fake transformieren
  Wenn Bild 9 mit Bild 2 übereinstimmt dann
    Bild 3 auf Bild 9 kopieren
    Bild 9 mit fake transformieren
    Wenn Bild 9 mit Bild inp übereinstimmt dann
      True zurückgeben
      weitermachen := False;                                  //keine weitere Berechnung angestellt
  Wenn weitermachen dann
    Fake um eins verlängern
    mom := letztes Element von Fake
    Typ des momentanen Elements von fake auf drehen setzen
    i := 0;
    Wiederhole
      Rotationswinke des Fake-Elements := winkel + i;
      Wenn bruteatack(inp, len + 1) Dann
        True zurückgeben
        i := 2000;
      I um den Symmetriewinkel erhöhen
    Bis I großer als 360 ist
    Wenn i <= 2000 (entspricht noch kein true beim Rotationstest) DANN
      Typ des momentanen Elements von fake auf spiegeln setzen
      Horizontal spiegeln, Vertikal nicht in fake schrieben
      Wenn bruteatack(inp, len + 1) Dann True Zurückgeben
      SONST
        Vertikal Spiegeln in Fake schreiben
        Wenn bruteatack(inp, len + 1) Dann True Zurückgeben
        SONST
          Nicht vertikal Spiegeln in Fake schreiben
          bruteatack(inp, len + 1) zurückliefern
    Länge von Fake um eins reduzieren
```

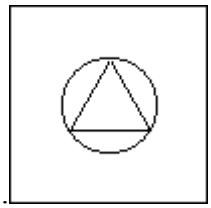
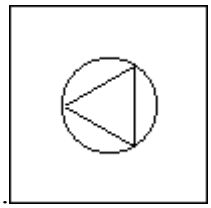
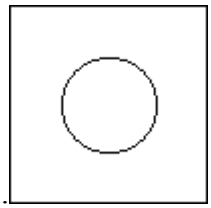
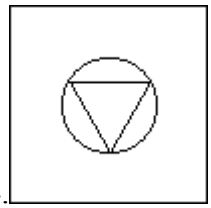
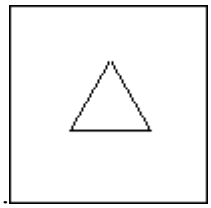
Beispiele:

Figur A  verhält sich zu Figur B  wie Figur C  zu:

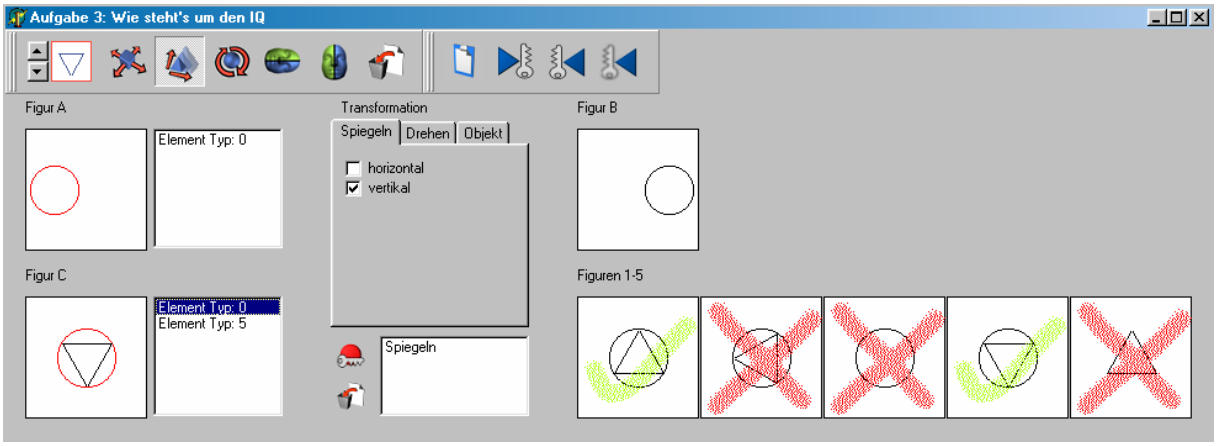
1.  2.  3.  4.  5. 

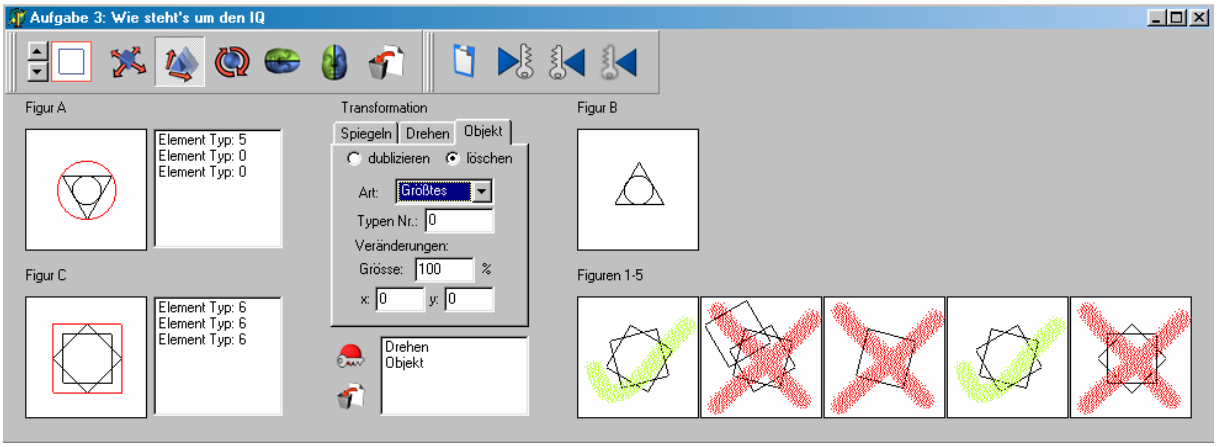
Transformation: Drehung um 180° Richtig: 2, 5

Figur A  verhält sich zu Figur B  wie Figur C  zu:

1.  2.  3.  4.  5. 

Transformation: Spiegelung and Vertikaler Richtig 1 und 4





Programmablaufprotokoll:

Um eine Quiz herzustellen, müssen erst einmal 2 Bildchen aus den Basisobjekten zusammengestellt werden.

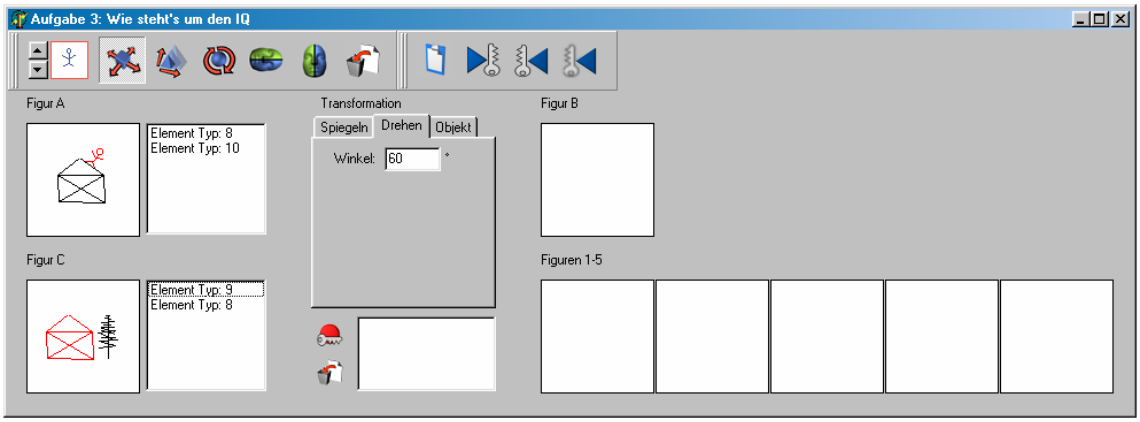
 Dazu wählt man sich den up und down Knöpfen ein passendes Basiselement aus. Die Schaltfläche ist danach aktiviert.

Wenn man jetzt auf eine der zwei linken Zeichenflächen klickt, wird dort das Bild eingefügt.

Dann kann man einen dieser  Knöpfe aktivieren um die eingefügten Basisobjekte passend zu positionieren, die Größe zu verändern und zu drehen. Außerdem können die Bilder horizontal und vertikal gespiegelt werden, und auch wieder gelöscht.

Diese Transformationen wirken sich immer auf dir rot markierten Objekte aus. Will man ein anderes transformieren, so muss man in der nebenstehenden Liste das passende Objekt anklicken.

Nachdem sie die zwei Ausgangsbilder A und C erstellt haben könnte das Programm ungefähr so aussehen:



Nun müssen noch die Transformation festgelegt werden. Dazu wählt man die gewünschte Karteikarte aus, in diesem Fall drehen, trägt in das Feld den Winkel ein und drückt auf diesen Knopf:  . Danach befindet sich die Transformation in der nebenstehenden Liste. Ist die Transformation unerwünscht so kann man sie mit dem  Knopf neben der Liste wieder löschen.

Nun kann man eine Transformation durchführen.

 Transformiert die Bilder A und C in die Bilder B und 1 bis 5.

 Berechnet für jedes der Bilder 1 bis 5 ob sie richtig oder falsch sind.

 Erstellt Bitmaps aus den allen Bilder, und fasst diese in einer HTML Datei zusammen.

Probleme:

Die Hauptprobleme bereitete mir natürlich der Aufgabenteil 3. Prinzipiell kann ein von mir geschriebenes Programm ja nicht intelligenter sein als ich (Hoffentlich!), also haben wir schon mal Schranken gefunden. Außerdem ist Intelligenz auch Definitionssache. Und ich habe mich an 2 verschiedne Varianten der Lösungsfindung bemüht.

Bei der ersten spielt meine Intelligenz eher eine Rolle als bei der Zweiten. Um dem Programm die verschiedenen Regeln zum Vergleichen zweier Transformationen beizubringen habe ich versucht möglicht allgemein über die Transformationen nachzudenken. Leider hat das nur Teilweise gefruchtet. Sobald es zu gewissen Objektkombinationen kommt, versteht es meine Logik nicht mehr. Ich denke aber das ich meine Vorgehensweise noch verbessern könnte.

Die zweite Variante ist eher die eines kleinen Kindes, das beim Holzsteckkasten die Kuh zwar dem richtig ausgesägten Loch zuweisen kann, aber dann erst einmal mehrere Drehungen braucht, bis diese dort reinpasst. Brute Force ist dank der Rechenleistung heutiger PC leicht möglich, aber immer nur eine Notlösung.

Quelltexte

OBJEKTE.PAS

```
UNIT objekte;

INTERFACE

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

TYPE
  TForm1 = CLASS(TForm)
    Image1: TImage;
    ListBox1: TListBox;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    x1: TEdit;
    y1: TEdit;
    x2: TEdit;
    y2: TEdit;
    r: TEdit;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    StaticText3: TStaticText;
    StaticText4: TStaticText;
    StaticText5: TStaticText;
    ListBox2: TListBox;
    StaticText6: TStaticText;
    StaticText7: TStaticText;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    StaticText8: TStaticText;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    Label1: TLabel;
    Edit1: TEdit;
    PROCEDURE Button2Click(Sender: TObject);
    PROCEDURE Button1Click(Sender: TObject);
    PROCEDURE Button3Click(Sender: TObject);
    PROCEDURE Button6Click(Sender: TObject);
    PROCEDURE FormCreate(Sender: TObject);
    PROCEDURE FormDestroy(Sender: TObject);
    PROCEDURE ListBox1Click(Sender: TObject);
    PROCEDURE Button4Click(Sender: TObject);
    PROCEDURE Button5Click(Sender: TObject);
    PROCEDURE ho(Sender: TObject);
    PROCEDURE CheckBox3Click(Sender: TObject);
    PROCEDURE Image1MouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    PROCEDURE Image1MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    PROCEDURE Edit1Change(Sender: TObject);
  Private
    { Private-Deklarationen }
  Public
    { Public-Deklarationen }
  END;

TYPE
  Ttypen = (kreis, linie);
  Tobj = RECORD
    x1, y1: Shortint;
    CASE typ: Ttypen OF
      kreis: (r: Byte);
      linie: (x2, y2: Shortint);
  END;
  Tobjekt = RECORD
    len: Byte;
    teile: ARRAY[0..15] OF Tobj;
    hor, vert: Boolean;
```

```
    winkel: Word;
  END;
VAR
  Form1: TForm1;
  objlib: ARRAY OF Tobjekt;
  momentan: Word = 0;

IMPLEMENTATION

{$R *.DFM}

PROCEDURE listitems2();
VAR
  i: Byte;
BEGIN

  form1.listbox2.Items.Clear;
  IF objlib[momentan].len <> 0 THEN
    FOR i := 0 TO objlib[momentan].len - 1 DO WITH objlib[momentan].teile[i] DO
        IF typ = linie THEN
          form1.listbox2.Items.Add('Linie (' + IntToStr(x1) + ',' + IntToStr(y1) + ',' +
            IntToStr(x2) + ',' + IntToStr(y2) + ')')
    ELSE
      form1.listbox2.Items.Add('Kreis (' + IntToStr(x1) + ',' + IntToStr(y1) + ',' + IntToStr(r) +
')')
END;

PROCEDURE TForm1.Button2Click(Sender: TObject);
BEGIN


  objlib[momentan].teile[objlib[momentan].len].typ := kreis;
  objlib[momentan].teile[objlib[momentan].len].x1 := StrToInt(x1.Text);
  objlib[momentan].teile[objlib[momentan].len].y1 := StrToInt(y1.Text);
  objlib[momentan].teile[objlib[momentan].len].r := StrToInt(r.Text);
  inc(objlib[momentan].len);
  listitems2;
END;

PROCEDURE TForm1.Button1Click(Sender: TObject);
VAR
  i: Byte;
BEGIN
  image1.Canvas.Brush.Style := bssolid;
  image1.Canvas.Rectangle(0,0,99,99);
  image1.Canvas.Brush.Style := bsClear;
  IF objlib[momentan].len <> 0 THEN
    FOR i := 0 TO objlib[momentan].len - 1 DO
    BEGIN
      IF objlib[momentan].teile[i].typ = kreis THEN
        WITH objlib[momentan].teile[i] DO image1.Canvas.Ellipse(50 + x1 - r,
            50 + y1 - r, 50 + x1 + r, 50 + y1 + r);

      IF objlib[momentan].teile[i].typ = linie THEN
        WITH objlib[momentan].teile[i] DO
        BEGIN

          image1.Canvas.MoveTo(50 + x1, 50 + y1);
          image1.Canvas.LineTo(50 + x2, 50 + y2);
        END;

    END;
END;

PROCEDURE TForm1.Button3Click(Sender: TObject);
BEGIN


  objlib[momentan].teile[objlib[momentan].len].typ := linie;
  objlib[momentan].teile[objlib[momentan].len].x1 := StrToInt(x1.Text);
  objlib[momentan].teile[objlib[momentan].len].y1 := StrToInt(y1.Text);
  objlib[momentan].teile[objlib[momentan].len].x2 := StrToInt(x2.Text);
  objlib[momentan].teile[objlib[momentan].len].y2 := StrToInt(y2.Text);
  inc(objlib[momentan].len);
  listitems2;
END;

PROCEDURE TForm1.Button6Click(Sender: TObject);
VAR
  i, k: Integer;
BEGIN
  i := -1;
  REPEAT
    inc(i)
  UNTIL listbox2.Selected[i];
  FOR k := i + 1 TO length(objlib[momentan].teile) - 1 DO
    objlib[momentan].teile[k - 1] := objlib[momentan].teile[k];
  dec(objlib[momentan].len);
  listitems2;
END;

PROCEDURE TForm1.FormCreate(Sender: TObject);
VAR
  f: FILE OF Tobjekt;
BEGIN
```

```
  assignfile(f, 'objlib.dat');
  reset(f);
  REPEAT
    setlength(objlib, length(objlib) + 1);
    Read(f, objlib[length(objlib) - 1]);
    listbox1.Items.Add('Element Nr.' + IntToStr(length(objlib)));
  UNTIL EOF(f);

  closefile(f);


END;

PROCEDURE TForm1.FormDestroy(Sender: TObject);
VAR
  f: FILE OF Tobjekt;
VAR
  i: Byte;
BEGIN
  assignfile(f, 'objlib.dat');
  rewrite(f);
  FOR i := 0 TO length(objlib) - 1 DO Write (f, objlib[i]);
  closefile(f);

END;

PROCEDURE TForm1.ListBox1Click(Sender: TObject);
VAR
  i: Integer;
BEGIN
  i := -1;
  REPEAT
    inc(i)
  UNTIL listbox1.selected[i];
  momentan := i;

  checkbox2.Checked := objlib[momentan].hor;
  checkbox3.Checked := objlib[momentan].vert;
  edit1.Text := IntToStr(objlib[momentan].winkel);
  listitems2;
  button1.Click;
END;

PROCEDURE TForm1.Button4Click(Sender: TObject);
BEGIN
  setlength(objlib, length(objlib) + 1);
  objlib[length(objlib) - 1].len := 0;
  listbox1.Items.Add('Element Nr.' + IntToStr(length(objlib)));
END;

PROCEDURE TForm1.Button5Click(Sender: TObject);
BEGIN
  setlength(objlib, length(objlib) - 1);
  listbox1.Items.Delete(length(objlib));
END;

PROCEDURE TForm1.ho(Sender: TObject);
BEGIN
  objlib[momentan].hor := checkbox2.Checked;
END;

PROCEDURE TForm1.CheckBox3Click(Sender: TObject);
BEGIN
  objlib[momentan].vert := checkbox3.Checked;
END;

PROCEDURE TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
BEGIN
  IF ssRight In shift THEN
  BEGIN
    x2.Text := IntToStr(x - 50);
    y2.Text := IntToStr(y - 50);
    r.Text := IntToStr(round(sqrt(sqr(StrToInt(x1.Text) - x + 50) + sqr(StrToInt(y1.Text) - y +
50))));
  END ELSE IF ssleft In shift THEN
  BEGIN
    x1.Text := IntToStr(x - 50);
    y1.Text := IntToStr(y - 50);
  END;

END;

PROCEDURE TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
BEGIN
  IF ssRight In shift THEN
  BEGIN
    x2.Text := IntToStr(x - 50);
    y2.Text := IntToStr(y - 50);
    r.Text := IntToStr(round(sqrt(sqr(StrToInt(x1.Text) - x + 50) + sqr(StrToInt(y1.Text) - y +
50))));
  END ELSE IF ssleft In shift THEN
  BEGIN
    x1.Text := IntToStr(x - 50);
```

```
    y1.Text := IntToStr(y - 50);
  END;
END;

PROCEDURE TForm1.Edit1Change(Sender: TObject);
BEGIN
  objlib[momentan].winkel := StrToInt(edit1.Text);
END;

END.
```

UNIT1.PAS

```
UNIT Unit1;

INTERFACE

USES
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Buttons, ComCtrls, ToolWin, math, ImgList;

TYPE
  TForm1 = CLASS(TForm)
    add: TSpeedButton;
    changeadd: TUpDown;
    move: TSpeedButton;
    resize: TSpeedButton;
    rotate: TSpeedButton;
    ControlBar1: TControlBar;
    CoolBar1: TCoolBar;
    horspiegeln: TSpeedButton;
    vertspiegeln: TSpeedButton;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    bildalist: TListBox;
    figa: TImage;
    loeschen: TSpeedButton;
    StaticText3: TStaticText;
    bildclist: TListBox;
    figc: TImage;
    CoolBar2: TCoolBar;
    berechnen: TSpeedButton;
    zurueckrechnen: TSpeedButton;
    transformarten: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    transhor: TCheckBox;
    transvert: TCheckBox;
    StaticText4: TStaticText;
    transwinkel: TEdit;
    dubliz: TRadioButton;
    loesch: TRadioButton;
    transart: TComboBox;
    transnr: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    transgr: TEdit;
    transx: TEdit;
    transy: TEdit;
    Label5: TLabel;
    transformlist: TListBox;
    transformadd: TSpeedButton;
    transformkill: TSpeedButton;
    figb: TImage;
    StaticText5: TStaticText;
    StaticText6: TStaticText;
    ImageList1: TImageList;
    zurueckrechnen2: TSpeedButton;
    PROCEDURE FormCreate(Sender: TObject);
    PROCEDURE addMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    PROCEDURE anderebuttons(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    PROCEDURE addMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    PROCEDURE FormResize(Sender: TObject);
    PROCEDURE figMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    PROCEDURE addchanged(Sender: TObject; Button: TUDBtnType);
    PROCEDURE figMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    PROCEDURE horspiegelnClick(Sender: TObject);
    PROCEDURE vertspiegelnClick(Sender: TObject);
    PROCEDURE loeschenClick(Sender: TObject);
    PROCEDURE bildalistClick(Sender: TObject);
    PROCEDURE bildclistClick(Sender: TObject);
    PROCEDURE transformaddClick(Sender: TObject);
    PROCEDURE transformkillClick(Sender: TObject);
    PROCEDURE berechnenClick(Sender: TObject);
    PROCEDURE exportClick(Sender: TObject);
    PROCEDURE zurueckrechnenClick(Sender: TObject);

  Private
    { Private-Deklarationen }
  Public
    fig: ARRAY[1..5] OF Timage;                                  //die 5 auszuwählenden Bilder
  END;

  Ttypen = (kreis, linie);                                        //die 2 Typen in der objlib
  Tobj = RECORD                                                    //Element aus einem Objekt
    x1, y1: Shortint;
    CASE typ: Ttypen OF
```

```
      kreis: (r: Byte);
      linie: (x2, y2: Shortint);
  END;
  Tobjekt = RECORD                                    //Basisobjekt mit Symetrieangaben und Bauplan
    len: Byte;
    teile: ARRAY[0..15] OF Tobj;
    hor, vert: Boolean;
    winkel: Word;
  END;
  Telement = RECORD                                                  //Element aus einem TBild
    typ: Byte;
    x, y: Integer;
    groesse: Real;
    winkel: Word;
    hor, vert: Boolean;
  END;

  Tbild = RECORD                                  //Bauplan eines Bildes und das markierte Element
    mark: Word;
    element: ARRAY OF Telement;
  END;

  Ttrans = (spiegeln, drehen, hinzu);                                  //Transformationstypen
  Ttransformation = RECORD

    CASE typ: Ttrans OF                             //Parameter zu den einzelnen Transformationen
      spiegeln: (hor, vert: Boolean);
      drehen: (winkel: Word);
      hinzu: (loeschen: Boolean; art, typennr, x, y: Shortint; gr: Real);

  END;
  TARRAYTrans = ARRAY OF Ttransformation;          //Array von verschiedenen Transformationen
VAR
  Form1: TForm1;
  objlib: ARRAY OF Tobjekt;                                                  //Objektdatenbank
  bild: ARRAY [1..9] OF Tbild;                                          //8+1 verschiedene Bilder
  mombild: Byte = 1;                                                  //momentan aktiviertes Bild
  transformation, fake: Tarraytrans;              //benutzerdefinierte Transformation + manipulierte

  winkel, symwink: Integer;                          //Einschränkung für die Bruteforce Attack


IMPLEMENTATION

{$R *.DFM}

PROCEDURE drawpic(inp: Tbild; VAR output: TImage);   //Zeichnet die einzelnen Objekte des Bildes auf
VAR                                                                          //den Canvas des Timages
  i, k, typ: Byte;
  winkel: Real;
  groesse: Real;
  x, y, hor, vert: Integer;
BEGIN
  output.Canvas.Brush.Style := bssolid;
  output.canvas.Rectangle(0,0,99,99);
  output.Canvas.Brush.Style := bsClear;

  IF length(inp.element) <> 0 THEN
    FOR k := 0 TO length(inp.element) - 1 DO                          //Schleife für alle Elemente
    BEGIN

      IF k = inp.mark THEN output.Canvas.pen.Color := clred;
      typ := inp.element[k].typ;
      x := inp.element[k].x;
      y := inp.element[k].y;
      groesse := inp.element[k].groesse;
      winkel := inp.element[k].winkel / 180 * PI;                                 //Winkel umrechnen

      IF inp.element[k].hor THEN hor := -1                 //Horizontale und vertikale Spiegelung
      ELSE
        hor := 1;
      IF inp.element[k].vert THEN vert := -1
      ELSE
        vert := 1;

      IF objlib[typ].len <> 0 THEN
        FOR i := 0 TO objlib[typ].len - 1 DO
        BEGIN
          IF objlib[typ].teile[i].typ = kreis THEN                                 //Kreis zeichen
            WITH objlib[typ].teile[i] DO
              output.Canvas.Ellipse(x + 50 + vert * round(groesse * cos(winkel) * x1 -
                groesse * sin(winkel) * y1) - round(groesse * r),
                y + 50 + hor * round(groesse * sin(winkel) * x1 +
                groesse * cos(winkel) * y1) - round(groesse * r),
                x + 50 + vert * round(groesse * cos(winkel) * x1 -
                groesse * sin(winkel) * y1) + round(groesse * r),
                y + 50 + hor * round(groesse * sin(winkel) * x1 +
                groesse * cos(winkel) * y1) + round(groesse * r));


          IF objlib[typ].teile[i].typ = linie THEN                                 //Linie zeichnen
            WITH objlib[typ].teile[i] DO
            BEGIN
              output.Canvas.MoveTo(x + 50 + vert * round(groesse * cos(winkel) * x1 -
                groesse * sin(winkel) * y1),
```

```
              y + 50 + hor * round(groesse * sin(winkel) * x1 + groesse * cos(winkel) * y1));
            output.Canvas.LineTo(x + 50 + vert * round(groesse * cos(winkel) * x2 -
              groesse * sin(winkel) * y2),
              y + 50 + hor * round(groesse * sin(winkel) * x2 + groesse * cos(winkel) * y2));
          END;

      END;
    IF k = inp.mark THEN output.Canvas.pen.Color := clblack;
  END;

END;


PROCEDURE Tform1.FormCreate(Sender: TObject);                          //beim Starten des Programmes
VAR
  f: FILE OF Tobjekt;
  i: Integer;
  unwichtig: TUDBtnType;
BEGIN

  application.HintPause := 0;                                 //damit sich die Hints immer sofort zeigen
  application.HintShortPause := 0;

  bild[2].mark := 255;                                             //auf Bild 2 kann nicht markiert sein

  FOR i := 1 TO 5 DO                                                          //erstellen der Bilder 1-5
  BEGIN
    bild[i + 3].mark := 255;
    fig[i] := Timage.Create(self);
    WITH fig[i] DO
    BEGIN

      Parent := form1;
      Width := 99;
      Height := 99;
      Top := 216;
      left := 364 + i * 100;
    END;
  END;


  assignfile(f, 'objlib.dat');                                          //Laden der objlib Variable
  reset(f);
  REPEAT
    setlength(objlib, length(objlib) + 1);
    Read(f, objlib[length(objlib) - 1]);
  UNTIL EOF(f);
  closefile(f);

  changeadd.Max := length(objlib) - 1;                //den "Objekt hinzufügen" Knopf initialisieren
  addchanged(Sender, unwichtig);
  add.Glyph.Transparentcolor := clred;

  drawpic(Bild[1], figa);
  drawpic(Bild[2], figb);
  drawpic(Bild[3], figc);
  FOR i := 1 TO 5 DO drawpic(bild[i + 3], fig[i]);

END;

PROCEDURE TForm1.addMouseDown(Sender: TObject;        //"Objekt hinzufügen" Knopf nach unten drücken
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
BEGIN
  IF Not (add.Down) And (x > 0) And (x < 73) And (y > 0) And (y < 57) THEN
    WITH changeadd DO
    BEGIN
      left := left + 1;
      top := top + 1;
    END;

END;

PROCEDURE TForm1.anderebuttons(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);           //"Objekt hinzufügen" Knopf liften wenn andere Knöpf
BEGIN                                                                                  //aktiviert werden
  IF add.Down And (x > 0) And (x < 73) And (y > 0) And (y < 57) THEN
    WITH changeadd DO
    BEGIN
      left := left - 1;
      top := top - 1;
    END;
END;



PROCEDURE TForm1.FormResize(Sender: TObject);                      //Obere Leiste an Breite anpassen
BEGIN
  controlbar1.Width := clientwidth;
END;



PROCEDURE TForm1.addchanged(Sender: TObject; Button: TUDBtnType);
VAR                        //"Objekt hinzufügen" Knopf soll auf ein anderes Objekt umgestellt werden
```

```
  i: Byte;
BEGIN
  IF Not (add.down) THEN
  BEGIN                                        //Kopf künstlich drücken wenn er noch nicht aktiviert ist
    addmousedown(Sender, mbLeft, [ssLeft], 2,2);
    add.down := True;
  END;

  WITH add.Glyph DO
  BEGIN
    canvas.Pen.Color := RGB(255,81,63);           //altes Bild übermalen und neues Objekt in 50%iger
    Canvas.Brush.Style := bssolid;                                          //auf den Button malen
    Canvas.Rectangle(0,0,33,33);
    Canvas.Brush.Style := bsClear;
    canvas.Pen.Color := RGB(65,90,151);
    IF objlib[changeadd.position].len <> 0 THEN
      FOR i := 0 TO objlib[changeadd.position].len - 1 DO
      BEGIN
        IF objlib[changeadd.position].teile[i].typ = kreis THEN
          WITH objlib[changeadd.position].teile[i] DO
            Canvas.Ellipse(16 + round((x1 - r) / 2), 16 + round((y1 - r) / 2),
              16 + round((x1 + r) / 2), 16 + round((y1 + r) / 2));

        IF objlib[changeadd.position].teile[i].typ = linie THEN
          WITH objlib[changeadd.position].teile[i] DO
          BEGIN

            Canvas.MoveTo(16 + round(x1 / 2), 16 + round(y1 / 2));
            Canvas.LineTo(16 + round(x2 / 2), 16 + round(y2 / 2));
          END;

      END;
  END;

END;


PROCEDURE TForm1.addMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);  //"Objekt hinzufügen" Knopf wird gedrückt und Maus ausserhalb
BEGIN                                                  //ausgelassen --> Button bleibt nicht unten
  IF Not (add.Down) And ((x < 0) Or (x > 73) Or (y < 0) Or (y > 57)) THEN
    WITH changeadd DO
    BEGIN
      left := left - 1;                                //UpDown Knopf wieder richtig positionieren
      top := top - 1;
    END;
END;


PROCEDURE TForm1.figMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);                          //Maus wir auf Bild A oder C gedrückt
BEGIN
  IF Sender = figa THEN mombild := 1                                    //Entscheidung Bild A oder C
  ELSE
    mombild := 3;
  IF abs(x - 50) < 5 THEN x := 50;                        //x und y Werte werden auf Achsen gesnappt
  IF abs(y - 50) < 5 THEN y := 50;
  IF add.Down THEN                                //Wenn Hinzufügen Knopf aktiv ist wird er deaktiviert
  BEGIN
    move.Down := True;
    WITH changeadd DO
    BEGIN
      left := left - 1;
      top := top - 1;
    END;                                                    //Objekt wird dem Bild hinzugefügt
    bild[mombild].mark := length(bild[mombild].element);
    setlength(bild[mombild].element, bild[mombild].mark + 1);
    bild[mombild].element[bild[mombild].mark].typ := changeadd.Position;
    bild[mombild].element[bild[mombild].mark].x := x - 50;
    bild[mombild].element[bild[mombild].mark].y := y - 50;
    bild[mombild].element[bild[mombild].mark].groesse := 1;
    IF mombild = 1 THEN bildalist.Items.Add('Element Typ: ' + IntToStr(changeadd.Position))
    ELSE                                                  //Objekt wird dem Listenfeld hinzugefügt
      bildclist.Items.Add('Element Typ: ' + IntToStr(changeadd.Position));
  END;
  IF mombild = 1 THEN drawpic(bild[mombild], figa)
  ELSE
    drawpic(bild[mombild], figc)
END;


PROCEDURE TForm1.figMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
BEGIN                                  //Maus wird auf Bild A oder C im gedrücken zustand bewegt
  IF Sender = figa THEN mombild := 1                          //Maus wir auf Bild A oder C gedrückt
  ELSE
    mombild := 3;
  IF abs(x - 50) < 5 THEN x := 50;                                      //Entscheidung Bild A oder C
  IF abs(y - 50) < 5 THEN y := 50;

  IF (ssLeft In Shift) And (length(bild[mombild].element) <> 0) THEN
  BEGIN
    IF add.Down Or move.Down THEN                                   //aktives Element wird verschoben
    BEGIN
      bild[mombild].element[bild[mombild].mark].x := x - 50;
      bild[mombild].element[bild[mombild].mark].y := y - 50;
```

```
                                                                    //Hintfenster wird ausgegeben
      IF mombild = 1 THEN  figa.Hint := 'x: ' + IntToStr(x - 50) + '; y: ' + IntToStr(y - 50)
      ELSE
        figc.Hint := 'x: ' + IntToStr(x - 50) + '; y: ' + IntToStr(y - 50);
      application.ActivateHint(mouse.CursorPos);                  //Hintfenster richtig Positioniert

    END;

    IF rotate.Down THEN                                              //aktives Element wird gedreht
    BEGIN
      bild[mombild].element[bild[mombild].mark].winkel :=
        round((arctan2(x - 50, 50 - y) + PI) / PI * 180);
      bild[mombild].element[bild[mombild].mark].winkel :=
        bild[mombild].element[bild[mombild].mark].winkel Mod
        objlib[bild[mombild].element[bild[mombild].mark].typ].winkel;

      IF mombild = 1 THEN figa.Hint := 'Winkel: ' +                  //Ausgabe des Hint Fensters
          IntToStr(bild[mombild].element[bild[mombild].mark].winkel) + '°'
      ELSE
        figc.Hint := 'Winkel: ' + IntToStr(bild[mombild].element[bild[mombild].mark].winkel) + '°';
      application.ActivateHint(mouse.CursorPos);
    END;

    IF resize.Down THEN                                        //Grösse des aktiven Elements verändern
    BEGIN
      bild[mombild].element[bild[mombild].mark].groesse :=
        sqrt(sqr(x - bild[mombild].element[bild[mombild].mark].x - 50) +
        sqr(y - bild[mombild].element[bild[mombild].mark].y - 50)) / 20;
      IF mombild = 1 THEN figa.Hint := 'Grösse: ' +                  //Ausgabe des Hint Fensters
          IntToStr(round(bild[mombild].element[bild[mombild].mark].groesse * 100)) + '%'
      ELSE
        figc.hint := 'Grösse: ' +
          IntToStr(round(bild[mombild].element[bild[mombild].mark].groesse * 100)) + '%';

      application.ActivateHint(mouse.CursorPos);
    END;

    IF mombild = 1 THEN drawpic(bild[mombild], figa)                          //Bild neu zeichen
    ELSE
      drawpic(bild[mombild], figc)

  END;

END;


PROCEDURE TForm1.horspiegelnClick(Sender: TObject);     //aktives Element wird horizontal gespiegelt
BEGIN
  IF length(bild[mombild].element) <> 0 THEN
  BEGIN
    bild[mombild].element[bild[mombild].mark].hor := horspiegeln.Down;
    IF mombild = 1 THEN drawpic(bild[mombild], figa)
    ELSE
      drawpic(bild[mombild], figc)
  END;
END;


PROCEDURE TForm1.vertspiegelnClick(Sender: TObject);      //aktives Element wird vertikal gespiegelt
BEGIN
  IF length(bild[mombild].element) <> 0 THEN
  BEGIN
    bild[mombild].element[bild[mombild].mark].vert := vertspiegeln.Down;
    IF mombild = 1 THEN drawpic(bild[mombild], figa)
    ELSE
      drawpic(bild[mombild], figc)
  END;
END;


PROCEDURE TForm1.loeschenClick(Sender: TObject);        //aktives Element wird aus dem Bild gelöscht
VAR
  i: Byte;
BEGIN
  IF length(bild[mombild].element) <> 0 THEN
  BEGIN

    IF mombild = 1 THEN bildalist.Items.Delete(bild[mombild].mark)
    ELSE                                                    //Element aus der Liste entfernen
      bildclist.Items.Delete(bild[mombild].mark);
    FOR i := bild[mombild].mark + 1 TO length(bild[mombild].element) - 1 DO
      bild[mombild].element[i - 1] := bild[mombild].element[i];//Element aus der Datenbank entfernen
    setlength(bild[mombild].element, length(bild[mombild].element) - 1);

    bild[mombild].mark := 0;

    IF mombild = 1 THEN
    BEGIN
      IF bildalist.Items.Count <> 0 THEN                    //Markierung der Liste verschieben
        FOR i := 1 TO bildalist.Items.Count - 1 DO IF bildalist.Selected[i] THEN bild[mombild].mark
:= i;
      drawpic(bild[mombild], figa);                                           //Bild neu zeichnen
    END ELSE
    BEGIN
      IF bildclist.Items.Count <> 0 THEN                    //Markierung der Liste verschieben
```

```
        FOR i := 1 TO bildclist.Items.Count - 1 DO IF bildclist.Selected[i] THEN bild[mombild].mark
:= i;
      drawpic(bild[mombild], figc);                                    //Bild neu zeichnen
    END;
  END;
END;


PROCEDURE TForm1.bildalistClick(Sender: TObject);          //Element durch Klick auf Liste aktivieren
VAR
  i: Byte;
BEGIN
  mombild := 1;
  FOR i := 0 TO bildalist.Items.Count - 1 DO IF bildalist.Selected[i] THEN bild[1].mark := i;
  drawpic(bild[1], figa);
  horspiegeln.Down := bild[1].element[bild[1].mark].hor;          //Spiegeln Knöpfe für aktualisieren
  vertspiegeln.down := bild[1].element[bild[1].mark].vert;
END;


PROCEDURE TForm1.bildclistClick(Sender: TObject);          //Element durch Klick auf Liste aktivieren
VAR
  i: Byte;
BEGIN
  mombild := 3;
  FOR i := 0 TO bildclist.Items.Count - 1 DO IF bildclist.Selected[i] THEN bild[3].mark := i;
  drawpic(Bild[3], figc);
  horspiegeln.Down := bild[3].element[bild[3].mark].hor;          //Spiegeln Knöpfe für aktualisieren
  vertspiegeln.down := bild[3].element[bild[3].mark].vert;
END;


PROCEDURE TForm1.transformaddClick(Sender: TObject);         //Fürgt Transformation zur Abfolge hinzu
BEGIN
  setlength(transformation, length(transformation) + 1);
  IF transformarten.ActivePageIndex = 0 THEN                                        //Spiegelung
  BEGIN

    WITH transformation[length(transformation) - 1] DO
    BEGIN
      typ := spiegeln;
      hor := transhor.Checked;
      vert := transvert.Checked;
    END;
    transformlist.Items.Add('Spiegeln');

  END ELSE IF transformarten.ActivePageIndex = 1 THEN                                //Rotation
  BEGIN

    WITH transformation[length(transformation) - 1] DO
    BEGIN
      typ := drehen;
      winkel := StrToInt(transwinkel.Text);
    END;
    transformlist.Items.Add('Drehen');
  END ELSE
  BEGIN
    WITH transformation[length(transformation) - 1] DO                     //Hinzufügen Entfernen
    BEGIN
      typ := hinzu;
      loeschen := loesch.Checked;
      art := transart.ItemIndex;
      typennr := StrToInt(transnr.Text);
      x := StrToInt(transx.Text);
      y := StrToInt(transy.Text);
      gr := StrToInt(transgr.Text) / 100;

    END;
    transformlist.Items.Add('Objekt');
  END;
END;


PROCEDURE TForm1.transformkillClick(Sender: TObject);              //Transformation wieder aus der
VAR                                                                          //Abfolge löschen
  i: Byte;
BEGIN

  i := 0;
  IF length(transformation) <> 0 THEN
    IF length(transformation) = 1 THEN
    BEGIN
      setlength(transformation, 0);
      transformlist.Items.Clear;
    END ELSE
    BEGIN
      WHILE Not (transformlist.Selected[i]) DO inc(i);
      transformlist.Items.Delete(i);
      REPEAT
        transformation[i] := transformation[i + 1];
        inc(i);
      UNTIL length(transformation) < i + 2;
      setlength(transformation, length(transformation) - 1);
    END;
```

```
END;


PROCEDURE objedit(nr: Byte; inp: Ttransformation);   //Transformiert einzelne Bilder(eine Anweisung)
VAR
  i, obje: Byte;
  win: Real;
  savey: Integer;
BEGIN

  IF inp.typ = hinzu THEN                                                    //Element hinzufügen
  BEGIN
    obje := 0;
    IF length(bild[nr].element) > 1 THEN
    BEGIN
      IF inp.art = 0 THEN FOR i := 0 TO length(bild[nr].element) - 1 DO          //Größtes Element
          IF bild[nr].element[i].groesse > bild[nr].element[obje].groesse THEN obje := i;
      IF inp.art = 1 THEN FOR i := 0 TO length(bild[nr].element) - 1 DO         //Kleinstes Element
          IF bild[nr].element[i].groesse < bild[nr].element[obje].groesse THEN obje := i;
      IF inp.art = 2 THEN FOR i := 0 TO length(bild[nr].element) - 1 DO          //Element nach Typ
          IF bild[nr].element[i].typ = inp.typennr THEN obje := i;
    END;

    IF inp.loeschen THEN                                                  //Löschen eines Elements
    BEGIN
      IF length(bild[nr].element) < 2 THEN setlength(bild[nr].element, 0)           //Alles löschen
      ELSE
      BEGIN

       IF length(bild[nr].element) - 1 <> obje THEN FOR i := obje TO length(bild[nr].element) - 2 DO
            bild[nr].element[i] := bild[nr].element[i + 1];       //Löschen + aufrücken der Elemente

        setlength(bild[nr].element, length(bild[nr].element) - 1);
      END;
    END ELSE                                                                  //Element hinzufügen
    BEGIN
      setlength(bild[nr].element, length(bild[nr].element) + 1);
      bild[nr].element[length(bild[nr].element) - 1] := bild[nr].element[obje];
      WITH bild[nr].element[length(bild[nr].element) - 1] DO
      BEGIN
        x := x + inp.x;                                          //Grösse und Position ändern
        y := y + inp.y;
        groesse := groesse * inp.gr;
      END;
    END;
  END;

  IF inp.typ = spiegeln THEN                                                  //Spiegelung
    IF length(bild[nr].element) <> 0 THEN
      FOR i := 0 TO length(bild[nr].element) - 1 DO
        WITH bild[nr].element[i] DO
        BEGIN
          IF inp.vert THEN                                                  //an der Vertikalen
          BEGIN
            winkel := (360-winkel) mod objlib[typ].winkel;
            x := -x;                                                        //Position spiegeln
            IF Not (objlib[typ].vert) THEN vert := Not (vert);                //Objekt spiegeln
          END;
          IF inp.hor THEN                                                 //an der Honizontalen
          BEGIN
            winkel := (360-winkel) mod objlib[typ].winkel;
            y := -y;                                                        //Position spiegeln
            IF Not (objlib[typ].hor) THEN hor := Not (hor);                   //Objekt spiegeln
          END;
        END;

  IF inp.typ = drehen THEN                                                    //Drehung
  BEGIN

    win := inp.winkel / 180 * PI;

    IF length(bild[nr].element) > 0 THEN
      FOR i := 0 TO length(bild[nr].element) - 1 DO
        WITH bild[nr].element[i] DO
        BEGIN
          savey := y;                                           //x und y Koordinaten rotieren
          y := round(sin(win) * x + cos(win) * y);
          x := round(cos(win) * x - sin(win) * savey);

          IF (bild[nr].element[i].vert Xor bild[nr].element[i].hor) THEN      //drehung des Objektes
            winkel := (360 - inp.winkel + winkel) Mod objlib[typ].winkel  //Spiegelung wird beachtet
          ELSE
            winkel := (inp.winkel + winkel) Mod objlib[typ].winkel
        END;
  END;
END;


PROCEDURE transformpic(nr: Byte; inp: TarrayTrans);   //Führt objedit für die ganze Transformations-
VAR                                                                           //abfolge durch
  i: Byte;
BEGIN
  IF length(inp) <> 0 THEN
    FOR i := 0 TO length(inp) - 1 DO
    BEGIN
```

```
      objedit(nr, inp[i]);
    END;
END;


FUNCTION edittrans(inp: TarrayTrans): TarrayTrans;        //Verändert die Transformationsanweisungen
VAR
  pos: Byte;
BEGIN

  IF (random(length(inp) + 2) = 0) Or (length(inp) = 0) THEN       //Hinzufügen einer Transformation
  BEGIN                                                  //Tritt weniger häufig auf wenn schon
mehrere
    setlength(inp, length(inp) + 1);                                    //Transformationen vorhanden
    pos := length(inp) - 1;

    CASE random(3) OF                                                  //Entscheidung zwischen
      0:
        BEGIN
          inp[pos].typ := spiegeln;                                                  //Spiegeln
          IF random > 0.5 THEN inp[pos].vert := Not (inp[pos].vert);
          IF random > 0.5 THEN inp[pos].HOR := Not (inp[pos].hor);
        END;
      1:
        BEGIN                                                                        //Drehen
          inp[pos].typ := drehen;
          inp[pos].winkel := (random(3) + 1) * 90;
        END;
      2:
        BEGIN                                                        //oder Hinzufügen bzw. Löschen
          inp[pos].typ := hinzu;
          IF random > 0.5 THEN inp[pos].loeschen := True
          ELSE
            inp[pos].loeschen := False;
          inp[pos].art := random(3);
          inp[pos].typennr := random(length(objlib));
          inp[pos].gr := round(exp((random(7) - 3) / 4) * 5) / 5;

          IF round(inp[pos].gr * 5) / 5 = 1 THEN REPEAT

              inp[pos].x := (random(3) - 1) * 20;
              inp[pos].y := (random(3) - 1) * 20;
            UNTIL (inp[pos].x <> 0) Or (inp[pos].y <> 0);
        END;
    END;

                                                       //schon bestehende Transformationen verändern
  END ELSE
  BEGIN
    pos := random(length(inp) - 1);                                              //eine Auswählen
    IF inp[pos].typ = spiegeln THEN
    BEGIN                                                                      //Spiegelung verändern
      IF random > 0.5 THEN inp[pos].vert := Not (inp[pos].vert);
      IF random > 0.5 THEN inp[pos].HOR := Not (inp[pos].hor);
    END;
    IF inp[pos].typ = drehen THEN                                            //Drehung verändern
    BEGIN
      IF random > 0.5 THEN inp[pos].winkel := inp[pos].winkel + (random(2) + 1) * 90;
      IF random > 0.5 THEN inp[pos].winkel := (360 - inp[pos].winkel) mod 360;
    END;
    IF inp[pos].typ = hinzu THEN                                    //Hinzufügen/Entfernen verändern
    BEGIN
      CASE random(5) OF
        0: inp[pos].loeschen := Not (inp[pos].loeschen);
        1: inp[pos].art := random(3);
        2: inp[pos].typennr := random(length(objlib));
        3: inp[pos].gr := round(exp((random(7) - 3) / 5) * 5) / 5;
        4: IF random > 0.5 THEN inp[pos].x := -inp[pos].x
          ELSE
            inp[pos].y := -inp[pos].y;
      END;
    END;
  END;
  edittrans := inp;                                                      //Veränderungen weitergeben
END;


FUNCTION comparebild(nr1, nr2: Byte; ohne: Boolean): Boolean;                 //Vergleicht 2 Bilder
VAR
  i, k, falsche: integer;
  falsch: Boolean;
BEGIN
  IF (length(bild[nr1].element) <> length(bild[nr2].element)) And not(ohne) THEN
    comparebild := False                  //Wenn ohne true dann muss die anzahl nicht übereinstimmen
  ELSE
  BEGIN
    falsche := 0;
    IF length(bild[nr1].element) <> 0 THEN                          //Alle Objektparameter vergleichen
      FOR i := 0 to length(bild[nr1].element) - 1 DO
      BEGIN
        falsch := false;
        k := -1;
        REPEAT
          inc(k);
          if k > length(bild[nr2].element) - 1 THEN
```

```
          BEGIN
            falsch := true;
            inc(falsche);
            k := 0;
          END;
        UNTIL falsch or (
          (round(bild[nr1].element[i].groesse * 10) = round(bild[nr2].element[k].groesse * 10)) AND
          (bild[nr1].element[i].typ = bild[nr2].element[k].typ) AND
          (round(bild[nr1].element[i].x / 3) = round(bild[nr2].element[k].x / 3)) AND
          (round(bild[nr1].element[i].y / 3) = round(bild[nr2].element[k].y / 3)))
          AND ((
          (bild[nr1].element[i].winkel = bild[nr2].element[k].winkel) AND
          (bild[nr1].element[i].hor = bild[nr2].element[k].hor) AND
          (bild[nr1].element[i].vert = bild[nr2].element[k].vert))
          or (
          (objlib[bild[nr1].element[i].typ].hor xor objlib[bild[nr1].element[i].typ].vert) AND
          (bild[nr1].element[i].winkel+round(objlib[bild[nr1].element[i].typ].winkel/2)
            mod objlib[bild[nr1].element[i].typ].winkel = bild[nr2].element[k].winkel) AND
          not((bild[nr1].element[i].hor = bild[nr2].element[k].hor) AND
          (bild[nr1].element[i].vert = bild[nr2].element[k].vert))));
      END;
   if falsche <= abs(length(bild[nr2].element)-length(bild[nr1].element)) then comparebild := true
   ELSE
     comparebild := false;
  END;

END;


PROCEDURE bildcopy(von, auf: Byte);         //Überträgt ein Bild auf ein anderes Element für Element
VAR
  i: Byte;
BEGIN
  setlength(bild[auf].element, 0);
  IF length(bild[von].element) <> 0 THEN
  BEGIN
    setlength(bild[auf].element, length(bild[von].element));
    FOR i := 0 TO length(bild[auf].element) - 1 DO bild[auf].element[i] := bild[von].element[i];
  END;
END;


PROCEDURE TForm1.berechnenClick(Sender: TObject);                        //Generiert Bild B und 1-5
VAR
  i, k, richtig, antiendless: Byte;
  nochmal: Boolean;
BEGIN

  richtig := random(5) + 1;                                          //100% richtiges Bild auswählen
  bildcopy(1, 2);                                                           //Bild A auf B kopieren
  transformpic(2, transformation);                                          //Bild B transformieren
  bildcopy(3, richtig + 3);                                                 //Bild C auf ? kopieren
  transformpic(richtig + 3, transformation);                                //Bild ? transformieren

  FOR i := 1 TO 5 DO IF i <> richtig THEN                        //Schleife für die restlichen Bilder
    BEGIN
      setlength(fake, length(transformation));                   //fake Transformation der originalen
      IF length(transformation) <> 0 THEN                                              //angleichen
        FOR k := 0 TO length(transformation) - 1 DO fake[k] := transformation[k];

      antiendless := 0;
      REPEAT
        nochmal := True;
        inc(antiendless);
        bildcopy(3, i + 3);                                                 //Bild C auf i Kopieren
        transformpic(i + 3, edittrans(fake));               //fake verändern und Bild transformieren
        IF i > 1 THEN                                                  //Testen ob 2 Bilder gleich sind
          FOR k := 1 TO i - 1 DO IF comparebild(k + 3,i + 3, False) THEN nochmal := False;
        IF richtig > i THEN IF comparebild(richtig + 3,i + 3, False) THEN nochmal := False;

      UNTIL nochmal Or (antiendless > 254);  //Solange durchführen bis ein neues Bild entstanden ist

    END;

  drawpic(bild[2], figb);                                                          //Bilder zeichen
  FOR i := 1 TO 5 DO drawpic(bild[i + 3], fig[i]);
END;


PROCEDURE TForm1.exportClick(Sender: TObject);                             //HTML Datei ausgeben
VAR
  i: Byte;
  verz, dateiname: String;
  f: textfile;
BEGIN
                                                                 //Frage nach Verzeichnis und Name
  verz := inputbox('Verzeichnis Auswahl', 'Verzeichnis für HTML Ausgabe eingeben:',
    ExtractFilePath(Application.EXEName) + 'HTML\');
  dateiname := inputbox('Dateiname Auswahl', 'Dateiname für HTML Ausgabe eingeben:', '');
  {$I-}
  mkdir(verz);
  {$I+}
  IOresult;
  assignfile(f, verz + dateiname + '.htm');
  rewrite(f);                                                                 //HTML Text ausgeben
```

```
  Write(f, '<html><head><title>HTML Ausgabe von Ausgabe3 (Krückl Viktor)</title>');
  Write(f, '<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">');
  Write(f, '</head><body bgcolor="#FFFFFF"><p>Figur A <img src="' + dateiname +
    '1.bmp" width="99" height="99"> verhält sich zu Figur   B <img src="' +
    dateiname + '2.bmp" width="99" height="99">');
  Write(f, ' wie Figur C <img src="' + dateiname +
    '3.bmp" width="99" height="99">   zu:</p><p>1.<img src="' + dateiname +
    '4.bmp" width="99" height="99"> 2.<img src="' + dateiname + '5.bmp" width="99" height="99">');
  Write(f, ' 3.<img src="' + dateiname + '6.bmp" width="99" height="99"> 4.<img src="' +
    dateiname + '7.bmp" width="99" height="99">   5.<img src="' + dateiname +
    '8.bmp" width="99" height="99"></p></body></html>');
  closefile(f);
  i := 1;
                                                                          //Bilder abspeichern
  WITH figa DO
  BEGIN
    Picture.Bitmap.PixelFormat := pf1bit;
    Picture.Bitmap.SaveToFile(verz + dateiname + IntToStr(i) + '.bmp');
    Picture.Bitmap.PixelFormat := pf24bit;
  END;
  inc(i);

  WITH figb DO
  BEGIN
    Picture.Bitmap.PixelFormat := pf1bit;
    Picture.Bitmap.SaveToFile(verz + dateiname + IntToStr(i) + '.bmp');
    Picture.Bitmap.PixelFormat := pf24bit;
  END;
  inc(i);

  WITH figc DO
  BEGIN
    Picture.Bitmap.PixelFormat := pf1bit;
    Picture.Bitmap.SaveToFile(verz + dateiname + IntToStr(i) + '.bmp');
    Picture.Bitmap.PixelFormat := pf24bit;
  END;

  FOR i := 4 TO 8 DO
    WITH fig[i - 3] DO
    BEGIN
      Picture.Bitmap.PixelFormat := pf1bit;
      Picture.Bitmap.SaveToFile(verz + dateiname + IntToStr(i) + '.bmp');
      Picture.Bitmap.PixelFormat := pf24bit;
    END;
END;


FUNCTION ggt(a, b: Word): Word;                                  //Bildet den GGT zweier Zahlen
VAR
  d: Word;
BEGIN
  REPEAT
    IF b <> 0 THEN d := a Mod b;
    a := b;
    b := d;
  UNTIL b = 0;
  ggt := a;
END;


FUNCTION kgV(a, b: Word): Word;                                  //Bildet den kgV zweier Zahlen
BEGIN
  kgV := round(a * b / ggt(a, b));
END;


FUNCTION winkelmoeg(w1, w2, mod1, mod2: Word): Word;//Testet ob 2 Winkel gleich sind unter Beachtung
VAR                                          //das sich die Objekte nach mod Grad wieder entsprechen
  x, y: Word;
BEGIN
  x := 0;
  y := 0;
  IF (w1 + x * mod1 <> w2 + y * mod2) THEN REPEAT
      IF w1 + x * mod1 > w2 + y * mod2 THEN inc(y)
      ELSE
        inc(x);
    UNTIL (w1 + x * mod1 = w2 + y * mod2) Or (x * mod1 >= 360) Or (y * mod1 >= 360);
  IF (x * mod1 >= 360) Or (y * mod1 >= 360) THEN winkelmoeg := 360
  ELSE                                                 //Nicht möglich --> 360Grad wird ausgegeben
    winkelmoeg := w1 + x * mod1;                              //Möglich --> resultierender Winkel
END;


FUNCTION bruteatack(inp, len: Byte): Boolean;            //Bruteforce Attack auf eine Transformation
VAR
  i, mom: Integer;
  weitermachen: Boolean;
BEGIN
  weitermachen := True;
  bruteatack := False;
  IF len < 3 THEN                                                          //maximal 2 Tief
  BEGIN
    bildcopy(1, 9);                          //Test ob fake Transformation für Bild 1 und 2 zutrifft
    transformpic(9, fake);                                          //dazu wird Bild 9 erstellt
    IF comparebild(2,9, True) THEN
```

```
    BEGIN
      bildcopy(3, 9);                          //Test ob fake Transformation für Bild 3 und inp zutrifft
      transformpic(9, fake);
      IF comparebild(inp, 9, True) THEN
      BEGIN
        bruteatack := True;                          //Wenn beide stimmen dann wird true ausgegeben und
        weitermachen := False;                                  //keine weitere Berechnung angestellt
      END;
    END;
    IF weitermachen THEN                                  //Noch keine richtige Transformation gefunden
    BEGIN
      setlength(fake, length(fake) + 1);                      //Fake um eine Transformation erweitern
      mom := length(fake) - 1;
      fake[mom].typ := drehen;                                                          //Drehen testen
      i := 0;
      REPEAT                                              //für alle Grob eingeschränkten Winkel testen
        fake[mom].winkel := winkel + i;
        IF bruteatack(inp, len + 1) THEN
        BEGIN                                    //Wenn richtiges Ergebnis eine Stufe höher weitergeben
          bruteatack := True;
          i := 2000;
        END;
        inc(i, symwink);
      UNTIL i > 360;
      IF i <= 2000 THEN
      BEGIN                                                                  //Spiegelungen testen
        fake[mom].typ := spiegeln;
        fake[mom].hor := True;
        fake[mom].vert := False;                                                          //hor spiegeln
        IF bruteatack(inp, len + 1) THEN bruteatack := True
        ELSE
        BEGIN
          fake[mom].vert := True;                                                  //hor+vert spiegeln
          IF bruteatack(inp, len + 1) THEN bruteatack := True
          ELSE
          BEGIN
            fake[mom].hor := False;                                                       //vert spiegeln
            bruteatack := bruteatack(inp, len + 1);
          END;
        END;
      END;
      setlength(fake, mom);                                             //fake wieder verkleinern
    END;
  END;
END;

FUNCTION richtigesergebnis(inp: Byte): Boolean;                 //ermittelt ob Bild richtig ist mit
VAR                                                                                   //brute force
  i: Byte;
  objgleich: Boolean;
  dummyw1, dummyw2: Integer;
  scale: Real;
BEGIN

  IF length(bild[1].element) - length(bild[2].element) = length(bild[3].element) -
    length(bild[inp].element) THEN                                  //Veräderung der Objektanzahl gleich
  BEGIN
    objgleich := True;

    IF length(bild[1].element) < length(bild[2].element) THEN            //Test ob die Objektgrössen
    BEGIN                                                                 //proportional sind
      scale := bild[1].element[0].groesse / bild[3].element[0].groesse;
      FOR i := 0 TO length(bild[2].element) - length(bild[1].element) - 1 DO
      BEGIN
        IF round(scale * 50) <> round(bild[2].element[length(bild[1].element) + i].groesse /
          bild[inp].element[length(bild[3].element) + i].groesse * 50) THEN
          objgleich := False;
      END;
    END;

    IF min(min(length(bild[1].element), length(bild[2].element)),
      min(length(bild[3].element), length(bild[inp].element))) <> 0 THEN
      FOR i := 0 TO min(min(length(bild[1].element), length(bild[2].element)),
        min(length(bild[3].element), length(bild[inp].element))) - 1 DO
      BEGIN
        IF (bild[1].element[i].typ = bild[2].element[i].typ) And
          (bild[3].element[i].typ <> bild[inp].element[i].typ) THEN objgleich := False;
      END;

    IF objgleich THEN
    BEGIN
                                  //Ermitteln der möglichen Roatationswinkel aus den Objektdrehungen
      winkel := 0;
      symwink := 1;
      FOR i := 0 TO min(length(bild[1].element), length(bild[2].element)) - 1 DO
      BEGIN
        IF objlib[bild[1].element[i].typ].winkel <> 1 THEN
        BEGIN
          IF bild[2].element[i].hor Xor bild[2].element[i].vert THEN
            dummyw1 := objlib[bild[2].element[i].typ].winkel - bild[2].element[i].winkel
          ELSE
            dummyw1 := bild[2].element[i].winkel;
          IF bild[1].element[i].hor Xor bild[1].element[i].vert THEN
            dummyw2 := objlib[bild[1].element[i].typ].winkel - bild[1].element[i].winkel
          ELSE
```

```
          dummyw2 := bild[1].element[i].winkel;
        winkel := winkelmoeg(winkel, dummyw1 - dummyw2, symwink,
          objlib[bild[1].element[i].typ].winkel);
        IF symwink = 1 THEN   symwink := objlib[bild[1].element[i].typ].winkel
        ELSE
          symwink := kgV(symwink, objlib[bild[1].element[i].typ].winkel);
      END;
    END;


    if symwink>1 then symwink := symwink div 2;
    winkel := winkel Mod symwink;

    setlength(fake, 0);
    richtigesergebnis := bruteatack(inp, 0);                          //Bruteforce Suche starten

  END ELSE
    richtigesergebnis := False;
END ELSE
  richtigesergebnis := False;
END;

FUNCTION richtigesergebnis2(inp: Byte): Boolean;                  //Test ob richtiges Ergebnis ohne
VAR                                                                       //brute force
  i: Byte;
  objgleich: Boolean;
  dummyw1, dummyw2: Integer;
  rot1, rot2, winkel1, symwink1, winkel2, symwink2: Integer;
  hor1, hor2, vert1, vert2, hora1, hora2, verta1, verta2: Shortint;
  hors1, hors2, verts1, verts2: Boolean;
  scale: Real;
BEGIN

  IF length(bild[1].element) - length(bild[2].element) = length(bild[3].element) -
    length(bild[inp].element) THEN               //Stimmt die Abänderung der Elementanzahl überein?
  BEGIN
    objgleich := True;

    IF length(bild[1].element) < length(bild[2].element) THEN
    BEGIN                                                 //Grössenvergleich der einzelnen Elemente
      scale := bild[1].element[0].groesse / bild[3].element[0].groesse;
      FOR i := 0 TO length(bild[2].element) - length(bild[1].element) - 1 DO
      BEGIN
        IF round(scale * 50) <> round(bild[2].element[length(bild[1].element) + i].groesse /
          bild[inp].element[length(bild[3].element) + i].groesse * 50) THEN
          objgleich := False;
      END;
    END;

    IF min(min(length(bild[1].element), length(bild[2].element)),
      min(length(bild[3].element), length(bild[inp].element))) <> 0 THEN
      FOR i := 0 TO min(min(length(bild[1].element), length(bild[2].element)),
        min(length(bild[3].element), length(bild[inp].element))) - 1 DO
      BEGIN
        IF (bild[1].element[i].typ = bild[2].element[i].typ) And
          (bild[3].element[i].typ <> bild[inp].element[i].typ) THEN objgleich := False;
      END;

    IF objgleich THEN
    BEGIN

      hors2 := False;                                   //Ermittlung der Spiegelung an den Elementen
      verts2 := False;
      FOR i := 0 TO min(length(bild[3].element), length(bild[inp].element)) - 1 DO
      BEGIN
        IF Not (objlib[bild[3].element[i].typ].hor) THEN
        BEGIN
          IF Not (bild[3].element[i].hor Xor bild[inp].element[i].hor) THEN hor2 := 1
          ELSE
            hor2 := -1;
          hors2 := True;
        END;
        IF Not (objlib[bild[3].element[i].typ].vert) THEN
        BEGIN
          IF Not (bild[3].element[i].vert Xor bild[inp].element[i].vert) THEN vert2 := 1
          ELSE
            vert2 := -1;
          verts2 := True;
        END;
      END;

      hors1 := False;
      verts1 := False;
      FOR i := 0 TO min(length(bild[1].element), length(bild[2].element)) - 1 DO
      BEGIN
        IF Not (objlib[bild[1].element[i].typ].hor) THEN
        BEGIN
          IF Not (bild[1].element[i].hor Xor bild[2].element[i].hor) THEN hor1 := 1
          ELSE
            hor1 := -1;
          hors1 := True;
        END;
        IF Not (objlib[bild[1].element[i].typ].vert) THEN
        BEGIN
          IF Not (bild[1].element[i].vert Xor bild[2].element[i].vert) THEN vert1 := 1
```

```
    ELSE
      vert1 := -1;
    verts1 := True;
  END;
END;
                                        //Ermittlung der Spiegelung an den Koordinatenveränderungen

hora1 := 0;
verta1 := 0;
FOR i := 0 TO min(length(bild[1].element), length(bild[2].element)) - 1 DO
  IF (abs(bild[1].element[i].x) = abs(bild[2].element[i].x)) And
    (abs(bild[1].element[i].y) = abs(bild[2].element[i].y)) THEN
  BEGIN

    IF (bild[2].element[i].x <> 0) THEN
    BEGIN
      verta1 := bild[1].element[i].x Div bild[2].element[i].x;
    END;
    IF (bild[2].element[i].y <> 0) THEN
    BEGIN
      hora1 := bild[1].element[i].y Div bild[2].element[i].y;
    END;
  END;

hora2 := 0;
verta2 := 0;
FOR i := 0 TO min(length(bild[3].element), length(bild[inp].element)) - 1 DO
  IF (abs(bild[3].element[i].x) = abs(bild[inp].element[i].x)) And
    (abs(bild[3].element[i].y) = abs(bild[inp].element[i].y)) THEN
  BEGIN
    IF bild[inp].element[i].x <> 0 THEN
    BEGIN
      verta2 := bild[3].element[i].x Div bild[inp].element[i].x;
    END;
    IF bild[inp].element[i].y <> 0 THEN
    BEGIN
      hora2 := bild[3].element[i].y Div bild[inp].element[i].y;
    END;
  END;

winkel1 := 0;                                                  //Bilddrehung Anhand den Elementen
symwink1 := 1;
FOR i := 0 TO min(length(bild[1].element), length(bild[2].element)) - 1 DO
BEGIN
  IF bild[2].element[i].hor Xor bild[2].element[i].vert THEN
    dummyw1 := objlib[bild[2].element[i].typ].winkel - bild[2].element[i].winkel
  ELSE
    dummyw1 := bild[2].element[i].winkel;
  IF bild[1].element[i].hor Xor bild[1].element[i].vert THEN
    dummyw2 := objlib[bild[1].element[i].typ].winkel - bild[1].element[i].winkel
  ELSE
    dummyw2 := bild[1].element[i].winkel;
  winkel1 := winkelmoeg(winkel1, dummyw1 - dummyw2, symwink1,
    objlib[bild[1].element[i].typ].winkel);
  IF symwink1 = 1 THEN   symwink1 := objlib[bild[1].element[i].typ].winkel
  ELSE
    symwink1 := kgV(symwink1, objlib[bild[1].element[i].typ].winkel);
END;

winkel2 := 0;
symwink2 := 1;
FOR i := 0 TO min(length(bild[3].element), length(bild[inp].element)) - 1 DO
BEGIN
  IF bild[inp].element[i].hor Xor bild[inp].element[i].vert THEN
    dummyw1 := objlib[bild[inp].element[i].typ].winkel - bild[inp].element[i].winkel
  ELSE
    dummyw1 := bild[inp].element[i].winkel;
  IF bild[3].element[i].hor Xor bild[3].element[i].vert THEN
    dummyw2 := objlib[bild[3].element[i].typ].winkel - bild[3].element[i].winkel
  ELSE
    dummyw2 := bild[3].element[i].winkel;
  winkel2 := winkelmoeg(winkel2, dummyw1 - dummyw2, symwink2,
    objlib[bild[3].element[i].typ].winkel);
  if symwink2 = 1 then symwink2 := objlib[bild[3].element[i].typ].winkel
  ELSE
    symwink2 := kgV(symwink2, objlib[bild[3].element[i].typ].winkel);
  winkel2 := winkel2 Mod symwink2;
END;

rot1 := 360;                                                 //Bilddrehung anhand der Objektrotation
FOR i := 0 TO min(length(bild[1].element), length(bild[2].element)) - 1 DO
BEGIN
  IF (abs(bild[1].element[i].x) > 10) Or (abs(bild[1].element[i].y) > 10) THEN
  BEGIN
    rot1 := round((arctan2(bild[2].element[i].x, bild[2].element[i].y) -
      arctan2(bild[1].element[i].x, bild[1].element[i].y)) / PI * 180);
  END;
END;
IF rot1 < 0 THEN inc(rot1, 360);
IF rot1 > 360 THEN dec(rot1, 360);
rot2 := 360;
FOR i := 0 TO min(length(bild[3].element), length(bild[inp].element)) - 1 DO
BEGIN
  IF (abs(bild[3].element[i].x) > 10) Or (abs(bild[3].element[i].y) > 10) THEN
  BEGIN
```

```
          rot2 := round((arctan2(bild[inp].element[i].x, bild[inp].element[i].y) -
            arctan2(bild[3].element[i].x, bild[3].element[i].y)) / PI * 180);
        END;
      END;
      IF rot2 < 0 THEN inc(rot2, 360);
      IF rot2 > 360 THEN dec(rot2, 360);


      rot1 := round(rot1 / 15) * 15;                                    //Ergebnisse vergröbern
      rot2 := round(rot2 / 15) * 15;


      winkel1 := winkel1 Mod symwink1;
      winkel2 := winkel2 Mod symwink2;

      richtigesergebnis2 := False;


      IF (hora1 * hora2<>-1) And (verta1 * verta2<>-1) THEN            //Die entstandenen Parameter
        IF winkelmoeg(winkel1, winkel2, symwink1, symwink2) < 360 THEN     //miteinander vergleichen
        BEGIN
          IF Not ((hors1 And hors2) Or (verts1 And verts2)) THEN
          BEGIN
            if ((hora1=0) or (verta1=0)) then
              if (winkelmoeg(rot1, rot2, 180,180) < 360) then richtigesergebnis2 := True;
            if ((hora1=0) and (verta1=0)) then richtigesergebnis2 := True;
          END ELSE IF hors1 And hors2 And verts1 And verts2 THEN
          BEGIN
            IF (hor1 = hor2) And (vert1 = vert2) THEN richtigesergebnis2 := True
          END ELSE IF (Not (hors1 And hors2) And (vert1 = vert2)) Or
            (Not (verts1 And verts2) And (hor1 = hor2)) THEN
            richtigesergebnis2 := True;
        END;

    END ELSE
      richtigesergebnis2 := False;
  END ELSE
    richtigesergebnis2 := False;
END;


PROCEDURE TForm1.zurueckrechnenClick(Sender: TObject);        //Startet die Suche nach den richtigen
VAR                                                          //Ergebnissen mit oder ohne brute force
  i: Byte;
BEGIN
  FOR i := 1 TO 5 DO
  BEGIN
    IF ((Sender = zurueckrechnen) And richtigesergebnis(i + 3)) Or
      ((Sender = zurueckrechnen2) And richtigesergebnis2(i + 3)) THEN
      imagelist1.Draw(fig[i].Canvas, 1,1, 0)                                          //Richtig
    ELSE
      imagelist1.Draw(fig[i].Canvas, 1,1, 1);                                          //Falsch
    fig[i].Refresh;
  END;
END;

END.
```

```
                        _____________________________
                       | (c) 2002 by Krueckl Viktor GmbR |
                       |=================================|
                       |  Gesellschaft mit beschränkter  |
                       |_________Rechtschreibung_________|
```