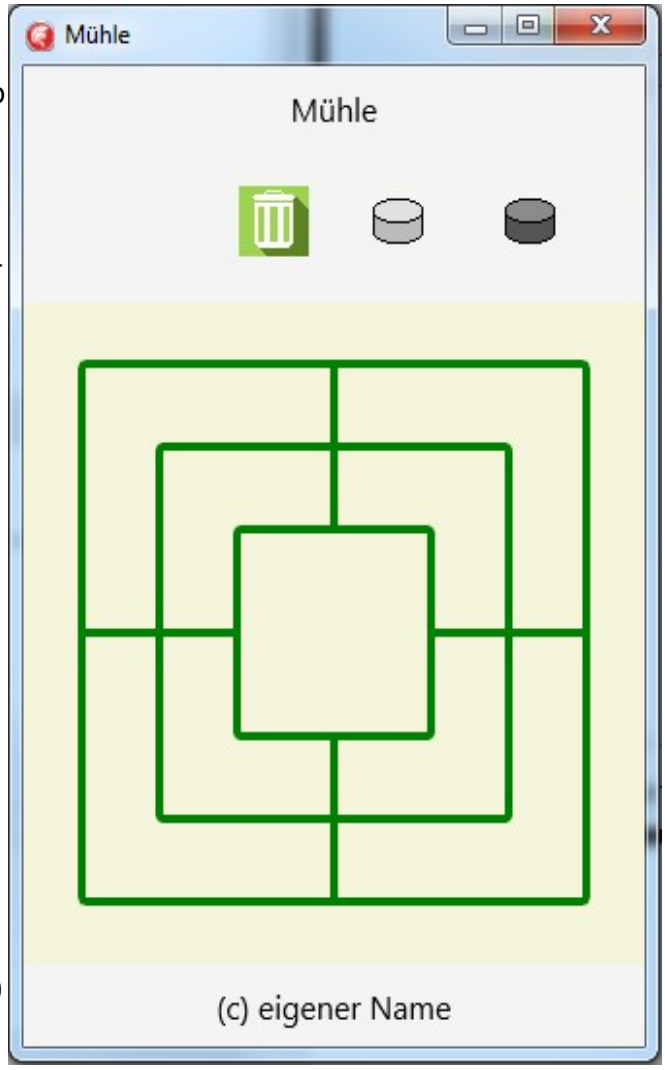
A
1. Öffne ein neues Projekt in Delphi XE7. Datei-> neu-> geräteübergreifende Anwendung.
2. Wähle “Leere Anwendung“. Ein leeres Fenster erscheint. Links unten im Objektinspektor siehst du einige Eigenschaften des Fensters und kannst sie ändern. Wähle dort unter Height und Width die geignete Höhe und Breite deines Formularfensters. Lege aus der Tool-Palette rechts unten folgende Objekte auf dem Formular ab:
3. Zwei Toolbar-Objekte. Setze bei der unteren Toolbar „align“ auf „bottom“. Lege auf diesen Toolbar-Objekten jeweils ein Label-Objekt ab und beschrifte es unter Text im Objektinspektor mit „Mühle“ und deinem Copyright-Vermerk. Setze bei beiden Label-Objekten „align“ auf „client“ und „TextSettings->HorzAlign“ auf „center“.



4. Ein Image-Objekt. Setze „align“ auf „bottom“ und vergrößere die Höhe des Objekts durch ziehen, bis dass das Image-Objekt quadratisch ist. Damit nun ein schönes Mühlefeld entsteht, mache eine Bildschirmkopie des rechts abgebildeten Mühlebretts, ließ es in den kostenlosen IDA-Bildeditor (oder ein anderes Zeichenprogramm) ein, speichere es als png-Datei ab und wähle dieses Bild für Image1 im Objektinspektor als MultiressourceBitmap – fertig!
5. Als globale Variablen werden unter der Deklaration von Form1 definiert:

```
RelPos      :TPointF;
DragImage   :tImage;
```

6. Zunächst ein Image-Objekt der Größe 50x50 für einen weißen Stein. Bild selbst kann im IDA-Bildeditor entworfen werden und dann als MultiressourceBitmap in das Image-Objekt eingelesen werden. Dabei sollte vor dem Einlesen als „transparente Farbe“ „weiß“ gewählt werden, damit der Rand des Spielsteins transparent bleibt.
7. Im „OnMouseDown“- Ereignis wird nun eingegeben

```
DragImage:= Sender as tImage;
DragImage.BringToFront;
RelPos:=PointF(X,Y);
```

Dadurch wird das zu ziehende Objekt in DragImage gespeichert und ganz nach vorne gebracht. Dann werden die Mauszeiger-Korrekturwerte gesetzt durch `RelPos:=PointF(X,Y);` Sie enthalten, an welchem Aufhängepunkt man mit der Maus den Stein bewegt.

8. Im „OnMouseMove“-Ereignis wird eingegeben

```
if (Sender as tImage)=DragImage then
   if ssLeft in Shift then  // Linke Maustaste
      with DragImage do begin
        Position.X:= Position.X + X - RelPos.X;
        Position.Y:= Position.Y + Y - RelPos.Y;
      end
   else DragImage:= nil
else DragImage:= nil
```

9. und im „OnMouseUp“-Ereignis wird zunächst eingegeben `DragImage:= nil`.
   Somit gibt es kein „DragImage“ mehr, das gezogen werden kann.
   Damit nun die Steine in einem Papierkorb verschwinden können, muss dieses „OnMouseUp“-Ereignis noch ergänzt werden. Somit ergibt sich insgesamt::

```
DragImage:= nil;
with (Sender as tImage) do begin
   if (Position.X>=Papierkorb.Position.X - Papierkorb.Width/3) and
      (Position.X<=Papierkorb.Position.X + Papierkorb.Width/3) and
      (Position.Y>=Papierkorb.Position.Y - Papierkorb.Height/3)and
      (Position.Y<=Papierkorb.Position.Y + Papierkorb.Height/3)
   then visible:= false;
 end;
```

10. Das Image-Objekt dieses einzigen weißen Steins wird nun kopiert und zunächst achtmal eingefügt. Dabei werden auch alle Ereignis-Methoden kopiert. Dann wird es ein weiteres Mal eingefügt. Um nun daraus das Image-Objekt für den ersten schwarzen Stein zu schaffen, muss lediglich bei MultiressourceBitmap als Bild ein schwarzer Stein eingefügt werden. Nun wird dieses Image-Objekt erneut kopiert und achtmal eingefügt. Die OnMouse-Ereignisse werden dabei automatisch in alle Image-Objekte, die als Steine vorgesehen sind, kopiert. Nun ist alles vorhanden, um Mühle zu spielen.
11. Ergänzung: Wer die Spielsteine am Anfang lose am Rand des Brettes verteilen möchte, kann noch in der OnShow-Methode des Formulars eingeben:

```
procedure TForm1.FormShow(Sender: TObject);
var i:integer;
begin
   for i:= 1 to 9 do
      with tImage(FindComponent('Image'+IntToStr(i+1))) do begin
         Position.X:= 80 + random(60);
         Position.y:= 30 + random(60);
      end;
   for i:= 1 to 9 do
      with tImage(FindComponent('Image'+IntToStr(10+i))) do begin
         Position.X:= 200 + random(60);
         Position.y:=  30 + random(60);
      end;
end;
```

12. Will man das Mühebrett selbst zeichnen, kommen die Zeichen-Prozeduren von Delphi zum Einsatz. Dazu gibt man alternativ in der „OnShow“-Methode ein:

```
procedure TForm1.FormShow(Sender: Tobject);
var P1,P2: TPointF;  MyRect: TRectF;  i:integer;
begin
  Rand:= 30; Abstand:= 40;
  with Image1 do Bitmap:= tBitmap.Create(trunc(Width),trunc(Height));
  with Image1.Bitmap.Canvas do begin
    Stroke.Color:= tAlphacolorRec.Green;
    BeginScene;
      Clear(tAlphacolorRec.Beige);
      StrokeThickness:= 4;
      for i:=0 to 2 do begin
         MyRect:= RectF(Rand+i*Abstand,Rand+i*Abstand,Width-Rand-
                     i*Abstand,Width-Rand-i*Abstand);
         DrawRect(MyRect, 2, 2, AllCorners, 1.0);
      end;
      P1.X:= Width/2;P1.Y:= Rand;
      P2.X:= P1.X;P2.Y:= Rand+2*Abstand;
      DrawLine(P1,P2,100);
      P1.X:= Width/2;P1.Y:= Width-Rand-2*Abstand;
      P2.X:= P1.X;P2.Y:= Width-Rand;
      DrawLine(P1,P2,100);
      P1.X:= Rand;P1.Y:= Width/2;
      P2.X:= Rand+2*Abstand;P2.Y:= P1.Y;
      DrawLine(P1,P2,100);
      P1.X:= Width-Rand-2*Abstand;P1.Y:= Width/2;
      P2.X:= ??? (Überlegen!) ;P2.Y:= P1.Y;
      DrawLine(P1,P2,100);
     EndScene;
   end;
end;
```

„Rand“ und „Abstand“ werden dazu als integer-Variablen unter der Form1 deklariert.