➲ AUFGABE 2: WO BIN ICH?

Lösungsidee und Programmdokumentation:

Die Wände des Labyrinths, durch das sich Ludwig bewegen soll, spalte ich in horizontale und vertikale Wände auf. Für horizontale Linien gilt: Wenn das Labyrinth 15*15 Felder groß sein soll, brauche ich 16 verschiedene x-Koordinaten und 15 y-Koordinaten. Das Feld (1, 1) wird dann von den Wänden `hor[0,0], hor[1,0], vert[0,0]` und `vert[0,1]` umschlossen. Hinzu kommen noch die Koordinaten des Käse und Ludwigs Startkoordinaten. Daraus Resultiert mein ersten zwei Typendefinitionen:

```
type kor = record                                          //x und y Koordinaten
       x, y: shortint end;
     lab = record                                  //Record mit allen Labyrinthparametern
       vert: array[0..15, 0..14] of boolean;                        //Vertikalwände
       hor:  array[0..14, 0..15] of boolean;                      //Horizontalwände
       kaese, start : kor;                                 //Käse- und Startposition
       end;
```

Eine Variable `hirn` vom Typ `lab` stellt im ganzen Programm immer Ludwigs Grundwissen dar.
Damit man deutlich sehen kann, dass Ludwig zu keinem Zeitpunkt auf `hirn.start` zugreift, verwende ich die Function `mausschaut`. Sie benötigt Ludwigs Abweichung vom Startpunkt und gibt dafür die Wanddaten für das Feld auf dem sich Ludwig wirklich gerade befindet aus. Dafür habe ich eine 3 Typendefinition verwendet:

```
feld = record                                  //Welche Wand dieses Feldes sind vorhanden
  o, u, r, l : boolean;
  end;
```

Ludwigs Positionsfindung durch die Procedure `zurechtfinden` funktioniert dann so. Wenn Ludwig das Labyrinth noch nicht gesehen hat, könnten alle Felder des Labyrinths Startpunkt sein. Also werden in dem `moeglich` ARRAY alle Feldkoordinaten gespeichert, bis auf die Position, welche mit Käse belegt ist, da das Ludwig durch die Function Kaesefeld erfährt. Da Ludwig seine echte Position nicht kennt arbeitet er mit Variable `delta`, die seine Abweichung vom Startpunkt beinhaltet. Ludwig holt sich mit der function `mausschaut(delta)` die aktuellen Wanddaten und speichert sie in `diesesfeld`. Dann nimmt sich Ludwig jede Feldkoordinate aus dem `moeglich` array, addiert `delta` hinzu holt sich mit der `feldart` function die Wanddaten und vergleicht sie mit `diesesfeld`. Wenn sich die zwei Variablen unterscheiden, wirft Ludwig die zugehörigen Koordinaten aus dem moeglich ARRAY heraus. Wenn die function `Kaesefeld` true wird, hat Ludwig den Käse gefunden und der moeglich ARRAY und Ludwig rechnet aus der Käseposition und dem momentanen delta auf die Startposition zurück.
Danach muss sich Ludwig entscheiden auf welches Feld er weitergeht. Er nimmt alle offenen Richtungen in den `offen` array auf. Ein Schritt Nach Oben entspricht 0, nach unten 1, nach rechts 2 und nach links 3. Der ARRAY `schritte` besitzt dann für diese Werte das passende Koordinatendelta.
Dann nimmt Ludwig per Zufall eine Richtung aus dem Array und speichert sie in der `nextstep` Variable. Wenn die „Intelligenz“ Schaltfläche aktiviert ist und sich Ludwig zwischen mehreren Richtungen entscheiden kann, nimmt er sich die Richtung aus dem `offen` Array bei dem die möglichen Felder sich am stärksten unterscheiden. Er geht dabei so vor:
In einer Schleife für alle Möglichen Richtungen befinden sich zwei Schleifen die alle Indizes-Paare für den `moeglich` Array durchlaufen. Unterscheiden sich die Felder die von den `moeglich + schritte` Koordinaten beschrieben werden, wird für diesen Schrittrichtung der auswahl couter um eins erhöht. Wenn dieser Vergleich für alle Felder abgeschlossen ist, vergleicht er die entstandenen Summen im `auswahl` array. Die Richtung mit dem höchsten zugehörigen `auswahl` Wert wird in der `nextstep` variable gespeichert.
Damit sich Ludwig nicht Gänge mehrmals anschaut wird für jedes `delta` ein boolean Wert im `bekannt` array gespeicht. Wenn das `delta` das sich aus `nextstep` ergibt einen true `bekannt`-Wert hat, nimmt sich Ludwig aus den anderen Schritten im offen array eine Alternative und führt den selben Test durch. Wenn keine Schrittrichtung im offen array vorhanden ist, die Ludwig auf ein noch nicht besuchtes Feld führt behält er die schon vorher ausgewählte Schrittrichtung bei.
Darauf berechnet er das neue `delta` und setzt den boolean Wert aus dem bekannt array für dieses Feld auf true.
Diese Prozeduren führt Ludwig so lange durch, bis er nur noch eine Koordinate im moeglich array stehen hat.

Nachteilig hat sich in meinem Programm die Programmiersprache Pascal verhalten. Ich hätte so gerne Koordinaten Paare addiert oder Variablem vom Typ Feld einfach verglichen. Das geht aber nicht. Darum hab ich die function `vergleiche` geschrieben, die zwei variablen vom Typ `feld` vergleicht und bei Gleichheit true zurückgibt. Um zwei Koordinaten zu addieren verwende ich die function `vadd`.

Die ganzen Sachen die Ludwig wirklich für seine Wegfindung braucht, (bis auf ein paar globale Variablen) befinden sich in der zurechtfinden.pas. Die anderen Programmabschnitte, die sich eigentlich nicht auf die Programmieranweisung beziehen und nur dafür da sind, damit windowsverwöhnte Menschen nur klicken brauchen, befinden ich in der aufgabe2.pas.

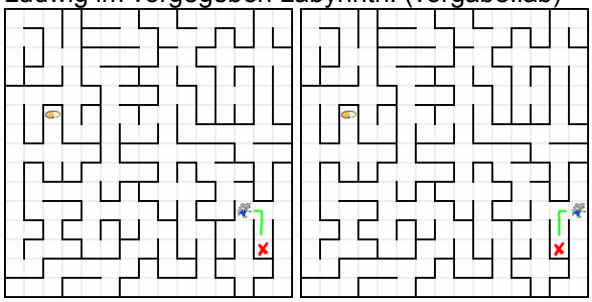
Quelltext (ZURECHTFINDEN.PAS)

```
function feldart(inp:kor) : feld;       //Gibt aus welche Wände um das Feld vorhanden sind
BEGIN
 feldart.u := hirn.hor[inp.x-1, inp.y-1];
 feldart.o := hirn.hor[inp.x-1, inp.y];
 feldart.r := hirn.vert[inp.x, inp.y-1];
 feldart.l := hirn.vert[inp.x-1, inp.y-1];
END;


function vadd(v1, v2: kor):kor;                                  //Addiert zwei Koordinaten
BEGIN
 vadd.x := v1.x + v2.x;
 vadd.y := v1.y + v2.y;
END;


function vergleich(f1, f2:feld):boolean;         //Vergleicht zwei Felder anhand der Wände
BEGIN
 if (f1.o = f2.o)and(f1.r = f2.r)and(f1.u = f2.u)and(f1.l = f2.l) then
  vergleich := true else vergleich := false;
END;


function mausschaut(inp:kor) : feld;      //Gibt einen Wandstatus aus ohne dass die genaue
BEGIN                                                 //Position des Feldes benötigt wird
  mausschaut := feldart(vadd(inp,hirn.start));
END;


function kaesefeld(inp:kor):boolean;            //true wenn Ludwig auf das Käsefeld trifft
BEGIN
 if (hirn.start.x+inp.x = hirn.kaese.x) and
    (hirn.start.y+inp.y = hirn.kaese.y)
    then kaesefeld := true else kaesefeld := false;
END;


procedure zurechtfinden();
const schritte: array[0..3] of kor = ((x:0; y:1),(x:0; y:-1), (x:1; y:0), (x:-1; y:0));

var x, y, i                 : integer;                                  //3x Zählervariablen
    oanz,  nextstep, momanz : integer;                        //offen Anzahl, Schrittrichtung
    delta : kor;                                        //x, y Abweichung Ludwigs vom Startpunkt
    diesesfeld : feld;                                     //Art des Feldes auf dem Ludwig steht
    offen   : array[0..3] of byte;                    //nicht durch Wände versperrte Richtungen
    auswahl : array[0..3] of byte;           //Anzahl der Unterschiedlichkeiten nach Schritt
    bekannt: array[-14..15,-14..15] of boolean;          //true wenn Ludwig schon drauf war
    zuganzahl : word;                                                  //Anzahl der Schritte
BEGIN
 manz := 0;                 //Alle Felder werden zu den Möglichen Startpunkten hinzugefügt
 for x := 1 to 15 do  for y := 1 to 15 do
  if (hirn.kaese.x<>x) or (hirn.kaese.y <> y) then begin
   inc(manz);
   moeglich[manz].x := x;
   moeglich[manz].y := y;
   END;                                          //Alle Felder hat Lundwig noch nicht gesehen
 for x:= -14 to 15 do for y := -14 to 15 do bekannt[x, y] := false;
 zuganzahl := 0;                                                  //Schrittanzahl auf 0 gesetzt
 delta.x := 0;                                             //Abweichung vom Startpunkt auch 0,0
 delta.y := 0;
 bekannt[0,0] := true;                                 //Feld mit abweichung 0,0 schon bekannt
```

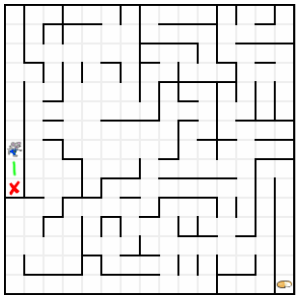
```
 while manz>1  do BEGIN            //Wärenddessen mehr als eine Startmöglichkeit vorhaden:
  diesesfeld := mausschaut(delta);      //Ludwig erhält Informationen &uUML;ber momentanes
Feld

  form1.zugzahl.text := inttostr(zuganzahl);     //Der Zuganzahlencouter wird aktualisiert
  form1.zugzahl.Refresh;
  inc(zuganzahl);

  i := 1;
  while i <= manz do                          //Schleife für alle noch möglichen Zugstarts
   if not(vergleich(diesesfeld,feldart(vadd(moeglich[i], delta)))) then BEGIN
     for x := i+1 to manz do moeglich[x-1] := moeglich[x];    //Wenn keine Übereinstimmung
     dec(manz);                                              //Zugstart aus dem Array entfernen
   END else inc(i);

  if kaesefeld(delta) then BEGIN                             //Wenn Ludwig auf dem Käsefeld:
   manz := 1;                                                //Bestimmt er sine Startpostion
   moeglich[1].x := hirn.kaese.x-delta.x;
   moeglich[1].y := hirn.kaese.y-delta.y;
   END;

  redraw;                                                    //Zeichnent das Labyrinth neu
  putpicture(vadd(delta, hirn.start), 1);             //Ludwigs echte Postion im Labyrinth
  form1.karte.Refresh;

  oanz := 0;                              //Alle möglichen Schritte in offen Array aufnehmen
  if not(diesesfeld.o) then begin offen[oanz] :=0; inc(oanz); end;
  if not(diesesfeld.u) then begin offen[oanz] :=1; inc(oanz); end;
  if not(diesesfeld.r) then begin offen[oanz] :=2; inc(oanz); end;
  if not(diesesfeld.l) then begin offen[oanz] :=3; inc(oanz); end;

  nextstep := trunc(random(oanz));      //Per Zufall die nächste Schrittrichtung festlegen

                                                  //Wenn die Intelligenz Box aktiviert ist
  if (oanz > 1) and form1.intelligenz.Checked then BEGIN
   for i := 0 to 3 do auswahl[i] := 0;
   for i := 0 to oanz-1 do                              //Alle erlaubten Schrittrichtungen
    for x := 1 to manz-1 do                                    //erste Vergleichspartner
     for y := x to manz do                                   //zweiter Vergleichspartner
      if not(vergleich(feldart(vadd(moeglich[x], schritte[offen[i]])),
             feldart(vadd(moeglich[y], schritte[offen[i]])))))     //Wenn Wandeigenschaften
             then inc(auswahl[i]); //nicht übereinstimmen: Counter erhöhen für Schrittrich
   for i := 0 to oanz-1 do       //Nimmt sich die Schrittrichtung mit dem höchsten Counter
    if not(bekannt[delta.x+schritte[i].x, delta.y+schritte[i].y]) then
     if auswahl[i] > auswahl[nextstep] then BEGIN
      nextstep := i;
     END;
  END;


  if (bekannt[delta.x+schritte[offen[nextstep]].x, delta.y+schritte[offen[nextstep]].y])
      and (oanz<>1) then       //Wenn Ludwig auf ein Feld gehen würde auf dem er schon war
  for i := 0 to oanz-1 do                                      //schaut er alle Felder durch
    if not(bekannt[delta.x+schritte[offen[i]].x, delta.y+schritte[offen[i]].y]) then
          nextstep := i;       //ob nicht eins darunter ist das er noch nicht betreten hat


  if manz = 1 then ShowMessage('Ludwig sagt: "Ich bin gerade an der Position (' +
      inttostr(moeglich[1].x+delta.x) + ', ' + inttostr(moeglich[1].y+delta.y)+
      ') und bei ('+inttostr(moeglich[1].x)+', '+inttostr(moeglich[1].y)+') gestartet."');
                        //Wenn Ludwig mit seiner Suche fertig ist gibt er seine Postion an

  delta := vadd(delta, schritte[offen[nextstep]]);//Neues Delta dach dem Schritt berechnen
  bekannt[delta.x, delta.y] := true;          //Diese Position als schon gesehen markieren

  sleep(80);                                                                 //80ms Warten
```

Beispiele:

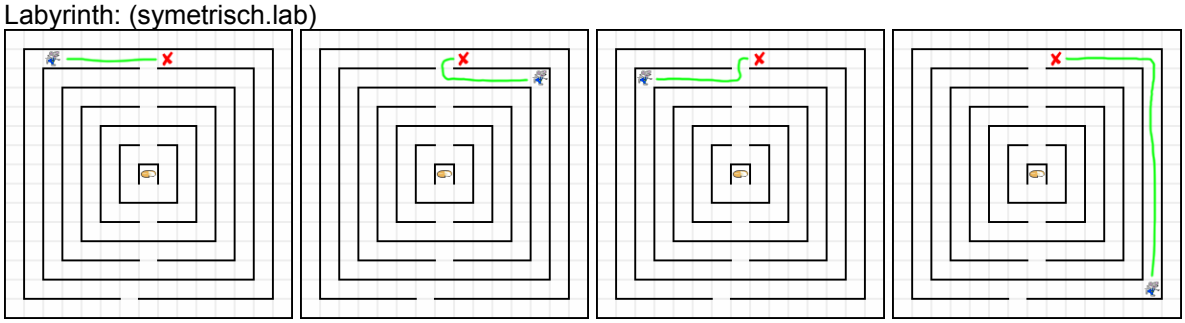
1. Ludwig im vorgegeben Labyrinth: (vorgabe.lab)



2. Ausschlusskriterium Käse: (käsegang.lab)



3. Labyrinth: (symetrisch.lab)



Programmablaufprotokoll:

Wenn das Programm gestartet wird, lädt es erst einmal normal.lab. Das Labyrinth besteht nur aus den Außenwänden. Der Käse befindet sich an Position (1,1) und Ludwig startet an Position (15,15). Man hat die Möglichkeit sich selbst ein Labyrinth zu malen. Dazu drückt man die linke Maustaste in der Nähe der einzelnen Trennwände. War die Wand nicht vorhanden, wird sie aktiviert und schwarz gezeichnet oder andersherum. Auch das Ziehen mit gedrückter Maustaste, um eine längere Wand zu erzeugen, ist möglich. Will man die Position des Käses oder Ludwigs Startposition verschieben, so muss man Ludwig oder den Käse anklicken, die Maustaste gedrückt halten und zur gewünschten Position hinschieben. Mit den Buttons Labyrinth speichern/laden kann man sich seine Labyrinthe speichern und nachher wieder öffnen. Wenn man Ludwig aussetzt, wird die Procedure zurechtfinden gestartet und man sieht Ludwig durch das Labyrinth wandern. Wenn Ludwig sich auskennt, kommt eine Nachricht auf dem Bildschirm, in der er seine Startposition und seine momentane Position angibt. Die Fragezeichen während der Wegsuche symbolisieren die Positionen, an denen Ludwig aus seiner Sicht überall gestartet sein könnte. Die Checkbox Intelligenz bewirkt, dass Ludwig die Felder mit einer höheren Unterschiedlichkeit bevorzugt betritt (meist spielt dies aber in der Länge der Suche keine Rolle).

Quelltext (AUFGABE2.PAS):

```
unit aufgabe2;

interface

uses
  Windows, Forms, Classes, StdCtrls, Dialogs, Controls, ImgList, ExtCtrls, sysutils;

type
  TForm1 = class(TForm)
    karte: TImage;
    laden: TButton;
    speichern: TButton;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    ImageList1: TImageList;
    start: TButton;
    intelligenz: TCheckBox;
    Label1: TLabel;
    zugzahl: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure karteMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
```

```
    procedure karteMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure karteMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure ladenClick(Sender: TObject);
    procedure speichernClick(Sender: TObject);
    procedure OpenDialog1CanClose(Sender: TObject; var CanClose: Boolean);
    procedure SaveDialog1CanClose(Sender: TObject; var CanClose: Boolean);
    procedure startClick(Sender: TObject);
   end;

type kor = record                                                    //x und y Koordinaten
       x, y: shortint end;
     lab = record                                  //Record mit allen Labyrinthparametern
       vert: array[0..15, 0..14] of boolean;                              //Vertikalwände
       hor:  array[0..14, 0..15] of boolean;                             //Horizontalwände
       kaese, start : kor;                                      //Käse- und Startposition
       end;
     feld = record                                //Welche Wand dieses Feldes sind vorhanden
       o, u, r, l : boolean;
       end;
                                    //Ludwigs Bewegungsrichtungen: oben unten rechts links


var
  Form1: TForm1;
  f: file of lab;                                    //Dateityp zum Laden eines Labyorinthes
  hirn : lab;                                                        //Das Wissen der Maus
  moeglich: array[1..225] of kor;                   //Alle möglichen Startpunkte von Ludwig
  manz : byte = 1;                                          //die Anzahl dieser Startpunkte
  zeichnen, art, dragstart, dragkaese : boolean;    //Eigenschaften beim Labyrinthzeichnen

implementation

{$R *.DFM}

procedure vertline(x, y: byte; status:boolean);
BEGIN
with form1.karte do BEGIN
  if status then Canvas.Pen.color := $00000000               //Stiftfarbe je nach Existenz
                else canvas.Pen.color := $00EEEEEE;                  //der Wand Festlegen
  canvas.moveto(x*20+1, (14-y)*20+1);                                    //Linie zeichnen
  Canvas.LineTo(x*20+1, (14-y)*20+21);
  END;
END;


procedure horline(x, y: byte; status:boolean);
BEGIN
 with form1.karte do BEGIN
  if status then Canvas.Pen.color := $00000000               //Stiftfarbe je nach Existenz
                else canvas.Pen.color := $00EEEEEE;                  //der Wand festlegen
  canvas.moveto(x*20+1, (15-y)*20+1);                                     //Linie zeichen
  Canvas.LineTo(x*20+21, (15-y)*20+1);
  END;
END;


procedure putpicture(inp:kor; i: byte);
begin                                                  //Bitmap in das Labyorinth setzen
 form1.ImageList1.Draw(form1.karte.Canvas,(inp.x-1)*20+3,(15-inp.y)*20+3,i);
END;


procedure redraw ();
var x, y : byte;
BEGIN
 form1.karte.Canvas.Rectangle(1,1,302,302);                              //Hintergrund malen
 for x := 0 to 15 do for y:= 0 to 14 do if not(hirn.vert[x, y]) then vertline(x, y, false);
 for x := 0 to 14 do for y:= 0 to 15 do if not(hirn.hor[x, y]) then horline(x, y, false);
 for x := 0 to 15 do for y:= 0 to 14 do if hirn.vert[x, y]  then vertline(x, y, true);
 for x := 0 to 14 do for y:= 0 to 15 do if hirn.hor[x, y] then horline(x, y, true);
                                          //Alle Absperrungen auf den Bildschirm zeichen
 putpicture(hirn.kaese, 0);                                                //Käse zeichnen
 if manz=1 then putpicture(hirn.start, 2) else //Im Normalbetrieb den Startpunkt markieren
    for x := 1 to manz do  putpicture(moeglich[x], 3);    //sonst alle Startmöglichkeinten
  form1.karte.Refresh;
 END;
```

```
{$I zurechtfinden.pas}

procedure TForm1.FormCreate(Sender: TObject);
begin
 randomize;
 assignfile (f, 'normal.lab');                              //hirn aus normal.lab auslenen
 reset(f);
 read(f, hirn);
 closefile (f);
 karte.Canvas.Pen.Width := 2;                             //Stiftbreite auf 2 Pixel setzen
 redraw;
end;


procedure TForm1.karteMouseMove(Sender: TObject; Shift: TShiftState; X,  Y: Integer);
 var rx, ry, dx, dy : byte;
begin
 rx :=trunc(x/20);                                          //x, y Koordinaten das Feldes
 ry :=trunc(y/20);
 dx := x mod 20;                                          //x, y Koordinaten im Feld Intern
 dy := y mod 20;

 if zeichnen then
  if ((dy<4) or (dy>16)) and ((dx>5) and (dx<15)) then begin      //Maus auf Honrizontaler
   if dy > 10 then inc(ry);
   hirn.hor[rx, 15-ry] := art;
   horline(rx, 15-ry, art);
   end else
  if ((dx<4) or (dx>16)) and ((dy>5) and (dy<15)) then begin         //Maus auf Vertikaler
   if dx > 10 then inc(rx);
   hirn.vert[rx, 14-ry] := art;
   vertline(rx, 14-ry, art);
   end;

 if dragkaese then begin                                                  //Den Kase dragen
  hirn.kaese.x := rx+1;
  hirn.kaese.y := 15-ry;
  redraw;
  end else
 if dragstart then begin                                                 //Den Startpunkt dragen
  hirn.start.x := rx+1;
  hirn.start.y := 15-ry;
  redraw;
  end;
end;


procedure TForm1.karteMouseDown(Sender: TObject; Button: TMouseButton;
                                Shift: TShiftState; X, Y: Integer);
 var rx, ry, dx, dy : byte;
begin
 rx :=trunc(x/20);                                          //x, y Koordinaten das Feldes
 ry :=trunc(y/20);
 dx := x mod 20;                                          //x, y Koordinaten im Feld Intern
 dy := y mod 20;

 if ((dy<4) or (dy>16)) and ((dx>5) and (dx<15)) then begin  //Maus auf einer Horizontalen
  if dy > 10 then inc(ry);                    //über die mitte des Feldes --> nächse Linie
  art :=  not(hirn.hor[rx, 15-ry]);                          //die Art der Linie invertieren
  hirn.hor[rx, 15-ry] := art;                                       //im Mäusehirn speichern
  horline(rx, 15-ry, art);
  zeichnen := true;
  end else
 if ((dx<4) or (dx>16)) and ((dy>5) and (dy<15)) then begin    //Maus auf einer Vertikalen
  if dx > 10 then inc(rx);                    //über die mitte des Feldes --> nächse Linie
  art :=  not(hirn.vert[rx, 14-ry]);                         //die Art der Linie invertieren
  hirn.vert[rx, 14-ry] := art;                                      //im Mäusehirn speichern
  vertline(rx, 14-ry, art);
  zeichnen := true;
  end else
 if ((dx>3) and (dx<17)) and ((dy>3) and (dy<17)) then begin       //im Inneren des Feldes
  if (hirn.kaese.x = rx+1) and (hirn.kaese.y = 15-ry) then dragkaese := true;
  if (hirn.start.x = rx+1) and (hirn.start.y = 15-ry) then dragstart := true;
  end
end;
```

```
procedure TForm1.karteMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
 zeichnen := false;                    //Wenn man die Maustaste loslässt wird das Zeichnen
 dragkaese := false;                                  //und die Dragfunktionen deaktiviert
 dragstart := false;
end;


procedure TForm1.ladenClick(Sender: TObject);
begin
 opendialog1.Execute;                          //den vorgefertigten Öffnen Dialog ausführen
end;


procedure TForm1.speichernClick(Sender: TObject);
begin
 savedialog1.Execute;                        //den vorgefertigten Speichen Dialog ausführen
end;


procedure TForm1.OpenDialog1CanClose(Sender: TObject; var CanClose: Boolean);
begin
 assignfile (f, opendialog1.FileName);       //Labyrinth mit Käse- und Startposition laden
 reset(f);
 read(f, hirn);
 closefile (f);
 redraw;
end;


procedure TForm1.SaveDialog1CanClose(Sender: TObject; var CanClose: Boolean);
begin
 assignfile (f, savedialog1.FileName); //Labyrinth mit Käse- und Startposition abspeichern
 rewrite(f);
 write(f, hirn);
 closefile (f);
end;


procedure TForm1.startClick(Sender: TObject);
begin
 zurechtfinden;                                      //Ludwif darf im Labyrinth herumtollen
end;

end.
```