Zum Spiel

Flying Bird ist ein Spiel, bei dem ein Vogel von links nach rechts kommen zwischen Säulen durchfliegen muss. Das Spiel orientiert sich am 2013 für Smartphone entwickeltem „Flappy Bird“.

Programmierkenntnisse

- Array
- Images
- Objektpositionen in einem Fenster
- Ereignisverarbeitung (ButtonClick)

Algorithmischer Kern

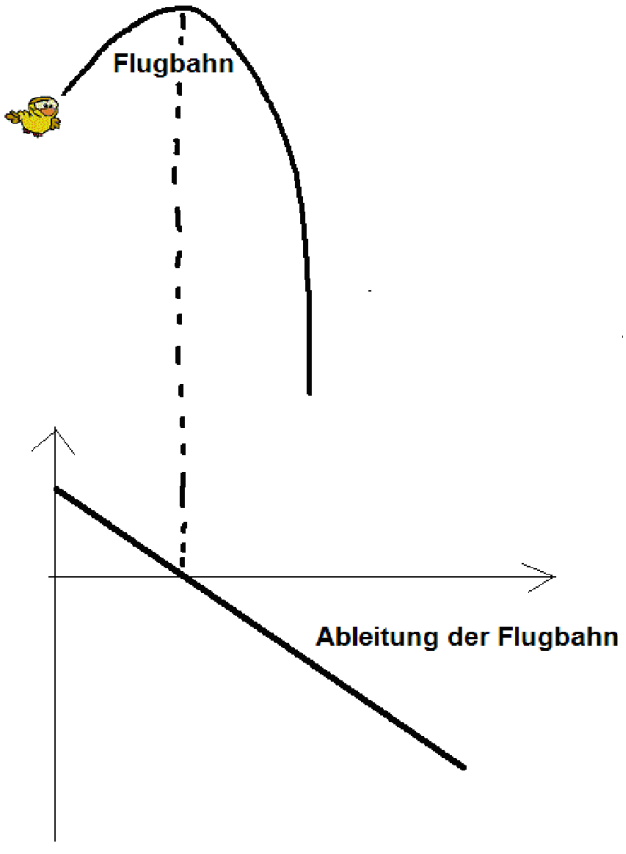
- Kollision von Objekten
- Parabel und Ableitung zur Flugbahnbestimmung
- Implementieren eines Timers

---

1. Schritt: Der Vogel

Der Vogel fliegt auf Mausklick parabelförmig bis zum Scheitelpunkt nach oben und sinkt dann immer schneller ab. Dabei bleibt die x-Koordinate des Vogels unverändert, da die Bewegung durch den sich bewegenden Hintergrund simuliert wird.

Die "Flugbahn" des Vogels wird über einen Timer gesteuert. Würde sich der Vogel nicht nur nach oben oder unten bewegen, so würde seine Flugbahn eine nach unten geöffnete Parabel beschreiben. Ein Flügelschlag setzt den Vogel dabei auf einen Punkt vor dem Scheitelpunkt.



Von dieser Flugbahn ist allerdings nur die Vertikalbewegung parallel zur y-Achse zu sehen.
Damit der Vogel sich tatsächlich auf der Parabel bewegt, muss sich die Position des Vogels um die Momentangeschwindigkeit ändern. Im obigen Beispiel steigt der Vogel zunächst recht schnell und steigt umso langsamer, je näher er dem Scheitel kommt. Ist der Scheitel überrundet, so sinkt der Vogel erst langsam und wird dann immer schneller.
Die Momentangeschwindigkeit entspricht bei der Parabel der Ableitung, hier also einer Geraden.

Zur Anzeige des Vogels wird ein Image verwendet, das entsprechend des Flugs bewegt wird. Die Flugbahn über die Parabel verlangt zwei Koordinaten (x und y → XPosition, YPosition). Lazarus und Python stellen eine Grafikkomponente zur Verfügung, bei Java sollte eine entsprechende Komponente als Klasser erstellt und benutzt werden.

Lazarus

```
var
  Form1: TForm1;
  XPosition : integer;
  YPosition : integer;
  Bird : TImage;

procedure TForm1.FormCreate(Sender: TObject);
begin
   XPosition := -5;
   Bird := Image1;
   Bird.BringToFront;
end;
```

Python

```
XPosition = -5
YPosition = 400

#GUI
Fenster = tkinter.Tk()
Fenster.title("Flappy Bird")
Bird = tkinter.PhotoImage(file="FlappyBirdGif.gif")

canvas=tkinter.Canvas(Fenster, width=600, height=600, bg='white')
canvas.pack()

b = canvas.create_image(200,YPosition,image=Bird, anchor=tkinter.NW)
```

Wenn Images (oder andere Komponenten) in einem Fenster erstellt werden, so werden sie intern durchnummeriert. Mit der Nummer kann das entsprechende Objekt bewegt werden.
Beispiel:

```
  # canvas.create_image(200,YPosition,image=Bird, anchor=tkinter.NW)
  # canvas.move(1,10,-20)
```

würde das erste erstellt Objekt –hier unseren Vogel- um 10 in x-Richtung und um -20 in y-Richtung bewegen.

Das Arbeiten mit diesen Nummern ist allerdings sehr unsicher, da die Nummern in Erstellreihenfolge der Objekte angelegt werden. Vertauscht sich da was, ist alles hin.

Um diese Unsicherheit mit den Nummern zu umgehen, kann eine Referenz auf den Vogel angelegt werden:

```
  # b = canvas.create_image(200,YPosition,image=Bird, anchor=tkinter.NW)
  # canvas.move (b, 10, -20)
```

bewegt nun immer unseren Vogel, egal, wann er erstellt wurde.

Java

```
// externe Klasse, um Bilder im Formular anzuzeigen

import java.awt.*;
```

```
public class Bildanzeige extends Canvas {

  private Image Image1 = null;

  Bildanzeige( String s )  {
     Image1 = Toolkit.getDefaultToolkit().getImage( s );
     // Objekt Image1 vom Typ Image enthält die Grafik-Ressource
     if (Image1 != null)
         repaint();  // Grafik wird neu gezeichnet
     }

  public void paint( Graphics g )  {
      g.drawImage(Image1, 0, 0, this );
  }

  public void anzeigen (String s) {
     Image1.flush(); // Bild freigeben
     Image1 = Toolkit.getDefaultToolkit().getImage( s );
     if (Image1 != null)
        repaint();
     }
} // Ende der Klasse "Bildanzeige"


// GUI
public class GUI extends JFrame {
   public Bildanzeige BildVogel = new Bildanzeige("FlappyBirdGif.gif");

  int XPosition = -5;
  int YPosition;


  private Canvas canvas1 = new Canvas();
  // Ende Attribute


public GUI(String title) {
    // Frame-Initialisierung
    super(title);
    setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    int frameWidth = 524;
    int frameHeight = 503;

    // Damit der Java-Editor eine feste Fenstergröße beibehält,
    // wird über setSize() die Größe fest gesetzt.
    int Breite = 600;
    int Hoehe = 500;
    setSize(Breite, Hoehe);
    [. . . ]

    cp.setBackground(Color.WHITE);

    // Eigene Komponenten + Bilder
    BildVogel.setBounds(100, 300, 51, 36);
    cp.add(BildVogel);


    // Die Canvas ist notwendig, um die Mausklicks abzufangen.
    // FensterAdapter klappte nicht???
    // Die Canvas nach den Bildern, sonst sieht man nichts
    cp.add(canvas1);
    canvas1.setBounds(-8, 24, 585, 449);
    // wieder auf die Fenstergröße anpassen, sonst ändert's der
    // Java-Editor ständig
    canvas1.setSize(Breite, Hoehe);
    canvas1.setLocation(0,0);
```

```
} // Ende Konstruktor
}// Ende GUI-Klasse
```

Die Gerade der Parabelableitung könnte so implementiert werden:

Lazarus

```
function Gerade (x: integer) : integer;
begin
  result := -3*x+4
end;
```

Python

```
def Gerade(x):
     return -3*x + 4
```

Java

```
int Gerade (int x){
    int y = -3*x+4;
    return y;
    }
```

Im Timer wird die x-Position des Vogel erhöht und der entsprechende y-Wert berechnet. Der x-Wert wird allerdings nicht dargestellt, nur die y-Position.

Lazarus

```
procedure TForm1.Timer1Timer(Sender: TObject);
var i: integer;
begin
  XPosition := XPosition +1;
  YPosition := Image1.top -  Gerade(XPosition)  ;
  Bird.Top:= YPosition;
end;
```

Python

```
timerOn = True

[. . . ]

def Timer():
      global XPosition
      global YPosition
      global timerOn
      XPosition = XPosition +1
      YPosition = Gerade (XPosition)

 canvas.move(b, 0,  -YPosition)

 # b ist die Referenz auf den Vogel
 # über move wird das referenzierte Objekt in x-Richtung
 # und y-Richtung bewegt

      if (timerOn == True):
           Fenster.after(50, Timer)
```

Java

```
public class GUI extends JFrame {
    Timer t;

ActionListener Bewegen = new ActionListener() {
    public void actionPerformed (ActionEvent evt) {
      XPosition = XPosition +1;
      YPosition =  (BildVogel.getY() - Gerade(XPosition));
```

```
      BildVogel.setLocation(100, YPosition);
      }
  };

    // Timer starten
    t = new Timer(100, Bewegen);
    t.start();

  } // end of public GUI
```

Beim Klick wird die X-Position auf -5 gesetzt, der Vogel startet bzgl. der y-Koordinate wieder vor dem Scheitelpunkt, er fliegt also etwas nach oben, bevor es wieder runter geht:

Lazarus

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
   XPosition := -5;
end;
```

Python

```
def ButtonDown(event):
      global XPosition
      XPosition = -5

#GUI
canvas.bind(sequence="<Button-1>", func = ButtonDown)
```

Java

```
public void canvas1_MouseClicked(MouseEvent evt) {
    XPosition = -5;
  } // end of canvas1_MouseClicked
```

Alternativ kann statt des Mausklicks auch eine Taste gedrückt werden. Über die Bibliothek LCLType kann auf die Tasten über sog. virtuelle Tastencodes zugegriffen werden. Werden Tasteneingaben und grafische Bedienelemente (Button …) gleichzeitig genutzt, so muss der Tastendruck vom Formular vor einer Komponente empfangen werden → *KeyPrewiew muss true* sein.

Lazarus

```
uses
  [. . .]  LCLType;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift:
TShiftState );
begin
  if key = VK_SPACE then
      XPosition := -5;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   [. . .]
   // wegen virtuellen tastencodes mit visuellen Komponenten
   Form1.KeyPreview:=true;
end;
```

Python

```
def key(event):
      global XPosition
```

	`XPosition = -5` `# GUI` `Fenster.bind("<Key>", key)`
Java	`// bevor die Eingaben entgegen genommen werden,` `// einmal mit der Maus aufs Fenster geklickt werden` `// → Warum weiß ich leider nicht` `public void canvas1_KeyPressed(KeyEvent evt) {` `  XPosition = -5;` `} // end of canvas1_KeyPressed`

2. Schritt – Kollision des Vogels mit dem Bildschirmrand oben und unten

In einer Funktion wird mit jeder Timerausführung getestet, ob die Y-Koordinate des Vogel größer oder kleiner der Fensterdimensionen ist. Verlässt der Vogel den Bildschirm, stoppt der Timer.

Lazarus	`function testeKollisionObenUnten: boolean;` `begin` `  result := false;` `  if (Bird.Top < 0) or (Bird.top + Bird.Height > Form1.Height) then` `  result := true;` `end;` `procedure TForm1.Timer1Timer(Sender: TObject);` `var i: integer;` `begin` `  [. . .]` `  if testeKollisionObenUnten then` `  timer1.Enabled:=false;` `end;`
Python	`def testeKollisionObenUnten():` `  global timerOn` `  a = canvas.bbox(1)` `  u = canvas.winfo_height()` `  if (a[1]<0):` `  timerOn = False` `  if ((a[1]+Bird.height()) > canvas.winfo_height()):` `  timerOn = False` `# Bei Python ist die Position eines Objekts auf dem` `# Formular (Canvas) bbox aubfragbar. bbox liefert` `# ein 4-Tupel zurück (X1,Y1, X2, Y2), das die linke obere` `# und die rechte untere Ecke eines Rechtecks um das` `# entsprechende Objekt markiert.`
Java	`public boolean testeKollisionObenUnten( ) {` `  boolean Ergebnis = false;` `  if ((BildVogel.getY() < 0) | (BildVogel.getY() +` `  BildVogel.getHeight() > canvas1.getHeight()-20)) {` `  Ergebnis = true;` `  }` `  return Ergebnis;` `}`

	`` ActionListener Bewegen = new ActionListener() { public void actionPerformed (ActionEvent evt) { [. . .] if (testeKollisionObenUnten()) { t.stop(); } // end of if } }; ```

3. Schritt – Die Säulen und die Säulenbewegung

Insgesamt reichen 4 Säulen aus (insgesamt also 2 Säulenpaare), um die Bewegung zu simulieren. Für die spätere Kollisionsabfrage und die Bewegung lohnt die Verwaltung der Säulenimages in einem Array.
Die Bewegung erfolgt im Timer, die Säulen (Hindernisse) werden, sobald sie den linken Bildrand erreichen, in einer Prozedur wieder nach rechts gesetzt und zufällig in der Höhe versetzt.

Lazarus

```
var
  Hindernis : array [1..4] of TImage;

// Die Verschiebung erfolgt paarweise,
//daher läuft die Schleife nur von 1 bis 2

procedure HindernisseNachRechts;
var i : integer;
begin
  for i := 1 to 2 do
    if Hindernis[i].left < 0 then
        begin
        Hindernis[i].Left := Form1.Width;
        Hindernis[i+2].Left:=Form1.Width;
        Hindernis[i].top := Form1.Height DIV 2 + 50 + 50 - random(100);
        Hindernis[i+2].top := Hindernis[i].top - 100 -
Hindernis[i].Height;
        end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   randomize;
   [. . .]
   Hindernis[1] := Image2;
   Hindernis[2] := Image4;
   Hindernis[3] := Image3;
   Hindernis[4] := Image5;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var i: integer;
begin
  [. . .]
  for i := 1 to 4 do
      Hindernis[i].Left:=Hindernis[i].left-10;
  HindernisseNachRechts;
end;
```

Python

```
def Timer():
      [. . .]
```

```
      # Säulen warden nach links bewegt
      for i in range(4):
            canvas.move(s[i], -10,0)

      # Säulen werden beim Erreichen des linken Rands
      # nach rechts gesetzt
      for i in range(4):
            a = canvas.bbox(s[i])
            if (a[0] < 0):
                  canvas.move(s[i],600,0)

      if (timerOn == True):
           Fenster.after(50, Timer)


#GUI
Saeule = tkinter.PhotoImage(file="Saeule.gif")
[. . .]
s[0] = canvas.create_image(300,500, image=Saeule, anchor=tkinter.NW)
s[1] = canvas.create_image(300,0,image=Saeule, anchor=tkinter.NW)
s[2] = canvas.create_image(580,450,image=Saeule, anchor=tkinter.NW)
s[3] = canvas.create_image(580,0,image=Saeule, anchor=tkinter.NW)
```

Java

```
public class GUI extends JFrame {
  [. . .]
      public Bildanzeige [] Saeulen = new Bildanzeige[4];
      // Die Zuweisung erfolgt im GUI-Konstruktor
      // Eine Alternative wäre die direkte Array-Deklaration:
      //
      // public Bildanzeige [] Saeulen = new Bildanzeige[]
      //   {new Bildanzeige("Saeule.gif"), new Bildanzeige("Saeule.gif"),
      //       new Bildanzeige("Saeule.gif"), new
Bildanzeige("Saeule.gif")}  ;

public void HindernisseNachRechts() {
    for (int i=0; i<3; i=i+2) {
      if (Saeulen[i].getX() < 0) {
        Saeulen[i].setLocation(canvas1.getWidth(), canvas1.getHeight() /
2 +50 +20 - (int) (Math.random()*150));
        Saeulen[i+1].setLocation(canvas1.getWidth(),  Saeulen[i].getY() -
150 - Saeulen[i].getHeight()  );
      }
    }
  } // Ende HindernisseNachRechts

ActionListener Bewegen = new ActionListener() {
      [. . .]
      for (int i = 0;i<4 ;i++ ) {
        Saeulen[i].setLocation(Saeulen[i].getX()-10,
Saeulen[i].getY());
      } // end of for

      HindernisseNachRechts();
    }
  };

public GUI(String title) {
[. . .]
  for (int i=0 ; i<4 ;i++ ) {
      Saeulen[i] = new Bildanzeige("Saeule.gif");
    } // end of for

    Saeulen[0].setBounds(300,0,90,200);
    cp.add(Saeulen[0]);
    Saeulen[1].setBounds(300,400,90,200);
    cp.add(Saeulen[1]);
    Saeulen[2].setBounds(600,0,90,200);
```

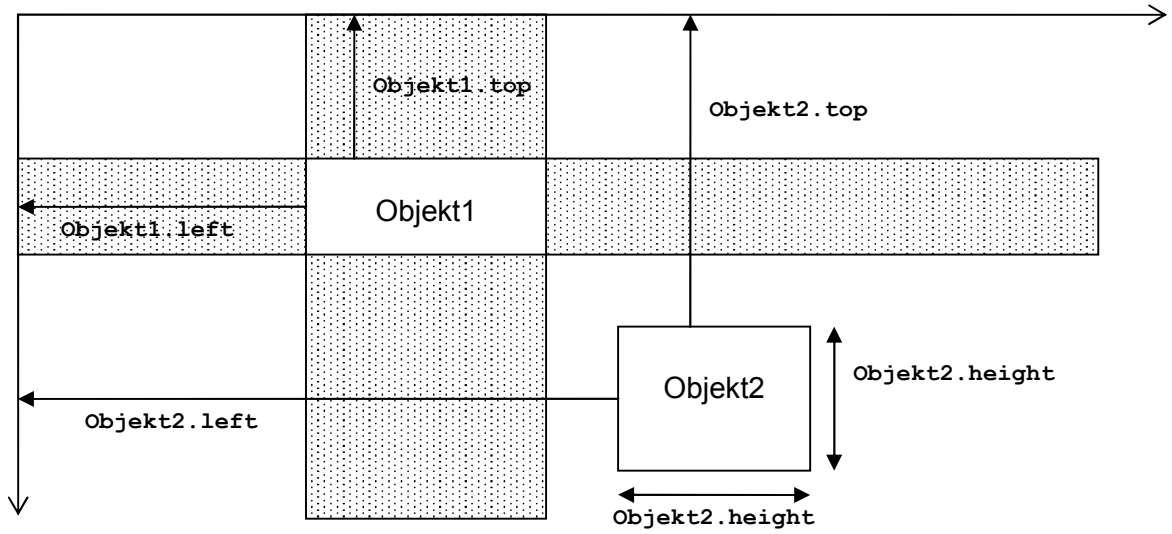
```
    cp.add(Saeulen[2]);
    Saeulen[3].setBounds(600,400,90,200);
    cp.add(Saeulen[3]);
}
}// Ende Bewegen
```

4. Schritt: Kollisionsabfrage des Vogels mit den Säulen

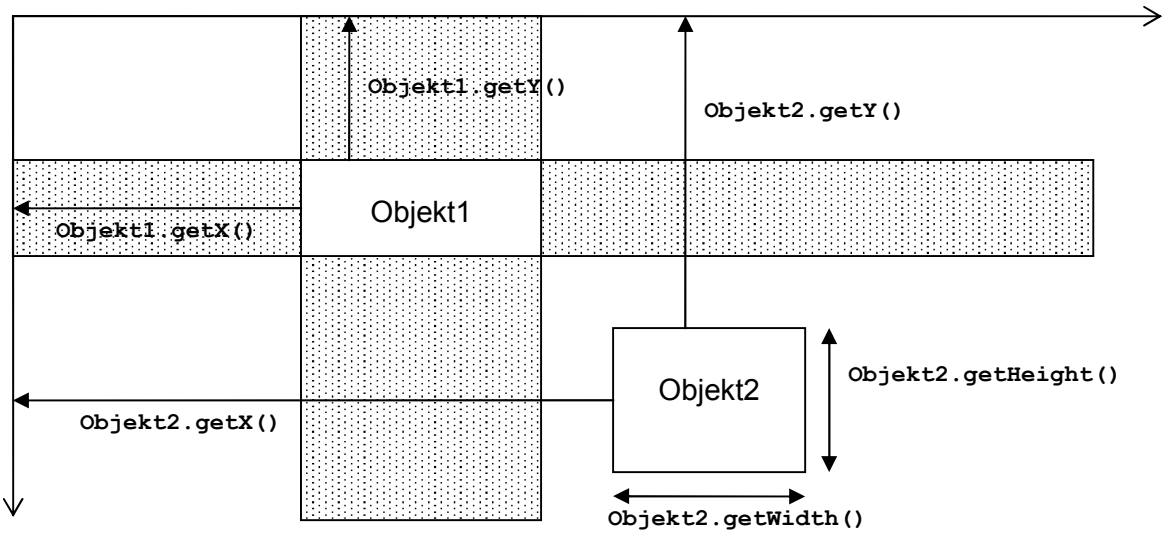
Kollisionsabfragen zwischen zwei Objekten ist im Bereich der Spiele eine oft auftauchende Standardoperation. Zu beachten ist, dass der Ursprung des Bildschirmkoordinatensystems oben links liegt und nur in die positiven Bereche ragt.

Objekt2 kollidiert mit Objekt2, wenn es gleichzeitig in den horizontalen Schraffurbereich und in den vertikalen Schraffurbereich des Objekts1 hineinragt.

Lazarus



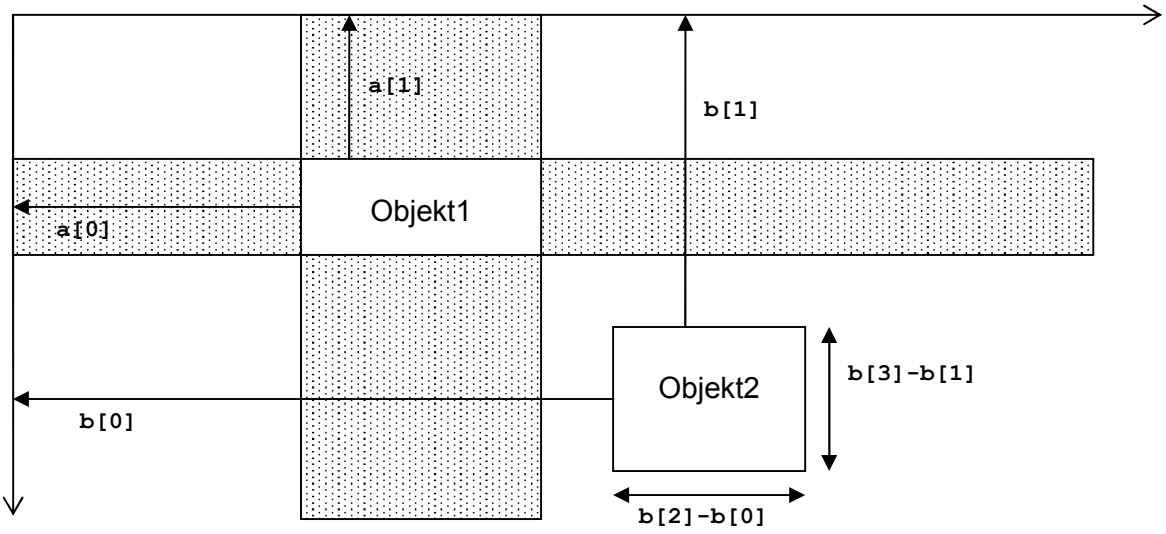
Java



Python

Die Funktion *bbox* liefert ein 4-Tupel, das die linke obere und die rechte untere Ecke des Rahmens eines Objekts (x1, y1, x2, y2) beinhaltet

```
a = canvas.bbox(Objekt1)
b = canvas.bbox(Objekt2)
```



Damit sieht die Kollisionsabfrage so aus:

Lazarus

```
function testeKollisionMitHindernis: boolean;
var i : integer;
begin
  result := false;
  for i := 1 to 4 do
   if (Bird.top < Hindernis[i].Top + Hindernis[i].Height) and
       (Bird.top + Bird.Height > Hindernis[i].top) and
        (Bird.left < Hindernis[i].Left + Hindernis[i].Width) and
         (Bird.left + bird.Width >Hindernis[i].left)
        then
            result := true;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var i: integer;
begin
[. . . ]
  if testeKollisionMitHindernis then
     timer1.Enabled:=false;
end;
```

Python

```
def Timer():
      [. . .]
      if testeKollisionMitHindernis():
         timerOn = False


def testeKollisionMitHindernis():
    global timerOn
    Rueckgabe = False
    for i in range (3):
          Hindernis = canvas.bbox(s[i])
          Vogel = canvas.bbox(b)
          if ((Vogel[1] < Hindernis[3]) and (Vogel[3] >  Hindernis[1])
                 and (Vogel[2] > Hindernis[0]) and
                  (Vogel[0] <  Hindernis[2])):
                Rueckgabe = True
    return Rueckgabe
```

Java

```
public boolean testeKollisionMitHindernis() {
  boolean Ergebnis = false;
  for (int i = 0; i<4; i++) {
    if ((BildVogel.getY() < (Saeulen[i].getY() + Saeulen[i].getHeight()))
      & ((BildVogel.getY() + BildVogel.getHeight()) > Saeulen[i].getY())
      & (BildVogel.getX() < Saeulen[i].getX() + Saeulen[i].getWidth())
      & (BildVogel.getX() + BildVogel.getWidth() > Saeulen[i].getX())){
           Ergebnis = true;
      }
    }
    return Ergebnis;
  }

ActionListener Bewegen = new ActionListener() {
    [. . . ]
      if (testeKollisionMitHindernis()) {
        t.stop();
      }
  };
```