Spiele programmieren mit Lazarus, Java und Python

1. Einsatz eines Timers

Lazarus

Timersymbol in der Reiterleiste und Timerprozedur

Timer an: `Timer1.enabled := true;`
Timer aus: `Timer1.enabled := false;`

Python

I) Implementieren des Timers
In Python gibt es (meines Wissens) keine Klasse für einen Timer. Hier muss ein wenig "getrickst" werden, indem eine normale Prozedur –ich nenne sie "Timer"- definiert wird, die sich wieder rekursiv zeitverzögert aufruft. Die Prozedur 'after' des Fensterobjekts, die zeitverzögert eine Prozedur aufrufen kann, ist hier dienlich.
Gestartet wird der Timer über den "after"-Aufruf in der GUI.

```
def Timer():
    [. . .]
    Fenster.after(500, Timer)

#GUI
Fenster = tkinter.Tk()
Fenster.after(50, Timer)
```

II) An- und Ausschalten des Timers
Das An- und Ausschalten des Timers kann über eine boolesche Variable geregelt werden.

```
timerOn = True
timerOn = False

def Timer():
    if (timerOn == True):
       Fenster.after(500, Timer)

#GUI
Fenster = tkinter.Tk()
```

Java

I) Implementieren des Timers
Für den Timer gibt es mehrer Klassen. Wir können den Timer der javax.swing nehmen. Wir der Timer nicht explizit importiert, kann es zu einer Fehlermeldung kommen, da in der java.awt auch ein Timer implementiert ist.

```
import javax.swing.Timer;

Timer t;

ActionListener TimerProzedur = new ActionListener() {
    public void actionPerformed (ActionEvent evt) {
    [. . .]
    }
}

t = new Timer(500, TimerProzedur);
```

II) Ein- und Ausschalten des Timers

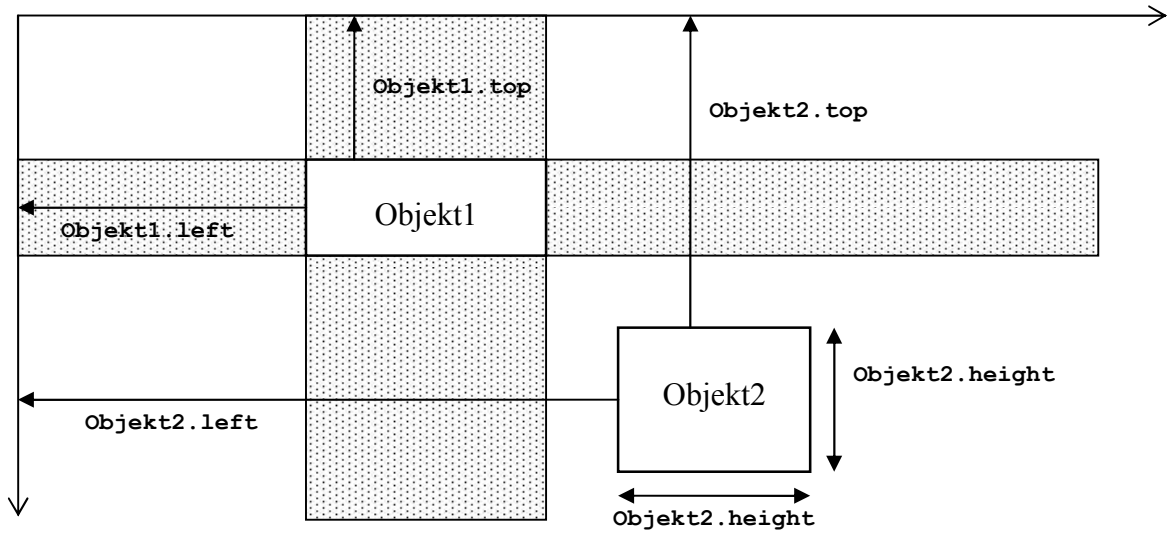
```
t.start();
t.stop();
```

2. Kollision zweier Objekte

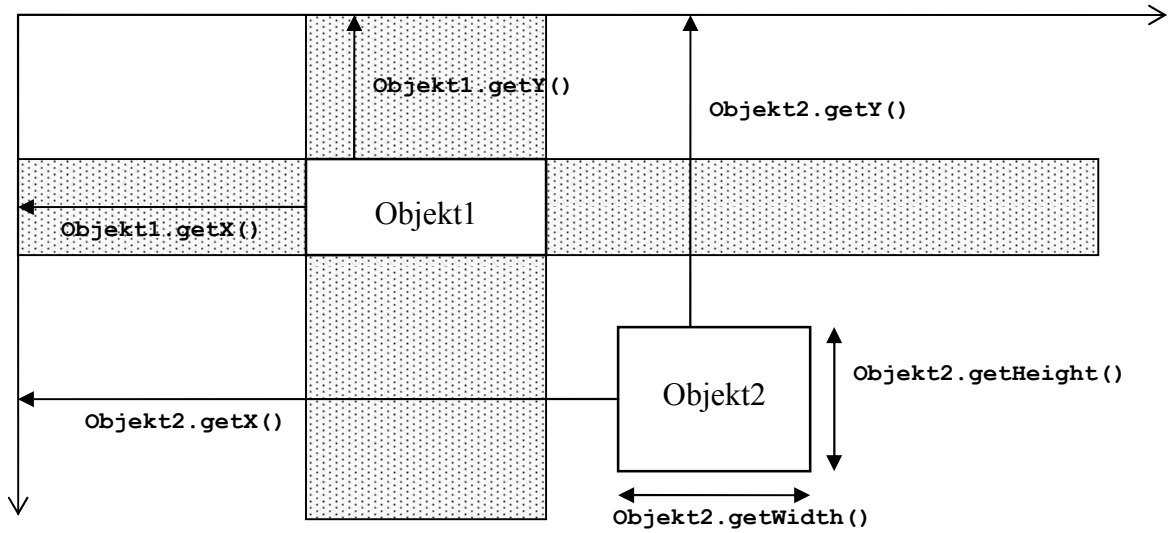
Der Ursprung des Bildschirmkoordinatensystems liegt oben links und geht auf x- und y-Achse in den positiven Bereich.

Objekt2 kollidiert mit Objekt1, wenn es gleichzeitig in den horizontalen Schraffurbereich und in den vertikalen Schraffurbereich des Objekts1 hineinragt.

Lazarus



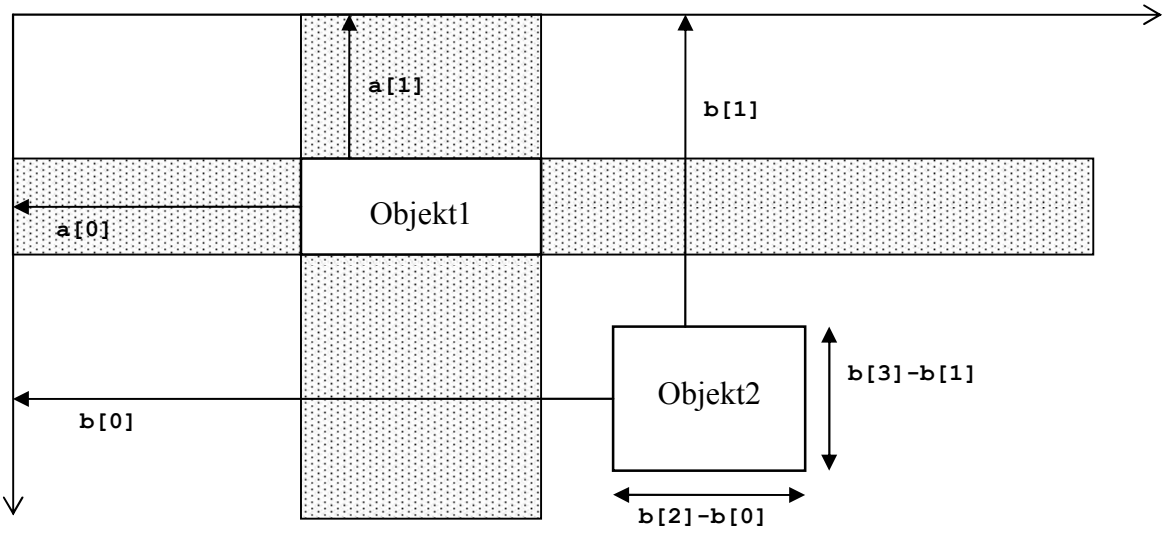
Java



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 testeKollision: boolean;
begin
  result := false;

   if (objekt1.top < objekt2.Top + objekt2.Height) and
       (objekt1.top + objekt1.Height > objekt2.top) and
        (objekt1.left < objekt2.Left + objekt2.Width) and
         (objekt1.left + objekt1.Width > objekt2.left)
        then
            result := true;
end;
```

Python

```
def testeKollision():
  Rueckgabe = False
  a = canvas.bbox(objekt1)
  b = canvas.bbox(objekt2)
  if ((a[1]<b[3]) and (a[3]>b[1]) and (a[2]>b[0]) and (a[0]<b[2])):
       Rueckgabe = True
  return Rueckgabe
```

Java

```
public boolean testeKollision() {
  boolean Ergebnis = false;
    if ((objekt1.getY() < (objekt2.getY() + objekt2.getHeight()))
      & ((objekt1.getY() + objekt1.getHeight()) > objekt2.getY())
      & (objekt1.getX() < objekt2.getX() + objekt2.getWidth())
      & (objekt1.getX() + objekt1.getWidth() > objekt2.getX())){
           Ergebnis = true;
           }
    return Ergebnis;
  }
```

3. Einbinden von Bildern für Spieler- oder Gegenspielerfiguren

Lazarus

--> Image-Komponente

Bild laden über die Eigenschaft: Picture → Ladedialog
Bild proportional darstellen über Eigenschaft: proportional := true

Java

I) Neue Klasse für die Bildanzeige
Zur Darstellung von Bildern definiert man am Besten eine neue Klasse "Bildanzeige", die von einer bilddarstellungsfähigen Klasse abgeleitet wird (Canvas oder JPanel). Die Funktion "anzeigen" kann neue Bilder zur Laufzeit laden.

```
// 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"
```

II) Bild im Formular anlegen
Ein erzeugtes Objekt vom Typ "Bildanzeige" kann nun im Frame angezeigt werden.

```
   public Bildanzeige a = new Bildanzeige("FlappyBirdGif.gif");

   a.setBounds(100, 300, 51, 36);
   cp.add(a);
```

Python

Ähnlich wie bei Java muss ein Image in ein Objekt vom Typ PhotoImage geladen werden, um dann auf einer Zeichenfläche (canvas) angezeigt zu werden. Als Formate sind neben „gif“ und „pgm“ angeblich noch „jpg“ möglich.

```
#GUI
Bild = tkinter.PhotoImage(file="FlappyBirdGif.gif")

canvas=tkinter.Canvas(Fenster, width=600, height=600, bg='white')
canvas.pack()
b = canvas.create_image(200,300,image=Bild, anchor=tkinter.NW)
```

4. Bewegungen von Objekten (z.B. Bildern)

Lazarus

In Lazarus können die Attribute left und top direkt verändert werden.

```
  Objekt1.left  := Objekt1.left + 10;
  Objekt1.top  := Objekt1.top -20;
```

Java

Hier können die Koordinaten der linken oberen Ecke x und y ausgelesen und über "setLocation()" neu belegt werden.

```
objekt1.setLocation(objekt1.getX()+10, objekt1.getY()-20);
```

Python

Wenn Images (oder andere Komponenten) in einem Fenster erstellt werden, so werden sie intern durchnummeriert. Mit der Nummer kann das entsprechende Objekt relativ zu seiner aktuellen Position bewegt werden.

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

   canvas.move(1,10,-20)
```

Hier wird das erste erstellte Objekt –unser Bild- um 10 in x-Richtung und um -20 in y-Richtung bewegt.

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, sollte eine Referenz auf das Bild angelegt werden:

```
   Bild = tkinter.PhotoImage(file="FlappyBirdGif.gif")
   b = canvas.create_image(200,200,image=Bild, anchor=tkinter.NW)

   canvas.move (b, 10, -20)
```

5. Tastatureingaben und virtuelle Tastencodes

Lazarus

Um Tastatureingaben abzufangen, sollte zunächst das Formularattribut "keyPreview" auf true gesetzt werden.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Form1.KeyPreview:=true;
end;
```

In der Unit LCLType sind die sog. virtuellen Tastencodes enthalten, die in der FormKeyDown in der Referenzvariablen Key bei Tastendruck abgefangen und abgefragt werden können.

```
uses
  LCLType;


procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  case Key of
     VK_UP : [. . .]     // Pfeil hoch
     VK_RIGHT : [. . .]  // Pfeil rechts
     VK_DOWN : [. . .]   // Pfeil unten
     VK_LEFT : [. . . ]  // Pfeil links
     VK_SPACE : [. . .]  // Leertaste
  end;
end;
```

Java

Um Tastatureingaben auf z.B. der Canvas abzufangen, sollte zunächst der Fokus auf das Canvas gesetzt werden.

```
canvas1.requestFocus();
```

In der Klasse "KeyEvent" sind die sog. virtuellen Tastencodes enthalten, die in einem *KeyListener* bei Tastendruck über "getKeyCode()" abgefangen und abgefragt werden können.

```
canvas1.addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent evt) {
        canvas1_KeyPressed(evt);

        if (evt.getKeyCode() == KeyEvent.VK_RIGHT ) {        //Pfeil hoch
          [. . .]
        } else if (evt.getKeyCode() == KeyEvent.VK_LEFT ) {  // Pfeil rechts
          [. . .]
        } else if (evt.getKeyCode() == KeyEvent.VK_UP ) {    // Pfeil hoch
          [. . .]
        } else if (evt.getKeyCode() == KeyEvent.VK_DOWN ) {   //Pfeil unten
          [. . .]
        }
        repaint();
      }
    });
```

Python

Ereignisse (Tastenereignisse/ Mausereignisse) können an eine Komponente "gebunden" werden. Bei der Bindung wird der Verbindung Komponente-Taste eine Prozedur zur Ausführung zugeordnet.

```
def rechts (event):
    [. . .]

def links (event):
  [. . .]

def hoch(event):
  [. . .]

def runter(event):
  [. . .]

#GUI
Fenster = tkinter.Tk()
Fenster.bind("<KeyPress-Left>", links)
Fenster.bind("<KeyPress-Right>", rechts)
Fenster.bind("<KeyPress-Down>", runter)
Fenster.bind("<KeyPress-Up>", hoch)
#GUI ENDE
```

6. Listenverarbeitende visuelle Komponente (ListBox)

Lazarus

Komponente: ListBox

I) Markierte Zeilennummer auslesen:

```
procedure TForm1.ListBox1Click(Sender: TObject);
var index : Integer;
begin
    index := ListBox1.ItemIndex+1;    // Nummer fängt bei 0 an
end;
```

II) Markierten Text auslesen:

```
s := Listbox1.GetSelectedText;
```

III) Befüllen:

```
ListBox1.Items.append('Beispiel');
```

Java

Hier kann die Klasse "JList" als Liste benutzt werden. Um eine Liste zu befüllen braucht es eine Adapter-Klasse ListModel.

```
private JList jList1 = new JList();
private DefaultListModel jList1Model = new DefaultListModel();

jList1.setModel(jList1Model);
```

I) Eine Liste zu befüllen führt nun über den Adapter:

```
jList1Model.removeAllElements();
jList1Model.addElement("Beispieltext");
```

II) Das Auslesen der Nummer der geklickten Zeile:

```
public void jList1_MouseClicked(MouseEvent evt) {
    int index = jList1.getSelectedIndex();
  }
```

Python

Komponente: ListBox

```
def Liste1Click(event):
     [. . .}


#GUI

Liste1 = tkinter.Listbox (width=30, height = 10)
Liste1.bind('<<ListboxSelect>>', Liste1Click)

# Die doppelten eckigen Klammern sind richtig
```

I) Markierte Zeilennummer auslesen:

```
def Liste1Click(event):
     Tupel = Liste.curselection()
     index = int(Tupel[0])
```

II) Markierten Text auslesen:

```
def Liste1Click(event):
   Tupel = Liste1.curselection()
   index = int(Tupel[0])
   s = Liste1.get(index)
```

III) Befüllen:

```
Liste1.delete("0","end")
Liste1.insert(" Beispieltext"))
```

7. Mausclicks auf visuelle Komponenten

Lazarus

Die Struktur der Ereignisprozeduren (*hier: MouseDown*) in **Lazarus** erhält man am leichtesten, wenn man für eine entsprechende Beispielkomponente auf dem Formular die Prozedur auswählt und dann die *Komponente* und *„Sender: TObject“* weglässt:

procedure ~~TForm1.Button1~~MouseDown (~~Sender: TObject;~~
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

Bei abgeleiteten Komponenten wird einfach die MouseDown (oder eine andere) überschrieben.

```
   procedure MouseDown(Button: TMouseButton;
             Shift: TShiftState; X, Y: Integer); override;
```

```
procedure TMineButton.MouseDown(Button: TMouseButton;
                Shift: TShiftState; X,Y: Integer);
begin
  if button = mbLeft then [. . .]

  if button = mbRight then [. . .]
```

Java

In unserer **Java**-Version wird im Constructor einfach ein Mouse-Listener hinzugefügt, der bei **mousePressed** die Methode *LinksClick()* bzw. *RechtsClick()* aufruft. Die Maustasten werden durch das Attribut **evt.isMetaDown() == true** unterschieden. True bei rechter Maustaste, false bei linker Maustaste.

```
      this.addMouseListener(new MouseAdapter() {
        public void mousePressed(MouseEvent evt) {
          if (evt.isMetaDown() == true) {
              //rechte Maustaste- bei false linke Maustaste
             RechtsClick();
          } else {
             LinksClick();
             }
          }
      });
}

void LinksClick() {}

void RechtsClick() {}
```

Python

Die Click-Methode muss bei **Python** separate als einfache Methode implementiert und über die Methode „bind“ an das Mausereignis <ButtonPress-1> für die linke Maustaste bzw. <ButtonPress-3> für die rechte Maustaste gebunden werden.

```
          widget.bind(event, handler)
```

linke Maustaste:	<Button-1>, <ButtonPress-1> oder <1>
mittlere Maustaste:	<Button-2>, <ButtonPress-2> oder <2>
rechte Maustaste	<Button-3>, <ButtonPress-3> oder <3>
linke Taste loslassen	<ButtonRelease-1>
Doppelclick	<Double-Button-1>

http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

```
    def __init__ (self, master):
       Label.__init__(self, master)
       self.bind(sequence="<ButtonPress-1>", func=self.LinksClick)
    self.bind(sequence="<ButtonPress-3>", func=self.RechtsClick)
```