# Themen

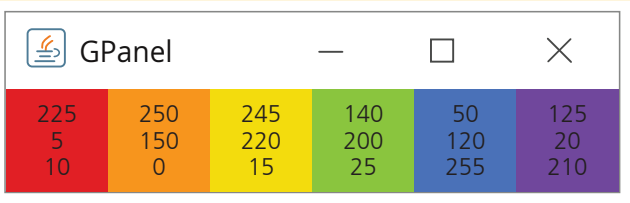


weitere Themen siehe Seite 55

# Themen

# Farben in Graustufen umwandeln

Für das Umwandeln eines Farbbildes in ein Graustufenbild werden die RGB-Farbwerte verwendet, die für jedes einzelne Pixel mit Hilfe der Funktion `getPixelColor(x,y)` ❶ ermittelt werden.




Anhand der drei Werte für rot, grün und blau wird dann ein Grauwert g ermittelt ❷, der mit der Funktion `makeColor()` zu einer Graustufe kombiniert wird ❸.

Es gibt unterschiedliche Methoden für das Ermitteln des Grauwerts. Im Beispiel wird jeweils der größte der drei RGB-Farbwerte verwendet, also 225, 250, 245 usw. ❷.

Das Ergebnis dieser besonders einfachen Methode ist ein Graustufenbild mit sehr hellen Farbflächen.

```
from gpanel import *
makeGPanel(Size(240, 40))
bild = getImage("D:/.../farben.png")

graustufen = []
for x in range(240):
    for y in range(40):
        pixel = bild.getPixelColor(x,y) ❶
        rot = int(pixel.red*255)
        gruen = int(pixel.green*255)
        blau = int(pixel.blue*255)
        g = max(rot, gruen, blau)  ❷
        grau = makeColor(g, g, g)  ❸
        if g not in graustufen:
            graustufen.append(g)
        bild.setPixelColor(x, y, grau)

image(bild, 0, 0)
print(graustufen)
```

```
[225, 250, 245, 200, 255, 210]
```


Die umgekehrte Methode – das Ermitteln des Grauwerts anhand des kleinsten der drei RGB-Farbwerte – führt entsprechend zu Graustufenbildern mit sehr dunklen Farbflächen.

Bei einer weiteren Methode wird der Grauwert als Ganzzahl des Mittelwerts der drei RGB-Farbwerte berechnet ❺ (in RGB-Farbangaben sind nur ganze Zahlen zulässig).

Das Ergebnis ist hier ein relativ dunkles und kontrastarmes Graustufenbild.

```
g = int((rot + gruen + blau)/3)  ❺
```

```
[80, 133, 160, 121, 141, 118]
```


Da einfache Umwandlungsmethoden wie die drei genannten unnatürliche Graustufenbilder liefern, wird für das hochauflösende Fernsehen (HDTV) eine Formel genutzt, in der die drei Farbwerte unterschiedlich stark berücksichtigt werden:
grau = 0,2126 · rot + 0,7152 · grün + 0,0722 · blau.

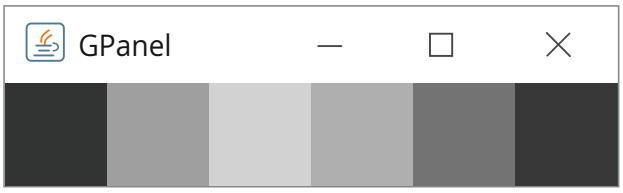
Wie unterschiedlich die Gewichtung der Farben bei der Berechnung des Grauwertes ist, zeigt die Tortengrafik.

Die Formel berücksichtigt nicht nur die reinen Farbwerte, sondern auch die unterschiedliche Helligkeit der Farben, die das menschliche Auge besser erkennen kann als kleine Farbabweichungen.

Das Ergebnis der Formel sind plastische Graustufenbilder, die dank einer größeren Helligkeitsspanne deutlich kontrastreicher sind.

```
g = int(0.2126 * rot + 0.7152 * gruen
+ 0.0722 * blau)
```

```
[52, 160, 210, 174, 114, 56]
```



# Farben in Graustufen umwandeln

**Aufgabe 1**

Erstelle ein Programm, das

a) die Anzahl der unterschiedlichen Farben im Bild koenigssee.png ermittelt und ausgibt,

b) das Bild mit Hilfe der Formel für das hochauflösende Fernsehen (HDTV) in ein Graustufenbild umwandelt,

c) die Anzahl der unterschiedlichen Graustufen im umgewandelten Bild ermittelt und ausgibt


## Beispiellösung

```
from gpanel import *
makeGPanel(Size(240, 160))
bild = getImage("D:/.../koenigssee.png")
image(bild, 0, 0)

farben = []
graustufen = []
for x in range(240):
    for y in range(160):
        pixel = bild.getPixelColor(x, y)
        rot = int(pixel.red*255)
        gruen = int(pixel.green*255)
        blau = int(pixel.blue*255)
        farbePixel = [rot, gruen, blau]
        if farbePixel not in farben:
            farben.append(farbePixel)
        g = int(0.2126 * rot + 0.7152 * gruen + 0.0722 * blau)
        grau = makeColor(g, g, g)
        if g not in graustufen:
            graustufen.append(g)

print("Anzahl Farben:",len(farben))
print("Anzahl Graustufen:",len(graustufen))
```

Anzahl Farben: 31439

Anzahl Graustufen: 237

**Aufgabe 2**

a) Wie viele unterschiedliche Farben sind im RGB-Farbraum möglich?

b) Wie viele unterschiedliche Graustufen sind in einem Graustufenbild möglich?

a) Der RGB-Farbraum entsteht durch die Mischung der drei Grundfarben Rot, Grün und Blau, die jeweils in 256 Farbtönen auftreten können. Daraus ergeben sich 256 x 256 x 256 = 16 777 216 Farben.

b) Graustufenbilder enthalten für Rot, Grün und Blau denselben Wert. Dadurch sind nur 256 unterschiedliche Werte, also 256 unterschiedliche Graustufen möglich.

# Farben in Graustufen umwandeln

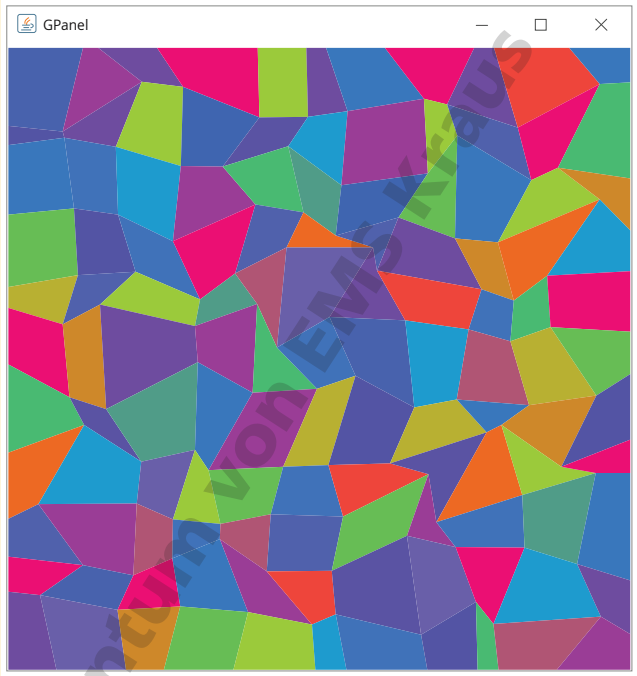
**Aufgabe 3**

a) Erstelle ein Programm, das das Bild polygon.png mit der Mittelwertmethode in ein Graustufenbild umwandelt.

b) Was fällt dir auf? Erkläre deine Beobachtung.

## Beispiellösung

```
from gpanel import *
makeGPanel(Size(500,500))
bild = getImage("D:/.../polygon.png")
image(bild, 0, 0)

for x in range(500):
    for y in range(500):
        pixel = bild.getPixelColor(x,y)
        rot = int(pixel.red*255)
        gruen = int(pixel.green*255)
        blau = int(pixel.blue*255)
        g = int((rot + gruen + blau)/3)
        grau = makeColor(g, g, g)
        bild.setPixelColor(x,y,grau)
image(bild, 0, 0)
```




Alle Farben des Polygons enthalten Farbwerte, deren Mittelwert denselben Wert ergibt. Dadurch verwandelt sich das Bild bei der Umwandlung in Graustufen in eine homogene graue Fläche.

**Aufgabe 4**

Notiere für die vier Methoden der Graustufenumwandlung jeweils drei Farben mit ihren Farbwerten, die nach der Umwandlung zu demselben Grauton werden.

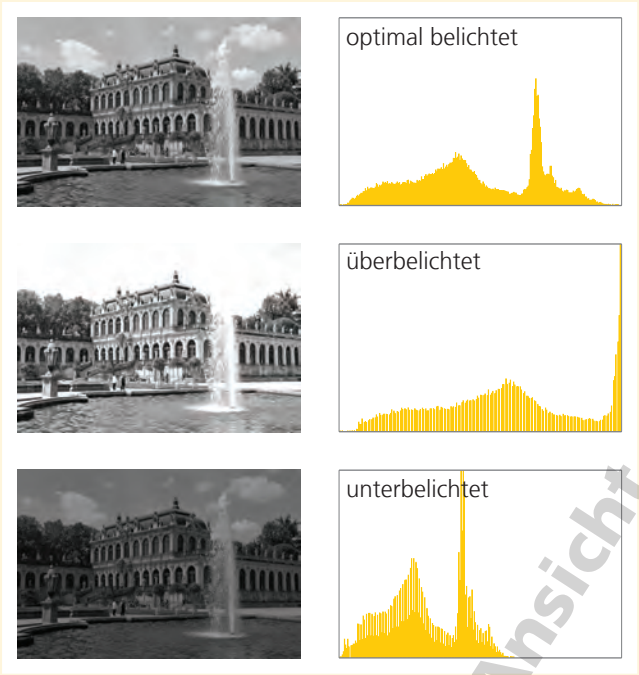
## Beispiellösung

| Methode | | rot | grün | blau | grau |
|---|---|---|---|---|---|
| kleinster Farbwert | 1 | 150 | 255 | 195 | 150 |
| | 2 | 165 | 150 | 235 | 150 |
| | 3 | 240 | 200 | 150 | 150 |
| größter Farbwert | 1 | 25 | 80 | 150 | 150 |
| | 2 | 150 | 35 | 0 | 150 |
| | 3 | 140 | 150 | 90 | 150 |
| Mittelwert | 1 | 65 | 255 | 130 | 150 |
| | 2 | 195 | 0 | 255 | 150 |
| | 3 | 240 | 115 | 95 | 150 |
| Formel HDTV | 1 | 255 | 110 | 240 | 150 |
| | 2 | 50 | 195 | 0 | 150 |
| | 3 | 130 | 145 | 255 | 150 |

# Histogramme

Um die Belichtung von Fotos zu beurteilen, nutzen Profis so genannte Histogramme. Darin ist die Häufigkeit, mit der die einzelnen Graustufen in einem Foto vorkommen, als Balkendiagramm dargestellt, vom Grauwert 0 (schwarz) ganz links bis zum Grauwert 255 (weiß) ganz rechts. Je mehr Pixel einer Graustufe es gibt, desto höher ist der jeweilige Balken.

Ist ein Foto ausgewogen belichtet, verteilen sich die Balken über die gesamte Breite, berühren aber weder links noch rechts den Rand. Das Foto enthält also keine schwarzen und rein weißen Pixel.




Das Histogramm eines überbelichteten Fotos liegt am rechten Rand an. Ein solches Foto enthält zahlreiche rein weiße Pixel. Bei einem unterbelichteten Foto sammeln sich die Balken hingegen im linken Bereich des Histogramms. Dunkle Bereiche des Fotos können auch schwarze Pixel enthalten.

Die in weißen und schwarzen Bereichen eines Bildes vorhandenen Details lassen sich auch durch nachträgliches Retuschieren nicht mehr zum Vorschein bringen.

Möchte man ein Histogramm für ein Graustufenbild erstellen, muss man zunächst eine Liste schreiben, die für jedes einzelne Pixel den Grauwert enthält. Anschließend wird gezählt, wie häufig die einzelnen Graustufen von 0 bis 255 in dieser Liste vorkommen.

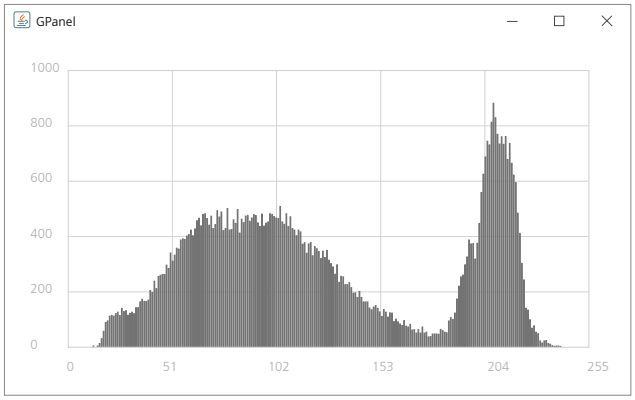
```
#Häufigkeit Graustufen zählen
histogramm = []
zaehler = 0
for n in range(256):
    for i in range(len(graustufen)):
        if graustufen[i] == n:
            zaehler = zaehler + 1
    haeufigkeit = [n,zaehler]
    histogramm.append(haeufigkeit)
    zaehler = 0
```

Das Ergebnis ist eine mehrdimensionale Liste – in unserem Beispiel heißt sie **histogramm** –, die in jeder der 256 Teillisten einen Grauwert mit seiner Häufigkeit enthält.

Diese Liste wird verwendet, um das Histogramm zu zeichnen. Im Grafikfenster wird dafür mit der Funktion **drawGrid()** ein Koordinatensystem gezeichnet. In unserem Beispiel reicht die x-Achse des Koordinatensystems von 0 bis 255, die y-Achse von 0 bis 1000, beide Achsen enthalten fünf Abschnitte, alles ist silbergrau gefärbt.

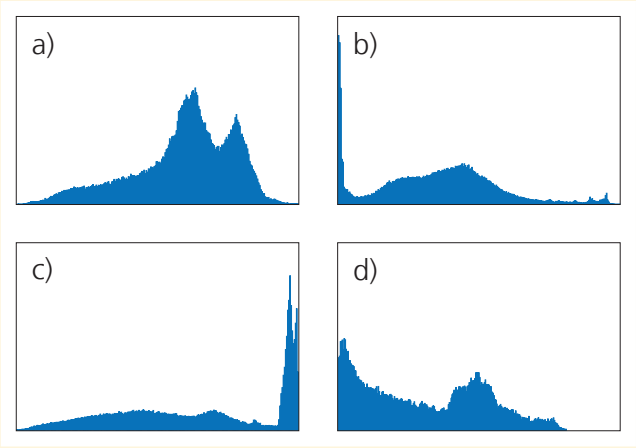
Die 256 Balken des Histogramms werden als Rechtecke in das Koordinatensystem gezeichnet. Die linke untere Ecke jedes einzelnen Rechtecks befindet sich bei x = n / y = 0. Der y-Wert der rechten oberen Ecke ist der zweite Wert der jeweiligen Teilliste aus der mehrdimensionalen Liste **histogramm**.

```
#Histogramm zeichnen
makeGPanel(-30,275,-100,1100)
drawGrid(0,255,0,1000,5,5,"silver")

setColor("DimGray")
for n in range(256):
    fillRectangle(n,0,n+1,histogramm[n][1])
```



# Histogramme

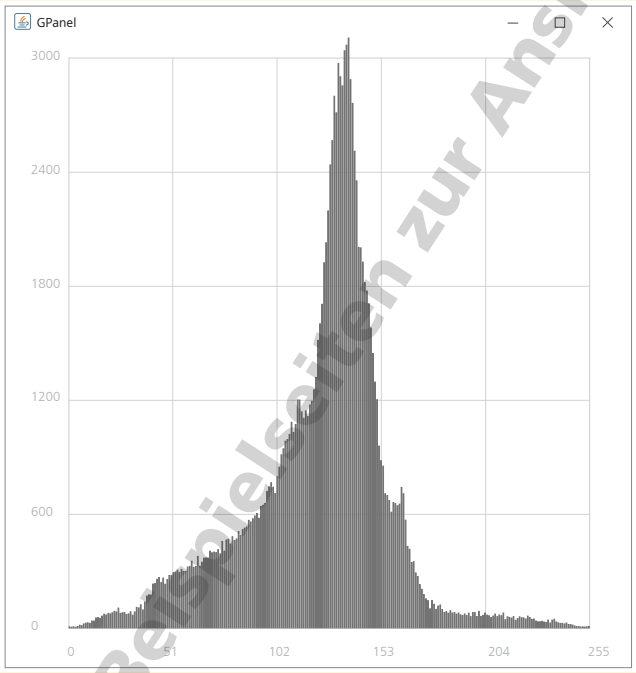
**Aufgabe 1**
Notiere für die folgenden vier Histogramme, ob die Fotos, zu denen sie gehören, unterbelichtet, ausgewogen belichtet oder überbelichtet sind.




a) ausgewogen belichtet
b) unterbelichtet
c) überbelichtet
d) unterbelichtet

**Aufgabe 2**
Erstelle ein Programm, das für das Bild moewen.png ein Histogramm zeichnet.

Probiere aus, welche Parameter du für die Höhen von Grafikfenster und Koordinatensystem einstellen musst, damit die Häufigkeitsbalken des Histogramms ins Koordinatensystem passen.




Beispiellösung

```
from gpanel import *
#Grauwerte Bild einlesen
bild = getImage("D:/.../moewen.png")
graustufen = []
for x in range(450):
    for y in range(300):
        pixel = bild.getPixelColor(x,y)
        grau = int(pixel.red*255)
        graustufen.append(grau)

#Häufigkeit Graustufen zählen
histogr = []
zaehler = 0
for n in range(256):
    for i in range(len(graustufen)):
        if graustufen[i] == n:
            zaehler = zaehler + 1
    haeufigkeit = [n,zaehler]
    histogr.append(haeufigkeit)
    zaehler = 0

#Histogramm zeichnen
makeGPanel(-30,275,-200,3100)
drawGrid(0,255,0,3000,5,5,"silver")

setColor("DimGray")
for n in range(256):
    fillRectangle(n,0,n+1,histogr[n][1])
```

# Histogramme

**Aufgabe 3**
Ermittle im Bild fachwerk.png die überbelichteten Bereiche, in denen sich weiße Pixel befinden, und färbe sie blau ein.

## Beispiellösung

```
from gpanel import *
makeGPanel(Size(400, 620))
window(0, 400, 0, 620)
bild = getImage("D:/.../fachwerk.png")
image(bild, 0, 320)

for x in range(400):
    for y in range(300):
        pixel=bild.getPixelColor(x,y)
        g=int(pixel.red*255)
        if g == 255:
            farbe=makeColor(0,114,188)
        else:
            farbe=makeColor(g,g,g)
        bild.setPixelColor(x,y,farbe)
image(bild, 0, 0)
```


**Aufgabe 4**
Ermittle im Bild burg.png die unterbelichteten Bereiche, in denen sich schwarze Pixel befinden, und färbe sie gelb ein.

## Beispiellösung

```
from gpanel import *
makeGPanel(Size(450, 620))
window(0, 450, 0, 620)
bild = getImage("D:/.../burg.png")
image(bild, 0, 320)

for x in range(450):
    for y in range(300):
        pixel=bild.getPixelColor(x,y)
        g=int(pixel.red*255)
        if g == 0:
            farbe=makeColor(255,204,0)
        else:
            farbe=makeColor(g,g,g)
        bild.setPixelColor(x,y,farbe)
image(bild, 0, 0)
```

# Histogramme

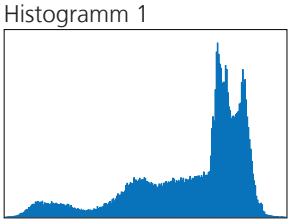
**Aufgabe 5**

Ordne diese vier Bilder und Histogramme paarweise einander zu.

Überprüfe deine Zuordnung, indem du mit dem Programm aus Aufgabe 2 Histogramme für die vier Bilder erstellst.

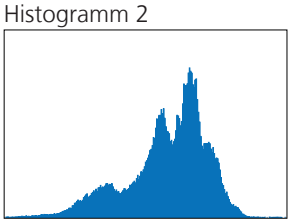
valencia.png

Histogramm 1



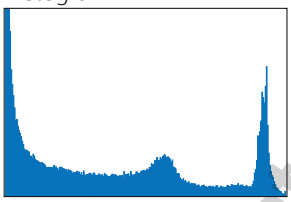
fluss.png

Histogramm 2



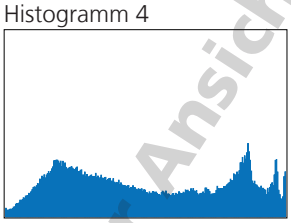
taj-mahal.png

Histogramm 3



schottland.png

Histogramm 4



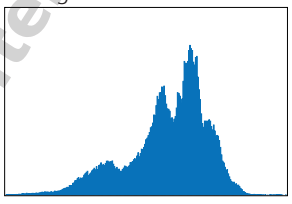
## Beispiellösung

```
from gpanel import *

#Grauwerte Bild einlesen
bild=getImage("D:/.../valencia.png")
#bild=getImage("D:/.../fluss.png")
#bild=getImage("D:/.../taj-mahal.png")
#bild=getImage("D:/.../schottland.png")
graustufen = []
for x in range(450):
    for y in range(300):
        pixel = bild.getPixelColor(x,y)
        grau = int(pixel.red*255)
        graustufen.append(grau)

#Häufigkeit Graustufen zählen
histogr = []
zaehler = 0
for n in range(256):
    for i in range(len(graustufen)):
        if graustufen[i] == n:
            zaehler = zaehler + 1
    haeufigkeit = [n,zaehler]
    histogr.append(haeufigkeit)
    zaehler = 0

#Histogramm zeichnen
makeGPanel(-30,275,-200,3100)
drawGrid(0,255,0,3000,5,5,"silver")

setColor("DimGray")
for n in range(256):
    fillRectangle(n,0,n+1,histogr[n][1])
```

## Richtige Zuordnung der Histogramme zu den Fotos

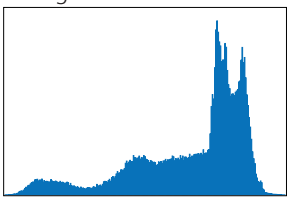
valencia.png

Histogramm 2



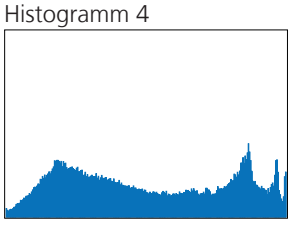
taj-mahal.png

Histogramm 1



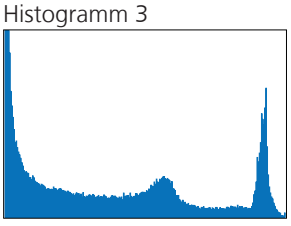
fluss.png

Histogramm 4



schottland.png

Histogramm 3



# Figuren (Actors)

Die zweite wichtige Klasse der Bibliothek ist die Klasse Actor. Von ihr werden alle Klassen abgeleitet, die das Aussehen und das Verhalten von Figuren definieren.

### Figuren erzeugen

Der Bezug unserer Klasse Apfel zur Basisklasse Actor wird hergestellt, indem Actor als Parameter in Klammern angefügt wird ❶. Bei den Funktionen, die die Klasse Apfel von der Basisklasse geerbt hat, wird die Zugehörigkeit durch das vorangestellte Actor. ausgedrückt ❸.

Die Funktion `__init__` definiert die wichtigsten Eigenschaften einer Figur. Dazu gehören beispielsweise auch der Dateiname und der Pfad der Figur.

Mit dem Parameter `self` (englisch für selbst) ❷ wird dabei festgelegt, dass sich die Funktionen auf das jeweilige Objekt selbst beziehen.

Figuren werden erzeugt, indem die Klasse `Apfel()` aufgerufen wird. Dabei kann man den Figuren auch Namen geben und die Funktion diesen Namen zuweisen ❹.

### Figuren einfügen

Im Hauptprogramm wird das Spielfenster erzeugt. Mit der Funktion `addActor()` werden hier die Figuren in das Spielfenster gesetzt.

Der Funktion werden dabei zwei Parameter mitgegeben:

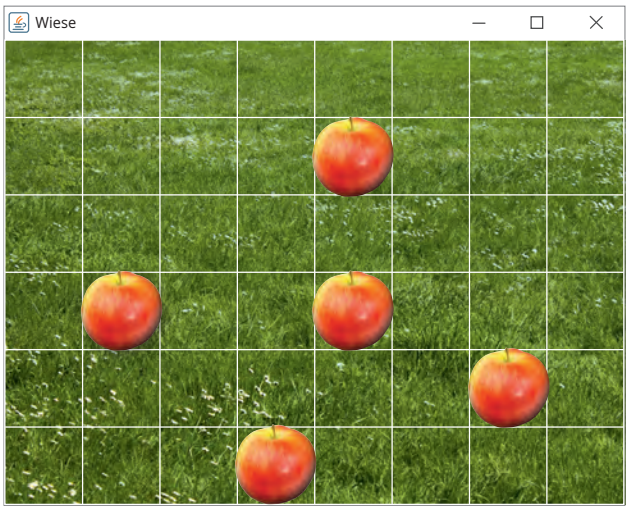
- die einzufügende Figur durch Notieren des Namens ❺ oder der Klasse `Apfel()` ❻
- die Position, also das Kästchen im Gitterraster, in dem die Figur platziert werden soll, durch Einfügen der Funktion `Location(x, y)`.

Es ist möglich, mehrere Figuren im Spielfenster zu platzieren, ohne ihnen eigene Namen zu geben ❼.

Der Parameter `x` in der Funktion `Location()` bezeichnet die horizontale und der Parameter `y` die vertikale Position. Die Kästchen werden dabei von der oberen linken Ecke ausgehend gezählt. Beim Zählen beginnt man wie üblich bei Null.

Mit der Funktion `getRandomEmptyLocation()` ❽ lassen sich Figuren auch in einem zufällig ausgewählten, leeren Kästchen platzieren.

```
from gamegrid import *

#Klasse Apfel
class Apfel(Actor):  ❶
    def __init__(self):  ❷
        Actor.__init__(self,  ❸
        "bilder/apfel.png") 1)

#Hauptprogramm
makeGameGrid(8, 6, 60, Color.white,
    "D:/.../wiese.png", False) 1)

setTitle("Wiese")

elstar = Apfel()  ❹
jonagold = Apfel()

addActor(elstar, Location(1, 3))  ❺
addActor(jonagold, Location(4, 1))
addActor(Apfel(), Location(6, 4))  ❻
addActor(Apfel(), Location(3, 5))  ❼

addActor(Apfel(),
getRandomEmptyLocation())  ❽ 1)

show()
```



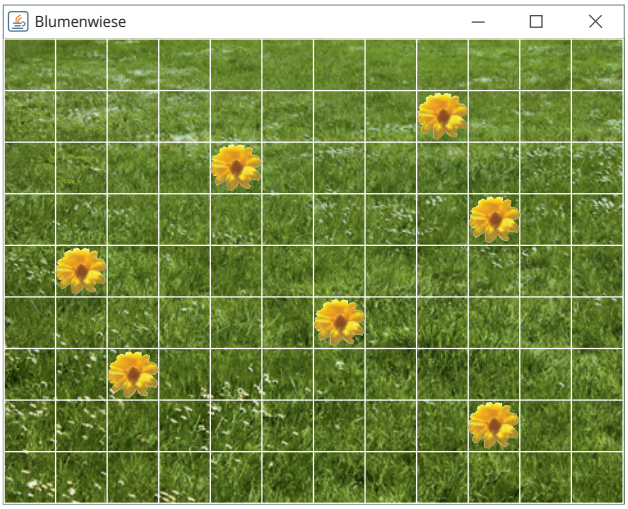



1) Zeilenumbrüche innerhalb der Programmzeilen sind hier und in den Beispielprogrammen der folgenden Lektionen durch das Layout bedingt. Sie können im Programmablauf zu Fehlermeldungen führen und sollten daher vermieden werden.

# Figuren (Actors)

**Aufgabe 1**
Erzeuge ein Spielfenster, das aussieht wie im folgenden Bild. Verwende die Bilder wiese.png und blume_gelb.png.

Platziere anschließend noch weitere fünf Blumen in zufällig gewählten leeren Kästchen.

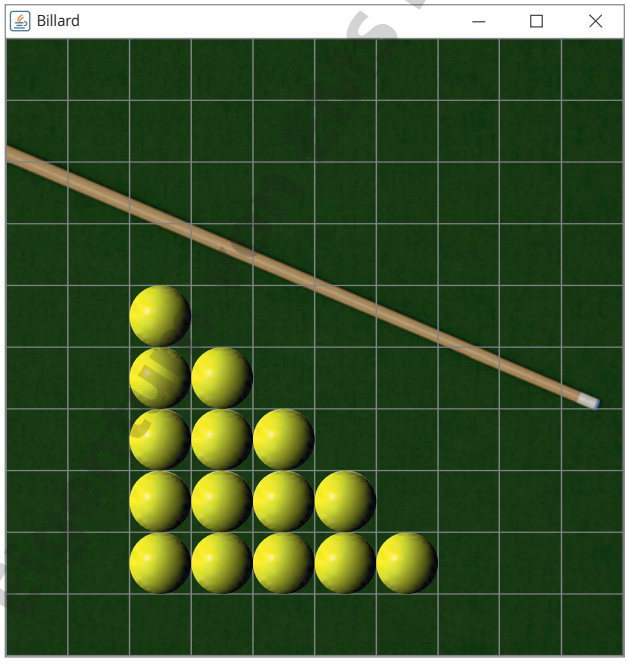



## Beispiellösung

```
from gamegrid import *

# Klasse Blume
class Blume(Actor):
    def __init__(self):
        Actor.__init__(self,
        "D:/.../blume_gelb.png")

# Hauptprogramm
makeGameGrid(12, 9, 40, Color.white,
"D:/.../wiese.png", False)
setTitle("Blumenwiese")

addActor(Blume(), Location(1, 4))
addActor(Blume(), Location(2, 6))
addActor(Blume(), Location(4, 2))
addActor(Blume(), Location(6, 5))
addActor(Blume(), Location(8, 1))
addActor(Blume(), Location(9, 3))
addActor(Blume(), Location(9, 7))

for i in range(5):
    addActor(Blume(),
    getRandomEmptyLocation())

show()
```

**Aufgabe 2**
Erzeuge ein Spielfenster, das aussieht wie im folgenden Bild. Verwende die Bilder billard.png und kugel_gelb.png.

Platziere die Kugeln mit Hilfe von for-Schleifen auf dem Spielfenster.




## Beispiellösung

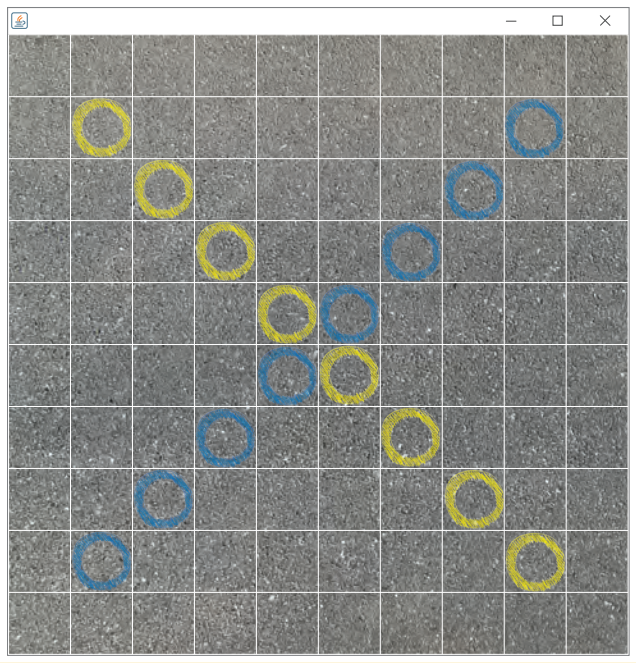
```
from gamegrid import *

# Klasse Kugel
class Kugel(Actor):
    def __init__(self):
        Actor.__init__(self,
        "D:/.../kugel_gelb.png")

# Hauptprogramm
makeGameGrid(10, 10, 60, Color.gray,
"D:/.../billard.png", False)

setTitle("Billard")

for zeile in range(4, 9):
    for spalte in range(2, zeile - 1):
        addActor(Kugel(),
        Location(spalte, zeile))

show()
```

# Figuren (Actors)

**Aufgabe 3**

Erzeuge ein Spielfenster, das aussieht wie im folgenden Bild. Verwende die Bilder asphalt.png, kringel_gelb.png und kringel_blau.png.

Platziere die Kringel mit Hilfe von for-Schleifen auf dem Spielfenster.



## Beispiellösung

```
from gamegrid import *

# Klasse KringelGelb
class KringelGelb(Actor):
    def __init__(self):
        Actor.__init__(self,
        "D:/.../kringel_gelb.png")

# Klasse KringelBlau
class KringelBlau(Actor):
    def __init__(self):
        Actor.__init__(self,
        "D:/.../kringel_blau.png")

# Hauptprogramm
makeGameGrid(10, 10, 60, Color.white,
"D:/.../asphalt.png", False)

setTitle("")

for spalte in range(1, 9):
    for zeile in range(1, 9):
        if spalte == zeile:
            addActor(KringelGelb(),
            Location(spalte, zeile))

for spalte in range(1, 9):
    for zeile in range(1, 9):
        if spalte == 9 - zeile:
            addActor(KringelBlau(),
            Location(spalte, zeile))

show()
```

**Aufgabe 4**

a) Erstelle mit einem Bildbearbeitungsprogramm ein quadratisches Hintergrundbild im Format gif, png oder jpg und füge es in ein Spielfenster ein.

b) Erstelle mit einem Bildbearbeitungsprogramm eine Figur im Format gif, png oder jpg. Sie soll so groß sein, dass sie ein Kästchen des Spielfensters ausfüllt.

c) Erzeuge ein Spielfenster mit dem Hintergrundbild und platziere darin mehrmals deine Figur.

# Tastatursteuerung und Kollision im Gitter

In Computerspielen möchte man Figuren nicht immer automatisch bewegen. Häufig sollen Spieler deren Bewegungen beeinflussen können, beispielsweise über die Tastatur.

### Tastatur-Events

Bei der Steuerung über Tastatur-Events registriert das Programm Ereignisse `(e)` wie z. B. einen Tastendruck und führt daraufhin eine so genannte Rückruffunktion (auch Callbackfunktion) aus. Die Rückruffunktion in unserem Beispiel heißt `onKeyPressed(e)` ❺ und definiert, was beim Drücken bestimmter Tasten geschehen soll.

Die Funktion muss beim Erzeugen des Spielfensters dem Parameter `keyPressed` der Funktion `makeGameGrid()` zugewiesen werden ❼.

Nach jedem Tastendruck wird abgefragt, welche Taste gedrückt wurde. In unserem Beispiel wird die Figur nach dem Drücken der Cursortasten mit `setDirection()` in die entsprechende Richtung ausgerichtet ❻. Durch den vorangestellten Namen der Figur `pin.` werden alle Anweisungen dieser Figur zugeordnet.

Damit das Programm mit dem Registrieren von Tastendrücken beginnt, wird es mit `doRun()` ❽ gestartet.

### Kollision im Gitter

Befinden sich in einem Spielfenster zwei unterschiedliche Figuren, von denen eine bewegt wird, kann es vorkommen, dass sie zusammenstoßen (kollidieren). In unserem Beispiel ist in der Funktion `platzen()` ❶ definiert, was im Falle einer Kollision mit der Seifenblase geschehen soll. Mit der Funktion `getOneActorAt()` wird dabei geprüft, ob sich im aktuellen Kästchen der Seifenblase eine Figur der Klasse Pin befindet ❷.

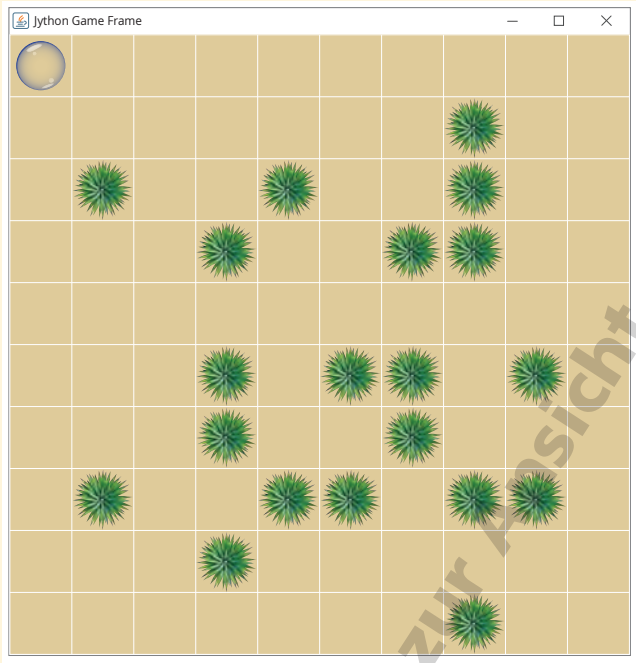
Solange sich keine Figur der Klasse Pin im Kästchen der Seifenblase befindet, gibt die Funktion `None` zurück. Sobald eine Figur der Klasse Pin in das Kästchen der Seifenblase bewegt wird, wird nicht mehr `None` zurückgegeben ❸. In diesem Fall platzt die Seifenblase, indem sie mit der Funktion `hide()` unsichtbar gemacht wird ❹.

```
from gamegrid import *

class Pin(Actor):
    def __init__(self):
        Actor.__init__(self, True,
        "D:/.../pin.png")

class Seifenblase(Actor):
    def __init__(self):
        Actor.__init__(self,
        "D:/.../seifenblase.png")
    def act(self):
        self.platzen()   ❶

    #Kollision mit Pin
    def platzen(self):   ❶
        piks = getOneActorAt(self.
        getLocation(), Pin)  ❷
        if piks != None:  ❸
            self.hide()  ❹

#Tastatur-Events
def onKeyPressed(e):  ❺
    keyCode = e.getKeyCode()
    if keyCode == 37: #nach links
        pin.setDirection(180)  ❻
    elif keyCode == 38: #nach oben
        pin.setDirection(270)
    elif keyCode == 39: #nach rechts
        pin.setDirection(0)
    elif keyCode == 40: #nach unten
        pin.setDirection(90)
    pin.move()

makeGameGrid(10, 10, 60, Color.white,
False, keyPressed = onKeyPressed)  ❼
setBgColor(230, 230, 230)

pin = Pin()
addActor(pin, Location(0, 1))
for i in range(20):
    addActor(Seifenblase(),
    getRandomEmptyLocation())

show()
doRun()  ❽
```

# Tastatursteuerung und Kollision im Gitter

**Aufgabe 1**

- Erzeuge ein Spielfenster mit hellbraunem Hintergrund und lege ein Gitterraster darüber.
- Platziere eine Figur (seifenblase.png) oben links im Spielfenster.
- Platziere anschließend 20 Kakteen (kaktus.png) in zufällig gewählte leere Kästchen.
- Die Seifenblase soll mit den Cursortasten im Spielfenster bewegt werden können.
- Sobald die Seifenblase einen Kaktus berührt, soll sie platzen (unsichtbar werden).
- Zusatzaufgabe:
  Finde im Programm heraus, was die Funktion `getOneActorAt()` liefert, wenn nicht `None` geliefert wird.




Wenn nicht None geliefert wird, liefert die Funktion getOneActorAt()

```
org.python.proxies.
__main__$Kaktus$51@66ca5a5e
```

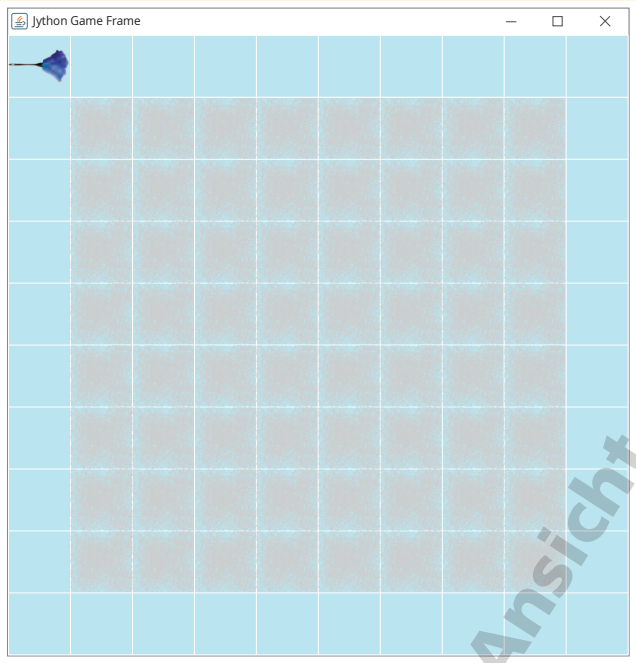
## Beispiellösung

```
from gamegrid import *

class Kaktus(Actor):
    def __init__(self):
        Actor.__init__(self, True,
        "D:/.../kaktus.png")

class Seifenblase(Actor):
    def __init__(self):
        Actor.__init__(self,
        "D:/.../seifenblase.png")
    def act(self):
        self.platzen()
    def platzen(self): #Kollision
        piks = getOneActorAt(self.
        getLocation(), Kaktus)
        if piks != None:
            self.hide()

# Tastatur-Events
def onKeyPressed(e):
    keyCode = e.getKeyCode()
    if keyCode == 37: #nach links
        bubble.setDirection(180)
    elif keyCode == 38: #nach oben
        bubble.setDirection(270)
    elif keyCode == 39: #nach rechts
        bubble.setDirection(0)
    elif keyCode == 40: #nach unten
        bubble.setDirection(90)
    bubble.move()

makeGameGrid(10, 10, 60, Color.white,
False, keyPressed = onKeyPressed)
setBgColor(230, 210, 160)
bubble = Seifenblase()
addActor(bubble, Location(0, 0))

for i in range(20):
    addActor(Kaktus(),
    getRandomEmptyLocation())

show()
doRun()
```

# Tastatursteuerung und Kollision im Gitter

**Aufgabe 2**

- Erzeuge ein Spielfenster mit hellblauem Hintergrund und lege ein Gitterraster darüber.
- Platziere eine Figur (staubwedel.png) oben links im Spielfenster.
- Platziere Figuren (staub.png) in allen inneren Kästchen, wie im folgenden Bild dargestellt.
- Der Staubwedel soll mit den Cursortasten im Spielfenster bewegt werden können.
- Sobald der Staubwedel den Staub in einem Kästchen berührt, soll dieser verschwinden (unsichtbar werden).



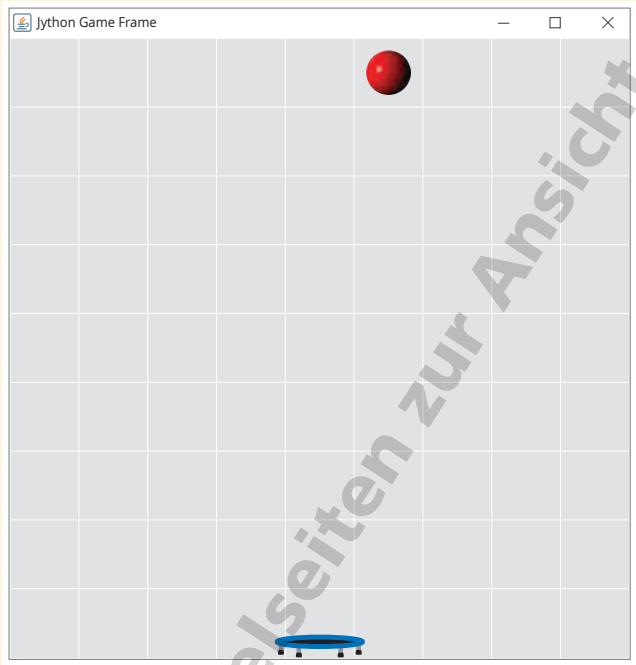

## Beispiellösung

```
from gamegrid import *

class Staubwedel(Actor):
    def __init__(self):
        Actor.__init__(self, True,
        "D:/.../staubwedel.png")

class Staub(Actor):
    def __init__(self):
        Actor.__init__(self,
        "D:/.../staub.png")
    def act(self):
        self.putzen()
    def putzen(self): #Kollision
        wedeln = getOneActorAt(self.
        getLocation(), Staubwedel)
        if wedeln != None:
            self.hide()

#Tastatur-Events
def onKeyPressed(e):
    keyCode = e.getKeyCode()
    if keyCode == 37: #nach links
        staubwedel.setDirection(180)
    elif keyCode == 38: #nach oben
        staubwedel.setDirection(270)
    elif keyCode == 39: #nach rechts
        staubwedel.setDirection(0)
    elif keyCode == 40: #nach unten
        staubwedel.setDirection(90)
    staubwedel.move()

makeGameGrid(10, 10, 60, Color.white,
False, keyPressed = onKeyPressed)
setBgColor(191, 239, 255)
staubwedel = Staubwedel()
addActor(staubwedel, Location(0, 0))

for i in range(1, 9):
    for j in range(1, 9):
        addActor(Staub(),Location(i,j))

show()
doRun()
```

# Tastatursteuerung und Kollision im Gitter

**Aufgabe 3**

- Erzeuge ein Spielfenster mit 9 x 9 Kästchen und hellgrauem Hintergrund und lege ein Gitterraster darüber.
- Platziere eine Figur (trampolin.png) unten in der Mitte des Spielfensters.
- Das Trampolin soll mit den Cursortasten nach links und rechts bewegt werden können. Dabei soll es sich nicht auf den Kopf drehen.
- Platziere eine Kugel (kugel_rot.png) an einer zufälligen Position in der obersten Kästchenreihe.
- Die Kugel soll sich nach unten bewegen.
- Sobald sie sich unten aus dem Fenster herausbewegt hat, soll sie unsichtbar werden.
- Sobald die Kugel das Trampolin berührt, soll sie ihre Richtung ändern und sich wieder nach oben bewegen.
- Sobald sie in der obersten Kästchenreihe ankommt, soll sich ihre X-Position auf einen zufälligen Wert ändern. Von dort soll sich die Kugel wieder nach unten bewegen.




## Beispiellösung

```
from gamegrid import *
from random import *
seed()

class Trampolin(Actor):
    def __init__(self):
        Actor.__init__(self, True,
        "D:/.../trampolin.png")

class Kugel(Actor):
    def __init__(self):
        Actor.__init__(self,
        "D:/.../kugel_rot.png")
    def act(self):
        self.move()
        if self.getY() == 0:
            self.setX(randint(0, 8))
            self.turn(180)
        if self.getY() > 8:
            self.hide()
        self.springen()
    def springen(self):
        spring = getOneActorAt(self.
        getLocation(), Trampolin)
        if spring != None:
            self.turn(180)

# Tastatur-Events
def onKeyPressed(e):
    keyCode = e.getKeyCode()
    if keyCode == 37: #nach links
        trampolin.setVertMirror(True)
        trampolin.setDirection(180)
    elif keyCode == 39: #nach rechts
        trampolin.setVertMirror(False)
        trampolin.setDirection(0)
    trampolin.move()

makeGameGrid(9, 9, 60, Color.white,
False, keyPressed = onKeyPressed)
setBgColor(230, 230, 230)
setSimulationPeriod(400)
trampolin = Trampolin()
addActor(trampolin, Location(4, 8))
addActor(Kugel(),
Location(randint(0, 8), 0), 90)

show()
doRun()
```