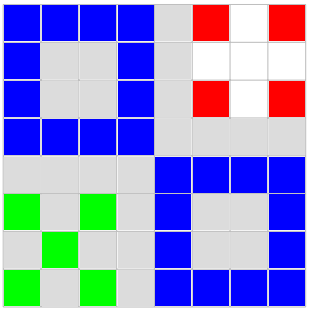


Ivo Blöchliger, KSBG

Basierend auf den Unterrichtsmaterialen
von Christan Haas, KS Heerbrugg

OxoCard

Version 1.1

30.10.2019

1 LED-Matrix steuern

LED Matrix

In diesem Kapitel lernen Sie, wie die *LED Matrix* gesteuert werden kann. Dazu werden Sie bereits bekannte Programmierstrukturen anwenden können. Somit werden Sie in der Lage sein eigene Bilder und Animationen auf der OxoCard zu erstellen. Mit *Zufallszahlen* werden wir auch zufällige Bilder erzeugen.

Zufallszahlen

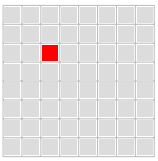
1.1 Einzelne Pixel steuern

RGB Farben

Programm 1: Eine LED auf Rot schalten.

```
from oxocard import *

dot(2,2,RED)
```

Mit `dot(2,2,RED)` wird eine LED mit der Farbe Rot eingeschaltet. Die Koordinaten $x = 2, y = 2$ stehen dabei für die 3. LED von links und von oben (das Zählen beginnt bei 0, vgl. Abb. 1).



Ausgabe von `dot(2,2,RED)`

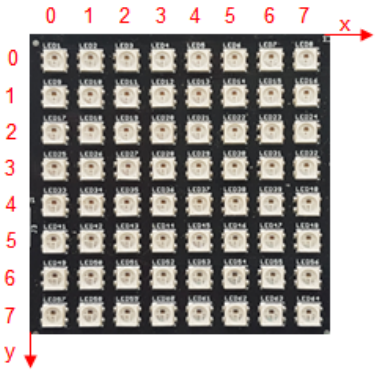


Abbildung 1 Koordinatensystem der Pixelmatrix. Quelle: tigerjython4kids.ch

`RED` ist eine durch das API bereitgestellte Konstante und steht für das Zahlentripel $(255, 0, 0)$. Die drei Zahlen stehen für den Anteil (angegeben als ganze Zahl zwischen 0 und 255) von Rot (**r**ed), Grün (**g**reen), und Blau (**b**lue). Entsprechend heisst diese Art der Farbendarstellung *RGB* -Farbmodell.

RGB

Neben RED werden einige weitere Farbtripel im OxoCard API vordefiniert bereitgestellt.

Statt `dot(2,2,RED)` könnte man also mit dem gleichen Resultat `dot(2,2,(255,0,0))` aufrufen. Beachten Sie, dass `BLACK` bedeutet, dass das entsprechende Pixel ausgeschaltet wird.

`BLACK`

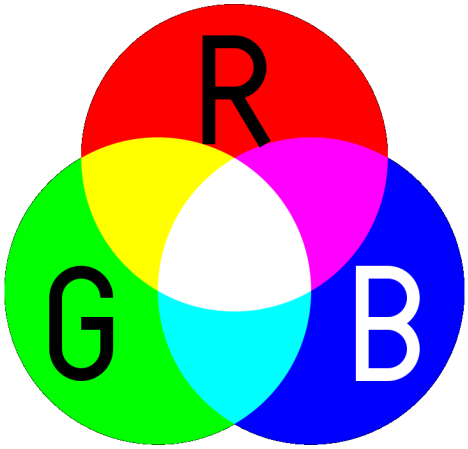


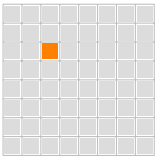
Abbildung **2** RGB-Modell: Jede Farbe lässt sich aus den Grundfarben Rot (R), Grün (G) und Blau (B) mischen. Ebenso sind die Mischfarben Yellow, Cyan und Magenta dargestellt.

Konstante	R	G	B	Tripel
RED	255	0	0	(255,0,0)
GREEN	0	255	0	(0,255,0)
BLUE	0	0	255	(0,0,255)
BLACK	0	0	0	(0,0,0)
WHITE	255	255	255	(255,255,255)
YELLOW	255	255	0	(255,255,0)
CYAN	0	255	255	(0,255,255)
MAGENTA	255	0	255	(255,0,255)

Base Colors des OxoCard APIs

Aufgabe 1.1.

Versuchen Sie durch Mischen der Farbwerte die Farbe *Orange* zu mischen. Zeichnen Sie einen entsprechenden Punkt mit der OxoCard.



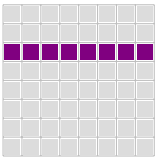
Mit 255 wird die entsprechende Farbe mit voller Leuchtkraft verwendet. Sie können ein schwächeres Rot (welches eher wie Weinrot erscheint) z.B. durch (125,0,0) erreichen.

Aufgabe 1.2.

Mischen Sie eigene Farben und malen Sie damit ein Bild.
Wenn Sie die gleiche Farbe mehrmals verwenden möchten, lohnt es sich diese zunächst zu speichern (vgl. Prog. 2).

Programm 2: Eigene Farbe

```
from oxocard import *

Violett=(128,0,128)
for i in range(0,8):
        dot(i,2,Violett)
        sleep(0.5)
```



Ausgabe von `Eigene Farbe`

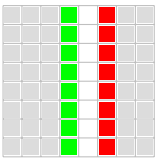
`sleep()`

`sleep(0.5)` bewirkt, dass zwischen dem Malen der einzelnen Punkte 0.5 Sekunden gewartet wird. [1]

Aufgabe 1.3.

Schreiben Sie eine Funktion `senkrechterStrich(i,c)`, welche als Parameter eine Spaltennummer `i` und eine Farbe `c` (als RGB Tripel) erwartet und einen senkrechten Strich in der i–ten Zeile zeichnet.
Rechts sehen Sie das Ausgabebild nach den Aufrufen:

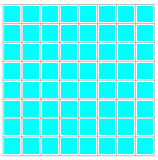
```
senkrechterStrich(3,GREEN)
senkrechterStrich(4,WHITE)
senkrechterStrich(5,RED)
```



Aufgabe 1.4.

Schreiben Sie eine Funktion `fuelle(c)`, welche alle LEDs mit der Farbe c leuchten lässt.
Tipp: Nutzen Sie die Funktion `senkrechterStrich(i,c)` aus der vorherigen Aufgabe. Rechts sehen Sie die Ausgabe nach dem Aufruf:

```
fuelle(CYAN)
```



In der obigen Aufgabe haben Sie, sofern Sie dem Tipp gefolgt sind, die Funktion `fuelle(c)` mit einer Schleife mit jeweiligem Aufruf der Funktion `senkrechterStrich(i,c)` gelöst. Selbstverständlich wäre dies auch in nur einer Funktion möglich. Dann haben wir zwei verschachtelte **`for`**-Schleifen.

Programm 3: Fuelle2

```
from oxocard import *

def fuelle2(c):
    for i in range(0,8):
        for j in range(0,8):
            dot(i,j,c)
```

[1] `sleep()` ist eine Funktion des `time` Moduls, welches vom Modul `oxocard` importiert wird.

```
fuelle2(CYAN)
```

Der Autor des OxoCard APIs [2] hat daran gedacht, dass man wohl häufig den ganzen Bildschirm entweder löschen oder mit einer Farbe füllen möchte. Daher steht der Befehl `clear(c)` im Modul `oxocard` zur Verfügung, welcher bei Aufruf ohne Parameter `clear()` alle LEDs ausschaltet. Bei Aufruf mit einer Farbe (z.B. `clear(BLUE)` wird alles auf diese Farbe gesetzt.
Wenn man sich den Programmcode dazu ansieht, so erkennt man, dass im Wesentlichen genau das gleiche wie bei unserer eigenen Funktion gemacht wird.

`clear(c)`

Aufgabe 1.5.

Lassen Sie das Programm 3 mal laufen und nachher ein (sonst bis auf den «import» leeres Programm) mit clear(CYAN). Fällt Ihnen ein kleiner Unterschied auf?

In unserem Programm wird im Gegensatz zu `clear(CYAN)` nach jeder Pixeländerung das Bild neu gezeichnet (*gerendert*). Dadurch wird nicht das ganze Display auf einmal in der gewünschten Farbe gefüllt.

Rendering

Wir können dies Unterbinden indem wir explizit sagen, wann das neu erstellte Bild gezeichnet wird. Dies ist mit dem Befehl `repaint()` möglich. Damit nicht mehr automatisch nach jeder Änderung die LED Matrix gerendert wird, müssen wir dies zudem deaktivieren `enableRepaint(False)`. Das API Modul `oxocard` merkt sich alle Änderungen, schickt nun aber die neue Farbkonstellation erst nach dem Befehl `repaint()` runter zu Micropython, welches dafür sorgt, dass der Mikrocontroller die LEDs umschaltet. Die entsprechend ergänzte Füllfunktion sieht dann so aus:

`repaint()`

Programm 4: Fuelle3

```
from oxocard import *

enableRepaint(False)
def fuelle3(c):
    for i in range(0,8):
        for j in range(0,8):
            dot(i,j,c)
    repaint()

fuelle3(CYAN)
```

[2] Aegidius Plüss, aplu.ch

1.2 Eigene Objekte zeichnen

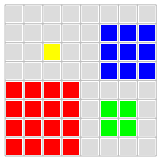
Wir haben bereits gesehen, wie wir einzelne Pixel steuern können. Wir wollen nun eigene Objekte programmieren.

Rechteck Zunächst wollen wir Quadrate und Rechtecke zeichnen. Dabei ist es hilfreich, dass das `oxocard`-API mögliche Fehler für uns ausbügelt. Eigentlich sollte ein Aufruf der Form `dot(10,10,RED)` einen Fehler produzieren, da es kein Pixel an der Stelle $(10, 10)$ gibt. Unser Programm stürzt in so einem Fall dennoch nicht ab, da im API überprüft wird, ob das Pixel existiert. Gibt es das Pixel nicht, wird die Eingabe ignoriert. [3]

Aufgabe 1.6.

Schreiben Sie mithilfe verschachtelter **for**-Schleifen (vgl. 3) eine Funktion `quadrat(x,y,s,c)`, welches an der Stelle (x, y)[a] ein Quadrat der Seitenlänge s in der Farbe c zeichnet. Ein Ausgabebeispiel sehen Sie rechts.
Aufrufe:

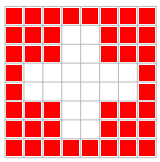
```
quadrat(0,4,4,RED)
quadrat(5,1,3,BLUE)
quadrat(5,5,2,GREEN)
quadrat(1,2,1,YELLOW)
```



[a] (x,y) soll für die Koordinaten der linken oberen Ecke stehen

Aufgabe 1.7.

Zeichnen Sie mithilfe der Quadratfunktion und `clear(c)` ein Schweizer Kreuz. Nutzen Sie `repaint()` (vgl. 4), um das Bild erst zu Zeichnen, wenn alle Pixel gesetzt wurden, also wenn das Schweizer Kreuz fertig ist.



Schweizer Kreuz

Ganz ähnlich wie Quadrate lassen sich auch Rechtecke zeichnen, man braucht nun offensichtlich zwei Parameter für die Grösse (Breite und Höhe).

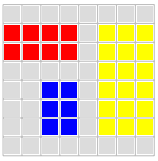
Aufgabe 1.8.

Schreiben Sie mithilfe verschachtelter **for**-Schleifen (vgl. 3) eine Funktion `rechteck(x,y,b,h,c)`, welches an der Stelle (x, y) ein Rechteck mit der Breite b und Höhe h in der Farbe c zeichnet. Ein Ausgabebeispiel sehen Sie rechts.
Aufrufe:

```
rechteck(0,1,4,2,RED)
```

[3] Noch besser wäre es natürlich, wenn wir eine Warnung erhalten würden: Pass auf, du willst Pixel (10,10) steuern, ich habe aber nur 64 Pixel, also nur bis zu (7,7).

```
rechteck(2,4,2,3,BLUE)
rechteck(5,1,3,6,YELLOW)
```



Aufgabe 1.9.

Zeichnen Sie mithilfe der Funktion `rechteck(x,y,b,h,c)` einige Länderflaggen.

Sie können eine kleine Flaggenanimation gestalten, indem Sie nacheinander Ihre Länderflaggen zeichnen lassen und dazwischen mit `sleep(1)` kleine Pausen einbauen (statt 1 kannst du beliebige Sekundenwerte für die Pause angeben). Dazu ist es hilfreich die einzelnen Flaggen als separate Funktionen zu definieren.

Programm 5: Flaggen

```
(...)
enableRepaint(False)
while True:
        frankreich()
        repaint()
        sleep(1)
        schweiz()
        repaint()
        sleep(1)
        italien()
        repaint()
        sleep(1)
```

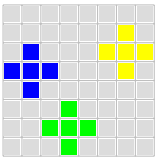
Aufgabe 1.10.

Erstellen Sie eigene Rechteckmuster und lassen Sie diese nacheinander in einer Animation auf der OxoCard abspielen.

Kreuz Wir möchten nun ein kleines Kreuz als Funktion definieren.

Aufgabe 1.11.

Schreiben Sie eine Funktion `kreuz(x,y,c)`, welche ein Kreuz mit Zentrum (x, y) zeichnet. Das Kreuz soll immer die gleiche Grösse (5 Pixel) haben. Ausgabebeispiel rechts: `kreuz(3,6,GREEN) kreuz(1,3,BLUE) kreuz(6,2,YELLOW)`



Linie Nun wollen wir noch eine Funktion programmieren, welche Linien für uns zeichnet.

Aufgabe 1.12.

Wie ist eine Linie (Strecke) eindeutig festgelegt? Überlegen Sie sich zwei Arten, wie eine Linie allgemein definiert werden kann!

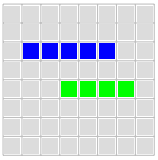
Sehr wahrscheinlich haben Sie sich folgende zwei Varianten überlegt.

- Eine Linie ist eindeutig festgelegt durch Startpunkt, Länge und Richtung.
- Eine Linie ist eindeutig festgelegt durch Start- und Endpunkt.

Beide Varianten lassen sich programmieren. In den Richtungen sind wir mit der OxoCard natürlich etwas eingeschränkt. Wir wollen zunächst einen Strich nach Rechts programmieren.

Aufgabe 1.13. *(`rechtsStrich`)*

Programmieren Sie eine Funktion `rechtsStrich(x,y,l,c)`, welche startend in (x, y) einen Strich nach rechts mit Länge l in der Farbe c zeichnet. Rechts sehen Sie die Ausgabe zu:
`rechtsStrich(3,4,3,GREEN)`
`rechtsStrich(1,2,4,BLUE)`



Ein Informatikprofessor an der ETH hat einmal gesagt, dass es als Informatiker gut ist, faul zu sein... Was macht also ein fauler (aber schlauer!) Informatiker bei dieser Aufgabe? ... Er nutzt die Rechtecksfunktion von 1.2 und schreibt:

```
def rechtsStrich(x,y,l,c):
        rechteck(x,y,l,1,c)
```

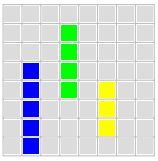
Fertig!
Wenn also die Funktion `rechtsStrich` aufgerufen wird, so wird an der gewünschten Stelle mit der angegebenen Farbe ein Rechteck mit Höhe 1 gezeichnet. Ähnlich wie oben kann man auch einen Strich nach links, oben und unten programmieren. Diagonale Striche sind ebenfalls nicht viel komplizierter.
Als Beispiel ist hier noch die Funktion `obenStrich` angegeben, welche einen

Strich nach oben zeichnet. Beachte, dass die **for**-Schleife immer nach oben läuft, d.h. wir müssen die tiefste y–Koordinate zunächst berechnen.

```
def obenStrich(x,y,l,c):
    for j in range(y-l+1,y+1):
        dot(x,j,c)
```

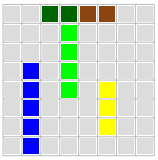
Rehts die Ausgabe zu:

```
obenStrich(3,4,3,GREEN)
obenStrich(1,7,4,BLUE)
obenStrich(5,6,2,YELLOW)
```



Aufgabe 1.14.

Zeichnen Sie eigene Bilder, Sie sehen rechts ein Beispiel. Nutzen Sie die definierten Funktionen!
Farben und die RGB Werte finden Sie z.B. hier http://www.markusbader.de/tricky/rgb_blau.html.



2 Buttons

BUTTONS

In diesem Kapitel lernen Sie, wie Sie die Oxocard-Tasten (*Buttons*) verwenden können. Damit werden Sie in der Lage sein, interaktive Programme zu entwickeln, d.h. Programme, bei welchen der Benutzer durch Drücken von Tasten den Verlauf steuern kann.
Zwei unterschiedliche Programmierarten begegnen uns in diesem Kapitel, *Polling* und *Events* (*Callbacks*).

POLLING

EVENTS

CALLBACKS

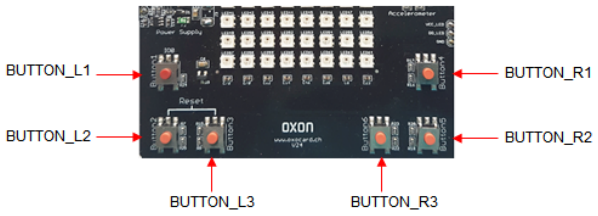


Abbildung **3** Die Buttons der Oxocard. Quelle: tigerjython4kids.ch

2.1 Polling

Unter Polling[4] versteht man die regelmässige Abfrage eines Status von Soft- oder Hardware. Bei den Buttons können wir z.B. abfragen, ob er gedrückt wird oder nicht. Unser Programm kann also den Status des Buttons abfragen und je nach Zustand (gedrückt oder nicht gedrückt) ein unterschiedliches Verhalten zeigen.

OXOBUTTON

Damit wir einfach mit Buttons arbeiten können, liefert uns das API mit dem Modul *oxobutton* ein Objekt `Button`[5]. Die Klasse `Button` aus dem Modul `oxobutton` enthält einige hilfreiche Funktionen[6]. So können wir für einen von uns im Programm erstellten Button jeweils abfragen, ob er gerade gedrückt wird oder ob er seit der letzten Abfrage gedrückt wurde.

Wiederum ist das Modul `oxobutton` ein Bindeglied zwischen unserer höheren Programmiersprache und der Hardware. Sie können das Modul selber erkunden und versuchen zu erkennen, was es genau macht. Du findest es im Installationsordner von TigerJython unter *bin>Lib>oxocardmodules*. Es übersetzt z.B. unsere Buttonbezeichnung (`Button_L1` usw.) in die entsprechende Nummer des Anschlusspins, an welchem der Button am Mikrocontroller angeschlossen ist.

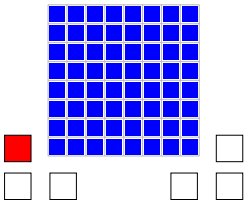
Für unser Programm erzeugen wir ein Objekt `Button` beispielsweise mittels `button1=Button(BUTTON_L1)`. Nun steht `button1` für den linken oberen Button.

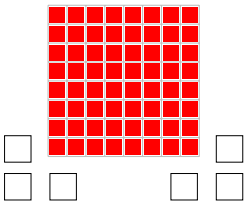
Methode `isPressed()` Die einfachste Verwendung ist nun abzufragen, ob der Button gedrückt wird und in einer Verzweigung je nachdem den einen oder anderen Programmteil auszuführen. Die Methode `isPressed()` gibt `True` zurück, falls der Button gerade gedrückt wird. Ansonsten gibt sie `False` zurück.
Wir betrachten zunächst ein einfaches Beispiel
Das folgende Beispielprogramm durchläuft die **`while`** Schleife unendlich oft (denn die Bedingung `True` ist immer erfüllt). In jedem Durchlauf wird bei unserem Button `linksOben` die Methode `isPressed()` aufgerufen. Die Syntax dazu lautet `linksOben.isPressed()`. Wir geben das Objekt an und nach dem Punkt die Methode die aufgerufen werden soll.

[4] to poll (engl.): befragen, abstimmen
[5] man spricht von einer Klasse
[6] Funktionen aus einer Klasse Objekt nennen wir auch Methoden

Programm 6: Button Beispiel

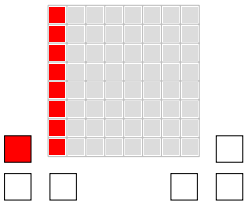
```
from oxocard import *
from oxobutton import *

TasteLinksOben=Button(BUTTON_L1)

while True:
    if TasteLinksOben.isPressed():
        clear(BLUE)
    else:
        clear(RED)
```

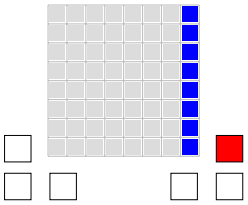

Button gedrückt


Button nicht gedrückt

Aufgabe 2.1.

Schreiben Sie ein Programm `linksrechts()`, welches bei Drücken der linken oberen Taste einen roten senkrechten Strich am linken Rand und bei Drücken der rechten oberen Taste rechts einen blauen senkrechten Strich zeichnet. Wird nichts gedrückt, soll der Bildschirm ausgeschaltet (`clear(BLACK)`) werden. Für das Zeichnen des Striches können Sie die Funktion `senkrechterStrich(c)` von letztem Kapitel verwenden. Nutzen Sie auch `repaint()` (vgl. letztes Kapitel) damit jeweils erst das Resultat gezeichnet wird.


Button L1 gedrückt


Button R1 gedrückt

Drücken Sie doch mal bei Ihrem Programm abwechslungsweise in schneller Abfolge links und rechts. Fällt Ihnen etwas auf?

Wenn alles geklappt hat, werden Sie feststellen, dass teilweise trotz Drücken der Taste der Balken nicht erscheint. Dies liegt daran, dass das Programm nur genau dann, wenn `isPressed()` aufgerufen wird, überprüft, ob die Taste gedrückt wird. Nun läuft aber die **`while`**-Schleife ständig durch und braucht jeweils einige Zeit für die anderen Befehle.
So kann es passieren, dass das Klicken „verpasst “wird.

Methode `wasPressed()` Um das Problem des „Verpassens“zu beheben, stellt das Modul `oxobutton` die Methode `wasPressed()` zur Verfügung. Diese gibt `True` zurück, falls seit dem letzten Aufruf von `wasPressed()` der Button gedrückt wurde. Um die Unterschiede zu erkennen betrachten wir das folgende Beispiel:

Programm 7: is was

```
from oxocard import *
```

```
from oxobutton import *

TasteLinksOben=Button(BUTTON_L1)
TasteRechtsOben=Button(BUTTON_R1)
enableRepaint(False)
while True:
    clear(BLACK)
    if TasteLinksOben.wasPressed():
        clear(RED)
    if TasteRechtsOben.isPressed():
        clear(BLUE)
    repaint()
    sleep(0.1)
```

Es wird jeweils überprüft ob links gedrückt **wurde** bzw. ob rechts gedrückt **wird**. Am Ende jedes `while`-Durchgangs wird 0.1 Sekunden pausiert. Wie du erkennst, funktioniert das Drücken der linken Taste zuverlässig, das Drücken der rechten Taste jedoch nicht.

Aufgabe 2.2.

Schreiben Sie ein eigenes Programm mit Buttonsteuerung. Sie können selber wählen, welche Button, was bewirken sollen.

2.2 Events - Callback

Im vorherigen Abschnitt haben wir gesehen, wie man mit `isPressed()` und `wasPressed()` jeweils abfragen kann, ob eine Taste gedrückt wird bzw. wurde. Dieses laufende Nachfragen führt natürlich zu einem hohen Aufwand, zudem könnte es sein, das wir ein Klicken verpassen (wenn wir nicht gerade nachfragen) oder verspätet bemerken (wenn andere Programmteile durchlaufen werden).

EVENTS

CALLBACK-FUNKTIONEN

Für derartige Fälle gibt es die Programmiertechnik von *Events* und *Callback-funktionen* . Wir können eine Funktion definieren und dem System sagen, unter welchen Umständen diese aufgerufen werden soll.
Wir betrachten ein Beispiel, bei welchem ständig zufällige rote Punkte gezeichnet werden und durch Klicken wieder gelöscht werden sollen. Wir müssen dem Mikrocontroller (via `oxobutton`) also mitteilen: „Hey, wenn links oben gedrückt wird, dann starte neu!“

Programm 8: Callback

```
from oxocard import *
from oxobutton import *

def dasPassiertBeimKlick(pin):
```

```
    clear(BLACK)

TasteLinksOben=Button(BUTTON_L1, dasPassiertBeimKlick)

while True:
    x=randint(0,7)
    y=randint(0,7)
    dot(x,y,RED)
    sleep(0.1)
```

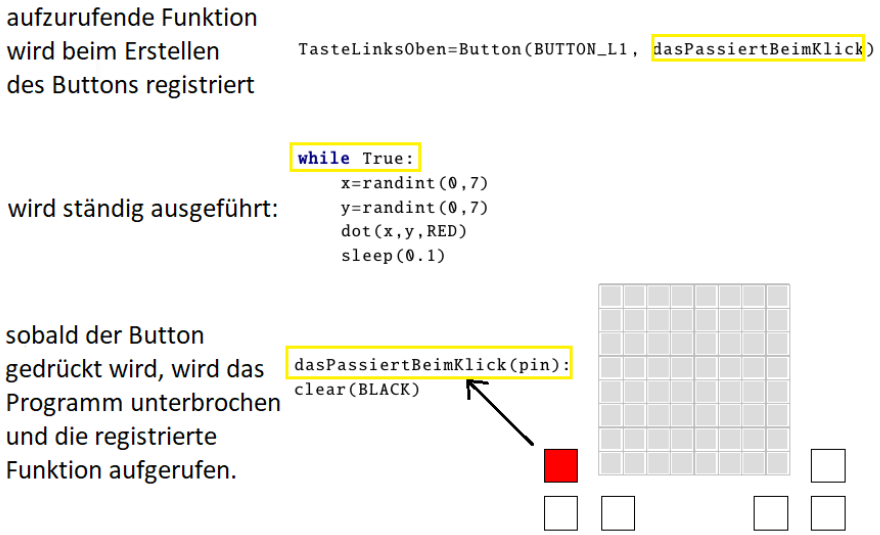


Abbildung **4** Das Programm Callback schematisch erklärt.

ZUFALLSZAHLEN

Im Beispiel werden *Zufallszahlen* genutzt, `randint(0,7)` erzeugt eine Zufallszahl zwischen 0 und 7. Wenn wir den Button erstellen, registrieren wir den Callback mit dem zweiten Parameter `dasPassiertBeimKlick`. Nun weiss der Mikrocontroller, dass beim Event *Klicken der linken oberen Taste* die Callbackfunktion `dasPassiertBeimKlick` aufgerufen werden soll. Wurde nun die Taste gedrückt (beachten Sie, dass die Taste kurz gedrückt gehalten werden muss), so wird die Funktion `dasPassiertBeimKlick(pin)` aufgerufen, welche dafür sorgt, dass der Hintergrund schwarz wird.[7]

ZUFALLSFARBEN

Mit Zufallszahlen lassen sich auch *Zufallsfarben* erzeugen. Wenn wir auch noch Buttons nutzen, können wir schöne Programme schreiben. In folgendem Beispiel werden jeweils zufällige Leds in zufälliger Farbe erleuchtet. Mit der linken Taste lässt sich das Bild löschen, mit der rechten pausieren.

[7] Die Callback-Funktion braucht einen Parameter `pin`, denn sie erhält vom System als Übergabewert die Nummer des Anschlusspins des gedrückten Buttons. So können mehrere Buttons dieselbe Callbackfunktion nutzen und wir unterscheiden in der Funktion je nach Pin, was geschehen soll. Selbstverständlich können verschiedene Buttons auch separate Callbackfunktionen nutzen.

Methode	Polling	Callback
Eigenschaft	Zyklische Abfrage des Zustandes eines Buttons	Rückruffunktion, welche bei Drücken des Buttons aufgerufen wird.
Vorteile	Programmablauf ist übersichtlich. Einzelne Abfrage braucht nur wenig Zeit	Ressourcenschonend, die Funktion wird nur bei Drücken aufgerufen.
Nachteile	Andauerndes Abfragen verschlingt viel Rechenleistung und macht das Programm langsam. Drücken von Buttons kann verpasst werden	Programmablauf wird allenfalls zu einem unerwünschten Zeitpunkt unterbrochen.

Tabelle 1: Vergleich von Polling und Callback

Programm 9: Party

```
from oxocard import *
from oxobutton import *

def beiKlickLinks(pin):
    clear(BLACK)

def beiKlickRechts(pin):
    sleep(2)

linksOben=Button(BUTTON_L1, beiKlickLinks)
rechtsOben=Button(BUTTON_R1, beiKlickRechts)
while True:
    r=randint(0,255)
    g=randint(0,255)
    b=randint(0,255)
    x=randint(0,7)
    y=randint(0,7)
    dot(x,y,(r,g,b))
```

Aufgabe 2.3.

Schreiben Sie ein Programm **buttonpunkte**, welches nach Drücken des linken, oberen Buttons einen zufälligen Punkt in zufälliger Farbe zeichnet.

Im nächsten Beispiel wird ein Punkt durch Buttons gesteuert. Wir speichern jeweils die **x**-Koordinate. Bei Druck des Buttons links unten wandert ein weisser Punkt nach links. Bei Druck des Buttons links unten wandert ein weisser Punkt nach rechts.

GLOBAL

In diesem Programm kommt neu der Begriff **global** vor. Indem wir in einer Funktion vor Nutzung einer Variable diese mit dem Schlüsselwort **global** angeben (im Programm **global x**) wird die Variable ausserhalb der Funktion verändert (und nicht etwa eine lokale Kopie). So können wir im Programm die ausserhalb der Callback Funktion definierte Variablen **x** innerhalb der Callback Funktionen verändern.

Programm 10: Weisser Punkt waagrecht

```
from oxocard import *
from oxobutton import *

def zeichnePunkt():
    global x
    dot(x,3,WHITE)

def rechts(pin):
    global x
    x = x + 1
    clear()

def links(pin):
    global x
    x = x - 1
    clear()

taste_L = Button(BUTTON_L2, links)
taste_R = Button(BUTTON_R2, rechts)

x = 0

while True:
    zeichnePunkt()
```

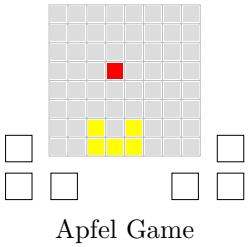
Ganz ähnlich lässt sich ein Punkt in zwei Dimensionen steuern:

Programm 11: Weisser Punkt waagrecht und senkrecht

```
def zeichnePunkt():
    global x
    global y
    dot(x,y,WHITE)

def rechts(pin):
    global x
    x = x + 1
```

```
    clear()

def links(pin):
    global x
    x = x - 1
    clear()

def oben(pin):
    global y
    y = y - 1
    clear()

def unten(pin):
    global y
    y = y + 1
    clear()

tasteL = Button(BUTTON_R3, links)
tasteR = Button(BUTTON_R2, rechts)
tasteO = Button(BUTTON_L1, oben)
tasteU = Button(BUTTON_L2, unten)

x = 0
y = 0

while True:
    zeichnePunkt()
```

Zum Abschluss betrachten wir noch ein kleines Game Beispiel. Zugegebenerweise ist die Steuerung etwas mühsam...



Apfel Game

Programm 12: Aepfel einsammeln mit Korb

```
from oxocard import *
from oxobutton import *

def beiKlickLinks(pin):
    global x
    x=x-1

def beiKlickRechts(pin):
    global x
    x=x+1
```

```
def korb(x):
    dot(x-1,7,YELLOW)
    dot(x,7,YELLOW)
    dot(x+1,7,YELLOW)
    dot(x+1,6,YELLOW)
    dot(x-1,6,YELLOW)

linksOben=Button(BUTTON_L1, beiKlickLinks)
rechtsOben=Button(BUTTON_R1, beiKlickRechts)

x=4
xApfel=randint(0,7)
yApfel=0
enableRepaint(False)
while True:
    clear(BLACK)
    korb(x)
    yApfel=yApfel+1
    dot(xApfel, yApfel, RED)
    repaint()
    if yApfel==6:
        if xApfel==x:
            sleep(0.5)
            clear(GREEN)
            repaint()
            sleep(1)
        else:
            sleep(0.5)
            clear(RED)
            repaint()
            sleep(1)
        xApfel=randint(0,7)
        yApfel=0
    sleep(0.3)
```

Aufgabe 2.4.

Schreiben Sie ein eigenes Programm mit Buttonsteuerung. Das kann ein kleines Spiel oder etwas anderes sein.