elektor

DAS MAGAZIN FÜR DIE RASPBERRY PI COMMUNITY

Das offizielle **RASPBERRY PI** Magazin

MagPi

Sonderheft Nr. 1 | magpi.de

Das offizielle **Raspberry Pi** Magazin

GUIs PROGRAMMIEREN

GRAFISCHE BENUTZER-OBERFLÄCHEN ERSTELLEN MIT PYTHON

60 SEITEN PRAXIS SONDERHEFT

März/April 2022 Nr. 2 | 9,95 €
Österreich 11,50 €
BeNeLux 11,50 €
Schweiz 19,50 Sfr

4 198283 109957 02

Das offizielle MagPi Magazin

über Raspberry Pi

12 MONATE
EIN PREIS
54,95 €

- **6 X MAGPI : GEDRUCKTE AUSGABE**
- **ZUGANG ZUM MAGPI ONLINE-ARCHIV**

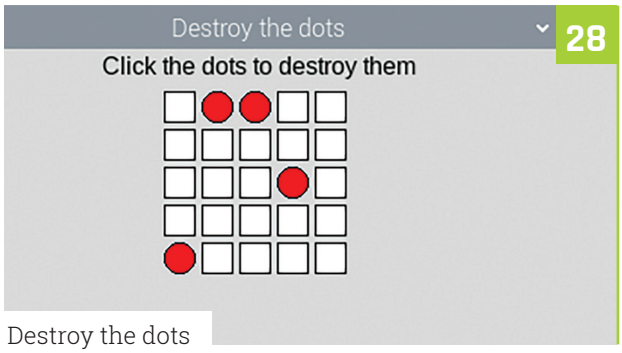
JETZT BESTELLEN AUF
WWW.MAGPI.DE/ABO

MagPi
www.magpi.de Magazine

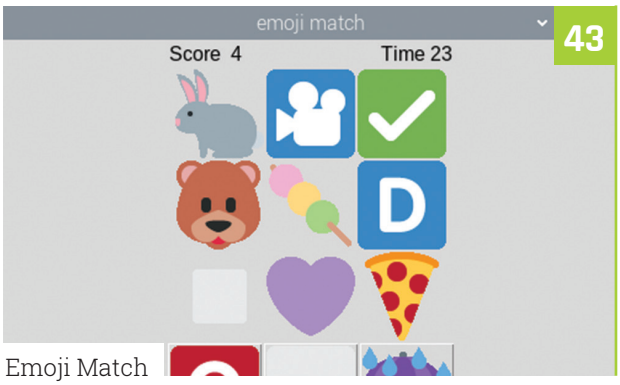
Inhalt

GUIs erstellen mit Python

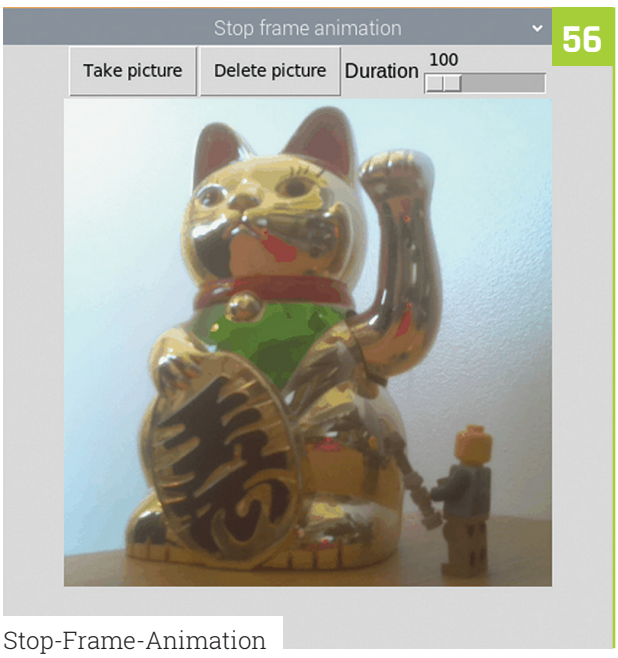
Nickname-Generator



Destroy the dots



Emoji Match



Stop-Frame-Animation

Teil 01

Grafische Benutzeroberflächen mit Python und guizero

Installieren Sie die Python-Bibliothek guizero und erstellen Sie Ihre eigenen grafischen Benutzeroberflächen.

MAKER
Laura Sach

Laura leitet das A-Level-Team bei der Raspberry Pi Foundation und erstellt Ressourcen für Studenten, die etwas über Informatik lernen wollen.

@CodeBoom

MAKER
Martin O'Hanlon

Martin arbeitet im Lernteam der Raspberry Pi Foundation, wo er Online-Kurse, Projekte und Lehrmaterial erstellt.

@martinohanlon

Mit einer grafischen Benutzeroberfläche (GUI) lassen sich Python-Programme leichter bedienen und interessant gestalten. Sie können verschiedene Komponenten, sogenannte „Widgets", zu Ihrer Oberfläche hinzufügen, um Informationen in das Programm einzugeben und auf dem Bildschirm darzustellen. In dieser Serie wenden wir uns der guizero-Bibliothek zu, die gerade auch für Anfänger gut geeignet ist.

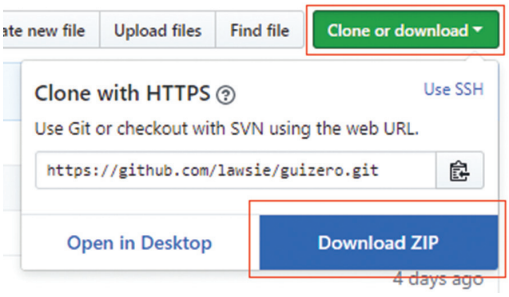
Das Standard-GUI-Paket von Python heißt *tkinter*, ist auf den meisten Plattformen bereits mit Python installiert und kann mit der guizero-Bibliothek auf einfache Weise genutzt werden.

01 Installation von guizero

Um die gezeigten Programme zu erstellen, müssen Sie die Python-Bibliothek guizero (**lawsie.github.io/guizero**) installieren. Sie ist als Python-Paket verfügbar, also als wiederverwendbarer Code, den Sie herunterladen, installieren und dann in Ihren Programmen verwenden können.

Wie Sie guizero installieren, hängt von Ihrem Betriebssystem und Ihren Administratorrechten ab.

Wenn Sie Zugriff auf die Kommandozeile oder ein Terminal haben, können Sie den folgenden Befehl verwenden:



▶ Eine alternative Möglichkeit, guizero zu installieren, besteht darin, die Zip-Datei von GitHub herunterzuladen.

```
pip3 install guizero
```

Unter **lawsie.github.io/guizero** finden Sie umfassende Installationsanweisungen für guizero und Optionen für die Installation, falls Sie keine Administrator-Rechte **für Ihren Computer haben**. Sie finden dort auch Installationen für Windows zum Download.

02 Hello World

Wenn Sie guizero installiert haben, überprüfen Sie, ob es funktioniert: Schreiben Sie eine kleine „Hello World"-App, die traditionell als erstes Programm beim Ausprobieren einer neuen Sprache geschrieben wird.

Öffnen Sie den Editor, in welchem Sie IhrenPython-Code schreiben werden. Am Anfang eines jeden guizero-Programms wählen Sie die benötigten Widgets aus der guizero-Bibliothek aus und importieren sie diese. Sie brauchen jedes Widget nur einmal zu importieren und können es dann beliebig oft in Ihrem Programm verwenden.

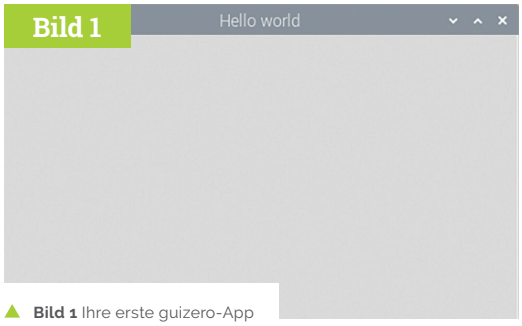
Fügen Sie oben auf der Seite den folgenden Code ein, um das App-Widget zu importieren:

```
from guizero import App
```

Alle guizero-Projekte beginnen mit einem Hauptfenster (*App* gnannt), das ein Container-Widget ist. Am Ende jedes guizero-Programms müssen Sie dem Programm sagen, dass es die App anzeigen soll, die Sie gerade erstellt haben.

Fügen Sie diese beiden Codezeilen unterhalb der Zeile ein, in der Sie das App-Widget importiert haben:

```
app = App(title="Hello world")
app.display()
```



▲ **Bild 1** Ihre erste guizero-App

Speichern Sie nun Ihren Code und führen Sie ihn aus. Sie sollten ein GUI-Fenster mit dem Titel „Hello World“ sehen (**Bild 1**). Herzlichen Glückwunsch, Sie haben soeben Ihre erste guizero-App erstellt!

03 Widgets

Widgets sind die Dinge, die auf der Benutzeroberfläche erscheinen, wie z. B. Textfelder, Schaltflächen, Schieberegler und sogar einfache Textstücke.

Alle Widgets werden zwischen der Codezeile zum Erstellen der App und der Zeile `app.display()` eingefügt. Hier ist die App, die Sie gerade erstellt haben, aber in diesem Beispiel haben wir ein Text -Widget hinzugefügt:

```
from guizero import App, Text
app = App(title="Hello world")
message = Text(app, text="Welcome to the
app")
app.display()
```

Haben Sie bemerkt, dass es zwei Änderungen gibt (**Bild 2**)? Es gibt jetzt eine zusätzliche Codezeile, um das Text-Widget hinzuzufügen, und wir haben auch Text zur Liste der zu importierenden Widgets in der ersten Zeile hinzugefügt.

Schauen wir uns den Code des Text-Widgets etwas genauer an:

```
message = Text(app, text="Welcome to the
app")
```

Wie jede Variable in Python braucht auch ein Widget einen Namen. Dieser heißt hier „message“. Dann geben wir an, dass dies ein „Text“-Widget sein soll.

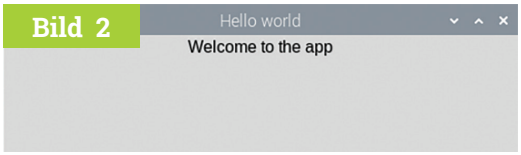
Innerhalb der Klammern stehen einige Parameter, die das Aussehen des Text-Widget bestimmen. Der erste Parameter, „app“, teilt dem Text mit, wo sich das Widget befindet. Alle Widgets müssen sich innerhalb eines Container-Widgets befinden. Die meiste Zeit werden Ihre Widgets direkt in einer App enthalten sein, aber es gibt, wie Sie noch sehen werden, auch einige andere Arten von Container-

01-helloworld.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

▶ Sprache: **Python 3**

```
001.  <from guizero import App, Text
002.  app = App(title="Hello world")
003.  message = Text(app, text="Welcome to the app")
004.  app.display()
```



◀ **Bild 2** Fügen Sie eine Textnachricht hinzu.

Widgets, in die Sie Dinge hinein stecken können. Schließlich weisen wir das Widget an, den Text „Willkommen in der App“ darzustellen.

Wir programmieren ein Fahndungsplakat

01 Schriftart- und Farbenvielfalt

Nachdem Sie nun eine Benutzeroberfläche erstellen können, wollen wir sie etwas interessanter gestalten. Sie können Text in verschiedenen Schriftarten, Größen und Farben hinzufügen, die Hintergrundfarbe ändern und auch Bilder hinzufügen. Um das alles zu üben, wollen wir ein „Wanted“-Plakat nach traditioneller Wild-West-Manier erstellen.

Als Erstes müssen Sie eine App erstellen. Fügen Sie in Ihrem Editor diesen Code ein, um ein einfaches App-Fenster zu erstellen:

```
from guizero import App

app = App("Wanted!")

app.display()
```

Wenn Sie den Code speichern und ausführen, sollten Sie eine Grafik sehen, die wie ein einfaches graues Quadrat mit dem obenstehenden Titel „Wanted!“ aussieht (**Bild 3**).

02 Hintergrundfarben

Lassen Sie uns den Hintergrund der App ein wenig ändern. Fahndungsplakate sehen meist so aus, als seien sie aus vergilbtem Pergament. Ändern wir die Hintergrundfarbe daher in Blassgelb.

Suchen Sie die Code-Zeile, in der Sie die App

Alles, was Sie brauchen

- Einen Computer (z. B. Raspberry Pi, AppleMac, Windows- oderLinux-PC)
- Internetverbindung
- Python 3 (**python.org**)
- Eine IDE (Code-Editor), z. B.: IDLE (wird mit Python 3 installiert), Thonny (**thonny.org**), Mu (**codewith.mu**),PyCharm (**jetbrains.com/pycharms**)
- Die guizero Python library (**lawsie.github.io/guizero**)

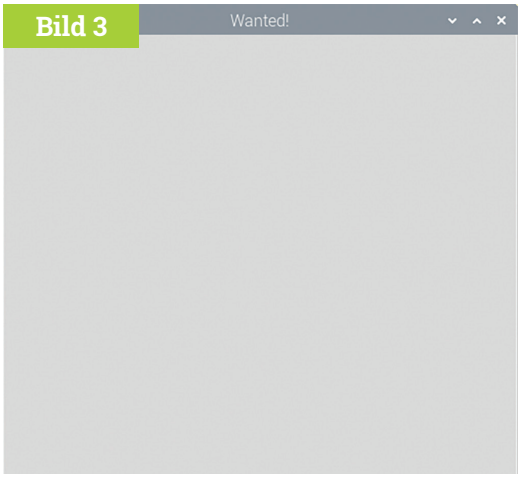


Bild 3
Die Basis-App.

erstellen. Unmittelbar nach dieser Codezeile fügen Sie eine weitere Codezeile ein, um die Eigenschaft `bg` (*background*) des Fensters zu ändern. Mit dieser Eigenschaft können wir die Farbe des Hintergrunds ändern:

```
from guizero import App

app = App("Wanted!")
app.bg = "yellow"

app.display()
```

Dies wird als Editieren einer Eigenschaft bezeichnet. Im Code müssen Sie das Widget angeben, um das es geht (app), die Eigenschaft, die Sie ändern möchten (bg) und den Wert, auf den Sie sie ändern möchten.

Vielleicht ist diese Farbe (**Bild 4**) ein bisschen zu knallig. Lassen Sie uns daher den Hex-Code einer helleren gelben Farbe nachschlagen. Es gibt viele Websites, auf denen Sie nach Farben suchen können, zum Beispiel auf **htmlcolorcodes.com** (**Bild 5**).

Wenn Sie die gewünschte Farbe ausgewählt

Bild 4 Ändern der Hintergrundfarbe.

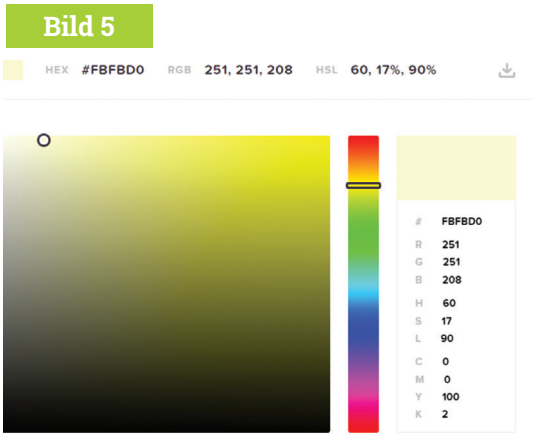


Bild 5 Auswählen eines Farbtons auf **htmlcolorcodes.com**

> bg ist die Abkürzung für 'background' und erlaubt uns, die Farbe des Hintergrunds zu ändern

haben, wird ihr Code auf der Website entweder als Hexadezimalwert (in diesem Fall #FBFBD0) oder als RGB-Wert (251, 251, 208) angezeigt. Sie können beide Formate zum Einstellen von Farben in *guizero* verwenden:

```
app.bg = "#FBFBD0"
app.bg = (251, 251, 208)
```

03 Hinzufügen von Text

Ihre App sollte ungefähr so aussehen wie in **Bild 6**. Fügen wir nun etwas Text zur GUI hinzu: Den typischen Fahndungsplakat-Schriftzug: „Wanted"!

guizero-Dokumentation

Auch wenn Sie Programmieranfänger sind, können Sie die volle Leistungsfähigkeit von guizero und allen anderen Bibliotheken, denen Sie begegnen, nutzen.

Die guizero-Dokumentation finden Sie unter **lawsie.github.io/guizero**. Wenn Sie zum Beispiel mehr über das Ändern von Widget-Properties erfahren möchten, klicken Sie dort auf das Widget, das Sie ändern möchten, und scrollen Sie nach unten, bis Sie den Abschnitt mit den Eigenschaften erreichen. Wenn Sie z. B. "Text" unter der Überschrift "Widgets" auswählen, werden alle editierbaren Eigenschaften eines Textes angezeigt. Die Dokumentation enthält oft hilfreichen Code, der Ihnen zeigt, wie Sie eine bestimmte Eigenschaft oder Methode verwenden können.

Bild 6

▲ **Bild 6** Blassgelber Hintergrund

Suchen Sie zunächst nach der Codezeile, in der Sie die App bereits importiert haben.

```
from guizero import App
```

Um einen Text erstellen zu können, fügen Sie „Text“ Ende der Liste an. Jetzt sieht die Zeile wie folgt aus:

```
from guizero import App, Text
```

Jedes Mal, wenn Sie einen neuen Widget-Typ verwenden möchten, fügen Sie dessen Namen an das Ende der Liste an. Es gibt keinen Grund, ständig neue Codezeilen hinzuzufügen: Bleiben Sie einfach bei einer Liste, damit Ihr Programm nicht zu unübersichtlich wird.

Da Sie nun Text verwenden können, fügen wir ein Stück Text hinzu. Denken Sie daran, dass alle Widgets auf der GUI zwischen der Codezeile, in der Sie die App erstellen, und der Codezeile, in der Sie sie anzeigen, eingefügt werden müssen. Ihr Code sollte jetzt so aussehen:

```
from guizero import App, Text

app = App("Wanted!")
app.bg = "#FBFBD0"

wanted_text = Text(app, "WANTED")

app.display()
```

Schauen wir uns die gerade hinzugefügte Code-Zeile einmal genauer an.

```
wanted_text = Text(app, „WANTED“)
```

Hier ist `wanted_text` der Name des Textes, damit wir später im Code darüber sprechen können – stellen Sie sich das wie den Namen einer Person vor. (Sie können Ihr Textstück auch *Dave* nennen – dem Computer ist das egal!)

Innerhalb der Klammern haben wir zwei Ausdrücke. Der zweite, **WANTED**, ist einfach, da es der Text ist, den wir auf dem Bildschirm anzeigen möchten. Das erste ist der Container, der dieses Stück Text steuert und *Master* genannt wird. In diesem Fall sagen wir, dass dieser Text von der App gesteuert werden soll. Wenn Sie zum ersten Mal mit dem Erstellen von GUIs beginnen, werden die meisten Ihrer Widgets die App als ihren *Master* haben, aber es gibt noch andere Container, die Widgets speichern können.

04 Änderung von Textgröße und -Farbe

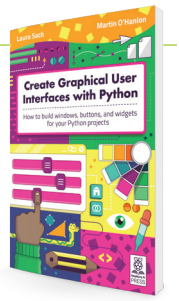
Die Buchstaben sind ziemlich klein (**Bild 7**). Ändern wir die Eigenschaft `text_size` auf genau dieselbe Weise, wie wir es getan haben, als wir die Hintergrundfarbe der App geändert haben. Erinnern Sie sich, dass Sie drei Dinge angeben mussten:

1. Den Namen des Widgets
2. Welche Eigenschaft geändert werden soll
3. Der neue, geänderte Wert

In diesem Fall geben Sie also das Widget (`wanted_size`), die zu ändernde Eigenschaft (`text_size`) und den neuen Wert (`50`) an. Fügen Sie eine neue Codezeile direkt unter der Zeile ein, in der Sie den Text erstellt haben, um die Eigenschaft zu ändern.

Bild 7

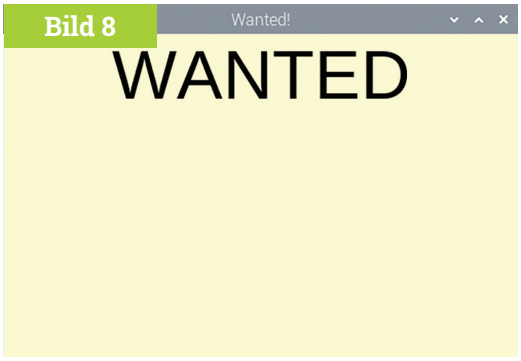
◀ **Bild 7** Der Text ist zu klein.



Grafische Benutzeroberflächen mit Python erstellen

Weitere Anleitungen zum Erstellen eigener GUIs mit guizero finden Sie in unserem neuen Buch *Create Graphical User Interfaces with Python*. Auf 156 Seiten finden Sie alle wichtigen Informationen und eine Reihe von interessanten Projekten. **magpi.cc/pythongui**

Bild 8



▶ **Bild 8** Größerer Text

```
wanted_text = Text(app, "WANTED")
wanted_text.text_size = 50
```

Sie haben jetzt einen Text mit größeren Buchstaben auf Ihrem Plakat (**Bild 8**). Versuchen Sie nun, die Schriftart dieses Textes zu ändern. Welche Schriftarten zur Verfügung stehen, hängt davon ab, welches Betriebssystem Sie verwenden, daher hier einige Vorschläge:

- **Times New Roman**
- **Verdana**
- **Courier**
- **Impact**

Kein Fahndungsplakat wäre vollständig ohne ein Bild, also fügen wir eines hinzu. **Bild 9** zeigt die Katze des Autors, weil sie ständig an Möbeln und Tapeten kratzt.

Speichern Sie eine Kopie des Bildes, das Sie verwenden möchten, im gleichen Ordner wie Ihr GUI-Programm. Sie können Bilder in anderen Ordnern verwenden, aber dann müssen Sie den Pfad zum Bild angeben, daher ist es viel einfacher, sie am Anfang einfach im gleichen Ordner zu speichern.

▶ **Bild 9** Das vollständige Poster.

02-wanted.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:

magpi.cc/guizerocode

▶ Sprache: **Python 3**

```
001.  from guizero import App, Text, Picture
002.
003.  app = App("Wanted!")
004.  app.bg = "#FBFBD0"
005.
006.  wanted_text = Text(app, "WANTED")
007.  wanted_text.text_size = 50
008.  wanted_text.font = "Times New Roman"

009.  cat = Picture(app, image="tabitha.png")
010.
011.  app.display()
```

Denken Sie daran, dass Widges immer am Anfang des Programms importiert werden müssen. Dann folgt ein Widget mit einem sinnvollen Namen nach der Codezeile, in der Sie die App erstellen, aber vor der abschließenden `app.display()`-Zeile.

Fügen Sie ‚Picture' zur Liste der Widgets hinzu, die beim Programmstart importiert werden sollen.

```
from guizero import App, Text, Picture
```

Erstellen Sie nun ein Picture-Widget mit zwei Parametern: der App und dem Dateinamen des Bildes. Dies ist der Code, den wir verwendet haben, weil unser Bild **tabitha.png** heißt.

```
cat = Picture(app, image="tabitha.png")
```

Führen Sie Ihren Code, der dem auf der letzten Seite ganz unten (**02-wanted.py**) ähneln sollte, erneut aus. Das Bild unter Ihrem Text wird jetzt angezeigt (**Bild 9**).

Jetzt können Sie Ihre neu erworbenen Kenntnisse nutzen, um Ihre grafischen Oberflächen nach Ihren Wünschen zu gestalten.

Bild 9

Bildbearbeitung

Da guizero eine Bibliothek für Einsteiger ist, enthält sie keine ausgefeilten Bildbearbeitungsfunktionen. Dazu ist eine zusätzliche Bibliothek namens "pillow" erforderlich. Sie können nicht-animierte GIF-Bilder auf jeder Plattform und PNG-Bilder auf allen Plattformen außer Mac verwenden. Wenn Sie also nicht sicher sind, ob Sie die zusätzlichen Bildbearbeitungsfunktionen installiert haben, bleiben Sie bei diesen Bildtypen.

Teil 02

Programmieren in Python

Nickname-Generator mit grafischer Benutzeroberfläche.

In Teil 1 dieses Tutorials haben Sie gelernt, wie Sie Ihre grafische Oberfläche mit einer Vielzahl von Optionen ausstatten können. Jetzt ist es Zeit, in den wirklich interaktiven Teil einzusteigen und eine Anwendung zu erstellen, die auf Ihre Eingaben reagiert. Wie wäre es zum Beispiel mit einem Button, der beim Anklicken einen Nickname erzeugt? Solche Phantasienamen werden häufig verlangt, wenn Sie sich irgendwo im Internet anmelden müssen – und oft fällt einem ausgerechnet dann kein brauchbarer Name ein.

Zum Programmieren der entsprechenden Software muss man nur ein einfaches Fenster erstellen und etwas Text hinzufügen. Der hier gezeigte Code enthält auch einige Kommentare; Zeilen, die mit einem # beginnen und den Umgang mit dem Programm vereinfachen.

Führen Sie den folgen Code aus. Sie sollten das in **Bild 1** gezeigte Fenster sehen

```
# Imports ---------------
from guizero import App, Text

# Functions -------------

# App -------------------
app = App("TOP SECRET")

# Widgets ---------------
title = Text(app, "Push the red button to find out your spy name")

# Display ---------------
app.display()
```

Schaltfläche hinzufügen

Bringen wir nun eine Schaltfläche (Button) auf unserer grafischen Oberfläche (GUI) an. Fügen Sie dazu ***PushButton*** zu Ihrer Liste der Importe hinzu. Schreiben Sie unterhalb des Text-Widgets, aber vor der Anzeige der App, folgende Codezeile:

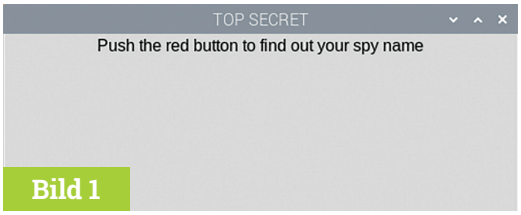


Bild 1 Anzeige des Textes in einem Fenster.

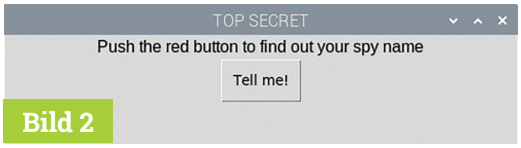


Bild 2 Sie haben jetzt eine Schaltfläche programmiert.

```
button = PushButton(app, choose_name, text="Tell me!")
```

Ihr Code sollte nun demjenigen im Programm **spy1.py** entsprechen. Beim Ausführen wird jedoch noch keine Schaltfläche erscheinen, sondern nur eine Fehlermeldung im Shell-Fenster:

```
NameError: name 'choose_name' is not defined
```

Das liegt daran, dass `choose_name` eine Art Befehl zum Aufruf einer Funktion mit gleichem Namen ist, die ausgeführt wird, wenn die Schaltfläche gedrückt wird. Diese Funktion existiert jedoch noch nicht. Die meisten GUI-Komponenten können mit einem Befehl verknüpft werden. Für eine Schaltfläche bedeutet dies: "Wenn die Schaltfläche gedrückt wird, führe diesen Befehl aus."

Erstellen Sie eine Funktion

Lassen Sie uns daher nun die Funktion `choose_name` schreiben, damit Ihre Schaltfläche etwas bewirken kann, wenn sie gedrückt wird.

Schauen Sie sich Ihr Programm an und suchen Sie den Abschnitt ***functions***. Hier sollten alle Funktionen stehen, die an GUI-Widgets angehängt werden, um sie vom Code für die Anzeige des Widgets getrennt zu halten. Fügen Sie folgenden Code in den Funktionsabschnitt ein:

```
def choose_name():
    print("Button was pressed")
```

Ihr Code sollte nun wie der Inhalt der Datei **spy2.py** aussehen. Wenn Sie die nun erscheinende Schaltfläche anklicken (**Bild 2**), sieht es so aus, als sei nichts passiert, aber in Ihrem Ausgabefenster (**Bild 3**) ist ein Text erschienen.

Für einen ersten Test genügt ein beliebiger Dummy-Text. Sie können dann die Anweisung zur Ausgabe des Textes später durch den eigentlichen Code für die Aufgabe ersetzen, die Ihre Schaltfläche

ausführen soll.

Geben Sie innerhalb Ihrer Funktion `choose_name` ein #-Symbol vor der Codezeile ein, die den Text "Button was pressed" ausgibt. Programmierer nennen dies "Auskommentieren". Sie sagen dem Computer damit, dass er diese Codezeile wie einen Kommentar behandeln soll, oder mit anderen Worten, sie ignorieren soll. Der Vorteil dieser Methode: Wenn Sie den Code jemals wieder verwenden wollen, können Sie ihn einfach wieder in Ihr Programm einbauen, indem Sie das #-Symbol entfernen.

spy1.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Text, PushButton
003.
004.  # Functions -------------
005.
006.  # App -------------------
007.  app = App("TOP SECRET")
008.
009.  # Widgets ---------------
010.  title = Text(app, "Push the red button to find out your spy name")
011.  button = PushButton(app, choose_name, text="Tell me!")
012.
013.  # Display ---------------
014.  app.display()
```

spy2.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Text, PushButton
003.
004.  # Functions -------------
005.
006.  # App -------------------
007.  app = App("TOP SECRET")
008.
009.  # Widgets ---------------
010.  title = Text(app, "Push the red button to find out your spy name")
011.  button = PushButton(app, choose_name, text="Tell me!")
012.
013.  # Display ---------------
014.  app.display()
```

Große rote Schaltfläche

Im Moment ist Ihre Schaltfläche weder groß noch Rot! Sie haben im ersten Teil dieser Tutorial-Serie Eigenschaften verwendet, um das Aussehen Ihres Textes auf dem "Wanted"-Poster zu ändern. Können Sie nun auch die Eigenschaften des *PushButton*-Widgets verwenden, um die Hintergrundfarbe und die Textgröße zu ändern?

Beachten Sie, dass es unter Umständen nicht möglich ist, die Farbe einer Schaltfläche unter macOS zu ändern, da einige Versionen des Betriebssystems dies nicht zulassen, aber Sie sollten auf jeden Fall in der Lage sein, die Textgröße zu ändern.

▲ **Bild 3** Der Text wird im Shell-Fenster ausgegeben.

Hinzufügen von Namen

Fügen Sie in einer neuen Zeile eine Liste mit Vornamen ein. Sie können die Namen in Ihrer Liste frei wählen und es können so viele Namen sein, wie Sie möchten, aber stellen Sie sicher, dass jeder Name zwischen Anführungszeichen steht und die Namen jeweils durch ein Komma getrennt sind. Eine Sammlung von Buchstaben, Zahlen und/oder Interpunktionszeichen zwischen Anführungszeichen wird als String bezeichnet. Jeder Name muss also ein String sein.

```
first_names = ["Barbara", "Woody",
"Tiberius", "Smokey", "Jennifer", "Ruby"]
```

Fügen Sie nun auch eine beliebig lange Liste mit frei gewählten, phantasievollen Nachnamen hinzu:

```
last_names = ["Spindleshanks", "Mysterioso",
"Dungeon", "Catseye", "Darkmeyer",
"Flamingobreath"]
```

Nun brauchen wir eine Möglichkeit, um einen zufälligen Namen aus jeder Liste auszuwählen, um einen aus Vor- und Nachname bestehenden Nickname zu erzeugen. Ihre erste Aufgabe besteht darin, eine neue Import-Zeile in Ihrem Import-Abschnitt hinzuzufügen:

```
from random import choice
```

Damit teilen Sie dem Programm mit, dass Sie eine Funktion namens ***choice*** verwenden möchten, die ein zufälliges Element aus einer Liste auswählt. Jemand hat bereits den Code geschrieben, der dies für Sie erledigt, und er steht in Python zu Ihrer Verfügung.

Fügen Sie in Ihrem Code für die Funktion `choose_name` direkt unter den Namenslisten eine Codezeile ein, die den Vornamen Ihres Nickname auswählt und ihn dann mit dem Nachnamen verkettet, mit einem Leerzeichen dazwischen. Verketten bedeutet, dass zwei Zeichenketten miteinander verbunden werden. Das Symbol (nicht nur) in Python für Verkettung ist ein Plus (+).

```
spy_name = choice(first_names) + " " +
choice(last_names)
print(spy_name)
```

Bild 4

▲ **Bild 4** Ausgeben eines Nickname.

Ihr Code sollte nun demjenigen **spy3.py** ähneln. Speichern Sie ihn und führen Sie ihn aus. Wenn Sie die Taste drücken, sollten Sie sehen, dass ein zufällig generierter Name in Ihrer Konsole oder Shell erscheint, und zwar an der gleichen Stelle, an der vorher die ursprüngliche Meldung "Button was pressed" erschien (**Bild 4**).

Name erscheint auf GUI

Das ist gut, aber wäre es nicht schöner, wenn der Nickname auch auf der grafischen Oberfläche erscheinen würde? Lassen Sie uns ein weiteres Text-Widget erstellen und es zur Anzeige des Namens verwenden.

Fügen Sie im Bereich ***Widgets*** ein neues Text-Widget hinzu, das den Namen anzeigen soll:

```
name = Text(app, text="")
```

Natürlich soll der Text nicht schon erscheinen, bevor der Button angeklickt wird. Also können Sie den Text auf `""` setzen, was als 'leerer String' bezeichnet wird. Kommentieren Sie innerhalb Ihrer Funktion `choose_name` die Codezeile aus, in der Sie den Nickname ausgeben.

Fügen Sie nun eine neue Codezeile am Ende der Funktion ein, um den Wert des ***name*** Text-Widgets auf den gerade erstellten `spy_name` zu setzen. Dies bewirkt, dass das Text-Widget sich selbst aktualisiert und den Namen anzeigt.

```
name.value = spy_name
```

Ihr endgültiger Code sollte wie der Code der Datei ***03-spy-name-chooser.py*** aussehen. Führen Sie das Programm aus und drücken Sie die Schaltfläche, um Ihren per Zufall ermittelten Nickname auf der grafischen Benutzeroberfläche angezeigt zu bekommen (**Bild 5**).

Wenn Ihnen der ermittelte Name nicht gefällt, können Sie den Vorgang beliebig oft wiederholen und sich immer wieder neue Namen vorschlagen lassen.

Bild 5

▲ **Bild 5** Das fertige Nickname-Wahlmenü.

spy3.py

▶ Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Text, PushButton
003.  from random import choice
004.
005.  # Functions -------------
006.  def choose_name():
007.      #print("Button was pressed")
008.      first_names = ["Barbara", "Woody", "Tiberius", "Smokey",
      "Jennifer", "Ruby"]
009.      last_names = ["Spindleshanks", "Mysterioso", "Dungeon",
      "Catseye", "Darkmeyer", "Flamingobreath"]
010.      spy_name = choice(first_names) + " " + choice(last_names)
011.      print(spy_name)
012.
013.  # App -------------------
014.  app = App("TOP SECRET")
015.
016.  # Widgets ---------------
017.  title = Text(app, "Push the red button to find out your spy name")
018.  button = PushButton(app, choose_name, text="Tell me!")
019.  button.bg = "red"
020.  button.text_size = 30
021.
022.  # Display ---------------
023.  app.display()
```

03-spy-name-chooser.py

▶ Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text, PushButton
004.  from random import choice
005.
006.  # Functions -------------
007.
008.  def choose_name():
009.      #print("Button was pressed")
010.      first_names = ["Barbara", "Woody", "Tiberius", "Smokey",
      "Jennifer", "Ruby"]
011.      last_names = ["Spindleshanks", "Mysterioso", "Dungeon",
      "Catseye", "Darkmeyer", "Flamingobreath"]
012.      spy_name = choice(first_names) + " " + choice(last_names)
013.      #print(spy_name)
014.      name.value = spy_name
015.
016.  # App -------------------
017.
018.  app = App("TOP SECRET")
019.
020.  # Widgets ---------------
021.
022.  title = Text(app, "Push the red button to find out your spy name")
023.  button = PushButton(app, choose_name, text="Tell me!")
024.  button.bg = "red"
025.  button.text_size = 30
026.  name = Text(app, text="")
027.
028.  # Display ---------------
029.
030.  app.display()
```

Teil 03

GUIs mit Python: Meme-Generator

Erstellen Sie Memes mit selbst-programmierter GUI-App.

Memes sind Fotos, die sich in den sozialen Medien verbreiten und sich, oft als Satire gedacht, zum Teil auf bekannte Personen beziehen und durch lustige Kommentare einen völlig neuen Sinn erhalten. Schreiben Sie ein Programm mit grafischer Benutzeroberfläche, das Memes erzeugt und auf den vorhergehenden Lektionen basiert.

Geben Sie einen Text und einen Bildnamen ein, und Ihre GUI wird beides zu einem Meme zusammensetzen. Beginnen Sie mit der Erstellung einer einfachen GUI mit einem oberen und einem unteren Textfeld. Dort geben Sie den Text ein, der in Ihrem Bild eingefügt wird. Mit der folgenden Zeile importieren Sie die benötigten Widgets.

```
from guizero import App, TextBox, Drawing
```

Fügen Sie dann diesen Code für die App hinzu:

```
app = App("meme")

top_text = TextBox(app, "top text")
bottom_text = TextBox(app, "bottom text")

app.display()
```

Das Meme wird in einem Drawing-Widget erstellt, das das Bild und den Text enthält.

Ein Meme erstellen

Fügen Sie folgenden Code direkt vor der Zeile `app.display()` ein. Die Höhe und Breite des Bildes sollte so eingestellt werden, dass es die verfügbare Fläche komplett ausfüllt.

```
meme = Drawing(app, width="fill",
height="fill")
```

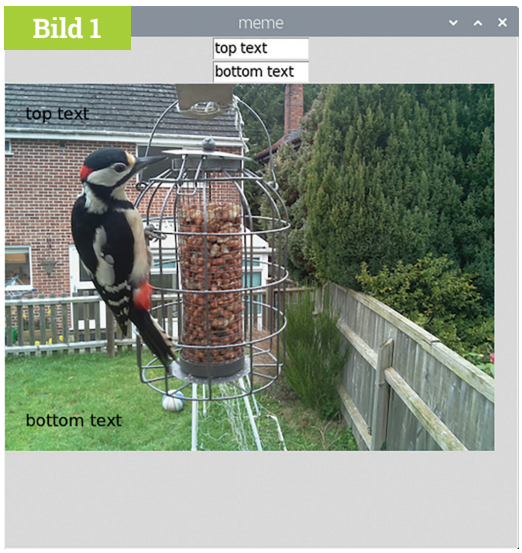
Das Meme wird erzeugt, wenn sich der Text im oberen und unteren Textfeld ändert. Dazu müssen wir eine Funktion erstellen, die das Meme zeichnet.

Die Funktion soll ein Bild einfügen (im Beispiel einen Specht) und den Text oben und unten in das Bild einsetzen.

Erinnern Sie sich, dass Sie **name.value** verwendet haben, um den Wert des Text-Widgets mit dem Nickname in Teil 2 dieser Serie zu bestimmen? Sie können die Eigenschaft `value` aber auch verwenden, um den Wert eines Text-Widgets abzurufen, so dass `top_text.value` in diesem Fall bedeutet: "Bitte frage den Wert ab, der in das Feld `top_text` eingegeben wurde".

```
def draw_meme():
    meme.clear()
    meme.image(0, 0, "woodpecker.png")
    meme.text(20, 20, top_text.value)
    meme.text(20, 320, bottom_text.value)
```

Die ersten beiden Zahlen in `meme.image(0, 0)` und `meme.text(20, 20)` sind die x- und y-Koordinaten, an denen das Bild und der Text gezeichnet werden sollen. Das Bild wird an der linken oberen Ecke angesetzt (Koordinaten 0, 0).
Nun muss noch die Funktion `draw_meme` aufgerufen



▲ **Bild 1** Meme mit vorgegebenem Standardtext.

werden. Fügen Sie folgenden Code direkt vor der Zeile `app.display` ein:

```
draw_meme()
```

Ihr Code sollte nun wie **meme1.py** aussehen. Wenn Sie Ihre App (**Bild 1**) ausführen, werden Sie feststellen, dass eine Textänderung im Meme nicht aktualisiert wird. Dazu müssen Sie Ihr Programm erst so ändern, dass es die Funktion `draw_meme` aufruft, wenn sich der Text ändert. Fügen Sie dazu einen Befehl zu den beiden TextBox-Widgets der App hinzu.

> Erzeugen Sie Ihr individuelles Meme, indem Sie den oberen und unteren Text ändern

```
top_text = TextBox(app, "top text",
command=draw_meme)
bottom_text = TextBox(app, "bottom text",
command=draw_meme)
```

Ihr Code sollte nun wie derjenige in **meme2.py** aussehen. Führen Sie ihn aus und aktualisieren Sie Ihr Meme, indem Sie den oberen und unteren Text ändern.

meme1.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, TextBox, Drawing
004.
005.
006.  # Functions -------------
007.
008.  def draw_meme():
009.      meme.clear()
010.      meme.image(0, 0, "woodpecker.png")
011.      meme.text(20, 20, top_text.value)
012.      meme.text(20, 320, bottom_text.value)
013.
014.
015.  # App -------------------
016.
017.  app = App("meme")
018.
019.  top_text = TextBox(app, "top text")
020.  bottom_text = TextBox(app, "bottom text")
021.
022.  meme = Drawing(app, width="fill", height="fill")
023.
024.  draw_meme()
025.
026.  app.display()
```

meme2.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, TextBox, Drawing
004.
005.
006.  # Functions -------------
007.
008.  def draw_meme():
009.      meme.clear()
010.      meme.image(0, 0, "woodpecker.png")
011.      meme.text(20, 20, top_text.value)
012.      meme.text(20, 320, bottom_text.value)
013.
014.
015.  # App -------------------
016.
017.  app = App("meme")
018.
019.  top_text = TextBox(app, "top text", command=draw_meme)
020.  bottom_text = TextBox(app, "bottom text", command=draw_meme)
021.
022.  meme = Drawing(app, width="fill", height="fill")
023.
024.  draw_meme()
025.
026.  app.display()
```

Top Tipp

Da die Codezeilen immer länger wurden, haben wir sie auf mehrere Zeilen aufgeteilt, um sie leichter lesbar zu machen. Dies hat keinen Einfluss auf die Funktion des Programms, sondern nur auf sein Aussehen.

Sie können das Aussehen Ihres Memes ändern, indem Sie die Parameter für `color`, `size`, und `font` des Textes ändern. Zum Beispiel:

```
meme.text(
    20, 20, top_text.value,
    color="orange",
    size=40,
    font="courier")
meme.text(
    20, 320, bottom_text.value,
    color="blue",
    size=28,
    font="times new roman",
    )
```

meme3.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, TextBox, Drawing
004.
005.
006.  # Functions -------------
007.
008.  def draw_meme():
009.      meme.clear()
010.      meme.image(0, 0, "woodpecker.png")
011.      meme.text(
012.          20, 20, top_text.value,
013.          color="orange",
014.          size=40,
015.          font="courier")
016.      meme.text(
017.          20, 320, bottom_text.value,
018.          color="blue",
019.          size=28,
020.          font="times new roman",
021.          )
022.
023.
024.  # App -------------------
025.
026.  app = App("meme")
027.
028.  top_text = TextBox(app, "top text", command=draw_meme)
029.  bottom_text = TextBox(app, "bottom text", command=draw_meme)
030.
031.  meme = Drawing(app, width="fill", height="fill")
032.
033.  draw_meme()
034.
035.  app.display()
```

Ihr Code sollte nun wie **meme3.py** aussehen. Probieren Sie verschiedene Stile aus, bis Sie etwas finden, das Ihnen gefällt (**Bild 2**).

Individuelle Anpassung

Für einen wirklich interaktiven Meme-Generator sollte der Benutzer in der Lage sein, Schriftart, Größe und Farbe selbst einzustellen. Dazu können Sie zusätzliche Widgets auf der Benutzeroberfläche bereitstellen. Für die Farbe und die Schriftart könnten Sie zum Beispiel eine Dropdown-Liste, auch Combo-Box genannt, verwenden. Die Schriftgröße wiederum könnte zum Beispiel mit einem Slider-Widget eingestellt werden. Ändern Sie zunächst Ihre Import-Anweisung, um die Combo- und Slider-Widgets einzubinden.

```
from guizero import App, TextBox, Drawing,
Combo, Slider
```

So erstellen Sie ein neues Combo-Widget zur Auswahl der Farben.

```
bottom_text = TextBox(app, "bottom text",
command=draw_meme)
color = Combo(app,
    options=["black", "white", "red",
"green", "blue", "orange"],
    command=draw_meme)
```

> Die Optionen werden in der Reihenfolge angezeigt, in der sie in die Liste eingetragen wurden

Der Parameter `options` legt fest, welche Farben der Benutzer aus der Combo-Box auswählen kann. Jede Farbe ist ein Element in einer Liste. Sie können auch beliebige andere Farben in die Liste aufnehmen.
Die Optionen werden in der Reihenfolge angezeigt, in der Sie sie in die Liste setzen. Die erste Option ist die Standardoption, die zuerst angezeigt wird. Farben, die von der Standardeinstellung abweichen, können über den ausgewählten Parameter (`selected`) abgerufen werden, z. B. `"blue"`.

Bild 2

▲ **Bild 2** Ändern der Schriftarten und Schriftfarben

```
color = Combo(app,
    options=["black", "white", "red",
"green", "blue", "orange"],
    command=draw_meme,
    selected="blue")
```

Jetzt kann eine Farbe ausgewählt werden. Als nächstes müssen Sie die Funktion `draw_meme` so ändern, dass der Wert von Combo verwendet wird, wenn Sie den Text in Ihrem Meme erstellen. Zum Beispiel:

```
meme.text(
    20, 20, top_text.value,
    color=color.value,
    size=40,
    font="courier")
```

Führen Sie dasselbe für den unteren Textblock aus. Ihr Programm sollte nun **meme4.py** ähneln.

Fügen Sie nach den obigen Schritten eine zweite Combo-Box zu Ihrer Anwendung hinzu, damit der Benutzer eine Schriftart aus dieser Liste von Optionen auswählen kann: [`"times new roman"`, `"verdana"`, `"courier"`, `"impact"`]. Denken Sie daran, die Funktion `draw_meme` so zu ändern, dass beim Hinzufügen des Textes der Wert der Schriftart (font) verwendet wird.

Erstellen Sie nun ein Slider-Widget, um die Größe des gewünschten Textes einzustellen.

meme4.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, TextBox, Drawing, Combo, Slider
004.
005.
006.  # Functions -------------
007.
008.  def draw_meme():
009.      meme.clear()
010.      meme.image(0, 0, "woodpecker.png")
011.      meme.text(
012.          20, 20, top_text.value,
013.          color=color.value,
014.          size=40,
015.          font="courier")
016.      meme.text(
017.          20, 320, bottom_text.value,
018.          color=color.value,
019.          size=28,
020.          font="times new roman",
021.          )
022.
023.
024.  # App --------------------
025.
026.  app = App("meme")
027.
028.  top_text = TextBox(app, "top text", command=draw_meme)
029.  bottom_text = TextBox(app, "bottom text", command=draw_meme)
030.
031.  color = Combo(app,
032.                options=["black", "white", "red", "green",
      "blue", "orange"],
033.                command=draw_meme, selected="blue")
034.
035.  meme = Drawing(app, width="fill", height="fill")
036.
037.  draw_meme()
038.
039.  app.display()
```

Drawing-Widget

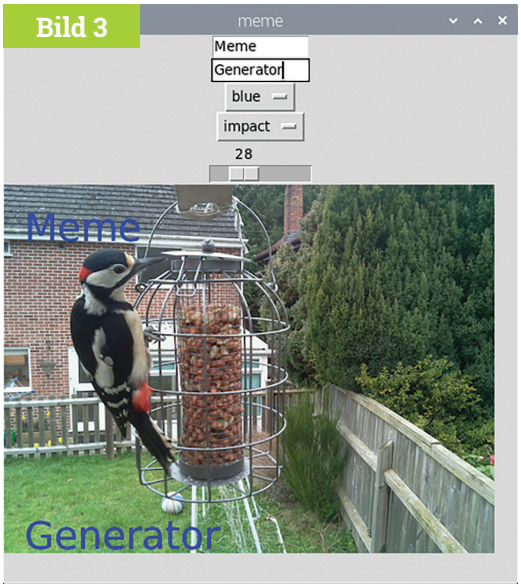
Das Drawing-Widget (Zeichnen) ist sehr vielseitig und kann zur Anzeige vieler verschiedener Formen, Muster und Bilder verwendet werden. Lesen Sie dazu auch Anhang C des Buches **magpi.cc/pythongui** oder werfen Sie einen Blick in die Online-Dokumentation: **lawsie.github.io/guizero/drawing**.

04-meme-generator.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, TextBox, Drawing, Combo, Slider
004.
005.
006.  # Functions -------------
007.
008.  def draw_meme():
009.      meme.clear()
010.      meme.image(0, 0, "woodpecker.png")
011.      meme.text(
012.          20, 20, top_text.value,
013.          color=color.value,
014.          size=size.value,
015.          font=font.value)
016.      meme.text(
017.          20, 320, bottom_text.value,
018.          color=color.value,
019.          size=size.value,
020.          font=font.value,
021.          )
022.
023.
024.  # App -------------------
025.
026.  app = App("meme")
027.
028.  top_text = TextBox(app, "top text", command=draw_meme)
029.  bottom_text = TextBox(app, "bottom text", command=draw_meme)
030.
031.  color = Combo(app,
032.                options=["black", "white", "red", "green",
      "blue", "orange"],
033.                command=draw_meme, selected="blue")
034.
035.  font = Combo(app,
036.               options=["times new roman", "verdana", "courier",
      "impact"],
037.               command=draw_meme)
038.
039.  size = Slider(app, start=20, end=50, command=draw_meme)
040.
041.  meme = Drawing(app, width="fill", height="fill")
042.
043.  draw_meme()
044.
045.  app.display()
```

Bild 3



▲ **Bild 3** Der fertige Meme-Generator.

```
  size = Slider(app, start=20, end=40,
command=draw_meme)
```

Der Bereich des Schiebereglers wird mit den Parametern `start` und `end` festgelegt. In diesem Beispiel hat also der kleinste verfügbare Text die Größe 20 und der größte Text die Größe 40.

Ändern Sie die Funktion `draw_meme` so, dass der Wert aus dem Slider für die Textgröße verwendet wird.

```
  meme.text(
      20, 20, top_text.value,
      color=color.value,
      size=size.value,
      font=font.value)
```

Ihr Code sollte nun demjenigen in **04-meme-generator.py** ähneln. Wenn Sie ihn auszuführen, werden Sie ein Ergebnis erhalten, das ähnlich aussieht wie in **Bild 3** dargestellt.

Können Sie die grafische Oberfläche so ändern, dass der Name der Bilddatei in eine TextBox eingegeben oder vielleicht aus einer Liste in einer Combo-Box ausgewählt werden kann? Das würde Ihre Anwendung in die Lage versetzen, auch Memes mit verschiedenen, auswählbaren Bildern zu erzeugen.

Teil 04

GUIs mit Python: Die schlechteste GUI der Welt

Durch Fehler lernen.

Es ist nun an der Zeit, mit verschiedenen Widgets, Farben, Schriftarten und Funktionen zu experimentieren. Hier ein paar Beispiele dafür, wie Sie es auf keinen Fall machen sollten.

Schlechte Lesbarkeit

Die richtige Wahl des Kontrastes zwischen Hintergrund und Schriftfarbe ist bei der Gestaltung einer grafischen Benutzeroberfläche sehr wichtig. Importieren Sie die Widgets am Anfang des Codes:

```
from guizero import App, Text
```

Erstellen Sie eine App mit einem Titel:

```
app = App("it's all gone wrong")
title = Text(app, text="Some hard to read text")

app.display()
```

Experimentieren Sie, indem Sie die Farben, die Schriftart und die Textgröße ändern (siehe Listing von **worst1.py**).

```
app = App("it's all gone wrong", bg="dark green")
title = Text(app, text="Some hard-to-read text", size="14", font="Comic Sans", color="green")
```

Blinkender Text

Es ist wichtig, dass der Text auf einer GUI auch lange genug sichtbar bleibt, um gelesen zu werden. Er sollte auf keinen Fall verschwinden oder anfangen zu blinken.

Alle Widgets in guizero können mit den Funktionen `hide()` und `show()` unsichtbar (oder wieder sichtbar) gemacht werden. Wenn Sie die `repeat`-Funktion in guizero verwenden, um eine Funktion im Sekundentakt auszuführen, können Sie Ihren Text verstecken oder ihn blinken lassen. Ein Beispiel:

```
def flash_text():
    if title.visible:
        title.hide()
    else:
        title.show()
```

Sorgen Sie mit `repeat` dafür, dass die Funktion `flash_text` alle 1000 Millisekunden (1 Sekunde) aufgerufen wird.

```
app.repeat(1000, flash_text)

app.display()
```

Ihr Code sollte nun wie **worst2.py** aussehen. Der Titeltext sollte im Sekundentakt auftauchen und wieder verschwinden.

Unpassende Eingabe-Elemente

Die Verwendung eines passenden Eingabe-Elementes, also Widgets, kann darüber entscheiden, ob eine GUI brauchbar oder unbrauchbar ist.

Welches Widget würden Sie zum Beispiel verwenden, um ein Datum einzugeben? Eine einzige TextBox? Mehrere Combo-Boxen? Eine TextBox wäre flexibler, aber komplizierter. Mehrere Combo-Boxen für Jahr, Monat und Tag wären einfacher zu dekodieren, aber umständlicher in der Anwendung. Die Verwendung eines Schiebereglers zum Einstellen von Datum und Uhrzeit (**Bild 2**), wie im

Bild 1 Umständlich, aber möglich: Texteingabe durch aneinandergereihte Combo-Boxen mit Einzelbuchstaben.

Codebeispiel von **worst3.py** ist allerdings keine gute Idee: Das Slider-Widget gibt eine Zahl zwischen 0 und 999.999.999 zurück. Dies ist die Anzahl der seit dem 1. Januar 1970 verstrichenen Sekunden. Die Funktion `ctime()` wird verwendet, um diese Zahl in ein Datum und eine Uhrzeit zu verwandeln.

Es gibt allerdings auch Anwender, die am liebsten nur die Maus benutzen möchten. Vielleicht wäre dann bei einigen Eingaben, die nur aus wenigen Buchstaben bestehen, eine Reihe von Comboboxen, die jeweils alle Buchstaben des Alphabets enthalten, besser **(Bild 1**). Beginnen Sie mit dem Importieren des guizero-Widgets und der `ascii_letters` Liste.

```
from guizero import App, Combo
from string import ascii_letters
```

`ascii_letters` ist eine Liste mit allen darstellbaren ASCII-Zeichen, die Sie für die Combo-Box verwenden können.

Erstellen Sie eine einzelne Combo-Box, die alle Buchstaben enthält und die App anzeigt.

```
a_letter = Combo(app, options=" " + ascii_
letters, align="left")

app.display()
```

Ihr Programm sollte nun der Datei **worst4.py** ähneln. Sie sehen eine einzelne Combo-Box, die alle Buchstaben plus ein Leerzeichen enthält und am linken Rand des Fensters ausgerichtet ist.

Um eine Reihe von Buchstaben zusammen zu bekommen, könnten Sie kontinuierlich Combo-Widgets zu Ihrer App hinzufügen, zum Beispiel:

```
a_letter = Combo(app, options=" " + ascii_
letters, align="left")
b_letter = Combo(app, options=" " + ascii_
letters, align="left")
c_letter = Combo(app, options=" " + ascii_
letters, align="left")
```

Alternativ könnten Sie auch eine for-Schleife verwenden, eine Liste von Buchstaben erstellen und jeden Buchstaben an die Liste anhängen, wie in **worst5.py** gezeigt.

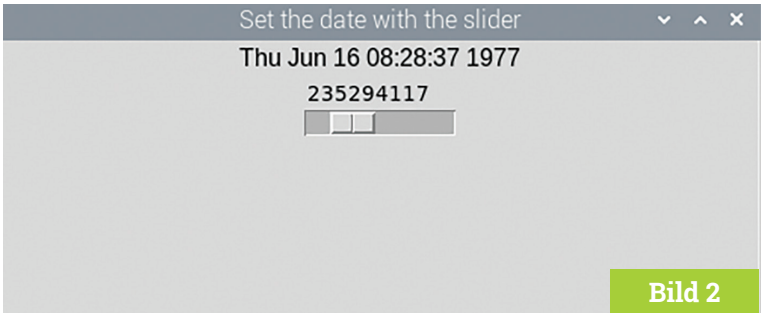
Letztlich ist es Geschmacksache. Die for-Schleife ist flexibler, da Sie so viele Buchstaben erstellen können, wie Sie möchten.

worst1.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text
004.
005.
006.  # App -------------------
007.
008.  app = App("it's all gone wrong", bg="dark green")
009.
010.  title = Text(app, text="Hard to read", size="14", font="Comic
011.  Sans", color="green")
012.
013.  app.display()
```



▲ **Bild 2** Umständlich, aber möglich: Ein Schieberegler zum Einstellen von Datum und Uhrzeit.

worst2.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text
004.
005.
006.  # Functions -------------
007.
008.  def flash_text():
009.      if title.visible:
010.          title.hide()
011.      else:
012.          title.show()
013.
014.
015.  # App -------------------
016.
017.  app = App("it's all gone wrong", bg="dark green")
018.
019.  title = Text(app, text="Hard to read", size="14", font="Comic
020.  Sans", color="green")
021.
022.  app.repeat(1000, flash_text)
023.
024.  app.display()
```

worst3.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Slider, Text
004.  from time import ctime
005.
006.
007.  # Functions -------------
008.
009.  def update_date():
010.      the_date.value = ctime(date_slider.value)
011.
012.
013.  # App -------------------
014.
015.  app = App("Set the date with the slider")
016.  the_date = Text(app)
017.  date_slider = Slider(app, start=0, end=999999999,
018.  command=update_date)
019.
020.  app.display()
```

worst4.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Combo
003.  from string import ascii_letters
004.
005.
006.  # App -------------------
007.
008.  app = App("Enter your name")
009.
010.  a_letter = Combo(app, options=" " + ascii_letters, align="left")
011.
012.  app.display()
```

Bild 3
Sinnlose Pop-up-Boxen

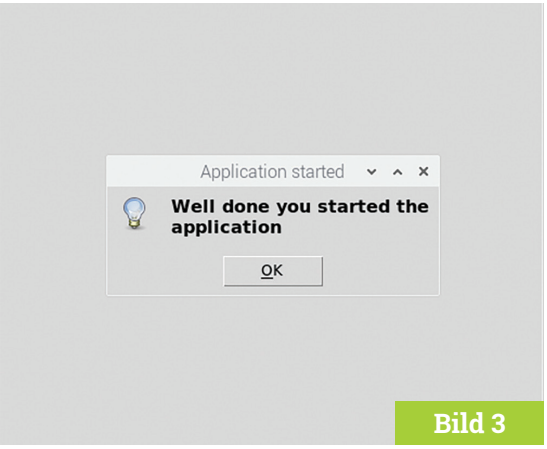


Bild 3

Das Pop-up-Widget *Window*

Pop-up-Boxen können verwendet werden, um den Benutzer vor Entscheidungen zu stellen. Ihre Programmierung und Anwendung ist sehr einfach.

Wenn Sie zusätzliche Informationen anzeigen oder nach zusätzlichen Daten fragen möchten, können Sie das Window-Widget verwenden, um mehrere Fenster zu erstellen.

Window wird auf ähnliche Weise wie **App** verwendet und hat viele der gleichen Funktionen.

```
from guizero import App, Window

app = App("Main window")
window = Window(app, "2nd Window")

app.display()
```

Ob das Pop-up-Widget **Window** auf dem Bildschirm zu sehen ist oder nicht, können Sie mit den Methoden **show()** und **hide()** festlegen.

```
window.show()
window.hide()
```

Eine App kann dazu gebracht werden, auf das Schließen eines Fensters zu warten, nachdem es angezeigt wurde, indem man True an den wait-Parameter von show übergibt. Zum Beispiel:

```
window.show(wait=True)
```

Weitere Informationen über die Verwendung mehrerer Fenster finden Sie in der guizero-Dokumentation:

lawsie.github.io/guizero/multiple_windows.

Pop-ups

Keine als abschreckendes Beispiel dienende GUI wäre vollständig ohne eine Pop-up-Box. guizero enthält eine Reihe von Pop-up-Boxen, die dazu verwendet werden können, den Benutzer unter anderem auf etwas Wichtiges hinzuweisen – oder ihn zu irritieren und zu ärgern!

Erstellen Sie zunächst eine Anwendung, die beim Start ein sinnloses Kästchen einblendet, um mitzuteilen, dass die Anwendung gestartet wurde.

```
from guizero import App

app = App(title="pointless pop-ups")

app.info("Application started", "Well done
you started the application")

app.display()
```

Sie werden sehen, dass eine "info"-Box erscheint (**Bild 3**). Der erste Parameter, der an *info* übergeben

Bild 4

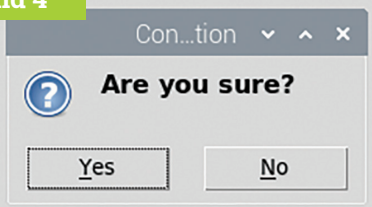


Bild 4 Ja, wir sind uns sicher!

wird, ist der Titel des Fensters, der zweite Parameter ist die Meldung.

Sie können den Stil dieser einfachen Pop-up-Box ändern, indem Sie *warn* oder *error* anstelle von info verwenden.

Pop-up-Boxen können auch verwendet werden, um Informationen vom Benutzer zu erhalten. Das einfachste ist ein Ja-Nein-Feld, das dem Benutzer eine Frage stellt und auf Antwort wartet, zum Beispiel, ob er eine Datei wirklich löschen möchte. Aber natürlich nicht jedes Mal, wenn er irgend eine Taste drückt! Importieren Sie das PushButton-Widget in Ihre Anwendung:

```
from guizero import App, PushButton
```

Erstellen Sie eine Funktion, die das `yesno`-Pop-up benutzt, um nach einer Bestätigung zu fragen.

```
def are_you_sure():
    if app.yesno("Confirmation", "Are you
sure?"):
        app.info("Thanks", "Button
pressed")
    else:
        app.error("Ok", "Cancelling")
```

Fügen Sie die Schaltfläche in Ihre GUI ein, die die Funktion aufruft, wenn sie gedrückt wird.

```
button = PushButton(app, command=are_you_
sure)
```

Ihr Code sollte nun der **Datei 05-worlds-worst-gui.py** ähneln. Wenn Sie die Anwendung ausführen und auf die Schaltfläche klicken, sehen Sie eine Pop-up-Box, die Sie auffordert, Ja oder Nein einzugeben (**Bild 4**).

Mehr über die Pop-up-Boxen in guizero erfahren Sie unter **lawsie.github.io/guizero/alerts**. Wie wäre es, all diese "Funktionen" in einer ultimativen GUI zu kombinieren?

worst5.py

Sprache: **Python 3**

```
# Imports ---------------

from guizero import App, Combo
from string import ascii_letters


# App -------------------

app = App("Enter your name")

name_letters = []
for count in range(10):
    a_letter = Combo(app, options=" " + ascii_letters,
align="left")
    name_letters.append(a_letter)

app.display()
```

05-worlds-worst-gui.py

Sprache: **Python 3**

```
from guizero import App, PushButton

def are_you_sure():
    if app.yesno("Confirmation", "Are you sure?"):
        app.info("Thanks", "Button pressed")
    else:
        app.error("Ok", "Cancelling")

app = App(title="pointless pop-ups")

button = PushButton(app, command=are_you_sure)

app.info("Application started", "Well done you started the
application")

app.display()
```

Teil 05

Grafische Benutzeroberflächen mit Python: Tic-Tac-Toe

Einfaches Spiel selbst programmiert.

Nachdem Sie gelernt haben, wie man eine einfache grafische Benutzeroberfläche (im Folgenden wieder GUI genannt) erstellt, wollen wir diese nun mit Hilfe ausgeklügelter Programmzeilen für ein Tic-Tac-Toe-Spiel nutzen.

Erstellen Sie eine neue Datei mit dem folgenden Code

```
# Imports ---------------
from guizero import App

# Functions -------------

# Variables -------------

# App -------------------
app = App("Tic tac toe")

app.display()
```

Das Spielfeld

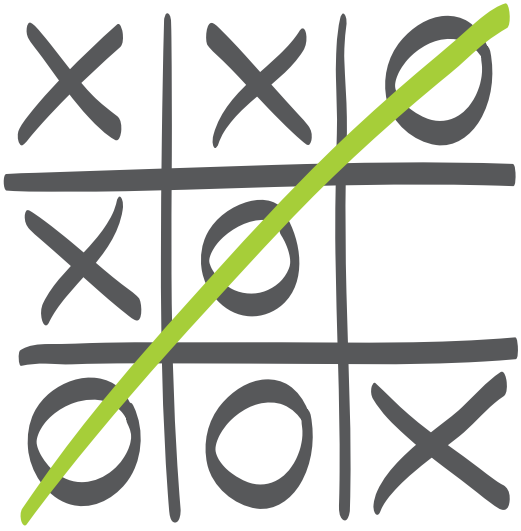
Beginnen wir damit, die Widgets zu erstellen, die das Spielfeld bilden werden. Ein traditionelles Tic-Tac-Toe-Spielfeld sieht wie in **Bild 1** dargestellt aus.

Jede einzelne Position eines Spielers wird durch eine Schaltfläche (Button) repräsentiert, so dass der Spieler, wenn er darauf klickt, anzeigen kann, wohin er ziehen möchte. Um die Schaltflächen in einem Raster anzuordnen, erstellen wir einen neuen Typ von Guizero-Widget namens ***Box***. Eine Box ist ein Container-Widget, das verwendet wird, um andere Widgets darin unterzubringen und sie zu gruppieren. Fügen Sie es zu den Importen am Anfang Ihres Codes hinzu:

```
from guizero import App, Box
```

Stellen Sie die Box so ein, dass sie ein Grid-Layout hat, und fügen Sie sie zu Ihrer App hinzu - vor der Zeile `app.display()`, wie bei allen Widgets.

Bild 1



▲ **Bild 1** Ein typisches Tic-Tac-Toe-Spiel.

```
board = Box(app, layout="grid")
```

Wenn Sie Ihr Programm an dieser Stelle ausführen, werden Sie nichts auf dem Bildschirm sehen, da die Box selbst unsichtbar ist.

Lassen Sie uns nun die neun Schaltflächen erstellen, die in der Box enthalten sind. Dazu können Sie eine verschachtelte Schleife verwenden, um sie automatisch zu erzeugen und ihnen die betreffenden Koordinaten zuzuweisen. Fügen Sie zunächst *PushButton* zu Ihrer Liste der zu importierenden Widgets hinzu und fügen Sie dann diesen Code direkt nach dem Code für das gerade erstellte Board ein.

```
for x in range(3):
    for y in range(3):
        button = PushButton(
            board, text="", grid=[x, y],
width=3
        )
```

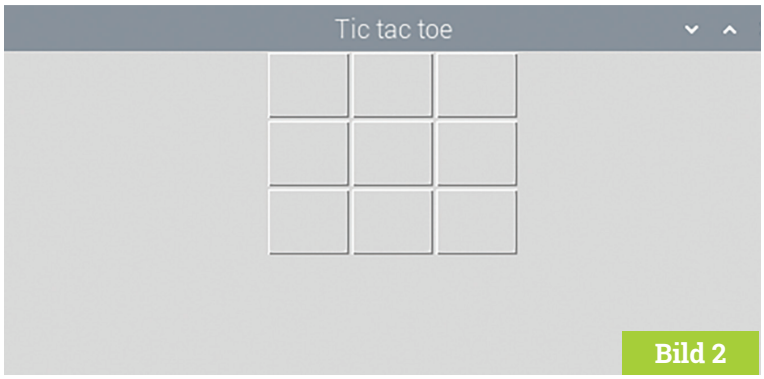


Bild 2 Ein Raster mit neun Feldern zum Spielen von Tic-Tac-Toe.

tictactoe1.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Box, PushButton
003.
004.  # Functions -------------
005.
006.  # Variables -------------
007.
008.  # App -------------------
009.  app = App("Tic tac toe")
010.
011.  board = Box(app, layout="grid")
012.  for x in range(3):
013.          for y in range(3):
014.          button = PushButton(
      board, text="", grid=[x, y], width=3)
015.
016.  app.display()
```

tictactoe2.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Box, PushButton
003.
004.  # Functions -------------
005.  def clear_board():
006.      new_board = [[None, None, None], [None, None, None], [
      None, None, None]]
007.      for x in range(3):
008.          for y in range(3):
009.              button = PushButton(
010.                  board, text="", grid=[x, y], width=3)
011.              new_board[x][y] = button
012.      return new_board
013.
014.  # Variables -------------
015.
016.  # App -------------------
017.  app = App("Tic tac toe")
018.
019.  board = Box(app, layout="grid")
020.  board_squares = clear_board()
021.
022   app.display()
```

Beachten Sie, dass es zwei Schleifenvariablen gibt: *x* von 0 bis 2 und *y* von 0 bis 2. Während der Iteration und der Erzeugung von Schaltflächen wird jede Schaltfläche zum Board, also dem zuvor erstellten Box-Container, hinzugefügt. Der Schaltfläche werden die Gitterkoordinaten `x, y` zugewiesen, was bedeutet, dass jede Schaltfläche eine bestimmte, geordnete Position auf dem Gitter einnimmt. Ihr Code sollte nun wie derjenige in der Datei **tictactoe1.py** aussehen. Das Ergebnis seiner Ausführung ist in **Bild 2** dargestellt.

Datenstruktur im Hintergrund

Mit Hilfe einer Schleife erzeugen Sie nun also automatisch neun Schaltflächen, und jede einzelne heißt `button`. Zu ihrer Unterscheidung benötigen Sie eine dahinter verborgene Datenstruktur, die durch eine zweidimensionale Liste repräsentiert wird.

Erstellen wir nun eine Funktion, die wir aufrufen können, um das Spielfeld zu löschen. Auf ihren Code kann nach einem Spiel immer wieder zugegriffen werden, um ein neues Spiel zu beginnen. Fügen Sie daher nun im Abschnitt *functions* eine neue Funktion namens `clear_board` hinzu.

```
def clear_board():
```

Die erste Aufgabe innerhalb dieser Funktion besteht darin, die Datenstruktur für das Spielfeld zu initialisieren. Gehen wir davon aus, dass Sie zu diesem Zeitpunkt noch keine Schaltflächen erstellt haben. So können Sie jede Position auf dem Spielfeld als `None` initialisieren. Das Element in der Liste existiert nun, hat aber noch keinen Wert. Fügen Sie den folgenden Code, eingerückt, in Ihre Funktion ein

```
    new_board = [[None, None, None], [None,
None, None], [None, None, None]]
```

Verschieben Sie als nächstes den verschachtelten Schleifencode aus Ihrem App-Abschnitt in die Funktion `clear_board`. Stellen Sie sicher, dass die Einrückung korrekt ist. Fügen Sie innerhalb der inneren (y) Schleife eine Codezeile hinzu, um eine Referenz auf jede Schaltfläche an ihrer Koordinatenposition (*x*, *y*) innerhalb der zweidimensionalen Liste zu speichern, damit Sie später darauf verweisen können.

```
    new_board[x][y] = button
```

Zum Schluss, nachdem die Schleifen beendet sind, geben Sie das soeben erstellte `new_board` zurück. Ihre Funktion sollte wie folgt aussehen:

```
def clear_board():
    new_board = [[None, None, None],
                 [None, None, None],
```

Reset Button

Am Anfang haben Sie eine Funktion namens **clear_board** geschrieben. Das mag Ihnen damals unnötig erschienen sein, aber sie wird wichtig, wenn das Spiel beendet ist. Da Tic-Tac-Toe ein recht kurzes Spiel ist, ist es sehr wahrscheinlich, dass jemand mehr als ein Spiel hintereinander spielen möchte. Trauen Sie es sich zu, Ihrem Spiel einen Reset-Button hinzufügen, der erst erscheint, wenn entweder jemand das Spiel gewonnen hat oder das Spiel unentschieden war? Die Schaltfläche sollte die Funktion **clear_board** aufrufen und die Variable **turn** sowie die Meldung, wer am Zug ist, zurücksetzen.

Tipp 1: Schauen Sie in der guizero-Dokumentation nach, wie Sie Widgets ein- und ausblenden können, so dass Ihre Schaltfläche nicht die ganze Zeit während des Spiels sichtbar ist.
Tipp 2: Erstellen Sie eine neue Funktion, die sich um alles kümmert, was notwendig ist, um das Spiel zurückzusetzen, und rufen Sie diese Funktion auf, wenn der Reset-Button gedrückt wird. Vergessen Sie nicht, dass Sie in Ihrer Funktion einige Variablen als global angeben müssen.

```
                [None, None, None]]
    for x in range(3):
        for y in range(3):
            button = PushButton(
                board, text="", grid=[x,
y], width=3
            )
            new_board[x][y] = button
    return new_board
```

Initialisieren Sie im Abschnitt *app* eine Liste namens `board_squares` und legen Sie fest, dass diese die neue Funktion aufruft, die Sie gerade erstellt haben.

```
board_squares = clear_board()
```

Dieser Variablen wird der Wert von `new_board` zugewiesen, was eine leere Oberfläche mit neun Schaltflächen ergeben sollte. Achten Sie darauf, dass Sie diese Variable erst nach dem Code zum Erzeugen der Box anlegen, da sonst versucht wird, Schaltflächen zu einem Container hinzuzufügen, der noch nicht existiert.

Ihr Code wird nun **tictactoe2.py** ähneln. Speichern Sie das Programm und führen Sie es aus. Sie sollten dasselbe Ergebnis sehen wie am Ende des letzten Schrittes, aber jetzt haben Sie eine versteckte, zweidimensionale Listendatenstruktur, mit der Sie auf die Schaltflächen zugreifen und sie beeinflussen können.

Mit einem `print`-Befehl können Sie die 2D-Liste `board_squares` anzeigen: `print(board_squares)`. Der Eintrag `[PushButton] object with text ""` wird dann neun Mal in der Shell erscheinen.

tictactoe3.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Box, PushButton, Text
003.
004.  # Functions -------------
005.  def clear_board():
006.      new_board = [[None, None, None], [None, None, None], [None,
      None, None]]
007.      for x in range(3):
008.          for y in range(3):
009.              button = PushButton(board, text="", grid=[x, y],
      width=3, command=choose_square, args=[x,y])
010.              new_board[x][y] = button
011.      return new_board
012.
013.  def choose_square(x, y):
014.      board_squares[x][y].text = turn
015.      board_squares[x][y].disable()
016.
017.  # Variables -------------
018.  turn = "X"
019.
020.  # App -------------------
021.  app = App("Tic tac toe")
022.
023.  board = Box(app, layout="grid")
024.  board_squares = clear_board()
025.  message = Text(app, text="It is your turn, " + turn)
026.
027.  app.display()
```

Erwecken Sie die Schaltflächen zum Leben

Im Moment passiert noch nichts, wenn Sie auf die Tasten drücken. Erstellen wir daher eine Funktion, die an die Schaltfläche angehängt wird, so dass die Schaltfläche bei Aktivierung entweder X oder O anzeigt, je nachdem, welcher Spieler sie ausgewählt hat. Erstellen Sie zunächst eine Variable im Variablenbereich, um zu registrieren, wer am Zug ist. Wir werden mit X beginnen.

```
turn = "X"
```

Auf der grafischen Benutzeroberfläche muss angezeigt werden, wer an der Reihe ist (**Bild3**), damit die Spieler nicht verwirrt werden. Fügen Sie **Text** zu Ihrer Liste der zu importierenden Widgets hinzu:

```
from guizero import App, Box, PushButton,
Text
```

Fügen Sie dann ein neues Text-Widget im App-Bereich hinzu, um den Spielzug anzuzeigen.

```
message = Text(app, text="It is your turn,
" + turn)
```

Wechseln Sie in den Bereich *functions* und erstellen Sie eine neue Funktion namens `choose_square`.

```
def choose_square(x, y):
```

tictactoe4.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Box, PushButton, Text
003.
004.  # Functions -------------
005.  def clear_board():
006.      new_board = [[None, None, None], [None, None, None], [None,
      None, None]]
007.      for x in range(3):
008.          for y in range(3):
009.              button = PushButton(board, text="", grid=[x, y],
      width=3, command=choose_square, args=[x,y])
010.              new_board[x][y] = button
011.      return new_board
012.
013.  def choose_square(x, y):
014.      board_squares[x][y].text = turn
015.      board_squares[x][y].disable()
016.      toggle_player()
017.
018.  def toggle_player():
019.      global turn
020.      if turn == "X":
021.          turn = "O"
022.      else:
023.          turn = "X"
024.      message.value = "It is your turn, " + turn
025.
026.  # Variables -------------
027.  turn = "X"
028.
029.  # App -------------------
030.  app = App("Tic tac toe")
031.
032.  board = Box(app, layout="grid")
033.  board_squares = clear_board()
034.  message = Text(app, text="It is your turn, " + turn)
035.
036.  app.display()
```

Bild 3 Ein Spieler muss wissen, wann er an der Reihe ist.

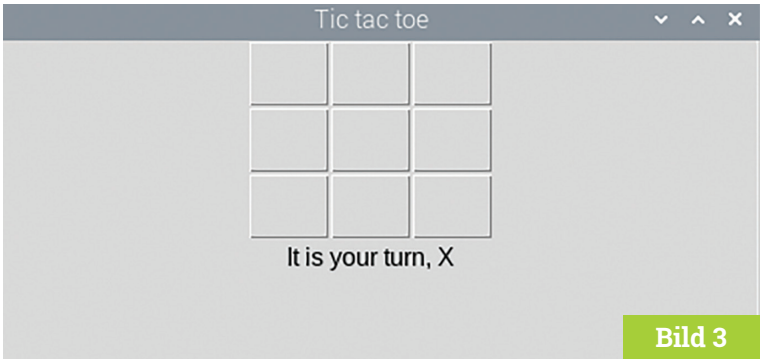


Bild 3

Diese Funktion benötigt zwei Argumente (*x* und *y*), damit Sie wissen, welches Quadrat auf dem Spielfeld angeklickt wurde. Fügen Sie den folgenden Code (eingerückt) innerhalb der Funktion ein, um den Text in der Schaltfläche, die angeklickt wurde, auf das Symbol des aktuellen Spielers zu setzen, und deaktivieren Sie dann die Schaltfläche, damit sie nicht erneut angeklickt werden kann.

```
board_squares[x][y].text = turn
board_squares[x][y].disable()
```

Verbinden Sie nun diese Funktion mit der Schaltfläche. Suchen Sie innerhalb Ihrer Funktion `clear_board` folgenden Code:

```
button = PushButton(board, text="",
grid=[x, y], width=3)
```

Ändern Sie sie so ab, dass sie wie die folgende Zeile aussieht:

```
button = PushButton(board, text="",
grid=[x, y], width=3, command=choose_square,
args=[x,y])
```

Sie haben hier zwei Dinge hinzugefügt: Erstens hängen Sie einen Befehl an, genau wie zuvor. Wenn die Schaltfläche gedrückt wird, wird die Funktion mit diesem Namen aufgerufen. Zweitens übergeben Sie dieser Funktion auch Argumente, nämlich die Koordinaten *x* und *y* der gedrückten Schaltfläche, damit Sie diese Schaltfläche in der Liste wiederfinden können. Ihr Code sollte nun wie das Programm in der Datei **tictactoe3.py** aussehen. Speichern Sie ihn und führen Sie ihn aus. Wenn Sie nun auf eine Schaltfläche klicken, wird sie zu einem X. Leider gibt in dieser Version des Spiels vorerst dieses Symbol!

Spielerwechsel

Hier daher eine Funktion, die von X auf O und umgekehrt wechselt.

```
def toggle_player():
    global turn
    if turn == "X":
        turn = "O"
    else:
        turn = "X"
```

Fügen Sie den Code in die Funktion `global turn` ein. Dies ist notwendig, damit Sie die *globale* Version der Variable `turn` ändern dürfen, also die Variable, die Sie bereits angelegt haben. Anderenfalls erstellt Python eine lokale Variable namens *turn* und ändert diese stattdessen, aber diese Änderung wird nach der Beendigung der Funktion nicht gespeichert.

tictactoe5.py

Sprache: **Python 3**

```
# Imports ---------------
from guizero import App, Box, PushButton, Text

# Functions -------------
def clear_board():
    new_board = [[None, None, None], [None, None, None], [None, None, None]]
    for x in range(3):
        for y in range(3):
            button = PushButton(board, text="", grid=[x, y], width=3, command=choose_square, args=[x,y])
            new_board[x][y] = button
    return new_board

def choose_square(x, y):
    board_squares[x][y].text = turn
    board_squares[x][y].disable()
    toggle_player()
    check_win()

def toggle_player():
    global turn
    if turn == "X":
        turn = "O"
    else:
        turn = "X"
    message.value = "It is your turn, " + turn

def check_win():
    winner = None

    # Vertical lines
    if (
        board_squares[0][0].text == board_squares[0][1].text == board_squares[0][2].text
    ) and board_squares[0][2].text in ["X", "O"]:
        winner = board_squares[0][0]
    elif (
        board_squares[1][0].text == board_squares[1][1].text == board_squares[1][2].text
    ) and board_squares[1][2].text in ["X", "O"]:
        winner = board_squares[1][0]
    elif (
        board_squares[2][0].text == board_squares[2][1].text == board_squares[2][2].text
    ) and board_squares[2][2].text in ["X", "O"]:
        winner = board_squares[2][0]

    # Horizontal lines
    elif (
        board_squares[0][0].text == board_squares[1][0].text == board_squares[2][0].text
    ) and board_squares[2][0].text in ["X", "O"]:
        winner = board_squares[0][0]
    elif (
        board_squares[0][1].text == board_squares[1][1].text == board_squares[2][1].text
    ) and board_squares[2][1].text in ["X", "O"]:
        winner = board_squares[0][1]
    elif (
        board_squares[0][2].text == board_squares[1][2].text == board_squares[2][2].text
    ) and board_squares[2][2].text in ["X", "O"]:
        winner = board_squares[0][2]

    # Diagonals
    elif (
        board_squares[0][0].text == board_squares[1][1].text == board_squares[2][2].text
    ) and board_squares[2][2].text in ["X", "O"]:
        winner = board_squares[0][0]
    elif (
        board_squares[2][0].text == board_squares[1][1].text == board_squares[0][2].text
    ) and board_squares[0][2].text in ["X", "O"]:
        winner = board_squares[0][2]

    if winner is not None:
        message.value = winner.text + " wins!"

# Variables -------------
turn = "X"

# App -------------------
app = App("Tic tac toe")

board = Box(app, layout="grid")
board_squares = clear_board()
message = Text(app, text="It is your turn, " + turn)

app.display()
```

Sie müssen auch sicherstellen, dass das Text-Widget den aktuellen Zug des Spielers korrekt anzeigt. Aktualisieren Sie nach der *if/else*-Anweisung in der Funktion `toggle_player` die Meldung wie folgt:

```
message.value = "It is your turn, " + turn
```

Gehen Sie zurück zu Ihrer Funktion `choose_square` und rufen Sie die Funktion *toggle_player* mit `toggle_player()` auf, sobald Sie den Text gesetzt und die Schaltfläche deaktiviert haben. Ihr Code sollte nun demjenigen in der Datei **tictactoe4.py** ähneln. Speichern und testen Sie das Programm erneut. Die angeklickten Quadrate werden nun abwechselnd entweder mit X oder O bezeichnet werden.

Haben wir einen Gewinner?

Zum Schluss müssen Sie eine Funktion schreiben, die prüft, ob es eine Reihe, Spalte oder Diagonale mit drei X oder O gibt, und wenn ja, den Gewinner des Spiels meldet.

Obwohl es wenig elegant erscheint, ist es der einfachste Weg zu prüfen, ob jemand gewonnen hat, wenn Sie die Prüfungen für jede vertikale, horizontale und diagonale Zeile einzeln codieren. Der folgende Code ist für eine vertikale, eine horizontale und eine diagonale Linie geeignet. Können Sie den Rest hinzufügen?

```
def check_win():
    winner = None

    # Vertical lines
```

Globale Variablen

Es ist normalerweise keine besonders gute Idee, globale Variablen zu verwenden, denn wenn Sie viele Funktionen in einem großen Programm haben, kann es extrem verwirrend werden, welcher Code wann den Wert einer Variablen ändert. In einem kleinen Programm wie diesem ist es jedoch nicht allzu schwierig, den Überblick zu behalten.
Denken Sie daran, dass es möglich ist, den Wert einer globalen Variablen aus einer Funktion heraus zu lesen und zu verwenden, ohne sie als global zu deklarieren. Um ihren Wert jedoch zu ändern, müssen Sie diese Variable explizit deklarieren. Die Funktionen in diesem Programm (und den meisten GUI-Programme in dieser Tutorial-Serie) verändern tatsächlich die Werte Ihrer Widgets als globale Variablen. Wenn zum Beispiel jemand das Spiel gewinnt, weisen sie **message.value** einen Wert zu, um anzuzeigen, wer gewonnen hat:

```
message.value = winner.text + " wins!"
```

In diesem Beispiel ist **message** eine globale Variable. Wieso können wir ihren Wert ändern, ohne sie als global zu deklarieren? Weil wir eine Eigenschaft des message-Widgets verwenden: **value**. Im Wesentlichen sagt dieser Code: "Hallo Python, kennst du das Widget namens *message*? Könntest du bitte seine Eigenschaft *value* ändern?" Python erlaubt die Änderung von Objekteigenschaften im globalen Bereich, aber es erlaubt Ihnen nicht, den Wert einer Variablen direkt zu ändern, ohne sie als global zu deklarieren.


```
    if (
        board_squares[0][0].text == board_
squares[0][1].text == board_squares[0][2].text
    ) and board_squares[0][2].text in ["X", "O"]:
        winner = board_squares[0][0]

    # Horizontal lines
    elif (
        board_squares[0][0].text == board_
squares[1][0].text == board_squares[2][0].text
    ) and board_squares[2][0].text in ["X", "O"]:
        winner = board_squares[0][0]

    # Diagonals
    elif (
        board_squares[0][0].text == board_
squares[1][1].text == board_squares[2][2].text
    ) and board_squares[2][2].text in ["X", "O"]:
        winner = board_squares[0][0]
```

Beachten Sie, dass die Funktion damit beginnt, eine boolesche Variable namens `winner` zu erzeugen. Wenn zum Zeitpunkt der Ausführung der langen *if/elif*-Anweisung der Wert dieser Variablen *True* ist, wissen Sie, dass jemand das Spiel gewonnen hat. Setzen Sie, nach dem Hinzufügen der verbleibenden Überprüfungen der Gewinnzeile, am Ende der Funktion noch den erforderlichen Code ein, um die Displaymeldung zu ändern, wenn es einen Gewinner gegeben hat:

```
if winner is not None:
    message.value = winner.text + " wins!"
```

Sie müssen nun sicherstellen, dass diese Funktion jedes Mal aufgerufen wird, wenn ein X oder O gesetzt wird, was jedem Drücken einer Schaltfläche entspricht. Fügen Sie am Ende der Funktion `choose_square` einen Aufruf von `check_win` hinzu, nur für den Fall, dass das gewählte Quadrat das Gewinner-Quadrat war.

Ihr Programm sollte nun wie dasjenige in der Datei **tictactoe5.py** aussehen. Starten Sie es und testen Sie das Spiel. Wenn Sie die betreffenden Zeilen in der Funktion `check_win` richtig geschrieben haben, sollte das Spiel korrekt erkennen, wenn ein Spieler gewonnen hat.

Der letzte Schliff

Im jetzigen Zustand lässt das Spiel Sie auch nach einem Sieg weiterspielen, bis alle Felder ausgewählt sind. Es wird Ihnen auch nicht mitgeteilt, ob das Spiel unentschieden endete. Sie könnten an diesem Punkt aufhören, aber wenn Sie das Spiel perfektionieren wollen, könnten Sie es mit ein paar weiteren kleinen Details noch ausgefeilter machen. Fügen wir zunächst weite Programmzeilen hinzu, um zu erkennen, ob das Spiel unentschieden ist. Dies ist der Fall, wenn alle Quadrate entweder ein X oder ein O enthalten und niemand gewonnen hat. Erstellen Sie im Funktionsbereich eine neue Funktion namens `moves_taken`:

```
def moves_taken():
```

Sie werden diese Funktion verwenden, um die Anzahl der ausgeführten Züge zu zählen, also legen wir eine Variable an, um den Zähler zu halten, beginnend bei 0..

```
def moves_taken():
  moves = 0
```

Erinnern Sie sich daran, dass wir bei der Erstellung von `board_squares` eine verschachtelte Schleife verwendet haben, um alle Quadrate auf dem Raster zu erzeugen? Wir brauchen hier eine weitere verschachtelte Schleife, um jedes einzelne Feld zu überprüfen und festzustellen, ob es mit einem X oder O ausgefüllt wurde oder ob es leer ist. Fügen Sie diesen Code für eine geschachtelte Schleife in die Funktion `moves_taken` ein:

```
for row in board_squares:
    for col in row:
```

Innerhalb der Schleife müssen wir prüfen, ob das betreffende Feld mit einem X bzw. einem O ausgefüllt ist. Wenn es ausgefüllt ist (egal womit), fügen Sie der Variablen `moves` eine 1 hinzu, um festzuhalten, dass dieses Feld gezählt wurde.

06-tictactoe.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.  from guizero import App, Box, PushButton, Text
003.
004.  # Functions -------------
005.  def clear_board():
006.      new_board = [[None, None, None],
      [None, None, None], [None, None, None]]
007.      for x in range(3):
008.          for y in range(3):
009.              button = PushButton(board, text="",
      grid=[x, y], width=3, command=choose_square,
      args=[x,y])
010.              new_board[x][y] = button
011.      return new_board
012.
013.  def choose_square(x, y):
014.      board_squares[x][y].text = turn
015.      board_squares[x][y].disable()
016.      toggle_player()
017.      check_win()
018.
019.  def toggle_player():
020.      global turn
021.      if turn == "X":
022.          turn = "O"
023.      else:
024.          turn = "X"
025.      message.value = "It is your turn, " + turn
026.
027.  def check_win():
028.      winner = None
029.
030.      # Vertical lines
031.      if (
032.          board_squares[0][0].text ==
      board_squares[0][1].text == board_squares[0][2].text
033.      ) and board_squares[0][2].text in ["X", "O"]:
034.          winner = board_squares[0][0]
035.      elif (
036.          board_squares[1][0].text ==
      board_squares[1][1].text == board_squares[1][2].text
037.      ) and board_squares[1][2].text in ["X", "O"]:
038.          winner = board_squares[1][0]
039.      elif (
040.          board_squares[2][0].text ==
      board_squares[2][1].text == board_squares[2][2].text
041.      ) and board_squares[2][2].text in ["X", "O"]:
042.          winner = board_squares[2][0]
043.
044.      # Horizontal lines
045.      elif (
046.          board_squares[0][0].text ==
      board_squares[1][0].text == board_squares[2][0].text
047.      ) and board_squares[2][0].text in ["X", "O"]:
048.          winner = board_squares[0][0]
049.      elif (
050.          board_squares[0][1].text ==
      board_squares[1][1].text == board_squares[2][1].text
051.      ) and board_squares[2][1].text in ["X", "O"]:
052.          winner = board_squares[0][1]
053.      elif (
054.          board_squares[0][2].text ==
      board_squares[1][2].text == board_squares[2][2].text
055.      ) and board_squares[2][2].text in ["X", "O"]:
056.          winner = board_squares[0][2]
057.
058.      # Diagonals
059.      elif (
060.          board_squares[0][0].text ==
      board_squares[1][1].text == board_squares[2][2].text
061.      ) and board_squares[2][2].text in ["X", "O"]:
062.          winner = board_squares[0][0]
063.      elif (
064.          board_squares[2][0].text ==
      board_squares[1][1].text == board_squares[0][2].text
065.      ) and board_squares[0][2].text in ["X", "O"]:
067.          winner = board_squares[0][2]
068.
069.      if winner is not None:
070.          message.value = winner.text + " wins!"
071.      elif moves_taken() == 9:
072.          message.value = "It's a draw"
073.
074.  def moves_taken():
075.      moves = 0
076.      for row in board_squares:
077.          for col in row:
078.              if col.text == "X" or col.text == "O":
080.                  moves = moves + 1
081.      return moves
082.
083.
084.  # Variables -------------
085.  turn = "X"
086.
087.  # App -------------------
088.  app = App("Tic tac toe")
089.
090.  board = Box(app, layout="grid")
091.  board_squares = clear_board()
092.  message = Text(app, text="It is your turn, " + turn)
093.
094.  app.display()
```

```
if col.text == "X" or col.text == "O":
    moves = moves + 1
```

Fügen Sie schließlich, sobald die Schleifen beendet sind, eine *return*-Anweisung hinzu, um die Anzahl der ausgeführten Züge zurückzugeben.

```
return moves
```

Rufen Sie nun diese Funktion innerhalb der Funktion `check_win` auf, um zu prüfen, ob ein Unentschieden vorliegt. Fügen Sie diesen Code nach dem Code ein, der auf einen Gewinner prüft:

```
if winner is not None:
    message.value = winner.text + " wins!"

# Add this code
elif moves_taken() == 9:
    message.value = "It's a draw"
```

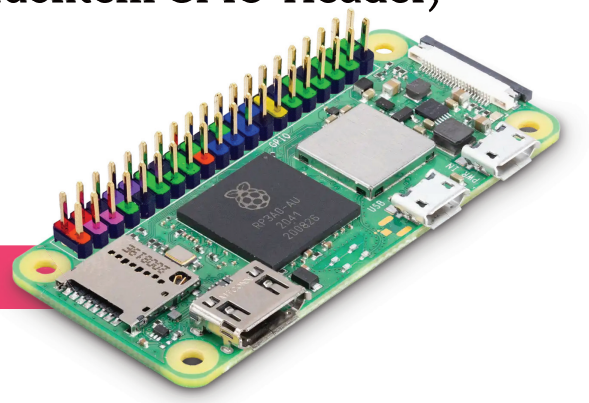
Ihr Code sollte nun demjenigen in der Datei **06-tictactoe.py** gleichen. Beim Ausführen prüft das Spiel nun, ob neun Züge gemacht wurden; wenn ja, ändert es die Meldung, um anzuzeigen, dass die Partie unentschieden war.

Spotlight

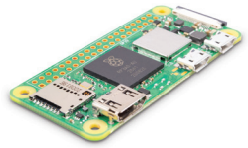
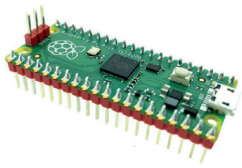
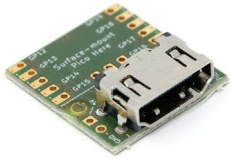
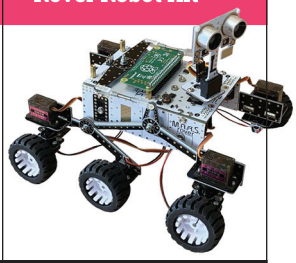
Raspberry Pi Zero 2 WH (mit bestücktem GPIO-Header)

Der Raspberry Pi Zero 2 W basiert auf einem System-in-Package (SiP), das den Broadcom BCM2710A1-Chip mit 512 MB LPDDR2 SDRAM integriert. Der aktualisierte Prozessor verhilft dem Raspberry Pi Zero 2 W zu 40% mehr Single-Thread-Leistung und fünfmal mehr Multi-Thread-Leistung als der ursprüngliche Single-Core-Raspberry Pi Zero.
In Kooperation mit Eurocircuits bietet Elektor den Raspberry Pi Zero 2 W auch mit bereits (professionell) angelötetem, farbig codiertem GPIO-Header an.

www.elektor.de/20157



RPi Bestseller

1. Raspberry Pi Zero 2 W	2. Raspberry Pi Pico RP2040	3. Raspberry Pi 4 (4 GB RAM)	4. DVI Sock für Raspberry Pi Pico	5. 4tronix M.A.R.S. Rover Robot Kit
				
www.elektor.de/19906	www.elektor.de/19568	www.elektor.de/18964	www.elektor.de/19925	www.elektor.de/19996

BÜCHER

Raspberry Pi für Funkamateure

Dieses Buch richtet sich an Funkamateure, die lernen wollen, wie man mit dem Raspberry Pi Elektronik-Projekte baut. Das Buch eignet sich für die gesamte Bandbreite von Anfängern bis hin zu alten Hasen im Amateurfunk.

www.elektor.de/20078

MSP430 Microcontroller Essentials

Der MSP430 ist eine beliebte Familie von Mikrocontrollern von Texas Instruments.
In diesem (englischsprachigen) Buch werden wir mit dem kleinsten Typ arbeiten, dem leistungsstarken MSP430G2553.
Wir werden uns die Fähigkeiten dieses (im P-DIP20-Gehäuse erhältlichen) Mikrocontrollers im Detail ansehen, da er sich gut für DIY-Projekte eignet.

www.elektor.de/20112

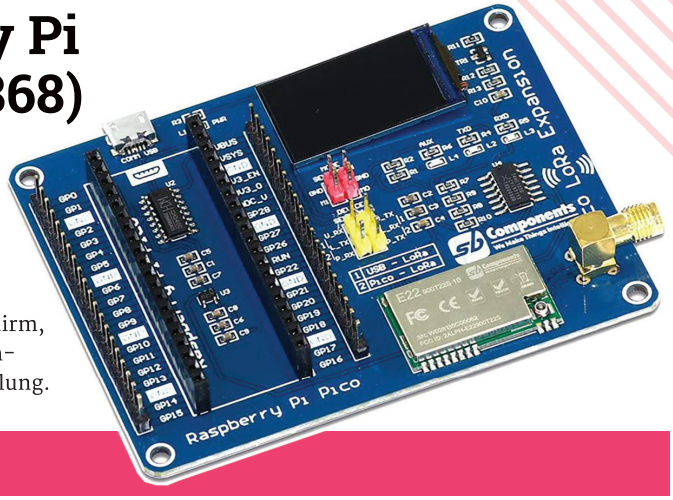
Kickstart to Python 3

Dieses (englischsprachige) Buch bietet Anfängern den idealen Einstieg in die Python-Programmierung. Das Buch ist in 10 Kapitel unterteilt. Im ersten Kapitel werden die Leser in die Grundlagen von Python eingeführt.
Es enthält eine detaillierte Anleitung zur Installation auf verschiedenen Plattformen wie macOS, Windows, FreeBSD und Linux. Es behandelt auch die anderen Aspekte der Python-Programmierung wie IDEs und Paketmanager.

www.elektor.de/20106

Spotlight

SB Components Raspberry Pi Pico LoRa Expansion (EU868)



Pico LoRa Expansion ist eine Datenübertragungsplatine mit geringem Stromverbrauch. Sie verfügt über einen integrierten CH340-USB-zu-UART-Wandler, einen Spannungspegelumsetzer (74HC125V), einen E22-900T22S/E22-400T22S-SMA-Antennenanschluss, der das 868-MHz-Frequenzband abdeckt, einen integrierten 1,14-Zoll-LCD-Bildschirm, einen IPEX-Antennenanschluss und eine LoRa-Spread-Spectrum-Modulationstechnologie mit automatischer Mehrstufenwiederholung.

www.elektor.de/20096

KITS, MODULE & ZUBEHÖR

Waveshare Ethernet/USB Hub Box für Raspberry Pi Zero (1x RJ45, 3x USB 2.0)

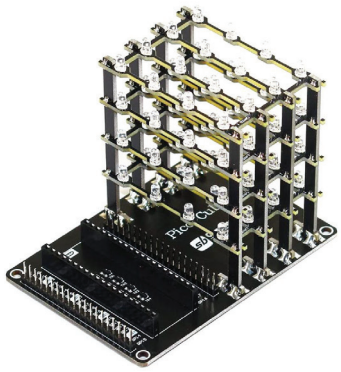
Die ETH-USB-Hub-Box ist ein Hub-Kit mit ETH/USB Hub HAT (B). Es ist perfekt auf die RPi Zero-Serie zugeschnitten, klein in der Größe, jeder Ausschnitt des Gehäuses ist genau auf den Anschluss ausgerichtet.

www.elektor.de/20084

SB Components Raspberry Pi Pico LED Cube (4x4x4)

Pico Cube ist ein 4x4x4 LED-Würfel HAT für Raspberry Pi Pico mit 5 V Betriebsspannung. Pico Cube ist ein monochromatischer blauer Würfel mit 64 LEDs, mit dem es Spaß macht, das Programmieren zu lernen.

www.elektor.de/20102



Seeed Studio re_computer case für Raspberry Pi, BeagleBone und Jetson Nano

Das re_computer-Gehäuse wurde speziell für das re_computer-System entwickelt, mit einer abnehmbaren Acrylabdeckung auf der Oberseite und einer stapelbaren Struktur, um die Möglichkeiten zu erweitern. Es ist mit den beliebtesten SBCs kompatibel, darunter Raspberry Pi, BeagleBone und Jetson Nano.

www.elektor.de/20093

Teil 06

Grafische Benutzeroberflächen mit Python: Destroy the dots

Lesen Sie, wie man mit einem speziellen Widget ein interessantes Spiel programmiert.

Sie haben im Tic-tac-toe-Spiel gesehen, wie man eine grafische Oberfläche erstellt, um dem Spieler ein rasterähnliches Spielfeld zu präsentieren. Eine alternative Möglichkeit bietet das Guizero-Widget namens *Waffle*, das sofort ein sichtbares Raster erzeugen kann und sehr nützlich für die Programmierung von zahlreichen Spielen ist.

Dieses Spiel heißt 'Destroy the dots' (zerstöre die Punkte) und entstand, weil Martin es für eine gute Idee hielt, ein Waffle-Widget mit einer Mischung aus Quadraten und farbigen Punkten zu versehen. Wir verwenden hier jedoch den Begriff „Kreise", weil man sich unter einem Punkt normalerweise das Gebilde vorstellt, das sich am Ende eines Satzes befindet und die zu zerstörenden „Punkte" außerdem mit den im Spiel zu erreichenden Punkten verwechselt werden könnten.

Worum geht's?

In diesem Spiel müssen Sie plötzlich auftauchende, rote Kreise zerstören, bevor diese die Oberhand gewinnen! Das Spielfeld besteht aus einem Gitter aus 5 mal 5 Quadraten. Die Quadrate werden sich nach und nach in rote Kreise verwandeln. Um einen Kreis zu zerstören, klicken Sie ihn an, und er verwandelt sich wieder in ein Quadrat. Das Ziel des Spiels ist es, so lange wie möglich zu „überleben", bevor Sie von den immer schneller auftauchenden Kreisen regelrecht überrannt werden (**Bild 1**).

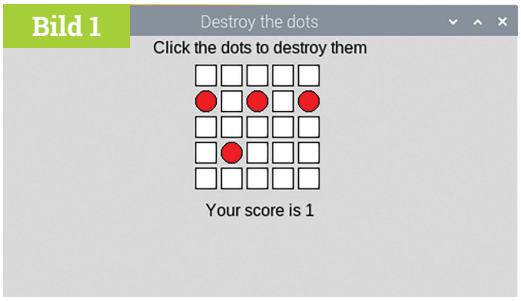


Bild 1 Zerstören Sie die roten Kreise, bevor diese das Spielfeld beherrschen.

Einrichten des Spiels

Beginnen wir damit, ein Guizero-Programm zu erstellen, das die Spielanleitung und ein Waffle enthält. Inzwischen sollten Sie mit dem Aufbau eines Standard-Guizero-Programms mit Abschnitten für Importe, Funktionen, Variablen und der App selbst vertraut sein.

Erstellen Sie also zunächst eine App und fügen Sie darin ein Text-Widget für die Spielanleitung und ein Waffle-Widget für das Spielbrett ein:

```
# Imports ---------------
from guizero import App, Text, Waffle

# App -------------------
app = App("Destroy the dots")

instructions = Text(app, text="Click the
dots to destroy them")
board = Waffle(app)

app.display()
```

Wenn Sie Ihr Programm ausführen, sehen Sie zu Anfang nur ein kleines 3 × 3-Gitter aus weißen Quadraten. Wenn Sie Ihr Raster größer machen wollen, können Sie die Eigenschaften *width* und *height* zu Waffle hinzufügen:

```
board = Waffle(app, width=5, height=5)
```

Ihr Code sollte nun dem im Listing von **destroy1.py** ähneln.

Kreise einsetzen

Als nächstes müssen Sie eine Funktion schreiben, die ein per Zufall ausgewähltes Quadrat auf dem Spielfeld findet und es in einen Punkt verwandelt. Beginnen Sie eine neue Funktion im Funktionsbereich mit dem Namen `add_dot()`:

```
def add_dot():
```

Um ein zufälliges Quadrat auf das Spielfeld auszuwählen, müssen Sie in der Lage sein, ein zufälliges Paar von Ganzzahlen als Koordinaten zu erzeugen. Fügen Sie in Ihrem imports-Abschnitt eine Zeile ein, um die Funktion `randint` aus der `random`-Bibliothek zu importieren, mit der Sie eine zufällige Ganzzahl erzeugen können.

```
from random import randint
```

destroy1.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text, Waffle
004.
005.  # Variables -------------
006.
007.  # Functions -------------
008.
009.  # App -------------------
010.
011.  app = App("Destroy the dots")
012.
013.  instructions = Text(app, text="Click the dots to destroy them")
014.  board = Waffle(app, width=5, height=5)
015.
016.  app.display()
```

destroy2.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text, Waffle
004.  from random import randint
005.
006.  # Variables -------------
007.
008.  GRID_SIZE = 5
009.
010.  # Functions -------------
011.
012.  def add_dot():
013.      x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
014.      while board[x, y].dotty == True:
015.          x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
016.      board[x, y].dotty = True
017.      board.set_pixel(x, y, "red")
018.
019.  # App -------------------
020.
021.  app = App("Destroy the dots")
022.
023.  instructions = Text(app, text="Click the dots to destroy them")
024.  board = Waffle(app, width=5, height=5)
025.  add_dot()
026.
027.  app.display()
```

Erzeugen wir zwei Variablen, *x* und *y*, die Sie für die Koordinaten eines Quadrates auf dem Gitter verwenden können. Beginnen Sie Ihren Code innerhalb Ihrer Funktion `add_dot()` wie folgt:

```
x, y = randint(0,4), randint(0,4)
```

Sie haben damit zwei zufällige Ganzzahlen zwischen 0 und 4 erzeugt, weil die Breite und Höhe des Gitters auf 5 festgelegt wurde, und die Zeilen und Spalten bei 0 beginnen. Wenn Sie zuvor andere Werte gewählt haben, müssen Sie die Werte hier an die Größe Ihres Gitters anpassen. Es gibt jedoch eine bessere Möglichkeit, solche Dinge zu verwalten (siehe Kasten *Konstanten verwenden*).

Kreis oder Rechteck?

Sie können nun die folgende Funktion verwenden, um eine zufällige Koordinate auf dem Gitter zu erzeugen:

```
def add_dot():
    x, y = randint(0,GRID_SIZE-1),
randint(0,GRID_SIZE-1)
```

Zu diesem Zeitpunkt wissen Sie nicht, ob die zufällig gewählte Koordinate bereits ein Kreis ist oder nicht. Dies mag zu Beginn des Spiels keinen Unterschied machen, wenn das Spielfeld überwiegend aus Quadraten besteht, aber wenn das Spielfeld mit Kreisen gefüllt wird, müssen Sie sicherstellen, dass das Feld tatsächlich ein Quadrat ist. Eine Möglichkeit, dies zu erreichen, besteht darin, eine Schleife laufen zu lassen, die prüft, ob das gewählte Feld bereits ein Kreis ist, und wenn dies der Fall ist, ein anderes zufälliges Feld auszuwählen:

```
x, y = randint(0,GRID_SIZE-1),
randint(0,GRID_SIZE-1)
while board[x, y].dotty == True:
    x, y = randint(0,GRID_SIZE-1),
randint(0,GRID_SIZE-1)
```

Es ist leicht nachvollziehbar, dass dies keine besonders effiziente Methode ist, um ein zufälliges Feld zu wählen, das noch nicht in einen Kreis verwandelt wurde, aber für dieses Spiel reicht es aus. Sobald diese Schleife beendet ist, können Sie sicher sein, dass die zufällig gewählte *x*- und *y*-Koordinaten definitiv ein Quadrat ist. Wandeln wir es in einen roten Kreis um. Fügen dazu Sie nach (nicht innerhalb) Ihrer `while`-Schleife die folgenden Zeilen ein:

```
board[x, y].dotty = True
board.set_pixel(x, y, "red")
```

Fügen Sie einen Aufruf Ihrer neuen Funktion

add_dot() im app-Abschnitt hinzu, und zwar hinter der Stelle, an der das Spielfeld erstellt wird. Ihr Programm sollte nun **destroy2.py** ähneln. Wenn Sie es ausführen, sollten Sie einen einzelnen an einer zufälligen Stelle erscheinenden, roten Kreis im Raster sehen. Wenn Sie das Programm erneut ausführen, wird der Kreis wahrscheinlich an einer anderen zufälligen Stelle erscheinen (**Bild 2**).

Zerstören Sie den Kreis

Bis jetzt gibt es nur einen Kreis - zerstören wir ihn! Keine Sorge: Sie werden später weitere Kreise zum Zerstören hinzufügen, aber sobald Sie einen Kreis zerstören können, können Sie auch alle anderen zerstören!

Erstellen Sie eine neue Funktion in Ihrem Funktionsbereich mit einem passenden Namen - **destroy_dot** - und geben Sie ihr zwei Parameter, *x* und *y*.

```
def destroy_dot(x, y):
```

Diese Funktion prüft, ob das Gebilde mit den Koordinaten *x*, *y* ein Kreis ist (und nicht ein Quadrat). Sie können dazu den gleichen Code wie zum Erzeugen eines Kreises verwenden - der Code `board[x, y].dotty` gibt *True* zurück, wenn die Koordinate ein Kreis ist, und *False*, wenn sie ein Quadrat ist.

```
if board[x,y].dotty == True:
```

Wenn die Koordinate ein Kreis ist, wird sie in ein Quadrat umgewandelt, indem die Eigenschaft *dotty* auf *False* gesetzt wird. Auch die Farbe wird wieder auf Weiß geändert:

```
if board[x,y].dotty == True:
    board[x,y].dotty = False
    board.set_pixel(x, y, "white")
```

Diese Funktion muss immer dann ausgelöst werden, wenn das Spielfeld angeklickt wird. Suchen Sie die bereits vorhandene Codezeile, die das Spielfeld erstellt, und fügen Sie einen Befehl wie diesen ein:

```
board = Waffle(app, width=GRID_SIZE,
height=GRID_SIZE, command=destroy_dot)
```

Dadurch wird die Funktion `destroy_dot` aufgerufen, sobald ein Element auf dem Spielfeld angeklickt wird.
Beachten Sie, dass ein Waffle-Widget automatisch zwei Parameter an jede Befehlsfunktion übergibt; dies sind immer die **x**- und **y**-Koordinaten des Pixels, das angeklickt wurde, um den Befehl auszulösen.

Ihr Code sollte nun wie der in der Datei **destroy3.py**

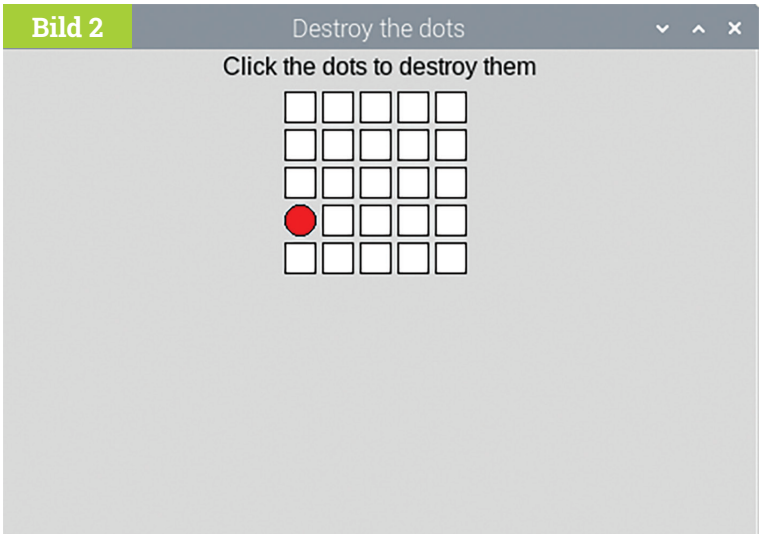
> An diesem Punkt wissen Sie noch nicht, ob an der zufällig gewählten Koordinate ein Kreis erscheint oder nicht

destroy3.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text, Waffle
004.  from random import randint
005.
006.  # Variables -------------
007.
008.  GRID_SIZE = 5
009.
010.  # Functions -------------
011.
012.  def add_dot():
013.      x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
014.      while board[x, y].dotty == True:
015.          x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
016.      board[x, y].dotty = True
017.      board.set_pixel(x, y, "red")
018.
019.  def destroy_dot(x, y):
020.      if board[x,y].dotty == True:
021.          board[x,y].dotty = False
022.          board.set_pixel(x, y, "white")
023.
024.
025.  # App -------------------
026.
027.  app = App("Destroy the dots")
028.
029.  instructions = Text(app, text="Click the dots to destroy them")
030.  board = Waffle(app, width=GRID_SIZE, height=GRID_SIZE,
031.  command=destroy_dot)
032.  add_dot()
033.
034.  app.display()
```

Bild 2



▲ **Bild 2** Erzeugen eines zufälligen roten Kreises.

Konstanten verwenden

Die Höhe und Breite von *Waffle* wurde im Programm an einer bestimmten Stelle auf 5 gesetzt. Wenn Sie die Größe des Gitters ändern wollen, müssen Sie diese Stelle im Programm finden und die Zahl ändern, was unübersichtlich sein kann. Eine bessere Programmierpraxis wäre es, eine Konstante in Ihrem Variablenabschnitt namens GRID_SIZE zu definieren und sie gleich 5 zu setzen:

```
GRID_SIZE = 5
```

Dann können Sie, anstatt die Abmessungen von Waffle mit der Zahl 5 zu definieren, Folgendes schreiben

```
board = Waffle(app, width=GRID_SIZE, height=GRID_SIZE)
```

Wenn Sie sich später entscheiden, die Größe des Rasters zu ändern, können Sie einfach den Wert dieser Konstante ändern. Wenn Sie bereits beim Schreiben des Programms an solche Dinge denken, vermeiden Sie spätere Kopfschmerzen.

destroy4.py

▸ Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text, Waffle
004.  from random import randint
005.
006.  # Variables -------------
007.
008.  GRID_SIZE = 5
009.
010.  # Functions -------------
011.
012.  def add_dot():
013.      x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
014.      while board[x, y].dotty == True:
015.          x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
016.      board[x, y].dotty = True
017.      board.set_pixel(x, y, "red")
018.      board.after(1000, add_dot)
019.
020.  def destroy_dot(x,y):
021.      if board[x,y].dotty == True:
022.          board[x,y].dotty = False
023.          board.set_pixel(x, y, "white")
024.
025.
026.  # App -------------------
027.
028.  app = App("Destroy the dots")
029.
030.  instructions = Text(app, text="Click the dots to destroy them")
031.  board = Waffle(app, width=GRID_SIZE, height=GRID_SIZE,
032.  command=destroy_dot)
033.  board.after(1000, add_dot)
034.
035.  app.display()
```

aussehen. Testen Sie Ihr Programm, indem Sie es ausführen und auf den Kreis klicken. Der Kreis muss sich nun wieder in ein weißes Quadrat verwandeln. Wenn Sie auf ein Quadrat klicken, das keinen Kreis enthält, sollte nichts passieren.

Mehr Kreise!

Jetzt ist es an der Zeit, das Spiel zu einer echten Herausforderung zu machen, indem wir ständig neue Kreise auftauchen lassen. Fangen wir damit an, jede Sekunde einen neuen zufälligen Kreis hinzuzufügen. Dazu müssen Sie einen Aufruf der Funktion `add_dot` nach jeder Sekunde einplanen, indem Sie eine eingebaute Funktion von Guizero namens `after` verwenden.

Entfernen Sie in Ihrem App-Abschnitt den Aufruf von `add_dot()` und ersetzen Sie ihn durch eine neue Codezeile:

```
board.after(1000, add_dot)
```

Diese Codezeile bedeutet: *Rufe nach 1000 Millisekunden (1 Sekunde) die Funktion* `add_dot` *auf.*

Wenn Sie das Programm jetzt ausführen, erhalten Sie immer noch einen einzelnen Kreis, aber er erscheint erst nach einer Verzögerung von einer Sekunde auf dem Raster. Suchen Sie Ihre Funktion `add_dot` und fügen Sie die gleiche Codezeile am Ende der Funktion ein.

Dadurch wird jedes Mal, wenn ein neuer Kreis hinzugefügt wird, ein neuer Aufruf von `add_dot` gestartet. Der nächste Kreis wird ebenfalls mit einer Sekunde Verzögerung erzeugt, so dass, wenn Sie das Programm ausführen, jede Sekunde ein neuer Kreis auf dem Gitter erscheint (**Bild 3**).

Versuchen Sie, Ihr Programm zu starten, das nun wie die Version **destroy4.py** aussehen sollte. Da Sie bereits die Methode zum „Zerstören“ eines Kreises geschrieben haben, sollte ein Klick auf einen beliebigen Kreis zu dessen Entfernung führen. Wenn Sie das Spiel jedoch eine Weile spielen, werden Sie feststellen, dass es ziemlich einfach ist, mit dem Tempo von einem Punkt pro Sekunde Schritt zu halten, und es fast unmöglich ist, das Spiel zu verlieren.

Sie müssen daher noch zwei Dinge hinzufügen – einen Punktestand, um zu verfolgen, wie viele Kreise Sie zerstört haben, und eine Möglichkeit, das Spiel schwieriger zu machen, damit es zu einer echten Herausforderung wird.

Punktzahl-Angabe hinzufügen

Das Hinzufügen einer Punktzahl ist ziemlich einfach und erfolgt in drei Schritten:

- Fügen Sie eine Variable hinzu, um den Punktestand zu speichern; Ihr Wert sollte bei 0 beginnen.

- Zeigen Sie auf der GUI eine Meldung mit dem aktuellen Punktestand an.
- Jedes Mal, wenn die Funktion **destroy_dot** aufgerufen und ein Kreis zerstört wird, addieren Sie eine 1 zu *score* und aktualisieren die Anzeige der Meldung.

Versuchen Sie, den Code mit dem bereits Gelernten selbst zu ergänzen.

Hinweis: Um die Variable score von der Funktion **destroy_dot** aus zu aktualisieren, müssen Sie sie als global deklarieren.
Tipp: Wenn Sie die Fehlermeldung *variable is referenced before assignment* (auf die Variable wird vor Zuweisung eines Wertes zugegriffen) erhalten, stellen Sie sicher, dass Ihr Variablenabschnitt vor Ihrem Funktionsabschnitt steht. Die Lösung finden Sie am Ende des Artikels.

Lösung der Aufgabe: Punktzahl-Angabe hinzufügen

Fügen Sie zunächst eine Variable in Ihren Variablen-Abschnitt ein:

```
score = 0
```

Als nächstes fügen Sie ein neues Text-Widget in den app-Abschnitt ein, um den Punktestand anzuzeigen:

```
score_display = Text(app, text="Your score
is " + str(score))
```

Schließlich addieren Sie jedes Mal eine 1 zum Punktestand, wenn ein Punkt zerstört wird:

```
def destroy_dot(x,y):

    # Declare score global
    global score

    # This code already exists
```

Grafische Benutzeroberflächen mit Python erstellen

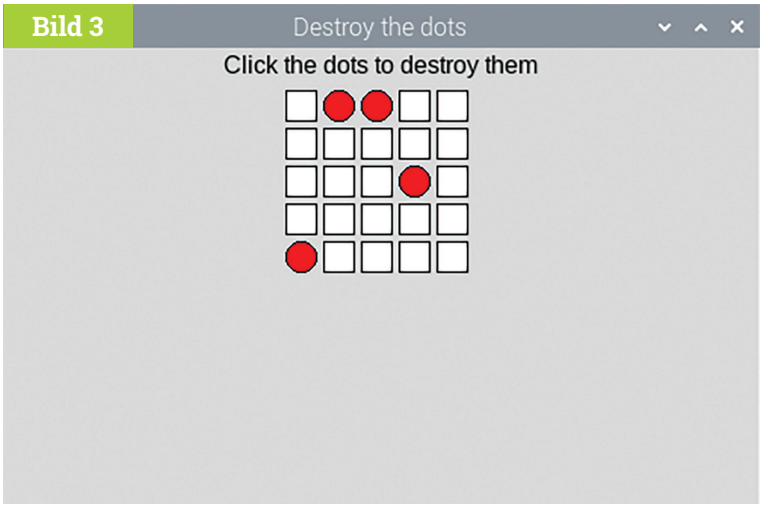
Weitere Anleitungen, wie Sie mit Guizero eigene GUIs erstellen können, finden Sie in diesem Buch mit dem Titel: *Create Graphical User Interfaces with Python*. Die 156 Seiten sind vollgepackt mit wichtigen Informationen und einer Reihe von spannenden Projekten.
magpi.cc/pythongui

destroy5.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text, Waffle
004.  from random import randint
005.
006.  # Variables -------------
007.
008.  GRID_SIZE = 5
009.  score = 0
010.
011.  # Functions -------------
012.
013.  def add_dot():
014.      x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
015.      while board[x, y].dotty == True:
016.          x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
017.      board[x, y].dotty = True
018.      board.set_pixel(x, y, "red")
019.      board.after(1000, add_dot)
020.
021.  def destroy_dot(x,y):
022.      global score
023.      if board[x,y].dotty == True:
024.          board[x,y].dotty = False
025.          board.set_pixel(x, y, "white")
026.          score += 1
027.          score_display.value = "Your score is " + str(score)
028.
029.
030.  # App -------------------
031.
032.  app = App("Destroy the dots")
033.
034.  instructions = Text(app, text="Click the dots to destroy them")
035.  board = Waffle(
      app, width=GRID_SIZE, height=GRID_SIZE, command=destroy_dot)
036.  board.after(1000, add_dot)
037.  score_display = Text(app, text="Your score is " + str(score))
038.
039   app.display()
```

Bild 3



▲ **Bild 3** Jede Sekunde wird ein neuer Kreis auf das Spielfeld gebracht.

```
        if board[x,y].dotty == True:
            board[x,y].dotty = False
            board.set_pixel(x, y, "white")

            # Add 1 to score and display it on
the GUI
            score += 1
            score_display.value = "Your score is
" + str(score)
```

destroy6.py

▶ Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text, Waffle
004.  from random import randint
005.
006.  # Variables -------------
007.
008.  GRID_SIZE = 5
009.  score = 0
010.
011.  # Functions -------------
012.
013.  def add_dot():
014.      x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
015.      while board[x, y].dotty == True:
016.          x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
017.      board[x, y].dotty = True
018.      board.set_pixel(x, y, "red")
019.
020.      speed = 1000
021.      if score > 30:
022.          speed = 200
023.      elif score > 20:
024.          speed = 400
025.      elif score > 10:
026.          speed = 500
027.      board.after(speed, add_dot)
028.
029.  def destroy_dot(x,y):
030.      global score
031.      if board[x,y].dotty == True:
032.          board[x,y].dotty = False
033.          board.set_pixel(x, y, "white")
034.          score += 1
035.          score_display.value = "Your score is " + str(score)
036.
037.
038.  # App -------------------
039.
040.  app = App("Destroy the dots")
041.
042.  instructions = Text(app, text="Click the dots to destroy them")
043.  board = Waffle(
      app, width=GRID_SIZE, height=GRID_SIZE, command=destroy_dot)
044.  board.after(1000, add_dot)
045.  score_display = Text(app, text="Your score is " + str(score))
046.
047.  app.display()
```

Ihr Code (ohne die optionalen Kommentare) sollte nun der Datei destroy5.py ähneln. Testen Sie Ihr Spiel. Ihr Punktestand muss nun jedes Mal um 1 steigen, wenn Sie auf einen Punkt klicken.

Setzen Sie den Spieler unter Druck

Jetzt, da Sie den Punktestand des Spielers verfolgen können, können Sie diesen nutzen, um den Spieler unter Druck zu setzen. Erinnern Sie sich, dass Sie einen Aufruf von *after* innerhalb der Funktion `add_dot` verwendet haben, um einen weiteren Punkt in 1000 Millisekunden zu erzeugen? Gehen Sie zurück und suchen Sie diese Zeile – Sie werden sie ein wenig ändern: Erstellen Sie zunächst eine variable `speed` und setzen Sie diese auf 1000. Die Zeitverzögerung wird nun (wie in den beiden unteren Zeilen gezeigt) nicht mehr durch einen Absolutwert, sondern durch die Variable *speed* bestimmt. Da diese im Moment ebenfalls noch einen Wert von 1000 Millisekunden besitzt, wird dies vorläufig noch keine Auswirkungen auf das Spiel haben.

```
speed = 1000
board.after(speed, add_dot)
```

Die folgenden Zeilen zeigen, wie Sie den Zeitdruck bei laufendem Spiel jedoch immer weiter erhöhen können, und zwar in Abhängigkeit vom aktuellen Spielstand:

```
speed = 1000
if score > 30:
    speed = 200
elif score > 20:
    speed = 400
elif score > 10:
    speed = 500
board.after(speed, add_dot)
```

Hier sehen Sie, dass, wenn der Spieler mehr als 10 Punkte hat, die neuen Kreise alle 500 ms erscheinen, wenn er mehr als 20 Punkte hat, erscheint ein Kreis alle 400 ms, und so weiter. Das macht das Spiel umso schwieriger, je mehr Punkte Sie erreicht haben. Speichern Sie Ihren Code – **destroy6.py** – und testen Sie das Spiel, um den Unterschied zu sehen. Sie können die Zahlen ändern oder weitere elif-Bedingungen hinzufügen, wenn Sie den Schwierigkeitsgrad noch weiter erhöhen wollen.

Spiel-Ende

Jetzt müssen Sie nur noch herausfinden, wann der Spieler das Spiel verloren hat. Das ist der Fall, wenn sich jedes Feld in einen roten Kreis verwandelt hat.

Erinnern Sie sich, dass Sie bei Tic-Tac-Toe mit verschachtelten Schleifen geprüft haben, ob alle

Felder gefüllt waren und das Spiel unentschieden war? Die gleiche Methode können Sie auch hier verwenden, um in einer Schleife jedes Quadrat im Raster zu überprüfen, ob es einen roten Kreis enthält. Fügen Sie in Ihrer Funktion `add_dot` kurz vor dem Aufruf von `after` den Code für eine verschachtelte Schleife ein, um alle Quadrate auf dem Spielfeld zu durchlaufen:

```
all_red = True
for x in range(GRID_SIZE):
    for y in range(GRID_SIZE):
```

Die erste Zeile beginnt mit der Annahme, dass alle Quadrate rot sind. Die verschachtelte Schleife liefert der Reihe nach die Koordinaten jedes Quadrats auf dem Gitter als Werte `x` und `y`, so dass Sie überprüfen können, ob das stimmt.

Fügen Sie innerhalb der zweiten Schleife den Code ein, um herauszufinden, ob das aktuelle Pixel rot ist, und wenn nicht, ändern Sie die Variable `all_red` auf False.

```
all_red = True
for x in range(GRID_SIZE):
    for y in range(GRID_SIZE):
        if board[x,y].color != "red":
            all_red = False
```

Nachdem beide Schleifen beendet sind (achten Sie darauf, dass Sie den folgenden Code nicht einrücken), prüfen Sie, ob alle Gitter-Elemente einen roten Kreis enthalten. Wenn ja, hat der Spieler verloren, also geben Sie eine Meldung aus:

```
if all_red:
    score_display.value = "You lost! Score:
" + str(score)
```

Wenn der Spieler nicht verloren hat, sollte das Spiel fortgesetzt werden. Fügen Sie ein `else:` hinzu und rücken Sie darin die bereits vorhandene Methode `after` ein, da wir nur dann einen neuen Punkt hinzufügen wollen, wenn der Spieler nicht verloren hat:

```
else:
    board.after(speed, add_dot)
```

Achten Sie darauf, die *after*-Zeile, die Sie hier schon haben, einzurücken und nicht noch eine weitere hinzuzufügen, sonst fängt Ihr Spiel an, sich seltsam zu verhalten und mehrere Punkte pro Sekunde zu erzeugen!

Ihr endgültiger Code sollte dem in der Datei **07-destroy-the-dots.py** ähneln. Viel Spaß mit dem Spiel! .

> Nachdem beide Schleifen beendet sind, prüfen Sie, ob alle Quadrate durch rote Kreise ersetzt wurden. Wenn ja, hat der Spieler verloren

07-destroy-the-dots.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Text, Waffle
004.  from random import randint
005.
006.  # Variables -------------
007.
008.  GRID_SIZE = 5
009.  score = 0
010.
011.  # Functions -------------
012.
013.  def add_dot():
014.      x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
015.      while board[x, y].dotty == True:
016.          x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
017.      board[x, y].dotty = True
018.      board.set_pixel(x, y, "red")
019.
020.      speed = 1000
021.      if score > 30:
022.          speed = 200
023.      elif score > 20:
024.          speed = 400
025.      elif score > 10:
026.          speed = 500
027.
028.      all_red = True
029.      for x in range(GRID_SIZE):
030.          for y in range(GRID_SIZE):
031.              if board[x,y].color != "red":
032.                  all_red = False
033.      if all_red:
034.          score_display.value = "You lost! Score: " + str(score)
035.      else:
036.          board.after(speed, add_dot)
037.
038.  def destroy_dot(x,y):
039.      global score
040.      if board[x,y].dotty == True:
041.          board[x,y].dotty = False
042.          board.set_pixel(x, y, "white")
043.          score += 1
044.          score_display.value = "Your score is " + str(score)
045.
046.  # App -------------------
047.
048.  app = App("Destroy the dots")
049.
050.  instructions = Text(app, text="Click the dots to destroy them")
051.  board = Waffle(
      app, width=GRID_SIZE, height=GRID_SIZE, command=destroy_dot)
052.  board.after(1000, add_dot)
053.  score_display = Text(app, text="Your score is " + str(score))
054.
055.  app.display()
```

Teil 07

Flood It: GUIs erstellen mit Python

Lernen Sie, wie man mit einem Raster aus Quadraten ein spannendes Spiel erstellt.

▶ **Bild 1** Füllen Sie die Quadrate mit einer Farbe.

***Flood It* ist ein etwas komplexeres Spiel auf Basis sogenannter Waffles, bei dem es darum geht, alle Felder des Spielbretts, im Folgenden Matrix genannt, mit gleicher Farbe zu füllen.** Beginnend mit dem linken oberen Feld, wählt der Spieler eine Farbe, die er ändern möchte.

Ziel des Spiels

Das obere linke Feld in **Bild 1** ist zum Beispiel blau. Der Spieler kann entweder das einzelne lila Feld darunter oder das gelbe Feld rechts davon umfärben. Das gelbe Feld zu ändern wäre der bessere Zug, da laut Spielregel dann alle angrenzenden gelben Felder ebenfalls umgefärbt würden. Auf diese Weise geht es weiter, bis alle Felder gefüllt sind. Der Witz an der Sache ist, dass dem Spieler jedoch nur eine begrenzte Anzahl von Zügen erlaubt ist und er daher wohl überlegt vorgehen muss. Am Ende des Abschnitts ***Start der Füllung*** wird die Bedienung des Spiels anhand von illustrierten Beispielen (**Bilder 8 bis 10**) noch eingehender erklärt.

Das Spiel einrichten

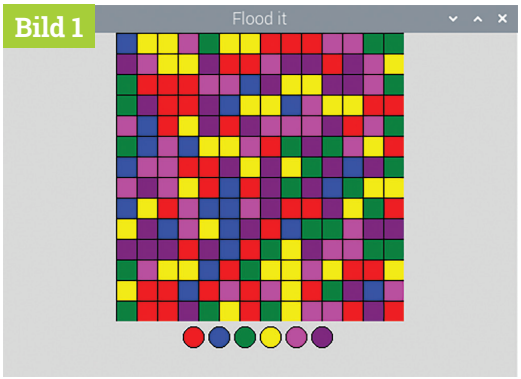
Laden Sie (von **magpi.cc/floodit**) die Startdatei floodit_starter.py herunter und öffnen Sie sie. Speichern Sie sie an einem geeigneten Ort.

Weisen Sie den Variablen im Abschnitt variables entsprechende Werte zu:

- `colours` – eine Liste von sechs Farben als Strings. Diese können entweder allgemeine Farbnamen oder Hex-Farben sein. Die Farbnamen "weiß", "schwarz", "rot", "grün", "blau", "cyan", "gelb" und "magenta" sind immer verfügbar.
- `board_size` – die Breite bzw. Höhe der Matrix als Ganzzahl; wir haben 14 gewählt. Das Brett ist immer ein Quadrat.
- `moves_limit` – die Anzahl der Züge, die der Spieler machen darf, bevor er verliert, als ganze Zahl; wir haben 25 gewählt.

Erstellen Sie im Abschnitt für Apps ein App-Widget und geben Sie ihm einen Titel.

```
app = App("Flood it")
app.display()
```



Wenn Sie diese Zeilen ausführen, erhalten Sie ein standardmäßig beschriftetes Fenster (**Bild 2**).

Erstellen Sie das Spielbret

Das Spielbrett ist ein Gitter aus Quadraten, denen jeweils eine zufällig ausgewählte Farbe aus der zuvor erstellten Liste zugewiesen wird. Fügen Sie innerhalb der Anwendung ein Waffle-Widget hinzu.

Damit wird ein Raster für die Matrix, erzeugt.

```
board = Waffle(app)
```

Führen Sie Ihr Programm aus und Sie werden sehen, dass die Quadrate im Gitter etwas zu klein sind (**Bild 3**). Fügen Sie der Codezeile, die Sie gerade geschrieben haben, noch die Parameter für die Breite und Höhe des Waffle-Widgets hinzu und machen Sie den Abstand zwischen den Gitterquadraten damit zu Null.

```
board = Waffle(app, width=board_size,
height=board_size, pad=0)
```

Das sieht schon besser aus (**Bild 4**).

Erstellen Sie die Palette

Die Palette zeigt dem Spieler, welche Farben er anklicken kann, um das Spielbrett mit Farben zu füllen. Die Palette des fertigen Spiels ist in Bild 5 unten zu sehen. Erstellen Sie in der Zeile nach der Erzeugung des Spielbretts (Matrix) ein weiteres Waffle-Widget mit dem Namen `palette`.

Bild 2 Flood it

▲ **Bild 2** Die übliche Titelleiste.

```
palette = Waffle(app)
```

Im vorigen Schritt haben Sie dem Waffle-Widget für die Matrix die notwendigen Parameter hinzugefügt. Dies wird nun auch für das Paletten Waffle-Widget durchgeführt, wobei die einzelnen Parameter durch ein Komma getrennt sind:

`width = 6`
(die Anzahl der Farben, die wir haben)
`height = 1`
`dotty = True`
(dadurch werden die Quadrate zu Kreisen)

Jetzt sollte das Ganze wie folgt aussehen:

```
palette = Waffle(app, width=6, height=1,
dotty=True)
```

Nach dem Ausführen des Codes erscheint eine leere Palette (**Bild 6**).

Einfärben der Matrix

Die Matrix sollte zu Beginn für jedes Quadrat eine zufällig ausgewählte Farbe aus der zuvor erstellten Farbliste enthalten. Schreiben Sie in die Zeile unter Ihrer Palette einen Aufruf der Funktion

```
fill_board()
```

Suchen Sie den Abschnitt für die Funktionen in Ihrem Programm und beginnen Sie mit dem Schreiben des Codes für diese neue Funktion

```
def fill_board():
```

Sie können eine verschachtelte Schleife schreiben, um jede Zeile und Spalte der Matrix zu durchlaufen. Jedes Pixel innerhalb eines Quadrats wird mit derselben, zufällig ausgewählten Farbe aus der Liste eingefärbt. Zum Einfärben verwenden Sie folgenden Code, bei dem die ?-Symbole durch die x- und y-Koordinaten des Pixels ersetzt werden:

```
board.set_pixel(?, ?, random.
choice(colours))
```

Versuchen Sie, den Code selbst zu schreiben, indem Sie das anwenden, was Sie in den vorangegangenen Teilen dieser Serie über verschachtelte Schleifen gelernt haben. Die Lösung finden Sie weiter unten, falls Sie nicht weiterkommen.

> Das Spielfeld sollte zu Beginn jedes einzelne Kästchen in einer zufällig gewählten Farbe darstellen

Tipp: *Verwenden Sie die Variable board_size, um zu wissen, wie oft die Schleife ausgeführt werden soll.*

```
def fill_boa
```

Wenn Sie den Code ausführen, sollten Sie eine Matrix aus bunten Quadraten sehen. Erscheint jedoch eine weiße Matrix, so überprüfen Sie, ob Sie den Funktionsaufruf `fill_board()` auch eingegeben haben (**Bild7**).

Hier ist eine Lösung, aber es gibt viele Möglichkeiten, wie Sie dies tun können:

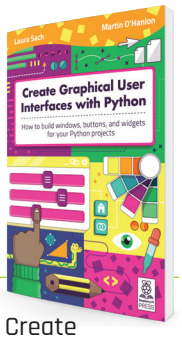
```
def fill_board():
    for x in range(board_size):
        for y in range(board_size):
            board.set_pixel(x, y, random.
choice(colours))
```

Eine alternative Lösung, die eine optimalere Funktion namens "Listcomprehension“ verwendet:

```
   board.set_pixel(x, y, random.
choice(colours))
```

Einsetzen der Farben in die Palette

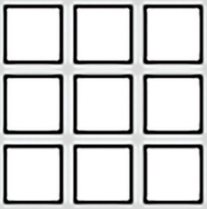
Nachdem Sie eine farbige Matrix erstellt haben, können Sie die Farben in der Palette einsetzen.



Create Graphical User Interfaces with Python

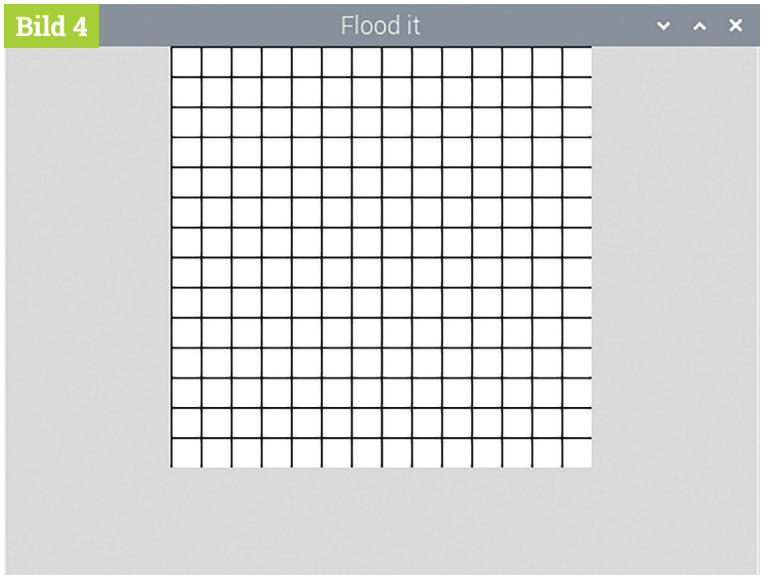
Weitere Anleitungen, wie Sie mit guizero eigene grafische Benutzeroberflächen erstellen können, finden Sie in unserem Buch *Create Graphical User Interfaces with Python*. Die 156 Seiten sind vollgepackt mit wichtigen Informationen und einer Reihe von spannenden Projekten. **magpi.cc/pythongui**

Bild 3 Flood it



▲ **Bild 3** TDie Quadrate des Gitters sind zu klein.

Bild 4



▲ **Bild 4** Ein Raster in der richtigen Größe, ohne Einfärbung.

▶ **Bild 6** Vorerst noch farblose Palette.

Schreiben Sie in die Zeile unter dem Code `fill_board()` den Aufruf einer Funktion:

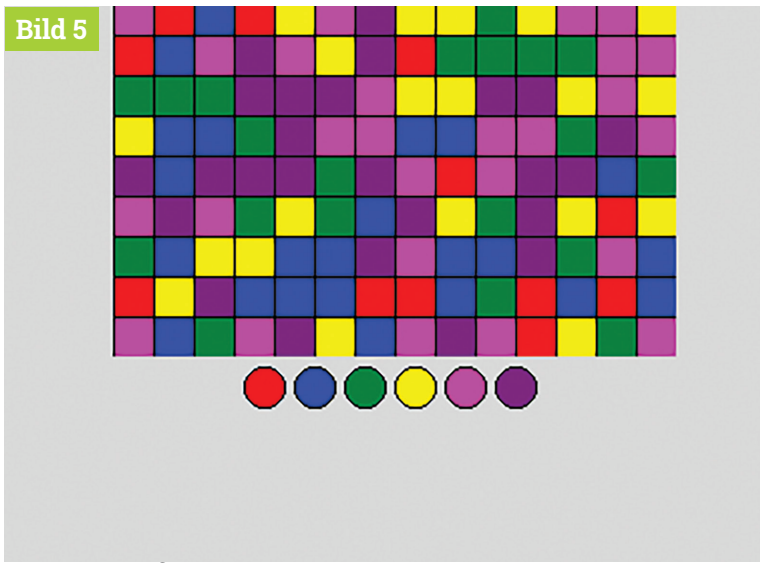
```
init_palette()
```

Suchen Sie den Abschnitt *functions* in Ihrem Programm und beginnen Sie, den Code für diese neue Funktion zu schreiben:

```
def init_palette():
```

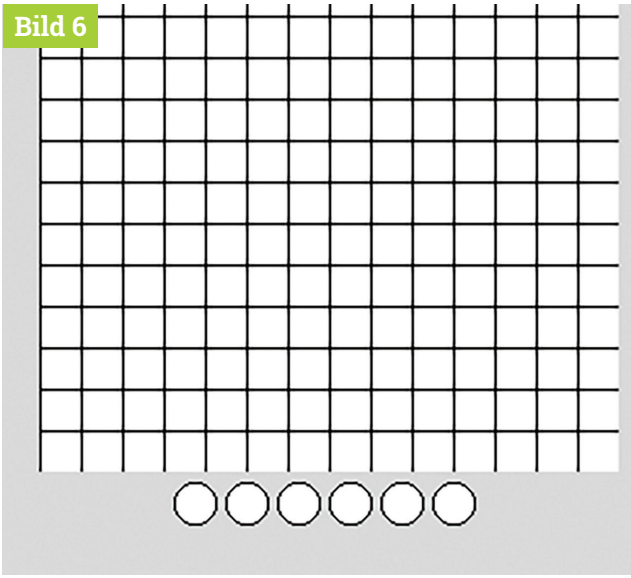
Hier werden alle Farben in der Liste in eine Schleife einbezogen. Jedem Kreis in der Palette wird eine Farbe zugewiesen. Um die Farbe der Kreise in der Palette zu ändern, können Sie dieselbe `set_pixel`-Methode verwenden, die Sie für das Spielfeld verwendet haben. Versuchen Sie, den Code selbst zu schreiben. Wenn Sie nicht weiter kommen, finden Sie im Kasten "Auf wie viele Arten kann man die Palette einfärben " eine Reihe von Lösungen.

Bild 5



▼ **Bild 5** Sie benötigen eine Palette (unten im Bild), auf welcher der Spieler eine Farbe auswählen kann.

Bild 6



Hinweis: *Alle Kreise in der Palette befinden sich in Zeile 0 des Waffle-Widgets..*

Start der Füllung

Wenn der Spieler auf eine Farbe in der Palette klickt, wird das Spielfeld mit dieser neuen Farbe eingefärbt, beginnend mit dem linken oberen Feld. Erstellen Sie im Funktionsbereich eine neue Funktion mit dem Namen `start_flood`, und zwar auf die gleiche Weise wie bei den letzten beiden Funktionen. Diese Funktion braucht zwei Parameter, die den x- und y-Koordinaten des angeklickten Quadrats entsprechen. Fügen Sie diese zwischen den eckigen Klammern ein, so dass Ihr Code am Ende wie folgt aussieht:

```
def start_flood(x, y):
```

Fügen Sie eine Codezeile (eingerückt) in die Funktion ein, um den Namen der angeklickten Farbe zu erhalten:

```
flood_colour = palette.get_pixel(x,y)
```

Dies ist die Farbe, mit der die betreffenden Quadrate auf dem Spielfeld eingefärbt werden.

Fügen Sie nun eine Codezeile hinzu, um die aktuelle Farbe des Startpixels zu ermitteln - dies ist immer das Pixel oben links auf der Matrix, an den Koordinaten 0, 0.

```
target = board.get_pixel(0,0)
```

Rufen Sie nun die Funktion `flood` auf, die bereits in der Starter-Datei für Sie geschrieben wurde. Diese

Funktion beginnt bei 0,0 und färbt alle Pixel, die mit dem linken oberen Pixel verbunden sind und die gleiche Farbe wie die `flood_colour` haben.

```
flood(0, 0, target, flood_colour)
```

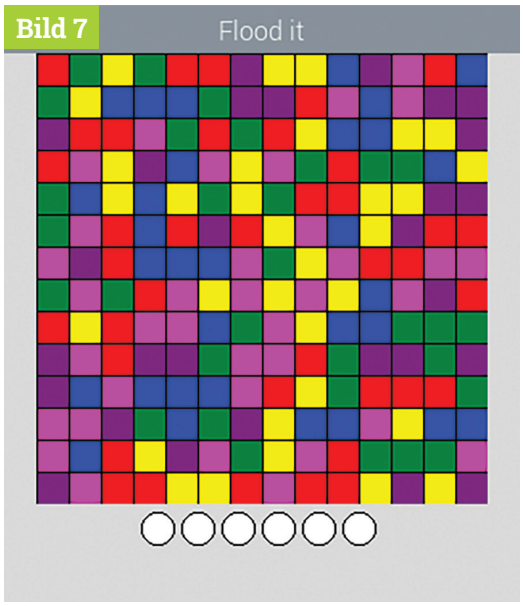
Diese Funktion sollte immer dann ausgeführt werden, wenn jemand auf eine Farbe in der Palette klickt. Suchen Sie also die Codezeile, in der Sie die Palette erstellt haben.

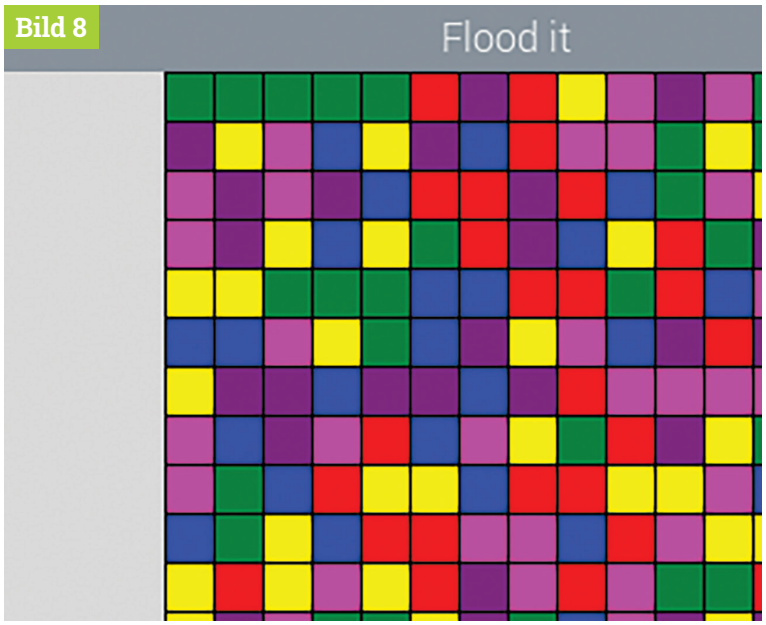
```
palette = Waffle(app, width=6, height=1,
dotty=True)
```

Fügen Sie einen weiteren Parameter hinzu, der ein Befehl ist. Wenn ein Kreis auf der Palette angeklickt wird, wird dieser Befehl ausgeführt: Die Funktion `start_flood`. Ihr Code sollte nun wie folgt aussehen:

```
palette = Waffle(app, width=6, height=1,
dotty=True, command=start_flood)
```

Testen Sie Ihren Code, indem Sie auf die Kreise in der Palette klicken. In unserem zufälligen Beispiel (**Bild 8**) ist das obere linke Quadrat grün. Wenn Sie auf der Palette auf lila klicken, wird das linke obere Quadrat lila und verbindet sich mit dem lila Quadrat darunter (**Bild 9**). Jetzt gibt es fünf lila Quadrate, die mit dem linken oberen Quadrat verbunden sind. Klicken wir nun auf rosa, um die darunterliegenden rosa Quadrate zu verbinden (**Bild 10**). Jetzt gibt es eine große Kette von rosa Quadraten. Setzen Sie das Spiel fort, indem Sie verschiedene Farben in der Palette auswählen. Ziel ist es, dass am Ende alle Quadrate die gleiche Farbe haben.





> Fügen Sie der GUI einen Text hinzu, der anzeigt, ob der Spieler gewonnen oder verloren hat

Das Spiel gewinnen

Wenn der Spieler es schafft, dass alle Quadrate im Gitter die gleiche Farbe haben, ist das Spiel zu Ende. Fügen wir zunächst einen Text in die GUI ein, der anzeigt, ob der Spieler gewonnen oder verloren hat. Das Textfeld wird zu Beginn leer sein. Fügen Sie unter dem Code für die Palette ein Text-Widget namens `win_text` hinzu.

```
win_text = Text(app)
```

Setzen Sie im Abschnitt der Variablen eine weitere Variable namens `moves_taken` ein und geben Sie ihr den Wert 0. Erstellen Sie nun eine Funktion namens `win_check`, die nach jedem Zug überprüft, ob der Spieler gewonnen hat. Zuerst müssen Sie angeben, dass Sie den Wert der globalen Variable `moves_taken` ändern dürfen.

```
global moves_taken
```

Dann addieren Sie 1 zur Variable `moves_taken`. Jedes Mal, wenn diese Funktion aufgerufen wird, wird die Anzahl der Züge inkrementiert:

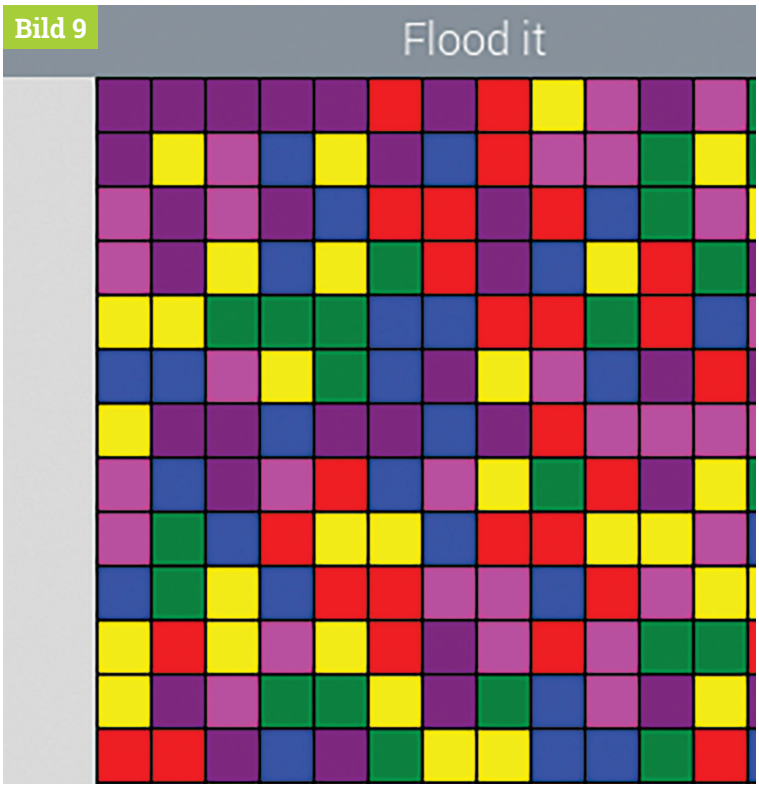
```
moves_taken += 1
```

Überprüfe, ob `moves_taken` kleiner als `moves_limit` ist oder nicht:

Bild 8 Hier ist das obere linke Feld grün.

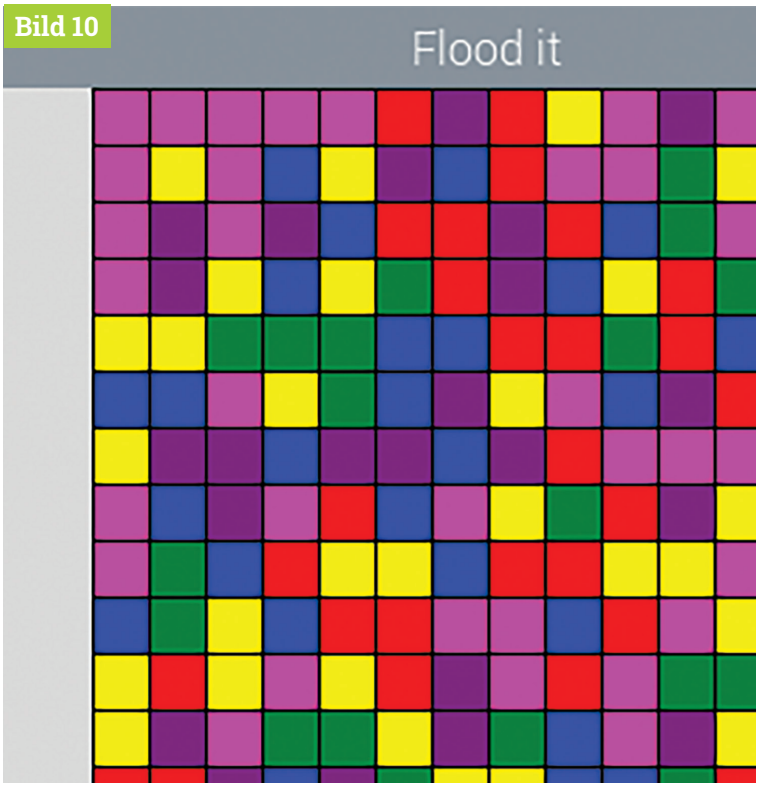
Bild 7 Jedes Feld des Spielbretts wird nach dem Zufallsprinzip eingefärbt.

Bild 9



▲ **Bild 9** Ein Klick auf violett verleiht dem Feld die entsprechende Farbe.

Bild 10



▼ **Bild 10** Klicken Sie auf rosa, um eine Verkettung in rosa zu erhalten.

```
    if moves_taken < moves_limit:

    else:
```

Wenn `moves_taken` nicht innerhalb des Limits liegt, bedeutet dies, dass der Spieler keine Züge mehr hat, also aktualisiere den Text, um zu sagen, dass er verloren hat:

```
    if moves_taken < moves_limit:

    else:
        win_text.value = "Du hast verloren :("
```

Wenn die Anzahl der ausgeführten Züge unter dem Limit liegt, prüfen Sie, ob alle Felder die gleiche Farbe haben, indem Sie die Funktion aufrufen, die bereits in der Startdatei geschrieben wurde. Achten Sie darauf, dass der folgende Code unterhalb der ersten **if**-Anweisung eingerückt wird:

```
    if all_squares_are_the_same():
        win_text.value = "Du hast gewonnen!"
```

Der fertige Code sollte wie folgt aussehen::

```
    def win_check():
        moves_taken += 1
        if moves_taken <= moves_limit:
            if all_squares_are_the_same():
                win_text.value = "Du hast
gewonnen!"
        else:
            win_text.value = "Du hast verloren
:("
```

Schließlich müssen Sie die Funktion `win_check` aufrufen, wenn ein Feld angeklickt wird. Am einfachsten ist es, wenn Sie den Funktionsaufruf am Ende der Funktion `start_flood` einfügen.

Nun ist es an der Zeit, das Spiel zu testen. Ein Beispiel für den Code ist in **08-floodit.py** zu sehen.

Funktionstest

Sie können testen, ob das Spiel funktioniert, indem Sie es ganz einfach spielen; allerdings kann es sehr lange dauern, bis man gewinnen kann! Eine einfachere Möglichkeit ist, die Variable `board_size` auf einen kleinen Wert wie zum Beispiel 5 zu ändern, um das Spiel dann auf einem viel kleineren Feld zu spielen. Um zu testen, ob man das Spiel auch verlieren kann, klicken Sie ganz einfach 25 Mal auf dieselbe Farbe!

Auf wie viele Arten kann man die Palette einfärben?

Hier ist eine Lösung, die eine Schleife und eine Variable verwendet, um zu verfolgen, welche Spalte Sie einfärben:

```
def init_palette():
   column = 0
    for colour in colours:
        palette.set_pixel(column, 0, colour)
        column += 1
```

Hier ist eine ähnliche Lösung, die **range** innerhalb der for-Schleife anstelle einer Zählervariablen verwendet:

```
def init_palette():
   for x in range(len(colours)):
      palette.set_pixel(x, 0, colours[x])
```

Hier ist noch eine andere Lösung, die die Indexfunktion **colours.index(colour)** verwendet. Dieser Code besagt: "Finde in der Liste der Farben die Position in der Liste der Farbe". Wenn Ihre Liste also zum Beispiel [**"green"**, **"blue"**, **"red"**] wäre, dann wäre der Index von grün 0, der Index von blau wäre 1 usw., wobei wir bei Null anfangen zu zählen.

```
def init_palette():
   for colour in colours:
    palette.set_pixel(colours.index(colour), 0, colour)
```

Sie können jede dieser Lösungen verwenden, oder Sie haben vielleicht selbst einen anderen Weg gefunden. Keine dieser Lösungen ist die "richtige Antwort": Oft gibt es viele verschiedene Möglichkeiten, eine Lösung zu kodieren.

08-floodit.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/floodit

Sprache: **Python 3**

```
# --------------------------------
# Imports
# --------------------------------

from guizero import App, Waffle, Text, PushButton, info
import random

# --------------------------------
# Variables
# --------------------------------

colours = ["red", "blue", "green", "yellow", "magenta",
"purple"]
board_size = 14
moves_limit = 25
moves_taken = 0

# --------------------------------
# Functions
# --------------------------------

# Recursively floods adjacent squares
def flood(x, y, target, replacement):
    # Algorithm from https://en.wikipedia.org/wiki/
Flood_fill
    if target == replacement:
        return False
    if board.get_pixel(x, y) != target:
        return False
    board.set_pixel(x, y, replacement)
    if y+1 <= board_size-1:   # South
        flood(x, y+1, target, replacement)
    if y-1 >= 0:              # North
        flood(x, y-1, target, replacement)
    if x+1 <= board_size-1:    # East
        flood(x+1, y, target, replacement)
    if x-1 >= 0:              # West
        flood(x-1, y, target, replacement)

# Check whether all squares are the same
def all_squares_are_the_same():
    squares = board.get_all()
    if all(colour == squares[0] for colour in squares):
        return True
else:
        return False

def win_check():
    global moves_taken
    moves_taken += 1
    if moves_taken <= moves_limit:
        if all_squares_are_the_same():
            win_text.value = "You win!"
    else:
        win_text.value = "You lost :("

def fill_board():
    for x in range(board_size):
        for y in range(board_size):
            board.set_pixel(x, y, random.choice(colours))

def init_palette():
    for colour in colours:
        palette.set_pixel(colours.index(colour), 0,
colour)

def start_flood(x, y):
    flood_colour = palette.get_pixel(x,y)
    target = board.get_pixel(0,0)
    flood(0, 0, target, flood_colour)
    win_check()

# --------------------------------
# App
# --------------------------------

app = App("Flood it")

board = Waffle(app, width=board_size,
height=board_size, pad=0)
palette = Waffle(app, width=6, height=1, dotty=True,
command=start_flood)

win_text = Text(app)

fill_board()
init_palette()

app.display()
```

Teil 08

GUIs mit Python: Emoji Match

Programmieren Sie ein lustiges Spiel zum Auffinden von Bildern.

In diesem Beitrag zeigen wir Ihnen, wie Sie ein Spiel programmieren, in dem es um das Auffinden von Emojis geht (Bild 1). Ziel des Spiels ist es, das eine Emoji zu finden, das in zwei verschiedenen Bildgruppen gleichzeitig vorkommt. Gar nicht so einfach, wie es auf den ersten Blick scheint. Für jede richtige Zuordnung gibt einen Punkt, für eine falsche Zuordnung wird ein Punkt abgezogen.

Emojis laden

Um das Spiel zu erstellen, brauchen wir einen Vorrat an Emojis. Sie können zum Beispiel die für Twitter erstellten Emojis verwenden (**twemoji.twitter.com**). Laden Sie die Datei **emojis.zip** von **magpi.cc/guizeroemojis** herunter, öffnen Sie die zip-Datei und kopieren Sie den Ordner **emojis** in den Ordner, in dem Sie Ihren Code speichern. Das Spiel muss neun Emojis nach dem Zufallsprinzip auswählen und sie in einem Gitter anordnen. Eine einfache Möglichkeit, dies zu tun, ist, alle Emojis in eine Liste zu schreiben und sie zufällig zu mischen.

Erstellen Sie ein neues Programm mit den üblichen kommentierten Zeilen für die verschiedenen Abschnitte (Importe, Variablen, Funktionen, App), so wie wir es in den vorangegangenen Teilen getan haben. Fügen Sie unter Importe nun ein:

```
import os
from random import shuffle
```

Geben Sie dann unter *variables* den folgenden Code ein, der eine gemischte Liste von Emojis in der Form **path/emoji_file_name** erstellt.

```
# set the path to the emoji folder on your
computer
emojis_dir = "emojis"
emojis = [os.path.join(emojis_dir, f) for f
in os.listdir(emojis_dir)]
shuffle(emojis)
```

Die Variable **emojis_dir** ist der Pfad zu den Emojis auf Ihrem Computer. Sie teilt dem Code, der die Emojis lädt, mit, wo sie zu finden sind. Testen Sie Ihr Programm. Versuchen Sie, die `emojis`-Liste mit `print(emojis)` auf den Bildschirm zu bringen. Sie sollten eine lange Liste von Dateinamen sehen. Die Liste sollte bei jedem Durchlauf in einer anderen Reihenfolge erscheinen.

▲ **Bild 1** Das fertige Spiel.

Anzeige der Emojis

Als Nächstes muss der Code zwei 3×3-Gitter aus Picture- und PushButton-Widgets erstellen, in denen die Emojis angezeigt werden.

Ändern Sie Ihr Programm, um eine Guizero-App und eine Box zu erstellen, die die Bild-Widgets in einem ***grid***-Layout enthält. Fügen Sie im Abschnitt *imports* diese Zeile hinzu, um die erforderlichen Widgets zu importieren:

```
from guizero import App, Box
```

Fügen Sie im Abschnitt *app* den folgenden Code ein:

```
app = App("emoji match")

pictures_box = Box(app, layout="grid")
```

Das Box-Widget ist sehr nützlich für die Gestaltung der GUI. Es ist ein unsichtbarer Bereich Ihrer GUI, in dem Sie Widgets gruppieren können. Eine Box kann ihr eigenes Layout, ihre eigene Größe und ihren eigenen Hintergrund (bg) haben. Sie können auch ein- oder ausgeblendet werden, was bedeutet, dass Sie eine Sammlung von Widgets einfach unsichtbar machen können. Wenn Sie die Box sehen möchten, können Sie einen Rahmen hinzufügen, indem Sie den Parameter auf *True* setzen.

```
  pictures_box = Box(app, layout="grid",
border=True)
```

Fügen Sie nun das Picture-Widget zu Ihren Importen hinzu:

```
  from guizero import App, Box, Picture
```

Fügen Sie im app-Abschnitt den Code ein, um die Picture-Widgets zu erstellen und sie zu einer Liste hinzuzufügen.

```
  pictures = []

  for x in range(0,3):
      for y in range(0,3):
          picture = Picture(pictures_box,
grid=[x,y])
          pictures.append(picture)
```

Um den einzelnen Picture-Widgets Koordinaten zuzuweisen, werden zwei `for`-Schleifen verwendet. Beide durchlaufen den Bereich 0-2. Die eine weist ihren Wert der Variablen x zu, die andere der Variablen y. Die Rasterposition jedes Widgets wird anhand der x- und y-Werte festgelegt. Die Widgets werden an eine Liste angehängt, damit sie später im Spiel referenziert werden können. Führen Sie dasselbe für PushButton-Widgets durch, um das zweite 3×3-Raster zu erstellen. Fügen Sie zunächst das Widget zu Ihren Importen hinzu::

```
  from guizero import App, Box, Picture,
PushButton
```

Fügen Sie im Abschnitt app Zeilen hinzu, so dass es wie folgt aussieht:

```
  app = App("emoji match")

  pictures_box = Box(app, layout="grid")
  buttons_box = Box(app, layout="grid")

  pictures = []
  buttons = []

  for x in range(0,3):
      for y in range(0,3):
          picture = Picture(pictures_box,
grid=[x,y])
          pictures.append(picture)

          button = PushButton(buttons_box,
grid=[x,y])
          buttons.append(button)
```

emoji1.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

Sprache: **Python 3**

```
001.  # ------------------------------
002.  # Imports
003.  # ------------------------------
004.
005.  import os
006.  from random import shuffle
007.  from guizero import App, Box, Picture, PushButton
008.
009.  # ------------------------------
010.  # Variables
011.  # ------------------------------
012.
013.  # set the path to the emoji folder on your computer
014.  emojis_dir = "emojis"
015.  emojis = [os.path.join(emojis_dir, f) for f in os.listdir(
      emojis_dir)]
016.  shuffle(emojis)
017.
018.  # ------------------------------
019.  # Functions
020.  # ------------------------------
021.
022.  def setup_round():
023.      for picture in pictures:
024.          picture.image = emojis.pop()
025.
026.      for button in buttons:
027.          button.image = emojis.pop()
028.
029.  # ------------------------------
030.  # App
031.  # ------------------------------
032.
033.  app = App("emoji match")
034.
035.  pictures_box = Box(app, layout="grid")
036.  buttons_box = Box(app, layout="grid")
037.
038.  pictures = []
039.  buttons = []
040.
041.  for x in range(0,3):
042.      for y in range(0,3):
043.          picture = Picture(pictures_box, grid=[x,y])
044.          pictures.append(picture)
045.
046.          button = PushButton(buttons_box, grid=[x,y])
047.          buttons.append(button)
048.
049.  setup_round()
050.
051.  app.display()
```

Bild 2

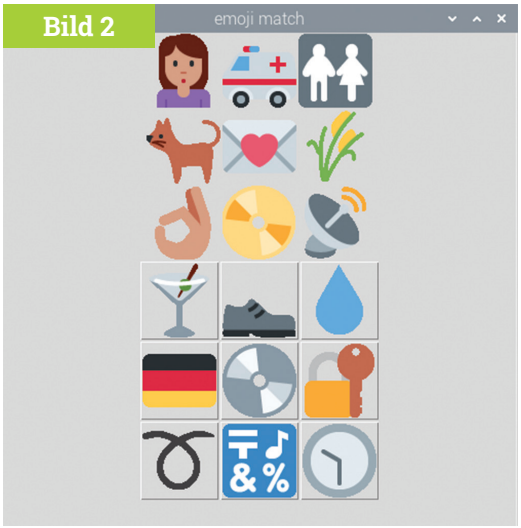


Bild 2 Keine passenden Emojis.

Erstellen Sie im Abschnitt *functions* eine Funktion, die jede Runde des Spiels neu einrichtet.

```
def setup_round():
    for picture in pictures:
        picture.image = emojis.pop()

    for button in buttons:
        button.image = emojis.pop()
```

Um jedem `picture`- und `button`-Widget ein Emoji zuzuweisen, wird die Eigenschaft `image` auf ein Element aus der Liste der `emojis` gesetzt. Die Emojis werden mit der Funktion `pop()` ausgewählt, die das letzte Element in einer Liste entfernt. Wir haben diese Funktion verwendet, weil sie verhindert, dass ein Emoji mehr als einmal im Spiel auftaucht. Rufen Sie am Ende Ihres Programms die Funktion `setup_round` auf und zeigen Sie die App an.

```
setup_round()

app.display()
```

Ihr Programm sollte nun dem Programm mit dem Namen **emoji1.py** ähneln (siehe vorherige Seite). Beim Test sollten Sie zwei Raster mit neun Emojis sehen.

Passende Emojis

Vorerst sind alle Emojis in Ihrer Anwendung noch unterschiedlich (**Bild 2**). Im nächsten Schritt wählen Sie ein anderes Emoji aus, das Sie anpassen möchten, und aktualisieren ein Bild und eine Schaltfläche, damit sie dasselbe Emoji haben.

Fügen Sie `randint` zu Ihrer `random` Import-Zeile hinzu. Damit erhalten Sie eine Zahl von 0 bis 8 für jedes Bild und jeden Button.

```
from random import shuffle, randint
```

emoji2.py

Sprache: **Python 3**

```
001.  # ------------------------------
002.  # Imports
003.  # ------------------------------
004.
005.  import os
006.  from random import shuffle, randint
007.  from guizero import App, Box, Picture, PushButton
008.
009.  # ------------------------------
010.  # Variables
011.  # ------------------------------
012.
013.  # set the path to the emoji folder on your computer
014.  emojis_dir = "emojis"
015.  emojis = [os.path.join(emojis_dir, f) for f in os.listdir(emojis_dir)]
016.  shuffle(emojis)
017.
018.  # ------------------------------
019.  # Functions
020.  # ------------------------------
021.
022.  def setup_round():
023.      for picture in pictures:
024.          picture.image = emojis.pop()
025.
026.      for button in buttons:
027.          button.image = emojis.pop()
028.
029.      matched_emoji = emojis.pop()
030.
031.      random_picture = randint(0,8)
032.      pictures[random_picture].image = matched_emoji
033.
034.      random_button = randint(0,8)
035.      buttons[random_button].image = matched_emoji
036.
037.  # ------------------------------
038.  # App
039.  # ------------------------------
040.
041.  app = App("emoji match")
042.
043.  pictures_box = Box(app, layout="grid")
044.  buttons_box = Box(app, layout="grid")
045.
046.  pictures = []
047.  buttons = []
048.
049.  for x in range(0,3):
050.      for y in range(0,3):
051.          picture = Picture(pictures_box, grid=[x,y])
052.          pictures.append(picture)
053.
054.          button = PushButton(buttons_box, grid=[x,y])
055.          buttons.append(button)
056.
057.  setup_round()
058.
059.  app.display()
```

emoji3.py

Sprache: **Python 3**

```
# ------------------------------
# Imports
# ------------------------------

import os
from random import shuffle, randint
from guizero import App, Box, Picture, PushButton,
Text

# ------------------------------
# Variables
# ------------------------------

# set the path to the emoji folder on your computer
emojis_dir = "emojis"
emojis = [os.path.join(emojis_dir, f) for f in
os.listdir(emojis_dir)]
shuffle(emojis)

# ------------------------------
# Functions
# ------------------------------

def setup_round():
    for picture in pictures:
        picture.image = emojis.pop()

    for button in buttons:
        button.image = emojis.pop()
        button.update_command(
match_emoji, args=[False])

    matched_emoji = emojis.pop()

    random_picture = randint(0,8)
    pictures[random_picture].image = matched_emoji

    random_button = randint(0,8)
    buttons[random_button].image = matched_emoji

    buttons[random_button].update_command(
match_emoji, [True])

def match_emoji(matched):
    if matched:
        result.value = "correct"
    else:
        result.value = "incorrect"

    setup_round()

# ------------------------------
# App
# ------------------------------

app = App("emoji match")

pictures_box = Box(app, layout="grid")
buttons_box = Box(app, layout="grid")

pictures = []
buttons = []

for x in range(0,3):
    for y in range(0,3):
        picture = Picture(pictures_box, grid=[x,y])
        pictures.append(picture)

        button = PushButton(buttons_box, grid=[x,y])
        buttons.append(button)

result = Text(app)

setup_round()

app.display()
```

Fügen Sie dann diesen Code (eingerückt) am Ende der Funktion `setup_round` ein, um ein weiteres Emoji aus der Liste zu ziehen und es als *image* eines zufälligen Bildes und einer zufälligen Schaltfläche zu definieren.

```
  matched_emoji = emojis.pop()

  random_picture = randint(0,8)
  pictures[random_picture].image = matched_
emoji

  random_button = randint(0,8)
  buttons[random_button].image = matched_
emoji
```

Ihr Code sollte nun wie derjenige im Programm **emoji2.py** aussehen. Führen Sie Ihr Programm jetzt aus. Eines der Emojis sollte übereinstimmen. Schauen Sie genau hin – das passende Emoji kann schwer zu erkennen sein.

Überprüfe die Vermutung

Jedes Mal, wenn einer der PushButtons gedrückt wird, muss geprüft werden, ob es sich um das passende Emoji handelt und das Ergebnis „richtig" oder „falsch" auf dem Bildschirm angezeigt werden. Nachdem der Spieler geraten hat, wird eine neue Runde gestartet und ein anderer Satz von Emojis angezeigt. Ihre App benötigt ein Text-Widget, mit dem das Ergebnis anzeigt wird. Fügen Sie es zu Ihren Importen hinzu:

```
  from guizero import App, Box, Picture,
PushButton, Text
```

Fügen Sie diese Zeile in Ihre App-Sektion ein:

```
  result = Text(app)
```

Erstellen Sie eine neue Funktion, die aufgerufen wird, wenn eine der Emoji-Tasten gedrückt wird. Sie zeigt "richtig" oder "falsch" an und

emoji4.py

Sprache: **Python 3**

```
# ------------------------------
# Imports
# ------------------------------

import os
from random import shuffle, randint
from guizero import App, Box, Picture, PushButton, Text

# ------------------------------
# Variables
# ------------------------------

# set the path to the emoji folder on your computer
emojis_dir = "emojis"
emojis = [os.path.join(emojis_dir, f) for f in
os.listdir(emojis_dir)]
shuffle(emojis)

# ------------------------------
# Functions
# ------------------------------

def setup_round():
    for picture in pictures:
        picture.image = emojis.pop()

    for button in buttons:
        button.image = emojis.pop()
        button.update_command(match_emoji, args=[False])

    matched_emoji = emojis.pop()

    random_picture = randint(0,8)
    pictures[random_picture].image = matched_emoji

    random_button = randint(0,8)
    buttons[random_button].image = matched_emoji

    buttons[random_button].update_command(
match_emoji, [True])

def match_emoji(matched):
    if matched:
        result.value = "correct"
        score.value = int(score.value) + 1
    else:
        result.value = "incorrect"
        score.value = int(score.value) - 1

    setup_round()

def reduce_time():
    timer.value = int(timer.value) - 1
    # is it game over?
    if int(timer.value) < 0:
        result.value = "Game over! Score = " + score.value
        # hide the game
        pictures_box.hide()
        buttons_box.hide()
        timer.hide()
        score.hide()

# ------------------------------
# App
# ------------------------------

app = App("emoji match")

score = Text(app, text="0")
timer = Text(app, text="30")

pictures_box = Box(app, layout="grid")
buttons_box = Box(app, layout="grid")

pictures = []
buttons = []

for x in range(0,3):
    for y in range(0,3):
        picture = Picture(pictures_box, grid=[x,y])
        pictures.append(picture)

        button = PushButton(buttons_box, grid=[x,y])
        buttons.append(button)

result = Text(app)

setup_round()

app.repeat(1000, reduce_time)

app.display()
```

ruft `setup_round` auf, um die nächste Gruppe von Emojis zu erstellen.

```
def match_emoji(matched):
    if matched:
        result.value = "correct"
    else:
        result.value = "incorrect"

    setup_round()
```

Die falschen Emoji-Schaltflächen übergeben *False* an die `match_emoji`-Funktion; die passenden Emoji übergeben *True*. Aktualisieren Sie die Funktion `setup_round` so, dass alle falschen Schaltflächen die Funktion `match_emoji` aufrufen.

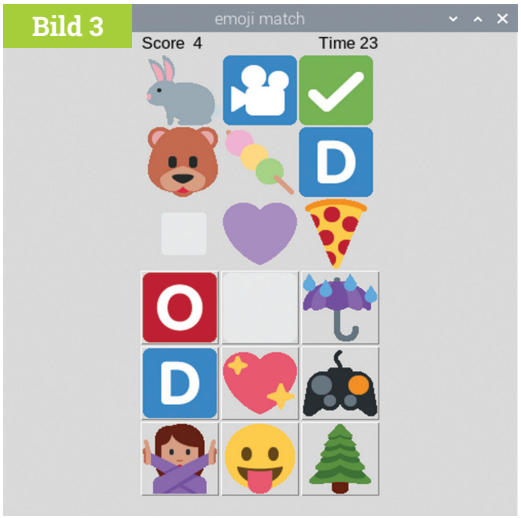


Bild 3 Mit Kasten für Spielstand und Timer.

```
        for button in buttons:
            button.image = emojis.pop()
            button.update_command(match_emoji,
args=[False])
```

Die `update_command`-Methode legt die Funktion fest, die aufgerufen wird, wenn die Schaltfläche gedrückt wird. Die `args`-Liste `[False]` wird als Parameter für die `match_emoji`-Funktion verwendet. Aktualisieren Sie schließlich den Befehl für die passende Schaltfläche, so dass `match_emoji` aufgerufen, aber diesmal ***True*** als Argument übergeben wird.

```
        buttons[random_button].update_
command(match_emoji, [True])
```

Ihr Code sollte nun demjenigen von **emoji3.py** ähneln. Spielen Sie das Spiel. In jeder Runde gibt es ein passendes Emoji – drücken Sie den Knopf für das passende Bild. Haben Sie alles richtig gemacht?

Hinzufügen einer Punktzahl und eines Timers

Im Moment dauert das Spiel ewig (oder bis keine Emojis mehr in der Liste sind). Fügen Sie eine Punktzahl und einen Timer hinzu, der bis zum Ende des Spiels herunter zählt, um dem Spiel einen gewissen Reiz zu verleihen und Sie herauszufordern. Erstellen Sie im App-Bereich zwei Text-Widgets, die den Punktestand und den Timer anzeigen.

```
score = Text(app, text="0")
timer = Text(app, text="30")
```

Der Timer ist auf 30 eingestellt, was der Anzahl der Sekunden in jeder Runde entspricht. Ändern Sie die Funktion `match_emoji` so, dass sie entweder 1 zur Punktzahl des Spielers hinzufügt oder von ihr subtrahiert.

Verwendung anderer Bilder

Das Emoji-Match-Spiel verwendet Bild-Schaltflächen, damit der Benutzer auswählen kann, welches Emoji passt. Sie können jedes PushButton-Widget in einen Bild-Button verwandeln, indem Sie den Bild-Parameter setzen; zum Beispiel

```
button = PushButton(app, image="my_picture.gif")
```

Die Schaltfläche wird an die Größe des Bildes angepasst. Die Art des Bildes, das Sie verwenden können, hängt von Ihrem Betriebssystem und der Art der Installation von guizero ab, obwohl jedes Setup GIF-Bilder unterstützt. Um die von Ihrem Setup unterstützten Bilddateitypen zu finden, können Sie Folgendes ausführen:

```
from guizero import system_config
print(system_config.supported_image_types)
```

Weitere Informationen zur Bildunterstützung in guizero finden Sie unter **lawsie.github.io/guizero/images**.

```
def match_emoji(matched):
    if matched:
        result.value = "correct"
        score.value = int(score.value) + 1
    else:
        result.value = "incorrect"
        score.value = int(score.value) - 1
```

Um den Timer zu erstellen, werden Sie eine Funktion von guizero verwenden, die die Anwendung auffordert, kontinuierlich alle 1 Sekunde eine Funktion aufzurufen. Erstellen Sie eine Funktion, die den Wert des Timers um 1 verringert.

```
def reduce_time():
    timer.value = int(timer.value) - 1
```

Bevor die App angezeigt wird, verwenden Sie die Funktion `app.repeat()`, um die Funktion `reduce_time` jede Sekunde (1000 Millisekunden) aufzurufen

```
app.repeat(1000, reduce_time)

app.display()
```

Wenn Sie Ihr Spiel jetzt starten, werden Sie feststellen, dass der Timer mit 30 beginnend im Sekundentakt herunter zählt. Leider wird der Countdown über 0 hinaus fortgesetzt und hört nie auf. Aktualisieren Sie die Funktion `reduce_time`, um zu prüfen, ob der Timer kleiner als Null ist, und stoppen Sie in diesem Fall das Spiel

09-emoji-match.py

Sprache: **Python 3**

```
001.  # ------------------------------
002.  # Imports
003.  # ------------------------------
004.
005.  import os
006.  from random import shuffle, randint
007.  from guizero import App, Box, Picture,  PushButton, Text
008.
009.  # ------------------------------
010.  # Variables
011.  # ------------------------------
012.
013.  # set the path to the emoji folder on your computer
014.  emojis_dir = "emojis"
015.  emojis = [os.path.join(emojis_dir, f) for f in
016.  os.listdir(emojis_dir)]
017.  shuffle(emojis)
018.
019.  # ------------------------------
020.  # Functions
021.  # ------------------------------
022.
023.  def setup_round():
024.      for picture in pictures:
025.          picture.image = emojis.pop()
026.
027.      for button in buttons:
028.          button.image = emojis.pop()
029.          button.update_command(match_emoji, args=[False])
030.
031.      matched_emoji = emojis.pop()
032.
033.      random_picture = randint(0,8)
034.      pictures[random_picture].image = matched_emoji
035.
036.      random_button = randint(0,8)
037.      buttons[random_button].image = matched_emoji
038.
039.      buttons[random_button].update_command(
      match_emoji, [True])
040.
041.  def match_emoji(matched):
042.      if matched:
043.          result.value = "correct"
044.          score.value = int(score.value) + 1
045.      else:
046.          result.value = "incorrect"
047.          score.value = int(score.value) - 1
048.
049.      setup_round()
050.
051.  def reduce_time():
052.      timer.value = int(timer.value) - 1
053.      # is it game over?
054.      if int(timer.value) < 0:
055.          result.value = "Game over! Score = " + score.value
056.          # hide the game
057.          game_box.hide()
058.
059.  # ------------------------------
060.  # App
061.  # ------------------------------
062.
063.  app = App("emoji match")
064.
065.  game_box = Box(app, align="top")
066.
067.  top_box = Box(game_box, align="top", width="fill")
068.  Text(top_box, align="left", text="Score ")
069.  score = Text(top_box, text="4", align="left")
070.  timer = Text(top_box, text="30", align="right")
071.  Text(top_box, text="Time", align="right")
072.
073.  pictures_box = Box(game_box, layout="grid")
074.  buttons_box = Box(game_box, layout="grid")
075.
076.  pictures = []
077.  buttons = []
078.
079.  for x in range(0,3):
080.      for y in range(0,3):
081.          picture = Picture(pictures_box, grid=[x,y])
082.          pictures.append(picture)
083.
084.          button = PushButton(buttons_box, grid=[x,y])
085.          buttons.append(button)
086.
087.  result = Text(app)
088.
089.  setup_round()
090.
091.  app.repeat(1000, reduce_time)
092.
093.  app.display()
```

```
def reduce_time():
    timer.value = int(timer.value) - 1
    # is it game over?
    if int(timer.value) < 0:
        result.value = "Game over! Score =
" + score.value
        # hide the game
        pictures_box.hide()
        buttons_box.hide()
        timer.hide()
        score.hide()
```

Wenn der Timer weniger als 0 beträgt, wird die Meldung ***game over*** angezeigt und die Widgets des Spiels werden ausgeblendet, damit der Benutzer nicht mehr spielen kann. Siehe ***emoji4.py***, um eine Vorstellung davon zu bekommen, wie Ihr Code jetzt aussehen sollte. Starten Sie ihn und spielen Sie das Emoji-Match-Spiel. Fordern Sie einen Freund oder ein Familienmitglied zu einem Spiel heraus.

Vielleicht möchten Sie die Punkte- und Timer-Widgets in eine Box packen, damit sie besser angeordnet werden können (**Bild3**). Das komplette **09-emoji-match.py Listing** zeigt, wie man das macht.

Teil 09

GUIs erstellen mit Python: Einfaches Grafikprogramm

Selbstgemachte Software zum Zeichnen und Malen.

In diesem Beitrag erfahren Sie, wie Sie eine einfache Anwendung zum zeichnen von beliebigen Linien und Rechtecken programmieren können. Beachten Sie, dass Sie die Oberfläche Ihres Zeichenprogramms so gestalten können, wie Sie möchten – sie muss nicht dem gezeigten Beispiel (**Bild 1**) entsprechen. Dieses Tutorial gliedert sich in vier Abschnitte:

- Zeichnen von Punkten, die der Maus folgen
- Zeichnen von Linien zwischen den Punkten
- Ändern der Farben und Linienbreiten
- Rechtecke zeichnen

Zeichnen von Punkten

Der erste Schritt besteht darin, ein einfaches Programm zu schreiben, die das Widget ***Drawing*** und das Ereignis `when_mouse_dragged` verwendet, um Punkte auf dem Bildschirm zu zeichnen. Fügen Sie im Importbereich Ihres ansonsten leeren Programms folgende Widgets hinzu:

```
from guizero import App, Drawing
```

Erstellen Sie eine neue Funktion:

```
def draw(event):
    painting.oval(
        event.x - 1, event.y - 1,
        event.x + 1, event.y + 1,
        color="black")
```

… und dazu noch diesen Code zum App-Bereich:

```
app = App("Paint")

painting = Drawing(app, width="fill",
height="fill")
```

Bild 1

▲ **Bild 1** Das fertige Spiel.

```
painting.when_mouse_dragged = draw

app.display()
```

Ihr Code sollte nun dem Programm ***paint1.py*** ähneln. Das Widget ***Drawing*** füllt den gesamten verfügbaren Platz im Fenster aus. Wenn die Maus über die Zeichnung gezogen wird, wird die Funktion ***draw*** aufgerufen, die Punkte auf den Bildschirm zeichnet.

Die Zeichenfunktion wird jedes Mal aufgerufen, wenn ein Ereignis ausgelöst wird. Das Ereignis, das die x- und y-Position der Maus enthält, wird als Variable an die Funktion übergeben.

Allerdings gibt es ein Problem. Wenn Sie Ihre Maus zu schnell bewegen, zeichnet Ihr Programm eine Reihe von Punkten und keine durchgehende Linie (**Bild 2**). Das sieht nicht sehr schön aus, denn zwischen den Punkten gibt es Lücken, weil nicht für jedes Pixel, das die Maus überquert, ein Ereignis ausgelöst wird.

Linien zwischen den Punkten

Um dieses Problem zu lösen, werden Sie das Programm so ändern, dass es Linien zwischen den Punkten zeichnet. Auf diese Weise entsteht eine durchgehende Linie, wie sie auch von einem Stift oder Pinsel erzeugt wird. Sie müssen dazu das Ereignis `when_left_button_pressed` verwenden, um die Position zu speichern, an der die Linie beginnt. Zeichnen Sie dann eine gerade Linie zwischen dem Startpunkt der Linie und der nächsten Position, zu der die Maus gezogen wurde. Erstellen Sie eine neue Funktion, die aufgerufen wird, wenn die Maus gedrückt wird:

```
    def start(event):
        painting.last_event = event
```

… und fügen Sie folgenden Code zum App-Bereich hinzu:

```
    painting.when_left_button_pressed =
start
```

Fügen Sie nun das Widget ***Picture*** zu Ihren Importen hinzu::

```
from guizero import App, Box, Picture
```

Die Position, an der die Linie beginnt, wird in der Variablen `last_event` gespeichert. Ändern Sie die Funktion `draw`, um eine Linie zwischen dem Beginn der Linie und der Stelle, an welche die Maus gezogen wurde, zu zeichnen.

```
def draw(event):
        painting.line(
            painting.last_event.x, painting.
last_event.y,
            event.x, event.y,
            color="black",
            width=3
        )

        painting.last_event = event
```

> Das Programm muss geändert werden, damit auch Linien zwischen den Punkten auf dem Bildschirm erscheinen

paint1.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

Sprache: **Python 3**

```
001.  # simple paint app, just draw dots
002.
003.  # ------------------------------
004.  # Imports
005.  # ------------------------------
006.
007.  from guizero import App, Drawing
008.
009.  # ------------------------------
010.  # Functions
011.  # ------------------------------
012.
013.  def draw(event):
014.
015.      painting.oval(
016.          event.x - 1, event.y - 1,
017.          event.x + 1, event.y + 1,
018.          color="black")
019.
020.  # ------------------------------
021.  # App
022.  # ------------------------------
023.
024.  app = App("Paint")
025.
026.  painting = Drawing(app, width="fill", height="fill")
027.
028.  painting.when_mouse_dragged = draw
029.
030.  app.display()
```

Indem Sie die Variable `last_event` auf die aktuelle Position der Maus aktualisieren, wird beim nächsten Ziehen der Maus eine weitere Linie zwischen diesem und dem nächsten Punkt gezeichnet. Ihr Programm sollte wie das Programm ***paint2.py*** aussehen. Testen Sie es und stellen Sie sicher, dass Ihr „Zeichenstift" jetzt richtig funktioniert.

Ändern der Linienbreite und -Farbe

Sie verfügen bisher nur über eine Farbe und eine Stärke für Ihren Zeichenstift, was die Möglichkeiten zum Zeichnen einschränkt. Als Nächstes werden Sie Ihre grafische Benutzeroberfläche so ändern, dass Sie verschiedene Farben und Strichstärken wählen können.

Fügen Sie der grafischen Benutzeroberfläche zwei Widgets hinzu, die eine Farbe und eine Breite für die Linie festlegen:

```
 from guizero import App, Drawing, Combo,
Slider
```

Fügen Sie zusätzlich noch diese Zeile hinzu:

```
 color = Combo(app, options=["black",
"white", "red", "green", "blue"])
 width = Slider(app, start=1, end=10)
```

Vielleicht möchten Sie auch die Hintergrundfarbe Ihres Bildes ändern, damit es anders aussieht. Im Beispiel haben wir eine Combobox und einen Slider verwendet, aber Sie können auch andere Widgets wählen.

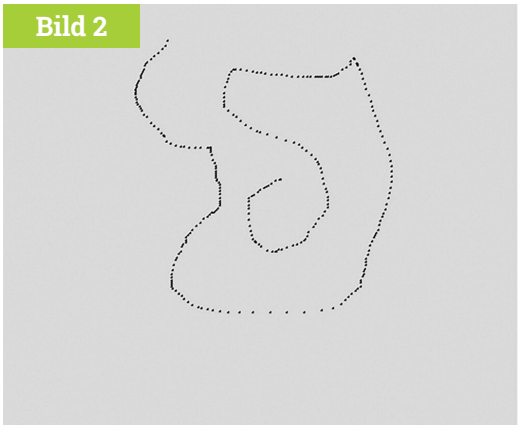
```
    painting.line(
        painting.last_event.x, painting.
last_event.y,
        event.x, event.y,
        color=color.value,
        width=width.value
    )
```

Testen Sie Ihren Code, der nun ähnlich wie derjenige von **paint3.py** sein sollte. Wenn alles richtig funktioniert, können Sie jetzt die Farbe und die Strichstärke auswählen.

Zeichnen von Formen

Sie werden Ihre Paint-Anwendung nun so erweitern, dass Sie gefüllte Rechtecke zeichnen können. Bei gedrückter Maustaste erscheint das Rechteck. Eine seiner Ecken befindet sich dort, wo sich der Mauszeiger beim Drücken befand. Die gegenüberliegende Ecke folgt der über den Bildschirm bewegten Maus, so dass sich die Größe

Bild 2



▲ **Bild 2** Keine gute Methode zum zeichnen von Linien.

paint2.py

▶ Sprache: **Python 3**

```
001.  # drawing lines by tracking when the mouse is clicked
002.
003.  # ------------------------------
004.  # Imports
005.  # ------------------------------
006.
007.  from guizero import App, Drawing
008.
009.  # ------------------------------
010.  # Functions
011.  # ------------------------------
012.
013.  def start(event):
014.      painting.last_event = event
015.
016.  def draw(event):
017.      painting.line(
018.          painting.last_event.x, painting.last_event.y,
019.          event.x, event.y,
020.          color="black",
021.          width=3
022.      )
023.
024.      painting.last_event = event
025.
026.  # ------------------------------
027.  # App
028.  # ------------------------------
029.
030.  app = App("Paint")
031.
032.  painting = Drawing(app, width="fill", height="fill")
033.
034.  painting.when_left_button_pressed = start
035.  painting.when_mouse_dragged = draw
036.
037.  app.display()
```

> Eine einzige Farbe und nur eine Stärke für den Stift schränken die Möglichkeiten zum zeichnen stark ein

Kreise und Ellipsen zeichnen

Im Beispiel aus dem ersten Abschnitt besteht jeder einzelne Punkt einer Linie in Wirklichkeit aus einem winzig kleinen Kreis. Können Sie Ihr Programm so ändern, dass es auch große Kreise bzw. Ellipsen zeichnet und dabei einen ähnlichen Prozess wie beim Zeichnen von Rechtecken verwendet? Tipp: Sehen Sie sich das Listing **10-paint.py** an, das auch die Werkzeuge ordentlich in einem Kasten ausrichtet.

Das Widget **Drawing** unterstützt auch das Zeichnen von Dreiecken und Polygonen. Schauen Sie sich die Dokumentation an (**lawsie.github.io/guizero/drawing**) und überlegen Sie, wie Sie diese Funktion verwenden können, um weitere interessante Formen zu erstellen, um damit die Möglichkeiten Ihres Zeichenprogramms zu erweitern.

paint3.py

Sprache: **Python 3**

```
# widgets to set the color and width

# ------------------------------
# Imports
# ------------------------------

from guizero import App, Drawing, Combo, Slider

# ------------------------------
# Functions
# ------------------------------

def start(event):
    painting.last_event = event

def draw(event):
    painting.line(
        painting.last_event.x, painting.last_event.y,
        event.x, event.y,
        color=color.value,
        width=width.value
    )

    painting.last_event = event

# ------------------------------
# App
# ------------------------------

app = App("Paint")

color = Combo(app, options=["black", "white", "red", "green", "blue"])
width = Slider(app, start=1, end=10)

painting = Drawing(app, width="fill", height="fill")

painting.when_left_button_pressed = start
painting.when_mouse_dragged = draw

app.display()
```

paint4.py

Sprache: **Python 3**

```
# adding different drawing shapes

# ------------------------------
# Imports
# ------------------------------

from guizero import App, Drawing, Combo, Slider

# ------------------------------
# Functions
# ------------------------------

def start(event):
    painting.last_event = event
    painting.first_event = event
    painting.last_shape = None

def draw(event):
    if shape.value == "line":
        painting.line(
            painting.last_event.x, painting.last_event.y,
            event.x, event.y,
            color=color.value,
            width=width.value
        )

    if shape.value == "rectangle":

        if painting.last_shape is not None:
            painting.delete(painting.last_shape)

        rectangle = painting.rectangle(
            painting.first_event.x, painting.first_event.y,
            event.x, event.y,
            color=color.value
        )

        painting.last_shape = rectangle

    painting.last_event = event

# ------------------------------
# App
# ------------------------------

app = App("Paint")

color = Combo(app, options=["black", "white", "red", "green", "blue"])
width = Slider(app, start=1, end=10)
shape = Combo(app, options=["line", "rectangle"])

painting = Drawing(app, width="fill", height="fill")

painting.when_left_button_pressed = start
painting.when_mouse_dragged = draw

app.display()
```

und die Proportionen des Rechtecks beliebig ändern lassen. Wenn die Maustaste losgelassen wird, wird es dann endgültig auf den Bildschirm gezeichnet.

Zu diesem Zweck müssen Sie Ihr Programm so ändern, dass es kontinuierlich Rechtecke zeichnet und löscht, bis die Maustaste losgelassen wird. Fügen wir Ihrer GUI ein Widget hinzu, mit dem Sie auswählen können, ob Sie eine Linie oder ein Rechteck zeichnen möchten. Fügen Sie dieses Widget in den App-Bereich ein:

```
shape = Combo(app, options=["line",
"rectangle"])
```

Ändern Sie die Zeichenfunktion so, dass nur Linien gezeichnet werden, wenn die Option `line` ausgewählt ist:

```
    if shape.value == "line":
        painting.line(
            painting.last_event.x,
painting.last_event.y,
            event.x, event.y,
            color=color.value,
            width=width.value
        )
```

Testen Sie Ihr Programm, um sicherzustellen, dass `line` auch dann noch funktioniert, wenn zuvor `rectangle` ausgewählt wurde.

Erstellen Sie zwei neue Variablen, die eine wichtige Rolle spielen, wenn die Maustaste gedrückt wird:

```
def start(event):
    painting.last_event = event
    painting.first_event = event
    painting.last_shape = None
```

Diese Variablen werden beim Zeichnen und Löschen des Rechtecks verwendet, bevor die Maustaste losgelassen wird.

> Das Programm zeichnet vor dem Loslassen der Maustaste ständig ein Rechteck und löscht es dann

```
    if shape.value == "rectangle":

        if painting.last_shape is not None:
            painting.delete(painting.last_
shape)

        rectangle = painting.rectangle(
            painting.first_event.x,
painting.first_event.y,
            event.x, event.y,
            color=color.value
        )

        painting.last_shape = rectangle
```

Das Programm zeichnet nun kontinuierlich ein Rechteck, löscht es dann und zeichnet es erneut, bis Sie die Taste loslassen.

Ihr komplettes Programm sollte jetzt ähnlich aussehen wie **paint4.py**. Viel Spaß beim Ausprobieren! Haben Sie schon eine Idee für Ihre erste Computergrafik?

Benutzerdefinierte Ereignisse

Um Ihre Zeichenanwendung dazu zu bringen, auf die Mausposition zu reagieren, haben Sie benutzerdefinierte Ereignisse verwendet (Custom Events). Die Ereignisse funktionieren ähnlich wie die normalen Widget-Befehlsparameter, indem Sie sie in eine Funktion einbinden, die aufgerufen wird, wenn das Ereignis eintritt.

Wenn Ihre Funktion aufgerufen wird, wird eine Variable übergeben, die Informationen über das aufgetretene Ereignis enthält, zum Beispiel die x- und y-Koordinaten der Maus. Die meisten Widgets, einschließlich der App selbst, unterstützen die folgenden Ereignisse:

- wenn geklickt wirdt – `when_clicked`
- wenn die linke Maustaste gedrückt wird – `when_left_button_pressed`
- wenn die linke Maustaste losgelassen wird – `when_left_button_released`
- wenn die rechte Maustaste gedrückt wird – `when_right_button_pressed`
- wenn die rechte Maustaste losgelassen wird – when_right_button_released
- wenn eine Taste gedrückt wird – `when_key_pressed`
- wenn eine Taste losgelassen wird – `when_key_released`
- wenn die Maus ein Widget betrittt – `when_mouse_enters`
- wenn die Maus ein Widget verlässt – `when_mouse_leaves`
- wenn die Maus über ein Widget gezogen wird – `when_mouse_dragged`

Diese Ereignisse können verwendet werden, um Ihre GUIs interaktiver zu gestalten.

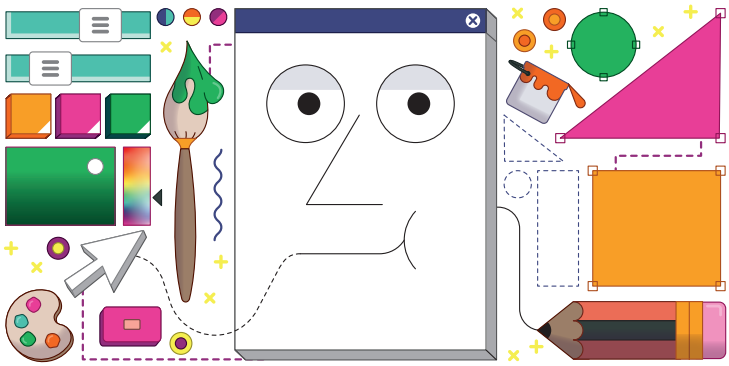
10-painting.py

Sprache: **Python 3**

```
# styled up

# ------------------------------
# Imports
# ------------------------------

from guizero import App, Drawing, Combo, Slider, Box, Text

# ------------------------------
# Functions
# ------------------------------

def start(event):
    painting.last_event = event
    painting.first_event = event
    painting.last_shape = None

def draw(event):
    if shape.value == "line":
        painting.line(
            painting.last_event.x, painting.last_event.y,
            event.x, event.y,
            color=color.value,
            width=width.value
        )

    else:
        if painting.last_shape is not None:
            painting.delete(painting.last_shape)

        if shape.value == "rectangle":

            painting.last_shape = painting.rectangle(
                painting.first_event.x,
                painting.first_event.y,
                event.x, event.y,
                color=color.value
            )

        if shape.value == "oval":

            painting.last_shape = painting.oval(
                painting.first_event.x,
                painting.first_event.y,
                event.x, event.y,
                color=color.value
            )

    painting.last_event = event

# ------------------------------
# App
# ------------------------------

app = App("Paint")
app.font = "impact"

tools = Box(app, align="top", width="fill", border=True)

Text(tools, text="Tool", align="left")
shape = Combo(tools, options=["line", "rectangle", "oval"], align="left")

Text(tools, text="Colour", align="left")
color = Combo(tools, options=["black", "white", "red", "green", "blue"], align="left")

Text(tools, text="Width", align="left")
width = Slider(tools, start=1, end=10, align="left")

painting = Drawing(app, width="fill", height="fill")

painting.when_left_button_pressed = start
painting.when_mouse_dragged = draw

app.display()
```

Create Graphical User Interfaces with Python

Weitere Anleitungen, wie Sie mit guizero eigene GUIs erstellen können, finden Sie in dem Buch *Create Graphical User Interfaces with Python*. Die 156 Seiten sind vollgepackt mit wichtigen Informationen und einer Reihe von spannenden Projekten.
magpi.cc/pythongui



Teil 10

Stop-Frame-Animation: GUIs erstellen mit Python

Animiertes Stop-Frame-GIF.

Dieses Projekt verwendet ein Raspberry Pi-Kameramodul und guizero, um eine Animation nach dem Stop-Frame-Prinzip zu erzeugen (Bild 1).

Zu diesem Projekt benötigen Sie einen Raspberry Pi mit einem offiziellen Kameramodul (oder einer High Quality Kamera). Wenn Sie Hilfe beim Anschließen des Kameramoduls brauchen, schauen Sie sich die Anleitung „Erste Schritte mit dem Kameramodul“ unter **rpf.io/picamera** an.

Sie müssen guizero mit der optionalen `images`-Funktionalität installieren, was mit folgendem Befehl im Terminal erreicht wird:

```
pip3 install guizero[images]
```

Dieses Projekt ist in mehrere Abschnitte unterteilt:

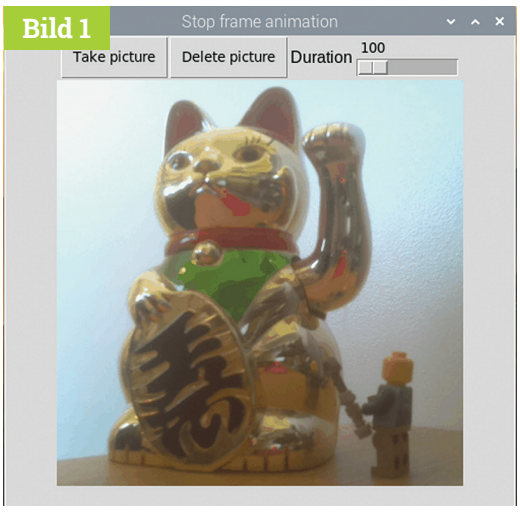
1. Aufnahme eines Bildes mit der Kamera und Anzeige auf einer GUI
2. Mehrere Bilder aufnehmen und sie in einem GIF speichern
3. Dem Benutzer die Möglichkeit geben, das GIF zu ändern
4. Aufräumen der GUI

Ein Bild aufnehmen

Beginnen Sie mit dem Schreiben des folgenden Programms:

```
# Imports ---------------
from guizero import App, Picture, PushButton
from picamera import PiCamera

# Functions -------------
def capture_image():
    camera.capture("frame.jpg")
    viewer.image = "frame.jpg"

# Variables -------------
camera = PiCamera(resolution="400x400")
# App -------------------
app = App(title="Stop frame animation")

take_next_picture = PushButton(app,
text="Take picture", command=capture_image)
viewer = Picture(app)

app.display()
```

Beachten Sie, dass die Verarbeitungszeit mit der Auflösung zunimmt. Der Wert 400 × 400 ist zwar klein, aber sehr schnell zu verarbeiten. Wenn die betreffende Schaltfläche auf der grafischen Oberfläche angeklickt wird, wird die Funktion `capture_image` aufgerufen. Die Funktion verwendet die Kamera, um ein Bild aufzunehmen und es als ***frame.jpg*** zu speichern. Das Bild wird dann im Picture-Widget angezeigt. Testen Sie das Programm (**stopframe1.py**). Wenn Sie auf die Schaltfläche ***Take picture*** klicken, wird das Bild auf der Oberfläche angezeigt.



▲ **Bild 1** Eine einfache Stop-Frame-Animation.

Mehrere Bilder aufnehmen und in einer GIF-Datei speichern

Eine Animation besteht aus mehreren Bildern, die als *Frames* bezeichnet werden. Wenn sich jedes Einzelbild der Animation nur geringfügig vom letzten unterscheidet, und wenn diese mit einer bestimmten Mindest-Geschwindigkeit abgespielt werden, scheint sich die Animation zu bewegen.

In diesem Schritt werden Sie Ihre grafische Benutzeroberfläche so ändern, dass sie eine Liste aller aufgenommenen Bilder enthält. Sie werden die PIL (Python Imaging Library) verwenden, um die Bilder als animierte GIF-Dateien zu speichern, die dann im Viewer angezeigt werden.

Importieren Sie am Anfang Ihres Programms das Image-Modul von PIL:

```
from PIL import Image
```

Erstellen Sie eine Liste, um die Bilder Ihrer Animation zu speichern:

```
frames = []
```

Um zu verfolgen, wie viele Bilder aufgenommen wurden, importieren Sie ein Text-Widget, fügen es zu Ihrer App hinzu und setzen es auf 0:

```
from guizero import App, Picture,
PushButton, Text

total_frames = Text(app, text="0")
```

Jedes Mal, wenn ein neues Bild aufgenommen wird, müssen Sie es öffnen und an die Liste der Bilder anhängen:

```
def capture_image():
    camera.capture("frame.jpg")
    viewer.image = "frame.jpg"

    frame = Image.open("frame.jpg")
    frames.append(frame)
    total_frames.value = len(frames)
```

Die Länge der frames-Liste (`len`) wird dann verwendet, um den Text in `total_frames` zu aktualisieren. Ihr Programm sollte nun ähnlich aussehen wie ***stopmotion2.py***. Testen Sie es und vergewissern Sie sich, dass die gezeigte Anzahl der Bilder bei jeder Aufnahme um den Wert 1 zunimmt..

Als GIF speichern

Sie können PIL verwenden, um alle Bilder als ein „animiertes GIF" zusammenzufassen. Erstellen Sie eine neue Funktion `save_animation`, um die Bilder als ***animation.gif*** zu speichern.

stopframe1.py

LADEN SIE DEN KOMPLETTEN CODE HERUNTER:
magpi.cc/guizerocode

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Picture, PushButton
004.  from picamera import PiCamera
005.
006.  # Functions -------------
007.
008.  def capture_image():
009.      camera.capture("frame.jpg")
010.      viewer.image = "frame.jpg"
011.
012.  # App -------------------
013.
014.  app = App(title="Stop frame animation")
015.
016.  camera = PiCamera(resolution="400x400")
017.  take_next_picture = PushButton(app, text="Take picture",
018.  command=capture_image)
019.  viewer = Picture(app)
020.
021.  app.display()
```

```
def save_animation():
 if len(frames) > 0:
     viewer.show()
     frames[0].save(
         "animation.gif",
         save_all=True,
         append_images=frames[1:])
     viewer.image = "animation.gif"
 else:
     viewer.hide()
```

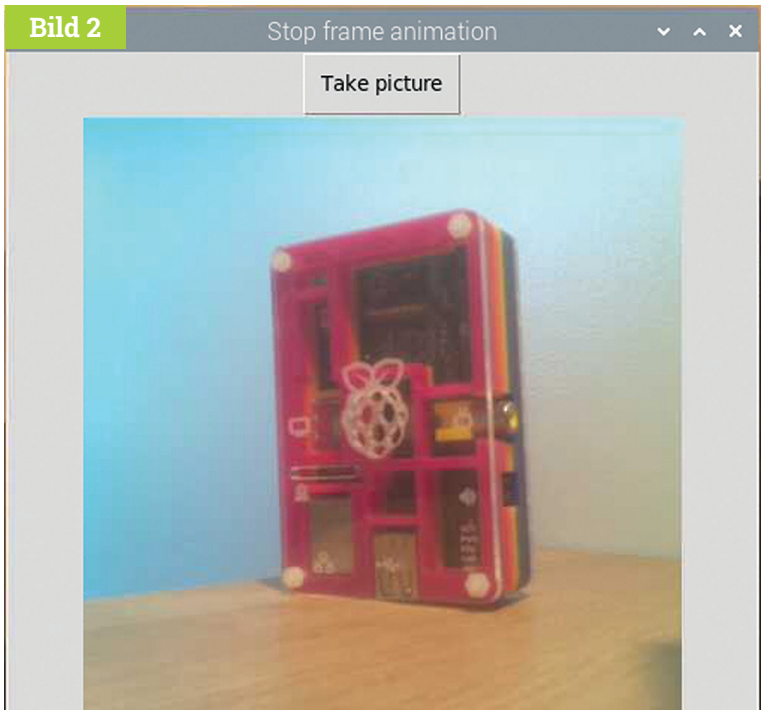
Hier passiert eine Menge, aber wenn man den Code aufschlüsselt, sieht man, wie alles funktioniert. Wenn die Anzahl der Bilder in der Liste größer als 0 ist, wird der Viewer angezeigt, andernfalls wird er ausgeblendet.

```
if len(frames) > 0:
     viewer.show()
     ...
 else:
     viewer.hide()
```

Die Bilder werden dann in einer Datei namens **animation.gif** gespeichert. Das heißt, eigentlich wird zunächst das erste Bild (`frames[0]`) gespeichert, und die verbleibenden Bilder (`frames[1:]`) werden angehängt, und alle zusammen werden dann in einer einzigen animierten GIF-Datei untergebracht.

```
frames[0].save(
        "animation.gif",
        save_all=True,
```

Bild 2



▲ **Bild 2** Aufnahme eines Bildes.

stopframe2.py

▶ Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Picture, PushButton, Text
004.  from picamera import PiCamera
005.  from PIL import Image
006.
007.  # Functions -------------
008.
009.  def capture_image():
010.      camera.capture("frame.jpg")
011.      viewer.image = "frame.jpg"
012.
013.      frame = Image.open("frame.jpg")
014.      frames.append(frame)
015.      total_frames.value = len(frames)
016.
017.  # Variables -------------
018.
019.  frames = []
020.
021.  camera = PiCamera(resolution="400x400")
022.
023.  # App -------------------
024.
025.  app = App(title="Stop frame animation")
026.
027.  total_frames = Text(app, text="0")
028.  take_next_picture = PushButton(app, text="Take picture",
029.  command=capture_image)
030.  viewer = Picture(app)
031.
032.  app.display()
```

```
        append_images=frames[1:])
```

animation.gif wird dann auf dem Viewer angezeigt:

```
    viewer.image = "animation.gif"
```

Rufen Sie die Funktion `save_animation` am Ende der Funktion `capture_image` auf, um die Animation zu erstellen und anzuzeigen.

```
def capture_image():
    camera.capture("frame.jpg")
    viewer.image = "frame.jpg"

    save_animation()
```

Ihr Code sollte jetzt ähnlich aussehen wie ***stopframe3.py***. Probieren Sie ihn aus.

Das letzte Bild löschen

Wenn Sie bei der Erstellung Ihres animierten GIFs einen Fehler machen, müssen Sie nach dem bisherigen Stand des Codes wieder von vorne beginnen. Sie sollten Ihre Benutzeroberfläche daher so ändern, dass das letzte Bild gelöscht werden kann, so dass Sie im Falle eines Fehlers die Änderung rückgängig machen können. Erstellen Sie eine neue Funktion, die das letzte Bild aus der Liste löscht oder ausblendet, die geänderte Animation speichert und dann anzeigt:

```
def delete_frame():
    if len(frames) > 0:
        frames.pop()
        total_frames.value = len(frames)

    save_animation()
```

Die Länge der frames-Liste wird überprüft, bevor versucht wird, das letzte Element zu löschen. Wenn Sie versuchen, ein Element aus einer leeren Liste zu entfernen, wird eine Fehlermeldung ausgegeben. Fügen Sie einen PushButton in die GUI ein, um die Funktion `delete_frames` aufzurufen, indem Sie diesen Code einfügen:

```
 delete_last_picture = PushButton(controls,
align="left", text="Delete last",
command=delete_frame)
```

Hinweis: Sie könnten die grafische Benutzeroberfläche auch so ändern, dass Sie nicht nur das letzte Bild, sondern jedes beliebige Bild löschen können.

stopframe3.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Picture, PushButton, Text
004.  from picamera import PiCamera
005.  from PIL import Image
006.
007.  # Functions -------------
008.
009.  def capture_image():
010.      camera.capture("frame.jpg")
011.      viewer.image = "frame.jpg"
012.
013.      frame = Image.open("frame.jpg")
014.      frames.append(frame)
015.      total_frames.value = len(frames)
016.
017.      save_animation()
018.
019.  def save_animation():
020.      if len(frames) > 0:
021.          viewer.show()
022.          frames[0].save(
023.              "animation.gif",
024.              save_all=True,
025.              append_images=frames[1:])
026.          viewer.image = "animation.gif"
027.      else:
028.          viewer.hide()
029.
030.  # Variables -------------
031.
032.  frames = []
033.
034.  camera = PiCamera(resolution="400x400")
035.
036.  # App -------------------
037.
038.  app = App(title="Stop frame animation")
039.
040.  total_frames = Text(app, text="0")
041.  take_next_picture = PushButton(app, text="Take
042.  picture", command=capture_image)
043.
044.  viewer = Picture(app)
045.
046.  app.display()
```

Ändern des Timings

Jedes Bild wird für die Standarddauer von 100 Millisekunden angezeigt. Fügen Sie ein Schieberegler-Widget in Ihre grafische Benutzeroberfläche ein, um die Dauer zu ändern.

Fügen Sie es zur Liste der Importe hinzu:

```
from guizero import App, Picture,
PushButton, Text, Slider
```

Erstellen Sie dann das Widget in der App:

```
   Text(app, text="Duration")
duration = Slider(app, start=100, end=1000,
command=save_animation)
```

Die Parameter `start` und `end` sind die Mindest- und Höchstwerte, die Sie für die Bilddauer festlegen können.

Jedes Mal, wenn der Schieberegler verändert wird, wird die Funktion `save_animation` ausgeführt. Aktualisieren Sie diese Funktion, um den Wert für die Dauer beim Speichern des GIFs zu verwenden:

```
frames[0].save(
        "animation.gif",
        save_all=True,
        append_images=frames[1:],
        duration=duration.value)
```

Ihr Code sollte nun dem von ***stopmotion4.py*** ähneln. Probieren Sie ihn aus.

> Jedes Bild wird für die Standarddauer von 100 Millisekunden angezeigt

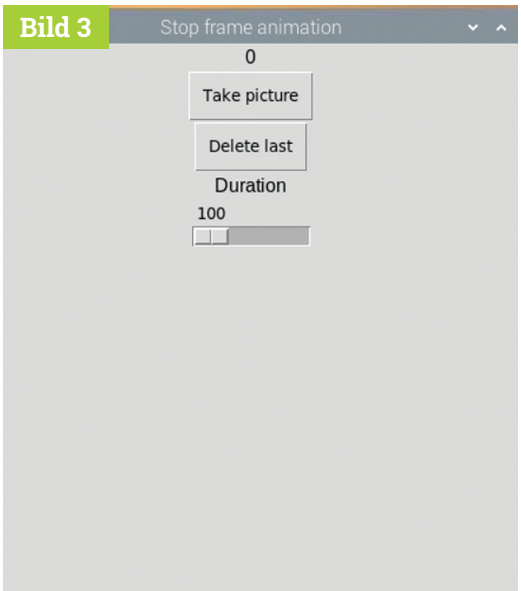


Bild 3 Am oberen Rand untereinander angeordnete Steuerelemente.

stopframe4.py

Sprache: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Picture, PushButton, Text,
004.  Slider
005.  from picamera import PiCamera
006.  from PIL import Image
007.
008.  # Functions -------------
009.
010.  def capture_image():
011.      camera.capture("frame.jpg")
012.      viewer.image = "frame.jpg"
013.
014.      frame = Image.open("frame.jpg")
015.      frames.append(frame)
016.      total_frames.value = len(frames)
017.
018.      save_animation()
019.
020.  def save_animation():
021.      if len(frames) > 0:
022.          viewer.show()
023.          frames[0].save(
024.              "animation.gif",
025.              save_all=True,
026.              append_images=frames[1:],
027.              duration=duration.value)
028.          viewer.image = "animation.gif"
029.      else:
030.          viewer.hide()
031.
032.  def delete_frame():
033.      if len(frames) > 0:
034.          frames.pop()
035.          total_frames.value = len(frames)
036.
037.      save_animation()
038.
039.  # Variables -------------
040.
041.  frames = []
042.
043.  camera = PiCamera(resolution="400x400")
044.
045.  # App -------------------
046.
047.  app = App(title="Stop frame animation")
048.
049.  total_frames = Text(app, text="0")
050.  take_next_picture = PushButton(app,
      text="Take picture", command=capture_image)
051.  delete_last_picture = PushButton(app,
      text="Delete last", command=delete_frame)
052.  Text(app, text="Duration")
053.  duration = Slider(app, start=100, end=1000,
      command=save_animation)
054.
055.  viewer = Picture(app)
056.
057.  app.display()
```

Ausrichten der Steuerelemente

Im Moment nehmen die Steuerelemente viel Platz am oberen Rand der GUI ein (**Bild 3**). Erstellen Sie eine Box und richten Sie diese am oberen Rand der GUI aus, um die Steuerelemente aufzunehmen, indem Sie diese zunächst zu den Importen hinzufügen.

```
    from guizero import App, Picture,
PushButton, Text, Slider, Box

  controls = Box(app, align="top")
```

Ändern Sie die Widgets so, dass sie sich in der Controls-Box befinden und setzen Sie den Parameter `align` auf `left`. Zum Beispiel:

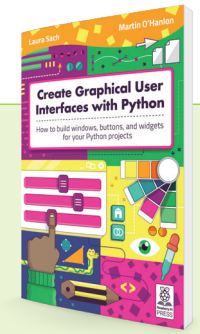
```
  total_frames = Text(controls, text="0",
align="left")
```

Wenn Sie die Widgets innerhalb des Rahmens nach links ausrichten, werden sie nebeneinander gestapelt.

Wiederholen Sie dies für die restlichen Steuerelemente, so dass sie alle in den oberen Rahmen eingefügt und nebeneinander aufgereiht werden. Ihr komplettes Programm sollte ähnlich aussehen wie das Programm mit dem Namen ***11-stop-motion.py***.

Grafische Benutzeroberflächen mit Python erstellen

(Create Graphical User Interfaces with Python)

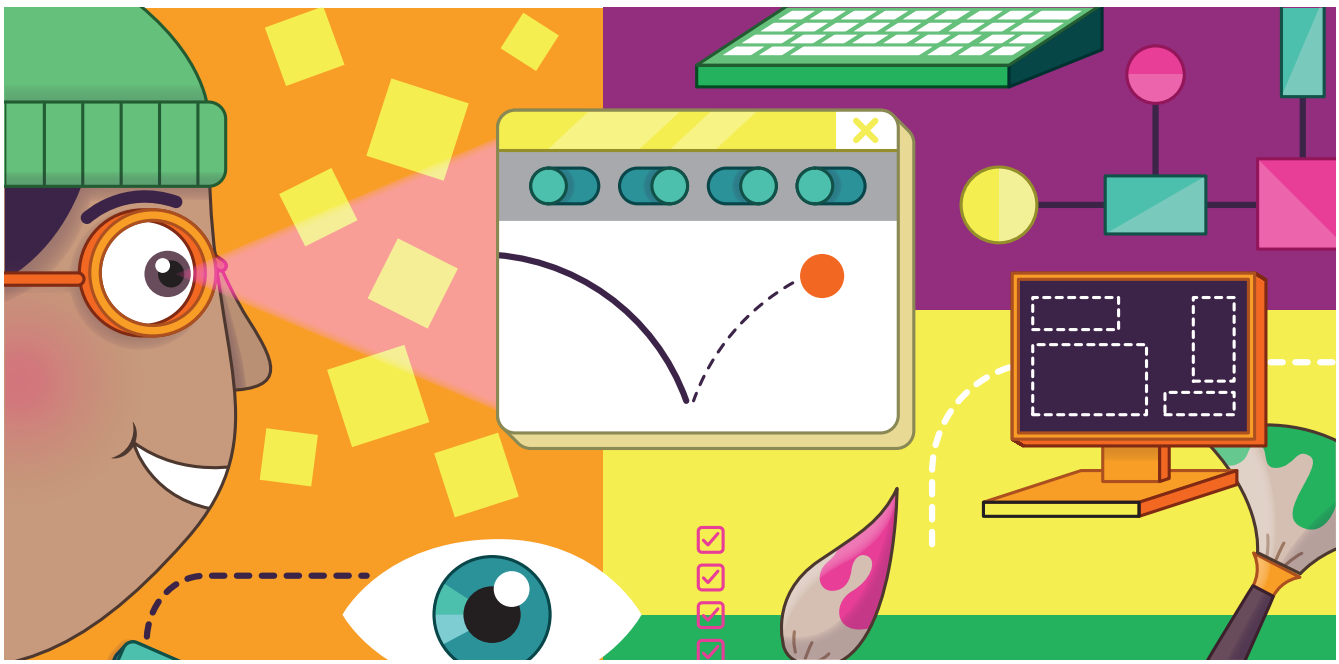


Weitere Anleitungen, wie Sie mit guizero eigene grafische Benutzeroberflächen erstellen können, finden Sie in unserem Buch *Grafische Benutzeroberflächen mit Python erstellen (Create Graphical User Interfaces with Python)*. Die 156 Seiten sind vollgepackt mit wichtigen Informationen und einer Reihe von spannenden Projekten.
magpi.cc/pythongui

11-stopmotion.py

Sprache: **Python 3**

```
# Imports ---------------

from guizero import App, Picture, PushButton, Text,
Slider, Box
from picamera import PiCamera
from PIL import Image

# Functions -------------

def capture_image():
    camera.capture("frame.jpg")
    viewer.image = "frame.jpg"

    frame = Image.open("frame.jpg")
    frames.append(frame)
    total_frames.value = len(frames)

    save_animation()

def save_animation():
    if len(frames) > 0:
        viewer.show()
        frames[0].save(
            "animation.gif",
            save_all=True,
            append_images=frames[1:],
            duration=duration.value)
        viewer.image = "animation.gif"
    else:
        viewer.hide()

def delete_frame():
    if len(frames) > 0:
        frames.pop()
        total_frames.value = len(frames)

    save_animation()

# Variables -------------

frames = []

camera = PiCamera(resolution="400x400")

# App -------------------

app = App(title="Stop frame animation")

controls = Box(app, align="top")
total_frames = Text(controls, text="0", align="left")
take_next_picture = PushButton(controls, align="left",
text="Take picture", command=capture_image)
delete_last_picture = PushButton(controls,
align="left", text="Delete last", command=delete_
frame)
Text(controls, align="left", text="Duration")
duration = Slider(controls, align="left", start=100,
end=1000, command=save_animation)

viewer = Picture(app)

app.display()
```



Der offizielle MagPi Newsletter

Das Pi Community Journal ist für alle Fans von Raspberry Pi Inhalten. Der Newsletter versorgt euch mit weiteren Projekten, Artikeln und Neuigkeiten, welche ihr sonst nicht bekommt. Außerdem habt ihr jeden Monat die Chance an unserer Verlosung teilzunehmen.

Also, worauf wartet ihr noch? Meldet euch noch heute für unseren Pi Community Journal unter www.magpi.de/journal an.

MagPi
www.magpi.de Magazine