elektor

HET MAGAZINE GESCHREVEN DOOR EN VOOR DE RASPBERRY PI-COMMUNITY

Het officiële RASPBERRY PI magazine

MagPi

Special Nr. 1 | magpi.nl

Het officiële Raspberry Pi-magazine

GUI's PROGRAMMEREN

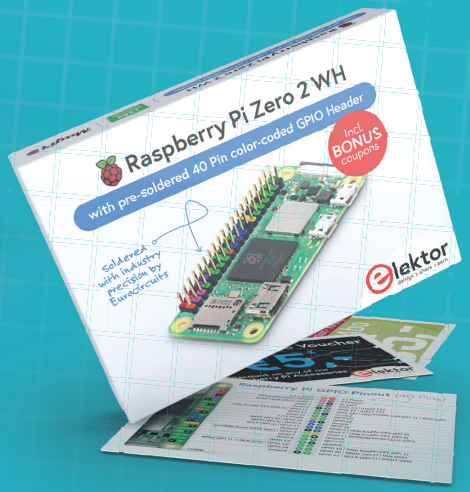
GRAFISCHE GEBRUIKER-INTERFACES MAKEN MET PYTHON

SPECIAL: **60 PAGINA'S** PRAKTIJK

Abonneer nu!

slechts
€ 54,95
per jaar
(6 nummers)

- **Zes edities van MagPi Magazine**
- **Gratis Raspberry Pi Zero 2 W met 40-pins kleur-gecodeerde GPIO-header**



NU TE BESTELLEN OP
WWW.MAGPI.NL/ABO

MagPi Magazine
www.magpi.nl

Inhoud

Maak GUI's met Python

02 01 Introductie in guizero

Installeer de guizero Python-bibliotheek en begin met het maken van je eigen GUI's.

07 02 Spionnennaamkiezer

Maak een interactieve GUI-toepassing.

10 03 Meme-generator

Ontwerp een GUI-applicatie die memes maakt.

15 04 's Werelds slechtste GUI

Leer een goede GUI ontwerpen door het eerst helemaal fout te doen!

19 05 Boter-kaas-en-eieren

Gebruik je GUI om een eenvoudig spel te besturen..

28 06 Vernietig de rondjes

Leer hoe je een Waffle gebruikt om een smakelijk spel te maken.

35 07 Flood it

Leer hoe je een Waffle gebruikt om een smakelijk spel te maken.

41 08 Emoji Match

Maak een leuk spel 'zoek het identieke plaatje'.

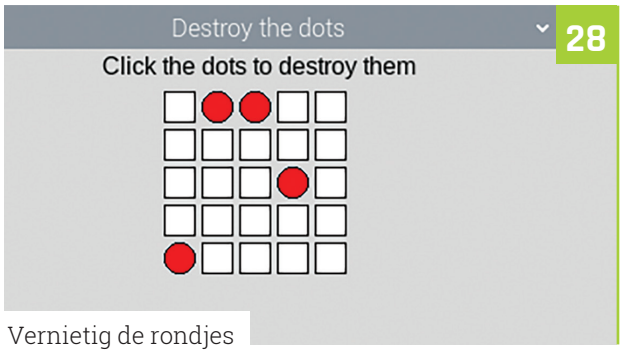
48 09 Paint

Maak een grappig tekenspel.

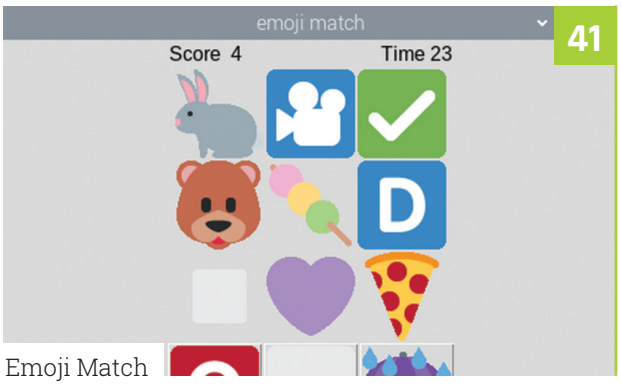
54 10 Stopframeanimatie

Bouw je eigen stopframeanimatie GIF-tool.

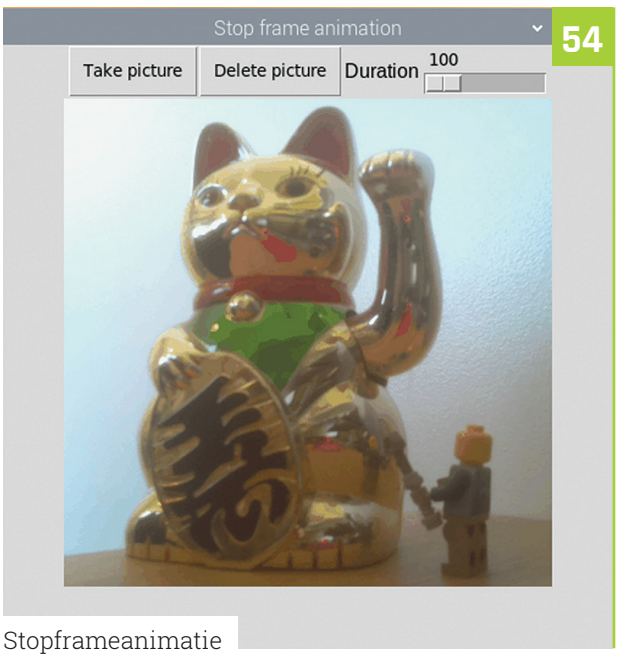
Spionnennaamkiezer



Vernietig de rondjes



Emoji Match



Stopframeanimatie

Deel 01

Maak GUI's met Python:

Introductie in guizero

Installeer de guizero Python-bibliotheek en begin met het maken van je eigen GUI's.

MAKER

Laura Sach

Laura leidt het A Level team van de Raspberry Pi Foundation en creëert zo middelen voor leerlingen om te leren over informatica.

@CodeBoom

MAKER

Martin O'Hanlon

Martin werkt in het leerteam van de Raspberry Pi Foundation, waar hij online cursussen, projecten en leermiddelen maakt.

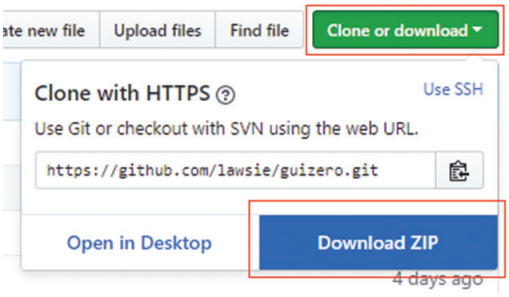
@martinohanlon

Een grafische gebruikersinterface (GUI) is een manier om je Python-programma's gebruiksvriendelijker en spannender te maken. Je kunt verschillende componenten, genaamd 'widgets', aan je interface toevoegen, waardoor er veel verschillende manieren zijn om informatie in het programma in te voeren en als uitvoer weer te geven. Je kunt mensen op een knop laten drukken, een stukje tekst laten weergeven of zelfs een optie laten kiezen uit een menu. In deze serie zullen we gebruik maken van de guizero-bibliotheek, die is ontwikkeld met als doel beginners te helpen bij het gemakkelijk bouwen van GUI's.

Python's standaard GUI-pakket heet tkinter, en is met Python op de meeste platformen al geïnstalleerd. De guizero-bibliotheek is een wrapper voor tkinter – dit betekent dat het een veel eenvoudigere manier biedt om Python's standaard GUI-bibliotheek te gebruiken.

01 Installeer guizero

Je moet de guizero Python-bibliotheek (**lawsie.github.io/guizero**) installeren om de programma's in dit artikel te kunnen maken. Het is beschikbaar als Python-pakket, wat herbruikbare code is die je kunt downloaden, installeren en vervolgens kunt gebruiken in je programma's.



Een alternatieve manier om guizero te installeren is door het zip-bestand van GitHub te downloaden.

Hoe je guizero installeert hangt af van je besturingssysteem en de rechten die je hebt om je computer te bedienen.

Als je toegang hebt tot de commandoregel of de terminal, kun je het volgende commando gebruiken:

```
pip3 install guizero
```

Uitgebreide installatie-instructies voor guizero zijn beschikbaar op **lawsie.github.io/guizero**, inclusief opties voor het installeren wanneer je geen administratorrechten hebt op je computer en downloadbare installaties voor Windows.

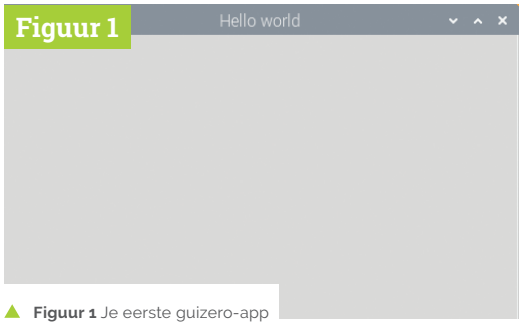
02 Hallo Wereld

Nu je guizero hebt geïnstalleerd, laten we eens kijken of het werkt en een kleine 'hello-world' app schrijven, wat traditioneel is voor programmeurs om te doen als eerste programma bij het gebruik van een nieuwe tool of taal.

Open de editor waar je je Python-code mee zult schrijven. Aan het begin van elk guizero-programma kies je de widgets die je nodig hebt uit de guizero-bibliotheek en importeer je ze. Je hoeft elke widget maar één keer te importeren en kunt hem vervolgens zo vaak als je wilt in je programma gebruiken.

Zet deze code bovenaan de pagina om de App-widget te importeren:

```
from guizero import App
```



▲ **Figuur 1** Je eerste guizero-app

Alle guizero projecten beginnen met een hoofdvenster dat een container-widget is die App heet. Aan het einde van elk guizero-programma moet je het programma vertellen dat de app die je net hebt gemaakt moet worden weergegeven. Voeg deze twee regels code toe onder de regel waar je de App-widget heeft geïmporteerd:

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

Sla nu je code op en voer die uit. Je zou nu een GUI-venster moeten zien met de titel 'Hello world' (**Figuur 1**). Gefeliciteerd, je hebt zojuist je eerste guizero-app gemaakt!

03 Widgets toevoegen

Widgets zijn de dingen die op de GUI verschijnen, zoals tekstvakken, knoppen, schuifregelaars en zelfs doodgewone stukjes tekst.

Alle widgets komen tussen de regel code om de App te maken en de regel `app.display()`. Hier is de app die je net hebt gemaakt, maar in dit voorbeeld hebben we een tekstwidget toegevoegd:

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

Is het je opgevallen dat er twee veranderingen zijn (**Figuur 2**)? Er is nu een extra regel code om de Text-widget toe te voegen, en we hebben ook Text toegevoegd aan de lijst van widgets om te importeren in de allereerste regel.

Laten we de code van de Text-widget wat beter bekijken:

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

Net als elke variabele in Python heeft een widget een naam nodig. Deze heet 'message'. Vervolgens geven we aan dat we willen dat dit een 'Text'-widget is.

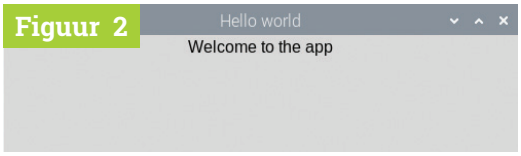
Tussen de haakjes staan enkele parameters om de Text-widget te vertellen hoe deze eruit moet zien. De eerste, 'app', vertelt de Text waar hij moet komen. Alle widgets moeten in een container-widget zitten. Meestal zitten je widgets direct in een App, maar je zult later ontdekken dat er ook andere soorten container-widgets zijn waar je dingen in kunt zetten. Tot slot zeggen we dat de widget de tekst "Welcome to the app" moet bevatten.

01-helloworld.py

DOWNLOAD DE VOLLEDIGE CODE: magpi.cc/guizerocode

Taal: **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()
```



◀ **Figuur 2** Een tekstbericht toevoegen.

Wanted-poster

01 Maak het spannender

Nu je een basis-GUI kunt maken, willen we het er wat spannender laten uitzien. Je kunt tekst in verschillende lettertypen, formaten en kleuren toevoegen, de achtergrondkleur veranderen en ook foto's toevoegen. Laten we, om dit allemaal te oefenen, een 'Wanted'-poster maken.

Allereerst moet je beginnen met het maken van een app. Zet deze code in je editor om het meest basale app-venster te maken:

```
from guizero import App

app = App("Wanted!")

app.display()
```

Sla je code op en voer deze uit en je zou een app moeten zien die eruitziet als een gewoon grijs vierkant met de titel 'Wanted!' bovenin (**Figuur 3**).

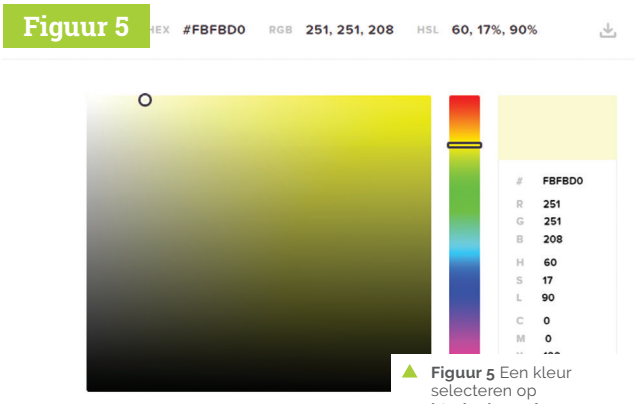
02 Achtergrondkleuren

Laten we de achtergrond van de app een beetje anders maken. Traditioneel zien wanted-posters eruit alsof ze van perkament zijn gemaakt, dus laten we een lichtgele kleur toevoegen als achtergrond.

Je hebt nodig

- Een computer (bijv. Raspberry Pi, Apple Mac, Windows- of Linux-pc)
- Internetverbinding
- Python 3 (**python.org**)
- Een IDE (code editor), bijv: IDLE (geïnstalleerd met Python 3), Thonny (**thonny.org**), Mu (codewith.mu), PyCharm (**jetbrains.com/pycharms**)
- De guizero Python-bibliotheek (**lawsie.github.io/guizero**)

Figuur 3
De basis-app



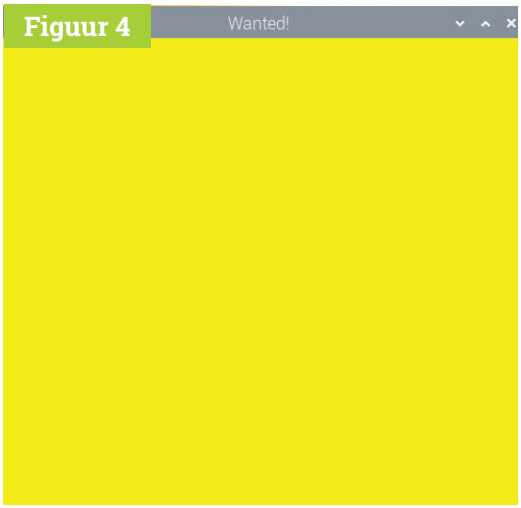
Figuur 5 Een kleur selecteren op **htmlcolorcodes.com**

Zoek de coderegel waar je de app maakt. Voeg direct onder deze regel nog een regel code toe om de eigenschap `bg` van het venster aan te passen. `bg` is een afkorting voor 'background' en laat ons de kleur van de achtergrond veranderen. Nu moet je code er zo uitzien:

```
from guizero import App

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

app.display()
```

Dit wordt het bewerken van een eigenschap genoemd. In de code moet je de widget waar je het over hebt (app), de eigenschap die je wilt veranderen (bg) en de waarde die je wilt veranderen opgeven.

Deze kleur (**Figuur 4**) is misschien iets te geel, laten we daarom de hex-code van een andere kleur geel opzoeken. Er zijn veel websites waar je kunt zoeken naar kleuren, je kunt bijvoorbeeld **htmlcolorcodes.com** proberen (**Figuur 5**).



Figuur 4 Verander de achtergrondkleur

Wanneer je de gewenste kleur hebt geselecteerd, zie je de code ervan op de site verschijnen als hexadecimaal (in dit geval #FBFBD0) of als RGB (251, 251, 208). Je kunt beide formaten gebruiken voor het instellen van kleuren in guizero; je kunt bijvoorbeeld de code voor het geel maken van je achtergrond verwijderen en dan een van deze opties in je programma proberen:

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

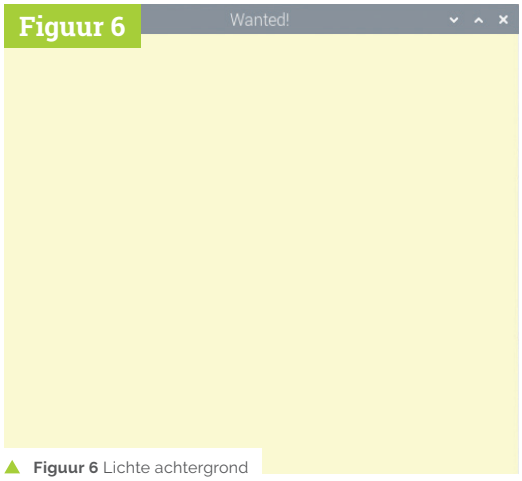
> bg is een afkorting voor 'background' en laat ons de kleur van de achtergrond veranderen

Lees de documentatie

Je vraagt je misschien af hoe je te weten komt welke eigenschappen een bepaalde widget heeft die je kunt veranderen. Zelfs als je een beginnend programmeur bent, is het de moeite waard om te leren hoe je documentatie moet lezen, omdat het je de volledige kracht van guizero en alle andere bibliotheken die je tegenkomt laat gebruiken.

De guizero documentatie is te vinden op **lawsie.github.io/guizero**. Daar klik je op het menu Widgets en op de widget die je wilt veranderen, scroll naar beneden tot je bij de sectie Properties komt. Als je bijvoorbeeld de widget 'Text' selecteert dan zie je hier alle eigenschappen van een stuk Text die je eventueel kunt wijzigen. De documentatie bevat ook vaak nuttige stukjes code die je laten zien hoe je een bepaalde eigenschap of methode kunt gebruiken, dus wees niet bang om een kijkje te nemen – je weet nooit wat je kunt leren!

Figuur 6



Figuur 6 Lichte achtergrond

03 Tekst toevoegen

Je app zou er ongeveer zoals **Figuur 6** moeten uitzien. Laten we nu wat tekst toevoegen aan de GUI. We beginnen met het toevoegen van de tekst die alle goede wanted-posters nodig hebben - het woord 'Wanted'!

Zoek eerst de regel code die je al hebt, waar je de App hebt geïmporteerd.

```
from guizero import App
```

Je moet Text importeren om een stuk tekst te kunnen maken, dus voeg het toe aan het einde van de lijst. Nu ziet de regel er zo uit:

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

Telkens wanneer je een nieuw type widget wilt gebruiken, voeg je de naam toe aan het einde van de lijst. Het is niet nodig om hele nieuwe regels code toe te voegen: houd het gewoon op een lijst in één regel, zodat je programma niet te ingewikkeld wordt.

Nu je tekst kunt gebruiken, willen we ook een stukje tekst toevoegen. Vergeet niet dat alle widgets op de GUI moeten worden toegevoegd tussen de regel code waar je de App maakt en de regel code waar je deze weergeeft. Je code moet er nu zo uitzien:

```
from guizero import App, Text

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

wanted_text = Text(app, "WANTED")

app.display()
```

Hier is `wanted_text` de naam van het stuk tekst. Dit is zodat we er later in de code over kunnen praten – denk aan de naam van een persoon. (Je zou je stuk tekst zelfs Dave kunnen noemen als je dat wilt – voor de computer maakt het niet uit).

Tussen de haakjes hebben we twee dingen. Het tweede, "WANTED", is eenvoudig omdat het de tekst is die we op het scherm willen weergeven. Het eerste is de container die dit stukje tekst bestuurt, die zijn 'master' wordt genoemd. In dit geval zeggen we dat deze tekst door de app moet worden bestuurd. Wanneer je voor het eerst start met het maken van GUI's, zullen de meeste van je widgets de app als hun 'master' hebben, maar er zijn andere containers die widgets kunnen opslaan en die je later zult leren kennen.

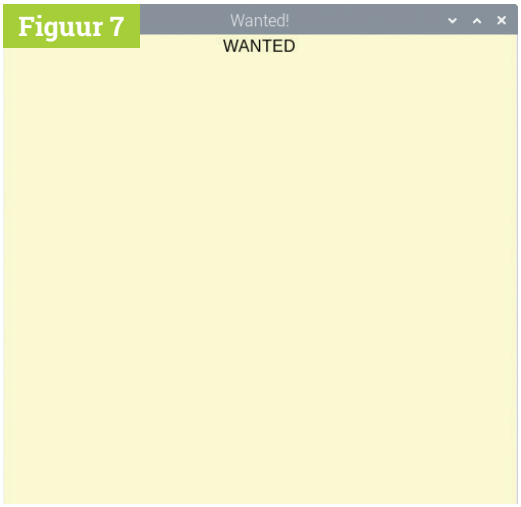
04 Wijzig grootte en kleur van tekst

Oh oh, deze tekst is vrij klein (**Figuur 7**). Laten we de eigenschap `text_size` op precies dezelfde manier veranderen als toen we de achtergrondkleur van de app veranderden. Vergeet niet dat je drie dingen moest specificeren:

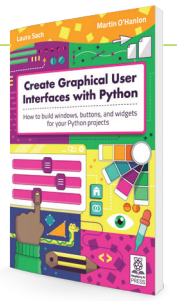
1. De naam van de widget
2. De eigenschap die veranderd moet worden
3. De nieuwe waarde die hij moet krijgen

Dus, in dit geval geef je de widget (`wanted_size`), de te wijzigen eigenschap (`text_size`) en de nieuwe waarde (`50`) op. Voeg een nieuwe regel code toe direct onder de regel waar je de tekst hebt aangemaakt, om de eigenschap te wijzigen.

Figuur 7



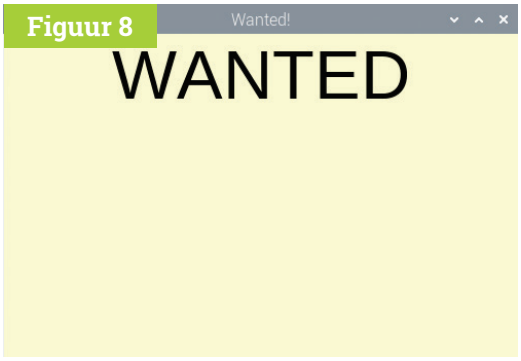
Figuur 7 De tekst is te klein



Create Graphical User Interfaces with Python

Voor meer tutorials over het maken van je eigen GUI's met guizero, kun je terecht in ons nieuwe boek, *Create Graphical User Interfaces with Python*. De 156 pagina's staan vol met essentiële informatie en een reeks spannende projecten. **magpi.cc/pythongui**

Figuur 8 Grotere tekst




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

Je hebt nu een grotere tekst op je poster (**Figuur 8**). Kijk of je nu het lettertype van deze tekst kunt veranderen in iets anders. Welke lettertypes beschikbaar zijn hangt af van welk besturingssysteem je gebruikt, dus hier zijn enkele suggesties:

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

Geen enkele 'wanted'-poster zou compleet zijn zonder foto, dus laten we er een toevoegen. Onze poster is voor de kat van de auteur, omdat zij altijd aan dingen krabt waar ze niet aan mag krabben.

Sla een kopie van de foto die je wilt gebruiken op in dezelfde map als je GUI-programma. Je kunt afbeeldingen in andere mappen gebruiken, maar als je dat doet zul je het pad naar de afbeelding moeten aangeven, dus het is een stuk makkelijker om ze in het begin gewoon in dezelfde map op te slaan.

Hopelijk ben je nu gewend aan het toevoegen van widgets. Vergeet niet dat ze altijd aan het begin van het programma moeten worden geïmporteerd, en vervolgens komt de widget die wordt aangemaakt met een sprekende naam onder de regel met code waarmee je de App maakt, maar boven de afsluitende regel met `app.display()`.

Voeg 'Picture' toe aan de lijst van de te importeren widgets aan het begin van het programma.

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

Maak nu een Picture-widget met twee parameters: de app en de bestandsnaam van de afbeelding. Dit is de code die we hebben gebruikt omdat onze foto **tabitha.png** heet.

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

Voer je code (die moet lijken op **02-wanted.py**) opnieuw uit en je zou de afbeelding onder je tekst moeten zien (**Figuur 9**).

Nu is het aan jou om je nieuwe GUI-vaardigheden te gebruiken en de poster vorm te geven zoals je maar wilt.

Figuur 9 De voltooide poster

02-wanted.py

DOWNLOAD DE VOLLEDIGE CODE:
magpi.cc/guizerocode

Taal: **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()
```

Beeldmanipulatie

Omdat guizero een bibliotheek is voor beginners en we de installatie zo eenvoudig mogelijk wilden maken, zitten er geen verfijnde beeldbewerkingsfuncties in, omdat hiervoor een extra bibliotheek nodig is die 'pillow' heet. Je kunt altijd niet-geanimeerde GIF-plaatjes op elk platform gebruiken, en PNG-plaatjes op alle platforms behalve Mac, dus als je niet zeker weet of je de extra beeldbewerkingsfuncties hebt geïnstalleerd, houd het dan op die twee bestandstypes.

Deel 02

GUI's maken met Python: Spionnennaamkiezer

Maak een interactieve GUI-toepassing.

Tot nu toe heb je in deze serie geleerd hoe je je GUI kunt aanpassen met een verscheidenheid aan verschillende opties. Nu is het tijd om echt interactief te worden en een GUI-applicatie te maken die echt reageert op input van de gebruiker. Wie wil nou niet op een grote rode knop drukken om een supergeheime spionnennaam te genereren?

Aangezien je al weet hoe je een app moet maken, maak dan eens een basisvenster en voeg wat tekst toe als je wilt. Hier is wat code om je op weg te helpen, en deze code bevat ook wat commentaar (de regels die beginnen met een #) om je te helpen je programma te structureren:

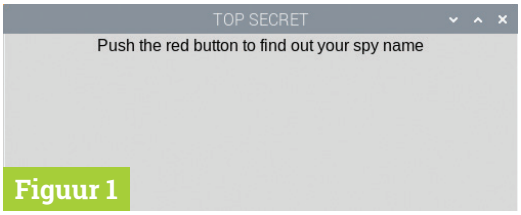
```
# Imports ---------------
from guizero import App, Text

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

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

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

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

Voer deze code uit en je zou een venster moeten zien met de tekst (**Figuur 1**).

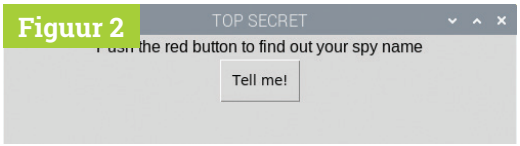
Voeg een knop toe

Laten we nu een knop toevoegen aan de GUI. Voeg PushButton toe aan je lijst van imports zodat je knoppen kunt gebruiken. (Let op dat je een hoofdletter B gebruikt!)

Onder de Text-widget, maar voordat de app wordt weergegeven, voeg je een regel code toe om een knop te maken.



Figuur 1 De tekst in een venster weergeven.



Figuur 2 Je hebt nu een knop.

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

Je code zou er nu uit moeten zien als **spy1.py**. Voer hem uit en er zal geen knop verschijnen, maar je zult een fout zien in het Shell-venster:

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

Dit komt omdat `choose_name` de naam is van een commando dat wordt uitgevoerd als de knop wordt ingedrukt. Aan de meeste GUI-componenten kan een commando gekoppeld worden. Voor een knop betekent het toevoegen van een commando "als de knop wordt ingedrukt, voer dan dit commando uit". Een GUI-programma werkt anders dan andere Python programma's die je misschien hebt geschreven omdat de volgorde waarin de commando's in het programma worden uitgevoerd volledig afhangt van de volgorde waarin de gebruiker op de knoppen drukt, de schuifregelaars beweegt, de vakjes aanvinkt, of interactie heeft met welke andere widgets je ook maar gebruikt. Het eigenlijke commando is bijna altijd de naam van een functie die moet worden uitgevoerd.

Maak een functie

Laten we de functie `choose_name` schrijven zodat je knop iets kan doen als hij wordt ingedrukt.

Zoek in je programma de sectie Functions op. Dit is waar alle functies moeten komen die aan GUI-widgets worden gekoppeld, om ze gescheiden te houden van de code voor het tonen van de widget. Voeg deze code toe in de sectie Functions:

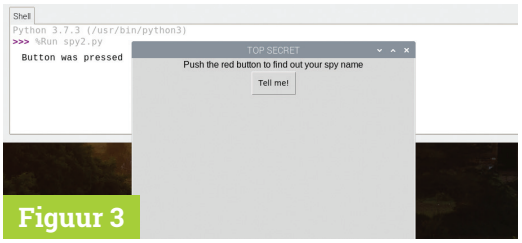
```
def choose_name():
    print("Button was pressed")
```

Je code zou er nu uit moeten zien als **spy2.py**. De knop zal nu verschijnen (**Figuur 2**). Als je op de knop drukt, lijkt het alsof er niets gebeurt, maar als je in je Shell- of uitvoervenster kijkt, zul je zien dat daar

wat tekst is verschenen (**Figuur 3**).

Het is slim om je functie eerst wat testtekst te laten weergeven, zodat je zeker weet dat de knop zijn opdrachtfunctie correct uitvoert wanneer hij wordt ingedrukt. Vervolgens kun je de `print` statement vervangen door de eigenlijke code voor de taak die je de knop wilt laten uitvoeren.

Typ in je `choose_name` functie een hekje (`#`) voor de regel code die 'Button was pressed' print. Programmeurs noemen dit 'uitcommentariëren', hiermee heb je de computer verteld om deze regel code te behandelen alsof het een commentaar is, of anders gezegd, je hebt de computer geïnstrueerd om het te negeren. Het voordeel van het uitcommentariëren van een regel code in plaats van die gewoon te verwijderen, is dat als je die code ooit weer wilt gebruiken, je hem gemakkelijk weer deel kunt laten uitmaken van je programma door het #-symbool te verwijderen.



Figuur 3

▲ **Figuur 3** Tekst wordt uitgevoerd naar het Shell-venster.

spy1.py

DOWNLOAD DE VOLLEDIGE CODE: magpi.cc/guizerocode

Taal: **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

Taal: **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()
```

Grote rode knop

Op dit moment is je knop niet groot of rood! Je hebt in deel één van deze tutorialserie eigenschappen gebruikt om het uiterlijk van je tekst op de 'Wanted' poster te veranderen. Kun je nu de eigenschappen van de PushButton widget gebruiken om de achtergrondkleur en de tekstgrootte te veranderen?

Let op: op macOS is het misschien niet mogelijk is om de kleur van een knop te wijzigen, omdat sommige versies van het besturingssysteem dit niet toestaan, maar je zou nog wel de tekstgrootte moeten kunnen wijzigen.

Namen toevoegen

Typ op een nieuwe regel een lijst met voornamen. Je kunt zelf bepalen welke namen er in je lijst komen en het mogen er zoveel zijn als je wilt, maar zorg ervoor dat elke naam tussen aanhalingstekens staat, en dat de namen van elkaar gescheiden zijn door een komma. Een verzameling letters, cijfers en/of leestekens tussen aanhalingstekens wordt een *string* genoemd, dus we zeggen dat elke naam een string moet zijn.

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

Voeg nu ook een lijst van achternamen toe:

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

Nu moet je een manier toevoegen om een willekeurige naam te kiezen uit elke lijst om je spionnennaam te vormen. Je eerste taak is het toevoegen van een nieuwe import-regel in je sectie imports:

```
from random import choice
```

Dit vertelt het programma dat je een functie wilt gebruiken, choice genaamd, die een willekeurig item uit een lijst kiest. Iemand anders heeft de code geschreven die dit voor je doet, en het is meegeleverd met Python zodat je het kunt gebruiken.

In je code voor de `choose_name` functie, net onder je lijsten met namen, voeg je een regel code toe om de voornaam van je spion te kiezen, en daar vervolgens een achternaam aan te hangen, met een spatie ertussen. Dit wordt 'concatenation' (aaneenschakeling) genoemd, dat betekent 'voeg twee strings samen' en het symbool in Python voor concatenation is een plus (+).

Figuur 4

▲ **Figuur 4** Het uitvoeren van een spionnennaam.

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

Je code zou nu moeten lijken op **spy3.py**. Sla hem op en voer hem uit. Wanneer je op de knop drukt, zou er een willekeurig gegenereerde spionnennaam moeten verschijnen in je console of Shell, op dezelfde plaats waar eerst de melding 'Button was pressed' verscheen (**Figuur 4**).

Zet de naam in de GUI

Dat is goed, maar zou het niet mooier zijn als de spionnennaam in de GUI zou verschijnen? Laten we een andere Text-widget maken en die gebruiken om de naam van de spion weer te geven.

Voeg In de sectie widgets een nieuwe Tekst-widget toe die de spionnennaam zal weergeven:

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

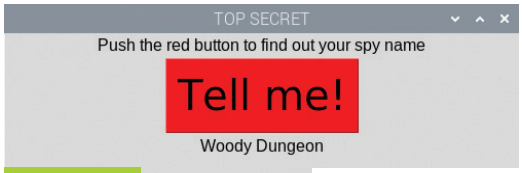
Wanneer je de widget maakt, wil je niet dat die iets van tekst laat zien omdat de persoon nog niet op de knop heeft gedrukt, dus kun je de tekst op `""` zetten, wat een 'lege string' wordt genoemd en niets weergeeft. In je `choose_name` functie, zet je een hekje vóór de regel code waarin je de spionnennaam print.

Voeg nu een nieuwe regel code toe aan het einde van de functie om de waarde van de Text-widget name in te stellen op de `spy_name` die je zojuist hebt gemaakt. Dit zal ervoor zorgen dat de Text-widget zichzelf update en de naam weergeeft.

```
name.value = spy_name
```

Je uiteindelijke code zou moeten zijn zoals in **03-spy-name-chooser.py**. Voer die uit en druk op de knop om je spionnennaam met trots te zien verschijnen op de GUI (**Figuur 5**).

Je kunt nogmaals op de knop drukken als je de naam die je krijgt niet leuk vindt, en het programma zal een willekeurige andere naam voor je genereren.



Figuur 5

▲ **Figuur 5** De voltooide spionnennaamkiezer.

spy3.py

Taal: **Python 3**

```
# Imports ---------------
from guizero import App, Text, PushButton
from random import choice

# Functions -------------
def choose_name():
    #print("Button was pressed")
    first_names = ["Barbara", "Woody", "Tiberius", "Smokey", "Jennifer", "Ruby"]
    last_names = ["Spindleshanks", "Mysterioso", "Dungeon", "Catseye", "Darkmeyer", "Flamingobreath"]
    spy_name = choice(first_names) + " " + choice(last_names)
    print(spy_name)

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

# Widgets ---------------
title = Text(app, "Push the red button to find out your spy name")
button = PushButton(app, choose_name, text="Tell me!")
button.bg = "red"
button.text_size = 30

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

03-spy-name-chooser.py

Taal: **Python 3**

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

from guizero import App, Text, PushButton
from random import choice

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

def choose_name():
    #print("Button was pressed")
    first_names = ["Barbara", "Woody", "Tiberius", "Smokey", "Jennifer", "Ruby"]
    last_names = ["Spindleshanks", "Mysterioso", "Dungeon", "Catseye", "Darkmeyer", "Flamingobreath"]
    spy_name = choice(first_names) + " " + choice(last_names)
    #print(spy_name)
    name.value = spy_name

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

app = App("TOP SECRET")

# Widgets ---------------

title = Text(app, "Push the red button to find out your spy name")
button = PushButton(app, choose_name, text="Tell me!")
button.bg = "red"
button.text_size = 30
name = Text(app, text="")

# Display ---------------

app.display()
```

Deel 03

GUI's maken met Python: Meme-generator

Ontwerp een GUI-applicatie die memes maakt.

Laten we de lessen die je geleerd hebt uit de vorige afleveringen gebruiken om een GUI te ontwerpen die memes maakt. Je voert de tekst en de naam van de afbeelding in en je GUI zal ze combineren tot je eigen meme met behulp van de Drawing widget.

Begin met het maken van een simpele GUI met twee tekstboxen voor de bovenste en onderste tekst. Dit is waar je de tekst zult invoeren die over je afbeelding heen zal worden gezet om je meme te maken. Voeg deze regel toe om de benodigde widgets te importeren.

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

Voeg dan deze code toe voor de app:

```
app = App("meme")

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

app.display()
```

De meme wordt gemaakt op een Drawing-widget die de afbeelding en de tekst zal bevatten.

Maak een meme

Voeg hem toe aan de GUI door het invoegen van onderstaande code net boven de regel `app.display()`. De hoogte en breedte van de Drawing-widget moeten worden ingesteld op 'fill' om de rest van de GUI te vullen.

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

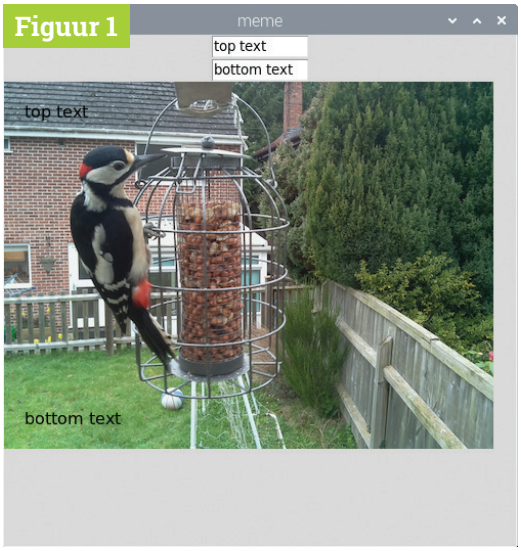
De meme zal worden gemaakt wanneer de tekst in de bovenste en onderste tekstvakken verandert. Hiervoor moeten we een functie maken die de meme samenstelt.

De functie moet de tekening leegmaken, een afbeelding maken (wij gebruiken een foto van een specht, maar je kunt elke foto gebruiken die je wilt) en de tekst boven en onder de afbeelding invoegen. Weet je nog dat je `name.value` gebruikte om de waarde van de Text-widget in te stellen met de naam van de spion in Deel 2 van deze serie? Je kunt ook de eigenschap value gebruiken om de waarde van een Text widget op te halen, dus in dit geval betekent `top_text.value` 'haal de waarde op die is getypt in de box top_text'.

```
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)
```

De eerste twee getallen in `meme.image(0, 0)` en `meme.text(20, 20)` zijn de x, y coördinaten van waar de afbeelding en de tekst getekend moeten worden. De afbeelding wordt getekend op positie 0, 0, dat is de linkerbovenhoek, zodat de afbeelding de hele tekening bedekt.

Roep tenslotte je `draw_meme` functie op vlak voordat je de app weergeeft. Voeg deze code in net



▲ **Figuur 1** Meme met niet opgemaakte tekst.

voor de regel `app.display`:

```
draw_meme()
```

Je code zou er nu uit moeten zien als **meme1.py**. Als je je app uitvoert (**Figuur 1**) en probeert de tekst boven en onder te veranderen, zul je merken dat deze niet wordt bijgewerkt in de meme. Om dit werkend te krijgen, moet je je programma veranderen om de *draw_meme* functie aan te roepen wanneer de tekst verandert, door een commando toe te voegen aan de twee TextBox widgets in de app.

> Werk je meme bij door de tekst boven en onder te veranderen

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

Je code zou er nu zo uit moeten zien in **meme2.py**. Voer het uit en pas je meme aan door de bovenste en onderste tekst te veranderen.

Je kunt het uiterlijk van je meme veranderen door de parameters `color`, `size` en `font` van de tekst te veranderen. Bijvoorbeeld:

meme1.py

DOWNLOAD DE VOLLEDIGE CODE:
magpi.cc/guizerocode

Taal: **Python 3**

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

from guizero import App, TextBox, Drawing


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

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)


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

app = App("meme")

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

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

draw_meme()

app.display()
```

meme2.py

Taal: **Python 3**

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

from guizero import App, TextBox, Drawing


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

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)


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

app = App("meme")

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

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

draw_meme()

app.display()
```

Tip

Deze regels code begonnen erg lang te worden, dus hebben we ze opgesplitst over een aantal regels om de leesbaarheid te vergroten. Het heeft geen invloed op wat het programma doet, alleen hoe het eruit ziet.

```
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",
    )
```

Je code zou er nu uit moeten zien als **meme3.py**. Probeer verschillende stijlen tot je iets vindt dat je bevalt (**Figuur 2**).

meme3.py

Taal: **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()
```

Pas je meme-generator aan

Voor een echt interactieve meme-generator moet de gebruiker het lettertype, de grootte en de kleur zelf kunnen instellen. Je kunt extra widgets op de GUI zetten om hen in staat te stellen dit te doen.

Het aantal beschikbare opties voor de kleur en het lettertype zijn beperkt, dus je zou hiervoor een drop-down lijst, ook bekend als een Combo box, kunnen gebruiken. De grootte kun je instellen met een Slider widget.

Pas eerst je import statement aan om de Combo en Slider widgets op te nemen.

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

Nadat je je TextBox widgets hebt gemaakt voor de bovenste en onderste tekst, maak je een nieuwe Combo widget zodat de gebruiker een kleur kan kiezen.

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

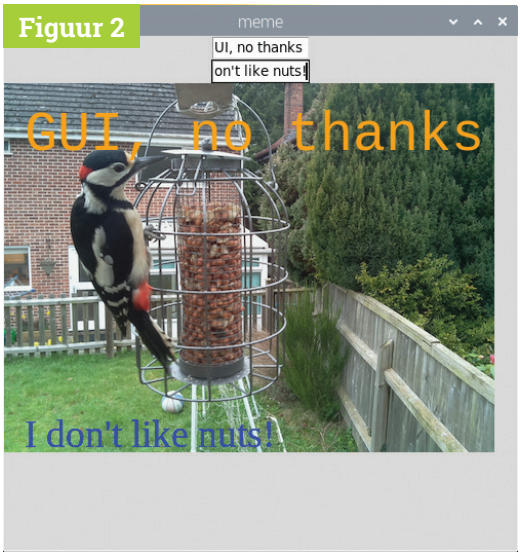
> De opties worden getoond in de volgorde waarin je ze in de lijst zet

De parameter `options` bepaalt welke kleuren de gebruiker kan kiezen uit de Combo. Elke kleur is een element in een lijst. Je kunt alle andere kleuren die je wilt aan de lijst toevoegen.

De opties worden getoond in de volgorde waarin je ze in de lijst zet. De eerste optie is de standaard, die als eerste wordt weergegeven. Als je een andere optie als standaard wilt hebben, kun je dat doen met de parameter `selected`, bijvoorbeeld `"blue"`.

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

Figuur 2



Figuur 2 Lettertypen en kleuren wijzigen

Nu kan je gebruiker een kleur kiezen. Vervolgens moet je de `draw_meme` functie zo aanpassen dat de waarde van Combo wordt gebruikt bij het maken van de tekst in je meme. Bijvoorbeeld:

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

Doe hetzelfde voor het blok code voor de onderste tekst. Je programma zou nu moeten lijken op **meme4.py**.

Voeg volgens de bovenstaande stappen een tweede Combo toe aan je applicatie zodat de gebruiker een lettertype kan kiezen uit deze lijst met opties: [`"times new roman"`, `"verdana"`, `"courier"`, `"impact"`]. Vergeet niet om de `draw_meme` functie aan te passen zodat de waarde van `font` gebruikt wordt bij het toevoegen van de tekst.

Maak een nieuwe Slider widget om de grootte van de tekst in te stellen die je gebruiker wil.

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

Het bereik van de schuif wordt ingesteld met de parameters `start` en `end`. In dit voorbeeld zal de kleinste beschikbare tekst dus 20 zijn en de grootste 40.

meme4.py

Taal: **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

De Drawing widget is zeer veelzijdig en kan worden gebruikt om veel verschillende vormen, patronen en afbeeldingen weer te geven. Voor meer informatie over de Drawing widget, zie Appendix C van het boek (magpi.cc/pythongui), of kijk eens naar de online documentatie: **lawsie.github.io/guizero/drawing**.

04-meme-generator.py

Taal: **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()
```

Figuur 3

▲ **Figuur 3** De voltooide meme-generator.

Pas de `draw_meme` functie aan om de waarde van je schuif te gebruiken bij het maken van de tekst van de meme.

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

Je code zou nu moeten lijken op die in **04-meme-generator.py**. Probeer het uit te voeren en je zou iets als **Figuur 3** moeten zien.

Kun je de GUI zo veranderen dat de naam van het afbeeldingsbestand in een TextBox kan worden ingevoerd of misschien uit een lijst in een Combo kan worden gekozen? Hiermee zou je applicatie ook in staat zijn om memes met verschillende afbeeldingen te genereren.

Deel 04

GUI's maken met Python: 's Werelds slechtste GUI

Leer een goede GUI ontwerpen door het eerst helemaal fout te doen!

Het is tijd om echt aan de slag te gaan met je GUI's en te experimenteren met verschillende widgets, kleuren, lettertypes, en features. Zoals bij de meeste experimenten, zul je het waarschijnlijk niet meteen de eerste keer goed doen! In feite, ga je de verkeerde manier ontdekken om je GUI te maken.

Moeilijk te lezen

De juiste keuze van GUI-kleur en -lettertype zijn belangrijk. Het is belangrijk dat het contrast tussen achtergrond en tekstkleur er voor zorgt dat je GUI makkelijk leesbaar is. Wat je niet moet doen is twee kleuren gebruiken die erg op elkaar lijken.
Importeer de widgets bovenaan in de code:

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

Maak een app met een titel:

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

app.display()
```

Experimenteer door de kleuren, het lettertype en de tekstgrootte te veranderen (zie listing **worst1.py**). Onze keuzes zijn niet de beste!

```
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")
```

Het is belangrijk dat tekst op een GUI ook lang genoeg blijft staan om gelezen te worden. Hij moet zeker niet verdwijnen of gaan knipperen.
Alle widgets in guizero kunnen onzichtbaar (of weer zichtbaar) gemaakt worden met de functies `hide()` en `show()`. Door de functie `repeat` in guizero te gebruiken om elke seconde een functie uit te voeren, kun je je tekst laten verdwijnen en verschijnen, zodat hij lijkt te knipperen.
Maak een functie die de tekst verbergt als hij zichtbaar is en toont als hij dat niet is:

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

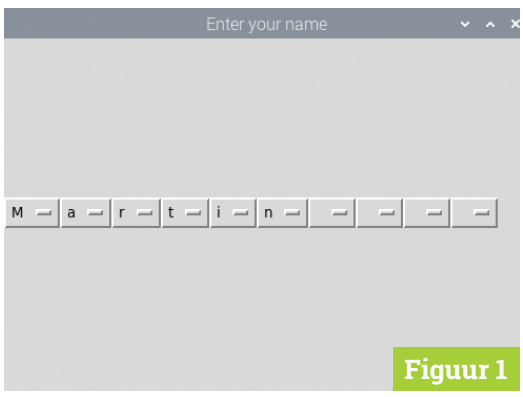
Voordat de app wordt weergegeven, gebruik je `repeat` om de functie `flash_text` elke 1000 milliseconden (1 seconde) te laten lopen.

```
app.repeat(1000, flash_text)

app.display()
```

Je code zou er nu uit moeten zien als **worst2.py**. Test je app: de titeltekst moet knipperen, en elke seconde verschijnen en verdwijnen.

De verkeerde widget

Het gebruik van een geschikte widget kan het verschil betekenen tussen een geweldige GUI en een die compleet onbruikbaar is.
Welke widget zou je gebruiken om een datum in te voeren? Een TekstBox? Meerdere Combo's? Een TextBox zou flexibeler zijn maar zou toetsing en formattering vereisen. Meervoudige Combo's voor



Figuur 1 Combo's om letters te kiezen.

jaar, maand en dag hoeven niet getoetst te worden, maar zijn langzamer in gebruik.

Het gebruik van een Slider om een datum en tijd in te stellen (**Figuur 2**), zoals in het codevoorbeeld **worst3.py**, is echter niet zo'n goed idee.

De Slider widget geeft een getal tussen 0 en 999.999.999 terug. Dit is het aantal seconden sinds 1 januari 1970. De functie `ctime()` wordt gebruikt om dit getal om te zetten in een datum en tijd.

Tekst-input krijgen van je gebruiker is eenvoudig: een TextBox of een multi-line TextBox zou aan al je behoeften moeten voldoen. Maar is dit te simpel? Vergt het te veel typwerk?

Hoe zit het met de gebruiker die alleen de muis wil gebruiken? Misschien is een reeks Combo's met in elk alle letters van het alfabet beter (**Figuur 1**)? Begin met het importeren van de guizero widgets en `ascii_letters`.

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

`ascii_letters` is een lijst met alle 'afdrukbare' ASCII-tekens die je kunt gebruiken als de opties voor de Combo.

Maak een enkele Combo die alle letters bevat en laat de app weergeven.

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

app.display()
```

Je programma zou nu moeten lijken op **worst4.py**. Als je het uitvoert, zie je een enkele Combo die alle letters bevat plus een spatie en is uitgelijnd aan de linkerkant van het venster.

Om een regel letters bij elkaar te krijgen, zou je een heleboel Combo widgets aan je app kunnen toevoegen, bijv:

```
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")
```

Door elk Combo widget links uit te lijnen, worden de widgets naast elkaar tegen de linkerrand weergegeven.

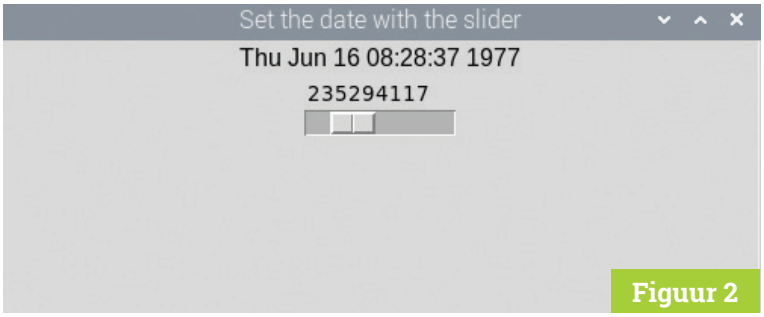
Als alternatief zou je een `for`-lus kunnen gebruiken, een lijst van letters maken, en elke letter aan de lijst toevoegen, zoals getoond in **worst5.py**.

worst1.py

DOWNLOAD DE VOLLEDIGE CODE: magpi.cc/guizerocode

Taal: **Python 3**

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

from guizero import App, Text


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

app = App("it's all gone wrong", bg="dark green")

title = Text(app, text="Hard to read", size="14", font="Comic
Sans", color="green")

app.display()
```



▲ **Figuur 2** Een schuifknop om datum en tijd in te stellen.

worst2.py

Taal: **Python 3**

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

from guizero import App, Text


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

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


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

app = App("it's all gone wrong", bg="dark green")

title = Text(app, text="Hard to read", size="14", font="Comic
Sans", color="green")

app.repeat(1000, flash_text)

app.display()
```

worst3.py

Taal: **Python 3**

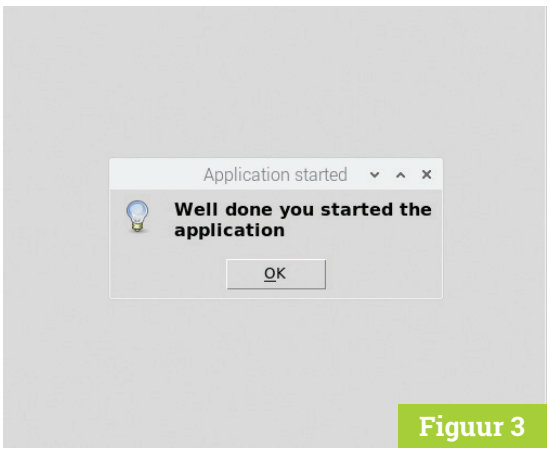
```
001.  # Imports ---------------
002.
003.  from guizero import App, Slider, Text
004.  from time import ctime
005.
006.
007.  # Functions -------------
008.
009.  def update_date():
010.      the_date.value = ctime(date_slider.value)
011.
012.
013.  # App -------------------
014.
015.  app = App("Set 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

Taal: **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()
```

Figuur 3
Zinloze pop-up.



Figuur 3

Window widget

Pop-up boxes kunnen worden gebruikt om gebruikers vragen te stellen, maar ze zijn erg eenvoudig.

Als je extra informatie wilt tonen of aanvullende gegevens wilt vragen, zou je de widget Window kunnen gebruiken om meerdere vensters te maken.

Window wordt op een vergelijkbare manier gebruikt als App en heeft veel van dezelfde functies.

```
from guizero import App, Window

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

app.display()
```

Je kunt bepalen of een Window op het scherm te zien is met de methodes **show()** en **hide()**.

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

Je kunt een app maken om te wachten tot een venster is gesloten nadat het is getoond, door True door te geven aan de wait parameter van show. Bijvoorbeeld

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

Je kunt meer te weten komen over het gebruik van meerdere vensters in de documentatie van guizero: **lawsie.github.io/guizero/multiple_windows**.

Probeer beide concepten en kijk welke je voorkeur heeft. De `for`-lus is flexibeler omdat je daarmee zoveel letters kunt maken als je wilt.

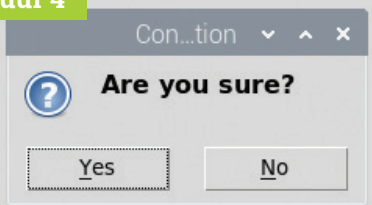
Pop-ups

Geen verschrikkelijke GUI zou compleet zijn zonder een pop-up box. guizero bevat een aantal pop-up boxen, die je kunt gebruiken om gebruikers iets belangrijks te laten weten of om nuttige informatie te verzamelen. Maar je kunt ze ook gebruiken om gebruikers te ergeren en te irriteren!

Maak eerst een applicatie die aan het begin een zinloze box laat verschijnen om te laten weten dat de applicatie gestart is.

```
from guizero import App

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

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

app.display()
```

Figuur 4



▲ **Figuur 4** Ja, we weten het zeker!

Als je de applicatie uitvoert, zul je zien dat er een 'info'-box verschijnt (**Figuur 3**). De eerste parameter die aan `info` wordt doorgegeven is de titel van het venster; de tweede parameter is de boodschap.

Je kunt de stijl van deze eenvoudige pop-up veranderen door `warn` of `error` te gebruiken in plaats van `info`.

Pop-up boxes kunnen ook worden gebruikt om informatie van de gebruiker te krijgen. De eenvoudigste is een `yesno` die de gebruiker een vraag stelt en een True of False antwoord krijgt. Dit is handig als je een gebruiker wilt laten bevestigen voordat hij iets doet, zoals het verwijderen van een bestand. Maar misschien niet elke keer als ze op een knop drukken! Importeer de PushButton widget in je applicatie:

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

Maak een functie die de `yesno` pop-up gebruikt om een bevestiging te vragen.

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

Voeg de knop toe aan je GUI die de functie aanroept als hij wordt ingedrukt.

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

Je code zou nu moeten lijken op **05-worlds-worst-gui.py**. Wanneer je het programma uitvoert en op de knop klikt, zul je een pop-up zien die je vraagt om te bevestigen met Yes of No (**Figuur 4**).
Je kunt meer te weten komen over de pop-up boxen in guizero op lawsie.github.io/guizero/alerts.
Misschien kun je al deze 'features' kunnen combineren tot één geweldige GUI?

worst5.py

Taal: **Python 3**

```
001.  # Imports ---------------
002.
003.  from guizero import App, Combo
004.  from string import ascii_letters
005.
006.
007.  # App -------------------
008.
009.  app = App("Enter your name")
010.
011.  name_letters = []
012.  for count in range(10):
013.      a_letter = Combo(app, options=" " + ascii_letters,
014.  align="left")
015.      name_letters.append(a_letter)
016.
017.  app.display()
```

05-worlds-worst-gui.py

Taal: **Python 3**

```
001.  from guizero import App, PushButton
002.
003.  def are_you_sure():
004.      if app.yesno("Confirmation", "Are you sure?"):
005.          app.info("Thanks", "Button pressed")
006.      else:
007.          app.error("Ok", "Cancelling")
008.
009.  app = App(title="pointless pop-ups")
010.
011.  button = PushButton(app, command=are_you_sure)
012.
013.  app.info("Application started", "Well done you started the
014.  application")
015.
016.  app.display()
```

Deel 05

GUI's maken met Python: Boter-kaas-en-eieren

Gebruik je GUI om een eenvoudig spel te besturen.

Nu je geleerd hebt hoe je een basis-GUI maakt, is het tijd om wat meer programmeerlogica achter de schermen toe te voegen om met je GUI een spelletje boter-kaas-en-eieren ('Tic tac toe' in het Engels) te besturen.

Maak een nieuw bestand met de volgende code:

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

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

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

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

app.display()
```

Opzetten van het bord

We beginnen met het maken van de widgets die het speelbord zullen vormen. Een traditioneel boter-kaas-en-eieren bord ziet eruit zoals in **Figuur 1**.

Je gebruikt knoppen om elke positie op het bord weer te geven, zodat de speler op een van de knoppen kan klikken om aan te geven waar hij heen wil. Om de knoppen in een raster te kunnen plaatsen, maken we een nieuw type guizero-widget, een Box.

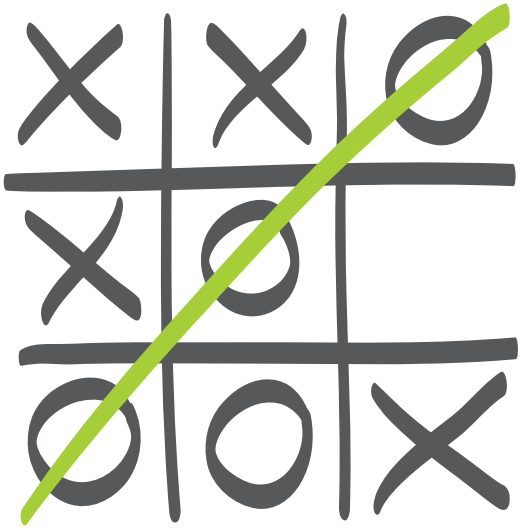
Een Box is een container-widget. Dit betekent dat hij gebruikt wordt om er andere widgets in te stoppen en te groeperen. Voeg hem toe aan de sectie imports bovenaan je code:

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

Stel de box in op een rasterlay-out en voeg hem toe aan je app - voor de `app.display()` regel, zoals met alle widgets.

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

Figuur 1



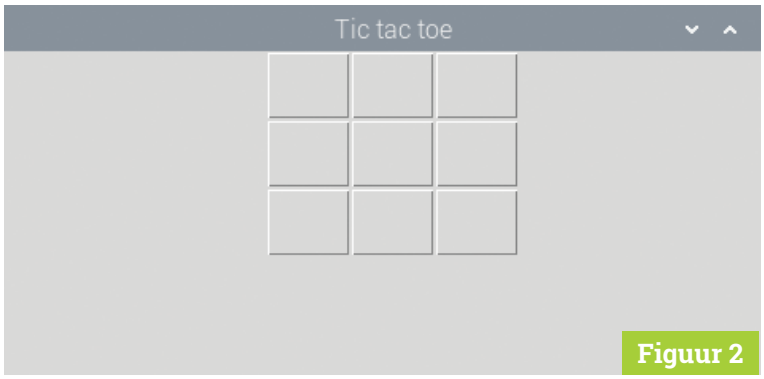
▲ **Figuur 1** Een typisch spelletje boter-kaas-en-eieren.

Als je je programma op dit moment uitvoert, zul je niets op het scherm zien omdat de Box zelf onzichtbaar is.

We maken nu de knoppen die er in moeten. Je hebt in totaal negen knoppen nodig. Dus in plaats van ze allemaal apart te maken, kun je een geneste lus gebruiken om ze automatisch te genereren en coördinaten te geven. Voeg eerst PushButton toe aan je lijst van te importeren widgets en voeg dan deze code toe onmiddellijk na de code voor het bord dat je net hebt gemaakt.

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

Merk op dat er twee lusvariabelen zijn: `x` van 0 tot 2 en `y` van 0 tot 2. Terwijl we itereren



Figuur 2 Een raster van negen knoppen om boter-kaas-en-eieren te spelen.

tictactoe1.py

DOWNLOAD DE VOLLEDIGE CODE: magpi.cc/guizerocode

Taal: **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

Taal: **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()
```

en knoppen genereren, zal elke knop worden toegevoegd aan het bord, dat wil zeggen de Box-container die je eerder hebt gemaakt. De knop krijgt de rastercoördinaten `x,y,` wat betekent dat elke knop netjes op een raster wordt geplaatst, elk op een andere positie!

Je code zou er nu uit moeten zien als **tictactoe1.py**. Het resultaat als je het uitvoert, is te zien in **Figuur 2**.

Onderliggende datastructuur

Het zal je opvallen dat wanneer je de knoppen aanmaakt via een lus, je automatisch negen knoppen aanmaakt en dat ze allemaal `button` heten. Hoe kun je in het programma naar elk van deze knoppen verwijzen?

Het antwoord is dat je een onderliggende datastructuur nodig hebt met een verwijzing naar elke knop. Hiervoor zullen we een tweedimensionale lijst gebruiken.

Laten we een functie maken die we kunnen aanroepen om het bord leeg te maken. Het is een goed idee om dit in een functie te doen, zodat je de code kunt hergebruiken nadat het spel gespeeld is om het bord te resetten en de speler een nieuw spel te laten beginnen.

Voeg in de sectie functions een nieuwe functie toe met de naam `clear_boar`.

```
def clear_board():
```

Je eerste taak in deze functie is om de datastructuur voor het bord te initialiseren. Laten we aannemen dat je op dit punt nog geen knoppen hebt gemaakt. Dus je kunt elke positie op het bord initialiseren als `None` - het element in de lijst bestaat nu, maar heeft nog geen waarde. Voeg de volgende regel, ingesprongen, toe aan je functie.

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

Verplaats vervolgens de geneste lus-code van je app-sectie naar de `clear_board`-functie. Zorg ervoor dat de inspringing correct is.

Voeg binnen de binnenste (y)-lus een regel code toe om een verwijzing naar elke knop op te slaan op zijn x,y-coördinaatpositie binnen de tweedimensionale lijst, zodat je er later naar kunt verwijzen.

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

Tot slot, nadat de lussen zijn afgelopen, zorg je met return dat de functie het new_board teruggeeft dat je zojuist gemaakt hebt. Je functie zou er als volgt uit moeten zien:

Het spel resetten

In het begin heb je een functie geschreven genaamd **clear_board**. Dit leek op dat moment misschien onnodig, maar in feite dacht het vooruit tot wanneer het spel afgelopen was. Aangezien boter-kaas-en-eieren een vrij kort spel is, is het waarschijnlijk dat iemand meer dan één spel achter elkaar wil spelen. Kun je een reset-knop toevoegen aan je spel, die alleen verschijnt als iemand het spel heeft gewonnen, of als het gelijkspel is geworden? De knop moet de functie **clear_board** aanroepen en de variabele **turn** resetten, evenals de melding wiens beurt het is.

Hint: je zult de documentatie van guizero moeten raadplegen om uit te vinden hoe je widgets kunt verbergen en tonen, zodat je knop niet de hele tijd zichtbaar is tijdens het spel.

Hint: maak een nieuwe functie die zorgt voor alles wat je moet doen om het spel te resetten, en roep die functie aan wanneer de reset-knop wordt ingedrukt. Vergeet niet dat je in je functie enkele variabelen als global moet specificeren.

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

In de app-sectie initialiseer je nu een lijst genaamd `board_squares` en stel je deze in om de nieuwe functie aan te roepen die je zojuist hebt gemaakt.

```
board_squares = clear_board()
```

Deze variabele krijgt de waarde van `new_board` dat je in de functie hebt aangemaakt, en dat moet een leeg bord zijn met negen knoppen. Zorg ervoor dat je deze variabele aanmaakt na de code voor het maken van de Box, anders zul je proberen knoppen toe te voegen aan een container die nog niet bestaat.

Je code zal er nu uit zien als **tictactoe2.py**. Sla het programma op en voer het uit en je zou een identiek resultaat moeten zien als op het einde van de vorige stap, maar nu heb je een verborgen datastructuur in een tweedimensionale lijst om naar de knoppen te verwijzen en ze te manipuleren.

Als je wilt zien hoe je 2D-lijst er uit ziet, zou je een print commando kunnen toevoegen om de lijst `board_squares` af te drukken: `print(board_squares)`.

tictactoe3.py

▶ Taal: **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()
```

Je zou dan negen keer `[PushButton] object with text ""` moeten zien in de shell.

Laat de knoppen werken

Op dit moment doen de knoppen niets als je er op drukt. Laten we een functie maken om aan de knop te koppelen, zodat wanneer hij wordt ingedrukt, de knop een X of een O laat zien, afhankelijk van welke speler hem heeft gekozen.

Maak eerst een variabele in de sectie variables om vast te leggen wiens beurt het is. Je kunt met elke speler beginnen, maar wij kiezen voor X.

```
turn = "X"
```

Dit betekent nu dat je op de GUI moet laten zien wiens beurt het is (**Figuur 3**), zodat de spelers niet in de war raken. Voeg Text toe aan je lijst van te importeren widgets:

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

Voeg dan een nieuwe Text widget toe in de app sectie om de beurt weer te geven.

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

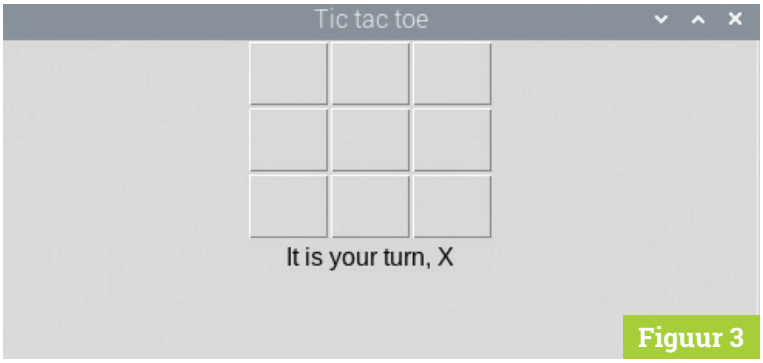
tictactoe4.py

Taal: **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()
```

Ga naar de sectie functions en maak een nieuwe functie genaamd `choose_square`.

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



Figuur 3 Laat je spelers weten wiens beurt het is.

Figuur 3

Je ziet dat deze functie twee argumenten accepteert – x en y. Dit is om te weten op welk vakje op het bord is geklikt.

Voeg de volgende code (ingesprongen) toe in de functie om de tekst in de knop waarop geklikt is in te stellen op het symbool van de huidige speler, en vervolgens de knop uit te schakelen zodat er niet meer op geklikt kan worden.

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

Tot slot, verbind deze functie met de knop. Zoek deze regel code in je functie `clear_board`:

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

Wijzig hem zodat hij eruit ziet als de regel hieronder:

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

Je hebt hier twee dingen toegevoegd. Ten eerste, je hangt er een commando aan, net als eerst. Wanneer de knop wordt ingedrukt, zal de functie met deze naam worden aangeroepen. Ten tweede geef je ook argumenten aan deze functie, namelijk de coördinaten x en y van de knop die werd ingedrukt, zodat je die knop weer kunt vinden in de lijst.

Je programma zou er nu uit moeten zien als **tictactoe3.py**. Sla het op en start het. Je kunt nu op een knop klikken en die zal veranderen in een X. Helaas is het in deze versie van het spel permanent de beurt van X!

Afwisselen tussen spelers

Als een speler eenmaal aan de beurt is, moet de variabele `turn` veranderen in die van de andere speler. Hier is een functie die van X naar O wisselt en vice versa.

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

Voeg de code toe in de sectie functions. Let op de eerste regel in de functie: `global turn`. Je moet dit specificeren zodat je de globale versie van de

tictactoe5.py

Taal: **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"]:
066.          winner = board_squares[0][2]
067.
068.      if winner is not None:
069.          message.value = winner.text + " wins!"
070.
071.  # Variables -------------
072.  turn = "X"
073.
074.  # App -------------------
075.  app = App("Tic tac toe")
076.
077.  board = Box(app, layout="grid")
078.  board_squares = clear_board()
079.  message = Text(app, text="It is your turn, " + turn)
080.
081.  app.display()
```

variabele `turn` mag wijzigen, d.w.z. die ene die je al hebt aangemaakt. Als je dit niet specificeert, maakt Python een lokale variabele genaamd `turn` en wijzigt die in plaats daarvan, maar die wijziging wordt niet opgeslagen als de functie wordt afgesloten.

Je moet er ook voor zorgen dat de Text widget de beurt van de huidige speler nauwkeurig weergeeft. Na het if/else statement in de functie `toggle_player` pas je de melding als volgt aan:

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

Ga terug naar de functie `choose_square` en roep de functie `toggle_player` aan - met `toggle_player()` - zodra je de tekst hebt ingesteld en de knop hebt uitgeschakeld. Je code zou nu moeten lijken op **tictactoe4.py**. Bewaar en test het programma opnieuw en je zou moeten merken dat je op vierkantjes kunt klikken en dat ze om beurten met X of O aangeduid worden.

Hebben we een winnaar?

Tot slot moet je een functie schrijven die controleert of er een rij, kolom of diagonaal van drie X'en of O's is, en zo ja, de winnaar van het spel meldt.

Hoewel het erg onelegant lijkt, de gemakkelijkste manier om te controleren of iemand gewonnen heeft, is het hard coderen van de controles voor elke verticale, horizontale en diagonale lijn apart.

De volgende code is voor één verticale lijn, één horizontale lijn, en één diagonale - kun jij de rest toevoegen?

Globale variabelen

Het is misschien een slecht idee om globale variabelen te gebruiken, want als je veel functies in een groot programma hebt, kan het erg verwarrend worden welke code de waarde van een variabele wijzigt en wanneer. In een klein programma als dit, is het niet al te moeilijk om het bij te houden.

Onthoud dat het mogelijk is om de waarde van een globale variabele te lezen en te gebruiken vanuit een functie zonder deze als global te declareren, maar om de waarde te wijzigen moet je dit expliciet declareren. De functies in dit programma (en de meeste GUI-programma's in deze tutorial serie) wijzigen eigenlijk de waarden van je widgets als globale variabelen. Bijvoorbeeld, wanneer iemand het spel wint, stel je de waarde van de melding in om weer te geven wie er gewonnen heeft:

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

In dit voorbeeld is **message** een globale variabele. Dus hoe kunnen we zijn waarde wijzigen zonder hem als global te declareren? Het antwoord is dat we gebruik maken van een eigenschap van de widget message, de eigenschap value. In wezen is wat deze code zegt: "Hé Python, ken je die widget daar die message heet? Nou, zou je zijn value-eigenschap kunnen wijzigen alsjeblieft?" Python staat wijziging toe via objecteigenschappen in de global scope, maar hij staat niet toe dat je rechtstreeks de waarde van een variabele wijzigt zonder deze als global te declareren.

Python 3 for Science and Engineering Applications

Als je de basis van Python onder de knie hebt en de taal verder wilt uitdiepen, dan is dit boek iets voor jou. Aan de hand van concrete voorbeelden die in verschillende toepassingen worden gebruikt, illustreert het boek vele aspecten van het programmeren (bv. algoritmen, recursie, datastructuren) en helpt het bij probleemoplossende strategieën. **www.elektor.nl/python-3-for-science-and-engineering-applications**

```
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]

    # 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]
```

Merk op dat de functie begint met het creëren van een booleaanse variabele genaamd winner. Als tegen de tijd dat het lange if/elif-statement is uitgevoerd, de waarde van deze variabele True is, weet je dat iemand het spel heeft gewonnen.
Na het toevoegen van de resterende checks op de winnende lijn, voeg je aan het eind van de functie wat code toe om de melding op het scherm te veranderen als er een winnaar is geweest:

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

Je moet er nu voor zorgen dat deze functie wordt aangeroepen telkens als er een X of O wordt geplaatst, wat overeenkomt met elke keer dat er op een knop wordt gedrukt. Voeg aan het einde van de functie `choose_square` een aanroep van `check_win` toe, voor het geval dat het gekozen vierkant het winnende vierkant was.

Je programma zou er nu uit moeten zien als tictactoe5.py. Start het en test het spel. Als je de checks in de functie `check_win` correct hebt geschreven, zou je moeten merken dat het spel correct detecteert wanneer een speler gewonnen heeft.

Gelijkspel

Op dit moment zal het spel je toelaten om verder te spelen zelfs nadat het gewonnen is, totdat alle vierkanten geselecteerd zijn. Het zal je ook niet vertellen of het spel gelijkspel was. Je zou op dit punt kunnen stoppen, maar als je echt de kers op de taart wilt zetten, zou het toevoegen van nog een paar kleine details je spel netter kunnen maken.

Laten we eerst wat code toevoegen om te detecteren of het spel gelijkspel is. Het spel is gelijkspel als alle vakjes een X of een O bevatten, en niemand heeft gewonnen. Maak in de sectie functions een nieuwe functie genaamd `moves_taken`:

```
def moves_taken():
```

Je gaat deze functie gebruiken om het aantal zetten te tellen dat gedaan is, dus laten we een variabele starten om de telling bij te houden, aanvankelijk beginnend bij 0.

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

Weet je nog, toen we `board_squares` maakten, we een geneste lus gebruikten om alle vierkanten op het raster te maken? We hebben hier nog een geneste lus nodig om elk vierkant te controleren en te bepalen of het ingevuld is met een X of O, of dat het leeg is. Voeg deze code voor een geneste lus toe aan de functie `moves_taken`:

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

Binnen de lus moeten we controleren of dat specifieke vierkant is ingevuld met een X of een O. Als dat zo is, voeg dan 1 toe aan de variabele `moves` om te registreren dat het vierkant is geteld.

06-tictactoe.py

Taal: **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!"
    elif moves_taken() == 9:
        message.value = "It's a draw"

def moves_taken():
    moves = 0
    for row in board_squares:
        for col in row:
            if col.text == "X" or col.text == "O":
                moves = moves + 1
    return moves


# 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()
```

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

Tot slot, zodra de lussen beëindigd zijn, voeg je een `return`-statement toe om het aantal gemaakte zetten terug te geven.

```
return moves
```

Roep nu deze functie aan binnen de functie `check_win`, om te controleren op een gelijkspel. Voeg deze code toe na de code die controleert op een winnaar:

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

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

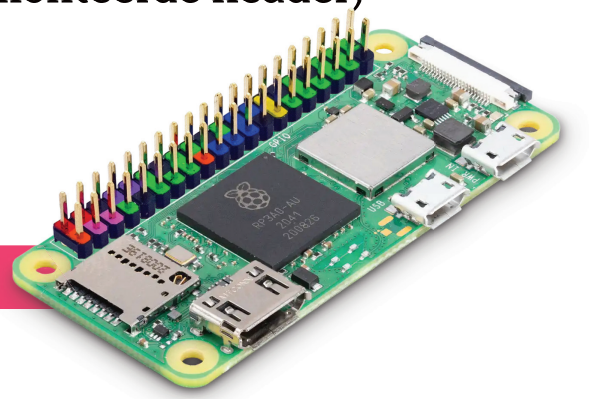
Je code zou moeten lijken op **06-tictactoe.py**. Als het spel wordt uitgevoerd, zal het nu controleren of er negen zetten zijn gedaan; als dat zo is, zal het de melding veranderen om te vertellen dat de partij gelijkspel was.

Spotlight

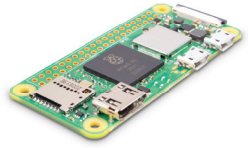
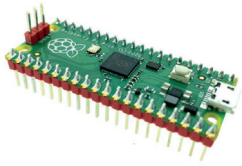
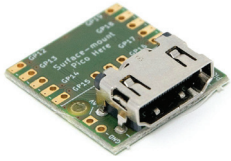
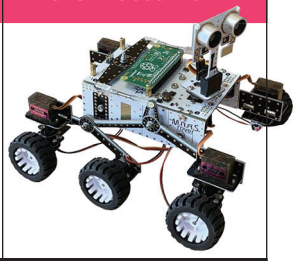
Raspberry Pi Zero 2 WH (incl. gemonteerde header)

De Raspberry Pi Zero 2 W is gebaseerd op een system-in-package (SiP), die de Broadcom BCM2710A1-chip integreert met 512 MB LPDDR2 SDRAM. De verbeterde processor biedt Raspberry Pi Zero 2 W 40% meer single-threaded prestaties en vijf keer meer multi-threaded prestaties dan de originele single-core Raspberry Pi Zero. Samen met Eurocircuits biedt Elektor een unieke versie van de Raspberry Pi Zero 2 W met 40-pins kleurgecodeerde GPIO-header die met industriële precisie is gesoldeerd.

www.elektor.nl/20157

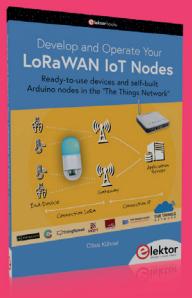


RPi Bestseller

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

BOEKEN

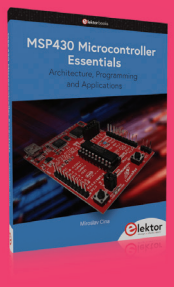
Develop and Operate Your LoRaWAN IoT Nodes



Dit boek toont de noodzakelijke stappen om te werken met LoRaWAN-Nodes met behulp van The Things Stack Community Edition TTS (CE) en het netwerk van gateways uit te breiden met een eigen gateway. Inmiddels zijn er LoRaWAN-gateways geschikt voor mobiel gebruik waarmee je via je mobiele telefoon verbinding kunt maken met de TTN-server.

www.elektor.nl/20147

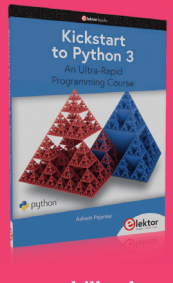
MSP430 Microcontroller Essentials



De MSP430 van Texas Instruments is een populaire microcontrollerfamilie. In dit boek gaan we aan de slag met het kleinste type, namelijk de krachtige MSP430G2553. We zullen de mogelijkheden van deze microcontroller in detail bekijken, omdat hij zeer geschikt is voor zelfbouwprojecten doordat deze beschikbaar is in een P-DIP20-package.

www.elektor.nl/20112

Kickstart to Python 3

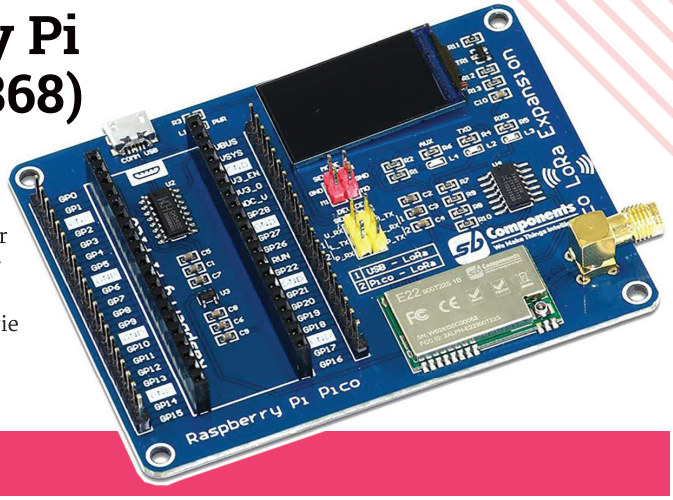


Dit boek is de allereerste stap voor beginners om Python te leren. Het boek is verdeeld in tien hoofdstukken. In het eerste hoofdstuk maken de lezers kennis met de basis van Python. Het bevat gedetailleerde instructies voor installatie op verschillende platforms zoals macOS, Windows, FreeBSD en Linux. Het behandelt ook de andere aspecten van Python-programmering, zoals IDE's en Package Manager.

www.elektor.nl/20106

Spotlight

SB Components Raspberry Pi Pico LoRa Expansion (EU868)



Pico LoRa Expansion is een datatransmissiekaart met laag stroomverbruik, wordt geleverd met een ingebouwde CH340 USB naar UART-converter, spanningsniveauvertaler (74HC125V), E22-900T22S/E22-400T22S SMA-antenneconnector die 868 MHz-frequentieband dekt, 1,14" LCD aan boord, IPEX-antenneconnector, LoRa Spread Spectrum Modulation-technologie met auto multi-level repeating.

www.elektor.nl/20096

KITS, MODULES & TOEBEHOREN

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

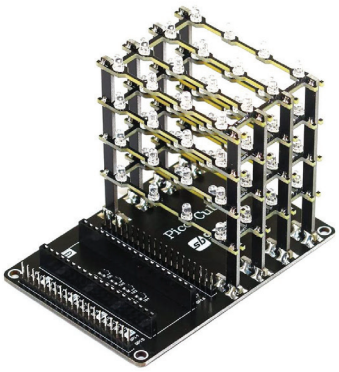
De ETH-USB-Hub-Box is een Hub-kit met ETH/USB Hub HAT (B). Het is op maat gemaakt voor de Raspberry Pi Zero-serie, klein van formaat, elke uitsparing van de behuizing is precies uitgelijnd met de connector.

www.elektor.nl/20084

SB Components Raspberry Pi Pico LED Cube (4x4x4)

Pico Cube is een 4x4x4 LED kubus HAT voor Raspberry Pi Pico met 5 V bedrijfsspanning. Pico Cube, met 64 monochrome blauwe LED's, is een leuke manier om te leren programmeren.

www.elektor.nl/20102



Seeed Studio re_computer case voor Raspberry Pi, BeagleBone en Jetson Nano

Re_computer behuizingen zijn speciaal ontworpen voor het re_computer systeem, met een verwijderbare transparante cover aan de bovenkant, en doordat ze stapelbaar zijn eindeloos uit te breiden. Re_computer behuizingen zijn compatibel met de meest populaire SBC's, waaronder Raspberry Pi, BeagleBone en Jetson Nano.

www.elektor.nl/20093

Deel 06

GUI's maken met Python: Vernietig de rondjes

Leer hoe je een Waffle gebruikt om een smakelijk spel te maken.

Je hebt in het spel boter-kaas-en-eieren gezien hoe je een GUI kunt maken op een rasterlay-out om de speler een rasterachtig bord te laten zien (pagina 47). Als je een spel maakt met een groter raster, is er een type guizero-widget Waffle (wafel) genaamd die direct een raster voor je kan maken, en echt handig is voor het maken van allerlei soorten spellen.

Een Waffle was oorspronkelijk een raster van vierkantjes in vroege versies van guizero. Dit spel heet 'Vernietig de rondjes' en het is ontstaan omdat Martin het een goed idee vond om een mix van vierkantjes en rondjes in een Waffle-widget te zetten.

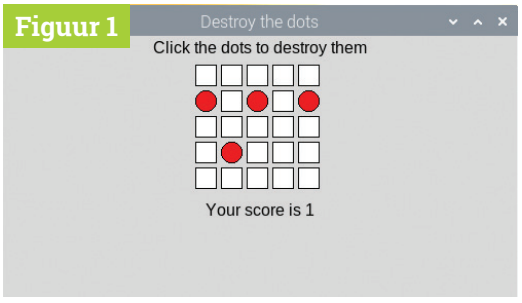
Doel van het spel

In dit spel moet je de rondjes vernietigen voordat ze jou vernietigen! Het bord bestaat uit een raster van vierkantjes. De vierkantjes zullen geleidelijk veranderen in rondjes. Om een rondje te vernietigen, klik je op het rondje en het zal weer veranderen in een vierkantje. Het doel van het spel is om het zo lang mogelijk vol te houden voordat je overspoeld wordt door rondjes (**figuur 1**).

Het spel opzetten

Laten we beginnen met het maken van een guizero-programma dat de instructies voor het spel en een Waffle bevat. Je zou nu bekend moeten zijn met de lay-out van een standaard guizero-programma met secties voor imports, functions, variables, en de app zelf.

Figuur 1



Figuur 1 Vernietig de rode rondjes voordat ze het bord overnemen.

Maak eerst een App en voeg daarbinnen een Text-widget toe voor de instructies en een Waffle widget voor het bord:

```
# 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()
```

Als je je programma uitvoert, zie je een klein 3×3 raster van witte vierkantjes. Als je je raster groter wilt maken dan dit, kun je de eigenschappen width en height toevoegen aan je Waffle:

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

Je code zou nu moeten lijken op die in de listing **destroy1.py**.

Laat de rondjes maar komen

Nu moet je een functie schrijven om een willekeurig vierkant op het bord te vinden en het in een rondje te veranderen. Begin een nieuwe functie in je sectie functions genaamd `add_dot()`:

```
def add_dot():
```

Om een willekeurig vierkant op het bord te kiezen, moet je een willekeurig paar gehele getallen als coördinaten kunnen genereren. Voeg een regel toe in je sectie imports om de functie `randint` uit de `random`-bibliotheek te importeren, waarmee je een willekeurig geheel getal kunt genereren.

```
from random import randint
```

destroy1.py

DOWNLOAD DE VOLLEDIGE CODE: magpi.cc/guizerocode

Taal: **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()
```

> Op dit punt weet je niet of de willekeurig gekozen coördinaat al een rondje is of niet

destroy2.py

Taal: **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()
```

Laten we twee variabelen maken, x en y, die je kunt gebruiken om te verwijzen naar een coördinaat op het raster. Begin je code in de functie `add_dot()`, als volgt:

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

Merk op dat je twee willekeurige getallen tussen 0 en 4 hebt gegenereerd, omdat je eerder de breedte en hoogte van het raster hebt ingesteld op 5 - de rijen en kolommen worden genummerd vanaf 0. Als je eerder andere waarden hebt gekozen, zul je de waarden hier moeten aanpassen aan de grootte van je raster. Er is echter een betere manier om dit soort aspecten te beheren (zie het kader 'Constanten gebruiken' op pagina 68).

Rondje of niet?

Nu je weet wat constanten zijn, kun je de volgende functie gebruiken om een willekeurige coördinaat op het raster te genereren:

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

Op dit punt weet je niet of de willekeurig gekozen coördinaat al een rondje is of niet. Dit lijkt misschien geen verschil te maken in het begin van een spel wanneer het bord voor het grootste deel uit vierkanten bestaat, maar als het bord gevuld wordt met rondjes, moet je er zeker van zijn dat het veld ook echt een vierkant is, anders zal het spel te gemakkelijk zijn. Een manier om dit te bereiken is door een lus te laten lopen die controleert of het gekozen vierkant al een rondje is, en als dat zo is, een ander willekeurig vierkant kiest:

```
  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)
```

Je zult je realiseren dat dit niet een bijzonder efficiënte methode is om een willekeurig vierkant te kiezen dat geen rondje is, maar het voldoet voor wat we nodig hebben in dit spel. Zodra deze lus is afgelopen, kun je er zeker van zijn dat de willekeurig gekozen x, y coördinaat zeker een vierkant is. Laten we hem omzetten in een rood rondje - na (niet binnen) je `while`-lus, voeg je de volgende regels toe:

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

Voeg een aanroep toe van je nieuwe functie `add_dot()` in de app-sectie nadat je het bord hebt gemaakt. Je programma moet er nu uitzien als **destroy2.py**. Als je het uitvoert, zou je een enkele willekeurig rood rondje in het raster moeten zien. Als je het programma opnieuw uitvoert, zal het rondje waarschijnlijk op een andere willekeurige plaats staan (**Figuur 2**).

Vernietig het rondje

Tot nu toe is er maar één rondje - laten we dat vernietigen! Maak je geen zorgen, je zult later meer rondjes toevoegen om te vernietigen. Maar als je er eenmaal één kunt vernietigen, kun je ze allemaal vernietigen!

Maak een nieuwe functie in de sectie functions met een echt bevredigende naam - destroy_dot - en geef het twee parameters, x en y.

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

Deze functie zal controleren of de coördinaat x,y een rondje is (in plaats van een vierkant). Je kan dit doen met dezelfde code als de code om een rondje te maken - de code `board[x, y].dotty` zal True teruggeven als die coördinaat een rondje is, en False als het een vierkant is.

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

Als de coördinaat een rondje is, verander het dan in een vierkant door zijn eigenschap `dotty` op False te zetten, en ook zijn kleur weer in wit te veranderen:

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

Deze functie moet getriggerd worden als er op het bord wordt geklikt. Zoek de regel code die je al hebt om het bord te maken en voeg een commando als dit toe:

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

Dit zal de functie `destroy_dot` aanroepen telkens als er op een veld op het bord wordt geklikt.

Merk op dat een Waffle-widget automatisch twee parameters doorgeeft aan een command-functie; dit zullen altijd de x en y coördinaten zijn van de pixel waarop geklikt is om het command te triggeren.

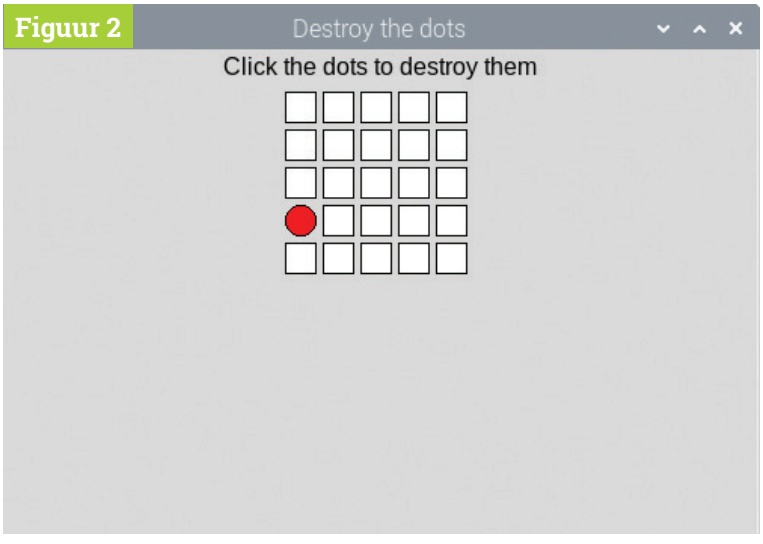
Je code zou er nu moeten uitzien als **destroy3.py**. Test je programma door het uit te voeren en op het rondje te klikken. Je zou nu moeten zien dat het rondje weer verandert in een wit vierkantje. Als je op

destroy3.py

Taal: **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()
```

Figuur 2



Figuur 2 Een willekeurige rood rondje genereren.

Constanten gebruiken

Het instellen van de hoogte en breedte van je Waffle op 5 staat bekend als het gebruik van een 'magisch getal' in een programma, omdat het specifieke getal hard gecodeerd is in het programma. Als je de grootte van het raster wilt veranderen, zul je overal in het programma dit getal moeten vinden en aanpassen, wat chaotisch kan zijn.

Een betere programmeerpraktijk zou zijn om een constante te definiëren in je sectie variables genaamd GRID_SIZE en deze gelijk te stellen aan 5:

```
GRID_SIZE = 5
```

Dan, in plaats van de afmetingen van je Waffle te definiëren met een magisch getal 5, kun je zetten:

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

Als je later besluit om de grootte van het raster te veranderen, kun je gewoon de waarde van deze constante veranderen. Nadenken over dit soort dingen op het moment dat je het programma schrijft zal je helpen om later hoofdpijn te voorkomen als je besluit om het te veranderen.

destroy4.py

Taal: **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()
```

een vierkantje klikt dat geen rondje is, zou er niets moeten gebeuren.

Meer rondjes!

Nu is het tijd om het spel echt een uitdaging te maken, door voortdurend opkomende rondjes toe te voegen. Laten we beginnen met elke seconde een nieuw willekeurig rondje toe te voegen. Om dit te doen, moet je elke seconde een aanroep van de functie `add_dot` inplannen door gebruik te maken van een ingebouwde functie van guizero genaamd `after`.

Verwijder in je app-sectie de aanroep van `add_dot()` en vervang deze door een nieuwe regel code:

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

Deze regel code betekent 'na 1000 milliseconden (1 seconde), roep de functie `add_dot`' aan.

Als je het programma nu uitvoert, krijg je nog steeds een enkel rondje, maar het verschijnt op het raster na een vertraging van 1 seconde. Hier komt het slimme gedeelte. Zoek je functie `add_dot` en voeg dezelfde regel code toe, aan het eind van de functie.

Dit zal een nieuwe aanroep van `add_dot` plannen telkens als een nieuw rondje klaar is met toegevoegd te worden. De volgende dot zal ook na 1 seconde verschijnen, dus als je het programma uitvoert, zou je elke seconde een nieuw rondje op het raster moeten zien verschijnen (**Figuur 3**).

Probeer je programma uit te voeren, dat er nu zou moeten uitzien als **destroy4.py**. Aangezien je de methode om een rondje te vernietigen al geschreven hebt, zou het klikken op een rondje het moeten verwijderen. Als je het spel echter een tijdje speelt, zul je merken dat het vrij eenvoudig is om het tempo van één rondje per seconde bij te houden en dat het bijna onmogelijk is om het spel te verliezen.

Je moet nog wel twee dingen toevoegen - een score om bij te houden hoeveel rondjes je hebt vernietigd, en een manier om het spel moeilijker te maken, zodat het een uitdaging wordt.

Een score toevoegen

Een score toevoegen is vrij eenvoudig en bestaat uit drie stappen::

- Voeg een variabele toe om de score bij te houden; de variabele moet beginnen bij 0.
- Toon een melding op de GUI met de huidige score.
- Telkens wanneer de functie destroy_dot wordt aangeroepen en er een rondje wordt vernietigd, voeg je 1 toe aan de score en update je de melding.

Probeer de code zelf toe te voegen met wat je al geleerd hebt.

Hint: *om de variabele score bij te werken vanuit de functie* `destroy_dot`*, moet je hem tot een global declareren.*

Hint: *als je een foutmelding krijgt dat er naar de variabele score wordt verwezen vóór de toewijzing, zorg er dan voor dat je sectie variables vóór je sectie functions komt.*

De oplossing staat hierondera, als je vastzit...

Oplossing: voeg een score toe

Voeg eerst een variabele toe in je sectie variables:

```
score = 0
```

Voeg vervolgens een nieuwe Text-widget toe in de sectie app om de score weer te geven:

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

Voeg ten slotte 1 toe aan de score telkens wanneer een rondje wordt vernietigd:

```
def destroy_dot(x,y):

    # Declare score global
    global score

    # This code already exists
    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)
```

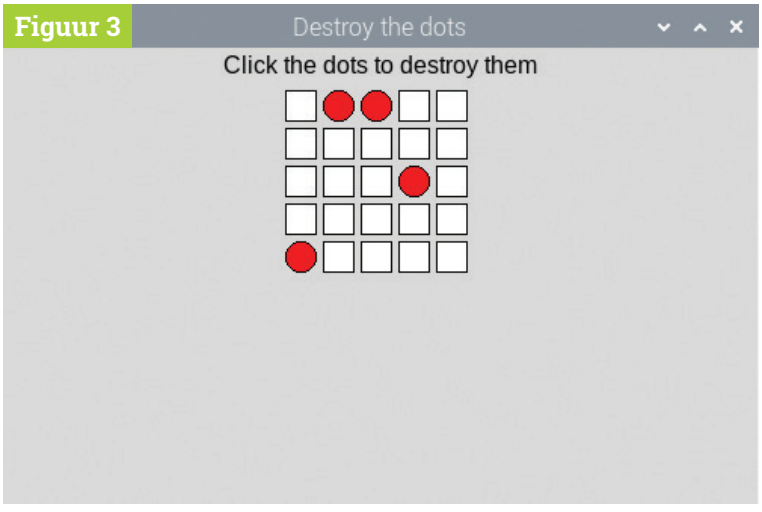
Je code (zonder de optionele commentaren) zou nu moeten lijken op **destroy5.py**. Test je spel en je zou

destroy5.py

Taal: **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()
```

Figuur 3



▲ **Figuur 3** Elke seconde verschijnt er een nieuw rondje.

> We moeten uitzoeken wanneer de speler het spel verloren heeft; dit gebeurt wanneer elk vierkantje veranderd is in een rood rondje

destroy6.py

Taal: **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()
```

je score met 1 moeten zien oplopen elke keer dat je op een rondje klikt.

Zet de speler onder druk

Nu dat je de score van de speler kunt bijhouden, kan je dit gebruiken om de speler onder druk te zetten en het opkomen van rondjes te versnellen als hij het goed doet.

Weet je nog dat je een after call gebruikte in de functie `add_dot` om een nieuw rondje te plannen na 1000 milliseconden (of 1 seconde)? Ga terug en zoek die regel op - we gaan hem een beetje veranderen.

Maak eerst een variabele `speed` aan en stel die in op 1000. Dan, in plaats van een aanroep te plannen om een rondje toe te voegen na 1000 ms, plan je hem om een rondje toe te voegen na speed milliseconden. Dit zal absoluut geen effect hebben op het spel... nog niet. Je plant nog steeds de volgende aanroep na 1000 ms, maar dat getal komt nu van de variabele speed in plaats van hard gecodeerd als een magisch getal.

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

Dit is hoe je de druk kunt opvoeren. Tussen deze twee regels code kun je wat code toevoegen om de snelheid van de rondjes in te stellen, afhankelijk van de huidige score. Hier is een voorbeeld:

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

Hier kun je zien dat als de speler meer dan 10 punten heeft, de nieuwe rondjes elke 500 ms verschijnen, als hij meer dan 20 punten heeft verschijnt er elke 400 ms een rondje, enzovoort. Dit maakt het spel veel moeilijker naarmate je meer punten hebt. Sla je code op - **destroy6.py** - en test het spel om het verschil te zien. Je kunt de getallen veranderen of meer `elif`-voorwaarden toevoegen als je de moeilijkheidsgraad nog verder wilt verhogen.

Game over

Alles wat je nu nog moet doen is uitzoeken wanneer de speler het spel verloren heeft; dit gebeurt wanneer elk vierkantje veranderd is in een rood rondje.

Weet je nog dat toen je boter-kaas-en-eieren maakte, je geneste lussen gebruikte om te controleren of alle vakjes gevuld waren en het gelijkspel was? Je kunt dezelfde methode hier ook gebruiken, om langs elk vierkantje op het raster te gaan en te controleren of het een rood rondje is. In de functie `add_dot`, net voor de aanroep van `after`, voeg je wat code toe voor een geneste lus om door alle vakjes op het bord te lopen:

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

De eerste regel begint met aan te nemen dat alle vierkanten rood zijn. De geneste lus geeft achtereenvolgens de coördinaten van elk vierkant op het raster, als de waarden `x` en `y`, zodat je kunt controleren of dit waar is.

Voeg wat code toe in de tweede lus om uit te zoeken of de huidige pixel rood is, en als dat niet zo is, verander dan de variabele `all_red` in `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
```

Nadat beide lussen zijn afgelopen (zorg ervoor dat je de volgende code unindent), controleer je of het raster alleen rode rondjes bevat. Zo ja, dan heeft de speler verloren, dus toon een boodschap:

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

Als de speler niet verloren heeft, moet het spel doorgaan. Voeg een `else:` toe en laat daarin de methode `after` die je al hebt, inspringen, want we willen alleen een nieuw rondje toevoegen als de speler niet verloren heeft:

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

Zorg ervoor dat je de after-regel die je hier al hebt, inspringt, in plaats van er nog een toe te voegen, of je spel zal zich vreemd gaan gedragen en meerdere rondjes per seconde genereren!

Je uiteindelijke code zou moeten lijken op die in **07-destroy-the-dots.py**. Veel plezier met het spel.

> Nadat beide lussen zijn afgelopen, controleer je of het raster alleen rode rondjes bevat. Zo ja, dan heeft de speler verloren

07-destroy-the-dots.py

Taal: **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()
```

Deel 07

GUI's maken met Python: Flood it

Leer hoe je een Waffle gebruikt om een smakelijk spel te maken.

Flood It is een spel waarbij het de bedoeling is om alle vakjes op het bord te vullen met dezelfde kleur. Beginnend met het vakje linksboven, kiest de speler een kleur om het bord mee vol te laten lopen. Het biedt een iets complexer spel op basis van Waffles.

Doel van het spel

In dit voorbeeld (**Figuur 1**) is het vakje linksboven blauw. De speler kan kiezen om die kleur of in het enkele paarse vakje eronder te laten lopen, of in het gele vakje rechts.

Het gele vakje vol gieten zou een betere zet zijn omdat dan alle aangrenzende gele vakjes ook vol zouden lopen, en de speler mag maar een beperkt aantal zetten doen voordat het spel eindigt.

Het spel opzetten

Download (van **magpi.cc/floodit**) en open het starterbestand, **floodit_starter.py**. Sla het op een logische plaats op.

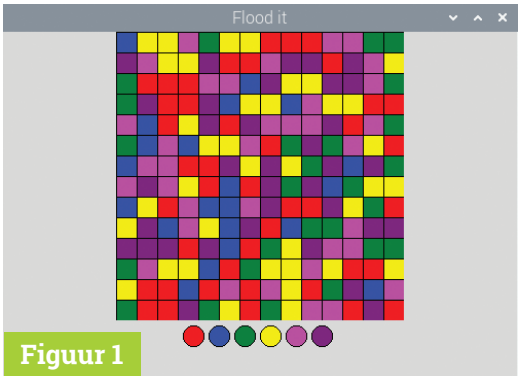
In de sectie Variables, geef je de variabelen enkele waarden:

- `colours` – een lijst van zes kleuren als strings. Dit kunnen gewone kleurnamen zijn of hex-kleuren. De kleurnamen "white", "black", "red", "green", "blue", "cyan", "yellow", en "magenta" zullen altijd beschikbaar zijn.
- `board_size` – de breedte/hoogte van het bord als een geheel getal; wij kozen 14. Het bord is altijd een vierkant.
- `moves_limit` – het aantal zetten dat de speler mag doen voordat hij verliest, als een geheel getal; wij kozen 25.

Maak in de sectie app een App-widget en geef hem een titel.

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

Als je dit laat runnen, resulteert het in een standaard venster met label (**Figuur 2**).



Figuur 1 Vul alle vakjes met één kleur.

Het bord

Het bord is een raster van vierkantjes, elk met een willekeurig gekozen kleur uit de lijst die je eerder hebt gemaakt.

Voeg in de app een Waffle-widget toe. Dit maakt een raster dat het bord wordt.

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

Start je programma en je zult zien dat het raster een beetje te klein is (**Figuur 3**).

Voeg aan de regel code die je net schreef parameters toe voor de breedte en hoogte van de Waffle, en zet de ruimte (padding) tussen de rastervierkantjes op nul.

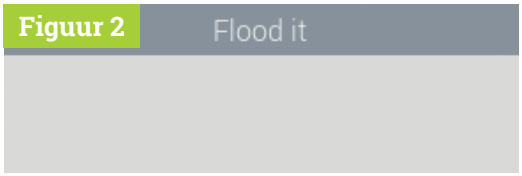
```
board = Waffle(app, width=board_size,
height=board_size, pad=0)
```

Dat is beter (**Figuur 4**, volgende pagina).

Het palet

Het palet laat de spelers zien op welke kleuren ze kunnen klikken om het bord te laten vollopen. Zij zullen op deze kleuren klikken om het spel te spelen. Het palet van het voltooide spel is te zien in **Figuur 5**.

Op de regel onder die waarmee je het bord hebt gemaakt, maak je nog een Waffle, maar deze keer moet hij `palette` heten.



▲ **Figuur 2** Het gebruikelijke venster met label.

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

Weet je nog dat je in de vorige stap de parameters hebt toegevoegd aan de Waffle van het bord? Voeg nu deze parameters toe aan de Waffle palette, elk gescheiden door een komma:

`width = 6` (het aantal kleuren dat we hebben)
`height = 1`
`dotty = True` (dit maakt van de vierkanten cirkels)

Dus, nu zou je moeten hebben:

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

Voer de code uit om een leeg palet te zie (**Figuur 6**).

Kleur het bord in

Het bord moet beginnen met elk vakje in een willekeurig gekozen kleur uit de lijst colours die je eerder hebt gemaakt. Zet op de regel onder je palet een aanroep van een functie

```
  fill_board()
```

Ga naar de sectie Functions in je programma, en begin met het schrijven van de code voor deze nieuwe functie:

```
  def fill_board():
```

Je kunt een geneste lus schrijven om door elke rij en kolom in het bord te lopen. Elke pixel wordt gekleurd met een willekeurig gekozen kleur uit de lijst. Om een pixel in te kleuren, gebruik je deze code, waar de ? tekens vervangen worden door de x-, y-coördinaten van de pixel:

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

Probeer de code zelf te schrijven met wat je geleerd hebt over geneste lussen in de vorige artikelen van deze serie - de oplossing staat hieronder als je vastloopt

Hint: *Gebruik de variabele board_size om te weten hoeveel keer je moet lussen.*

> Het bord moet beginnen met elk vakje in een willekeurig gekozen kleur

```
  def fill_boa
```

Als je je code uitvoert, zou je een kleurig bord moeten zien. Als je een wit bord ziet, controleer dan of je de functie `fill_board()` wel hebt aangeroepen (**Figuur 7**).

Hier is één oplossing, maar er zijn vele manieren waarop je dit kunt doen:

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

Een alternatieve oplossing die gebruik maakt van een meer geavanceerde optie genaamd list comprehension:

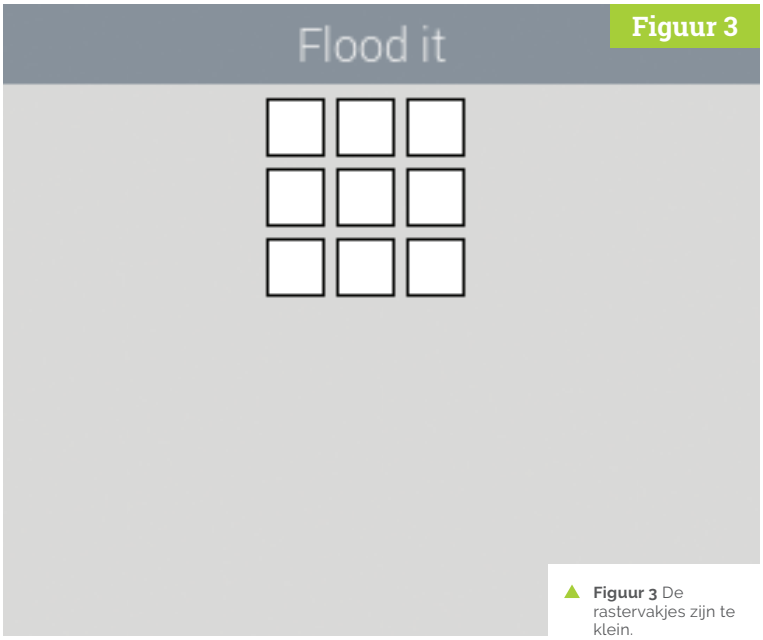
```
    board.set_pixel(x, y, random.
choice(colours))
```

Kleur in het palet

Laten we, nu je een kleurig bord hebt, ook het palet inkleuren. Op de regel onder je `fill_board()` code, zet je een aanroep van een functie:

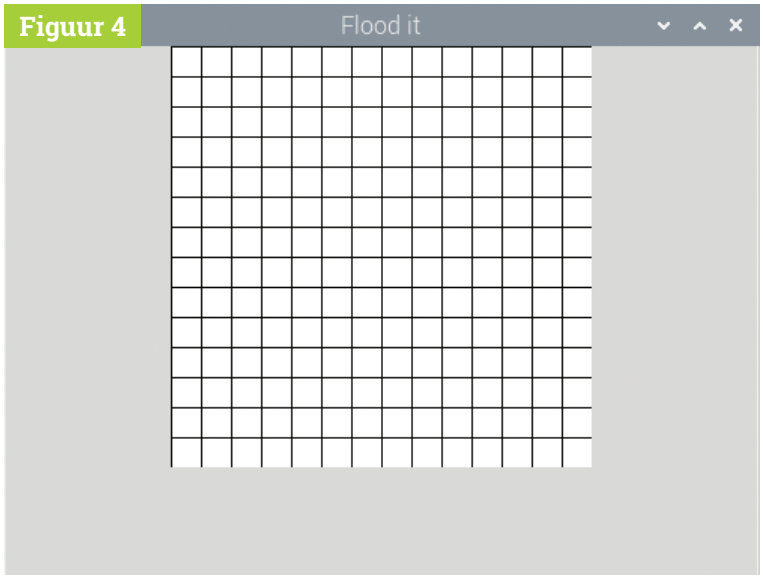
Python 3 Programming and GUIs

Dit boek is bedoeld voor ingenieurs, wetenschappers en hobbyisten die pc's willen integreren in hun hardware-projecten met behulp van grafische gebruikersinterfaces. Zowel desktop- als webgebaseerde toepassingen komen aan bod. De gebruikte programmeertaal is Python 3, een van de populairste talen op de markt, met snelheid bij het programmeren als belangrijk kenmerk.
www.elektor.nl/18192



▲ **Figuur 3** De rastervakjes zijn te klein.

Figuur 4



▲ **Figuur 4** Een raster met de juiste bordgrootte, zonder tussenruimtes.

▶ **Figuur 6** Een leeg palet.

```
init_palette()
```

Zoek de sectie Functions in je programma, en begin met het schrijven van de code voor deze nieuwe functie:

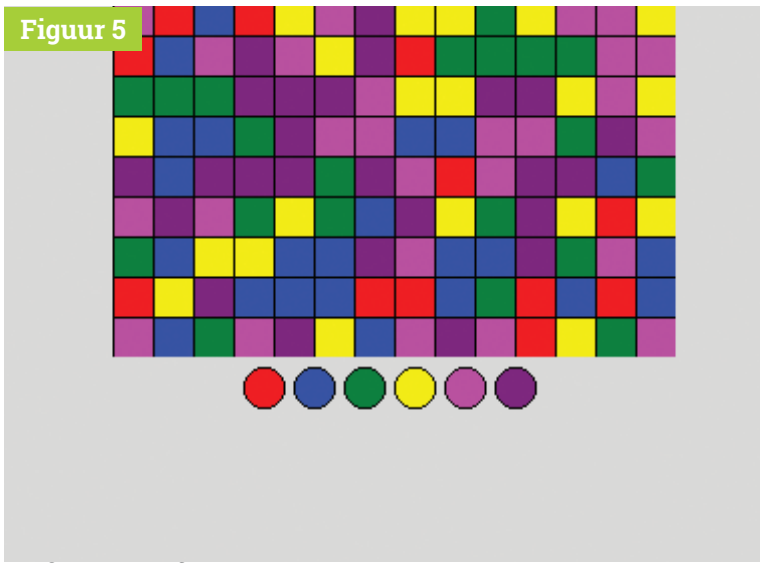
```
def init_palette():
```

Het idee hier is om door alle kleuren in de lijst te lopen, en er één toe te wijzen aan elk van de cirkels in het palet. Je kunt dezelfde `set_pixel` methode gebruiken als je voor het bord hebt gebruikt om de kleur van de cirkels in het palet te veranderen.

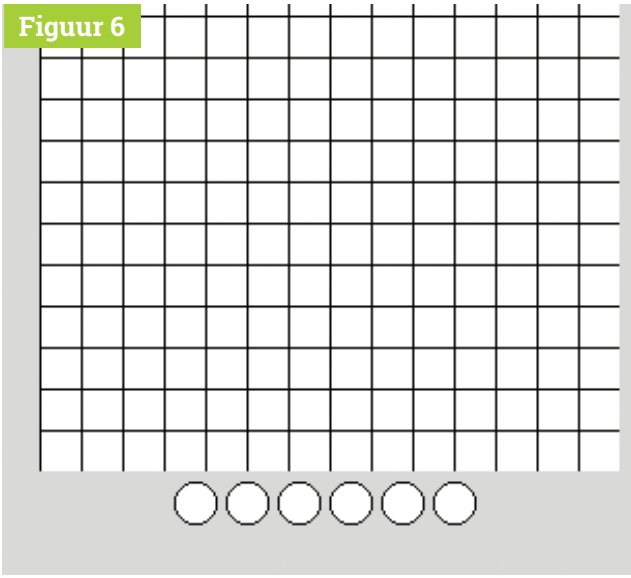
Probeer de code zelf te schrijven. Als je vastloopt, zie dan het kader 'Op hoeveel manieren kun je het palet kleuren' op de laatste pagina voor enkele mogelijke oplossingen.

▼ **Figuur 5** Je hebt een palet nodig waarmee de speler een kleur kan kiezen.

Figuur 5



Figuur 6



Hint: *Alle cirkels in het palet staan in rij 0 van de Waffle.*

Vollopen

Als de speler op een kleur in het palet klikt, moet het bord vollopen met die kleur, te beginnen met het vakje linksboven.

Maak in de sectie Functions een nieuwe functie aan genaamd `start_flood` op precies dezelfde manier als je deed voor de laatste twee functies. Deze functie moet twee parameters aannemen die de x- en y-coördinaten zijn van het vakje waarop werd geklikt. Voeg deze toe tussen de haakjes zodat je code er uiteindelijk als volgt uitziet

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

Voeg een regel code (ingesprongen) toe aan de functie om de naam te krijgen van de kleur waarop geklikt werd:

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

Dit zal de kleur zijn waarmee het bord wordt gevuld.

Voeg nu een regel code toe om de huidige kleur van de startpixel te krijgen - dit is altijd de pixel linksboven in het bord, op de coördinaten 0, 0.

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

Roep nu de functie `flood` op, die al voor je geschreven is in het starterbestand. Deze functie start op 0,0 en overstroomt alle pixels die grenzen aan de pixel linksboven en die dezelfde kleur hebben als de `flood_colour`.

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

Deze functie moet lopen telkens wanneer iemand klikt op een kleur in het palet, dus zoek de regel code waar je het palet hebt gemaakt.

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

Voeg nog een parameter toe, dat is een commando. Wanneer op een cirkel op het palet wordt geklikt, zal dit commando worden uitgevoerd. Het commando is de functie `start_flood`, dus je code zou er nu zo uit moeten zien:

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

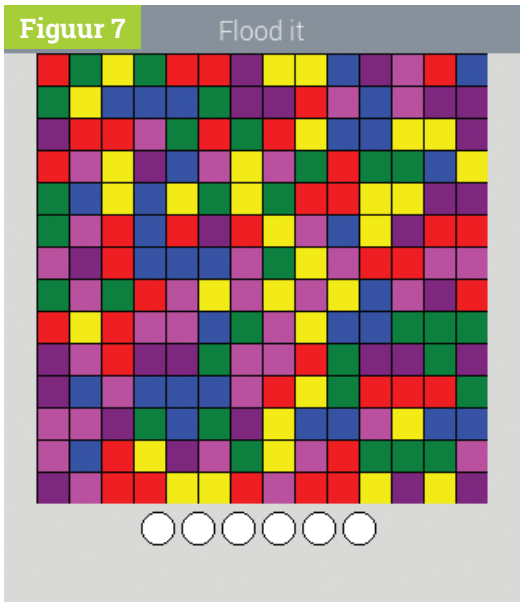
Test je code door te klikken op de cirkels op het palet.

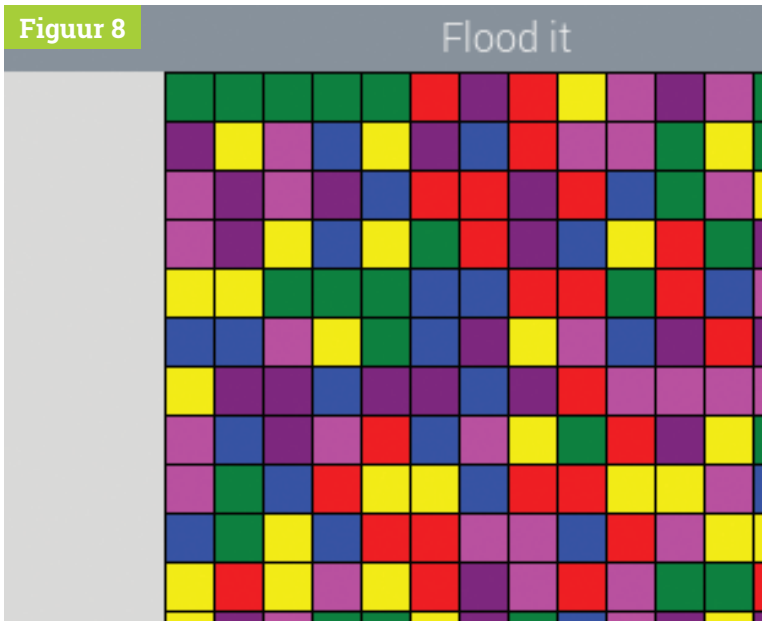
In ons willekeurige voorbeeld (**Figuur 8**) is het vakje linksboven groen. Als je op paars klikt op het palet, wordt het vakje paars en sluit het aan op het paarse vakje eronder (**Figuur 9**, volgende pagina). Nu zijn er vijf paarse vakjes verbonden met het vakje linksboven. Laten we nu op roze klikken om de roze vakjes eronder ook te verbinden (**Figuur 10**).

Nu is er een grote keten van roze vakjes. Ga verder met het spel door op verschillende kleuren in het palet te drukken. Het doel is om uiteindelijk alle vakjes dezelfde kleur te geven.

Winnen

Op dit moment gebeurt er niets als het de speler lukt om alle vakjes in het raster dezelfde kleur te geven. De speler mag ook een oneindig aantal beurten doen, omdat het aantal zetten dat hij gedaan heeft niet wordt bijgehouden.





> Voeg een stukje tekst toe aan de GUI om weer te geven of de speler gewonnen of verloren heeft

Figuur 8 Hier is het vakje linksboven groen.

Laten we eerst een stukje tekst toevoegen aan de GUI om aan te geven of de speler gewonnen of verloren heeft. De tekst zal blanco beginnen. Onder de code voor het palet, voeg je een Text-widget toe genaamd `win_text`.

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

In de sectie Variables, voeg je een variabele toe genaamd `moves_taken` en stel je deze in op 0. Maak nu een functie genaamd `win_check`, die na elke zet controleert of de speler gewonnen heeft.

Eerst moet je specificeren dat je de waarde van de globale variabele `moves_taken` wilt kunnen veranderen.

```
global moves_taken
```

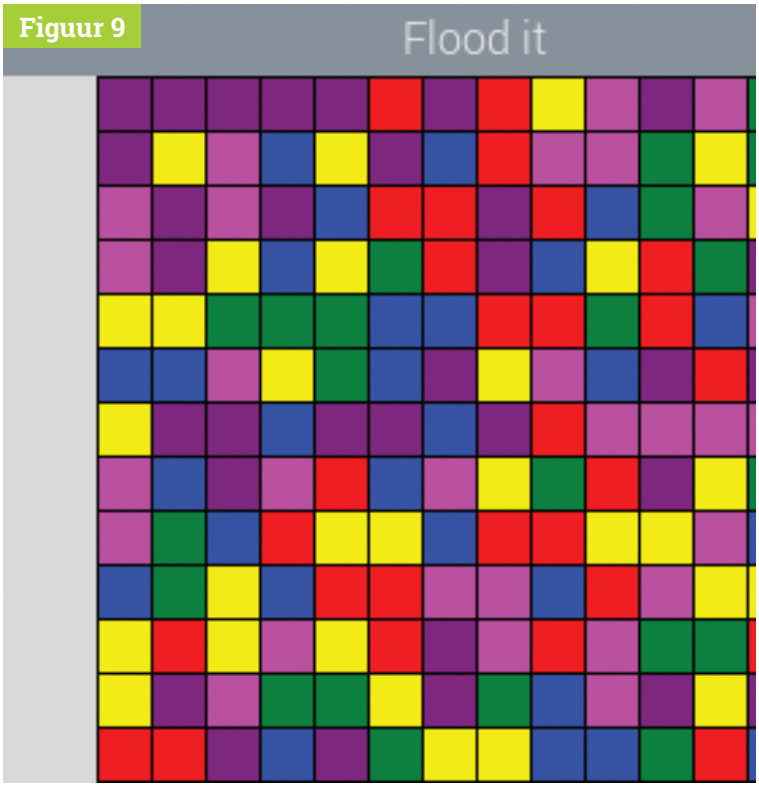
Voeg dan 1 toe aan de `moves_taken` variabele - elke keer dat deze functie wordt aangeroepen, voegen we een zet toe.

```
moves_taken += 1
```

Controleer of `moves_taken` kleiner is dan `moves_limit` of niet:

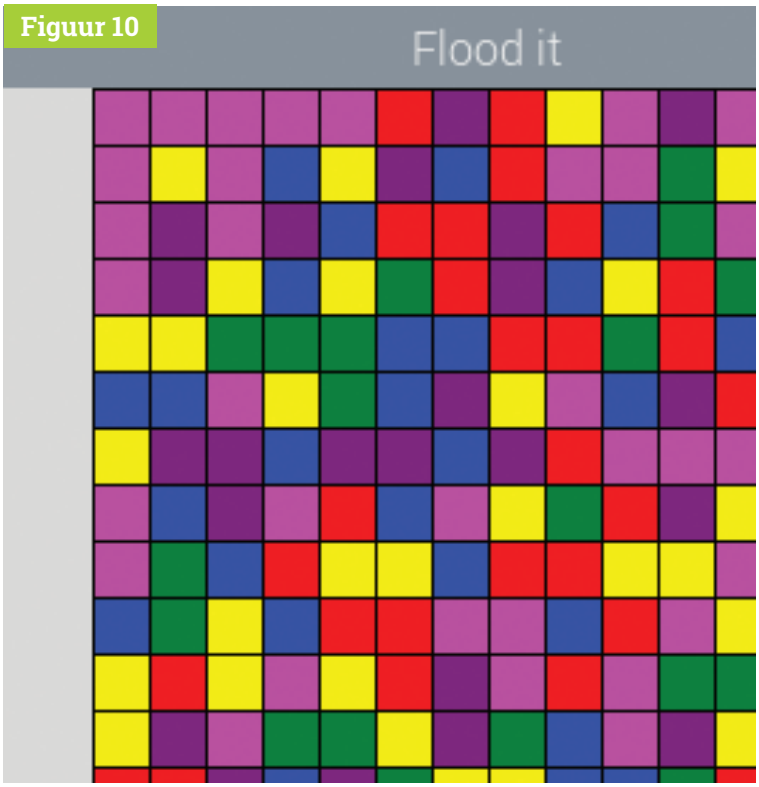
Figuur 7 Elk vakje van het bord wordt willekeurig gekleurd.

Figuur 9



▲ **Figuur 9** Door op paars te klikken wordt het paars.

Figuur 10



▼ **Figuur 10** Klik op roze voor een keten van roze.

```
if moves_taken < moves_limit:

else:
```

Als `moves_taken` niet binnen de limiet is, betekent dit dat de speler geen zetten meer heeft, dus werk je de tekst bij om te zeggen dat hij verloren heeft:

```
if moves_taken < moves_limit:

else:
    win_text.value = "You lost :("
```

Als het getal van `moves_taken` kleiner is dan de limiet, controleer dan of alle vakjes dezelfde kleur hebben door de functie aan te roepen die al voor je geschreven is in het starterbestand. Zorg ervoor dat de volgende code is ingesprongen onder het eerste if statement:

```
if all_squares_are_the_same():
      win_text.value = "You win!"
```

Het volledige stuk code moet er zo uitzien:

```
def win_check():
    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 :("
```

Tot slot moet je de `win_check` functie aanroepen telkens als er op een vakje geklikt wordt. De makkelijkste manier om dit te doen is om de functie-aanroep toe te voegen aan het einde van de functie `start_flood`.

Nu is het tijd om het spel te testen. Een voorbeeld van de code staat in de listing **08-floodit.py**.

Test je spel

Je kunt testen of het spel werkt door het te spelen; het kan echter lang duren om te testen of je kunt winnen! Een gemakkelijkere manier om dit te controleren is door de variabele `board_size` te veranderen in iets kleins zoals 5, en dan het spel te spelen op een veel kleiner raster om te zien of je kunt winnen.

Je kunt gemakkelijk testen of het spel je op de juiste manier laat verliezen door 25 keer op dezelfde kleur te klikken!

Op hoeveel manieren kun je het palet kleuren?

Hier is een oplossing die gebruik maakt van een lus en een variabele om bij te houden welke kolom je inkleurt:

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

Hier is een vergelijkbare oplossing die een **range** gebruikt in de **for**-lus in plaats van een teller-variabele:

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

Hier is een andere oplossing die de indexfunctie colours. **index(colour)** gebruikt. Deze code zegt 'Zoek in de lijst colours de positie in de lijst van colour'. Dus, bijvoorbeeld als je lijst [**"green"**, **"blue"**, **"red"**] was, dan zou de index van green 0 zijn, de index van blue zou 1 zijn, enz., in de wetenschap dat we beginnen te tellen vanaf nul.

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

Je kunt elk van deze oplossingen gebruiken, of je kunt zelf een andere manier bedacht hebben. Geen hiervan is het 'juiste antwoord': er zijn vaak veel verschillende manieren om een oplossing te coderen.

08-floodit.py

DOWNLOAD DE VOLLEDIGE CODE:
magpi.cc/floodit

Taal: **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()
```

Deel 08

GUI's maken met Python: Emoji Match

Maak een leuk spel 'zoek het identieke plaatje'.

We gaan een emoji-match-spel maken (figuur 1). Het doel van het spel is om de ene emoji te vinden die in twee verschillende blokken emoji's hetzelfde is. Je krijgt een punt voor elke juiste match en verliest een punt voor een onjuiste match.

Emoji's downloaden

Om het spel te maken, heb je een aantal emoji's nodig. Je kunt de emoji's gebruiken die voor Twitter zijn gemaakt (**twemoji.twitter.com**). Download het bestand **emojis.zip** van **magpi.cc/guizeroemojis**, open het zip-bestand, en kopieer de map **emojis** naar de map waar je je code opslaat.

Het spel moet negen willekeurige emoji's kiezen en ze in een raster rangschikken. Een eenvoudige manier om dit te doen is door alle emoji's in een lijst te zetten en ze in een willekeurige volgorde te plaatsen.

Maak een nieuw programma met de gebruikelijke commentaarregels voor de verschillende secties (Imports, Variabels, Functions, App), zoals we in de vorige artikelen in deze serie gedaan hebben. Onder imports, voeg je toe:

▲ **Figuur 1** Het voltooide spel.

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

Dan, onder variabelen, voer je deze code in die een lijst van emoji's in willekeurige volgorde maakt, elk in de vorm **pad/emoji_bestandsnaam**.

```
# 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)
```

De variabele **emojis_dir** is het pad van de emoji's op je computer; het vertelt de code die de emoji's laadt vertellen waar ze te vinden zijn.

Test je programma. Probeer de lijst `emojis` te printen naar het scherm met `print(emojis)`. Je zou een lange lijst van bestandsnamen moeten zien. De lijst moet elke keer in een andere volgorde staan als je het programma uitvoert.

De emoji's weergeven

Vervolgens moet de code twee 3×3 rasters maken van Picture- en PushButton-widgets die de emoji's zullen tonen.

Pas je programma aan om een guizero app te maken en een Box om de picture-widgets in te plaatsen met een `"grid"` lay-out. Voeg in de sectie imports deze regel toe om de benodigde widgets te importeren:

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

In de app-sectie, voeg je de volgende code toe:

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

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

De Box-widget is erg handig voor het opmaken van je GUI. Het is een onzichtbaar gebied van je GUI waar je widgets kunt groeperen. Een box kan zijn eigen layout, size, en bg (achtergrond) hebben. Ze kunnen ook verborgen of weergegeven worden,

> De widgets worden toegevoegd aan een lijst zodat er later in het spel naar verwezen kan worden

emoji1.py

DOWNLOAD DE VOLLEDIGE CODE:
magpi.cc/guizerocode

Taal: **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()
```

wat betekent dat je gemakkelijk een verzameling widgets onzichtbaar kunt maken.

Als je de box wilt zien, kun je een rand toevoegen door de parameter op True te zetten.

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

Voeg nu de Picture-widget toe aan je imports:

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

Voeg in de app-sectie de code toe om de Picture-widgets te maken en ze toe te voegen aan een lijst.

```
pictures = []

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

Om coördinaten toe te kennen aan elke Picture-widget, worden twee `for`-lussen gebruikt. Ze lopen beide door het bereik 0–2; de ene wijst zijn waarde toe aan de variabele `x` en de andere aan de variabele `y`. De rasterpositie van elke widget wordt ingesteld met behulp van de `x`- en `y`-waarden. De widgets worden toegevoegd aan een lijst, zodat er later in het spel naar verwezen kan worden.

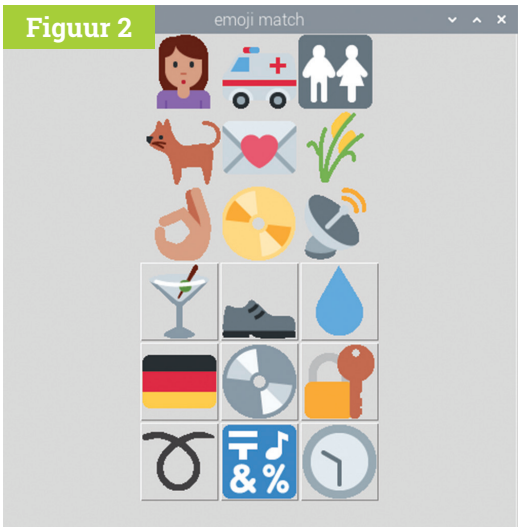
Doe hetzelfde voor de PushButton-widgets om het tweede 3×3 raster te maken. Voeg eerst de widget toe aan je imports:

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

Voeg in de app-sectie regels toe zodat die er als volgt uitziet:

```
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)
```

Figuur 2



▲ **Figuur 2** Geen overeenkomende emoji's.

Maak in de sectie functions een functie om elke ronde van het spel op te zetten.

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

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

Om aan elke `picture`- en `button`-widget een emoji toe te wijzen, wordt de eigenschap `image` ingesteld op een item uit de lijst `emojis`. Emoji's worden geselecteerd met `pop()`, die het laatste item in een lijst kiest en het dan uit de lijst verwijdert. We hebben deze functie gebruikt omdat hiermee wordt voorkomen dat een emoji meer dan een keer in het spel verschijnt.

Roep onderaan je programma de functie `setup_round` aan en geef de app weer.

```
setup_round()

app.display()
```

Je programma zou nu moeten lijken op **emoji1.py** (zie listing links). Test het en je zou twee rasters van negen emoji's moeten zien.

Gelijke emoji's

Op dit moment zullen alle emoji's in je app verschillend zijn (**Figuur 2**). In de volgende stap kies je een andere emoji die hetzelfde is, en werk je één plaatje en één knop bij zodat ze dezelfde matchende emoji hebben.

Voeg `randint` toe aan de regel `from random import`. Dit wordt gebruikt om een getal van 0 tot 8 te krijgen voor elke picture en button.

emoji2.py

Taal: **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

Taal: **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.  Text
009.
010.  # ------------------------------
011.  # Variables
012.  # ------------------------------
013.
014.  # set the path to the emoji folder on your computer
015.  emojis_dir = "emojis"
016.  emojis = [os.path.join(emojis_dir, f) for f in
017.  os.listdir(emojis_dir)]
018.  shuffle(emojis)
019.
020.  # ------------------------------
021.  # Functions
022.  # ------------------------------
023.
024.  def setup_round():
025.      for picture in pictures:
026.          picture.image = emojis.pop()
027.
028.      for button in buttons:
029.          button.image = emojis.pop()
030.          button.update_command(
      match_emoji, args=[False])
031.
032.      matched_emoji = emojis.pop()
033.
034.      random_picture = randint(0,8)
035.      pictures[random_picture].image = matched_emoji
036.
037.      random_button = randint(0,8)
038.      buttons[random_button].image = matched_emoji
039.
040.      buttons[random_button].update_command(
      match_emoji, [True])
041.
042.  def match_emoji(matched):
043.      if matched:
044.          result.value = "correct"
045.      else:
046.          result.value = "incorrect"
047.
048.      setup_round()
049.
050.  # ------------------------------
051.  # App
052.  # ------------------------------
053.
054.  app = App("emoji match")
055.
056.  pictures_box = Box(app, layout="grid")
057.  buttons_box = Box(app, layout="grid")
058.
059.  pictures = []
060.  buttons = []
061.
062.  for x in range(0,3):
063.      for y in range(0,3):
064.          picture = Picture(pictures_box, grid=[x,y])
065.          pictures.append(picture)
066.
067.          button = PushButton(buttons_box, grid=[x,y])
068.          buttons.append(button)
069.
070.  result = Text(app)
071.
072.  setup_round()
073.
074.  app.display()
```

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

Voeg daarna deze code (ingesprongen) toe onderaan de functie `setup_round` om een andere emoji uit de lijst te halen en deze in te stellen als het plaatje van een willekeurig picture en button.

```
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
```

Je code zou er nu uit moeten zien als **emoji2.py** (zie vorige pagina). Start nu je programma; een van de emoji's zou hetzelfde moeten zijn. Kijk goed – de overeenkomende emoji kan moeilijk te vinden zijn.

Controleer de gok

Elke keer dat een van de buttons wordt ingedrukt, moet worden gecontroleerd of dit de overeenkomende emoji is en moet het resultaat 'correct' of 'incorrect' op het scherm worden gezet. Na de gok van de speler wordt een nieuwe ronde opgezet en een andere set emoji's getoond.

Je app heeft een Text-widget nodig die het resultaat weergeeft. Voeg hem toe aan je imports:

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

Voeg deze regel toe in je app-sectie:

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

emoji4.py

Taal: **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.          pictures_box.hide()
058.          buttons_box.hide()
059.          timer.hide()
060.          score.hide()
061.
062.  # ------------------------------
063.  # App
064.  # ------------------------------
065.
066.  app = App("emoji match")
067.
068.  score = Text(app, text="0")
069.  timer = Text(app, text="30")
070.
071.  pictures_box = Box(app, layout="grid")
072.  buttons_box = Box(app, layout="grid")
073.
074.  pictures = []
075.  buttons = []
076.
077.  for x in range(0,3):
078.      for y in range(0,3):
079.          picture = Picture(pictures_box, grid=[x,y])
080.          pictures.append(picture)
081.
082.          button = PushButton(buttons_box, grid=[x,y])
083.          buttons.append(button)
084.
085.  result = Text(app)
086.
087.  setup_round()
088.
089.  app.repeat(1000, reduce_time)
090.
091.  app.display()
```

Maak een nieuwe functie die zal worden aangeroepen wanneer een van de emoji-buttons wordt ingedrukt. Deze zal 'correct' of 'incorrect' weergeven en `setup_round` aanroepen om de volgende set emoji's te maken.

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

Figuur 3 Met box voor score en timer.


```
        setup_round()
```

De onjuiste emoji-buttons zullen False doorgeven aan de functie `match_emoji`; de identieke emoji zal True doorgeven.

Update de functie `setup_round` zodat alle 'incorrecte' buttons de functie `match_emoji` aanroepen.

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

De methode `update_command` stelt de functie in die wordt aangeroepen wanneer de knop wordt ingedrukt. De `args`-lijst `[False]` wordt gebruikt als de parameters voor de functie `match_emoji`.

Update tot slot het commando voor de overeenkomende knop, zodat het `match_emoji` aanroept, maar deze keer True als argument doorgeeft.

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

Je code zou nu moeten lijken op **emoji3.py** (vorige pagina). Speel het spel. In elke ronde zal er een identieke emoji zijn – druk op de bijbehorende afbeeldingsknop. Had je het goed?

Score en timer toevoegen

Op dit moment gaat het spel eeuwig door (of tot de emoji's in de lijst op zijn). Voeg een score toe en een timer die aftelt tot het einde van het spel om het uitdagender te maken.

Maak in de sectie app twee Text-widgets om de score en de timer te tonen.

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

Andere afbeeldingen gebruiken

Het emoji match-spel gebruikt afbeeldingsknoppen om de gebruiker te laten kiezen welke emoji overeenkomt. Je kunt van elke PushButton-widget een afbeeldingsknop maken door de parameter image in te stellen; bijvoorbeeld:

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

De knop wordt geschaald naar de grootte van je afbeelding. Het type afbeelding dat je kunt gebruiken wordt bepaald door je besturingssysteem en hoe je guizero hebt geïnstalleerd, hoewel elke installatie GIF-afbeeldingen zal ondersteunen. Om te zien welke afbeeldingsbestanden door jouw installatie ondersteund worden, kun je het volgende uitvoeren

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

Je kunt meer te weten komen over de ondersteuning van afbeeldingen in guizero op **lawsie.github.io/guizero/images**.

De timer is ingesteld op `"30"`, wat het aantal seconden in elke ronde zal zijn.

Wijzig de functie `match_emoji` om 1 toe te voegen aan of af te trekken van de score van de speler.

```
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
```

Om de timer te maken, maak je gebruik van een functie van guizero waarmee je de applicatie kunt vragen om iedere seconde een functie aan te roepen.

Maak een functie die de waarde van de timer met 1 zal verminderen.

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

Voordat de app wordt weergegeven, gebruik je de functie `app.repeat()` om de `reduce_time` functie elke seconde (1000 milliseconden) aan te roepen.

```
app.repeat(1000, reduce_time)

app.display()
```

Als je nu je spel draait, zul je zien dat de timer aftelt vanaf 30. Helaas zal hij blijven aftellen voorbij 0 en nooit stoppen.

Update de functie `reduce_time` om te controleren of de timer minder dan nul is en stop dan het spel.

09-emoji-match.py

Taal: **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()
```

Als de timer minder dan 0 is, wordt het bericht 'game over' weergegeven en worden de widgets van het spel verborgen, zodat de gebruiker niet langer kan spelen.

Zie **emoji4.py** (vorige pagina) voor een beeld van hoe je code er nu uit zou moeten zien. Voer hem uit en speel het emoji match spel. Daag een vriend of familielid uit voor een spelletje.

Misschien wil je de score en timer widgets in een box zetten zodat ze netter op te maken zijn (**Figuur 3**) – zie de volledige listing **09-emoji-match.py** om te zien hoe je dit moet doen.

Deel 09

GUI's maken met Python: Paint

Maak een grappig tekenspel.

Je gaat een eenvoudig programma bouwen waarmee je kunt tekenen met lijnen en vormen (Figuur 1). Je zult je tekenprogramma in vier stappen maken:

- stippen tekenen die de muis volgen
- lijnen trekken tussen de stippen
- toevoegen van selectie voor kleuren en lijndikte
- vormen tekenen

Merk op dat je je programma vorm kunt geven zoals je maar wilt - het hoeft er niet zo uit te zien als dit.

Stippen tekenen

De eerste stap is het maken van een eenvoudig programma dat gebruik maakt van de Drawing-widget en de event `when_mouse_dragged` zal gebruiken om stippen (of ovalen op het scherm) te tekenen. Voeg de widgets toe in de Imports-sectie van je verder lege programma:

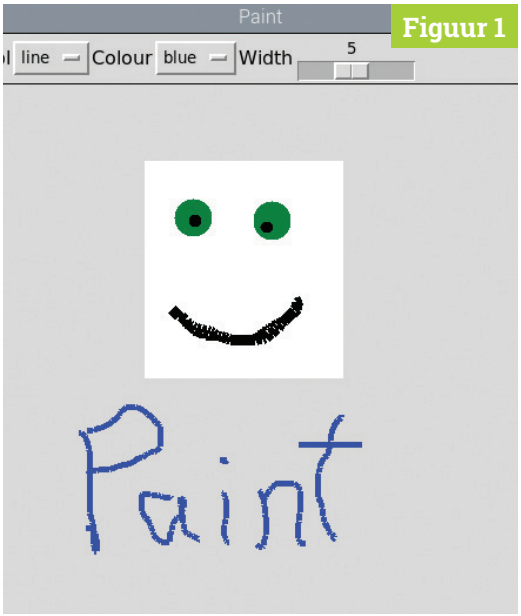
```
from guizero import App, Drawing
```

Maak een nieuwe functie:

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

Voeg deze code toe aan de App-sectie:

```
app = App("Paint")

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



Figuur 1

▲ **Figuur 1** Het voltooide spel.

```
painting.when_mouse_dragged = draw

app.display()
```

Je code zou moeten lijken op **paint1.py** (zie volgende pagina). De Drawing-widget vult alle beschikbare ruimte op het venster. Wanneer de muis over de tekening wordt gesleept, wordt de functie draw aangeroepen, die ovalen tekent op de tekening.

De functie draw wordt aangeroepen telkens wanneer een event wordt opgewekt. Het event dat de x- en y-positie van de muis bevat, wordt als variabele aan de functie doorgegeven.

Er is echter een probleem. Tenzij je de muis heel langzaam beweegt, tekent het programma een serie stippen, geen ononderbroken lijn (**Figuur 2**). Dit is geen erg goed penseel! Er zitten gaten tussen de stippen omdat er niet voor elke pixel waar de muis overheen gaat een event wordt opgeroepen.

Lijnen tussen de stippen

Om dit probleem op te lossen, passen we het programma aan om lijnen tussen de stippen te tekenen. Op die manier zal de lijn die gemaakt wordt ononderbroken zijn en meer lijken op een pen of penseel.

Je moet een `when_left_button_pressed` event gebruiken om de positie op te slaan van waar de lijn begint. Teken vervolgens een rechte lijn tussen waar de lijn begint en de volgende positie waar de muis naartoe werd gesleept. Maak een nieuwe functie die zal worden aangeroepen wanneer de muis wordt ingedrukt:

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

Voeg dit toe aan de App-sectie:

```
    painting.when_left_button_pressed =
start
```

Voeg nu de Picture-widget toe aan je Imports:

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

De positie van waar de lijn begint wordt opgeslagen in de variabele `last_event`.

Pas de `draw`-functie aan om een lijn te trekken tussen waar de lijn begint en waar de muis naartoe is gesleept.

```
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
```

> Je gaat het programma veranderen om lijnen tussen de stippen te trekken

paint1.py

DOWNLOAD DE VOLLEDIGE CODE:
magpi.cc/guizerocode

Taal: **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()
```

Door de `last_event` variabele te updaten naar de huidige positie van de muis, zal de volgende keer dat de muis wordt gesleept, een nieuwe lijn worden getekend tussen dit punt en het volgende. Je programma zou eruit moeten zien als **paint2.py**. Test het en controleer of je penseel nu goed werkt.

Verander de lijndikte en kleur

Je hebt maar één kleur en dikte voor je penseel, wat de tekeningen die je kunt maken beperkt. Pas nu de GUI aan zodat je verschillende kleuren en lijndiktes kunt kiezen.

Voeg twee widgets toe aan de GUI die een kleur en dikte van de lijn vastleggen.

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

Voeg deze regels toe aan de App-sectie:

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

Misschien wil je ook de achtergrondkleur van je tekening anders maken. Ook hebben we een Combo en een Slider gebruikt, maar je zou verschillende widgets kunnen kiezen.

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

Test je code, die nu lijkt op **paint3.py** (volgende pagina), en je kunt nu de kleur en lijndikte selecteren.

Vormen tekenen

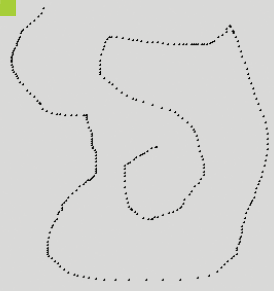
We gaan de paint-applicatie uitbreiden zodat je gevulde rechthoeken kunt tekenen. Wanneer je de muisknop indrukt, verschijnt de rechthoek en deze groeit als je de muis over het scherm sleept. Wanneer je de muisknop loslaat, wordt de rechthoek op het scherm getekend.

Hiervoor moet je het programma zo aanpassen dat het continu rechthoeken tekent en verwijdert totdat de muisknop wordt losgelaten. Laten we een widget toevoegen aan je GUI zodat je kunt kiezen of je een lijn of een rechthoek wilt tekenen. Voeg dit toe aan de App-sectie:

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

Wijzig de draw-functie zodat alleen lijnen worden getekend als de optie `"line"` is geselecteerd.

Figuur 2



▲ **Figuur 2** Niet zo'n goed penseel.

paint2.py

▶ Taal: **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()
```

> Je hebt maar één kleur en dikte voor je penseel, wat de tekening die je kunt maken beperkt

Ovalen toevoegen

Toen je de Paint-applicatie voor het eerst maakte, gebruikte je ovalen om stippen over het scherm te tekenen. Kun je je programma aanpassen om opnieuw ovalen te tekenen, gebruikmakend van een vergelijkbaar proces als bij het tekenen van rechthoeken? Hint: zie de listing **10-paint.py**, die ook de gereedschappen vorm geeft en ze netjes uitlijnt in een kader.

De Drawing-widget ondersteunt ook het tekenen van driehoeken en veelhoeken. Bekijk de documentatie (**lawsie.github.io/guizero/drawing**) en ontdek hoe je deze functie kunt gebruiken om andere vormen te maken.

paint3.py

Taal: **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

Taal: **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()
```

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

Test je programma om er zeker van te zijn dat de lijn nog steeds werkt en er niets gebeurt wanneer `"rectangle"` is geselecteerd.

Maak twee nieuwe variabelen om de eerste event en de laatste getekende vorm bij te houden wanneer de muisknop wordt ingedrukt.

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

Deze variabelen zullen worden gebruikt bij het tekenen en verwijderen van de rechthoek voordat de muisknop wordt losgelaten.

> Het programma zal continu een rechthoek tekenen en die weer verwijderen

```
    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
```

Het programma tekent continu een rechthoek, verwijdert deze en tekent hem opnieuw totdat je de knop loslaat.

Je volledige programma zou er uit moeten zien als **paint4.py**. Veel plezier bij het uitproberen – welke tekeningen kun jij ermee maken?

Aangepaste gebeurtenissen

Om je tekenapplicatie te laten reageren op de muispositie, heb je aangepaste events gebruikt. De events werken op een vergelijkbare manier als de normale widget-commando parameters, in die zin dat je ze instelt op een functie, die wordt aangeroepen wanneer dat event zich voordoet.

Wanneer je functie wordt aangeroepen, wordt er een variabele doorgegeven die informatie bevat over het event dat zich heeft voorgedaan, zoals de x- en y-coördinaten van de muis. De meeste widgets, inclusief de App zelf, ondersteunen de volgende events:

- wanneer geklikt – `when_clicked`
- wanneer de linkermuisknop wordt ingedrukt – `when_left_button_pressed`
- wanneer de linkermuisknop wordt losgelaten – `when_left_button_released`
- wanneer de rechtermuisknop wordt ingedrukt – `when_right_button_pressed`
- wanneer de rechtermuisknop wordt losgelaten – when_right_button_released
- wanneer een toets wordt ingedrukt – `when_key_pressed`
- wanneer een toets wordt losgelaten – `when_key_released`
- wanneer de muis een widget binnengaat – `when_mouse_enters`
- wanneer de muis een widget verlaat – `when_mouse_leaves`
- wanneer de muis over een widget wordt gesleept – `when_mouse_dragged`

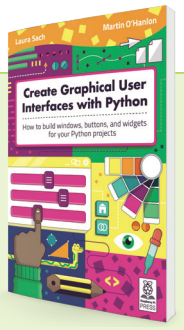
Deze events kun je gebruiken om je GUI's interactiever te maken.

10-painting.py

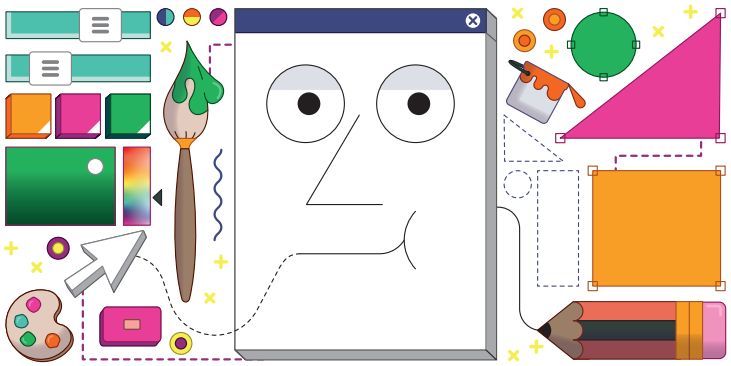
Taal: **Python 3**

```
# 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



Meer tutorials over hoe je je eigen GUI's kunt maken met guizero kun je vinden in ons boek, *Create Graphical User Interfaces with Python*. De 156 pagina's staan vol met essentiële informatie en een reeks van spannende projecten.
magpi.cc/pythongui



Deel 10

GUI's maken met Python: Stopframeanimatie

Bouw je eigen stopframeanimatie GIF-tool.

Dit project gebruikt een Raspberry Pi Camera Module en guizero om een stopframeanimatieprogramma te maken (Figuur 1).
Om dit project uit te voeren, heb je een Raspberry Pi nodig met de officiële Camera Module (of High Quality Camera). Als je hulp nodig hebt bij het aansluiten van de Camera Module, bekijk dan de handleiding 'Getting started with the Camera Module' op **rpf.io/picamera**.

Zorg dat guizero is geïnstalleerd met de optionele 'images'-functies, die je kunt installeren door dit commando uit te voeren in de terminal:

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

Dit project is opgedeeld in fasen:

1. Een foto nemen met de camera en deze weergeven op een GUI
2. Meerdere foto's nemen en ze opslaan in een GIF
3. De gebruiker toestaan de GIF te wijzigen
4. De GUI opruimen

Een foto nemen

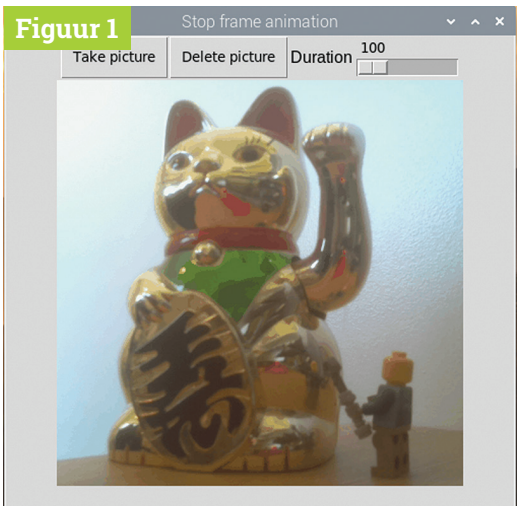
Begin met het maken van dit programma.

```
# 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()
```

Let op: hoe hoger de resolutie, des te langer de verwerkingstijd. 400×400 is klein maar zeer snel te verwerken.

De GUI bevat een PushButton en een Picture. Wanneer de knop wordt ingedrukt, wordt de functie `capture_image` aangeroepen. Deze functie gebruikt de camera om een beeld vast te leggen en op te slaan als **frame.jpg**. De foto wordt dan getoond in de Picture-widget.

Test het programma (**stopframe1.py**, volgende pagina). Als je op de knop 'Take picture' klikt, moet het beeld worden weergegeven in de GUI (**Figuur 2**).



▲ **Figuur 1** Een eenvoudige stopframeanimatie.

Meerdere foto's maken en opslaan in een GIF

Een animatie bestaat uit meerdere foto's, frames genoemd. Elk frame in de animatie zal iets anders zijn dan het vorige en wanneer ze samen snel na elkaar worden afgespeeld, zal de animatie lijken te bewegen.

In deze stap verander je je GUI om een lijst bij te houden van alle opgenomen frames en gebruik je PIL (Python Imaging Library) om de frames op te slaan als een geanimeerde GIF die zal worden weergegeven in de viewer.

Aan het begin van je programma, importeer je de Image-module uit PIL:

```
from PIL import Image
```

Maak een lijst om de frames van je animatie in op te slaan:

```
frames = []
```

Om bij te houden hoeveel frames er zijn genomen, importeer je een Text-widget, voeg je die toe aan je app, en zet je die op 0.

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

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

Telkens wanneer een nieuw beeld wordt vastgelegd, moet je het openen en toevoegen aan je lijst met frames:

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

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

De `len` (lengte) van de `frames`-lijst wordt dan gebruikt om de tekst in `total_frames` bij te werken. Je programma zou er nu ongeveer als **stopmotion2.py** uit moeten zien. Test het en zorg ervoor dat het aantal frames toeneemt elke keer als je een foto maakt.

Opslaan als GIF

Je kunt PIL gebruiken om alle frames op te slaan als één geanimeerde GIF. Maak een nieuwe functie `save_animation` om de frames op te slaan als **animation.gif**.

stopframe1.py

DOWNLOAD DE VOLLEDIGE CODE:
magpi.cc/guizerocode

Taal: **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()
```

Er gebeurt hier een heleboel, maar door de code uit te splitsen zie je hoe dit werkt. Als het aantal frames in de lijst groter is dan 0, wordt de viewer getoond, anders blijft hij verborgen.

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

De frames worden vervolgens opgeslagen in een bestand genaamd **animation.gif**. Het eerste frame (`frames[0]`) wordt opgeslagen, de resterende frames (`frames[1:]`) worden toegevoegd, en alles wordt opgeslagen in de geanimeerde GIF.

▲ **Figuur 2** Een foto maken.

stopframe2.py

▶ Taal: **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()
```

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

Animatie.gif wordt dan getoond in de viewer.

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

Roep de functie `save_animation` op aan het *einde* van de functie `capture_image` om de animatie te maken en weer te geven.

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

    save_animation()
```

Je code zou nu vergelijkbaar moeten zijn met **stopframe3.py** (volgende pagina). Test het uit.

Verwijder het laatste frame

Als je nu een fout maakt tijdens het maken van je geanimeerde GIF, moet je weer van voor af aan beginnen.

Je zou je GUI zo moeten aanpassen dat het laatst gemaakte frame verwijderd kan worden, zodat als er iets fout gegaan is, je het ongedaan kunt maken.

Maak een nieuwe functie die het laatste frame uit de lijst verwijdert, de gewijzigde animatie opslaat en vervolgens weergeeft.

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

    save_animation()
```

De lengte van de `frames`-lijst wordt gecontroleerd voordat geprobeerd wordt het laatste item eruit te gooien. Er zou een foutmelding verschijnen als je probeert een item uit een lege lijst te gooien.

Voeg een PushButton toe aan de GUI om de functie `delete_frame` aan te roepen, door deze code toe te voegen:

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

Opmerking: je kunt de GUI ook zo aanpassen dat je elk frame kunt verwijderen, niet alleen het laatste.

stopframe3.py

Taal: **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()
```

Veranderen van de timing

Elk frame wordt weergegeven gedurende de standaard duur van 100 milliseconden. Zet een Slider-widget in je GUI om de tijdsduur te kunnen veranderen.

Voeg hem toe aan de lijst van imports.

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

Maak dan de widget in de app.

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

De `start` en `end` parameters zijn de minimum- en maximumtijden die je kunt instellen voor de duur van het frame.

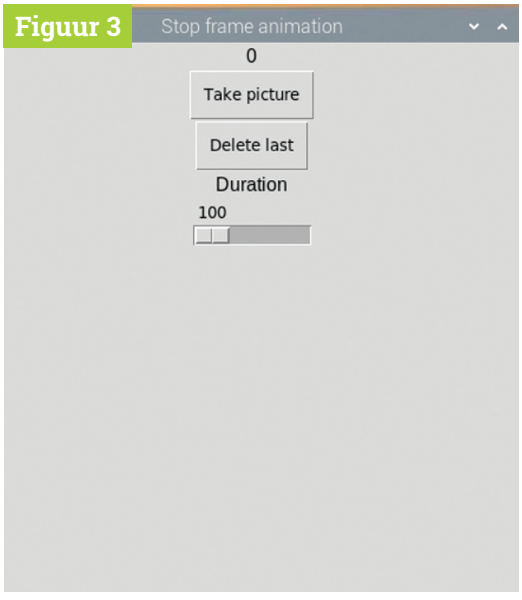
Telkens als de schuifbalk verandert, zal de functie `save_animation` worden uitgevoerd.

Breid de functie `save_animation` uit om de tijdsduurwaarde te gebruiken bij het opslaan van de GIF.

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

> Elk frame wordt weergegeven gedurende de standaardduur van 100 milliseconden

Figuur 3



▲ **Figuur 3** Bedieningselementen bovenaan gestapeld.

stopframe4.py

Taal: **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()
```

Je code zou nu moeten lijken op **stopmotion4.py**. Probeer het uit.

Lijn de controls uit

Op dit moment nemen de controls veel ruimte in beslag bovenin de GUI (**Figuur 3**).

Maak een Box en lijn deze uit met de bovenkant van de GUI om de controls in te plaatsen, door hem eerst toe te voegen aan de imports.


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

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

Pas de widgets aan zodat ze in de controls-box staan en zet de `align` parameter op `"left"`. Bijvoorbeeld:

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

Door widgets links uit te lijnen in de box, komen ze naast elkaar te staan.

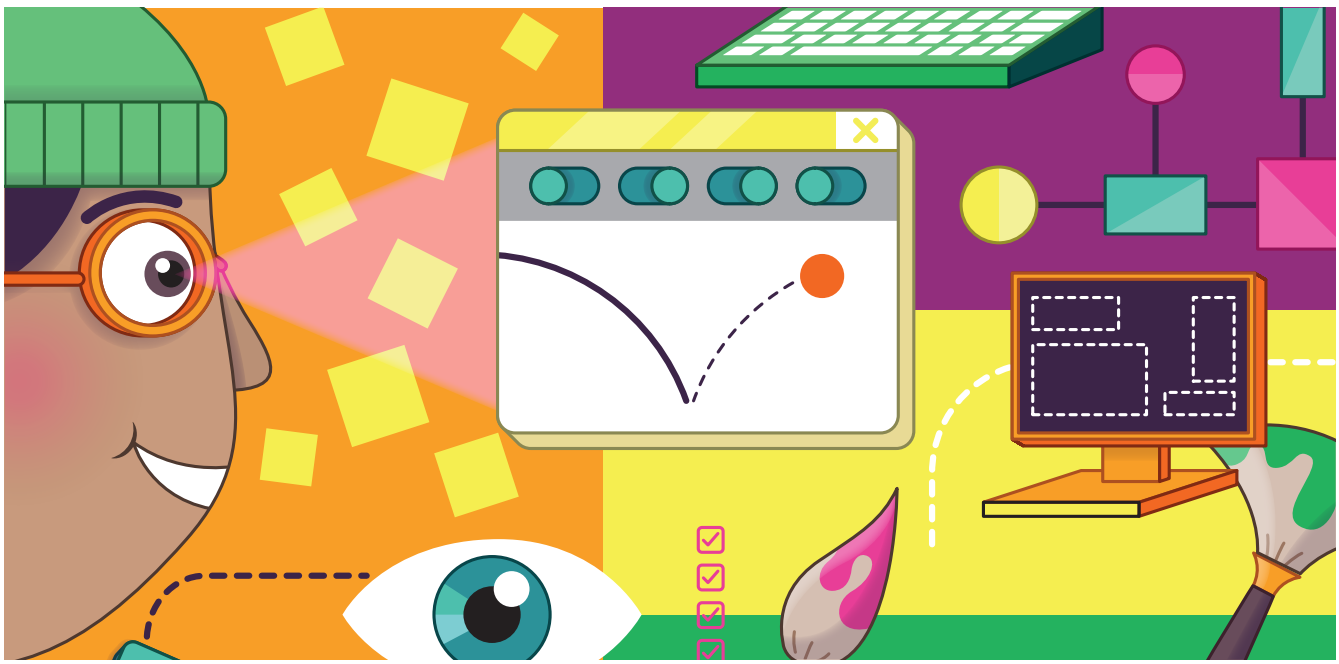
Herhaal dit voor de rest van de controls zodat ze allemaal in de bovenste box komen en naast elkaar worden uitgelijnd.

Je hele programma zou er nu ongeveer zo uit moeten zien als **11-stop-motion.py** (volgende pagina).

11-stopmotion.py

Taal: **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()
```



Het officiële MagPi E-zine

Elke week dat u zich niet inschrijft voor ons Pi Community Journal mist u leuke Raspberry Pi-gerelateerde artikelen en projecten!

Dus, waarom nog langer wachten? Abonneer u vandaag nog op **www.magpi.nl/journal**

MagPi
www.magpi.nl Magazine