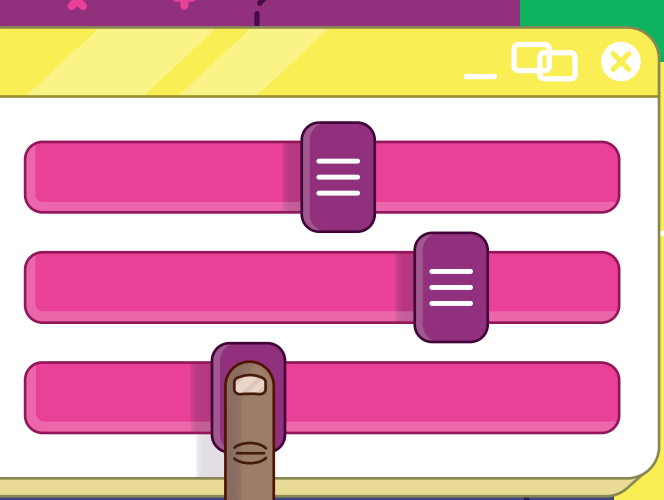
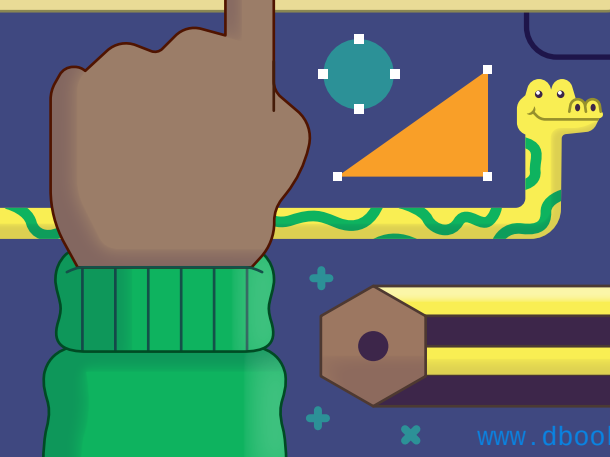
Laura Sach
Martin O'Hanlon

Create Graphical User Interfaces with Python

How to build windows, buttons, and widgets for your Python projects





www.dbooks.org

First published in 2020 by Raspberry Pi Trading Ltd, Maurice Wilkes Building,
St. John's Innovation Park, Cowley Road, Cambridge, CB4 0DS

Publishing Director: Russell Barnes • Editor: Phil King
Design: Critical Media
CEO: Eben Upton

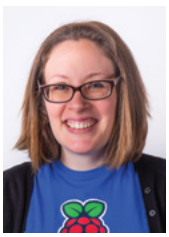
ISBN: 978-1-912047-91-8

The publisher and contributors accept no responsibility in respect of any omissions or errors relating to goods, products or services referred to or advertised in this book. Except where otherwise noted, the content of this book is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)

About the authors...

Martin O'Hanlon

Martin works in the learning team at the Raspberry Pi Foundation, where he creates online courses, projects, and learning resources. He contributes to the development of many open-source projects and Python libraries, including guizero. As a child, he wanted to be a computer scientist, astronaut, or snowboard instructor.



Laura Sach

Laura leads the A Level team at the Raspberry Pi Foundation, creating resources for students to learn about Computer Science. She somehow also manages to make cakes, hug cats, and wrangle a toddler.

Welcome!

This book will show you how to use Python to create some fun graphical user interfaces (GUIs) using the guizero library. The guizero library started with the belief that there must be an easier way for students in school to create Python GUIs. The library itself was born one day on a long train journey from Cambridge, as the side project of a secondary school teacher.

Guizero has grown significantly in terms of features, yet remained true to its original aim of being simple but flexible. It is a library for all beginners to create with, for teachers to scaffold learning with and for experts to save time with.

We hope that these projects and guizero brings you that little spark of excitement to your Python programs. That spark might be anything from a button that does something when you click it, to colours in your otherwise black and white Python programming, to a full multicoloured Waffle.

It turns out that with open-source software, even if you don't know how to get the whole way there, if you start, someone will help you. We are grateful to the many contributors who have put time and effort into creating guizero, and to the thousands of people who have used it in their projects. Enjoy your journey and be proud of your creations.

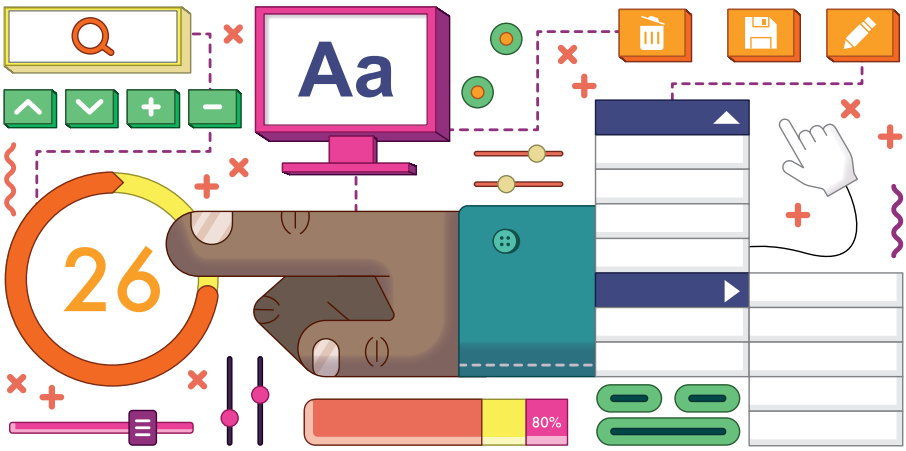
Laura and Martin

Contents

Chapter 1

Introduction to GUIs

How to install guizero and create your first app



WHAT YOU'LL NEED

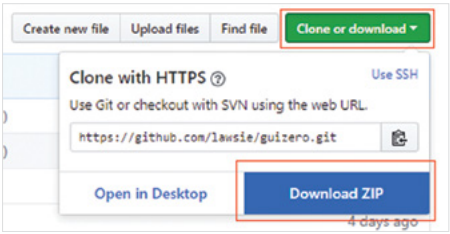
You will need a computer (e.g. Raspberry Pi, Apple Mac, Windows or Linux PC) and an internet connection for the software installation. You will also need the following software installed:

- **Python 3** (python.org) – see Appendix A
- **An IDE** (code editor), e.g.: IDLE (installed with Python 3), Thonny (thonny.org), Mu (codewith.mu), PyCharm (jetbrains.com/pycharm)
- **The guizero Python library** (lawsie.github.io/guizero)

A graphical user interface (GUI, pronounced 'gooey') is a way of making your Python programs easier to use and more exciting. You can add different components called 'widgets' to your interface, allowing lots of different ways for information to be entered in to the program and displayed as output. You might want to allow people to push a button, to display a piece of text, or even let them choose an option from a menu. In this book we will use the guizero library, which has been developed with the aim of helping beginners to easily create GUIs.

Python's standard GUI package is called tkinter, and is already installed with Python

on most platforms. The guizero library is a wrapper for tkinter – this means that it offers a much simpler way of using Python's standard GUI library.



An alternative way to install guizero is to download the zip file from GitHub

Installing guizero

You will need to install the guizero (**lawsie.github.io/guizero**) Python library to create the programs in this book. It is available as a Python package, which is reusable code you can download, install, and then use in your programs.

How you install of guizero will depend on your operating system and the permissions you have to control your computer.

If you have access to the command line / terminal, you can use the following command:

```
pip3 install guizero
```

Comprehensive installation instructions for guizero are available at **lawsie.github.io/guizero**, including options for installing when you have no administration rights to your computer and downloadable installations for Windows.

Hello World

Now that you have guizero installed, let's check that it's working and write a small 'hello world' app which is traditional for programmers to write as their first program when using a new tool or language.

AIMS OF GUIZERO

- Able to be used without installation
- Remove unnecessary code new learners find difficult to understand
- Sensible widget names
- Accessible to young children, but able be used for advanced projects
- Good-quality documentation with examples
- Generate helpful error messages

www.dbooks.org

Open up the editor where you will write your Python code. At the start of every guizero program, you will choose the widgets you need from the guizero library and import them. You only need to import each widget once, and then you can use it in your program as many times as you like.

At the top of the page, add this code to import the App widget:

```
from guizero import App
```

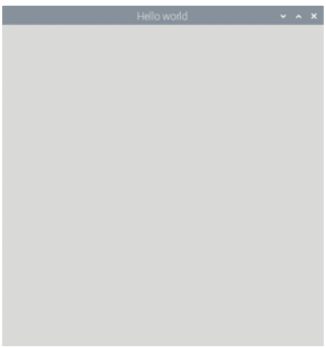


Figure 1 Your first guizero app

All guizero projects begin with a main window which is a container widget called an App. At the end of every guizero program, you must tell the program to display the app you have just created.

Add these two lines of code underneath the line where you imported the App widget:

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

Now save and run your code. You should see a GUI window with the title 'Hello world' (**Figure 1**). Congratulations, you've just created your first guizero app!

Adding widgets

Widgets are the things which appear on the GUI, such as text boxes, buttons, sliders, and even plain old pieces of text.

All widgets go between the line of code to create the App and the `app.display()` line. Here is the app you just made, but in this example we have added a Text widget:

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

Did you notice that there are two changes (**Figure 2**)? There is now an extra line of code to add the Text widget, and we have also added Text to the list of widgets to import on the very first line.

Let's look at the Text widget code in a bit more detail:

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

Just like any variable in Python, a widget needs a name. This one is called 'message'. Then we specify that we would like this to be a 'Text' widget. Inside the brackets are some parameters to tell the Text widget what it should look like. The first one, 'app', tells the Text where it will live. All widgets need to live inside a container widget. Most of the time your widgets will live directly inside an App, but you will discover later that there are also some other types of container widget you can put things in too. Finally, we tell the widget to contain the text "Welcome to the app".

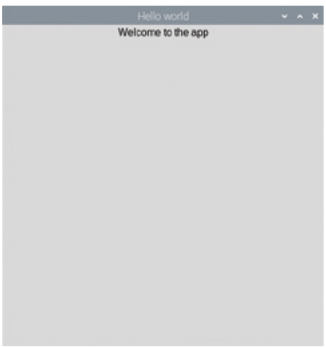


Figure 2 Add a text message

01-helloworld.py / Python 3

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

DOWNLOAD

magpi.cc/guizerocode

www.dbooks.org

Chapter 2

Wanted Poster

Use styled text and an image to create a poster

Now that you can create a basic GUI, let's make it look a bit more exciting. You can add text in different fonts, sizes and colours, change the background colour, and add pictures too. To practise all of this, let's create a 'Wanted' poster.

First of all, you need to start off by creating an app. In your editor, add this code to create the most basic app window:

```
from guizero import App

app = App("Wanted!")

app.display()
```

Save and run your code and you should see an app that looks like a plain grey square with the title 'Wanted!' at the top (**Figure 1**).

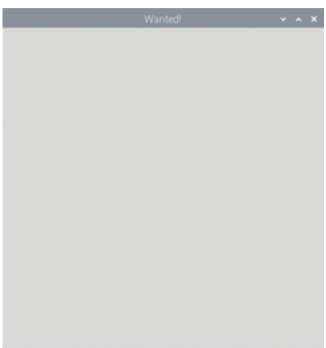


Figure 1 The basic app

Background colours

Let's make the background of the app a bit different. Traditionally, wanted posters look like they are made of parchment, so let's add a pale yellow colour instead as the background.

Find the line of code where you create the app. Immediately after this line of code, add one more line of code to modify the bg property of the window. In this case, bg is short for 'background' and will let us change the colour of the background. Now your code should look like this:

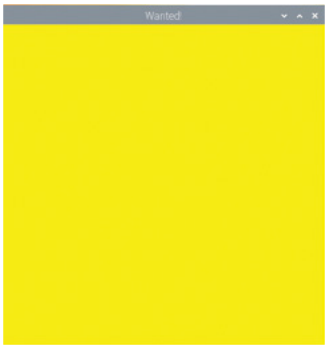


Figure 2 Background colour

```
from guizero import App

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

app.display()
```

This is called editing a property. In the code, you need to specify the widget you are talking about (app), the property you want to change (bg) and the value you want to change it to.

You might think this colour (**Figure 2**) is a bit too yellow, so let's look up the hex code of a different yellow colour. There are lots of websites where you can search for colours, for example you could try **htmlcolorcodes.com** (**Figure 3**).

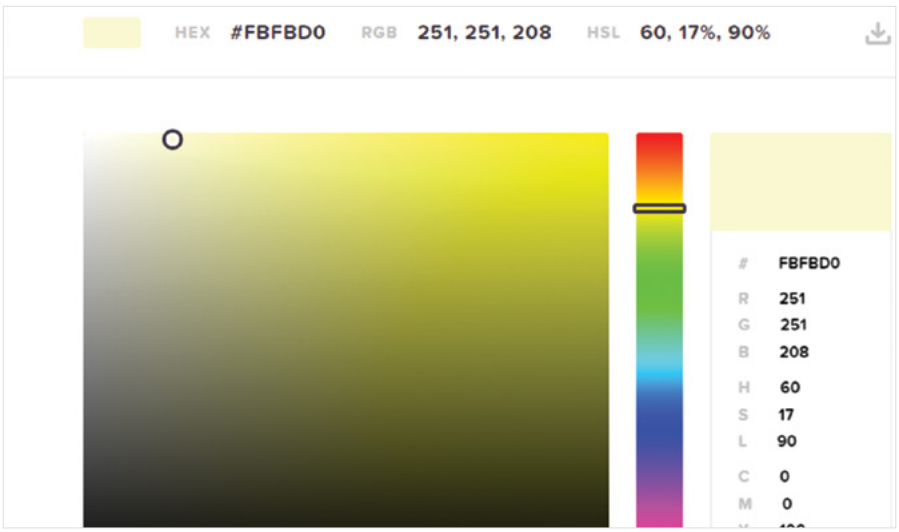


Figure 3 Selecting a shade on htmlcolorcodes.com

www.dbooks.org

When you have selected the colour you want, you will see its code displayed on the site either as hexadecimal (in this case #FBFBD0) or as RGB (251, 251, 208). You can use both of these formats for setting colours in guizero; for example, you could delete the code for making your background yellow and then try one of these options in your program:

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

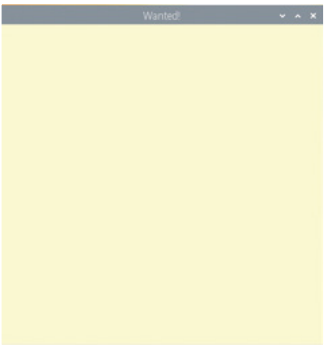


Figure 4 Pale background

Add some text

Your app should look something like **Figure 4**. Now let's add some text to the GUI. We will begin by adding the text that all good wanted posters need – the word 'Wanted'!

First, look for the line of code you already have where you imported the App.

```
from guizero import App
```

You need to import Text to be able to create a piece of text, so add it to the end of the list. Now the line looks like this:

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

Every time you want to use a new type of widget, add its name to the end of the list. There is no need to keep adding whole new lines of code: just stick with one list so that your program doesn't get too confusing.

Now that you can use text, let's add a piece of text. Remember that all widgets on the GUI must be added between the line of code where you create the App and the line of code where you display it. Your code should now look like this:

```
from guizero import App, Text

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

wanted_text = Text(app, "WANTED")

app.display()
```

Let's take a closer look at that line of code you just added.

```
wanted_text = Text(app, "WANTED")
```

Here, `wanted_text` is the name of the piece of text. This is so that we can talk about it later on in the code – think of it like a person's name. (You could even call your piece of text Dave if you want – the computer won't care!)

Inside the brackets we have two things. The second one, `"WANTED"`, is straightforward as it is the text we would like to display on the screen. The first is the container which controls this piece of text, which is called its 'master'. In this case we are saying that this text should be controlled by the app. When you first start creating GUIs, most of your widgets will have the app as their master, but there are other containers that can store widgets that you will learn about later on.

Change text size and colour

Uh oh, this text is pretty small (**Figure 5**). Let's change the `text_size` property in exactly the same way as you did when we changed the background colour of the app. Remember that you needed to specify three things:

1. The name of the widget
2. Which property to change
3. The new value to change it to

Figure 5 Text is too small

So, in this case you are going to specify the widget (`wanted_text`), the property to change (`text_size`) and the new value (`50`). Add one new line of code immediately under the line where you created the text, to change the property.

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

You now have larger text on your poster (**Figure 6**). See if you can now change the font of this text to something different. Which fonts are available depends on which operating system you are using, so here are some suggestions:

- Times New Roman
- Verdana
- Courier
- Impact

Figure 6 Larger text

www.dbooks.org

No 'wanted' poster would be complete without a picture, so let's add one. My poster is going to be for my cat, because she is always scratching things she shouldn't be.

Save a copy of the image you would like to use in the same folder as your GUI program. You can use images in other folders, but if you do you will have to provide the path to the image, so it's a lot easier to just store them in the same folder when you are starting out.

IMAGE MANIPULATION

Because guizero is a library for beginners and we wanted to make it as easy as possible to install, it does not come with the fancier image manipulation functions as these require an extra library called 'pillow'. You can always use non-animated GIF images on any platform, and PNG images on all platforms except Mac, so if you're not sure whether you have installed the extra image manipulation functions, stick to those image types.

Hopefully you're now getting used to adding widgets. Remember that they must always be imported at the top of the program, and then the widget created with a sensible name after the line of code where you create the App, but before the final `app.display()` line.

Add 'Picture' to the list of widgets to import at the start of the program.

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

Now create a Picture widget with two parameters: the app and the file name of the picture. This is the code we used because our picture was called **tabitha.png**.

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

Run your code (which should look like **02-wanted.py**) again and you should see the picture displaying below your text (**Figure 7**).

Now it's up to you to use your new GUI customisation skills to style your poster however you would like.

Figure 7 The finished poster

READING THE DOCS

You might be wondering how to find out what properties a particular widget has that you can change. Even if you are a complete beginner programmer, it is worth learning how to read documentation because it will let you use the full power of guizero and any other libraries you come across.

The guizero documentation can be found at **lawsie.github.io/guizero**. Once you are there, click on the widget you would like to change, and scroll down until you reach the properties section. For example, if you select 'Text' under the heading of widgets, you will see all of the properties of a piece of Text that you can possibly change. Documentation also often contains helpful snippets of code which show you how to use a particular property or method, so don't be scared of having a look through – you never know what you might learn!

02-wanted.py / Python 3

DOWNLOAD

magpi.cc/guizerocode

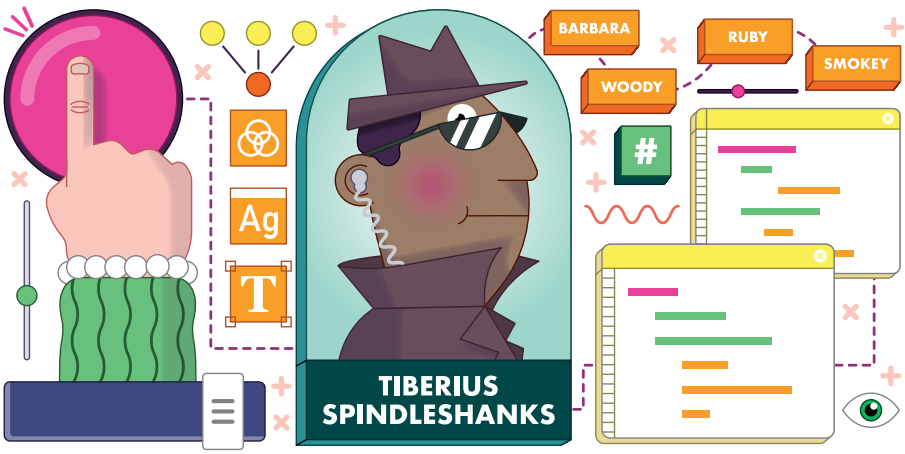
```
from guizero import App, Text, Picture

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

wanted_text = Text(app, "WANTED")
wanted_text.text_size = 50
wanted_text.font = "Times New Roman"

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

app.display()
```


www.dbooks.org

Chapter 3

Spy Name Chooser

Make an interactive GUI application



So far you've learnt how to customise your GUI with a variety of different options. It's now time to get into the really interactive part and make a GUI application that actually responds to input from the user. Who could resist pushing a big red button to generate a super secret spy name?

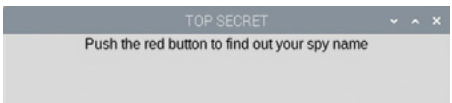


Figure 1 Displaying the text in a window

Since you already know how to create an app, why not go ahead and create a basic window and add some text if you like? Here is some code to get you started, and this code also includes some comments (the lines that start with a #) to help you structure your program:

```
# 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()
```

Run this code and you should see a window with the text (**Figure 1**).

Add a button

Let's go ahead and add a button to the GUI. Add PushButton to your list of imports so that you can use buttons. (Be careful to use a capital B!)

Underneath the Text widget, but before the app displays, add a line of code to create a button.

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

Your code should now look like **spy1.py** (page 22). Run it and no button will appear, but you'll see an error in the shell window:

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

This is because `choose_name` is the name of a command which runs when the button is pressed. Most GUI components can have a command attached to them. For a button, attaching a command means "when the button is pressed, run this command." A GUI program works differently to other Python programs you might have written because the order in which the commands are run in the program depends entirely on the order in which the user presses the buttons, moves the sliders, ticks the boxes or interacts with whichever other widgets you are using. The actual command is almost always the name of a function to run.

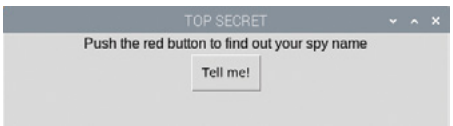


Figure 2 You now have a button

www.dbooks.org

Create a function

Figure 3 Text is output to the shell window

Let's write the function `choose_name` so your button has something it can do when it is pressed.

Look at your program and find the functions section. This is where you should write all of the functions which will be attached to GUI widgets, to keep them separate from the code for displaying the widget. Add this code in the functions section:

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

Your code should now look like **spy2.py**. The button will now appear (**Figure 2**). If you press the button, it may appear that nothing has happened, but if you look in your shell or output window, you will see that some text has appeared there (**Figure 3**).

Instructing your function to first print out some dummy text is a useful way of confirming that the button is activating its command function correctly when it is pressed. You can then replace the `print` statement with the actual code for the task you would like your button to perform.

Inside your `choose_name` function, type a # symbol in front of the line of code that prints "Button was pressed". Programmers call this 'commenting out', and what you have done here is told the computer to treat this line of code as if it were a comment, or in other words you have instructed the computer to ignore it. The benefit of commenting a line of code out instead of just deleting it is so that if you ever want to use that code again, you can easily make it part of your program again by removing the # symbol.

BIG RED BUTTON

At the moment, your button is not big or red! You used properties in the previous chapter to change the appearance of your text on the 'Wanted' poster, so can you use the properties of the PushButton widget to change the background colour and the text size?

Note that it may not bepossible to change the colour of a button on macOS, as some versions of the operating system will not allow you to do so, but you should still be able to alter the text size.

Add some names

Figure 4 Outputting a spy name

On a new line, add a list of first names. You can choose the names in your list and there can be as many names as you like, but make sure that each name is between quotes, and the names are each separated by a comma. A collection of letters, numbers, and/or punctuation between quotation marks is called a *string*, so we say that each name must be a string.

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

Now add a list of last names as well:

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

Now you will need to add a way of choosing a random name from each list to form your spy name. Your first job is to add a new import line in your imports section:

```
from random import choice
```

This tells the program that you would like to use a function called choice which chooses a random item from a list. Someone else has written the code which does this for you, and it is included with Python for you to use.

In your code for the `choose_name` function, just below your lists of names, add a line of code to choose your spy's first name, and then concatenate it together with the last name, with a space in between. Concatenate is a fancy word that means 'join two strings together' and the symbol in Python for concatenation is a plus (+).

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

Your code should now resemble **spy3.py**. Save and run it. When you press the button, you should see that a randomly generated spy name appears in your console or shell, in the same place where the original "Button was pressed" message showed up before (**Figure 4**).

www.dbooks.org

Put the name in the GUI

That's good, but wouldn't it be nicer if the spy name appeared on the GUI? Let's make another Text widget and use it to display the spy name.

In the widgets section, add a new Text widget which will display the spy name:

Figure 5 The finished spy name chooser

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

When you create the widget, you don't want it to display any text at all as the person won't have pressed the button yet, so you can set the text to "", which is called an 'empty string' and displays nothing. Inside your `choose_name` function, comment out the line of code where you print out the spy name.

Now add a new line of code at the end of the function to set the value of the name Text widget to the `spy_name` you just created. This will cause the Text widget to update itself and display the name.

```
name.value = spy_name
```

Your final code should be as in **03-spy-name-chooser.py**. Run it and press the button to see your spy name displayed proudly on the GUI (**Figure 5**).

You can press the button again if you don't like the name you are given, and the program will randomly generate another name for you.

spy1.py / Python 3

DOWNLOAD
magpi.cc/guizerocode

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

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

# 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!")

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

spy2.py / Python 3

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

# Functions -------------
def choose_name():
    print("Button was pressed")

# 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!")

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

spy3.py / 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()
```


www.dbooks.org

03-spy-name-chooser.py / 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()
```

www.dbooks.org

Chapter 4

Meme Generator

Create a GUI application which draws memes

Let's take the lessons you learnt from the previous chapters to create a GUI which draws memes. You will input the text and image name and your GUI will combine them into your own meme using the Drawing widget.

Start by creating a simple GUI with two text boxes for the top and bottom text. This is where you will enter the text which will be inserted over your picture to create your meme. Add this line to import the widgets needed.

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

Then add this code for the app:

```
app = App("meme")

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

app.display()
```

The meme will be created on a Drawing widget which will hold the image and text.

Create a meme

Add it to the GUI by inserting this code just before the `app.display()` line. The Drawing widget's height and width should be set to 'fill' the rest of the GUI.

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

The meme will be created when the text in the top and bottom text boxes changes. To do that, we will need to create a function which draws the meme.

The function should clear the drawing, create an image (we're using a photo of a woodpecker, but you can use any you want) and insert the text at the top and bottom of the image.

Remember when you used `name.value` to set the value of the Text widget with the spy name in Chapter 3? You can also use the value property to *get* the value of a Text widget, so in this case `top_text.value` means 'please get the value that is typed in the top_text box'.

```
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)
```

The first two numbers in `meme.image(0, 0)` and `meme.text(20, 20)` are the x, y co-ordinates of where to draw the image and text. The image is drawn at position `0, 0`, which is the top-left corner, so the image covers the whole of the drawing.

Finally, call your `draw_meme` function just before you display the app. Insert this code just before the `app.display` line:

```
draw_meme()
```

Your code should now look like **meme1.py**.

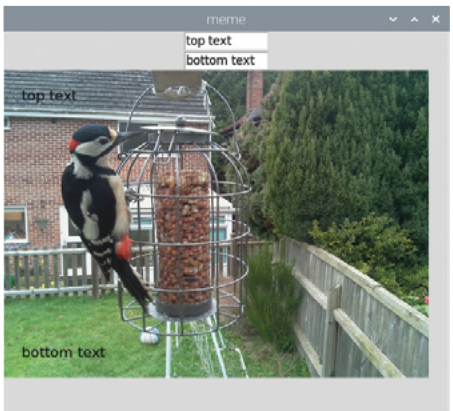


Figure 1 Meme with unstyled text

www.dbooks.org

If you run your app (**Figure 1**) and try changing the top and bottom text, you will notice that it doesn't update in the meme. To get this working, you will have to change your program to call the `draw_meme` function when the text changes, by adding a command to the two TextBox widgets to the app.

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

Your code should now look like that in **meme2.py**. Run it and update your meme by changing the top and bottom text.

You can then look of your meme by changing the color, size, and font parameters of the text. For example:

```
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",
    )
```

TIP

These lines of code were starting to get very long, so we have split them over a number of lines to make it easier to read. It doesn't affect what the program does, just how it looks.

Your code should now look like **meme3.py**. Try different styles until you find something you like (**Figure 2**).

Customise your meme generator

For a truly interactive meme generator, the user should be able to set the font, size, and colour themselves. You can provide additional widgets on the GUI to allow them to do this.

The number of options available for the colour and font are limited, so you could use a drop-down list, also known as a Combo, for this. The size could be set using a Slider widget.

Figure 2 Alter the fonts and colours

First, modify your import statement to include the Combo and Slider widgets.

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

After you have created your TextBox widgets for the top and bottom text, create a new Combo widget so the user can select a colour.

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

The `options` parameter sets what colours the user can select from the Combo. Each colour is an element in a list. You can add any other colours you want to the list.

The options are displayed in the order in which you put them in the list. The first option is the default, which is displayed first. If you want to have a different option as the default, you can do it using the `selected` parameter, e.g. `"blue"`.

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

Now your user can select a colour. Next, you need to change the `draw_meme` function to use Combo's value when creating the text in your the meme. For example:

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

Do the same for the bottom-text block of code. Your program should now resemble **meme4.py**.

Following the steps above, add a second Combo to your application so the user can select a font from this list of options: `["times new roman", "verdana", "courier", "impact"]`. Remember to change the `draw_meme` function to use the `font` value when adding the text.

Create a new Slider widget to set the size of the text your user wants.

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


www.dbooks.org

The range of the slider is set using the start and end parameters. So, in this example, the smallest text available will be 20 and the largest 40.

Modify the `draw_meme` function to use the value from your size slider when creating the meme's text.

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

Your code should now resemble that in **04-meme-generator.py**. Try running it and you should see something like **Figure 3**.

Can you change the GUI so that the name of the image file can be entered into a TextBox or perhaps selected from a list in a Combo? This would make your application capable of generating memes with different images too.

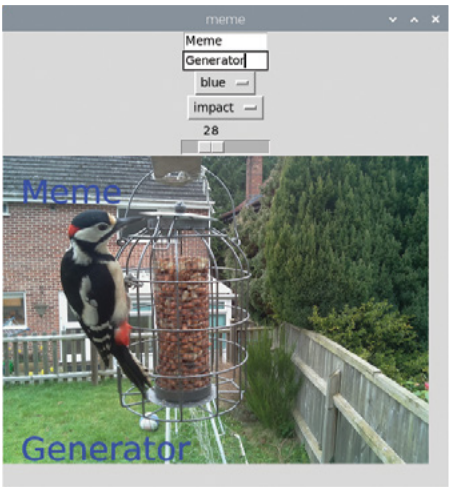


Figure 3 The finished meme generator

DRAWING WIDGET

The Drawing widget is really versatile and can be used to display lots of different shapes, patterns, and images.

To find out more about the Drawing widget, see Appendix C, or take a look at the online documentation: **lawsie.github.io/guizero/drawing**.

meme1.py / Python 3

DOWNLOAD
magpi.cc/guizerocode

```
# 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 / 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)
```


www.dbooks.org

meme2.py (cont.) / Python 3

```
# 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()
```

meme3.py / 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,
        color="orange",
        size=40,
        font="courier")
    meme.text(
        20, 320, bottom_text.value,
        color="blue",
        size=28,
        font="times new roman",
        )
```

meme3.py (cont.) / Python 3

```
# 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()
```

meme4.py / Python 3

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

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


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

def draw_meme():
    meme.clear()
    meme.image(0, 0, "woodpecker.png")
    meme.text(
        20, 20, top_text.value,
        color=color.value,
        size=40,
        font="courier")
    meme.text(
        20, 320, bottom_text.value,
        color=color.value,
        size=28,
        font="times new roman",
        )


# App -------------------
```


www.dbooks.org

meme4.py (cont.) / Python 3

```
app = App("meme")

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

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

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

draw_meme()

app.display()
```

04-meme-generator.py / Python 3

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

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


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

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

04-meme-generator.py / Python 3

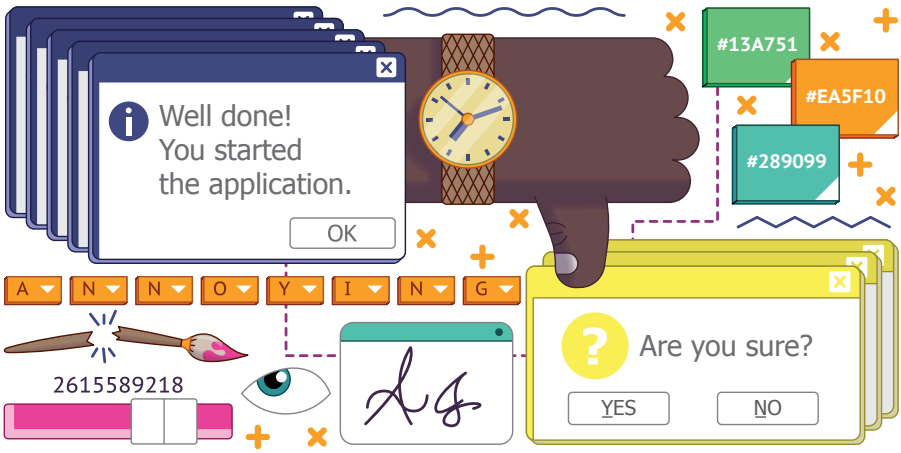
```
# App --------------------

app = App("meme")

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

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

font = Combo(app,
             options=["times new roman", "verdana", "courier",
"impact"],
             command=draw_meme)

size = Slider(app, start=20, end=50, command=draw_meme)

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

draw_meme()

app.display()
```


www.dbooks.org

Chapter 5

World's Worst GUI

Learn good GUI design by doing it all wrong first!



ts time to really go to town with your GUIs and experiment with different widgets, colours, fonts, and features. Like most experiments, it's likely that you won't get it right first time! In fact, you are going to explore the wrong way to approach creating your GUI.

It's hard to read

The right choice of GUI colour and font are important. It's important that the contrast between background and text colour ensure that your GUI is easily readable. What you shouldn't do it is use two very similar colours.

Import the widgets at the top of the code:

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

Create an app with a title:

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

```
app.display()
```

Experiment by changing the colours, font, and text size (see **worst1.py** listing, page 41). My choices are not the best!

```
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")
```

It's important that text on a GUI also stays around long enough to be read. It certainly shouldn't disappear or start flashing.

All widgets in guizero can be made invisible (or visible again) using the `hide()` and `show()` functions. Using the `repeat` function in guizero to run a function every second, you can make your text hide and show itself and appear to flash.

Create a function which will hide the text if it's visible and show it if it's not:

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

Before the app is displayed, use `repeat` to make the `flash_text` function run every 1000 milliseconds (1 second).

```
app.repeat(1000, flash_text)

app.display()
```

Your code should now look like **worst2.py**. Test your app: the title text should flash, appearing and disappearing once every second.

The wrong widget

Using an appropriate widget can be the difference between a great GUI and one which is completely unusable.

Which widget would you use to enter a date? A TextBox? Multiple Combos? A TextBox would be more flexible but would require validation and formatting. Multiple Combos for year, month, and day wouldn't require validating but would be slower to use.

www.dbooks.org

Figure 1 A slider to set date and time

Figure 2 Combos to choose letters

Using a Slider to set a date and time (**Figure 1**), as in the **worst3.py** code example, is not a great idea, though.

The Slider widget returns a number between 0 and 999,999,999. This is the number of seconds since 1 January 1970. The function `ctime()` is used to turn this number into a date and time.

Getting text from your user is simple: a TextBox or a multi-line TextBox should fulfil all your needs. Is it too simple, though. Does this require too much typing?

What about the user who just wants to use a mouse? Perhaps a series of Combos each containing all the letters in the alphabet would be better (**Figure 2**)?

Start by importing the guizero widgets and `ascii_letters`.

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

`ascii_letters` is a list containing all the 'printable' ASCII characters which you can use as the options for the Combo.

Create a single Combo which contains all the letters and displays the app.

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

app.display()
```

Your program should now resemble **worst4.py**. Running it, you will see a single Combo which contains all the letters plus a space and is aligned to the left of the window.

To get a line of letters together, you could continually add Combo widgets to your app, e.g.:

```
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")
```

By aligning each Combo widget to the left, the widgets are displayed next to each other against the left edge.

Alternatively, you could use a `for` loop, create a list of letters, and append each letter to the list, as shown in **worst5.py**.

Try both these approaches and see which you prefer. The `for` loop is more flexible as it allows you to create as many letters as you like.

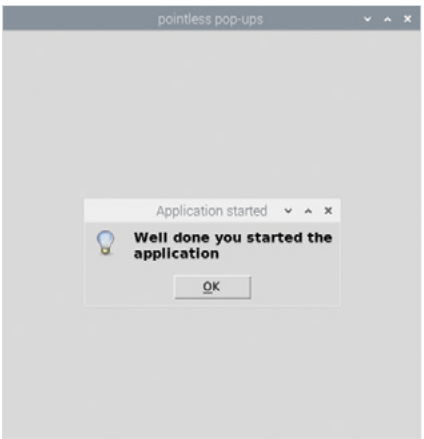


Figure 3 Pointless pop-up

Pop-ups

No terrible GUI would be complete without a pop-up box. guizero contains a number of pop-up boxes, which can be used to let users know something important or gather useful information. They can also be used to irritate and annoy users!

First, create an application which pops up a pointless box at the start to let you know the application has started.

```
from guizero import App

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

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

app.display()
```

Running your application, you will see that an 'info' box appears (**Figure 3**). The first parameter passed to info is the title of the window; the second parameter is the message.

You can change the style of this simple pop-up by using `warn` or `error` instead of `info`.

Pop-up boxes can also be used to get information from the user. The simplest is a `yesno` which will ask the user a question and get a True or False response. This is useful if you want a user to confirm before doing something, such as deleting a file. Perhaps not every time they press a button, though!

Import the PushButton widget into your application:

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

Create a function which uses the `yesno` pop-up to ask for confirmation.

www.dbooks.org

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

Add the button to your GUI which calls the function when it is pressed.

```
button = PushButton(app, command=are_you_sure)
```

Your code should now resemble **05-worlds-worst-gui.py**. When you run the application and press the button, you will see a pop-up asking to you confirm with a Yes or No (**Figure 4**).

You can find out more about the pop-up boxes in guizero at **lawsie.github.io/guizero/alerts**.

How about combining all of these 'features' into one great GUI?

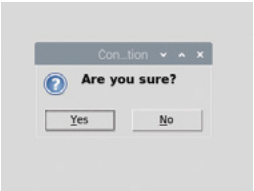


Figure 4 Yes, we're sure!

WINDOW WIDGET

Pop-up boxes can be used to ask users questions, but they are really simple.

If you want to do show additional information or ask for supplementary data, you could use the Window widget to create multiple windows.

Window is used in a similar way to App and has many of the same functions.

```
from guizero import App, Window

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

app.display()
```

You can control whether a Window is on screen using the `show()` and `hide()` methods.

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

An app can be made to wait for a window to be closed after it has been shown, by passing True to the `wait` parameter of `show`. For example:

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

You can find out more about how to use multiple windows in the guizero documentation: **lawsie.github.io/guizero/multiple_windows**.

worst1.py / Python 3

DOWNLOAD
magpi.cc/guizerocode

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

from guizero import App, Text


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

app = App("its all gone wrong", bg="dark green")

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

app.display()
```

worst2.py / Python 3

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

from guizero import App, Text


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

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


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

app = App("its 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()
```


www.dbooks.org

worst3.py / Python 3

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

from guizero import App, Slider, Text
from time import ctime


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

def update_date():
    the_date.value = ctime(date_slider.value)


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

app = App("Set the date with the slider")
the_date = Text(app)
date_slider = Slider(app, start=0, end=999999999, command=update_
date)

app.display()
```

worst4.py / Python 3

```
# Imports ---------------
from guizero import App, Combo
from string import ascii_letters


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

app = App("Enter your name")

a_letter = Combo(app, options=" " + ascii_letters, align="left")

app.display()
```

worst5.py / Python 3

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

from guizero import App, Combo
from string import ascii_letters


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

app = App("Enter your name")

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

app.display()
```

05-worlds-worst-gui.py / Python 3

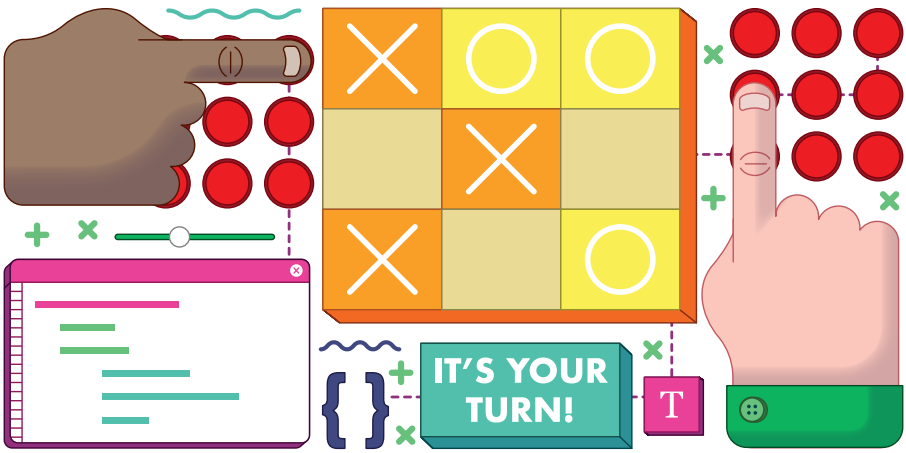
```
from guizero import App, PushButton

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

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

button = PushButton(app, command=are_you_sure)

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

app.display()
```


www.dbooks.org

Chapter 6

Tic-tac-toe

Use your GUI to control a simple game



Now that you have learnt how to make a basic GUI, let's add some more programming logic behind the scenes to make your GUI work as the means of controlling a game of tic-tac-toe (also known as noughts and crosses).

Create a new file with the following code:

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

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

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

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

app.display()
```

Create the board

Let's begin by creating the widgets which will make up the game board. A traditional tic-tac-toe board looks like the one shown in **Figure 1**.

You'll use buttons to represent each of the positions on the board, so that the player can click on one of the buttons indicating where they would like to move. To be able to lay out the buttons on a grid, let's create a new type of guizero widget called a Box.

A Box is a container widget. This means that it is used for containing other widgets and grouping them together. Add it to the imports at the top of your code:

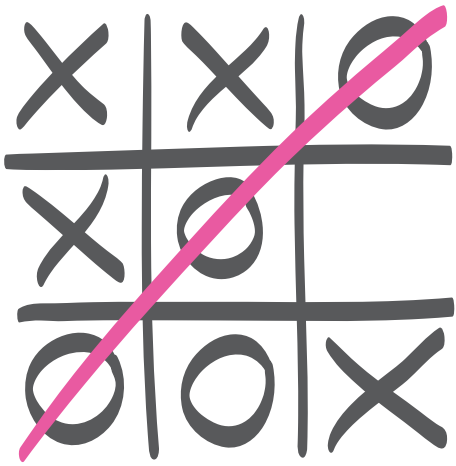


Figure 1 A typical game of tic-tac-toe

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

Set the Box to have a grid layout and add it to your app – before the `app.display()` line, as with all widgets.

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

If you run your program at this point, you won't see anything on the screen because the Box itself is invisible.

Now let's create the buttons to go inside it. You will need nine buttons in total, so instead of creating them individually, you can use a nested loop to generate them all automatically and give them co-ordinates. First, add PushButton to your list of widgets to import and then add this code immediately after the code for the board you just created.

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


www.dbooks.org

Figure 2 A grid of nine buttons to play tic-tac-toe

Notice that there are two loop variables: `x` from 0 to 2 and `y` from 0 to 2. As we iterate and generate buttons, each button will be added to the board, which is the Box container you created earlier. The button will be given the grid co-ordinates `x,y`, meaning that each button is neatly placed on a grid at a different position!

Your code should now look like **tictactoe1.py**. The result of running it is shown in **Figure 2**.

Underlying data structure

You might notice that when you create the buttons using a loop, you are creating nine buttons automatically and every single one is called `button`. How will you be able to refer to each of these buttons in the program?

The answer is that you need an underlying data structure to hold a reference to each button, and for this you will use a two-dimensional list.

Let's create a function which we can call to clear the board. It is a good idea to do this in a function so that you can reuse the code once the game has been played to reset the board and allow the player to begin a fresh game.

In the functions section, add a new function called `clear_board`.

```
def clear_board():
```

Your first job inside this function is to initialise the data structure for the board. Let's assume at this point you have not created any buttons, so you can initialise each position on the board as `None` – the element in the list now exists but does not yet have a value. Add the following line, indented, to your function.

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

Next, move the nested loop code from your app section into the `clear_board` function. Make sure the indentation is correct.

Inside the inner (y) loop, add a line of code to store a reference to each button at its x,y co-ordinate position within the two-dimensional list so that you can refer to it later.

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

Finally, after the loops end, return the `new_board` you have just created. Your function should look like this:

```
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 the app section, initialise a list called `board_squares` and set it to call the new function you just created.

```
board_squares = clear_board()
```

This variable will be assigned the value of the `new_board` you created within the function, which should be a blank board with nine buttons. Make sure that you create this variable after the code for creating the Box, otherwise you will be trying to add buttons to a container that does not yet exist.

Your code will now resemble **tictactoe2.py**. Save and run the program and you should see an identical result to the one you had at the end of the last step, but now you have a hidden two-dimensional list data structure to let you reference and manipulate the buttons.

If you want to see what your 2D list looks like, you could add a `print` command to print the `board_squares` list: `print(board_squares)`. You should then see nine lots of `[PushButton] object with text ""` appear in the shell.

www.dbooks.org

Make the buttons work

At the moment, your buttons don't do anything when you press them. Let's make a function to attach to the button, so that when it is pressed, the button displays either X or O depending on which player chose it.

First, create a variable in the variables section to record whose turn it is. You can choose to start with either player, but we will choose to start with X.

```
turn = "X"
```

This now means that you need to display on the GUI whose turn it is (**Figure 3**) so the players don't get confused. Add Text to your list of widgets to import:

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

Then add a new Text widget in the app section to display the turn.

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

Move to the functions section and create a new function called `choose_square`.

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

You will notice that this function takes two arguments – `x` and `y`. This is so that you know which square on the board has been clicked.

Figure 3 Let your players know whose turn it is

Add the following code (indented) inside the function to set the text inside the button that was clicked to the symbol of the current player, and then disable the button so it cannot be clicked on again.

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

Finally, connect this function to the button. Find this line of code inside your `clear_board` function:

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

Modify it so that it looks like the line below:

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

You have added two things here. Firstly, you are attaching a command, just as before. When the button is pressed, the function with this name will be called. Secondly, you are also providing arguments to this function, which are the co-ordinates x and y of the button which was pressed, so that you can find that button again in the list.

Your program should now look like **tictactoe3.py**. Save and run it. You will now be able to click on a button and it will change to an X. Unfortunately, in this version of the game it is permanently X's turn!

Alternating between players

Once one player has taken their turn, the turn variable should toggle to be the other player. Here is a function which will toggle from X to O and vice versa.

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

Add the code in your functions section. Notice the first line in the function: `global turn`. You need to specify this so that you are allowed to modify the *global* version of the `turn` variable, i.e. the one you already created. If you don't specify this, Python will create a local variable called `turn` and modify that instead, but that change won't be saved once the function exits.

www.dbooks.org

You also need to make sure that the Text widget accurately reports the current player's turn. After the if/else statement in the `toggle_player` function, update the message like this:

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

Go back to your `choose_square` function and call the `toggle_player` function – with `toggle_player()` – once you have set the text and disabled the button. Your code should now resemble **tictactoe4.py**. Save and test the program again and you should find that you can click squares and they will alternately be designated either X or O.

Do we have a winner?

Finally, you need to write a function which will check whether there is a row, column, or diagonal of three Xs or Os, and if so will report the winner of the game.

Although it seems very inelegant, by far the easiest way to check if someone has won is to hard-code the checks for each vertical, horizontal, and diagonal line individually.

The following code is for one vertical line, one horizontal line, and one diagonal – can you add the rest?

```
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]
```

Notice that the function begins by creating a Boolean variable called `winner`. If by the time the long if/elif statement has been executed, the value of this variable is True, you know that someone has won the game.

After adding the remaining winning line checks, add some code at the end of the function to change the display message if there has been a winner:

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

You now need to make sure that this function is called each time an X or O is placed, which corresponds to any time a button is pressed. Add a call to `check_win` at the end of the `choose_square` function, just in case the square that was chosen was the winning square.

Your program should now look like **tictactoe5.py**. Run it and test the game. If you wrote the tests in the `check_win` function correctly, you should find that the game detects correctly when a player has won.

RESET THE GAME

At the start, you wrote a function called `clear_board`. This may have seemed unnecessary at the time, but in actual fact it was thinking ahead to when the game has ended. Since tic-tac-toe is quite a short game, it is likely that someone might want to play more than one game in a row.

Can you add a reset button to your game, which only appears once either someone has won the game, or the game was a draw? The button should call the `clear_board` function and reset the `turn` variable as well as the message reporting whose turn it is.

Hint: *You will need to check the guizero documentation to find out how to hide and show widgets, so that your button is not visible all of the time during the game.*

Hint: *Create a new function which takes care of everything you need to do to reset the game, and call that function when the reset button is pressed. Don't forget that in your function you'll need to specify some variables as global.*

www.dbooks.org

Draw game

At the moment, the game will allow you to continue playing even after it has been won, until all of the squares are selected. It will also not tell you if the game was a draw. You could stop at this point, but if you really want to put the icing on the cake, adding a few more little touches could make your game more polished.

First, let's add some code to detect whether the game is a draw. The game is a draw if all of the squares contain either an X or an O, and no one has won. In the functions section, create a new function called moves_taken:

```
def moves_taken():
```

You're going to use this function to count the number of moves which have been made, so let's start a variable to keep count, initially beginning at 0.

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

Now, remember when we created the board_squares, we used a nested loop to create all of the squares on the grid? We're going to need another nested loop here to check each and every square and determine whether it has been filled in with an X or O, or whether it is blank.

GLOBAL VARIABLES

It is arguably a bad idea to use global variables because if you have many functions in a large program, it can become extremely confusing as to which code modifies the value of a variable and when. In a small program like this, it is not too difficult to keep track.

Remember that it is possible to read and use the value of a global variable from inside a function without declaring it global, but in order to modify its value you will need to explicitly declare this. The functions in this program (and most GUI programs in this book) are actually modifying the values of your widgets as global variables. For example, when someone wins the game, you set the value of the message to display who won:

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

In this example, message is a global variable. So how can we modify its value without declaring it as global? The answer is because we are using a *property* of the `message` widget, the property called value. Essentially what this code is saying is "Hey Python, you know that widget over there called message? Well, could you modify its value property please?" Python will allow modification through object properties in the global scope, but it won't allow you to directly modify the value of a variable without declaring it global.

Add this code for a nested loop to the `moves_taken` function:

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

Inside the loop, we need to check whether that particular square is filled in with an X or an O. If it is, add 1 to the `moves` variable to record that square has been counted.

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

Finally, once the loops have finished, add a `return` statement to return the number of moves taken.

```
return moves
```

Now, call this function inside the `check_win` function, to check for a draw. Add this code after the code that checks for a winner:

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

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

Your code should resemble **06-tictactoe.py**. When run, the game will now check whether nine moves have been taken; if they have, it will change the message to report that the game was a draw.

www.dbooks.org

tictactoe1.py / Python 3

DOWNLOAD
magpi.cc/guizerocode

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

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

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

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

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

app.display()
```

tictactoe2.py / Python 3

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

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

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

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

board = Box(app, layout="grid")
board_squares = clear_board()

app.display()
```

tictactoe3.py / 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()

# 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()
```


www.dbooks.org

tictactoe4.py / 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()

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

# 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()
```

tictactoe5.py / 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"]:
```


www.dbooks.org

tictactoe5.py (cont.) / Python 3

```
        winner = board_squares[2][0]

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

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

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

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

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

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

app.display()
```

06-tictactoe.py / 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
```


www.dbooks.org

06-tictactoe.py (cont.) / Python 3

```
    ) 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
```

06-tictactoe.py (cont.) / Python 3

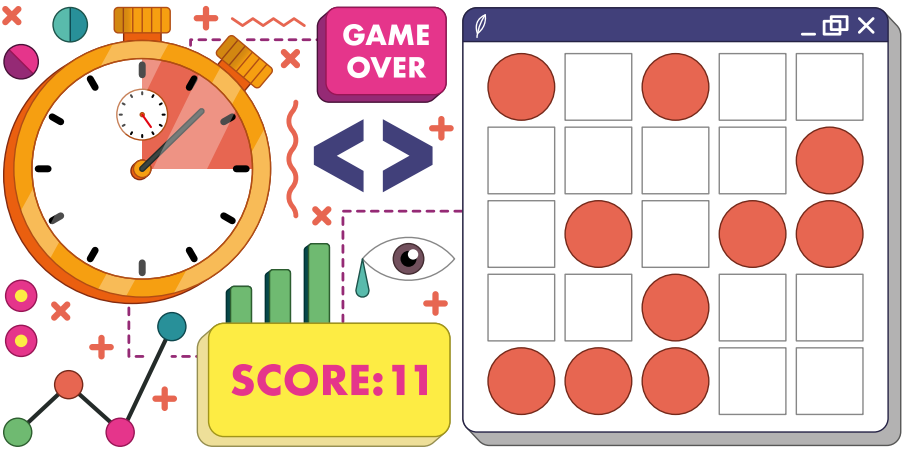
```
# Variables -------------
turn = "X"

# 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()
```


www.dbooks.org

Chapter 7

Destroy the Dots

Learn how to use a Waffle to create a tasty game



You saw in the Tic-tac-toe game how to create a GUI on a grid layout in order to present the player with a grid-like board. If you are making a game involving a larger grid, there is a type of guizero widget called a Waffle which can instantly create a grid for you, and is really useful for creating all kinds of fun games.

A Waffle was originally a grid of squares in early versions of guizero. This game is called 'Destroy the dots' and it came about because Martin thought it was a good idea to allow a Waffle widget to contain a mixture of squares and dots.

Aim of the game

In this game, you need to destroy the dots before they destroy you! The board consists of a grid of squares. The squares will gradually turn into dots. To destroy a dot, click on the dot and it will turn back into a square. The aim of the game is to last as long as possible before being overrun by dots (**Figure 1**).

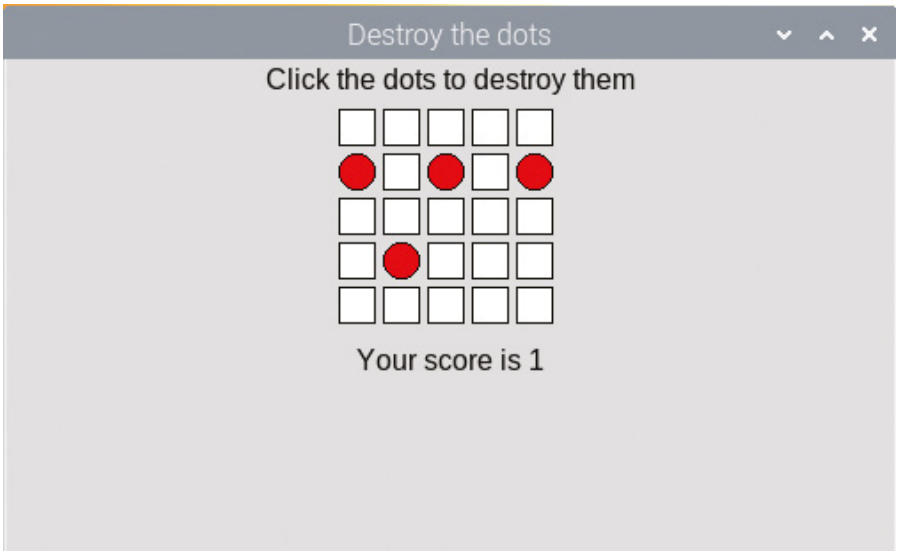


Figure 1 Destroy the red dots before they take over the board

Set up the game

Let's start by making a guizero program which contains the instructions for the game and a Waffle. By now you should be familiar with the layout of a standard guizero program with sections for imports, functions, variables, and the app itself.

First, create an App and inside it add a Text widget for the instructions and a Waffle widget for the board:

```
# 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()
```

If you run your program, you will see a small 3×3 grid of white squares. If you want to make your grid bigger than this, you can add width and height properties to your Waffle:

www.dbooks.org

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

Your code should now resemble **destroy1.py** (page 71).

Bring on the dots

Next you need to write a function to find a random square on the board and turn it into a dot. Begin a new function in your functions section called `add_dot()`:

```
def add_dot():
```

To choose a random square on the board, you need to be able to generate a random pair of integers as co-ordinates. Add a line in your imports section to import the `randint` function from the `random` library, which lets you generate a random integer.

```
from random import randint
```

Let's generate two variables, `x` and `y`, which you can use to reference a co-ordinate on the grid. Inside your `add_dot()` function, begin your code like this:

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

Notice that you have generated two random integers between 0 and 4, because earlier on you set the width and height of the grid to be 5 – the rows and columns will be numbered from 0. If you chose different values earlier on, you will need to adjust the values here to fit the size of your grid. However, there is a better way to manage aspects like this (see 'Using constants' box on page 70).

Dot or not?

Now that you know about constants, you can use the following function to generate a random co-ordinate on the grid:

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

At this point, you don't know whether the randomly chosen co-ordinate is already a dot or not. This might not seem to make any difference at the start of a game when the board is mostly squares, but as the board gets filled up with dots, you need to make sure that the space is actually a square, otherwise the game will be too easy. One way to achieve this is to run a loop which checks whether the chosen square is already a dot, and if it is, chooses another random square:

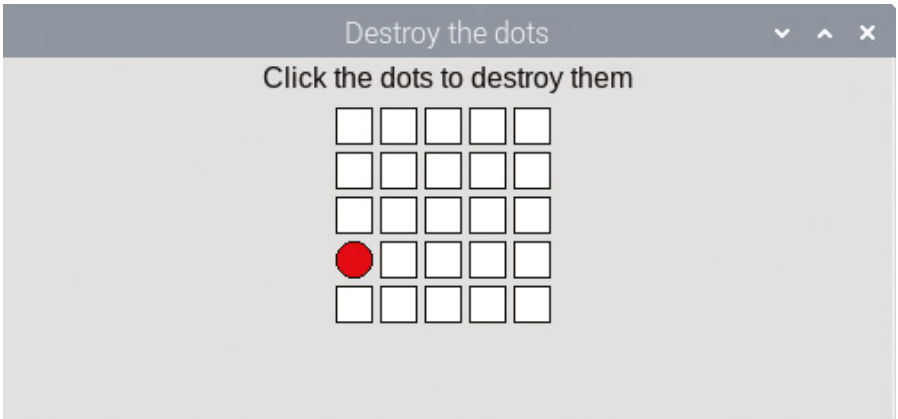


Figure 2 Generating a random red dot

```
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)
```

You might realise that this isn't a particularly efficient method of choosing a random square that is not a dot, but it will do for what we need in this game.

As soon as this loop finishes, you can be sure that the randomly chosen x, y co-ordinate is definitely a square. Let's convert it to a red dot – following (not inside) your `while` loop, add the following lines:

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

Add a call to your new `add_dot()` function in the app section after you've created the board. Your program should now resemble **destroy2.py**. When you run it, you should see a single random red dot in the grid. If you run the program again, the dot will probably be in a different random place (**Figure 2**).

Destroy the dot

So far there is only one dot – let's destroy it! Don't worry: you'll add more dots to destroy later on, but once you can destroy one, you can destroy them all!

Make a new function in your functions section with a really satisfying name – `destroy_dot` – and give it two parameters, `x` and `y`.

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


www.dbooks.org

This function will check whether the co-ordinate x,y is a dot (rather than a square). You can do this using the same code as the code to create a dot – the code `board[x, y].dotty` will return True if that coordinate is a dot, and False if it is a square.

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

If the co-ordinate is a dot, change it to a square by setting its `dotty` property to False, and also change its colour back to white:

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

This function needs to be triggered whenever the board is clicked. Find the line of code you already have which creates the board, and add a command like this:

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

This will call the `destroy_dot` function whenever a space on the board is clicked.

Note that a Waffle widget will automatically pass two parameters to any command function; these will always be the x and y co-ordinates of the pixel that was clicked on to trigger the command.

Your code should now look like **destroy3.py**. Test your program by running it and clicking on the dot. You should see the dot turn back into a white square. If you click on a square that is not a dot, nothing should happen.

More dots!

Now it's time to actually make the game a challenge, by adding continually spawning dots. Let's start off by adding a new random dot every second. To do this, you need to schedule a call to the `add_dot` function every second using a built-in feature of guizero called `after`.

In your app section, remove the call to `add_dot()` and replace it with a new line of code:

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

This line of code means 'after 1000 milliseconds (1 second), call the function `add_dot`'.

If you run the program now, you'll still get a single dot, but it will appear on the grid after a delay of 1 second.

Here's the clever bit. Find your `add_dot` function and add the same line of code to it, at the end of the function.

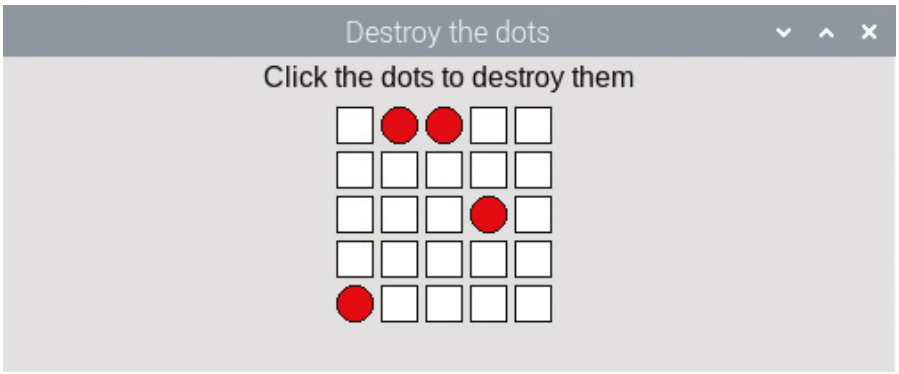


Figure 3 Every second, a new dot will appear

This will schedule a new call to `add_dot` every time a new dot finishes being added. The next dot is scheduled to appear in 1 second as well, so if you run the program you should see a new dot appearing on the grid every second (**Figure 3**).

Try running your program, which should now look like **destroy4.py**. Since you already wrote the method to destroy a dot, clicking on any dot should remove it. However, if you play the game for a while you will notice it is pretty easy to keep up with the pace of one dot every second and it is almost impossible to lose the game.

You still need to add two things – a score to keep track of how many dots you have destroyed, and a way of making the game get more difficult so that it becomes a challenge.

Add a score

Adding a score is pretty straightforward and takes three steps:

- Add a variable to keep track of the score; the variable should start at 0.
- Display a message on the GUI with the current score.
- Any time the `destroy_dot` function is called and a dot is destroyed, add 1 to score and update the message display.

Try to add the code yourself using what you have already learnt.

Hint: *To update the score variable from the* `destroy_dot` *function, you will need to declare it a global.*

Hint: *If you get an error saying that the variable score is referenced before assignment, make sure your variables section comes before your functions section in your program.*

The solution is shown overleaf if you are stuck…

www.dbooks.org

Solution: add a score

First, add a variable in your variables section

```
score = 0
```

Next, add a new Text widget in the app section to display the score:

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

Finally, add 1 to the score every time a dot is destroyed:

```
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)
```

Your code (without the optional comments) should now resemble **destroy5.py**. Test your game and you should see your score go up by 1 every time you click on a dot.

Put the player under pressure

Now that you can track the player's score, you can use it to put the player under pressure and speed up the spawn of dots if they are doing well.

Remember that you used an after call inside the `add_dot` function to schedule another dot in 1000 milliseconds (or 1 second)? Go back and find that line – you're going to change it a bit.

First, create a variable speed and set it to 1000. Then, instead of scheduling a call to add a dot after 1000 ms, schedule it to add a dot after speed milliseconds. This will have absolutely no effect on the game… yet. You are still scheduling the next call after 1000 ms, but that figure is now coming from the variable speed instead of being hard-coded as a magic number.

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

Now here's how you can ramp up the pressure. Between these two lines of code, you can add some code to set the speed of dots depending on the current score. Here is an example:

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

Here, you can see that if the player has got more than 10 points, the new dots will appear every 500ms, if they have more than 20 points a dot will appear every 400ms, and so on. This makes the game much harder the more points you have. Save your code – **destroy6.py** – and test the game to see the difference. You can alter the numbers or add more `elif` conditions if you want to increase the difficulty even further.

Game over

All that remains is to figure out when the player has lost the game; this happens when every square has turned into a red dot.

Remember that when you made Tic-tac-toe, you used *nested loops* to check whether all squares were filled and the game was a draw? You can use the same method here too, to loop through every square on the grid and check if it is a red dot. In your `add_dot` function, just before the call to `after`, add some code for a nested loop to loop through all squares on the board:

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

The first line begins by assuming that all squares are red. The nested loop will provide the coordinates of every square on the grid in turn, as the values `x` and `y` so that you can check whether this is true.

Add some code inside the second loop to find out whether the current pixel is red, and if it is *not*, change the `all_red` variable to 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
```


www.dbooks.org

After both loops end (make sure you unindent the following code), check whether the grid was all red dots. If it is, the player has lost so display a message:

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

If the player hasn't lost, the game should continue. Add an `else:` and inside it, indent the `after` method you already have, as we only want to add a new dot if the player has *not* lost:

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

Be careful to indent the `after` line you already have here rather than adding another one, or your game will start behaving strangely and generate multiple dots per second!

Your final code should resemble **07-destroy-the-dots.py**. Enjoy the game.

USING CONSTANTS

Setting the height and width of your Waffle to 5 is known as using a 'magic number' in a program, because the specific number is hard-coded into the program. If you want to change the size of the grid, you will need to find everywhere in the program this number appears and change it, which might be messy.

Better programming practice would be to define a *constant* in your variables section called `GRID_SIZE` and set it equal to 5:

```
GRID_SIZE = 5
```

Then, instead of defining your Waffle's dimensions with a magic number 5, you can put:

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

If you decide to change the size of the grid, you can just change the value of this constant.

Thinking about this type of thing at the time you write the program will help you to avoid headaches later if you decide to change it.

CHALLENGE

- Can you add a reset button which allows the player to begin a new game without having to rerun the program?
- Can you put even more pressure on the player by calculating how many red dots are on the board, and increasing the speed in proportion to the number of red dots?

destroy1.py / Python 3

DOWNLOAD
magpi.cc/guizerocode

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

from guizero import App, Text, Waffle


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


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


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

app = App("Destroy the dots")

instructions = Text(app, text="Click the dots to destroy them")
board = Waffle(app, width=5, height=5)

app.display()
```

destroy2.py / Python 3

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

from guizero import App, Text, Waffle
from random import randint


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

GRID_SIZE = 5


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

def add_dot():
    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)
    board[x, y].dotty = True
```


www.dbooks.org

destroy2.py (cont.) / Python 3

```
    board.set_pixel(x, y, "red")


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

app = App("Destroy the dots")

instructions = Text(app, text="Click the dots to destroy them")
board = Waffle(app, width=5, height=5)
add_dot()

app.display()
```

destroy3.py / Python 3

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

from guizero import App, Text, Waffle
from random import randint


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

GRID_SIZE = 5


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

def add_dot():
    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)
    board[x, y].dotty = True
    board.set_pixel(x, y, "red")

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

destroy3.py (cont.) / Python 3

```
# App -------------------

app = App("Destroy the dots")

instructions = Text(app, text="Click the dots to destroy them")
board = Waffle(app, width=GRID_SIZE, height=GRID_SIZE,
command=destroy_dot)
add_dot()

app.display()
```

destroy4.py / Python 3

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

from guizero import App, Text, Waffle
from random import randint

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

GRID_SIZE = 5

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

def add_dot():
    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)
    board[x, y].dotty = True
    board.set_pixel(x, y, "red")
    board.after(1000, add_dot)

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


# App -------------------
```


www.dbooks.org

destroy4.py (cont.) / Python 3

```
app = App("Destroy the dots")

instructions = Text(app, text="Click the dots to destroy them")
board = Waffle(app, width=GRID_SIZE, height=GRID_SIZE,
command=destroy_dot)
board.after(1000, add_dot)

app.display()
```

destroy5.py / Python 3

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

from guizero import App, Text, Waffle
from random import randint

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

GRID_SIZE = 5
score = 0

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

def add_dot():
    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)
    board[x, y].dotty = True
    board.set_pixel(x, y, "red")
    board.after(1000, add_dot)

def destroy_dot(x,y):
    global score
    if board[x,y].dotty == True:
        board[x,y].dotty = False
        board.set_pixel(x, y, "white")
        score += 1
        score_display.value = "Your score is " + str(score)
```

destroy5.py (cont.) / Python 3

```
# App -------------------

app = App("Destroy the dots")

instructions = Text(app, text="Click the dots to destroy them")
board = Waffle(app, width=GRID_SIZE, height=GRID_SIZE,
command=destroy_dot)
board.after(1000, add_dot)
score_display = Text(app, text="Your score is " + str(score))

app.display()
```

destroy6.py / Python 3

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

from guizero import App, Text, Waffle
from random import randint

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

GRID_SIZE = 5
score = 0

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

def add_dot():
    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)
    board[x, y].dotty = True
    board.set_pixel(x, y, "red")

    speed = 1000
    if score > 30:
        speed = 200
    elif score > 20:
        speed = 400
    elif score > 10:
```


www.dbooks.org

destroy6.py (cont.) / Python 3

```
        speed = 500
    board.after(speed, add_dot)

def destroy_dot(x,y):
    global score
    if board[x,y].dotty == True:
        board[x,y].dotty = False
        board.set_pixel(x, y, "white")
        score += 1
        score_display.value = "Your score is " + str(score)


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

app = App("Destroy the dots")

instructions = Text(app, text="Click the dots to destroy them")
board = Waffle(app, width=GRID_SIZE, height=GRID_SIZE,
command=destroy_dot)
board.after(1000, add_dot)
score_display = Text(app, text="Your score is " + str(score))

app.display()
```

07-destroy-the-dots.py / Python 3

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

from guizero import App, Text, Waffle
from random import randint

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

GRID_SIZE = 5
score = 0

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

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

07-destroy-the-dots.py (cont.) / Python 3

```
        x, y = randint(0,GRID_SIZE-1), randint(0,GRID_SIZE-1)
    board[x, y].dotty = True
    board.set_pixel(x, y, "red")

    speed = 1000
    if score > 30:
        speed = 200
    elif score > 20:
        speed = 400
    elif score > 10:
        speed = 500

    all_red = True
    for x in range(GRID_SIZE):
        for y in range(GRID_SIZE):
            if board[x,y].color != "red":
                all_red = False
    if all_red:
        score_display.value = "You lost! Score: " + str(score)
    else:
        board.after(speed, add_dot)

def destroy_dot(x,y):
    global score
    if board[x,y].dotty == True:
        board[x,y].dotty = False
        board.set_pixel(x, y, "white")
        score += 1
        score_display.value = "Your score is " + str(score)

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

app = App("Destroy the dots")

instructions = Text(app, text="Click the dots to destroy them")
board = Waffle(app, width=GRID_SIZE, height=GRID_SIZE,
command=destroy_dot)
board.after(1000, add_dot)
score_display = Text(app, text="Your score is " + str(score))

app.display()
```


www.dbooks.org