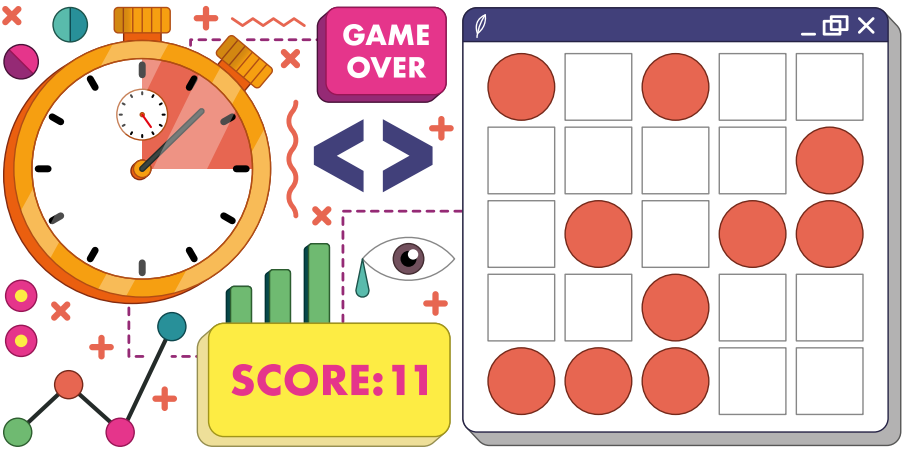
Chapter 7

# 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**).

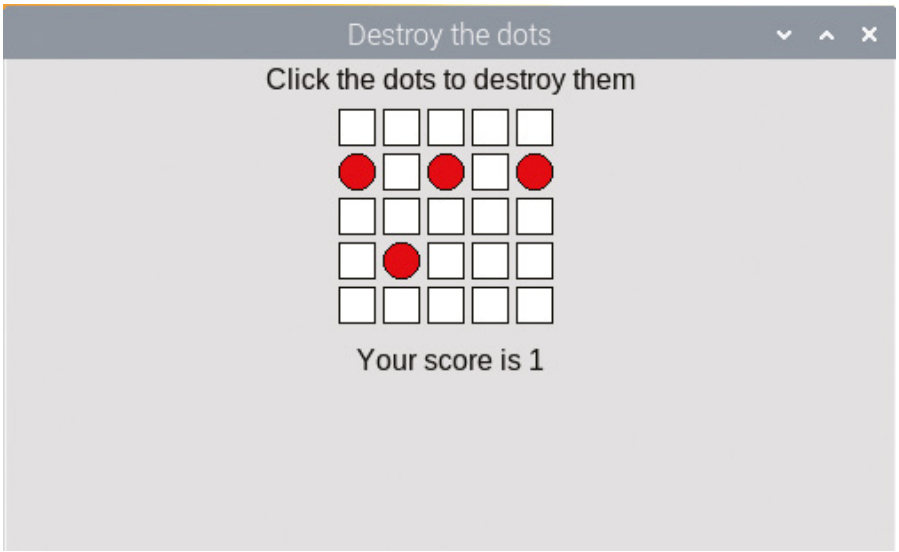



**Figure 1** Destroy the red dots before they take over the board

## 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:

```
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:

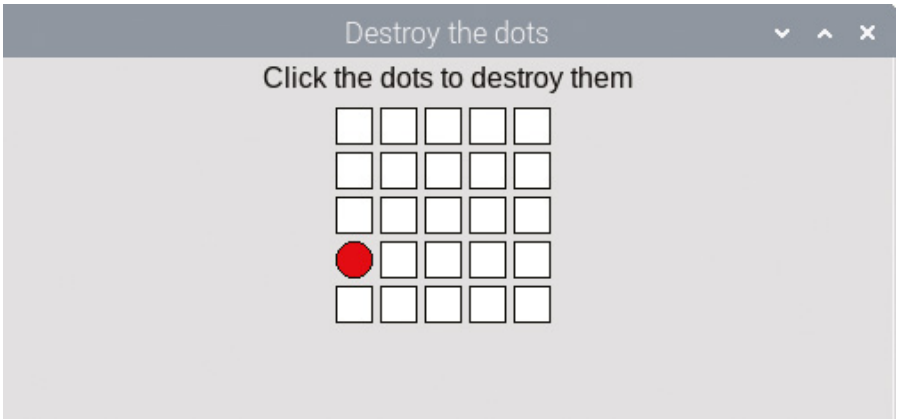



**Figure 2** Generating a random red dot

```
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):
```

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.

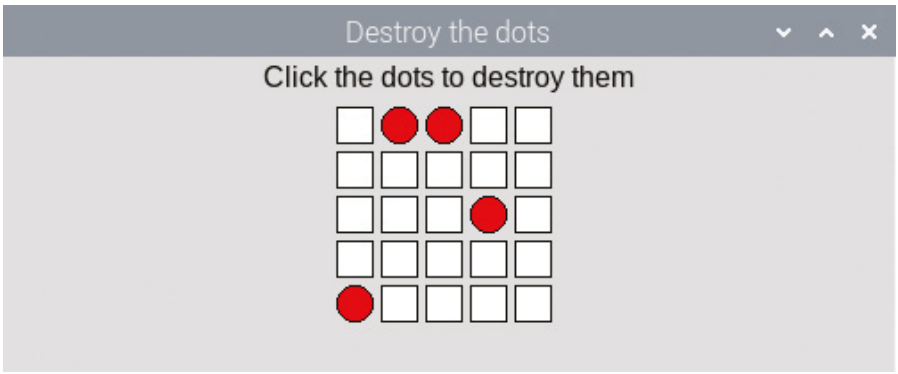



**Figure 3** Every second, a new dot will appear

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…

### 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
```

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
```

## 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 -------------------
```

## 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:
```

## 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()
```