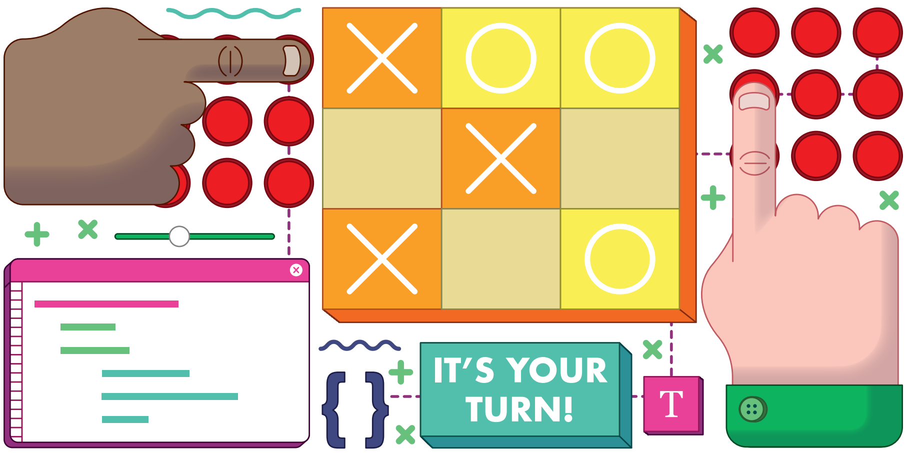
Chapter 6

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:

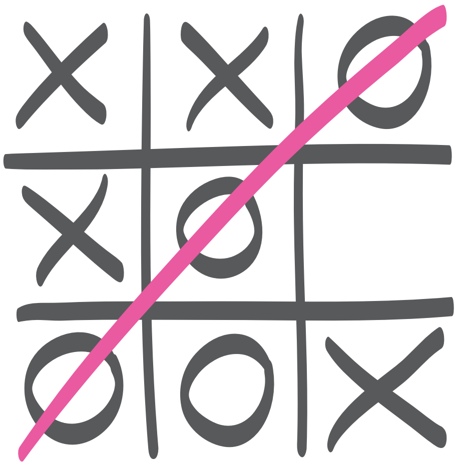


Figure 1 A typical game of tic-tac-toe

```
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
        )
```

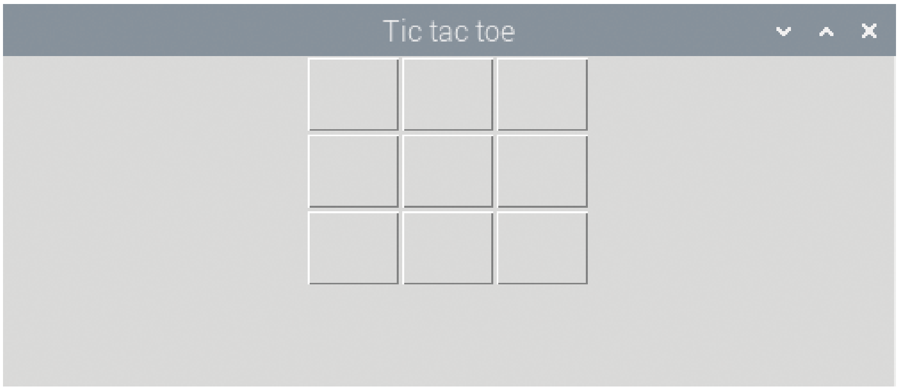


Figure 2 A grid of nine buttons to play tic-tac-toe

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.

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
```

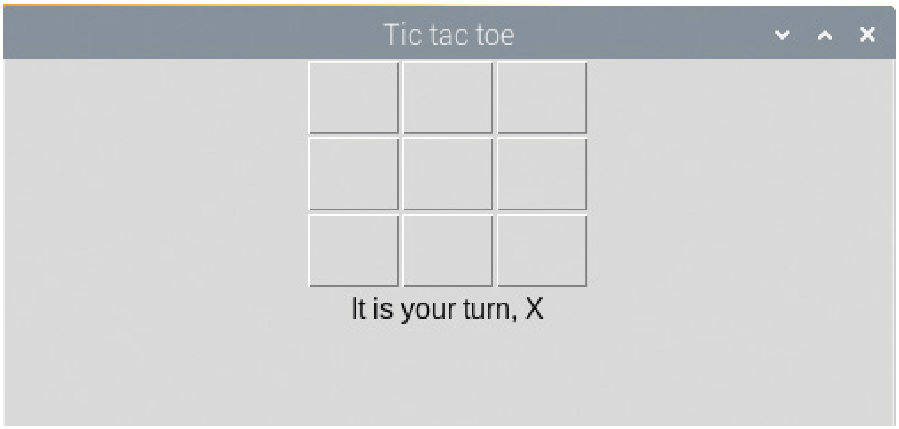
Then add a new Text widget in the app section to display the turn.

```
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.



It is your turn, X

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.

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.*

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.

tictactoe1.py / Python 3

DOWNLOAD

magpi.cc/guizerocode

```python
# 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

```python
# 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()
```

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"]:
```

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
```

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()
```