Téléchargement gratuit réservé aux membres d'Elektor : fichier PDF de 60 pages avec les parties 1 à 10 de cette série. Voir www.elektormagazine.fr/news/GUI-PDF-Special

5e partie

Création d'interfaces graphiques en Python avec guizero : jeu de morpion

Contrôlez la logique d'un jeu très simple à l'aide de widgets graphiques.

MAKER

Laura Sach

Laura dirige l'équipe *A Level* de la Fondation Raspberry Pi chargée des ressources pédagogiques en informatique à destination des étudiants.

@CodeBoom

MAKER

Martin O'Hanlon

Martin crée des cours, des projets et des ressources en ligne au sein de l'équipe *Learning* de la Fondation Raspberry Pi.

@martinohanlon

Le morpion, *tic-tac-toe* en anglais, est un jeu très simple où chaque joueur tente d'aligner trois symboles avant son adversaire (fig. 1). Vous apprendrez ici à contrôler le déroulement d'une partie à l'aide d'une logique de programmation interagissant en coulisses avec l'interface graphique du jeu.

Créez un nouveau fichier, et ajoutez le code suivant :

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

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

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

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

app.display()
```

Création du plateau

Nous utiliserons neuf boutons *PushButton* pour représenter les neuf cases composant la grille du jeu. Un clic sur une case déterminera la position choisie par le joueur pour le placement de son symbole. L'objet de *guizero* appelé *Box* va nous servir à créer la grille. Un widget *Box* est un conteneur invisible pouvant contenir d'autres widgets et les regrouper.

Importez donc *Box* en début de code :

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

Ajoutez la ligne suivante avant `app.display()`. Le second paramètre indique la disposition (**layout**) souhaitée des widgets contenus dans *Box*, d'où le **grid** (grille) passé en valeur.

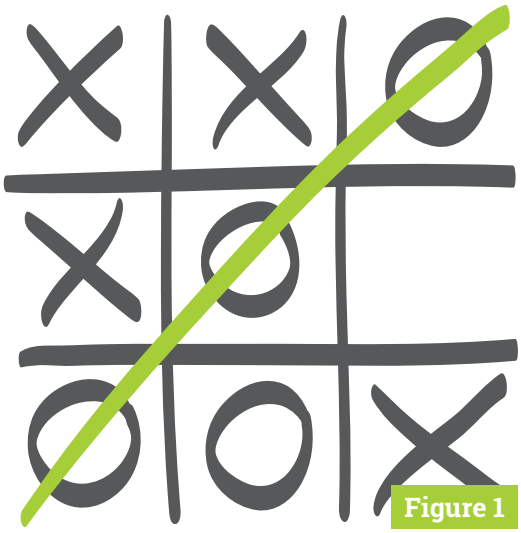


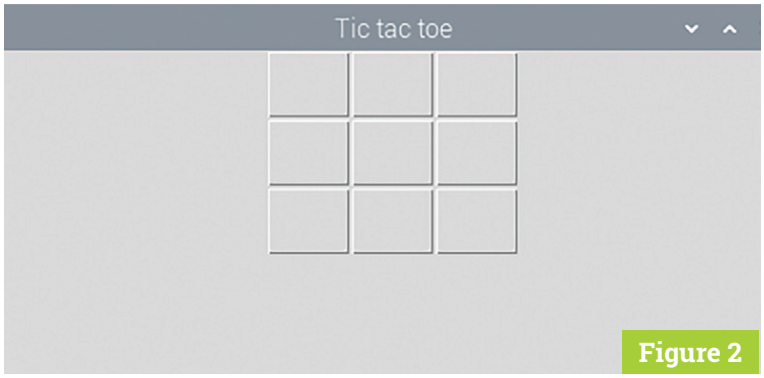
Figure 1

▲ **Figure 1** Le jeu du morpion.

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

Box étant un conteneur invisible, rien ne s'affichera dans la fenêtre **app** si vous lancez le code maintenant. Créons les neuf boutons de l'objet **board** (notre plateau) à l'aide d'une boucle imbriquée. Importez le widget *PushButton*, et ajoutez la boucle **for** ci-dessous, juste après la définition de l'objet **board :**

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



▲ **Figure 2** Création d'une grille comprenant neuf boutons.

tictactoe1.py

TÉLÉCHARGEZ LE CODE COMPLET :
magpi.cc/guizerocode

Langage : **Python 3**

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

tictactoe2.py

Langage : **Python 3**

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

Notez les deux variables de boucle : `x` allant de 0 à 2, et `y` allant aussi de 0 à 2. Un bouton `button` est créé et ajouté au conteneur `board` à chaque itération, et comme chaque bouton prend pour coordonnées les valeurs successives de `x` et de `y`, nous obtenons bien neuf boutons alignés sur une grille de 3 par 3 (**fig. 2**).

Le code correspondant est reproduit sur le listage **tictactoe1.py**.

Structure de données

Sans doute avez-vous remarqué que la boucle créait neuf boutons, mais que chacun d'entre eux était nommé `button`. Impossible dès lors de les différencier. Pour être à même de les utiliser dans le programme, nous avons besoin d'une structure de données contenant une référence vers chacun de ces boutons. Nous utiliserons à cet effet une liste à deux dimensions, c.-à-d. une liste de listes.

Partons d'une fonction dont le rôle sera d'effacer le plateau de jeu. Mettre ce code d'effacement dans une fonction est une bonne idée, nous pourrons le réutiliser en fin de jeu pour démarrer une nouvelle partie.

Dans la section *Functions*, définissez une fonction appelée `clear_board` :

```
def clear_board():
```

La première étape consiste à déclarer et initialiser la liste qui contiendra les neuf boutons. Comme nous n'avons encore créé aucun bouton, nous initialisons leurs positions à `None` – les éléments de la liste existeront, mais aucune valeur ne leur sera encore assignée. Ajoutez la ligne ci-dessous, indentée, au corps de la fonction :

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

Déplacez ensuite la boucle imbriquée précédente sous la définition de la liste, là encore en veillant aux indentations.

Dans la boucle interne (`y`), ajoutez la ligne ci-dessous pour remplir la liste `new_board` avec la référence vers les coordonnées `x` et `y` de chaque bouton..

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

Enfin, après la boucle, demandez à la fonction de retourner la liste `new_board` créée. Voici notre

Nouvelle partie

La fonction **clear_code()** définie en début de code semble inutile, mais ce n'est pas sans raison que nous l'avons écrite : comme une partie de morpion ne dure que très peu de temps, il est probable que les joueurs désireront en recommencer une autre aussitôt après.

Voilà pourquoi nous vous invitons à créer un bouton *Reset* s'affichant en fin de partie. Votre bouton devra appeler la fonction **clear_code()**, remettre à 0 la variable **turn**, et réinitialiser le message invitant le joueur X ou O à jouer.

Consultez la documentation de *guizero* pour apprendre à masquer et rendre visible un widget, et appliquez cette technique à votre bouton afin qu'il reste masqué durant la partie.

Indice : créez une nouvelle fonction traitant tout ce qu'il faut pour réinitialiser le jeu, et appelez-la lorsque le joueur clique sur le bouton *Reset*. N'oubliez pas qu'il vous faudra définir certaines de ses variables comme globales.

fonction :

```
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
```

Dans la section *App*, initialisez une liste appelée `board_squares` à l'aide de la fonction `clear_board()` :

```
board_squares = clear_board()
```

L'instruction ci-dessus affecte à la variable `board_squares` la valeur retournée par la fonction `clear_board()`, soit un plateau comprenant neuf cases vides (neuf objets `button`). Placez l'instruction après la définition du conteneur `board`, sinon le compilateur se plaindra qu'il ne peut pas ajouter des boutons dans un conteneur qui n'existe pas.

Enregistrez votre code (**tictactoe2.py**) et lancez-le. Le résultat devrait être identique au précédent, mais maintenant nous disposons d'une liste pour référencer et manipuler les boutons.

Pour voir le contenu de cette liste dans l'interpréteur de commande, ajoutez temporairement `print(new_board[x][y])` à la fin de la boucle imbriquée `y`, ou `print(board_squares)` avant la dernière ligne du code.

tictactoe3.py

▸ Langage : **Python 3**

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

Activation des boutons

À ce stade, il ne passe rien si nous cliquons sur une case du plateau. Créons donc une fonction qui affichera un X sur le bouton cliqué s'il s'agit du joueur X, un O s'il s'agit du joueur O.

Définissons d'abord dans la section *Variables* une variable `turn` qui stockera le nom du joueur devant jouer le prochain coup. Nous avons choisi arbitrairement X pour le premier coup à jouer.

```
turn = "X"
```

Le joueur X, lui, ignore bien sûr tout de notre code, aussi devons-nous l'avertir que c'est à son tour de jouer (**fig. 3**). Ajoutez *Text* aux widgets à importer :

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

Dans la section *App*, ajoutez un widget *Text* invitant le joueur X ou O à jouer :

tictactoe4.py

Langage : **Python 3**

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

Figure 3 Un widget *Text* pour appeler le joueur suivant.

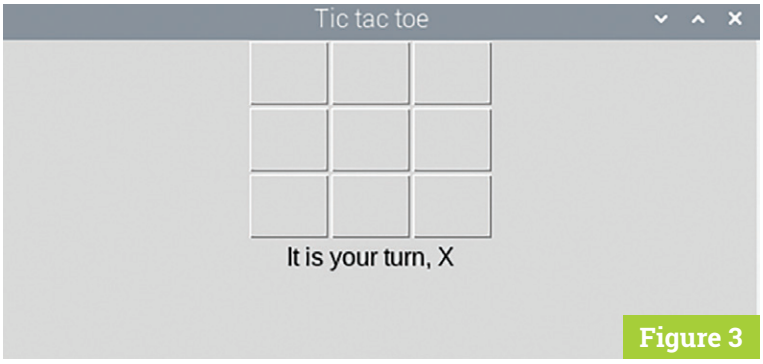


Figure 3

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

Dans la section *Functions*, définissez une fonction appelée `choose_square :`

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

Les deux arguments de la fonction sont les coordonnées de la case qui a été cliquée. Ajoutez à cette même fonction les deux instructions ci-dessous (en les indentant) ; la première affecte au texte affiché le symbole du joueur actuel, la seconde désactive le bouton pour qu'il ne puisse plus être cliqué.

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

Il nous faut maintenant « connecter » cette fonction au bouton. Dans la fonction **clear_board()**, repérez cette ligne :

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

Modifiez-la de façon à ce qu'elle devienne :

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

La première modification attache la commande `choose_square` à `button` de façon à ce que cette fonction soit appelée lorsque le bouton est cliqué. L'autre modification (`args=[x,y]`) fournit à `choose_square()` les arguments `x` et `y` du bouton cliqué (ses coordonnées), ce qui nous permettra de retrouver ce bouton dans la liste.

Enregistrez votre programme (listage **tictactoe3.py**) et exécutez-le. Un X apparaît bien lorsque nous cliquons sur une case, mais pour l'instant c'est toujours à X de jouer !

Chacun son tour

Pour qu'un joueur passe la main au suivant, nous devons changer la valeur de `turn` en conséquence. La fonction ci-dessous met `turn` sur O si elle était sur X, et vice versa.

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

tictactoe5.py

Langage : **Python 3**

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

```
turn = "X"
```

Ajoutez-la à la section *Functions*. Notez la première ligne : `global turn`. Si nous ne transformions pas ainsi en variable *globale* la variable `turn`, Python créerait et modifierait une variable *locale* appelée `turn`, donc indépendante de la variable `turn` définie et initialisée dans la section *Variables* ; son contenu ne serait de surcroît pas sauvegardé après l'exécution de la fonction.

Il faut également que le widget *Text* mette à jour le nom du joueur suivant. Pour cela ajoutez la ligne suivante après le bloc `if/else` de `toggle_player()` :

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

Il ne nous reste plus qu'à appeler `toggle_player()` depuis la fonction `choose_square()` – après l'affectation de `text` et la désactivation du bouton. Enregistrez votre code (listage **tictactoe4.py**) et exécutez-le : le programme devrait inviter tour à tour X et O à jouer, et les symboles correspondants s'afficher sur les cases cliquées.

Avons-nous un gagnant ?

Il nous faut maintenant écrire une fonction vérifiant si une rangée, colonne ou diagonale contient trois X ou trois O, et si c'est le cas désigner le vainqueur.

Même si elle n'est pas très élégante, la méthode la plus simple consiste à écrire explicitement tous les cas correspondant aux lignes, colonnes et

Variables globales

N'utiliser que des variables globales est une mauvaise idée. Dans un code aussi court que le nôtre, il est facile de retrouver quelles fonctions modifient la valeur d'une variable et à quel moment, mais si un code comprend de nombreuses fonctions il devient vite difficile de suivre toutes les répercussions qu'entraîne la modification d'une variable globale.

Souvenez-vous qu'il est possible de *lire* et *utiliser* la valeur d'une variable globale depuis une fonction sans la déclarer comme globale, mais cette déclaration est nécessaire pour *modifier* sa valeur. Les fonctions de notre jeu de morpion (et la plupart de celles de cette série) modifient les valeurs des widgets de façon globale. Prenons l'exemple d'un joueur gagnant la partie, dans ce cas nous modifions la valeur du widget Text **message** de la façon suivante :

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

Comment pouvons-nous modifier la valeur de **message** alors que nous ne l'avons pas déclarée comme variable globale ? L'explication vient de ce que nous utilisons la propriété du widget *message* appelée **value**. L'instruction demande au compilateur : « Hé Python, tu vois ce widget appelé message ? Pourrais-tu modifier sa propriété value, stp ? ». Python est d'accord, car il permet de modifier les propriétés d'un objet appartenant à l'espace de nommage global. Il interdit en revanche de modifier directement la valeur d'une variable non déclarée comme globale.

Python 3 for Science and Engineering Applications

Si vous maîtrisez les bases de Python et souhaitez explorer le langage plus en profondeur, ce livre (en anglais) est pour vous. Il illustre de nombreux aspects de la programmation au moyen d'exemples numériques et textuels concrets (chaos, modélisation de populations, jeu 2048, palindromes…). Découvrez un extrait sur : **www.elektor.fr/python-3-for-science-and-engineering-applications**

diagonales contenant trois symboles identiques, et à les comparer avec la partie en cours.

Le code ci-dessous est pour une ligne horizontale, une ligne verticale et une diagonale gagnantes. Sauriez-vous coder les autres cas ?

```
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]
```

La fonction débute par la déclaration d'une variable booléenne appelée `winner`. Si après l'exécution des blocs `if/elif` cette variable passe à `True`, c'est qu'un joueur a gagné.

Complétez le code avec le reste des cas possibles, puis ajoutez à la fin de la fonction la ligne ci-dessous modifiant la valeur du texte affiché s'il y a un vainqueur :

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

La fonction doit bien sûr être appelée à chaque placement d'un X ou d'un O, c.-à-d. après chaque clic sur un bouton. Ajoutez un appel à `check_win()` à la fin de la fonction `choose_square()`, ce qui vérifiera si la case qui vient d'être cliquée donne une formule gagnante.

Exécutez votre code (listage **tictactoe5.py**) et testez-le pour vérifier que vous avez correctement codé tous les cas gagnants laissés à votre sagacité.

Partie nulle

Notez que même si un des joueurs a gagné, la partie peut être poursuivie tant qu'il reste des cases à cliquer. L'autre point à retenir est que le programme ne détecte pas les cas de parties nulles. Nous pourrions nous dire « C'est comme ça », mais nous préférons la satisfaction d'avoir créé un jeu abouti.

Commençons donc par la détection des parties nulles. Une partie sera considérée comme telle si toutes les cases contiennent un X ou un O et que personne n'a gagné. Dans la section *Functions*, créez une fonction appelée `moves_taken` :

```
def moves_taken():
```

Cette fonction nous servira à compter le nombre de coups (*moves*) joués jusqu'à présent. Définissons et initialisons à 0 une variable stockant ce nombre :

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

Comme nous l'avions fait dans la fonction `board_squares()` pour créer les cases du jeu, nous allons reprendre la technique de la boucle imbriquée pour déterminer si chaque case a été remplie avec un symbole ou si elle est vide. Ajoutez au corps de `moves_taken()` les deux lignes ci-dessous parcourant toutes les rangées et colonnes :

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

Vérifions maintenant si la case en cours contient un X ou un O. Si c'est le cas, nous incrémentons la variable `moves` (lui ajoutons 1) :

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

Note de l'éditeur : cet article est paru initialement dans le magazine MagPi (le magazine officiel du Raspberry Pi). La maison d'édition Elektor publie ce magazine en néerlandais, allemand et français (www.magpi.fr).

06-tictactoe.py

Langage : **Python 3**

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

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

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

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

def check_win():
    winner = None

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

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

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

    if winner is not None:
        message.value = winner.text + " wins!"
    elif moves_taken() == 9:
        message.value = "It's a draw"

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


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

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

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

app.display()
```

Une fois la boucle terminée, nous retournons la somme totale contenue dans `moves` à l'aide de l'instruction `return` :

```
return moves
```

Pour vérifier si la partie est nulle, nous appelons cette fonction depuis `check_win()`, de la façon suivante, juste après le code vérifiant s'il y a un gagnant :

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

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

Le programme ainsi modifié (listage **06-tictactoe.py**) regarde si neuf coups ont été joués, et si c'est le cas annonce une partie nulle (*It's a draw*).

(VF : Hervé Moreau)