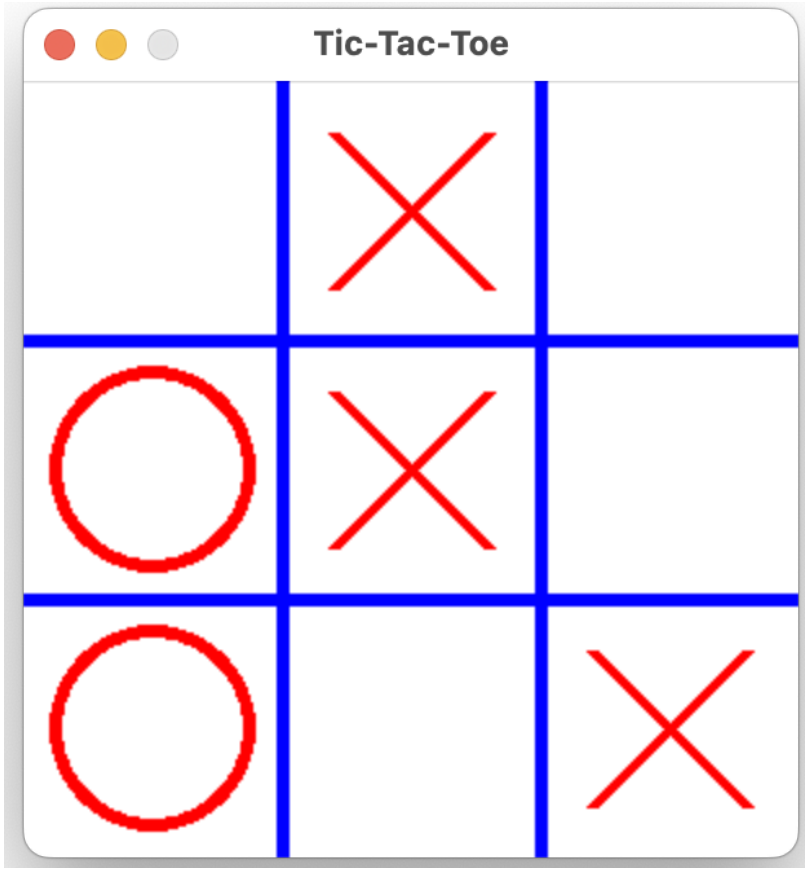
Difficulty: 🌶️🌶️🌶️🌶️🌶️	Language: Python	Requires: Laptop with Python, PyGame

Sheet link:



Ready for some classic gaming fun? Let's build a Tic-Tac-Toe game in Python using PyGame!

Remember, attention to detail is important. If you encounter an error, double-check your spelling and case sensitivity, as Python is particular about it.

For a smooth coding experience, we recommend using Mu, Thonny or any Python IDE with PyGame installed.

All the code for this project can be found at https://github.com/coder-dojo-ham/coder-dojo-worksheets/tree/main/python/tic-tac-toe.

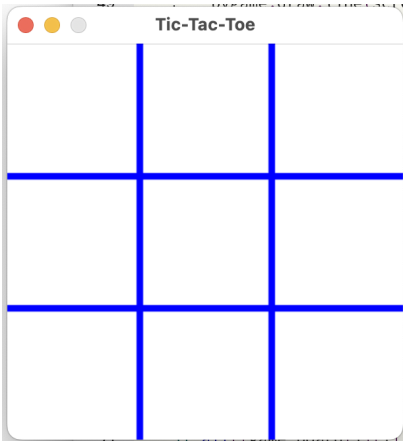
Step 1 - Setting Up Our Game Board

We're going to use PyGame to create our Tic-Tac-Toe game because it gives us an easy way to draw on the screen and handle game logic.

```
import pygame
import sys

# Initialize Pygame
pygame.init()

# Constants for our game window size and grid lines
WIDTH, HEIGHT = 300, 300
LINE_WIDTH = 5
LINE_COLOUR = (0, 0, 255)  # Blue colour in RGB
BG_COLOUR = (255, 255, 255)  # White background
GRID_SIZE = 3

# Set up the display
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Tic-Tac-Toe")


# Start with an empty game board
game_board = [[None, None, None],
              [None, None, None],
              [None, None, None]]

# Function to draw the grid
def draw_grid():
    # We'll add this in the next steps!
    pass

# Main loop to keep the game running
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    draw_grid()
    pygame.display.flip()

pygame.quit()
sys.exit()
```

Save and run this, fixing any problems before we carry on.

Step 2 - Drawing the grid



Now let's actually draw the grid. We will add lines on our window to form a 3x3 grid. Here's what you need to add to your `draw_grid` function:

```
# Constants for line properties
LINE_COLOUR = "blue"
LINE_WIDTH = 5


# Function to draw the grid lines
def draw_grid():
    screen.fill(BG_COLOUR)
    for i in range(1, GRID_SIZE):
        # Draw vertical lines
        pygame.draw.line(screen, LINE_COLOUR, (i * WIDTH //
GRID_SIZE, 0), (i * WIDTH // GRID_SIZE, HEIGHT), LINE_WIDTH)
        # Draw horizontal lines
        pygame.draw.line(screen, LINE_COLOUR, (0, i * HEIGHT //
GRID_SIZE), (WIDTH, i * HEIGHT // GRID_SIZE), LINE_WIDTH)
```

Testing and running this should show a grid!

Step 3 - Making the Board Interactive

It's time to make our game board respond to mouse clicks! We want to let our players click on a cell to place their X or O mark. We'll also track whose turn it is, so we know whether to draw an X or an O.

First, we'll need a way to determine which grid cell has been clicked. Let's add a function that calculates this based on the mouse position. Insert this before the main loop.

```
def get_cell_from_mouse_pos(mouse_pos):
    x, y = mouse_pos
    row = y // (HEIGHT // GRID_SIZE)
    col = x // (WIDTH // GRID_SIZE)
    return row, col
```

We need some constants and variables to handle who is currently playing. Insert this after the other constants near the top of the file:

```
# Additional constants and variable for the players
PLAYER_X = "X"
PLAYER_O = "O"
current_player = PLAYER_X
```

Next, we need to update our game loop to handle mouse click events and alternate between players' turns. Swap the main loop code for this:

```
# Main loop to keep the game running
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            # Get the cell that was clicked
            mouse_pos = pygame.mouse.get_pos()
            row, col = get_cell_from_mouse_pos(mouse_pos)

            # Make sure the cell is empty and add the player's
mark
            if game_board[row][col] is None:
                game_board[row][col] = current_player
                # Switch turns
                current_player = PLAYER_O if current_player ==
PLAYER_X else PLAYER_X
                print(f"Now playing: {current_player}")

    draw_grid()
    pygame.display.flip()
```

We're keeping track of the current player with current_player, which starts as PLAYER_X. Every time a cell is clicked, we add the current player's mark to the game_board and switch to the other player.

Step 3 - Drawing X's and O's

Now that we can click on the board and update the `game_board` array with the current player's symbol, we need to visually display these on the screen. We'll create two functions: one for drawing X's and one for drawing O's.

Here's how to draw an X. We'll add two diagonal lines to represent an X in the grid cell. Add this above draw_grid:

```
def draw_X(row, col):
    x_start = (col * WIDTH // GRID_SIZE) + 20
    y_start = (row * HEIGHT // GRID_SIZE) + 20
    x_end = ((col + 1) * WIDTH // GRID_SIZE) - 20
    y_end = ((row + 1) * HEIGHT // GRID_SIZE) - 20
    pygame.draw.line(screen, LINE_COLOUR, (x_start, y_start),
(x_end, y_end), LINE_WIDTH)
    pygame.draw.line(screen, LINE_COLOUR, (x_start, y_end),
(x_end, y_start), LINE_WIDTH)
```

And here's how to draw an O. We'll draw a circle in the grid cell. Add this above draw_grid:

```
def draw_O(row, col):
    center_x = (col * WIDTH // GRID_SIZE) + (WIDTH //
GRID_SIZE) // 2
    center_y = (row * HEIGHT // GRID_SIZE) + (HEIGHT //
GRID_SIZE) // 2
    pygame.draw.circle(screen, LINE_COLOUR, (center_x,
center_y), 40, LINE_WIDTH)
```

Now, we need to update our draw_grid function to call these functions when a player has marked a spot on the board:

```
# Update draw_grid function
def draw_grid():
    # (Previous code to draw the lines)
    # ...

    # Draw the X's and O's
    for row in range(GRID_SIZE):
        for col in range(GRID_SIZE):
            if game_board[row][col] == PLAYER_X:
                draw_X(row, col)
            elif game_board[row][col] == PLAYER_O:
                draw_O(row, col)
```

Run this, and you should now see player counters appear.

Step 4 - Going for the Win!

Our Tic-Tac-Toe game needs a way to determine when a player has won or if the game has ended in a draw. We will create functions to check for these conditions.

Checking for a Win

A player wins if they have 3 of their marks in a row, column, or diagonally. Here's how we can check for a win:

```
def check_win(player):
    # Check rows and columns
    for i in range(GRID_SIZE):
        if all([game_board[i][j] == player for j in
range(GRID_SIZE)]) or all([game_board[j][i] == player for j in
range(GRID_SIZE)]):
            return True
    # Check diagonals
    if all([game_board[i][i] == player for i in
range(GRID_SIZE)]) or all([game_board[i][GRID_SIZE - 1 - i] ==
player for i in range(GRID_SIZE)]):
        return True
    return False
```

The first loop goes over each row/column in the grid, checking for a win, uses the Python all function to check all items in smaller row and column loops.
The latter part is a special case for diagonals.

Checking for a draw

The game is a draw if all cells are filled and there is no winner:

```
def check_draw():
    return all([all(row) for row in game_board]) and not
check_win(PLAYER_X) and not check_win(PLAYER_O)
```

Updating the Game Loop

Now, we'll incorporate these checks into our main game loop. After a player makes a move and before we switch players, we'll check if the current player has won or if the game is a draw. If either condition is true, we'll reset the game board for a new game:

```
# Function to reset the game board
def reset_board():
    for row in range(GRID_SIZE):
        for col in range(GRID_SIZE):
            game_board[row][col] = None
```

We then need to use all this in an updated main loop:

```
running = True
while running:
    for event in pygame.event.get():
```

```
            if event.type == pygame.QUIT:
                running = False
            elif event.type == pygame.MOUSEBUTTONDOWN:
                mouse_pos = pygame.mouse.get_pos()
                row, col = get_cell_from_mouse_pos(mouse_pos)
                if game_board[row][col] is None:
                    game_board[row][col] = current_player
                    # Check for a win
                    if check_win(current_player):
                        print(f"{current_player} wins!")
                        reset_board()
                    # Check for a draw
                    elif check_draw():
                        print("It's a draw!")
                        reset_board()
                    else:
                        # Switch players if no win or draw
                        current_player = PLAYER_O if current_player
== PLAYER_X else PLAYER_X
                        print(f"Now playing: {current_player}")

        draw_grid()
        pygame.display.flip()
```

With these additions, your Tic-Tac-Toe game can now detect wins and draws, informing players about the game outcome. After announcing the result, it resets the game board, allowing for another game.

Remember to test each part of your code thoroughly to ensure everything works as expected.

Enhancements

- Graphical win conditions
- Displaying the current player
- Changing the colour of the player counters

Extension - Displaying the Current Player

Let's give the players some visual feedback on who's turn it is. We'll display a message on the screen that updates with the name of the current player.

First, we'll set up a Pygame font object at the beginning of our program, right after we initialize Pygame:

```
# Initialize Pygame
pygame.init()
```

```
# Font setup for displaying which player's turn it is
FONT = pygame.font.SysFont("arial", 24)  # You can choose any
font you like
```

Now, we'll create a function to render and display the "Now Playing" text:

```
def display_current_player():
    text_surface = FONT.render(f"Now playing:
{current_player}", True, (0, 0, 0))  # Black colour for the
text
    screen.blit(text_surface, (10, HEIGHT - 30))  # Position
the text at the bottom of the screen
```

Make sure to call display_current_player() in your main loop, after draw_grid() and before pygame.display.flip():

```
# Main loop to keep the game running
running = True
while running:
    for event in pygame.event.get():
        # Event handling goes here

    draw_grid()
    display_current_player()  # Update the display with the
current player
    pygame.display.flip()

pygame.quit()
sys.exit()
```

After making these changes, run your game again. You should now see the "Now playing" text at the bottom of the window, telling you whose turn it is.

Extension - making player counters a different colour

Somewhere in the player constants try adding a different colour for the players:

```
PLAYER_COLOUR = (255, 0, 0) # RED colour for players
```

And use this instead of LINE_COLOUR in the draw functions for the player counters.