Lesson 8: Tic-Tac-Toe

In this lesson, we will be creating a visual tic-tac-toe game using tkinter and 2D arrays. First, we'll focus on making the 2D array, then we will focus on drawing it.

PART 1: Setting up the Game

Our tic-tac-toe game board will be represented by a 2D array of integers. An empty spot on the board will be represented by the number 0. Therefore, we need a 3x3 2D array filled with 0s to start off. Remember that there is a quick way to fill a 2D array with one value. The code below is equivalent to writing:

board = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

```
board = [[0] * 3 for i in range(3)]
```

We'll have two more variables: player will be used to keep track whose turn it is and gameover will let us know when to end our game loop.

```
board = [[0] * 3 for i in range(3)]
gameover = False
player = 1
```

Let's set up our game loop. Our loop will keep going while gameover is false. For now, our loop will just print our 2D array row by row.

```
player = 1

while not gameover:
    for row in board:
        print(row)
    gameover = True # Delete this after testing
```

Run your program to test your code. You should see your 3x3 game board be printed out once. Delete the gameover = True line when you are done.

Part 2: Marking the Board

Let's create a function to mark parts of our board. This function will have two parameters: i and j. Remember that we use i to represent the rows of our 2D array and j to represent the columns of our 2D array. Since we will be changing our board and changing the player variable, we must tell the function that these variables are global variables.

```
def markBoard(i, j):
    global board, player

while not gameover:
    ...
```

Before we mark the board, we need to make sure that the spot they are trying to mark is empty (board[i][j] = 0). If it is, then we mark it by changing that spot in the 2D array to the player variable.

```
def markBoard(i, j):
    global board, player
    if board[i][j] == 0:
        board[i][j] = player
```

Next, since a spot was just marked, we need to switch the player variable. If player 1 just marked the board, it should now be player 2's turn. If player 2 just marked the board, it should now be player 1's turn.

```
def markBoard(i, j):
    global player, board
    if board[i][j] == 0:
        board[i][j] = player
        if (player == 1):
            player = 2
        else:
            player = 1
```

Let's test our function. For now, we will have get ask the player what spot to mark using the input function.

```
while not gameover:
    for row in board:
        print(row)
    print("Player", player, "'s turn")
    row = int(input("row: "))
    column = int(input("column: "))
    markBoard(row, column)
```

Run your program to test your code. Make sure that the player switches off and that the correct place on your board gets marked. Remember that the first row and column in a 2D array has an index of 0.

PART 3: Checking Who Won

Let's create a function that checks to see if someone has won the game. To check who won, we are going to need to write 8 conditionals! One for each row, column, and diagonal that can be used to win. We can create variables for each spot in our array.

tl	tc	tr
cl	cc	cr
bl	bc	br

```
def isGameOver():
    tl = board[0][0] # top-left
    tc = board[0][1] # top-center
    tr = board[0][2] # top-right
    cl = board[_][_] # center-left
    cc = board[_][_] # center-center
    cr = board[_][_] # center-right
    bl = board[_][_] # bottom-left
    bc = board[_][_] # bottom-center
    br = board[_][_] # bottom-right

while not gameover:
    ...
```

Next, let's right a conditional if a player has won by completely marking the first row. In the context of our 2D array, that would mean that all slots in the first row have the same number, as long as that number is not 0 because that would mean the spot is empty. The return keyword is used to give back an answer and end the function. In this case, we return true because someone has one.

```
def isGameOver():
    ...
    #CHECK ROWS
    if (tl != 0 and tl == tc and tc == tr): # top row
        return True
```

Try writing out the conditionals. For each conditional, you need to check that spots on the board are all equal to each other and that they not equal 0. If none of the conditionals are true, then we return False because the game is not over.

```
def isGameOver():
    ...
    #CHECK ROWS
    if (tl != 0 and tl == tc and tc == tr): # top row
        return True
    if (cl != _ and __ == __ and __ == __): # middle row
        return True
    if (bl != _ and __ == __ and __ == __): # bottom row
        return True

    #CHECK COLUMNS
    if (tl != _ and __ == __ and __ == __): # left column
        return True
    if (tc != _ and __ == __ and __ == __): # middle column
        return true
    if (tr != _ and __ == __ and __ == __): # right column
        return True

    #CHECK DIAGONALS
    if (tl != _ and __ == __ and __ == __): # left-right diagonal
        return True
    if (tr != _ and __ == __ and __ == __): # right-left diagonal
        return True
    return False
```

The last thing we need to do is call this function in our game loop. We can set the gameover variable equal to our function because it returns a value.

```
while not gameover:
    ...
    markBoard(row, column)
    gameover = isGameOver()
```

Our tic-tac-toe game is actually fully functional now on the shell. Test it out to make sure that it works before moving on. In the next part, we will make it more visual with tkinter.

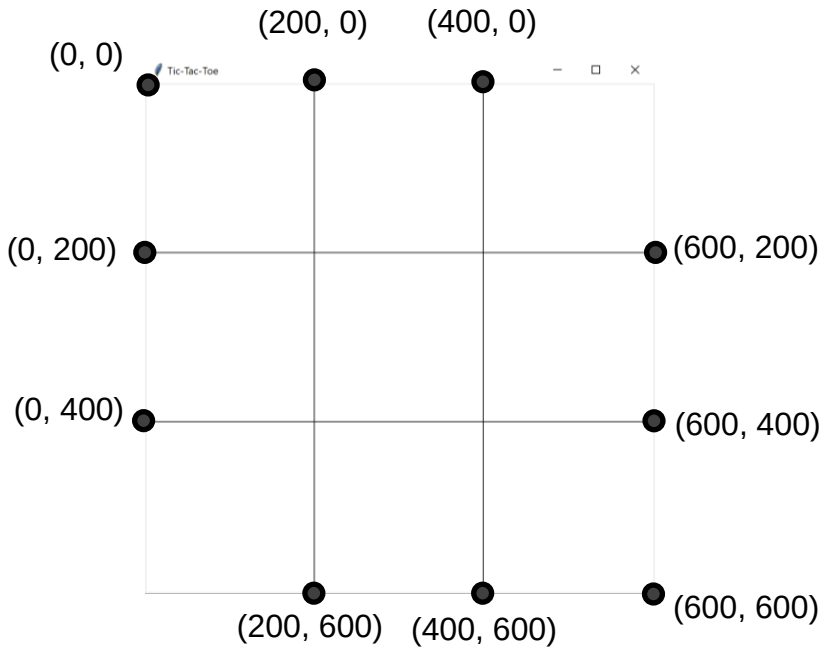
PART 4: Drawing the Grid and Detecting Mouse Clicks

Since we are using tkinter, the first thing we need to do is import it and set up our tk window with our canvas. Let's make the canvas 600 x 600. Add this code at the top of the page.

```
from tkinter import *

tk = Tk()
tk.title("Tic-Tac-Toe")
canvas = Canvas(tk, width = 600, height = 600, bg = "white")
canvas.pack()
```

Before we start drawing anything, let's take a look at how we want our tic-tac-toe board to look like. Each square on our board will be 200 x 200.



Now that we know the points of our lines, we can make a function that draws the tic-tac-toe grid. Since we only need to draw these lines once, we will call this function before our while loop. Run your program to make sure that your grid looks right. Don't forget to update your tk window at the end of your while loop!

```
def drawGrid():
    canvas.create_line(200, 0, 200, 600) #left vertical line
    canvas.create_line(400, 0, ___, ___) #right vertical line
    canvas.create_line(0, 200, ___, ___) #top horizontal line
    canvas.create_line(0, 400, ___, ___) #bottom horizontal line

drawGrid()
while not gameover:
    ...
    markBoard(row, column)
    gameover = isGameOver()
    tk.update()
```

Next, we need to make a function that can detect what square the player has clicked on. This function will be bound to a mouse click, so we can use the event parameter to get the mouse's x and y position when it was clicked.

```
def findAndMark(event):
    mouseX = event.x
    mouseY = event.y

drawGrid()
while not gameover:
```

Remember that we have a markBoard function that has two parameters: i and j, the row and column of our 2D array. We are going to need to use the mouse's position to determine what row and column we are in so we can call our markBoard function. First, first let's find out what column we are in using the mouse's x position. If the mouse's x position is less than 200, where the first vertical line is, then the mouse to left of that line and is in the first column so j = 0. What should j be if x < 400?

```
def findAndMark(event):
    mouseX = event.x
    mouseY = event.y
    i = j = 0
    if (x < 200):
        j = 0
```

```
    elif (x < 400):
        j = _
    else:
        j = 2
}
```

Similarly, we can use the mouse's Y position to figure out what row it is in. If the mouse's Y position is less than 200, where the top line is, then it is above the first line, so we are in the first row. Try to fill out the ret of the conditional yourself.

```
def findAndMark(event):
    ...
    elif (x < 600):
        j = 2
    if (y < 200):
        i = 0
    elif (y < ___):
        i = _
    else:
        i = _
}
```

The last part of the findAndMark function will be to call our markBoard function using the i and j that we have detected.

```
def findAndMark(event):
    else:
        i = _
    markBoard(i, j)
}
```

Next, we need to bind findAndMark to the mouse button and delete the lines of code that involve the shell in the while loop.

```
canvas.bind("<Button-1>", findAndMark)
while not gameover:
    for row in board:# DELETE
        print(row) #DELETE
    print("Player", player, "'s turn") #DELETE
    row = int(input("row: ")) #DELETE
    column = int(input("column: ")) #DELETE
    markBoard(row, column) #DELETE
```

```
    gameover = isGameOver())
```

PART 4: Drawing X and Os

To make sure our tkinter board always reflects the changes on our 2D array, we need a function that iterates through every spot in our 2D array. Remember to do that, we need two for loops: an outer one that goes through each one and an inner one that goes through each column.

```
def updateBoard():
    for i in range(3):
        for j in range(3):

drawGrid()
canvas.bind("<Button-1>", findAndMark)
```

Inside this function, we'll check if the spot in the array has a 1 in it. If it does, we will draw an X at that spot in the array. Otherwise, if it equals 2, then we will draw an O at that position. We will create the drawX and drawO functions next.

```
def updateBoard():
    for i in range(3):
        for j in range(3):
            if board[i][j] == 1:
                drawO(i, j)
            elif board[i][j] == 2:
                drawX(i, j)
```

To draw the X, we will draw 2 lines that cross over the square. Remember that the create_line function takes 4 arguments: the x and y position of one endpoint and the x and y position of the second endpoint. We can figure those points out based off the row and column of the square we want to draw the X in. Each square on our board is 200 x 200. So, we can just multiply i and j by 200 to find the y-position of the top side of our square and the x-position of the left side of our square. For the bottom and right sides all we need to do is add 200 to our top and left.

```
def drawX(i, j):
    left = j*200
    top = i*200
    right = left + 200
    bottom = top + 200
```

Now we can use these variables to draw two diagonal line. One that goes from the top-left to bottom-right and another that goes from the top-right to the bottom-left.

```
def drawX(i, j):
    left = j*200
    top = i*200
    right =left + 200
    bottom = top + 200
    canvas.create_line(left, top, right, bottom)
    canvas.create_line(right, top, left, bottom)
```

To draw the O, we can actually use the same code. Copy and paste your drawX function to make the drawO function. All you need to do is change the name, delete the last line, and change create_line to create_oval.

```
def drawO(i, j):
    left = j*200
    top = i*200
    right = left + 200
    bottom = top + 200
    canvas.create_oval(left, top, right, bottom)
```

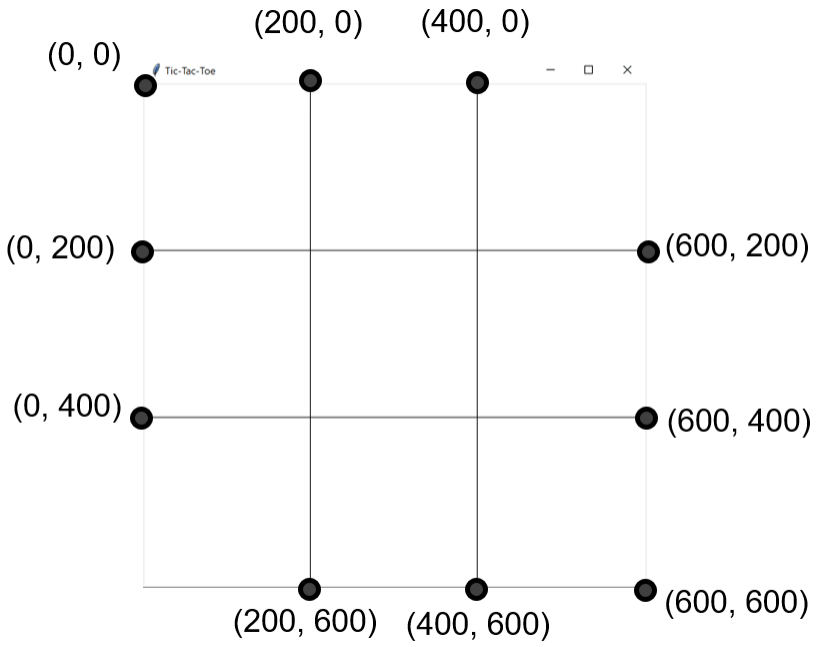
All that's left is to call our updateBoard function inside our while loop.

```
while not gameover:
    updateBoard()
    gameover = isGameOver()
    tk.update()
```

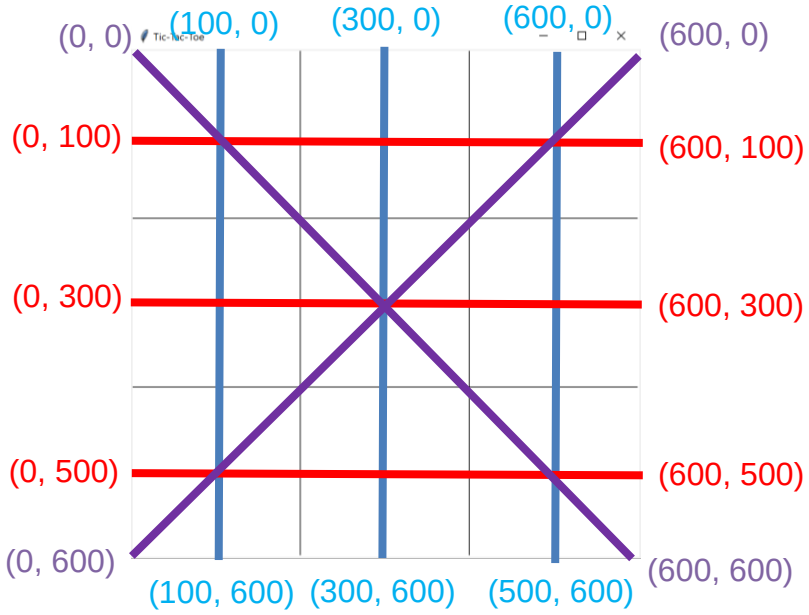
That's it! Your game is fully functional. If you have time, you can do the bonus section which draws the line that crosses over the winning group of symbols.

Bonus: Drawing a Line Across Winning Group

Let's take a look at our board again to remind ourselves of the position of the lines in our grid.



When a player wins, we will draw a line that goes in the middle of the lines shown above. The picture below shows you the coordinates of these lines. Columns are shown in blue, rows in red, and diagonals in purple.



These lines will get drawn when a player wins, so we will create them inside our isGameOver function. For each row, column and diagonal draw the appropriate line. Use the image above to help you. The width = 10 that you see here adjusts the thickness of the line that is drawn.

```
def isGameOver():
    ...
    #CHECK ROWS
    if (tl != 0 and tl == tc and tc == tr): # top row
        canvas.create_line(0, 100, 600, 100, width = 10)
        return True
    if (cl != 0 and cl == cc and cc == cr): # middle row
        canvas.create_line(0, 300, 600, 300, width = 10)
        return True
    if (bl != 0 and bl == bc and bc == br): # bottom row
        canvas.create_line(0, 500, 600, 500, width = 10)
        return True

    #CHECK COLUMNS
    if (tl != 0 and tl == cl and cl == bl): # left column
        canvas.create_line(___, _, ___, ___, width = 10)
        return True
    if (tc != 0 and tc == cc and cc == bc): # middle column
        canvas.create_line(___, _, ___, ___, width = 10)
        return true
    if (tr != 0 and tr == cr and cr == br): # right column
        canvas.create_line(___, _, ___, ___, width = 10)
        return True

    #CHECK DIAGONALS
    if (tl != 0 and tl == cc and cc == br): # left-right diagonal
        canvas.create_line(_, _, ___, ___, width = 10)
        return True
    if (tr != 0 and tr == cc and cc == bl): # right-left diagonal
        canvas.create_line(___, _, _, ___, width = 10)
        return True
    return False
```