Tics and Tacs and sometimes Toes

Presented by: Asrar Syed and Abir Chowdhury

Why did we choose Tic Tac Toe?????????????

Because it seemed the easiest and other stuff

We decided to not play the game in terminal as it would be simple and wanted to take things to another level.

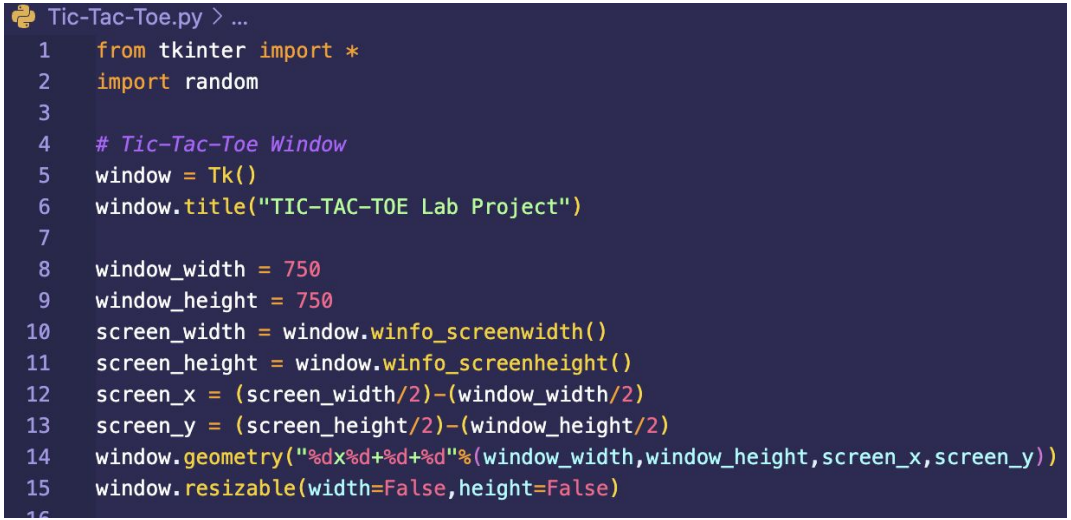
We decided we might as well add graphics and the way we did that was through using Tkinter.

Project Description

Together we both thought of a creative way to play a game of Tic Tac Toe using the python graphics module Tkinter. The program is made to allow for user inputs between two human players. The game randomly chooses between X and O and prompts the player of choosing to complete his/her turn. Players will take turns marking their spots from the available 9 spaces until a winner has been decided. After the game is complete and a winner has been decided the program will display a winner message. If the game ends in a tie the program will display a “TIE” message.

Creating the Window

In this slide we have a ss of our basic structure for the tkinter module. Diving into the code we first see that we are importing the tkinter module and all the functions that are in the module. Then we create the window size.



Special features in the game

After a winner has already been decided the game will not allow for any more inputs denying the user the ability to select another space. The game also features a “RESTART” button where the user is able to restart the game at any given point.

First section

Diving into the code we first see that we are importing the tkinter module and all the functions that are in the module.

Then we see the four core functions that keep the game running smoothly.

The first function determines whos turn it is.

The second function checks for a winner.

The third function verifies that the selected space is empty.

The fourth function allows for a new game.

Player Turn Function

To determine which players turn it is we created a player turn function. The function checks for both players(X and Y). It first checks if the space/button selected is available. If the button is available it then calls the “Check for winner” function to determine if the game is finished or still undecided. If the game is undecided then the program will switch players.

```
global turn
global xWins
global yWins
global ties

if buttons[row][column]['text'] == "" and winner_check() is False:

    if turn == players[0]:

        buttons[row][column]['text'] = turn

        if winner_check() is False:
            turn = players[1]
            ongoing.config(text=(players[1]+"'s turn"))

        elif winner_check() is True:
            ongoing.config(text=(players[0]+" wins"))
            xWins += 1

        elif winner_check() == "Tie":
            ongoing.config(text="Tie!")
            ties += 1

    else:

        buttons[row][column]['text'] = turn

        if winner_check() is False:
            turn = players[0]
            ongoing.config(text=(players[0]+"'s turn"))

        elif winner_check() is True:
            ongoing.config(text=(players[1]+" wins"))
            yWins += 1

        elif winner_check() == "Tie":
            ongoing.config(text="Tie Draw!")
            ties += 1
```

Checking for a Winner

In the function created to check for a winner we used a series of “if” statements to check for matching patterns across the board. The program then returns True or False statements, determining whether or not a pattern is present. If there is no pattern present the code returns a false statement resulting in a TIE.

*Some of the code was recycled from Lab06

```
# Function to find the Winner - I took some ideas from Lab 6 and reworked them to check all test cases
# Checks for both X and Y
def winner_check():

    # 3 ways to win Horizontal
    for row in range(3):
        if buttons[row][0]['text'] == buttons[row][1]['text'] == buttons[row][2]['text'] != "":
            return True

    # 3 ways to win Vertical
    for column in range(3):
        if buttons[0][column]['text'] == buttons[1][column]['text'] == buttons[2][column]['text'] != "":
            return True

    # 2 ways to win Diagonal
    if buttons[0][0]['text'] == buttons[1][1]['text'] == buttons[2][2]['text'] != "":
        return True

    elif buttons[0][2]['text'] == buttons[1][1]['text'] == buttons[2][0]['text'] != "":
        return True

    # Ways to Tie
    elif blank_spaces() == False:
        return 'Tie'

    else:
        return False
```

Second section

Following our functions we have our buttons and accessories such as the prompt messages and reset buttons.

In our game we created a button for each one of the 9 spaces on the board. To create our buttons we used two nested for loops and a list. Since we used a list in our code, we only needed two parameters in our for loops; one is to assign the row and the other to assign the column for each button on the board.

```
# players
players = ["X","O"]
turn = random.choice(players)

# Button list For Storing
buttons = [[0,0,0],[0,0,0],[0,0,0]]

# Font 1
Font1 = ("Comic Sans MS", 25, "bold")

# Labels
ongoing = Label(text=turn + "'s turn", font=(Font1))
ongoing.pack(ipadx=15, ipady=15, anchor=CENTER)

# Font 2
Font2 = ("Comic Sans MS", 60, "bold")

# Frame and Grid of Buttons
board = Frame(window, padx=20,pady=20)
board.pack()

for row in range(3):
    for column in range(3):
        buttons[row][column] = Button(board, text="",font=(Font2), width=4, height=2, highlightbackground="#C1E1C1",
                                      command= lambda row=row, column=column: player_turn(row,column))
        buttons[row][column].grid(row=row,column=column)
```