**Tic Tac Toe GUI with Python**

**In this section, we introduce you to a free Python project that creates a graphical user interface (GUI) for the Tic Tac Toe game. This game is very popular and very simple in itself. This is a two player game where there is a board with 3x3 squares .**

**In this game, the player can choose one between two symbols. Normally, X and O are used for this game. If the first player chooses X , the second player must play O and vice versa .**

**The player marks one of the 3x3 squares with his symbol (X or O) and aims to create a straight line horizontally, vertically or diagonally, with two goals :**

* **To win the game, make a straight line before your opponent .**
* **Prevent your competitor from creating a straight line .**

**If no one can make a straight line, the game ends in a draw. Therefore, only three outcomes can occur: one player wins, his opponent (human or computer) wins, or the game ends in a draw .**

**to create a GUI for this game ?**

**There are two basic logics in this game. Either both players are human or one is computer. I will prepare this GUI with Python for two people. Here is the complete code for this :**

**from tkinter import \***

**import numpy as np**

**size\_of\_board = 600**

**symbol\_size = ( size\_of\_board / 3 - size\_of\_board / 8 ) / 2**

**symbol\_thickness = 50**

**symbol\_X\_color = '#EE4035'**

**symbol\_O\_color = '#0492CF'**

**Green\_color = '#7BC043'**

**class Tic\_Tac\_Toe ():**

**# ------------------------------------------------- -----------------**

**# Initialization Functions:**

**# ------------------------------------------------- -----------------**

**def \_\_ init \_\_( self ):**

**self . window = Tk ()**

**self . windows . title ( 'Tic-Tac-Toe' )**

**self . canvas = Canvas ( self . window , width = size\_of\_board , height = size\_of\_board )**

**self . canvas . package ()**

**# Input from the user in the form of clicks**

**self . windows . bind ( '<Button-1>' , self . click )**

**self . initialize\_board ()**

**self . player\_X\_turns = True**

**self . board\_status = n.p. \_ zeros ( shape = ( 3 , 3 ))**

**self . player\_X\_starts = True**

**self . reset\_board = False**

**self . gameover = False**

**self . tie = False**

**self . X\_wins = False**

**self . O\_wins = False**

**self . X\_score = 0**

**self . O\_score = 0**

**self . tie\_score = 0**

**def mainloop ( self ):**

**self . windows . mainloop ()**

**def initialize\_board ( self ):**

**for i in range ( 2 ):**

**self . canvas . create\_line (( i + 1 ) \* size\_of\_board / 3 , 0 , ( i + 1 ) \* size\_of\_board / 3 , size\_of\_board )**

**for i in range ( 2 ):**

**self . canvas . create\_line ( 0 , ( i + 1 ) \* size\_of\_board / 3 , size\_of\_board , ( i + 1 ) \* size\_of\_board / 3 )**

**def play\_again ( self ):**

**self . initialize\_board ()**

**self . player\_X\_starts = not self . player\_X\_starts**

**self . player\_X\_turns = self . player\_X\_starts**

**self . board\_status = n.p. \_ zeros ( shape = ( 3 , 3 ))**

**# ------------------------------------------------- -----------------**

**# Drawing Functions:**

**# The modules required to draw required game based object on canvas**

**# ------------------------------------------------- -----------------**

**def draw\_O ( self , logical\_position ):**

**logical\_position = n.p. \_ array ( logical\_position )**

**# logical\_position = grid value on the board**

**# grid\_position = actual pixel values of the center of the grid**

**grid\_position = self . convert\_logical\_to\_grid\_position ( logical\_position )**

**self . canvas . create\_oval ( grid\_position [ 0 ] - symbol\_size , grid\_position [ 1 ] - symbol\_size ,**

**grid\_position [ 0 ] + symbol\_size , grid\_position [ 1 ] + symbol\_size , width = symbol\_thickness ,**

**outline = symbol\_O\_color )**

**def draw\_X ( self , logical\_position ):**

**grid\_position = self . convert\_logical\_to\_grid\_position ( logical\_position )**

**self . canvas . create\_line ( grid\_position [ 0 ] - symbol\_size , grid\_position [ 1 ] - symbol\_size ,**

**grid\_position [ 0 ] + symbol\_size , grid\_position [ 1 ] + symbol\_size , width = symbol\_thickness ,**

**fill = symbol\_X\_color )**

**self . canvas . create\_line ( grid\_position [ 0 ] - symbol\_size , grid\_position [ 1 ] + symbol\_size ,**

**grid\_position [ 0 ] + symbol\_size , grid\_position [ 1 ] - symbol\_size , width = symbol\_thickness ,**

**fill = symbol\_X\_color )**

**def display\_gameover ( self ):**

**if self . X\_wins :**

**self . X\_score += 1**

**text = 'Winner: Player 1 (X)'**

**color = symbol\_X\_color**

**elif self . O\_wins :**

**self . O\_score += 1**

**text = 'Winner: Player 2 (O)'**

**color = symbol\_O\_color**

**else :**

**self . tie\_score += 1**

**text = ' Its a tie'**

**color = gray**

**self . canvas . delete ( "all" )**

**self . canvas . create\_text ( size\_of\_board / 2 , size\_of\_board / 3 , font = " cmr 60 bold" , fill = color , text = text )**

**score\_text = 'Scores\n'**

**self . canvas . create\_text ( size\_of\_board / 2 , 5 \* size\_of\_board / 8 , font = " cmr 40 bold" , fill = Green\_color ,**

**text = score\_text )**

**score\_text = 'Player 1 (X) : ' + str ( self . X\_score ) + '\n'**

**score\_text += 'Player 2 (O): ' + str ( self . O\_score ) + '\n'**

**score\_text += 'Tie : ' + str ( self . tie\_score )**

**self . canvas . create\_text ( size\_of\_board / 2 , 3 \* size\_of\_board / 4 , font = " cmr 30 bold" , fill = Green\_color ,**

**text = score\_text )**

**self . reset\_board = True**

**score\_text = 'Click to play again'**

**self . canvas . create\_text ( size\_of\_board / 2 , 15 \* size\_of\_board / 16 , font = " cmr 20 bold" , fill = "gray" ,**

**text = score\_text )**

**# ------------------------------------------------- -----------------**

**# Logical Functions:**

**# The modules required to carry out game logic**

**# ------------------------------------------------- -----------------**

**def convert\_logical\_to\_grid\_position ( self , logical\_position ):**

**logical\_position = n.p. \_ array ( logical\_position , dtype = int )**

**return ( size\_of\_board / 3 ) \* logical\_position + size\_of\_board / 6**

**def convert\_grid\_to\_logical\_position ( self , grid\_position ):**

**grid\_position = n.p. \_ array ( grid\_position )**

**return n.p. \_ array ( grid\_position // ( size\_of\_board / 3 ), dtype = int )**

**def is\_grid\_occupied ( self , logical\_position ):**

**if self . board\_status [ logical\_position [ 0 ]][ logical\_position [ 1 ]] == 0 :**

**return False**

**else :**

**return True**

**def is\_winner ( self , player ):**

**player = - 1 if player == 'X' otherwise 1**

**# Three in a row**

**for i in range ( 3 ):**

**if self . board\_status [ i ][ 0 ] == self . board\_status [ i ][ 1 ] == self . board\_status [ i ][ 2 ] == player :**

**return True**

**if self . board\_status [ 0 ][ i ] == self . board\_status [ 1 ][ i ] == self . board\_status [ 2 ][ i ] == player :**

**return True**

**# Diagonals**

**if self . board\_status [ 0 ][ 0 ] == self . board\_status [ 1 ][ 1 ] == self . board\_status [ 2 ][ 2 ] == player :**

**return True**

**if self . board\_status [ 0 ][ 2 ] == self . board\_status [ 1 ][ 1 ] == self . board\_status [ 2 ][ 0 ] == player :**

**return True**

**return False**

**def is\_tie ( self ):**

**r , c = n.p. \_ where ( self . board\_status == 0 )**

**tie = False**

**if len ( r ) == 0 :**

**tie = True**

**return tie**

**def is\_gameover ( self ):**

**# Either someone wins or all the grid is occupied**

**self . X\_wins = self . is\_winner ( 'X' )**

**if not self . X\_wins :**

**self . O\_wins = self . is\_winner ( 'O' )**

**if not self . O\_wins :**

**self . tie = self . is\_tie ()**

**gameover = self . X\_wins or self . O\_wins or self . tie**

**if self . X\_wins :**

**print ( 'X wins' )**

**if self . O\_wins :**

**print ( 'O wins' )**

**if self . tie :**

**print ( ' Its a tie' )**

**return gameover**

**def click( self , event ):**

**grid\_position = [ event . x \_ event . y ]**

**logical\_position = self . convert\_grid\_to\_logical\_position ( grid\_position )**

**if not self . reset\_board :**

**if self . player\_X\_turns :**

**if not self . is\_grid\_occupied ( logical\_position ):**

**self . draw\_X ( logical\_position )**

**self . board\_status [ logical\_position [ 0 ]][ logical\_position [ 1 ]] = - 1**

**self . player\_X\_turns = not self . player\_X\_turns**

**else :**

**if not self . is\_grid\_occupied ( logical\_position ):**

**self . draw\_O ( logical\_position )**

**self . board\_status [ logical\_position [ 0 ]][ logical\_position [ 1 ]] = 1**

**self . player\_X\_turns = not self . player\_X\_turns**

**# Check if the game is concluded**

**if self . is\_gameover ():**

**self . display\_gameover ()**

**# print('Done')**

**else : # Play Again**

**self . canvas . delete ( "all" )**

**self . play\_again ()**

**self . reset\_board = False**

**game\_instance = Tic\_Tac\_Toe ()**

**game\_instance . mainloop ()**

Amirshayan Jalili

[Shayan138190@gmail.com](mailto:Shayan138190@gmail.com)

<https://github.com/Amirshayan2002>