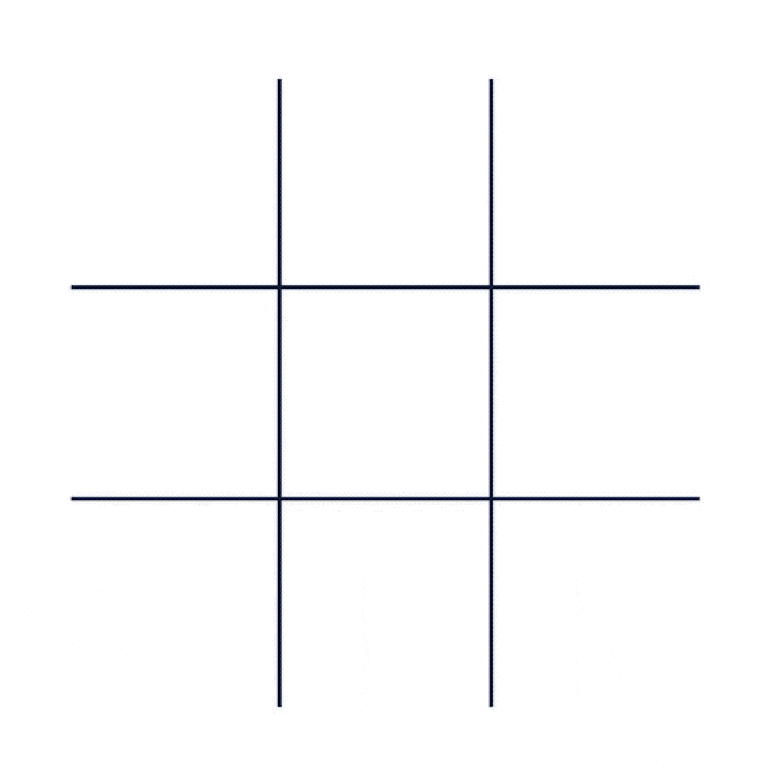
**Tic Tac Toe GUI with Python**

In this article, I will introduce you to an advanced Python project on Tic Tac Toe GUI with Python. This game is very popular and is quite simple in itself. It is a two-player game. In this game, there is a board with 3×3 squares.

[](tic-tac-toe.webp)

In this game a player can choose between two symbols with his opponent, the usual games use “X” and “O”. If the first player chooses “X”, then the second player must play with “O” and vice versa.

A player marks one of the 3×3 squares with his symbol (perhaps “X” or “O”) and he aims to create a straight line horizontally or vertically or diagonally with two intensions:

1. Create a straight line before your opponent to win the game.
2. Prevent his opponent from creating a straight line first.

If no one can logically create a straight line with its symbol, the game ends in a tie. So there are only three possible outcomes: one player wins, his opponent (human or computer) wins, or there is a tie.

## Tic Tac Toe GUI with Python

There are two basic logics in this game; when both players are human, and when one is a computer. I will prepare this Tic Tac Toe GUI with Python for two players.

1. from tkinter import \*
2. import numpy as np
4. size\_of\_board = 600
5. symbol\_size = (size\_of\_board / 3 - size\_of\_board / 8) / 2
6. symbol\_thickness = 50
7. symbol\_X\_color = '#EE4035'
8. symbol\_O\_color = '#0492CF'
9. Green\_color = '#7BC043'

12. **class** Tic\_Tac\_Toe():
13. # ------------------------------------------------------------------
14. # Initialization Functions:
15. # ------------------------------------------------------------------
16. def \_\_init\_\_(self):
17. self.window = Tk()
18. self.window.title('Tic-Tac-Toe')
19. self.canvas = Canvas(self.window, width=size\_of\_board, height=size\_of\_board)
20. self.canvas.pack()
21. # Input from user in form of clicks
22. self.window.bind('<Button-1>', self.click)
24. self.initialize\_board()
25. self.player\_X\_turns = True
26. self.board\_status = np.zeros(shape=(3, 3))
28. self.player\_X\_starts = True
29. self.reset\_board = False
30. self.gameover = False
31. self.tie = False
32. self.X\_wins = False
33. self.O\_wins = False
35. self.X\_score = 0
36. self.O\_score = 0
37. self.tie\_score = 0
39. def mainloop(self):
40. self.window.mainloop()
42. def initialize\_board(self):
43. **for** i in range(2):
44. self.canvas.create\_line((i + 1) \* size\_of\_board / 3, 0, (i + 1) \* size\_of\_board / 3, size\_of\_board)
46. **for** i in range(2):
47. self.canvas.create\_line(0, (i + 1) \* size\_of\_board / 3, size\_of\_board, (i + 1) \* size\_of\_board / 3)
49. def play\_again(self):
50. self.initialize\_board()
51. self.player\_X\_starts = not self.player\_X\_starts
52. self.player\_X\_turns = self.player\_X\_starts
53. self.board\_status = np.zeros(shape=(3, 3))
55. # ------------------------------------------------------------------
56. # Drawing Functions:
57. # The modules required to draw required game based object on canvas
58. # ------------------------------------------------------------------
60. def draw\_O(self, logical\_position):
61. logical\_position = np.array(logical\_position)
62. # logical\_position = grid value on the board
63. # grid\_position = actual pixel values of the center of the grid
64. grid\_position = self.convert\_logical\_to\_grid\_position(logical\_position)
65. self.canvas.create\_oval(grid\_position[0] - symbol\_size, grid\_position[1] - symbol\_size,
66. grid\_position[0] + symbol\_size, grid\_position[1] + symbol\_size, width=symbol\_thickness,
67. outline=symbol\_O\_color)
69. def draw\_X(self, logical\_position):
70. grid\_position = self.convert\_logical\_to\_grid\_position(logical\_position)
71. self.canvas.create\_line(grid\_position[0] - symbol\_size, grid\_position[1] - symbol\_size,
72. grid\_position[0] + symbol\_size, grid\_position[1] + symbol\_size, width=symbol\_thickness,
73. fill=symbol\_X\_color)
74. self.canvas.create\_line(grid\_position[0] - symbol\_size, grid\_position[1] + symbol\_size,
75. grid\_position[0] + symbol\_size, grid\_position[1] - symbol\_size, width=symbol\_thickness,
76. fill=symbol\_X\_color)
78. def display\_gameover(self):
80. **if** self.X\_wins:
81. self.X\_score += 1
82. text = 'Winner: Player 1 (X)'
83. color = symbol\_X\_color
84. elif self.O\_wins:
85. self.O\_score += 1
86. text = 'Winner: Player 2 (O)'
87. color = symbol\_O\_color
88. **else**:
89. self.tie\_score += 1
90. text = 'Its a tie'
91. color = 'gray'
93. self.canvas.**delete**("all")
94. self.canvas.create\_text(size\_of\_board / 2, size\_of\_board / 3, font="cmr 60 bold", fill=color, text=text)
96. score\_text = 'Scores \n'
97. self.canvas.create\_text(size\_of\_board / 2, 5 \* size\_of\_board / 8, font="cmr 40 bold", fill=Green\_color,
98. text=score\_text)
100. score\_text = 'Player 1 (X) : ' + str(self.X\_score) + '\n'
101. score\_text += 'Player 2 (O): ' + str(self.O\_score) + '\n'
102. score\_text += 'Tie                    : ' + str(self.tie\_score)
103. self.canvas.create\_text(size\_of\_board / 2, 3 \* size\_of\_board / 4, font="cmr 30 bold", fill=Green\_color,
104. text=score\_text)
105. self.reset\_board = True
107. score\_text = 'Click to play again \n'
108. self.canvas.create\_text(size\_of\_board / 2, 15 \* size\_of\_board / 16, font="cmr 20 bold", fill="gray",
109. text=score\_text)
111. # ------------------------------------------------------------------
112. # Logical Functions:
113. # The modules required to carry out game logic
114. # ------------------------------------------------------------------
116. def convert\_logical\_to\_grid\_position(self, logical\_position):
117. logical\_position = np.array(logical\_position, dtype=**int**)
118. **return** (size\_of\_board / 3) \* logical\_position + size\_of\_board / 6
120. def convert\_grid\_to\_logical\_position(self, grid\_position):
121. grid\_position = np.array(grid\_position)
122. **return** np.array(grid\_position // (size\_of\_board / 3), dtype=int)
124. def is\_grid\_occupied(self, logical\_position):
125. **if** self.board\_status[logical\_position[0]][logical\_position[1]] == 0:
126. **return** False
127. **else**:
128. **return** True
130. def is\_winner(self, player):
132. player = -1 **if** player == 'X' **else** 1
134. # Three in a row
135. **for** i in range(3):
136. **if** self.board\_status[i][0] == self.board\_status[i][1] == self.board\_status[i][2] == player:
137. **return** True
138. **if** self.board\_status[0][i] == self.board\_status[1][i] == self.board\_status[2][i] == player:
139. **return** True
141. # Diagonals
142. **if** self.board\_status[0][0] == self.board\_status[1][1] == self.board\_status[2][2] == player:
143. **return** True
145. **if** self.board\_status[0][2] == self.board\_status[1][1] == self.board\_status[2][0] == player:
146. **return** True
148. **return** False
150. def is\_tie(self):
152. r, c = np.where(self.board\_status == 0)
153. tie = False
154. **if** len(r) == 0:
155. tie = True
157. **return** tie
159. def is\_gameover(self):
160. # Either someone wins or all grid occupied
161. self.X\_wins = self.is\_winner('X')
162. **if** not self.X\_wins:
163. self.O\_wins = self.is\_winner('O')
165. **if** not self.O\_wins:
166. self.tie = self.is\_tie()
168. gameover = self.X\_wins or self.O\_wins or self.tie
170. **if** self.X\_wins:
171. print('X wins')
172. **if** self.O\_wins:
173. print('O wins')
174. **if** self.tie:
175. print('Its a tie')
177. **return** gameover




183. def click(self, event):
184. grid\_position = [event.x, event.y]
185. logical\_position = self.convert\_grid\_to\_logical\_position(grid\_position)
187. **if** not self.reset\_board:
188. **if** self.player\_X\_turns:
189. **if** not self.is\_grid\_occupied(logical\_position):
190. self.draw\_X(logical\_position)
191. self.board\_status[logical\_position[0]][logical\_position[1]] = -1
192. self.player\_X\_turns = not self.player\_X\_turns
193. **else**:
194. **if** not self.is\_grid\_occupied(logical\_position):
195. self.draw\_O(logical\_position)
196. self.board\_status[logical\_position[0]][logical\_position[1]] = 1
197. self.player\_X\_turns = not self.player\_X\_turns
199. # Check if game is concluded
200. **if** self.is\_gameover():
201. self.display\_gameover()
202. # print('Done')
203. **else**:  # Play Again
204. self.canvas.**delete**("all")
205. self.play\_again()
206. self.reset\_board = False

209. game\_instance = Tic\_Tac\_Toe()
210. game\_instance.mainloop()