|  |
| --- |
| 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.window.title('Tic-Tac-Toe') |
|  |

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

|  |
| --- |
| self.canvas.pack() |
|  |

|  |
| --- |
| # Input from user in form of clicks |
|  |

|  |
| --- |
| self.window.bind('<Button-1>', self.click) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| self.initialize\_board() |
|  |

|  |
| --- |
| self.player\_X\_turns = True |
|  |

|  |
| --- |
| self.board\_status = np.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.window.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 = np.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 = np.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 \n' |
|  |

|  |
| --- |
| 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 = np.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 = np.array(grid\_position) |
|  |

|  |
| --- |
| return np.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' else 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 = np.where(self.board\_status == 0) |
|  |

|  |
| --- |
| tie = False |
|  |

|  |
| --- |
| if len(r) == 0: |
|  |

|  |
| --- |
| tie = True |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return tie |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def is\_gameover(self): |
|  |

|  |
| --- |
| # Either someone wins or all grid 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 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()