# AI ASSIGNMENT -04

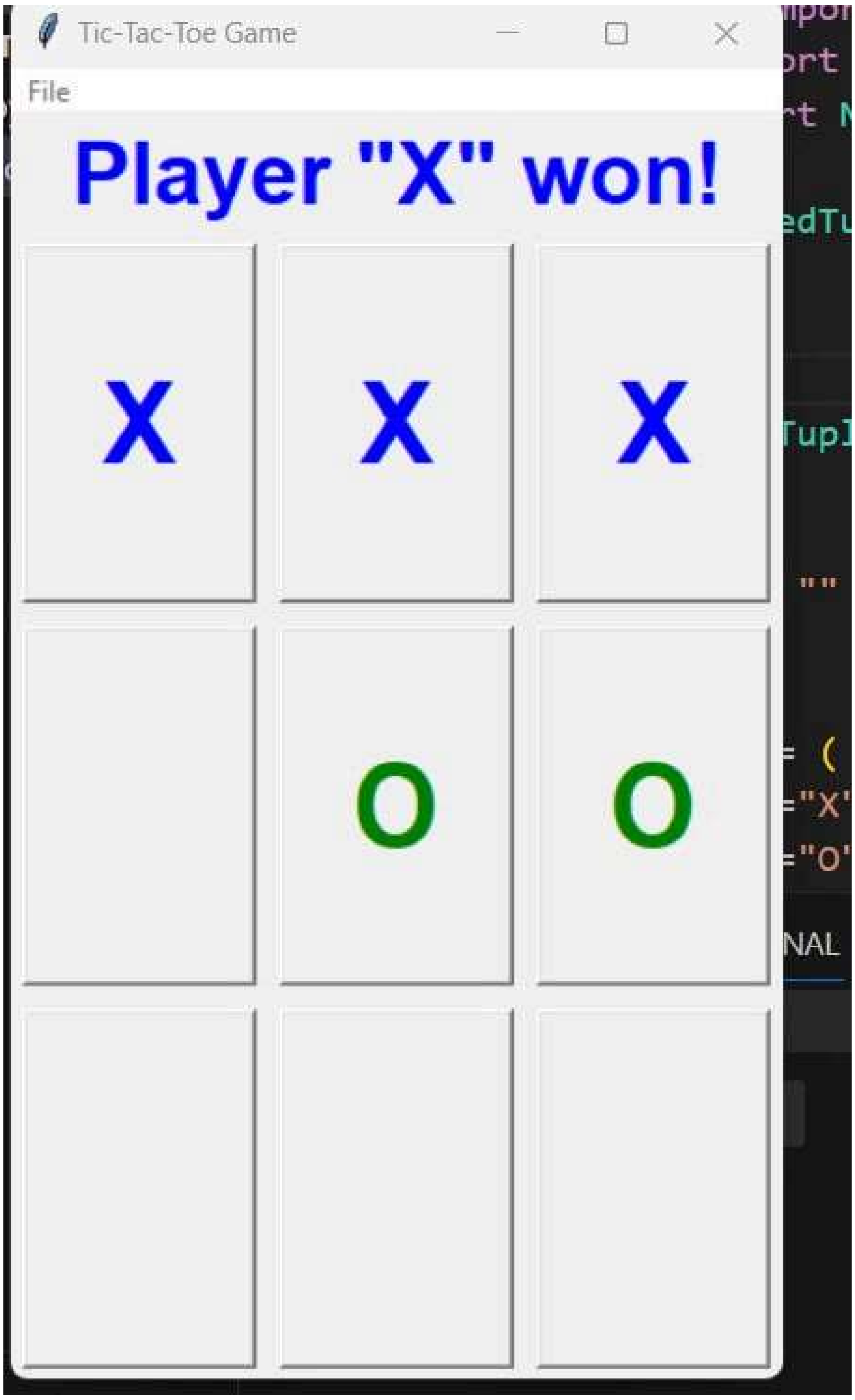
ROLL NO: U21CS052

NAME : PANCHAL GUNGUN PARESH

## TIC-TAC-TOE

```python
"""A tic-tac-toe game built with Python and Tkinter."""

import tkinter as tk
from itertools import cycle
from tkinter import font
from typing import NamedTuple

class Player(NamedTuple):
    label: str
    color: str

class Move(NamedTuple):
    row: int
    col: int
    label: str = ""

BOARD_SIZE = 3
DEFAULT_PLAYERS = (
    Player(label="X", color="blue"),
    Player(label="O", color="green"),
)

class TicTacToeGame:
    def __init__(self, players=DEFAULT_PLAYERS, board_size=BOARD_SIZE):
        self._players = cycle(players)
        self.board_size = board_size
        self.current_player = next(self._players)
        self.winner_combo = []
        self._current_moves = []
        self._has_winner = False
        self._winning_combos = []
        self._setup_board()

    def _setup_board(self):
        self._current_moves = [
            [Move(row, col) for col in range(self.board_size)]
            for row in range(self.board_size)
        ]
        self._winning_combos = self._get_winning_combos()

    def _get_winning_combos(self):
        rows = [
            [(move.row, move.col) for move in row]
            for row in self._current_moves
        ]
        columns = [list(col) for col in zip(*rows)]
        first_diagonal = [row[i] for i, row in enumerate(rows)]
        second_diagonal = [col[j] for j, col in
enumerate(reversed(columns))]
        return rows + columns + [first_diagonal, second_diagonal]

    def toggle_player(self):
        """Return a toggled player."""
        self.current_player = next(self._players)

    def is_valid_move(self, move):
        """Return True if move is valid, and False otherwise."""
        row, col = move.row, move.col
        move_was_not_played = self._current_moves[row][col].label == ""
        no_winner = not self._has_winner
        return no_winner and move_was_not_played

    def process_move(self, move):
        """Process the current move and check if it's a win."""
        row, col = move.row, move.col
        self._current_moves[row][col] = move
        for combo in self._winning_combos:
            results = set(self._current_moves[n][m].label for n, m in
combo)
```

```
            is_win = (len(results) == 1) and ("" not in results)
            if is_win:
                self._has_winner = True
                self.winner_combo = combo
                break

    def has_winner(self):
        """Return True if the game has a winner, and False otherwise."""
        return self._has_winner

    def is_tied(self):
        """Return True if the game is tied, and False otherwise."""
        no_winner = not self._has_winner
        played_moves = (
            move.label for row in self._current_moves for move in row
        )
        return no_winner and all(played_moves)

    def reset_game(self):
        """Reset the game state to play again."""
        for row, row_content in enumerate(self._current_moves):
            for col, _ in enumerate(row_content):
                row_content[col] = Move(row, col)
        self._has_winner = False
        self.winner_combo = []

class TicTacToeBoard(tk.Tk):
    def __init__(self, game):
        super().__init__()
        self.title("Tic-Tac-Toe Game")
        self._cells = {}
        self._game = game
        self._create_menu()
        self._create_board_display()
        self._create_board_grid()

    def _create_menu(self):
        menu_bar = tk.Menu(master=self)
        self.config(menu=menu_bar)
        file_menu = tk.Menu(master=menu_bar)
        file_menu.add_command(label="Play Again",
command=self.reset_board)
        file_menu.add_separator()
        file_menu.add_command(label="Exit", command=quit)
        menu_bar.add_cascade(label="File", menu=file_menu)

    def _create_board_display(self):
        display_frame = tk.Frame(master=self)
        display_frame.pack(fill=tk.X)
        self.display = tk.Label(
            master=display_frame,
            text="Ready?",
            font=font.Font(size=28, weight="bold"),
        )
        self.display.pack()

    def _create_board_grid(self):
        grid_frame = tk.Frame(master=self)
        grid_frame.pack()
        for row in range(self._game.board_size):
            self.rowconfigure(row, weight=1, minsize=50)
            self.columnconfigure(row, weight=1, minsize=75)
            for col in range(self._game.board_size):
                button = tk.Button(
                    master=grid_frame,
                    text="",
                    font=font.Font(size=36, weight="bold"),
                    fg="black",
                    width=3,
                    height=2,
                    highlightbackground="lightblue",
                )
                self._cells[button] = (row, col)
                button.bind("<ButtonPress-1>", self.play)
                button.grid(row=row, column=col, padx=5, pady=5,
sticky="nsew")

    def play(self, event):
        """Handle a player's move."""
        clicked_btn = event.widget
        row, col = self._cells[clicked_btn]
        move = Move(row, col, self._game.current_player.label)
        if self._game.is_valid_move(move):
            self._update_button(clicked_btn)
            self._game.process_move(move)
            if self._game.is_tied():
                self._update_display(msg="Tied game!", color="red")
            elif self._game.has_winner():
                self._highlight_cells()
                msg = f'Player "{self._game.current_player.label}" won!'
                color = self._game.current_player.color
                self._update_display(msg, color)
            else:
                self._game.toggle_player()
                msg = f"{self._game.current_player.label}'s turn"
                self._update_display(msg)

    def _update_button(self, clicked_btn):
        clicked_btn.config(text=self._game.current_player.label)
        clicked_btn.config(fg=self._game.current_player.color)

    def _update_display(self, msg, color="black"):
        self.display["text"] = msg
        self.display["fg"] = color

    def _highlight_cells(self):
        for button, coordinates in self._cells.items():
            if coordinates in self._game.winner_combo:
                button.config(highlightbackground="red")

    def reset_board(self):
        """Reset the game's board to play again."""
        self._game.reset_game()
        self._update_display(msg="Ready?")
        for button in self._cells.keys():
            button.config(highlightbackground="lightblue")
            button.config(text="")
            button.config(fg="black")

def main():
    """Create the game's board and run its main loop."""
    game = TicTacToeGame()
    board = TicTacToeBoard(game)
    board.mainloop()

if __name__ == "__main__":
    main()
```


Tic-Tac-Toe Game
File
Player "X" won!
X
X
X
O
O