Python Project 03: Tic Tac Toe Game

1. Project Overview

In this project, we will build a complete **Tic Tac Toe Game** using Python and Tkinter.

The project separates the **frontend UI** from the **game logic**. Tkinter handles the interface and button events, while the backend manages turns, winning combinations, draws, and game state. Tkinter supports event bindings and GUI widgets suitable for this type of interactive application. (Python documentation)

---

2. Features

- Two player game
- X and O turns
- 3 × 3 game board
- Winner detection
- Draw detection
- Score tracking
- Restart round
- Reset scores
- Winning line highlight
- Modern dark UI
- Button hover effect

---

3. Technologies

- Python 3
- Tkinter

- Object Oriented Programming

No external packages are required.

---

4. Folder Structure

```
tic_tac_toe/
□
□□□ main.py
□□□ ui.py
□□□ game.py
□□□ README.md
```

---

5. How the Game Works

The backend stores the board as a list of 9 positions:

```
0 | 1 | 2
---------
3 | 4 | 5
---------
6 | 7 | 8
```

For example:

```
["X", "", "O",
 "", "X", "",
 "O", "", ""]
```

After every move, the backend checks:

1. Is the selected position empty?
2. Place the current player's symbol.
3. Check all winning combinations.
4. Check whether the board is full.
5. Switch the player if the game continues.

---

6. Frontend Code

ui.py

```
import tkinter as tk
from game import TicTacToe


class TicTacToeUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Tic Tac Toe")
        self.root.resizable(False, False)
        self.root.configure(bg="#0f172a")

        self.game = TicTacToe()

        self.create_ui()
        self.update_ui()

    def create_ui(self):
        title = tk.Label(
            self.root,
            text="TIC TAC TOE",
            font=("Arial", 24, "bold"),
            fg="#38bdf8",
```

```
            bg="#0f172a"
        )
        title.pack(pady=(20, 5))

        self.status_label = tk.Label(
            self.root,
            text="Player X's Turn",
            font=("Arial", 14, "bold"),
            fg="white",
            bg="#0f172a"
        )
        self.status_label.pack(pady=(0, 15))

        score_frame = tk.Frame(
            self.root,
            bg="#0f172a"
        )
        score_frame.pack(pady=(0, 15))

        self.x_score_label = tk.Label(
            score_frame,
            text="X: 0",
            font=("Arial", 12, "bold"),
            fg="#38bdf8",
            bg="#0f172a"
        )
        self.x_score_label.pack(side="left", padx=30)

        self.o_score_label = tk.Label(
            score_frame,
            text="O: 0",
            font=("Arial", 12, "bold"),
            fg="#f472b6",
            bg="#0f172a"
```

```
        )
        self.o_score_label.pack(side="left", padx=30)

        board = tk.Frame(
            self.root,
            bg="#334155",
            padx=4,
            pady=4
        )
        board.pack()

        self.buttons = []

        for index in range(9):
            button = tk.Button(
                board,
                text="",
                font=("Arial", 32, "bold"),
                width=4,
                height=2,
                bg="#1e293b",
                fg="white",
                activebackground="#334155",
                activeforeground="white",
                relief="flat",
                borderwidth=0,
                command=lambda i=index: self.make_move(i),
                cursor="hand2"
            )

            button.grid(
                row=index // 3,
                column=index % 3,
                padx=3,
```

```
            pady=3
        )

        self.buttons.append(button)

    self.message_label = tk.Label(
        self.root,
        text="Click a square to make your move",
        font=("Arial", 10),
        fg="#94a3b8",
        bg="#0f172a"
    )
    self.message_label.pack(pady=12)

    button_frame = tk.Frame(
        self.root,
        bg="#0f172a"
    )
    button_frame.pack(pady=(0, 20))

    restart_button = tk.Button(
        button_frame,
        text="New Round",
        command=self.new_round,
        font=("Arial", 11, "bold"),
        bg="#2563eb",
        fg="white",
        activebackground="#1d4ed8",
        activeforeground="white",
        relief="flat",
        padx=15,
        pady=8,
        cursor="hand2"
    )
```

```
        restart_button.pack(side="left", padx=5)

        reset_button = tk.Button(
            button_frame,
            text="Reset Scores",
            command=self.reset_scores,
            font=("Arial", 11, "bold"),
            bg="#475569",
            fg="white",
            activebackground="#64748b",
            activeforeground="white",
            relief="flat",
            padx=15,
            pady=8,
            cursor="hand2"
        )
        reset_button.pack(side="left", padx=5)

    def make_move(self, index):
        result = self.game.make_move(index)

        if result:
            self.update_ui()

    def update_ui(self):
        for index, button in enumerate(self.buttons):
            value = self.game.board[index]

            button.config(
                text=value,
                fg=self.get_symbol_color(value)
            )

            if value:
```

```
                button.config(
                    state="disabled"
                )
            else:
                button.config(
                    state="normal"
                )

        self.x_score_label.config(
            text=f"X: {self.game.scores['X']}"
        )

        self.o_score_label.config(
            text=f"O: {self.game.scores['O']}"
        )

        if self.game.winner:
            self.status_label.config(
                text=f"Player {self.game.winner} Wins!"
            )

            self.message_label.config(
                text="Start a new round to play again"
            )

            self.highlight_winner()

        elif self.game.is_draw:
            self.status_label.config(
                text="It's a Draw!"
            )

            self.message_label.config(
                text="Start a new round to play again"
```

```
            )

        else:
            self.status_label.config(
                text=f"Player {self.game.current_player}'s Turn"
            )

    def get_symbol_color(self, symbol):
        if symbol == "X":
            return "#38bdf8"

        if symbol == "O":
            return "#f472b6"

        return "white"

    def highlight_winner(self):
        if not self.game.winning_combo:
            return

        for index in self.game.winning_combo:
            self.buttons[index].config(
                bg="#14532d"
            )

    def new_round(self):
        self.game.reset_board()

        for button in self.buttons:
            button.config(
                bg="#1e293b"
            )

        self.update_ui()
```

```
    def reset_scores(self):
        self.game.reset_scores()
        self.game.reset_board()

        for button in self.buttons:
            button.config(
                bg="#1e293b"
            )

        self.update_ui()
```

---

7. Backend / Game Logic Code

game.py

```
class TicTacToe:
    WINNING_COMBINATIONS = [
        (0, 1, 2),
        (3, 4, 5),
        (6, 7, 8),
        (0, 3, 6),
        (1, 4, 7),
        (2, 5, 8),
        (0, 4, 8),
        (2, 4, 6)
    ]

    def __init__(self):
        self.scores = {
            "X": 0,
            "O": 0
        }
```

```
        self.reset_board()

    def reset_board(self):
        self.board = [""] * 9
        self.current_player = "X"
        self.winner = None
        self.is_draw = False
        self.winning_combo = None

    def make_move(self, index):
        if self.winner or self.is_draw:
            return False

        if self.board[index] != "":
            return False

        self.board[index] = self.current_player

        if self.check_winner():
            self.winner = self.current_player
            self.scores[self.current_player] += 1
            return True

        if self.check_draw():
            self.is_draw = True
            return True

        self.switch_player()

        return True

    def check_winner(self):
        for combination in self.WINNING_COMBINATIONS:
```

```
            a, b, c = combination

            if (
                self.board[a] != ""
                and self.board[a] == self.board[b]
                and self.board[b] == self.board[c]
            ):
                self.winning_combo = combination
                return True

        return False

    def check_draw(self):
        return all(cell != "" for cell in self.board)

    def switch_player(self):
        if self.current_player == "X":
            self.current_player = "O"
        else:
            self.current_player = "X"

    def reset_scores(self):
        self.scores = {
            "X": 0,
            "O": 0
        }
```

---

8. Main File

main.py

```
import tkinter as tk
from ui import TicTacToeUI
```

```
def main():
    root = tk.Tk()
    TicTacToeUI(root)
    root.mainloop()


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

---

9. How Frontend and Backend Connect

main.py

TicTacToeUI

TicTacToe

Board State

Winner / Draw / Turn

TicTacToeUI

Updated Interface

The **UI** handles button clicks and displays the board.

The **game logic** handles:

- Player turns

- Board state
- Winner detection
- Draw detection
- Score
- Game reset

This keeps the interface separate from the actual game rules.

---

10. How to Run

Open the project folder:

```
cd tic_tac_toe
```

Run:

```
python main.py
```

No external packages are required.

---

11. Basic Testing

Test the following:

- X can make the first move.
- O gets the next turn.
- Players cannot select an occupied square.
- Three Xs in a row produce an X win.

- Three Os in a row produce an O win.
- All eight winning combinations work.
- A completely filled board without a winner produces a draw.
- New Round clears the board but keeps scores.
- Reset Scores clears both scores.
- Winning cells are highlighted.

---

12. Small Improvements

Students can extend the project by adding:

- Player name input
- Single player mode
- Computer opponent
- Easy, Medium and Hard AI
- Minimax algorithm
- Game history
- Sound effects
- Animations
- Custom themes
- Online multiplayer

Project Goal

This project teaches how to turn a simple game into a structured application using:

UI Events Game State Logic Result

The same approach can later be used when building larger applications.

haas.dev

https://dev-roast-app.vercel.app

Visit haas.dev for more resources and guides.