**TIC TAC TOE GAME**

**Project Report submitted in fulfilment of the**

**requirement for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

**(ARTIFICIAL INTELLIGENCE AND DATA SCIENCE)**

***By***

**TeamName:QBusters**

**A Rohith [Reg No:122210701101]**

**N Rakshith Krishna [Reg No:122210701135]**

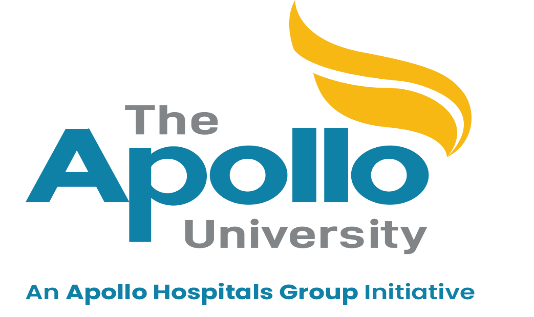
**L Rohith [Reg No:12221070126]**

**CH Ranga Kiran [Reg No:122210601111]**

**Under the Guidance of**

**Mr.Jayashankar**

**(Bytexl Trainer)**



**SCHOOL OF TECHNOLOGY**

**THE APOLLO UNIVERSITY**

**Murakambattu, Chittoor – 517127, Andhra Pradesh**

**Table of Contents**

| **Section No.** | **Title** | **Description** |
| --- | --- | --- |
| **1** | **Introduction** | Overview of the project, its purpose, and significance. |
| **2** | **Objective** | The main goals and desired outcomes of the project. |
| **3** | **Technology Overview** | Overview of the technologies used. |
| **3.1** | **- Python** | Explanation of why Python is chosen for the project. |
| **3.2** | **- Tkinter Library** | Overview of the Tkinter library and its role in building the GUI. |
| **4** | **System Design** | Design structure of the application/game. |
| **4.1** | **Game Flow Diagram** | Diagram outlining the game/application flow and interaction. |
| **4.2** | **- Class and Function Structure** | Overview of the key classes and functions in the project. |
| **5** | **Detailed Code Explanation** | In-depth breakdown and explanation of critical parts of the code. |
| **5.1** | **- Code Breakdown** | Detailed explanation of the core code sections and logic. |
| **6** | **Graphical User Interface (GUI)** | Description of the layout and structure of the user interface. |
| **7** | **Features** | Key features of the project and any unique functionalities. |
| **8** | **Challenges and Solutions** | Challenges faced during development and how they were resolved. |
| **9** | **Testing** | Explanation of testing methodologies used. |
| **9.1** | **- Unit Testing** | Description of the unit tests to ensure functionality. |
| **9.2** | **- User Testing** | Overview of user testing to validate usability. |
| **10** | **Conclusion** | Final thoughts and summary of the project. |
| **11** | **References** | A list of all references and resources used in the project. |

**Table of Figures**

|  |  |  |
| --- | --- | --- |
| **SI.NO** | **Figure Name** | **PAGE.NO** |
| **1** | **Fig 1.0** | **12** |
| **2** | **Fig 2.0** | **13** |
| **3** | **Fig 3.0** | **14** |
| **4** | **Fig 4.0** | **15** |

**1.Introduction**

This program is a graphical implementation of the classic **Tic-Tac-Toe** game, created using Python's **Tkinter** library for the graphical user interface (GUI). The game allows two players to compete against each other by taking turns to place their respective symbols (either "X" or "O") on a 3x3 grid, aiming to get three of their symbols in a row—horizontally, vertically, or diagonally.

The primary features of this game include:

**Player Choice:** Players can select their symbols at the start of the game, with "X" typically going first.

**Interactive Grid:** The game board is presented as a clickable 3x3 grid, with each cell responding to user clicks.

**Real-time Gameplay:** Players take turns placing their symbols on the grid, and the game updates the board after each move.

**Win Detection:** The program automatically checks for a winner after every move, identifying horizontal, vertical, or diagonal winning patterns.

**Game Restart:** Once a player wins or the game results in a draw, players are given the option to restart the game.

This implementation makes use of Tkinter widgets such as buttons and labels to create an easy-to-navigate interface. The program aims to offer an enjoyable gaming experience for users of all ages, while also demonstrating fundamental Python and Tkinter techniques like event handling, window layout management, and game logic implementation.

**2. Objective**

The primary objective of this program is to develop a user-friendly, interactive **Tic-Tac-Toe** game using **Python** and the **Tkinter** library. The goal is to create an engaging and intuitive gaming experience for two players. Specifically, the program aims to achieve the following key features:

* **Symbol Selection:** The game should allow two players to choose their respective symbols ("X" or "O") at the beginning of the game, ensuring that each player can easily customize their experience.
* **Dynamic 3x3 Game Board:** A visually appealing and functional 3x3 grid should be displayed to represent the game board. Each cell in the grid must be clickable, allowing players to place their symbol in the respective cell.
* **Turn Alternation:** The game should properly alternate turns between the two players. Player 1 typically uses "X" and starts the game, while Player 2 uses "O" and follows. The game must ensure that players cannot make moves out of turn.
* **Winner Detection:** The game must continuously monitor the state of the board and automatically detect when a player has achieved a winning condition. This includes horizontal, vertical, and diagonal combinations of three consecutive symbols.
* **Highlighting the Winner:** Once a winner is detected, the program should visually highlight the winning combination (such as changing the background color of the winning cells), making it clear which player has won.
* **Game Restart Option:** After a win, loss, or draw, the game should provide the players with an option to restart the game. This should reset the board to its initial state, allowing players to start a fresh game without closing the application.
* **User-Friendly Interface:** The game must have a simple and intuitive graphical interface with clear instructions, making it accessible for players of all ages. It should be easy to navigate and visually appealing, with the use of colors, labels, and buttons for a smooth gaming experience.

Through this implementation, the game aims to provide entertainment and challenge while demonstrating basic concepts in Python programming, event handling, and GUI design using Tkinter.

**3. Technology Overview**

This Tic-Tac-Toe game is developed using **Python** and **Tkinter**, two powerful and widely-used technologies for creating desktop applications. The combination of these tools provides a seamless environment for building interactive user interfaces and implementing game logic. Here's a deeper look at the technologies used in this project:

**Python**

* **Overview:** Python is an open-source, high-level programming language that emphasizes simplicity and readability, making it an excellent choice for both beginners and experienced developers. Its syntax is intuitive and easy to understand, enabling developers to focus on solving problems rather than managing complex code structures.
* **Why Python:** Python's versatility allows it to be used for a wide range of applications, from web development to data analysis. In this case, Python's simplicity and ability to handle basic game logic and GUI elements make it a perfect fit for this project. Python also has extensive community support, providing a wealth of tutorials, libraries, and tools that simplify the development process.
* **Key Features Used in this Project:**
  + **Functions and Loops:** To handle the game's core logic, such as alternating turns, checking for a winner, and restarting the game.
  + **Conditionals:** To detect win conditions and decide if a player has won, lost, or if the game has ended in a draw.
  + **Event Handling:** Python's built-in event handling capabilities allow the program to react to user actions like mouse clicks, enabling an interactive gaming experience.

**Tkinter Library**

* **Overview:** Tkinter is the standard GUI (Graphical User Interface) library that comes bundled with Python. It provides tools for creating windows, buttons, labels, frames, and other graphical components essential for building interactive desktop applications. Tkinter is based on the Tk GUI toolkit and is designed to be simple and lightweight, making it ideal for small to medium-scale applications.
* **Why Tkinter:** Tkinter is the most common library for creating GUI-based Python applications. It's simple to use and doesn't require additional installation, as it is part of Python's standard library. Tkinter's ability to create interactive elements, like buttons and labels, and its straightforward layout management, make it an excellent choice for this Tic-Tac-Toe game.
* **Key Features Used in this Project:**
  + **Widgets:** Tkinter provides various widgets such as buttons, labels, and frames that are used to create the game interface. Each cell in the Tic-Tac-Toe grid is a button widget, which is clickable by the players.
  + **Window Management:** Tkinter enables the creation of a main game window and handles the layout of the interface, ensuring the grid is properly aligned and all elements are visible.
  + **Event Binding:** Tkinter allows binding events (like mouse clicks) to functions, making it easy to update the game state and respond to user input in real time.

**Why These Technologies?**

* **Ease of Learning and Use:** Both Python and Tkinter are easy to learn, making them perfect for rapid development of small applications like this one.
* **Efficient Development:** Python’s concise syntax and Tkinter’s ready-made widgets significantly speed up development, allowing the focus to remain on game logic and player interaction.
* **Cross-Platform Compatibility:** Python and Tkinter are cross-platform, meaning the game can run on different operating systems (Windows, macOS, Linux) without modification.

Together, these technologies offer a powerful yet simple environment for creating a functional, enjoyable Tic-Tac-Toe game with a GUI

**4. System Design**

**Game Flow Diagram:**

The game flow describes the sequence of actions that take place from the moment the program starts to the end of the game. It ensures that both players can interact with the game, and the program correctly identifies the winner or a draw. Below is the breakdown of the game flow:

1. **Player Symbol Selection:**
   * The game starts with a "Choose Symbols" tab where players select their symbols .

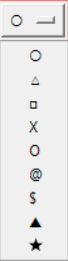
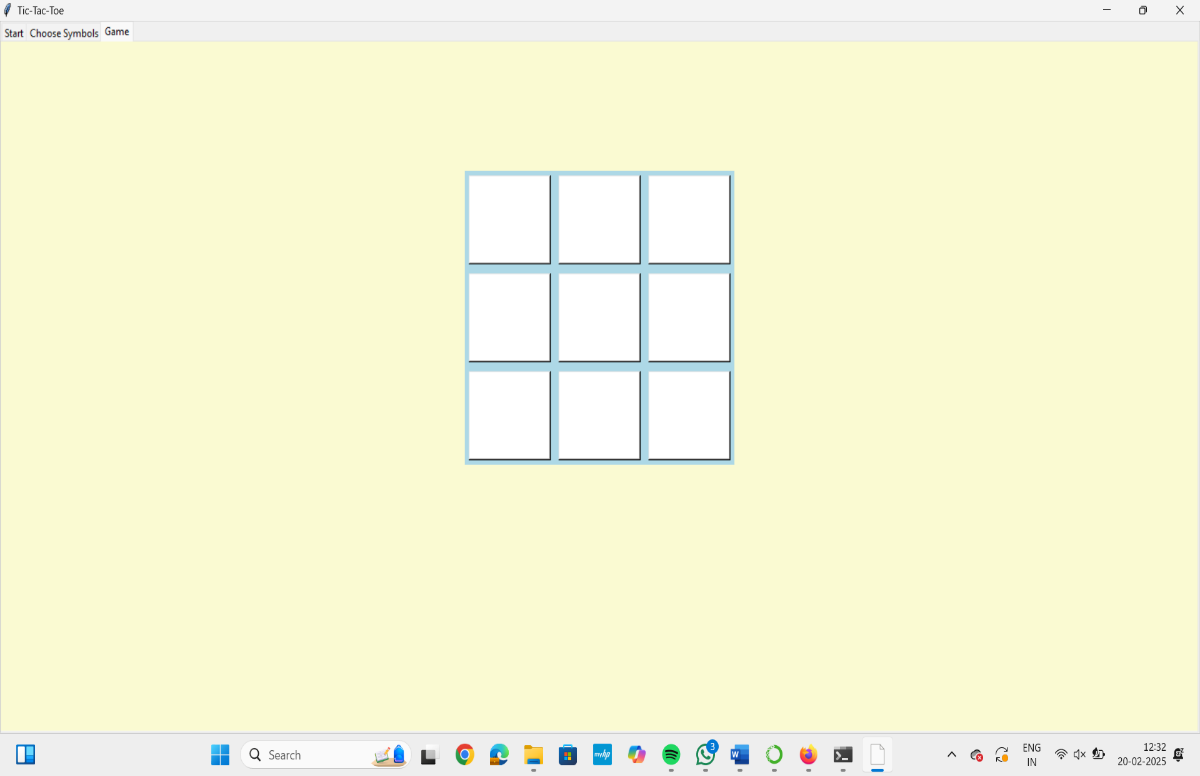


Fig 1.0

Once the symbols are chosen, the game moves to the main board interface.

1. **Game Board Creation:**
   * A 3x3 grid is created, represented by buttons (or clickable cells).
   * Each button is initially empty and reacts to player clicks by updating with the chosen symbol.
   * 

**Fig 2.0**

**Player Turn Logic:**

* + Players alternate turns. Player 1 (usually "X") places their symbol in an empty cell.
  + The program checks the board after each move to see if there is a winner or if the game is a draw.

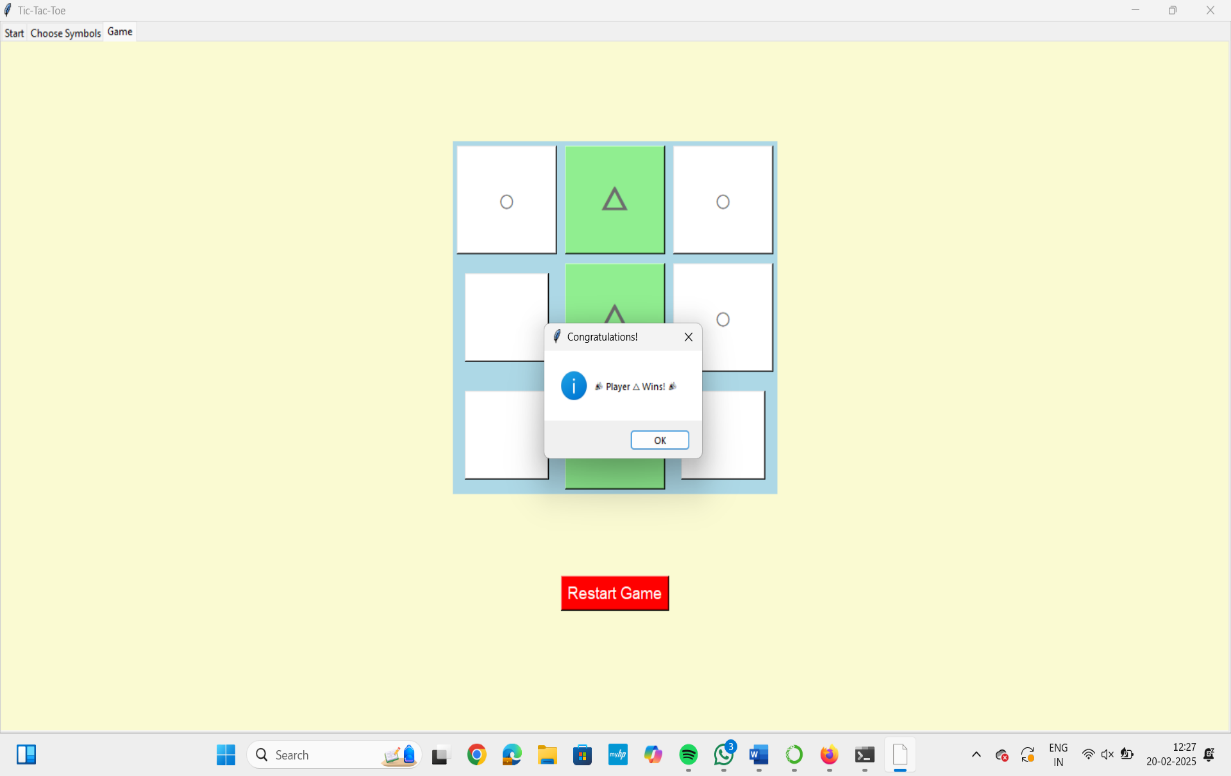
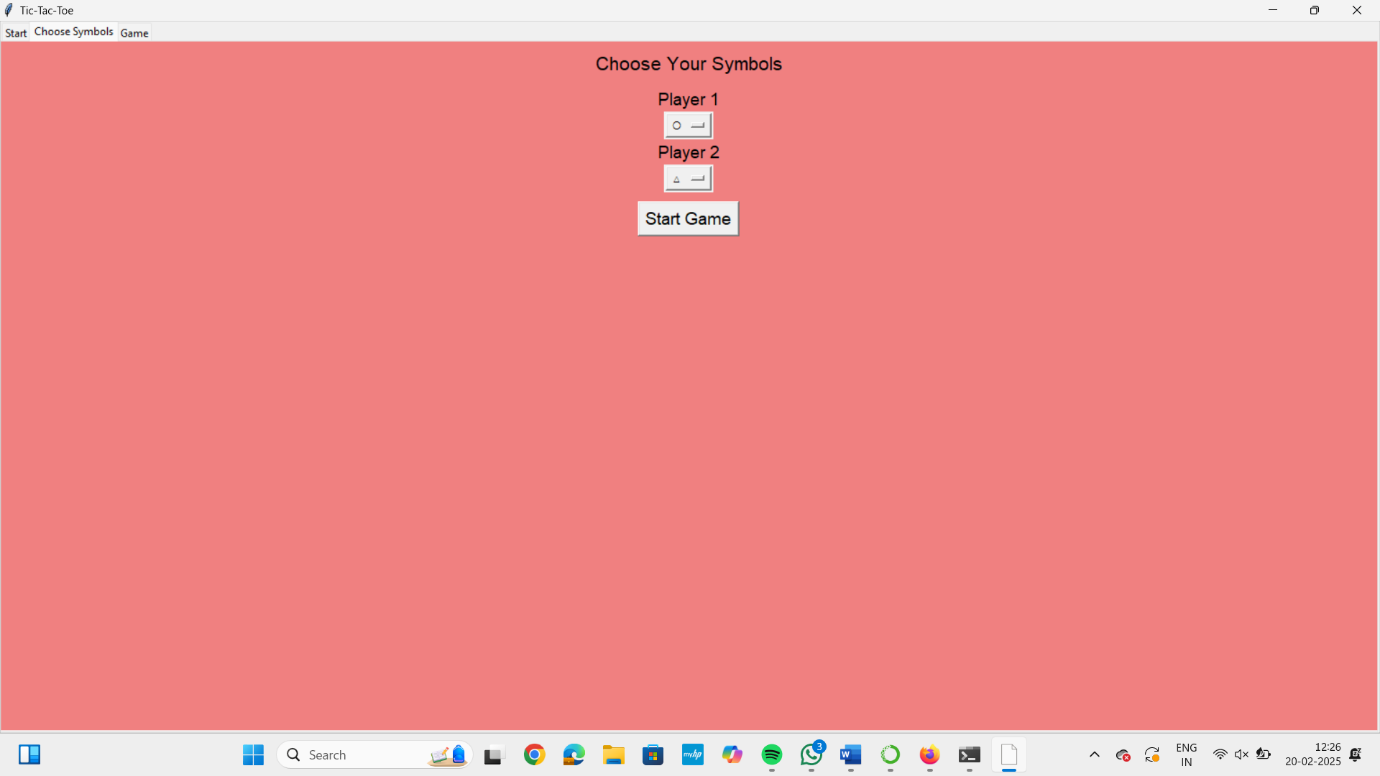
1. **Winner or Draw Detection:**
   * After each move, the program checks for a winning combination or a draw.
   * If a player wins, their winning cells are highlighted, and the game ends.
   * If there is a draw (i.e., all cells are filled without a winner), the game ends with a draw notification.
2. **Restart or Quit:**
   * Once a winner is detected or a draw is declared, the game prompts the players with options to either restart the game or quit.
   * 

Fig 3.0

**Class and Function Structure:**

The **System Design** is organized around key functions, which handle the game flow and interaction between players. Below is an overview of the main classes and functions used to structure the game:

**1. start\_game():**

* **Purpose:** Initializes the game, resets the board, and switches the user interface to the game board.
* 

**Fig 4.0**

**Logic:**

* + Resets any game variables or states (board, player symbols, etc.).
  + Sets up the game environment, including displaying the 3x3 grid and alternating player turns.

**2. restart\_game():**

* **Purpose:** Resets the game state to start a new round.
* **Logic:**
  + Clears all the symbols from the board.
  + Resets the game variables (turns, winner status).
  + Displays the "Choose Symbols" tab again, allowing players to reselect their symbols if desired.

**3. check\_winner():**

* **Purpose:** Checks the current board for any winning combination (horizontal, vertical, diagonal).
* **Logic:**
  + Analyzes each row, column, and diagonal to see if any player has achieved three symbols in a row.
  + Returns the winner (if any), or None if there is no winner.

**4. highlight\_winner():**

* **Purpose:** Highlights the winning cells on the board for better visual recognition.
* **Logic:**
  + Once a winner is found, this function changes the background color of the winning cells to visually emphasize the victory.

**5. is\_draw():**

* **Purpose:** Checks if the game has ended in a draw.
* **Logic:**
  + Verifies if all cells are filled and no winner has been found.
  + If true, returns a draw, and the game ends.

**6. on\_click():**

* **Purpose:** Handles the logic when a player clicks a cell on the board.
* **Logic:**
  + This function is triggered by the player's click event on a grid cell.
  + It places the appropriate symbol (either "X" or "O") in the clicked cell if it’s empty.
  + It then checks for a winner or a draw after the move is made and updates the UI accordingly.

**7. show\_restart\_button():**

* **Purpose:** Displays the restart button after the game ends.
* **Logic:**
  + Once a win or draw is detected, this function is responsible for making the "Restart Game" button visible.
  + The button lets the players restart the game, returning them to the initial state.

**8. create\_board():**

* **Purpose:** Creates or updates the game board in the GUI.
* **Logic:**
  + This function sets up the 3x3 grid layout, which consists of buttons (one for each cell).
  + It also updates the board by displaying the player's symbols in the cells as the game progresses.

**9. create\_dashboard():**

* **Purpose:** Creates the symbol selection interface for the players.
* **Logic:**
  + This function creates the initial interface where players choose their symbols ("X" or "O").
  + Once the symbols are selected, it switches the UI to the main game board.

**How the Game Works Together:**

* **Start Phase:** Upon launching the game, create\_dashboard() is executed to show the symbol selection interface. Once players select their symbols, start\_game() is called to transition to the game board.
* **Gameplay:** During the game, players interact with the board by clicking cells, triggering the on\_click() function. After each move, check\_winner() and is\_draw() are called to determine the game's outcome.
* **End of Game:** If a winner is found, highlight\_winner() is invoked to emphasize the winning cells, and show\_restart\_button() prompts players to either restart or quit the game.

This **System Design** ensures the game operates logically and efficiently, with clear separation of concerns in different functions. The design follows modular principles, making it easy to maintain and extend in the future.

**5. Detailed Code Explanation**

This section breaks down the core functions used in the Tic-Tac-Toe game to explain how the game logic is implemented.

**Functions Breakdown:**

**1. start\_game():**

* **Purpose:** Initializes the game by selecting the player symbols and resetting the game board.
* **Code Breakdown:**
  + Sets the player's symbols chosen in the "Choose Symbols" tab.
  + Resets the game state: clears the board, resets the turn counter, and initializes the game variables.
  + Switches the user interface to **Game Tab (tab3)**, where the players interact with the board.

**2. restart\_game():**

* **Purpose:** Resets the board and prepares the game for a new round.
* **Code Breakdown:**
  + Resets the 3x3 grid by clearing the text in the buttons (removing "X" or "O").
  + Resets other relevant variables, such as the turn counter.
  + Returns to the **Choose Symbols Tab (tab2)** if players want to reselect their symbols.

**3. check\_winner():**

* **Purpose:** Checks if any player has won by meeting one of the winning conditions (3 in a row).
* **Code Breakdown:**
  + Checks each row, column, and diagonal of the board for three consecutive symbols (either "X" or "O").
  + If a winner is found, it highlights the winning cells and displays a congratulatory message to the winning player.
  + If no winner is found, it returns None.

**4. highlight\_winner():**

* **Purpose:** Highlights the winning cells to visually emphasize the winning combination.
* **Code Breakdown:**
  + Once a winner is detected, this function iterates over the cells that form the winning combination (either a row, column, or diagonal).
  + The background color of these cells is changed to indicate the victory.

**5. is\_draw():**

* **Purpose:** Checks if the game has ended in a draw (when all cells are filled but no winner has been found).
* **Code Breakdown:**
  + Checks if all cells on the board are filled and that no winner exists.
  + If true, displays a message indicating a draw and prevents further moves.

**6. on\_click(row, col):**

* **Purpose:** Handles the logic when a player clicks on a cell in the 3x3 grid.
* **Code Breakdown:**
  + This function is triggered whenever a player clicks a button on the grid (represented as a cell in the game board).
  + It places the appropriate symbol ("X" or "O") in the clicked cell and disables the button to prevent further changes.
  + The function then checks if a winner exists or if the game has ended in a draw by calling check\_winner() and is\_draw().

**7. show\_restart\_button():**

* **Purpose:** Displays the "Restart Game" button after the game has ended (either due to a win or a draw).
* **Code Breakdown:**
  + Once the game concludes, this function makes the restart button visible, allowing players to start a new game.
  + The restart button is positioned below the game board and is clickable to reset the game.

**8. create\_board():**

* **Purpose:** Creates the 3x3 grid of buttons for the Tic-Tac-Toe board.
* **Code Breakdown:**
  + This function generates the board layout by placing a button in each cell of the 3x3 grid.
  + Each button corresponds to a cell and is linked to the on\_click() function, allowing players to make their move when they click a cell.
  + The buttons are placed in a grid layout using Tkinter’s geometry management.

**9. create\_dashboard():**

* **Purpose:** Creates the interface for selecting the player symbols.
* **Code Breakdown:**
  + This function generates the user interface where players choose their symbols (e.g., "X" or "O").
  + Once players select their symbols, the game proceeds by calling start\_game() to begin the game in the **Game Tab (tab3)**.

**6. Graphical User Interface (GUI)**

The GUI is structured using **Tkinter** and organized into three main tabs to guide the user through the game process:

**1. Start Tab (tab1):**

* **Purpose:** This tab serves as the entry point to the game.
* **UI Elements:**
  + A **welcome message** to greet players.
  + A **button** labeled "Start Game" that takes the player to the **Choose Symbols Tab (tab2)**.
* **Functionality:**
  + Once players click the "Start Game" button, they are directed to the next tab, where they can select their symbols.

**2. Choose Symbols Tab (tab2):**

* **Purpose:** This tab allows players to choose their respective symbols ("X" or "O") for the game.
* **UI Elements:**
  + Two **radio buttons** or **dropdown lists** that allow the players to select their symbols.
  + A **"Start Game" button** that begins the game once the players have selected their symbols.
* **Functionality:**
  + Players select their symbols, and once both players have made a choice, they can click the "Start Game" button to initialize the board and move to the **Game Tab (tab3)**.

**3. Game Tab (tab3):**

* **Purpose:** This is where the actual Tic-Tac-Toe game is played.
* **UI Elements:**
  + A **3x3 grid of buttons** that represent the game board, where each button corresponds to a cell on the grid.
  + A **turn indicator** (optional) showing whose turn it is (Player 1 or Player 2).
  + A **"Restart Game" button** that appears after the game ends (either due to a win or a draw), allowing players to start a new round.
* **Functionality:**
  + Players alternate turns by clicking on the buttons in the grid.
  + The game board updates after each move, checking for a winner or a draw.
  + When the game ends, the **Restart Game** button appears, prompting the players to start a new game.

**7. Features**

This Tic-Tac-Toe game offers a variety of features to enhance the user experience. Below are the key features implemented in the game:

**1. Symbol Selection:**

* **Feature Overview:** Players can choose their own symbols at the beginning of the game, allowing for a customized experience.
* **Details:**
  + Players can choose from a predefined list of symbols such as "X", "O", and other creative options like "△" or "●".
  + Each player selects one symbol, and the other player is automatically assigned the remaining symbol.
  + The symbol selection is handled through an easy-to-use **dropdown list** or **radio buttons** in the **Choose Symbols Tab (tab2)**.
  + This ensures that players can personalize their experience and feel more engaged in the game.

**2. Game Board:**

* **Feature Overview:** The game features a dynamically generated 3x3 board that updates as players make their moves.
* **Details:**
  + The board is created using **buttons**, each corresponding to a cell on the grid.
  + Each time a player clicks a button, their selected symbol (either "X", "O", etc.) is placed in the corresponding cell.
  + The board is designed to update immediately after every move, giving real-time feedback on the current state of the game.
  + This board is interactive, with buttons that can be clicked only if they are empty, preventing players from overwriting each other’s symbols.

**3. Winner Highlighting:**

* **Feature Overview:** When a player wins, the game visually highlights the winning combination for easy identification.
* **Details:**
  + After a player wins (by placing three of their symbols consecutively in a row, column, or diagonal), the winning cells are highlighted in **light green** (or any color of your choice).
  + This visual cue helps players quickly recognize the winner and adds to the excitement of the game.
  + The highlighted cells make it clear which symbols formed the winning combination, ensuring a satisfying and rewarding conclusion to the game.

**4. Draw Detection:**

* **Feature Overview:** If no one wins and the board is full, the game detects a draw and notifies the players.
* **Details:**
  + The game automatically checks whether all the cells are filled after each turn.
  + If the board is full and there is no winner, the game recognizes it as a draw and displays an appropriate message.
  + The message clearly informs the players that the game ended in a draw, ensuring no one is left uncertain about the game’s outcome.

**5. Restart Option:**

* **Feature Overview:** After the game ends (whether due to a win or draw), players can restart the game with a new board and start fresh.
* **Details:**
  + Once the game concludes, either through a win or draw, a **"Restart Game"** button appears below the board.
  + When the players click this button, the game resets to its initial state, clearing the board, resetting the turn counter, and returning to the **Choose Symbols Tab (tab2)**.
  + This provides a seamless way for players to enjoy multiple rounds of Tic-Tac-Toe without closing the game.

**8. Challenges and Solutions**

Developing a Tic-Tac-Toe game comes with several challenges, but by implementing effective solutions, these challenges can be addressed smoothly:

**Challenge 1: Handling the Game State and Updating the Board Based on User Interaction**

* **Problem:** The game state (i.e., the current state of the board, whose turn it is, etc.) needs to be managed and updated every time the user interacts with the game. Since the game board is dynamic and players take turns, it requires careful state management to ensure the game progresses correctly.
* **Solution:**
  + **Global board variable:** A global variable is used to represent the current state of the board. This board is typically a 2D array (3x3 matrix) where each element holds the symbol for a cell or is empty if no symbol is placed yet.
  + **Buttons array:** Each button on the Tkinter board is linked to a cell in the 2D array. When a player clicks a button, the corresponding cell in the array is updated, and the button's label is changed to reflect the player’s symbol.
  + This structure makes it easy to track the state of the game and allows for real-time updates on the board whenever a player makes a move.

**Challenge 2: Detecting and Highlighting the Winner or Draw Condition**

* **Problem:** Detecting a winner (a player who achieves three symbols in a row, column, or diagonal) or detecting a draw (when all cells are filled without a winner) requires careful checking of the board after every move.
* **Solution:**
  + **check\_winner() function:** This function checks all possible win conditions by analyzing the rows, columns, and diagonals. If three consecutive symbols of the same player are found, the function highlights those cells and declares the winner.
  + **is\_draw() function:** This function checks if the board is full and no winner has been found. If this is the case, the game is declared a draw.
  + Both functions are called after every move to ensure that the game state is checked for these conditions promptly.

**9. Testing**

Testing is an important part of ensuring the functionality and usability of the Tic-Tac-Toe game. Two types of testing are crucial for this project:

**Unit Testing:**

* **Purpose:** Unit tests help verify that individual functions and game logic work correctly.
* **What to Test:**
  + **check\_winner():** Write tests to check that the winner is detected accurately for all win conditions (rows, columns, diagonals).
  + **is\_draw():** Write tests to ensure that the game correctly identifies a draw condition when all cells are filled but no player has won.
* **Example Test Case:**
  + Test the check\_winner() function by manually setting a board state and ensuring the function correctly identifies the winner

**User Testing:**

* **Purpose:** User testing involves real players interacting with the game to ensure that the user interface (GUI) is intuitive and that the game behaves as expected.
* **What to Test:**
  + Ensure that all buttons are responsive and that users can interact with the game easily.
  + Test that the game rules (turn-taking, symbol placement, winner detection, etc.) are followed correctly by the game.
  + Observe how users react to various parts of the game (e.g., symbol selection, game progression, and the appearance of the restart button).
* **Feedback:** Based on the feedback from user testing, further tweaks can be made to improve the experience (e.g., improving visual feedback, adding animations, etc.).

**CODE:**

import tkinter as tk

from tkinter import messagebox

from tkinter import ttk

def start\_game():

global players, turn, board, buttons

players = [player1\_choice.get(), player2\_choice.get()]

turn = 0

board = [[' ' for \_ in range(3)] for \_ in range(3)]

create\_board()

notebook.select(tab3)

def restart\_game():

global turn, board

turn = 0

board = [[' ' for \_ in range(3)] for \_ in range(3)]

create\_board()

def check\_winner():

for row in range(3):

if board[row][0] == board[row][1] == board[row][2] and board[row][0] != ' ':

highlight\_winner([(row, 0), (row, 1), (row, 2)])

return board[row][0]

for col in range(3):

if board[0][col] == board[1][col] == board[2][col] and board[0][col] != ' ':

highlight\_winner([(0, col), (1, col), (2, col)])

return board[0][col]

if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':

highlight\_winner([(0, 0), (1, 1), (2, 2)])

return board[0][0]

if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':

highlight\_winner([(0, 2), (1, 1), (2, 0)])

return board[0][2]

return None

def highlight\_winner(cells):

for r, c in cells:

buttons[r][c].config(bg='lightgreen')

root.after(1000, lambda: messagebox.showinfo("Congratulations!", f"🎉 Player {board[cells[0][0]][cells[0][1]]} Wins! 🎉"))

def is\_draw():

if all(cell != ' ' for row in board for cell in row):

root.after(1000, lambda: messagebox.showinfo("Game Over", "It's a draw!"))

return True

return False

def on\_click(row, col):

global turn

if board[row][col] == ' ':

board[row][col] = players[turn % 2]

buttons[row][col].config(text=players[turn % 2], state=tk.DISABLED, font=('Arial', 28, 'bold'), fg='blue' if players[turn % 2] == players[0] else 'red')

winner = check\_winner()

if winner or is\_draw():

show\_restart\_button()

else:

turn += 1

def show\_restart\_button():

restart\_btn = tk.Button(tab3, text="Restart Game", font=('Arial', 14), command=restart\_game, bg='red', fg='white')

restart\_btn.place(relx=0.5, rely=0.8, anchor=tk.CENTER)

def create\_board():

global buttons

for widget in tab3.winfo\_children():

widget.destroy()

buttons = [[None for \_ in range(3)] for \_ in range(3)]

frame = tk.Frame(tab3, bg='lightblue')

frame.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

for i in range(3):

for j in range(3):

buttons[i][j] = tk.Button(frame, text=board[i][j], font=('Arial', 24), height=2, width=5, bg='white', command=lambda r=i, c=j: on\_click(r, c))

buttons[i][j].grid(row=i, column=j, padx=5, pady=5)

def create\_dashboard():

global player1\_choice, player2\_choice

for widget in tab2.winfo\_children():

widget.destroy()

tk.Label(tab2, text="Choose Your Symbols", font=('Arial', 16), bg='lightcoral').pack(pady=10)

symbols = ['○', '△', '□', 'X', 'O', '@', '$', '▲', '★']

player1\_choice = tk.StringVar(value='○')

player2\_choice = tk.StringVar(value='△')

tk.Label(tab2, text="Player 1", font=('Arial', 14), bg='lightcoral').pack()

tk.OptionMenu(tab2, player1\_choice, \*symbols).pack()

tk.Label(tab2, text="Player 2", font=('Arial', 14), bg='lightcoral').pack()

tk.OptionMenu(tab2, player2\_choice, \*symbols).pack()

tk.Button(tab2, text="Start Game", font=('Arial', 14), command=start\_game).pack(pady=10)

def create\_tabs():

global tab1, tab2, tab3, notebook

notebook = ttk.Notebook(root)

notebook.pack(expand=True, fill='both')

tab1 = tk.Frame(notebook, bg='lightblue')

tab2 = tk.Frame(notebook, bg='lightcoral')

tab3 = tk.Frame(notebook, bg='lightgoldenrodyellow')

notebook.add(tab1, text='Start')

notebook.add(tab2, text='Choose Symbols')

notebook.add(tab3, text='Game')

tk.Label(tab1, text="Welcome to Tic-Tac-Toe!", font=('Arial', 18), bg='lightblue').pack(pady=20)

tk.Button(tab1, text="Next", font=('Arial', 14), command=lambda: notebook.select(tab2)).pack(pady=10)

create\_dashboard()

root = tk.Tk()

root.title("Tic-Tac-Toe")

root.geometry("400x500")

root.configure(bg='lightgoldenrodyellow')

create\_tabs()

root.mainloop()

**Test Case 1: Start Game with Player 1 and Player 2 Choices**

**Objective:** Verify that the game starts correctly with the selected symbols for Player 1 and Player 2.

**Steps:**

1. Launch the game.
2. Navigate to the "Choose Symbols" tab.
3. Select symbols for Player 1 and Player 2 from the dropdown menu.
4. Click the "Start Game" button.

**Expected Result:**

* The game board appears with the selected symbols for both players.
* Player 1 begins the game.

**Test Case 2: Play a Turn and Switch Between Players**

**Objective:** Ensure that the game alternates between Player 1 and Player 2 after each turn.

**Steps:**

1. Start the game.
2. Player 1 clicks on an empty cell to place their symbol.
3. Player 2 clicks another empty cell to place their symbol.
4. Continue alternating turns.

**Expected Result:**

* The game alternates between Player 1 and Player 2 after each turn.
* The correct symbol for each player appears in the clicked cells.

**Test Case 3: Winning Scenario (Horizontal Line)**

**Objective:** Test the winning condition when a player wins by forming a horizontal line.

**Steps:**

1. Start the game.
2. Player 1 places their symbols in the top-left, top-middle, and top-right cells.
3. Player 1 wins with a horizontal line.

**Expected Result:**

* Player 1 wins with a horizontal line.
* The winning cells (top-left, top-middle, and top-right) are highlighted in light green.
* A message box appears congratulating Player 1.

**Test Case 4: Winning Scenario (Vertical Line)**

**Objective:** Test the winning condition when a player wins by forming a vertical line.

**Steps:**

1. Start the game.
2. Player 1 places their symbols in the top-left, middle-left, and bottom-left cells.
3. Player 1 wins with a vertical line.

**Expected Result:**

* Player 1 wins with a vertical line.
* The winning cells (top-left, middle-left, and bottom-left) are highlighted in light green.
* A message box appears congratulating Player 1.

**Test Case 5: Winning Scenario (Diagonal Line)**

**Objective:** Test the winning condition when a player wins by forming a diagonal line.

**Steps:**

1. Start the game.
2. Player 1 places their symbols in the top-left, middle-center, and bottom-right cells.
3. Player 1 wins with a diagonal line.

**Expected Result:**

* Player 1 wins with a diagonal line.
* The winning cells (top-left, middle-center, and bottom-right) are highlighted in light green.
* A message box appears congratulating Player 1.

**Test Case 6: Draw Condition**

**Objective:** Verify that the game ends in a draw when all cells are filled with no winner.

**Steps:**

1. Start the game.
2. Players alternate turns and fill all cells.
3. Ensure that no player wins by the end of the game.

**Expected Result:**

* A message box appears indicating that the game ended in a draw.

**Test Case 7: Restart Game After a Winner**

**Objective:** Test if the game can be restarted after a winner is declared.

**Steps:**

1. Play until a winner is declared.
2. Once the winner is displayed, click the "Restart Game" button.

**Expected Result:**

* The game board resets.
* Players can reselect their symbols and start a new game.

**Test Case 8: Restart Game After a Draw**

**Objective:** Test if the game can be restarted after a draw is declared.

**Steps:**

1. Play until the game ends in a draw.
2. Once the draw message is displayed, click the "Restart Game" button.

**Expected Result:**

* The game board resets.
* Players can reselect their symbols and start a new game.

**Test Case 9: Handle Invalid Cell Clicks (Occupied Cells)**

**Objective:** Ensure that the game does not allow players to click on already occupied cells.

**Steps:**

1. Start the game.
2. Player 1 places a symbol in the top-left cell.
3. Player 2 places a symbol in the top-middle cell.
4. Player 1 attempts to click on the top-left cell again (already occupied).
5. Player 2 attempts to click on the top-middle cell again (already occupied).

**Expected Result:**

* The board does not change.
* The turn does not switch unless a valid move is made.

**Test Case 10: Verify Initial Setup**

**Objective:** Ensure the default symbols are correctly set when starting the game.

**Steps:**

1. Launch the game.
2. Navigate to the "Choose Symbols" tab.
3. Check the default symbols selected for Player 1 and Player 2.

**Expected Result:**

* Player 1 defaults to the symbol '○'.
* Player 2 defaults to the symbol '△'.

**Test Case 11: UI Layout Check (Buttons, Fonts, Colors)**

**Objective:** Ensure that the user interface elements are laid out properly and visually consistent.

**Steps:**

1. Start the game.
2. Navigate through the tabs and observe the UI layout.
3. Check if buttons, fonts, and colors are consistent and appropriately placed.

**Expected Result:**

* The UI elements (buttons, fonts, and colors) should be consistent with the game design and should not overlap.
* The elements should be aligned correctly, and the game board should be displayed without visual issues.

**10. Conclusion**

This Tic-Tac-Toe project demonstrates how Python, along with the Tkinter library, can be used to develop a fully functional and interactive game with a graphical user interface (GUI). Key features such as **symbol selection**, **winner detection**, **game restart**, and **draw detection** ensure a smooth and engaging gameplay experience.

By addressing challenges such as managing game state and detecting winning or draw conditions, this project showcases the power of Python for building simple yet enjoyable desktop applications. The project also provides an opportunity to learn and apply core programming concepts such as event handling, state management, and GUI development with Tkinter.

**11. References**

Here are some references and resources that were useful in the development of this project:

* **Tkinter Documentation:**
  + Official Python Tkinter documentation: <https://docs.python.org/3/library/tkinter.html>
  + Tkinter tutorials and guides for building GUI applications.
* **Python Documentation:**
  + Python standard library documentation: <https://docs.python.org/3/>
  + General Python programming tutorials for beginners to advanced users.