Report on

Tic-Tac-Toe Game

By

Team CyberLoop

Divyam Raj (202410116100066)

Gaurav Chauhan (202410116100073)

Gaurav Kumar (202410116100074)

Harsh Solanki (202410116100087)

Session: 2024-2025 (II Semester)

Under the supervision of

Dr. Apoorv Jain (Assistant Professor)

KIET Group of Institutions, Delhi-NCR, Ghaziabad

DEPARTMENT OF COMPUTER APPLICATIONS
KIET GROUP OF INSTITUTIONS, DELHI-NCR,
GHAZIABAD-201206

TABLE OF CONTENTS

	Page Number
1. Introduction	3
2. Methodology	4-5
3. Typed Code	6-9
4. Output	10
5. Conclusion	11

Tic-Tac-Toe Game

Introduction

Tic-Tac-Toe is a classic two-player game widely known for its simple rules and strategic depth. The game consists of a 3x3 grid where players take turns marking a cell with their symbol—'X' or 'O'. The objective is to place three of their marks in a horizontal, vertical, or diagonal row to win the game. If all the cells are filled without a winner, the game ends in a draw.

This project aims to develop a Python-based Tic-Tac-Toe game where a player competes against the computer. The computer's moves follow a strategic approach that includes blocking the player's winning moves, attempting to win whenever possible, and making random moves when necessary. The player's input is validated to prevent incorrect entries, ensuring a smooth and error-free experience.

Additionally, this project demonstrates the importance of modular programming by organizing the code into separate functions for handling user input, generating computer moves, checking the game state, and executing the game loop. This structured approach enhances code readability and allows for future improvements, such as score tracking, multiplayer mode, or a graphical user interface (GUI). By implementing this game, we create an interactive program while exploring fundamental programming concepts applicable to more complex applications.

Methodology

The Tic-Tac-Toe game is implemented using a modular approach, where the game logic is divided into multiple functions. This structured programming methodology ensures clarity, maintainability, and scalability, making it easier to modify and expand the game in the future. Such an approach allows for easy debugging, enhances readability, and facilitates potential upgrades such as an AI-based opponent, a graphical user interface (GUI), and multiplayer mode.

The implementation consists of four major steps:

1. **Initializing the Game Board**
2. **Handling User and Computer Moves**
3. **Checking for a Winner or Draw**
4. **Executing the Game Loop**

Each step plays a crucial role in ensuring the game functions smoothly and provides an interactive user experience.

1. Initializing the Game Board

The game board is represented as a 3x3 grid stored in a list.

- The function print_board(board) is responsible for displaying the board in a visually structured format.
- The board is initialized as an empty list of nine spaces ([" "] * 9).

This initialization ensures a clean and well-structured game setup, allowing smooth gameplay.

2. Handling User and Computer Moves

Ensuring correct and valid moves from both the player and computer is essential for an error-free experience.

- The function get_player_move(board) prompts the user for a move and validates it to ensure they select an empty cell within the grid.
- The function get_computer_move(board) generates a move for the computer using a strategic approach:
    - First, it checks for a winning move.

- Then, it blocks the opponent's winning move if necessary.
- If neither is applicable, it tries to take the center.
- If the center is occupied, it randomly selects an available move.

By enforcing move validation and strategic play, we eliminate potential game-breaking scenarios caused by incorrect inputs.

3. Checking for a Winner or Draw

Once a move is made, the program determines if there is a winner or if the game results in a draw.

- The function check_win(board, player) examines all possible winning conditions (rows, columns, diagonals) to determine if either player has won.
- The function check_draw(board) verifies if all cells are filled without a winner, declaring the game a draw.

This logic ensures quick and efficient decision-making, displaying the appropriate outcome of the game.

4. Executing the Game Loop

The main function, play_game(), orchestrates the different steps of the game by calling the necessary functions in sequence:

1. The game board is initialized as an empty list.
2. The game enters a loop where the player and computer take turns making moves.
3. After each move, check_win(board, current_player) and check_draw(board) are called to determine the game's outcome.
4. If a winner is found or the game is a draw, the result is displayed, and the game ends.

This function serves as the central controller of the program, ensuring a structured flow of execution.

Typed Code

```
import random
def print_board(board):
  """Prints the Tic-Tac-Toe board."""
  print("-------------")
  for i in range(3):
    print("|", board[i*3], "|", board[i*3+1], "|", board[i*3+2], "|")
    print("-------------")


def check_win(board, player):
  """Checks if the given player has won the game."""
  win_conditions = [
      [0, 1, 2], [3, 4, 5], [6, 7, 8],  # Rows
      [0, 3, 6], [1, 4, 7], [2, 5, 8],  # Columns
      [0, 4, 8], [2, 4, 6]              # Diagonals
  ]
  for condition in win_conditions:
    if all(board[i] == player for i in condition):
      return True
  return False


def check_draw(board):
  """Checks if the game is a draw."""
```

```
  return all(cell != " " for cell in board)


def get_player_move(board):
    """Gets the player's move."""
    while True:
        try:
            move = int(input("Enter your move (1-9): ")) - 1
            if 0 <= move <= 8 and board[move] == " ":
                return move
            else:
                print("Invalid move. Try again.")
        except ValueError:
            print("Invalid input. Please enter a number between 1 and 9.")



def get_computer_move(board):
    """Gets the computer's move using a simple strategy."""
    # Check for winning move
    for i in range(9):
        if board[i] == " ":
            board[i] = "O"
            if check_win(board, "O"):
                return i
            board[i] = " "

    # Check for blocking move
    for i in range(9):
```

```
        if board[i] == " ":
            board[i] = "X"
            if check_win(board, "X"):
                return i
            board[i] = " "

    # Try to take the center
    if board[4] == " ":
        return 4

    # Choose a random available spot
    available_moves = [i for i, cell in enumerate(board) if cell == " "]
    return random.choice(available_moves) if available_moves else None



def play_game():
  """Plays a game of Tic-Tac-Toe."""
  board = [" "] * 9
  current_player = "X"

  while True:
    print_board(board)
    if current_player == "X":
        move = get_player_move(board)
    else:
        move = get_computer_move(board)
        print(f"Computer chose: {move+1}")
```

```
    board[move] = current_player

    if check_win(board, current_player):
      print_board(board)
      print(current_player, "wins!")
      break
    elif check_draw(board):
      print_board(board)
      print("It's a draw!")
      break

    current_player = "O" if current_player == "X" else "X"

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

Output


play_game()
Enter your move (1-9): 1
Computer chose: 5
Enter your move (1-9): 7
Computer chose: 4
Enter your move (1-9): 6
Computer chose: 2


Computer chose: 2
Enter your move (1-9): 8
Computer chose: 9
Enter your move (1-9): 3
It's a draw!

Conclusion

The development of the Tic-Tac-Toe game in Python highlights key programming concepts such as conditional statements, loops, user input validation, and strategic decision-making. By employing a modular approach, the code is structured and easy to maintain, making it adaptable for future enhancements.

This project also demonstrates how a simple game can be implemented using logical reasoning and AI-based strategies. The computer opponent utilizes a combination of defensive and offensive tactics, making gameplay more engaging. Additionally, implementing input validation ensures a smooth user experience by preventing invalid moves.

Overall, this project serves as a foundational step toward more complex game development, providing a solid understanding of algorithm design and structured programming. Future improvements, such as an advanced AI opponent, a graphical user interface, and multiplayer functionality, can further enhance the user experience.