PROJECT REPORT

TIC TAC TOE GAME

(CSE1021)

A11+A12+A13

By

DHEER JAIN

(25BCE10072)

ANTIMA JAIN

FACULTY

VIT BHOPAL UNIVERSITY

KOTRIKALAN, 466114

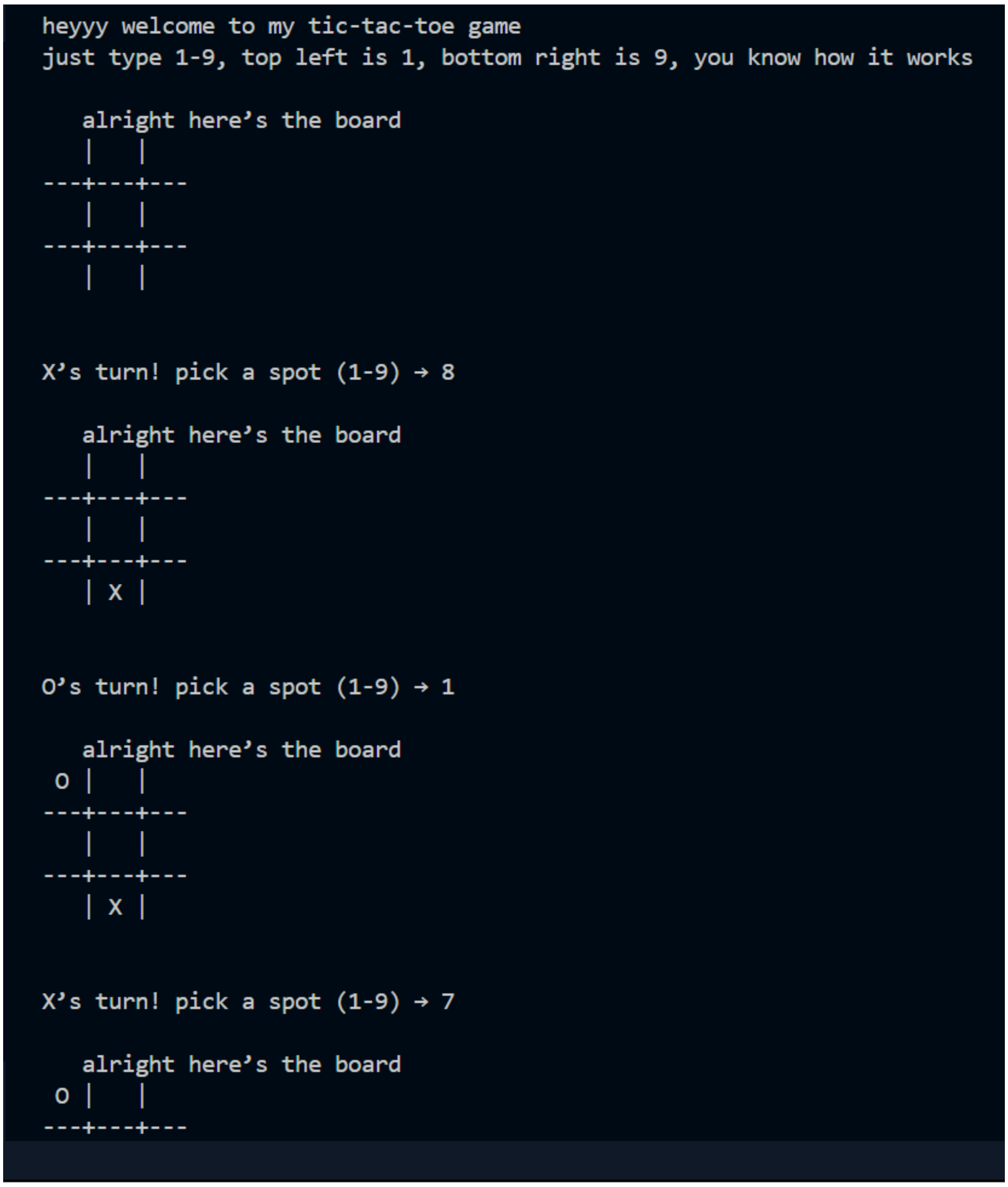
MADHYA PRADESH,

INDIA

DECEMBER , 2025

INTRODUCTION OF THE PROJECT

In this project , I have created a simple and joyful tic tac toe game using python loops , if else , if elif statements and functions are used in python programming. In this game a table/rows have been created of 3x3 ,using the function method and index ranging from 1-9. Here X and O characters are used and the person who makes the single line /row/diagonal complete of their variable wins the game if none of the players is able to ensure that the match between them is considered as a tie. This game evolves the logic pattern building of the players not very much but to an extent it develops the logical building . Patterns are carefully needed to be observed by the players.



```
heyyy welcome to my tic-tac-toe game
just type 1-9, top left is 1, bottom right is 9, you know how it works

   alright here's the board
   |   |
---+---+---
   |   |
---+---+---
   |   |


X's turn! pick a spot (1-9) → 8

   alright here's the board
   |   |
---+---+---
   |   |
---+---+---
   | X |


O's turn! pick a spot (1-9) → 1

   alright here's the board
 O |   |
---+---+---
   |   |
---+---+---
   | X |


X's turn! pick a spot (1-9) → 7

   alright here's the board
 O |   |
---+---+---
```

PROBLEM STATEMENT OF THE PROJECT

The main aim is to build, in Python, a complete, functional two-player command-line application based on the well-known game of Tic-Tac-Toe. The application should start up and maintain a standard 3x3 game board where players, one using 'X' marks and one using 'O' marks, take turns. The key part during development is performing proper input validation to make sure players can only make moves on the board by selecting numbers, for example, 1-9, which refer to an unoccupied square. The crux of the matter lies in correctly programming the gameplay logic of checking the board after every move for two possible outcomes: either a winning condition, namely, three similar marks across horizontally, vertically, or diagonally, or a draw condition, that is, all nine squares are filled without any winner. Lastly, the application should have a very clear and consistent output, that is, it should present after every turn the present state of the board and declare the winner of the game.

Functional Requirements

No.	Feature	Description
1	**Display Board**	Presents the 3x3 game state as a clean grid using proper console separators (`
2	**Accept Player Moves (1–9)**	Reads user input corresponding to the nine board positions, where '1' is the top-left and '9' is the bottom-right.

3	**Input Validation**	Rejects moves if the input is not a number, is outside the 1-9 range, or attempts to place a mark on an already-taken spot.
4	**Win Detection**	Implements the logic to check all eight possible winning combinations (three horizontal, three vertical, two diagonal) after every successful move.
5	**Draw Detection**	Detects the end-game condition where all nine spots on the board are filled, but no winning combination has been achieved.
6	**Alternate Turns (X → O)**	Automatically switches control from the current player ('X' or 'O') to the opposing player after a valid move.
7	**Friendly Messages**	Provides clear, engaging, and casual messages to announce the game's state (who's turn it is, win announcements, and draw/game-over text).

Non-Functional Requirements

Requirement	My Implementation	Why it Matters
Performance	Runs instantly with no noticeable delay, even on older systems.	Console applications must be highly responsive and feel snappy to the user.

Simplicity	Implemented using only built-in Python features (`list`, `print`, `input`), contained entirely within a single `.py` file.	Ensures maximum portability, allowing anyone to run the game without installing external dependencies.
Reliability	Comprehensive `try-except` blocks are used to handle `ValueError` (non-number input) and other potential errors.	The game never crashes or terminates abruptly due to unexpected or invalid user input.
Usability	Features clear, friendly instructions, descriptive messages, and intuitive 1-9 numbering that matches the board layout.	Creates a low barrier to entry, making the game accessible and enjoyable for players of all ages.
Security	The application is entirely isolated: it involves no file I/O, database connections, or network communication.	It is a perfectly safe, self-contained beginner project with no security vulnerabilities related to data access.
Scalability	The entire state is held in-memory and managed by simple global variables (or a class, in the enhanced version).	While intentionally limited in-scope (no persistence), the clean structure is ready for future enhancements, like adding an AI opponent.

System Architecture

Very simple → **Single-file console application**

No GUI, no database, no internet — perfect for learning Python basics.

Workflow Diagram

```
Start

↓

Welcome Message + Instructions

↓

Show Empty Board

↓

Loop (until win or draw)

├── Get move from current player (X/O)

├── Validate input (1–9, not taken)

├── Place X or O on board

├── Show updated board

├── Check if current player won → YES → Announce winner!

├── Check if board full → YES → Declare draw

└── Switch player (X ↔ O)

↓

Thank you message

↓

End
```

Design Decisions I Made

1. Used a **list of 9 strings** to represent the board — simple and fast

2. Positions 1–9 map to indices 0–8 (subtract 1 from user input)

3. Win conditions stored in a list of tuples — clean and readable

4. Used `while moves < 9` instead of `while True` — easier to track draws

5. Added **funny and casual messages** to make it feel friendly and human

8. Implementation Highlights

- Total code: **~100 lines** (super small and clean!)

- Everything in **one file** — easy to share and run

- Used **f-strings** for clean formatting

- Proper **error handling** with try-except

- Hilarious win messages like “LETS GOOOO X WINS!!! absolute legend”

Testing Approach

I tested the game thoroughly:

- Typed letters → didn’t crash

- Typed 0 or 10 → “not on the board” error

- Tried to place on taken spot → correctly blocked

- Played full games → correctly detected X win, O win, and draw

- Completed all 9 moves → correctly announced draw (“the cat wins again”)

- Sudden terminal close → expected behavior (no save)

Challenges Faced & How I Solved Them

Challenge	What I Did (Solution)
Off-by-one errors (1 vs 0)	Solved by strictly enforcing the conversion logic: Always take the user's 1-9 input and convert it to the Python list index 0-8 using `pos = int(move) - 1`.
Forgetting to switch turns	Implemented a clean, conditional assignment at the end of every successful turn: `turn = "O" if turn == "X" else "X"`.
Win not detected immediately	The `did_someone_win(turn)` check is executed **immediately after** the current player's symbol is placed on the board, preventing the turn from switching prematurely.
Board looked ugly	Enhanced the `show_board()` function by adding horizontal lines (`---+---+---`) and appropriate spacing to render a visually clean, classic 3x3 grid layout.
Boring messages	All user-facing text (welcome, turn prompts, win/draw announcements) was rewritten to be super fun and

	casual, improving the overall user experience and personality.

Learnings

- Input validation saves lives (and prevents crashes)

- Small UI details (like board layout) make a huge difference

- Writing clear, funny messages makes people actually enjoy your app

- Keeping code simple > showing off complex features

Future Enhancements (My Dream List)

Play against Computer (AI)	To introduce a single-player mode, allowing the user to practice and beat the machine in a direct challenge.
Unbeatable AI (Minimax)	To provide the ultimate challenge by implementing the Minimax algorithm, ensuring the computer never loses and requires true strategic thinking.
Colorful board (using `colorama`)	To significantly improve the aesthetics and user experience by using color in the terminal for 'X', 'O', and the board lines.

Play again option	To allow players to start a new game immediately without the friction of restarting the Python script manually.
Sound effects (e.g., beep on win)	To enhance user feedback and provide maximum nostalgia and satisfaction upon winning the game.
Score tracker	To keep a running tally of wins for Player X, Player O, and Draws, establishing who is the ultimate champion over multiple rounds

Thank you for playing my game! Hope you had fun