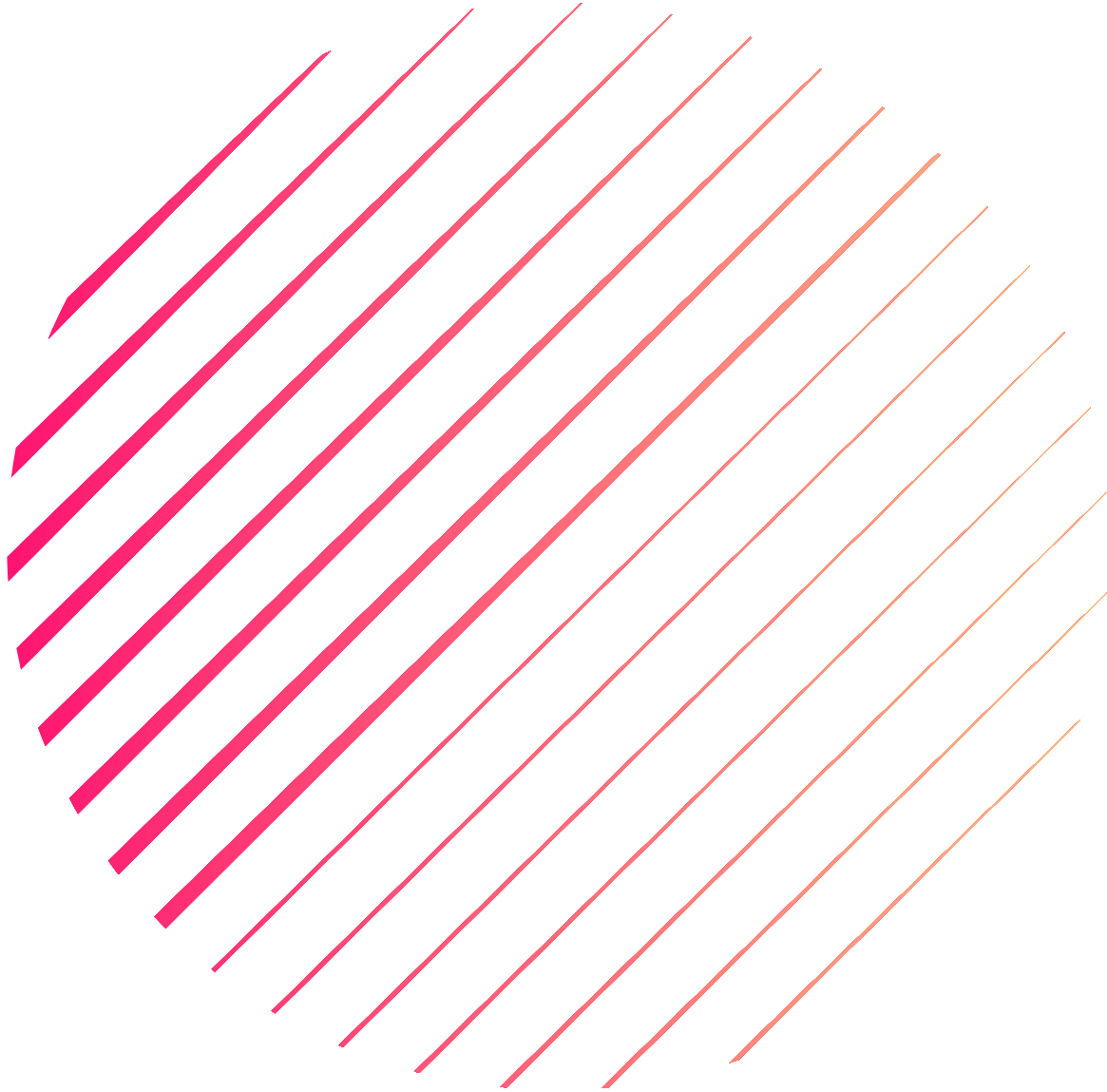


# N x N Tic-Tac-Toe Game



26 09 2025  
SE1012

SAJITH Y D  
IT25100449



## INTRODUCTION


This project implements a flexible Tic-Tac-Toe game. that supports board sizes of  $N \times N$  ( $N$  from 3 to 10) and up to three players (including computer players). The game is written in C and features input validation, dynamic memory allocation, game logging, and both human and computer play modes.

## DESIGN AND APPROACH

The program is structured into multiple files:

- game.c - Contains the main logic, game loop, and user interface.
- Hdr.h, hdr.c - Provide utility functions, such as timestamp generation for logging.

Key features:

- Dynamic board allocation for any size  $N$ .
  - Support for 1–3 players, with computer opponents.
  - Input validation for game mode, player count, and moves.
  - Game state logging with timestamps.
- 

---

## IMPLEMENTATION

### Main features:

- Game Modes: Single player, two players, three players, or exit.
- Player Types: Human and computer players are supported. The computer selects random valid moves.
- Board Representation: The board is a dynamically allocated 2D array.
- Win Checking: The game checks for wins in rows, columns, and both diagonals.
- Logging: Each move and the board state are logged to gamelog.txt with a timestamp.

### Important functions:

- `main()`: Handles user input for game setup and starts the game.
- `gameplay()`: Manages turns for each player and the computer, checks for win/draw, and logs the board.
- `drawBoard()`, `resetBoard()`, `addToBoard()`, `isValidMove()`: Board management utilities.
- `game_logic()`: Checks for win/draw, and updates the board.
- `writeFile()`: Logs the current board state to a file with a timestamp.

### Challenges faced during development:

- Extending the game from two players to 3 players with computer as opponent.
- Checking win, draw and continue conditions.
- Validating user inputs.
- Keeping the terminal window clean and simple.
- Some libraries and commands depend on the running OS.

---

## Solutions applied:

- Extended the `gameplay()` function with additional players turns and computer turn with using logical conditions.
- Defined a custom (enum) data type to store and return game states (WIN, DRAW, CONTINUE).
- Prompting user for an input until a valid input is received using do while loop to validate inputs.
- Cleaning the terminal with `cls/clear` system commands depending on the OS.
- Check the OS using `defined(_WIN32/ _WIN64/ __linux__ )` to decide and use of proper function and library according to the OS.

## TESTING

The program was tested with:

- Different board sizes (e.g., 3x3, 4x4, 5x5).
- All player configurations (1, 2, and 3 players).
- Edge cases, such as invalid moves and board boundaries.
- Computer vs. human play.

## RESULTS

- The game correctly alternates turns between players and computer.
- Win and draw conditions are detected and reported.
- Invalid moves are rejected and re-prompted.
- The board state is logged after each move with a timestamp.
- Memory is properly allocated and freed.

---

## CONCLUSION

This project demonstrates dynamic memory management, input validation, and modular programming in C. The flexible design allows for easy extension to more players or different board sizes. The logging feature provides a useful record of gameplay.

Possible improvements:

- Smarter computer AI.
- GUI interface.
- Undo/redo functionality.

## REFERENCES

- [1] "C Tutorial." W3Schools. <https://www.w3schools.com/c/index.php>.  
[2] "C Programming Tutorial." GeeksforGeeks.  
<https://www.geeksforgeeks.org/c/c-programming-language/>.  
[3] "Stack Overflow." Stack Overflow. <https://stackoverflow.com/>.