Project Report

Spring 2025

Name: Rakshith
Project: Game

Note: As you write each section, try to be as clear and detailed as possible. Your goal is to communicate your thought process and work clearly. Don't worry if you faced challenges or made mistakes; discussing these is a valuable part of learning and shows your problem-solving skills! Remember, there's no single 'right' way to do these tasks, so be creative and honest in your responses.

Problem Statement (2-3 Paragraphs):

The goal of this assignment was to build a working tic tac toe game that runs in the console. The program needed to let two players take turns placing X and O markers on a 3x3 board, keeping track of the moves, and correctly determining when the game ended. The assignment required detecting all winning conditions, identifying ties, and stopping the game with the proper result message.

The inputs for the program were the row and column the player wanted to select, both in the range 1-3. The output was the updated game board after each move along with messages for invalid moves, wins, or ties. Several things could go wrong during the game like entering values not in the range, typing letters or characters instead of numbers, or choosing a position that was already taken. I added checks for each of these situations and printed clear error messages so the game never crashes and always asks for a new move.

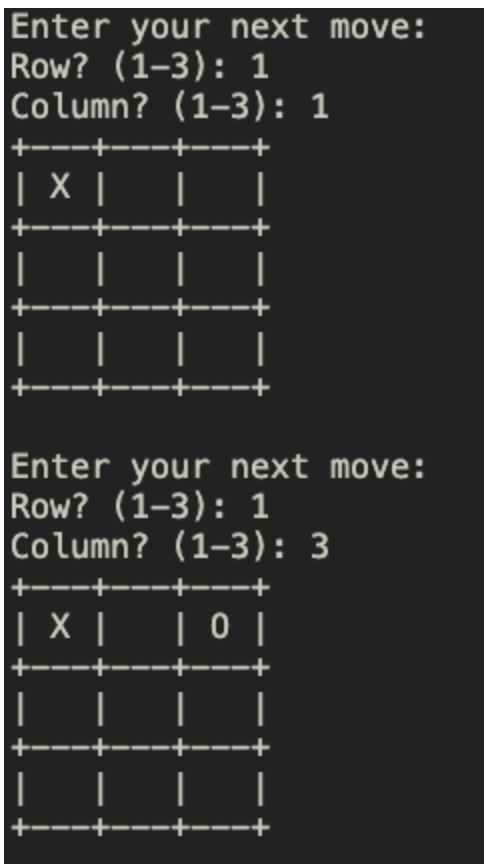
Design (1-3 Paragraphs):

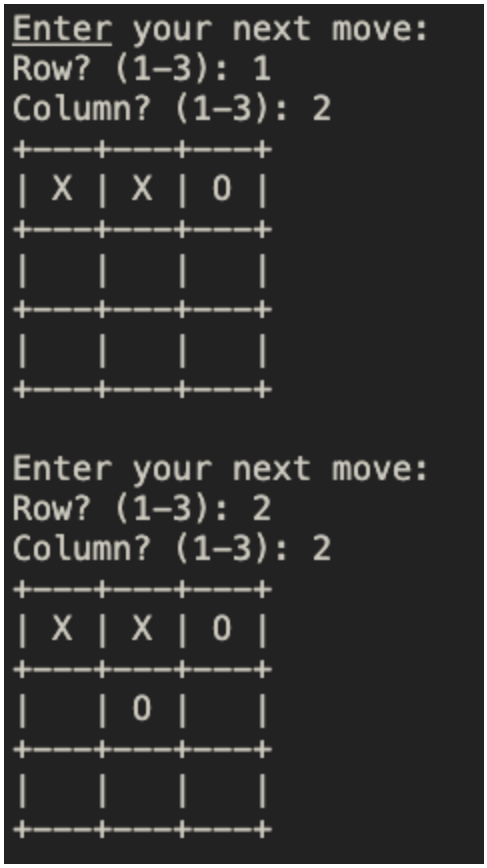
I used a single list containing nine elements to represent the entire game board. This matched the assignment requirement and kept the indexing simple. Each space started with a blank drink, and I used simple math to convert the row and column inputs into the correct index in the list. I created functions to display the board, check for wins, switch players, and check if the board is full.

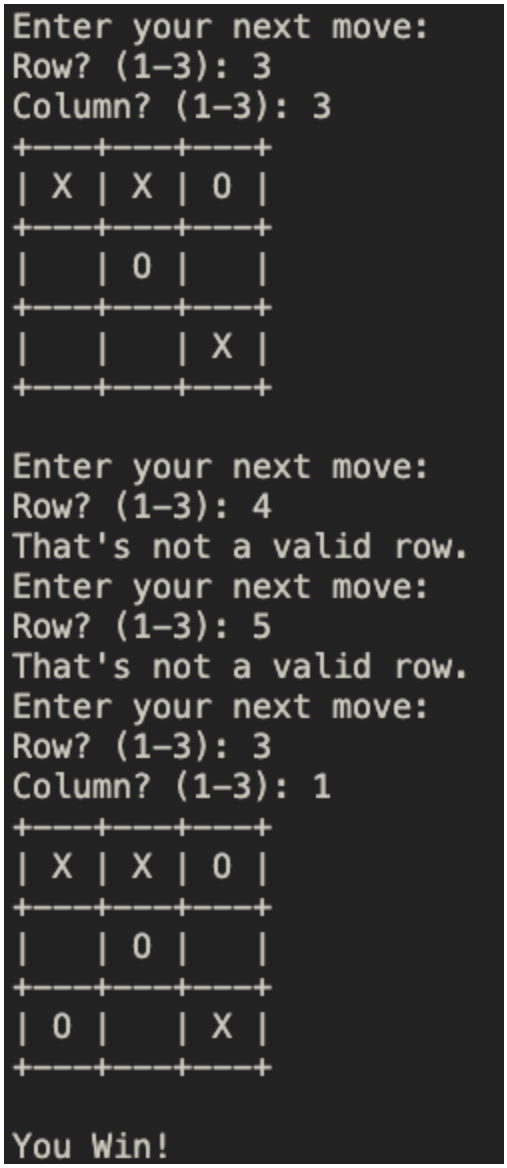
The solution is simple because each function has one job. The check_win function follows the regular game patterns by checking all rows, columns, and both diagonals using basic comparisons. The input validation uses simple conditional checks to ensure cleaner error handling. The approach works looking back because the logic is easy and avoids unnecessary complexity. The only thing I might change later would be grouping repeated win-checking patterns in a loop, but the current version meets the assignment expectations clearly.

Testing (1-2 Paragraphs + screenshots of 3 test cases):

I tested the program by running it multiple times with different outputs. Normal tests included choosing open positions on the board. Special cases, I put rows and columns outside the range and it printed the error message. My program worked as expected.


Enter your next move:
Row? (1-3): 1
Column? (1-3): 1
| X | | |
Enter your next move:
Row? (1-3): 1
Column? (1-3): 3
| X | | O |


Enter your next move:
Row? (1-3): 1
Column? (1-3): 2
| X | X | O |
Enter your next move:
Row? (1-3): 2
Column? (1-3): 2
| X | X | O |
| | O | |



Conclusion (1 paragraph)

This project turned out well and met all the requirements. I learned how to manage a board using a simple list, how to validate user input safety, and how to organize the program using functions for clarity. The game correctly recognizes wins, ties, and invalid moves, while keeping the gameplay smooth. If I needed to add onto this project, I would likely add a computer player or make the UI better, but for the goals, I think this project is complete.