Tic Tac Toe game

I am glad you are reviewing this doc. I wrote this to explain how I went about solving this question for you, my thought process and methodology, so you can better undestand my skills.

I’m very familiar with this game and have played it many times before. However, I quickly checked the rules and associations to make sure everything was clear.

Here is a summary of rules.

Two players: There are two players, X and O, in the game.

Toggle Turns: Players alternate between X and O.

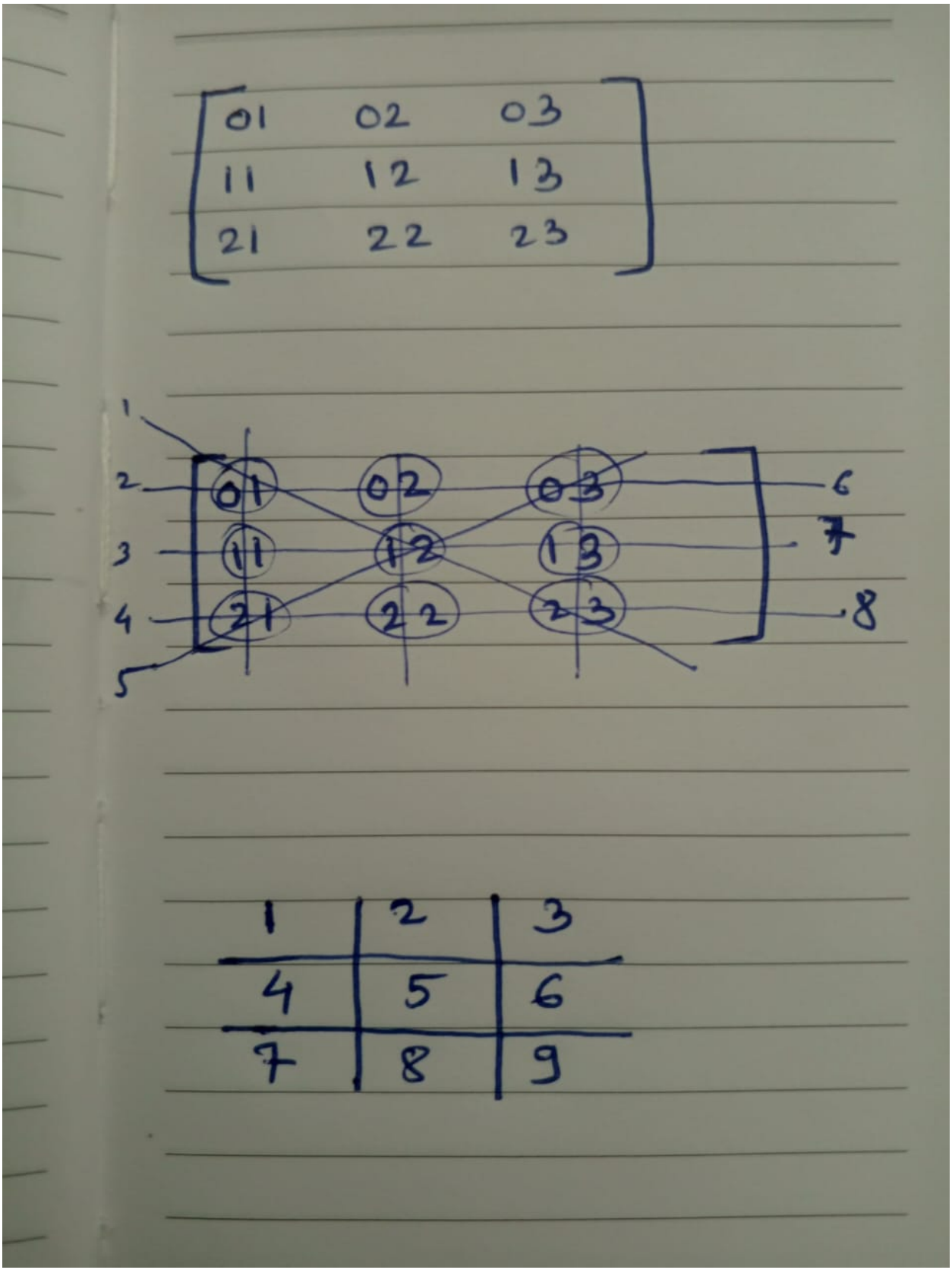
Outcome of the game: There are three possible scenarios for the player: WIN, LOSE, or DRAW.

Location: The game is played on a 3×3 board, which means there are three possibilities for each with 9 possible positions: see, X, or O.

Winning Position: The goal is to create a horizontal, vertical, or horizontal model of three consecutive moves. The first player to achieve this wins the game. If no one can reach this after 9 moves, the game ends in a draw.

How I decided on the user interface
According to the problem statement, it should be based on the text input . So I chose to create a simple terminal-based game where users input their moves through the terminal and see the displayed output.

A 2D structure was the most appropriate choice to store results and trends. It correctly represents the grid layout of the board and records the move points.


01 02 03
11 12 13
21 22 23
1
2
3
4
5
01
02
03
11
12
13
21
22
23
6
7
8
1 2 3
4 5 6
7 8 9

To solve the problem, I started by sketching a 2D array on paper, mapping out the possible row and column positions. I then focused on determining how to identify a winner or if the game ends in a draw. According to the game rules, a player wins if they manage to align three consecutive symbols (either O or X) vertically, horizontally, or diagonally. If no such alignment occurs after all moves, the game is a draw.

There are eight possible patterns that need to be checked to determine a win. Since each player can make a maximum of five moves, and X always starts the game, X can have up to five moves while O can have up to four. To form a winning pattern, a minimum of three moves is required, which means a draw can only occur if all positions are filled without a winning pattern.

To make it user-friendly, I decided to allow users to enter a single number rather than specifying row and column positions like (1,1) or (2,1). This number maps directly to the grid positions, simplifying the input process.

The problem statement emphasized avoiding global variables and modularizing the code. Therefore, I opted to use Object-Oriented Programming (OOP) principles. I created classes and functions to store game moves in object attributes and, upon game completion, to save these moves and results in a SQLite database.

I often find myself deciding whether to tackle the core problem first or to develop the user interface. In this case, I chose to begin with creating a user interface, including a friendly welcome message and a display of the game rules.

I started by displaying a welcome message and waiting for the user to press the Enter key to begin the game. Once the Enter key was pressed, I showed the initial board with position numbers, so users could select a number from 1 to 9 to make their move.

To manage the game state, I created a 2D array named `$board` to store the moves. Unfortunately, I initially forgot to add the project to Git, but I corrected this oversight.

Initially, I handled input for only one user, verifying if the input position was correctly updated with the respective position. Later, I added a toggle function to switch between users after each move and ensured that the symbols X and O were placed correctly according to the user's input.

I then developed a function to check if a player has won. This function returns true or false, indicating whether a player has won or not. If no winner is detected, the game continues with the next player. To avoid unnecessary checks, I added a condition that requires at least 5 moves to be played before checking for a win, as a player cannot win before this threshold.

With these elements in place, the game was functioning well. I then focused on fine-tuning, adjusting colors, and refactoring the code to enhance readability and maintain a clean codebase.

I written unit test code to check a couple of functions in my class. While I'm still learning the ins and outs of unit testing, I've managed to test two important

functions.

I’m genuinely excited about the chance to bring my skills to new and interesting projects. I hope you enjoy the game I’ve developed, and I’m looking forward to discussing how I can contribute to your team. Thanks for considering me for this opportunity—I can’t wait to chat more about how I can add value to your work!