Tic-Tac-Toe: 2D Arrays Assignment

The Project

You are finishing a two-player Tic-Tac-Toe game with a Swing window. Players take turns pressing 1 to 9 (squares are numbered left to right, top to bottom). A dialog announces a winner or a draw and lets you play again. The window, key handling, and game flow are already written. The game logic, which stores the board in a 2D array, is yours to write.

Your file: `TicTacToeModel.java`. Do not edit the other files. The board is `public final char[][] board = new char[3][3]`. An empty square holds `TicTacToeModel.EMPTY` (a space) and a taken square holds `'X'` or `'O'`. The array is indexed `board[row][col]`, with row and col from 0 to 2.

The key idea: the player types a square number `i` from 1 to 9, but the array needs a row and a column. You will check the square is empty, update the 2D array, and write the code to check if the game is over.

How to Test

The file `Tester.java` checks your model with no window needed. Run it after every step:

Each line is PASS or FAIL with a reason. Before you write anything, most tests fail (20 pass, 63 fail). That is expected. Your goal is 83 passed, 0 failed. Only work on the section for your current step and ignore the failures below it. When a test fails, read its message, then trace your code to debug.

To-Do List: Do These In Order

☐ **Step 1. Constructor (marked // A)**

- Use nested loops to set every square of `board` to `EMPTY`.
- **Test:** run the Tester. The "=== Constructor ===" section must be all PASS. Then run TicTacToe: every square should show its number.

☐ **Step 2. checkValid(int i)**

- Return `false` if `i` is outside 1 to 9. If it's ok, return `true` if that square is `EMPTY`.
- **Test:** "=== checkValid ===" must be all PASS. Some of these tests already pass before you start, because the stub says every move is valid. Make sure every one passes, including the out-of-range squares and the full board. Later steps depend on this method.

☐ **Step 3. updateBoard(int i) (marked // B)**

- The invalid-move check is given. Replace // B: convert i to row and col, then store the current player's mark. Use getCurrentPlayer() or the currentPlayer field.
- **Test:** "=== updateBoard ===" must be all PASS (it checks all 9 squares for both X and O, and that other squares do not change). Then run TicTacToe. You can place marks, and

pressing a taken square shows a message but does not use the turn. The game will never end yet.

☐ **Step 4. getWinner() (marked // C)**

- Return `'X'` or `'O'` if a player has three in a row, otherwise `EMPTY`. Check 3 rows, 3 columns, and 2 diagonals. A line of three blank squares is not a win.
- Hint: Build it in pieces: rows first, then run the Tester, then columns, then diagonals.
- **Test:** "=== getWinner ===" must be all PASS. This includes a draw board and a board with only two in a row.

☐ **Step 5. isGameOver() (marked // D)**

- The game is over if `getWinner()` is not `EMPTY`, or if no empty squares remain. Call your own getWinner() and use a nested loop to look for an `EMPTY`. Watch the order: a win on the 9th move must count as a win.
- **Test:** "=== isGameOver ===" must be all PASS.

☐ **Step 6. Final check**

- **Tester:** 83 passed, 0 failed.
- **Play:** run TicTacToe and play three games: an X win, an O win, and a draw. Try a win on the last square.

Hand In

- `TicTacToeModel.java` (only this file

Challenges:

1. **Start with a random player each game**
2. **Change the size of the board and the squares**
3. **Change the color of the squares, the font, the font size, etc**
4. **Make X and O show up in different colors**
5. **Modify the game to 4x4**