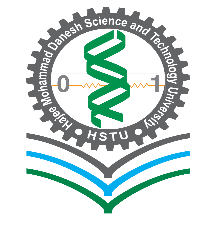
****

**HAJEE MOHAMMAD DANESH SCIENCE AND TECHNOLOGY UNIVERSITY DINAJPUR-5200**

**PROJECT REPORT: Tic-Tac-Toe**

**Submitted By**

**Name: Mahmudul Hasan Tamim**

**Roll: 2002002**

**Batch: 2020**

**Course Code: CSE254**

**Level: 2 Semester: II**

**Department: Computer Science and Engineering**

**Submitted To:**

**Dr. Md. Abdulla Al Mamun**

**Professor**

**Department of Computer Science and Technology,**

**Hajee Mohammad Science and Technology University**

**DATE OF SUBMISSION: 30 SEPTEMBER, 2024**

**Report on the Tic-Tac-Toe Java Program**

**1. Overview of the Program**

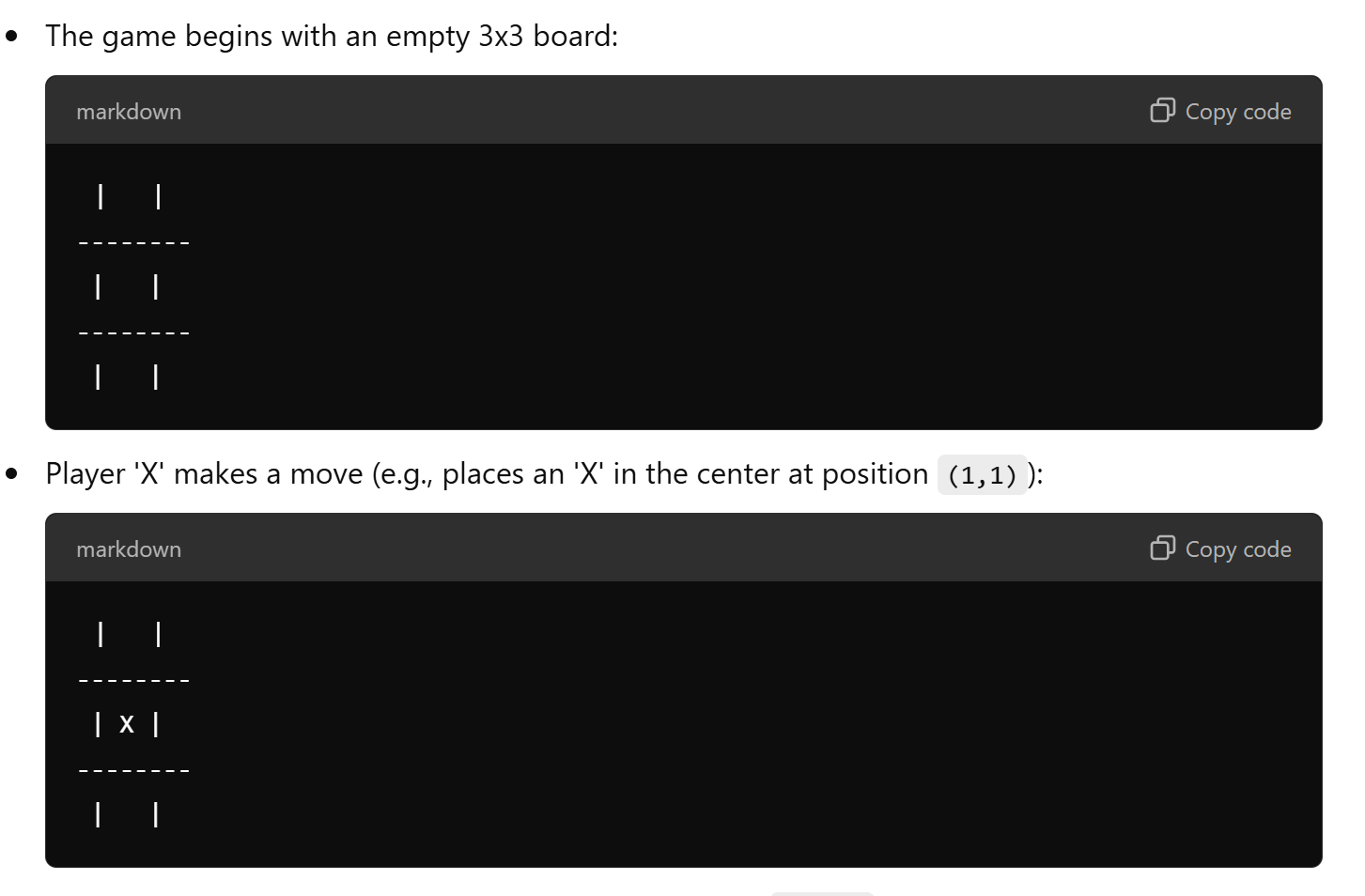
The provided Java program implements a simple text-based **Tic-Tac-Toe game** for two players (Player 'X' and Player 'O'). Players take turns entering their desired position on a 3x3 grid, and the game continues until one player wins or the game ends in a draw.

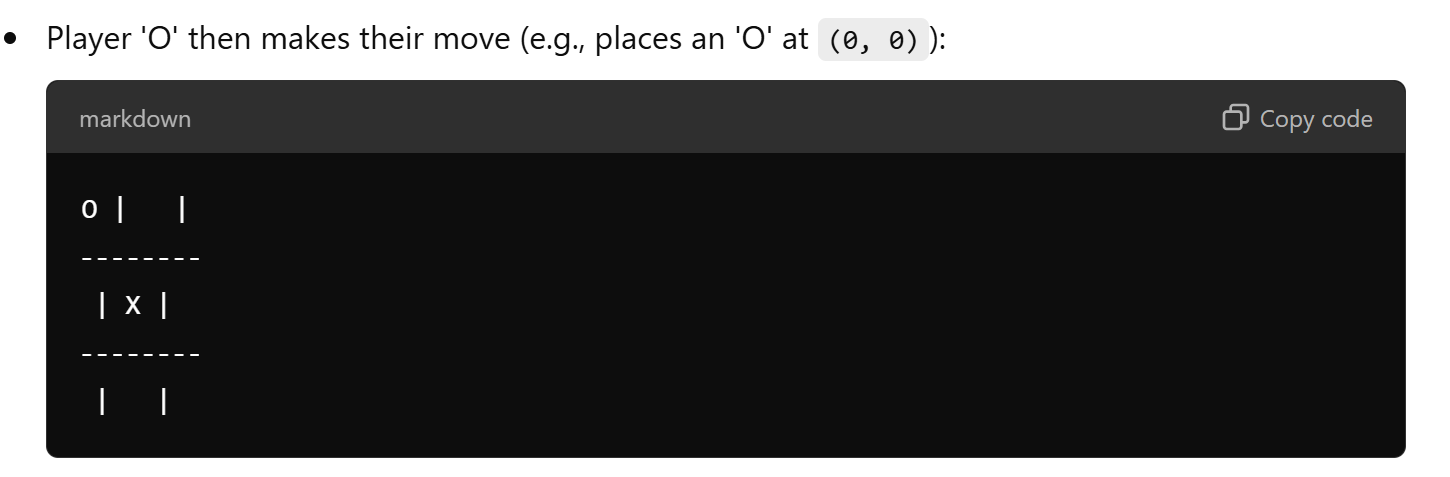
**2. Structure of the Program**

* **Initialization:** The board is represented by a 2D array of characters (char[][] board = new char[3][3];). Initially, all the positions are set to empty spaces (' ').
* **Game Loop:** The game operates in a while loop, where players take turns:
  + The board is printed to display the current state.
  + The player enters the row and column where they want to place their marker.
  + If the selected cell is empty, the player's marker is placed in that position, and the game checks if the player has won.
  + If the player wins, the game ends and declares the winner.
  + If not, the turn alternates between Player 'X' and Player 'O'.
* **Win Check:** The haveWon() method checks for a win condition by evaluating:
  + Rows: If all elements in any row are the same as the player's marker.
  + Columns: If all elements in any column are the same as the player's marker.
  + Diagonals: If the diagonal elements are the same as the player's marker.
* **Invalid Moves:** If a player tries to place their marker in a cell that is already occupied, the game outputs an error message ("Invalid move. Try again!") and the player gets another chance.

**3. Key Components**

* **Game Board:** A 3x3 grid is represented using a 2D character array char[][] board. Initially, all positions are set to ' ' to indicate they are empty.
* **User Input:** Player input is captured using the Scanner class. Players enter the row and column (0-indexed) where they want to place their marker.
* **Game Mechanics:**
  + The turn alternates between Player 'X' and Player 'O' using a ternary operator.
  + After every move, the game checks for a winning condition using the haveWon() method. This method checks for wins in rows, columns, and diagonals.
* **Win Conditions:**
  + **Rows:** A win occurs when all three elements in any row are the same as the player's symbol.
  + **Columns:** A win occurs when all three elements in any column are the same.
  + **Diagonals:** Two diagonals are checked, and a win occurs if all the elements in either diagonal match the player's symbol.
* **4. Example of Gameplay**
* Here’s a sample flow of the game:

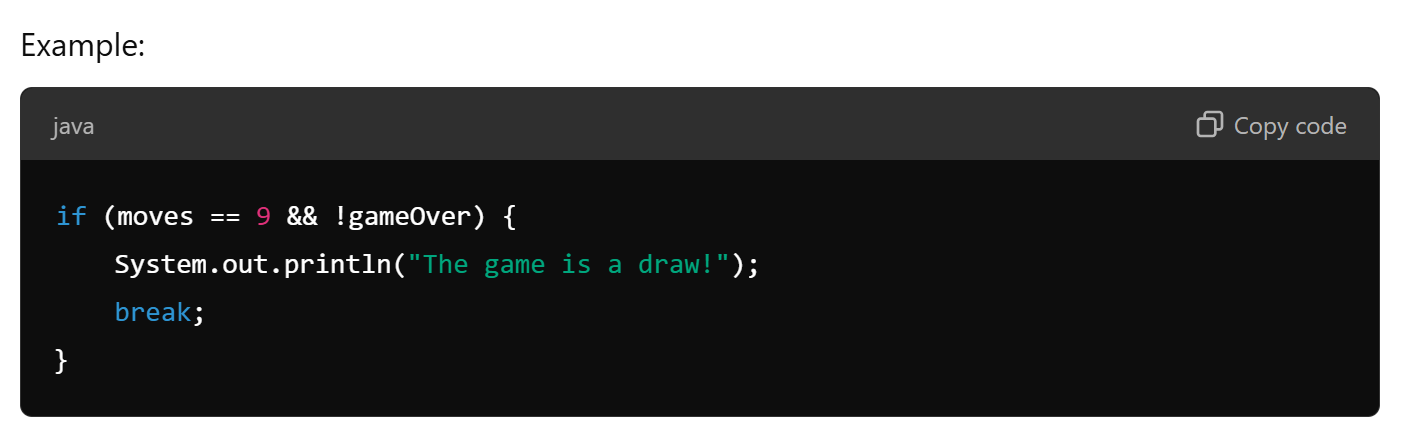




* The game continues, alternating between players until a win is detected, or all positions are filled (a draw).

**5. Potential Enhancements**

* **Draw Condition:** The current version does not handle the scenario where the board is full, and no player has won. A simple counter to track the number of moves could be added to detect a draw condition when the board is filled without a winner.



* **Input Validation:** The program assumes that players will always enter valid numbers (0, 1, or 2) for row and column. Input validation could be added to handle invalid inputs (e.g., negative numbers or numbers greater than 2).
* **Improved Board Display:** The current board display separates cells using |, but additional formatting could be applied to make the board easier to read, such as adding grid lines between rows.

**6. Code Readability and Efficiency**

* **Code Structure:** The code is clear and easy to follow, with appropriately named variables and functions.
* **Efficiency:** The program efficiently checks for a win after each move. Since it only evaluates rows, columns, and diagonals directly involved in the player's last move, it avoids unnecessary checks.

**7. Conclusion**

This Java program implements a fully functional Tic-Tac-Toe game with basic features like win detection and input validation. However, there is room for improvement, particularly in handling invalid inputs and draw conditions. The overall structure is simple and well-suited for beginner-level programming practice.

Top of Form

Bottom of Form