Tic Tac Toe Game

In Java

Study Course

B202 Advanced Programming

By

Chehab Hany Mohamed Elsayed Elsayed Elmenoufi

Student Number: GH1034223

Under the Guidance of

Prof. Mazhar Hameed

GitHub URL: https://github.com/Chippo90/Advanced-Programming/tree/main

Video Recording URL: https://youtu.be/njPwROlc2Jc

Gisma University of Applied Sciences

Berlin, Germany

December 2025

Table of Contents

1. Introduction

This project presents a Java implementation of a Tic Tac Toe game with a graphical user interface GUI. The purpose of the project is to demonstrate practical understanding of Object Oriented Programming through class design, encapsulation, interactions between objects, and GUI building using java.swing and java.awt.

The simple Tic Tac Toe game was selected because it provides a clear problem domain involving game logic, user interaction, win/draw detection, and basic artificial intelligence. The project also extends the traditional game by allowing the player to choose from four symbols: X, O, ▲, and ■**,** offering a more flexible user experience.

The main objective is to design a simple, clean, and interactive application that show an understanding of Java programming and OOP principles.

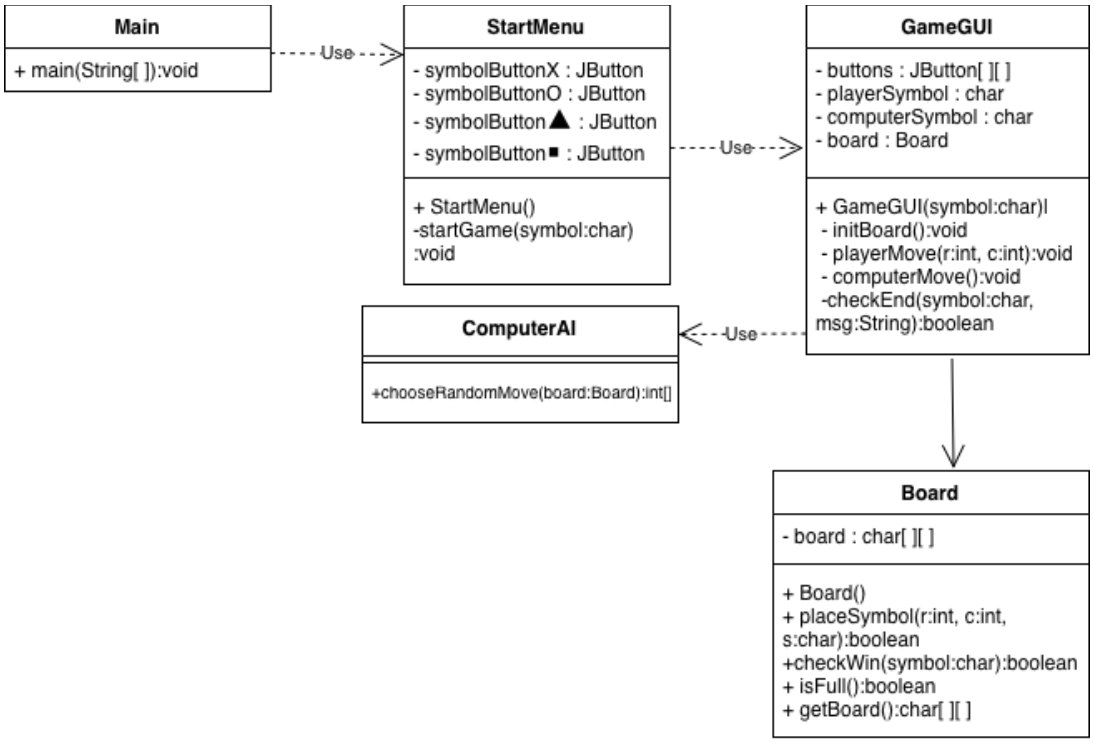
2. System Architecture

2.1 Overview

The system is divided into multi classes, each class is responsible for a single aspect of the system. This structure makes the project easier to understand and maintain:

- **Main**: Launches the application.
- **StartMenu**: Displays a symbol selection window.
- **GameGUI**: Represents the main game window and handles the game during playing.
- **Board:** Manages the game rules and logic.
- **ComputerAI**: Provides move selection for the computer's turn.

2.2 Class Diagram



2.3 Class Responsibilities

- **Main**
  - Program entry point.
  - Creates the start menu.

```
// Main class - entry point of the program
public class Main {
    public static void main(String[] args) {
        // Start the initial window where the user chooses a symbol
        new StartMenu();
    }
}
```

- **StartMenu**
  - Displays four buttons for symbol selection.

```
public StartMenu() {  1 usage
    // Set window title and size
    setTitle("Choose Your Symbol");
    setSize( width: 300,  height: 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Layout with 4 rows (one button per row)
    setLayout(new GridLayout( rows: 4,  cols: 1));

    // Buttons for the four symbol choices
    JButton xBtn = new JButton( text: "X");
    JButton oBtn = new JButton( text: "O");
    JButton triBtn = new JButton( text: "▲");
    JButton sqBtn = new JButton( text: "■");

    // Assign actions to buttons and start game with chosen symbol
    xBtn.addActionListener( ActionEvent e -> startGame( playerSymbol: 'X'));
    oBtn.addActionListener( ActionEvent e -> startGame( playerSymbol: 'O'));
    triBtn.addActionListener( ActionEvent e -> startGame( playerSymbol: '▲'));
    sqBtn.addActionListener( ActionEvent e -> startGame( playerSymbol: '■'));

    // Add buttons to window
    add(xBtn);
    add(oBtn);
    add(triBtn);
    add(sqBtn);

    // Make window visible
    setVisible(true);
}
```

  - Launches the main game window with the chosen symbol.

```
// Starts the game with the chosen symbol and closes this menu
private void startGame(char playerSymbol) {  4 usages
    new GameGUI(playerSymbol);
    this.dispose();
}
```

- **GameGUI**
  - Creates a 3x3 grid using JButtons.

```
private JButton[][] buttons = new JButton[3][3];   // 3×3 button grid  5 usages
private char playerSymbol;                         // Player's chosen symbol  4 usages
private char computerSymbol;                       // Computer auto picks another symbol  4 usages
private Board board;                               // Board logic object  6 usages
```

- Handles player input.

```
// Called when the player clicks a cell
private void playerMove(int row, int col) {  1 usage
    // Place player's symbol if cell is empty
    if (board.placeSymbol(row, col, playerSymbol)) {

        // Update button text
        buttons[row][col].setText(String.valueOf(playerSymbol));
        buttons[row][col].setEnabled(false);

        // Check if player won
        if (checkEnd(playerSymbol,  message: "You win!"))
            return;

        // Computer makes its move
        computerMove();
    }
```

- Delegates game logic to Board.

```
// Create and place buttons on the board
private void initBoard() {  1 usage
    Font font = new Font( name: "Arial", Font.BOLD,  size: 40);  // Large symbol font

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {

            // Create a new button for each cell
            JButton btn = new JButton( text: "");
            btn.setFont(font);

            int row = i, col = j; // Needed for lambda expression

            // Handle player's move when button is clicked
            btn.addActionListener( ActionEvent e -> playerMove(row, col));

            buttons[i][j] = btn;
            add(btn);
        }
    }
}
```

- Handles computer moves using ComputerAI.

```
// Computer makes a random move
private void computerMove() {  1 usage
    int[] move = ComputerAI.chooseRandomMove(board);
    int row = move[0];
    int col = move[1];

    // Update board and button
    board.placeSymbol(row, col, computerSymbol);
    buttons[row][col].setText(String.valueOf(computerSymbol));
    buttons[row][col].setEnabled(false);

    // Check if computer won
    checkEnd(computerSymbol,  message: "Computer wins!");
}
```

- **Board**
  - Stores the game board as a 3x3 char array.

```
private char[][] board = new char[3][3];   // Internal 3×3 character grid

public Board() {  1 usage
    clearBoard();
}

// Fills board with empty spaces
public void clearBoard() {  1 usage
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            board[i][j] = ' ';
}
```

- Provides methods for placing symbols, checking wins and draws.

```
// Places a symbol if the cell is empty
public boolean placeSymbol(int row, int col, char symbol) {  2 usages
    if (board[row][col] != ' ')
        return false;

    board[row][col] = symbol;
    return true;
}

// Checks all rows, columns, and diagonals for a win
public boolean checkWin(char symbol) {  1 usage

    // Check rows and columns
    for (int i = 0; i < 3; i++) {
        if (board[i][0] == symbol && board[i][1] == symbol && board[i][2] == symbol)
            return true;

        if (board[0][i] == symbol && board[1][i] == symbol && board[2][i] == symbol)
            return true;
    }

    // Check main diagonal
    if (board[0][0] == symbol && board[1][1] == symbol && board[2][2] == symbol)
        return true;

    // Check opposite diagonal
    if (board[0][2] == symbol && board[1][1] == symbol && board[2][0] == symbol)
        return true;

    return false;
}
```

- **ComputerAI**
  - Chooses a random empty position for the computer's move.

```
// Returns a random valid move (row, col)
public static int[] chooseRandomMove(Board board) {  1 usage
    List<int[]> moves = new ArrayList<>();
    char[][] grid = board.getBoard();

    // Collect all empty cells
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (grid[i][j] == ' ') {
                moves.add(new int[]{i, j});
            }
        }
    }

    // Pick one at random
    return moves.get((int)(Math.random() * moves.size()));
}
```

3. Implementation

3.1 Processes

- Planning the OOP structure.
- Building the game logic.
- Designing the GUI.
- Connecting the GUI to the board logic.
- Creating the computer opponent.
- Testing the game for Win/Draw.

3.2 OOP Principles

- **Encapsulation:** The project emphasizes encapsulation by separating UI, game logic, and AI into distinct classes. (Wilkinson, 2021)
- **Abstraction:** Each class exposes only the methods required by other components, promoting maintainability and clarity**.** (Wilkinson, 2021)
- **Modularity:** Instead of using inheritance, the design relies on modularity and composition, which keeps the responsibilities clean and avoids unnecessary coupling. (Wilkinson, 2021)

3.3 GUI Implementation

The GUI was built using:

- **Jframe**: Windows for StartMenu and GameGUI. (*JFrame (Java Platform SE 8)*, no date)
- **Jbutton**: Game cells and selection buttons. (*JButton (Java Platform SE 8)*, no date)
- **GridLayout**: Used for the 3×3 board. (*GridLayout (Java Platform SE 8)*, no date)
- **JoptionPane**: Displaying Win/Draw messages. (*JOptionPane (Java Platform SE 8)*, no date)

3.4 Game Logic

The board is stored in a char[ ][ ] array where: (*Tic-Tac-Toe Game in Java*, 15:34:58+00:00)

- ' ' means empty
- 'X', 'O', '▲', '■' represent Player/Computer symbols
- Key methods:
    - **placeSymbol**(): Validates and places symbols. (Technologies, 2024)

```
// Places a symbol if the cell is empty
public boolean placeSymbol(int row, int col, char symbol) {  2 usages
    if (board[row][col] != ' ')
        return false;

    board[row][col] = symbol;
    return true;
}
```

    - **isFull**(): Checks for draw conditions. (Technologies, 2024)

```
// True if no empty cells remain
public boolean isFull() {  1 usage
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (board[i][j] == ' ')
                return false;
    return true;
}
```

- **checkWin**(): Checks rows, columns, and diagonals. (Technologies, 2024)

```java
// Checks all rows, columns, and diagonals for a win
public boolean checkWin(char symbol) {  1 usage

    // Check rows and columns
    for (int i = 0; i < 3; i++) {
        if (board[i][0] == symbol && board[i][1] == symbol && board[i][2] == symbol)
            return true;

        if (board[0][i] == symbol && board[1][i] == symbol && board[2][i] == symbol)
            return true;
    }

    // Check main diagonal
    if (board[0][0] == symbol && board[1][1] == symbol && board[2][2] == symbol)
        return true;

    // Check opposite diagonal
    if (board[0][2] == symbol && board[1][1] == symbol && board[2][0] == symbol)
        return true;

    return false;
}
```

3.5 Computer AI

- Stores them in a list

```java
// Returns a random valid move (row, col)
public static int[] chooseRandomMove(Board board) {  1 usage
    List<int[]> moves = new ArrayList<>();
    char[][] grid = board.getBoard();
```

- Scans the board for empty cells

```java
// True if no empty cells remain
public boolean isFull() {  1 usage
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (board[i][j] == ' ')
                return false;
    return true;
}
```

- Randomly selects one

```java
// Pick one at random
return moves.get((int)(Math.random() * moves.size()));
```

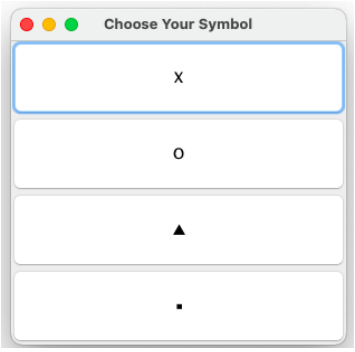
4. Results

4.1 Functions

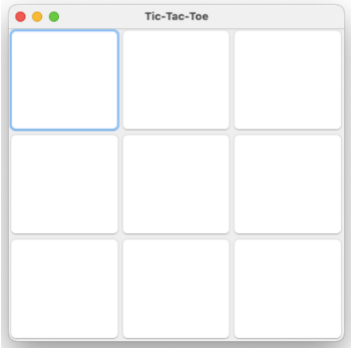
- The GUI is responsive and working properly.
- Shape selection works correctly.
- The player and AI take turns without error.
- Win, lose and draw detection logic works correctly.
- The program handles invalid moves by disabling used cells.

4.2 Screenshots

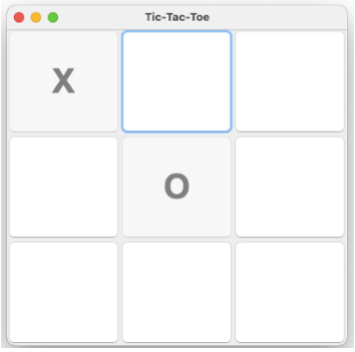
- **Start Menu:**



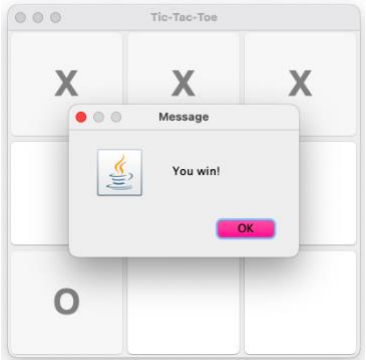
- **Game Board:**



- **Player/Computer Move:**



- **Winning Message:**



- **Losing Message:**

- **Drawing Message:**

5. Challenges and Solutions

5.1 Challenge #1: Connecting GUI and Logic

- **Description**: Swing had to update both the UI and the backend board.
- **Solution**: Separate logic into Board while GUI is getting the changes.

5.2 Challenge #2: Win Detection

- **Description**: Incorrect win detection due to edge cases in diagonals.
- **Solution**: Tested each win scenario individually and corrected the logic.

5.3 Challenge #3: Disabling Buttons

- **Description**: Buttons could still be clicked after being used.
- **Solution**: Used setEnabled(false) immediately after placing a symbol.

5.4 Challenge #4: Simple AI

- **Description**: Needed a quick and simple approach.
- **Solution**: Scanned empty cells and selected one randomly.

6. Conclusion and Future Work

Overall, the project achieved its objectives and strengthened understanding of Java programming and interface design. This project successfully demonstrates:

- GUI development using Java Swing.
- Application of OOP principles.
- Game logic implementation.
- Basic AI behavior.
- Modular software design.

- **Future improvements:**
  - Smarter AI with several levels (Easy, Medium and Hard).
  - Scoreboard and memory for previous games and high scores.
  - Reset and restart buttons.
  - Animations, sounds and themes.
  - Multiplayer mode.

7. References

GridLayout (Java Platform SE 8) (no date). Available at: https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html (Accessed: November 23, 2025).

JButton (Java Platform SE 8) (no date). Available at: https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html (Accessed: November 23, 2025).

JFrame (Java Platform SE 8) (no date). Available at: https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html (Accessed: November 23, 2025).

JOptionPane (Java Platform SE 8) (no date). Available at: https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html (Accessed: November 23, 2025).

Technologies, C. (2024) "Building a Tic-Tac-Toe Game in Java," *Medium*, 24 August. Available at: https://configr.medium.com/building-a-tic-tac-toe-game-in-java-d0853e20a04e (Accessed: November 23, 2025).

Tic-Tac-Toe Game in Java (15:34:58+00:00) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/java/tic-tac-toe-game-in-java/ (Accessed: November 23, 2025).

Wilkinson, D. (2021) "Object-Oriented Principles Explained," *CodeX*, 18 May. Available at: https://medium.com/codex/object-oriented-principles-explained-2d1d4bdd3be7 (Accessed: November 23, 2025).