Assignment:

Implementing Tic Tac Toe Game Using Java graphics

Instructions:

In this assignment, we will be implementing a Tic Tac Toe game using Java GUI. The game should have the following features:

1. A 3x3 grid of buttons representing the game board
2. Two players represented by 'X' and 'O'
3. A label indicating whose turn it is
4. A message dialog indicating the winner or a draw
5. An option to restart or exit the game

Description:

Step 1: Create a new Java project in wer favorite IDE (Integrated Development Environment).

Step 2: Create a new Java class called "TicTacToeGUI" that extends the JFrame class and implements the ActionListener interface.

Step 3: Declare the following variables in the class:

- A two-dimensional array of JButtons representing the game board
- A JLabel indicating whose turn it is
- A char representing the current player ('X' or 'O')
- A JFrame for displaying the game over message dialog

Step 4: Create a constructor for the class that does the following:

- Sets the title of the frame to "Tic Tac Toe"
- Sets the default close operation to exit on close
- Sets the size of the frame to 600x600

- Creates a JPanel with a GridLawet of 3x3 to represent the game board

- Creates a JButton for each cell of the game board and adds it to the panel

- Sets the font of each JButton to Sans Serif, bold, and size 100

- Adds the panel and the JLabel to the frame

- Sets the location of the frame to the center of the screen

- Sets the frame visible

Step 5: Implement the actionPerformed method of the ActionListener interface. The method should do the following:

- Get the source of the event (the JButton that was clicked)

- Check if the button is empty

- If the button is empty, set the text of the button to the current player ('X' or 'O')

- Check if the game is over (i.e., if a player has won or if the game is a draw)

- If the game is over, display a message dialog indicating the winner or a draw and give the user the option to restart or exit the game

- If the game is not over, switch to the next player and update the JLabel indicating whose turn it is

Step 6: Implement the checkForWinner method that checks if a player has won the game. The method should do the following:

- Check for horizontal wins

- Check for vertical wins

- Check for diagonal wins

Step 7: Implement the checkForDraw method that checks if the game is a draw. The method should do the following:

- Check if all cells of the game board are occupied

Step 8: Implement the resetGame method that resets the game board and the JLabel indicating whose turn it is. The method should do the following:

- Set the text of each JButton to empty

- Set the current player to 'X'

- Update the JLabel indicating whose turn it is

Step 9: Implement the gameOver method that displays a message dialog indicating the winner or a draw and gives the user the option to restart or exit the game. The method should do the following:

- Create an array of options for the message dialog ('Restart' and 'Exit')

- Display the message dialog with the winner or a draw message and the options array

- If the user selects 'Restart', dispose of the JFrame and call the resetGame method

- If the user selects 'Exit', exit the program

Implementation:

Here is the implementation of the Tic Tac Toe game using Java GUI:

```
package tictactoe_game;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TicTacToeGUI extends JFrame implements ActionListener {
    private JButton[][] cells = new JButton[3][3];
    private JLabel statusLabel = new JLabel("X's turn");
    private char currentPlayer = 'X';
    JFrame frame = new JFrame();
    public TicTacToeGUI() {
        super("Tic Tac Toe");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 600);
        JPanel gridPanel = new JPanel(new GridLawet(3, 3));
        for (int row = 0; row < 3; row++) {
            for (int col = 0; col < 3; col++) {
```

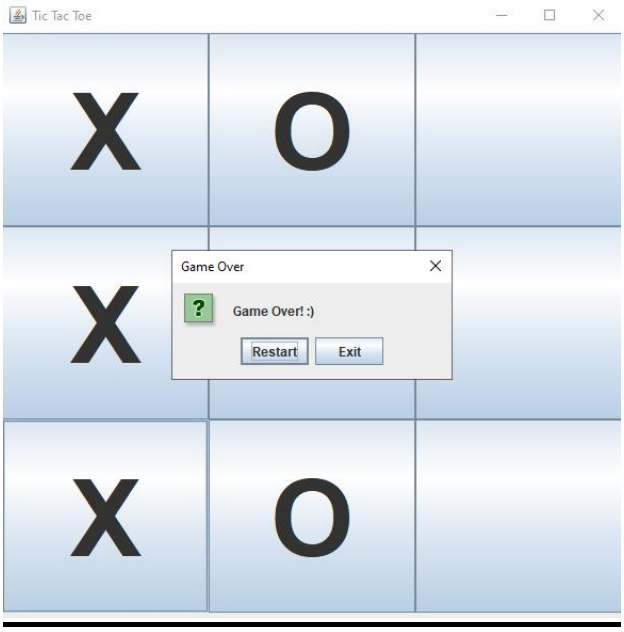
```
            cells[row][col] = new JButton();
            cells[row][col].addActionListener(this);
            cells[row][col].setFont(new Font(Font.SANS_SERIF, Font.BOLD, 100));
            gridPanel.add(cells[row][col]);
        }
    }
    add(gridPanel, BorderLawet.CENTER);
    add(statusLabel, BorderLawet.SOUTH);
    setLocationRelativeTo(null);
    setVisible(true);
}
public void actionPerformed(ActionEvent e) {
    JButton cell = (JButton) e.getSource();
    if (cell.getText().equals("")) {
        cell.setText(Character.toString(currentPlayer));
        if (checkForWinner()) {
            JOptionPane.showMessageDialog(this, currentPlayer + " wins!");
            gameOver();
        } else if (checkForDraw()) {
            JOptionPane.showMessageDialog(this, "Draw!");
            resetGame();
        } else {
            currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
            statusLabel.setText(currentPlayer + "'s turn");
        }
    } else {
        JOptionPane.showMessageDialog(this, "Invalid move!");
    }
}
private boolean checkForWinner() {
```

```
    for (int row = 0; row < 3; row++) {
        if (cells[row][0].getText().equals(cells[row][1].getText()) &&
            cells[row][1].getText().equals(cells[row][2].getText()) &&
            !cells[row][0].getText().equals("")) {
            return true;
        }
    }
    for (int col = 0; col < 3; col++) {
        if (cells[0][col].getText().equals(cells[1][col].getText()) &&
            cells[1][col].getText().equals(cells[2][col].getText()) &&
            !cells[0][col].getText().equals("")) {
            return true;
        }
    }
    if (cells[0][0].getText().equals(cells[1][1].getText()) &&
        cells[1][1].getText().equals(cells[2][2].getText()) &&
        !cells[0][0].getText().equals("")) {
        return true;
    }
    if (cells[0][2].getText().equals(cells[1][1].getText()) &&
        cells[1][1].getText().equals(cells[2][0].getText()) &&
        !cells[0][2].getText().equals("")) {
        return true;
    }
    return false;
}
private boolean checkForDraw() {
    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 3; col++) {
            if (cells[row][col].getText().equals("")) {
```

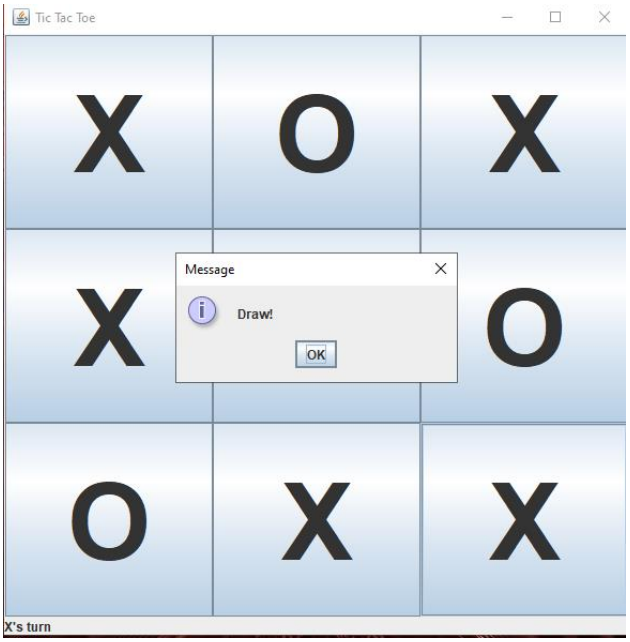
```
                return false;
            }
        }
    }
    return true;
}
private void resetGame() {
    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 3; col++) {
            cells[row][col].setText("");
        }
    }
    currentPlayer = 'X';
    statusLabel.setText("X's turn");
}
private void gameOver() {
    Object[] options = {"Restart", "Exit"};
    int choice = JOptionPane.showOptionDialog(frame, currentPlayer + " wins!", "Game Over",
JOptionPane.YES_NO_OPTION, JOptionPane.PLAIN_MESSAGE, null, options, options[0]);
    if (choice == JOptionPane.YES_OPTION) {
        dispose();
        new TicTacToeGUI();
    } else {
        System.exit(0);
    }
}

public static void main(String[] args) {
    new TicTacToeGUI();
}}
```

Output:



Sample 1



Sample 2

Challenge:

One of the challenges in implementing the Tic Tac Toe game using Java GUI is to handle user input correctly. In particular, we need to ensure that the user cannot make an invalid move (e.g., clicking on a cell that has already been occupied). To handle this, we can check whether the cell is empty before allowing the user to make a move.

Conclusion:

In this assignment, we have learned how to implement a Tic Tac Toe game using Java GUI. We have seen how to create a 3x3 grid of buttons, switch between two players, and display a message dialog indicating the winner or a draw. Additionally, we have learned how to handle user input and reset or exit the game upon completion. By completing this assignment, we have gained valuable experience in Java programming and GUI development that will be useful in future projects.