Report On

Tic Tac Toe Game

Submitted in partial fulfillment of the requirements of the Course project in Semester III Of Second YearArtificial Intelligence and Data Science

by
Saurabh Mane (Roll No. 25)
Vaibhav Narute (Roll No. 33)

Supervisor
Ms. Sejal D'mello

University of Mumbai

Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

(2023-24)

Vidyavardhini's College of Engineering & Technology
Department of Artificial Intelligence and Data Science

CERTIFICATE

This is to certify that the project entitled “Tic Tac Toe Game” is a bonafide work of " Saurabh Mane (Roll No. 25), Vaibhav Narute (Roll No. 33) submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester III of Second Year Artificial Intelligence and Data Science engineering.

Supervisor

Ms. Sejal D’mello

Dr. Tatwadarshi P. N.
Head of Department

1. ABSTRACT

This project presents the design and implementation of a classic Tic Tac Toe game using the Java programming language and the Eclipse Integrated Development Environment (IDE). The objective of this project is to create an interactive and user-friendly application that allows two players to engage in a game of Tic Tac Toe. The project involves the use of object-oriented programming principles, graphical user interfaces (GUI), and event handling in Java.

The implementation of the game includes a GUI that displays the game board, which is a 3x3 grid, and allows players to take turns placing their respective X and O markers. The application provides real-time feedback on the game's progress, including determining the winner or a draw condition. To enhance the user experience, the game also includes features such as player input validation, a reset option for starting new games, and a clear display of the game's outcome.

The use of Eclipse IDE streamlines the development process by providing a robust integrated development environment for writing, debugging, and testing the Java code. The project leverages Java's Swing library for creating the graphical user interface, allowing for a visually appealing and interactive game experience.

This Tic Tac Toe game project serves as a practical example of how to develop a simple yet engaging application in Java, making it accessible for both beginner and intermediate programmers. It also demonstrates the power of Eclipse IDE in facilitating Java application development, making it easier for developers to create applications with graphical interfaces.

Table of Contents

Pg. No

Problem Statement

The traditional Tic Tac Toe game, a two-player board game, provides an excellent platform for practicing and demonstrating various aspects of Java programming, including object-oriented design, graphical user interface (GUI) development, and event handling. The problem at hand is to design and implement a Tic Tac Toe game using the Java programming language in the Eclipse Integrated Development Environment (IDE).

Game Logic and User Interaction:

Develop a Java program that can simulate a two-player Tic Tac Toe game, allowing users to take turns to place their markers (X and O) on a 3x3 grid.

Graphical User Interface (GUI):

Create an intuitive and visually appealing GUI using Java's Swing library, ensuring that players can interact with the game board seamlessly. The GUI should display the current game state, player information, and the game outcome.

Event Handling:

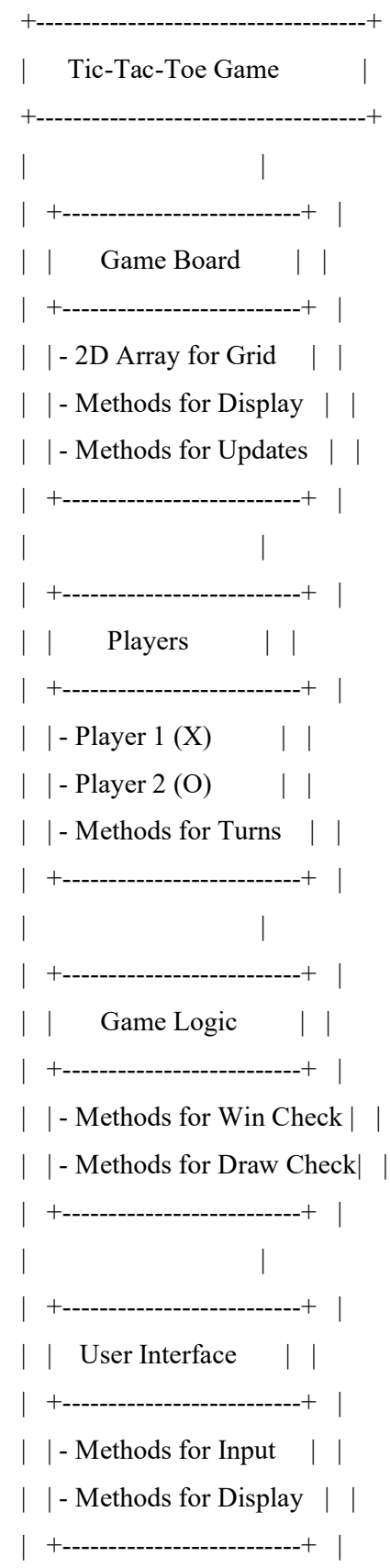
Implement event handling for user interactions, such as mouse clicks to mark their moves on the grid. Additionally, provide event-driven mechanisms to detect game-winning conditions or a draw.

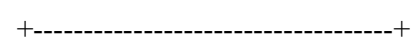
Validation and Error Handling:

Ensure that the program validates user inputs and handles potential errors gracefully, such as preventing invalid moves or notifying players of the game's outcome.

Block diagram, its description and working

BLOCK DIAGRAM :





Description of Components:

1. Tic-Tac-Toe Game:
   - Orchestrates the game.
   - Manages the game flow and interactions between components.

2. Game Board:
   - Contains the 2D array/grid representing the game board.
   - Provides methods for displaying and updating the board.

3. Players:
   - Represents Player 1 (X) and Player 2 (O).
   - Manages player turns and moves.

4. Game Logic:
   - Handles win-check and draw-check.
   - Ensures the game continues until there's a winner or a draw.

5. User Interface:
   - Deals with user interactions and displays.
   - Manages user input and game state display.

Working of the Tic-Tac-Toe Game:

1. Start the game by initializing the game board, players, and game logic.

2. Enter the game loop, taking turns between Player 1 and Player 2. The user interface displays the current board, and players make moves by selecting grid positions.

3. After each move, the game logic checks for a win. If a player wins, the game ends, displaying the winner. If there's a draw, the game ends with a draw message.

4. If no win or draw is detected, the game continues with the next player's turn.

5. The game loop repeats until a win or draw condition is met, concluding the game.

Feel free to copy and use this description and working in your Java Tic-Tac-Toe game project.

Module Description

Brief description of software & hardware used and its programming

Software:

Java: Java is a widely-used programming language that provides a platform-independent and object-oriented approach to software development. It is known for its portability, security, and extensive libraries.

Eclipse Integrated Development Environment (IDE): Eclipse is a popular open-source IDE used for Java development. It offers features such as code editing, debugging, and project management. Eclipse also supports various plugins and extensions for different programming languages and tools.

Hardware:
The hardware used for Java programming with Eclipse can vary depending on the specific requirements of your project. In general, you will need a computer with the following minimum specifications:

Processor: A modern multi-core processor (e.g., Intel Core i5 or AMD Ryzen series) for faster compilation and execution.

RAM: At least 4GB of RAM, though 8GB or more is recommended for smoother performance when working on larger projects.

Storage: Sufficient storage space for the operating system, Eclipse IDE, and your project files. A solid-state drive (SSD) can improve overall system responsiveness.

Display: A monitor with a comfortable screen size and resolution for coding and debugging.

Programming:
Java programming in Eclipse typically involves the following steps:

Installation: Install Java Development Kit (JDK) on your system. Eclipse requires a compatible JDK for Java development.

Eclipse Setup: Download and install the Eclipse IDE for Java developers. Configure the IDE as needed.

Project Creation: Create a new Java project in Eclipse, which serves as the container for your source code files.

Writing Code: Write and edit Java source code in Eclipse using its code editor. Eclipse provides features like code completion, syntax highlighting, and error checking to aid your development.

Building: Use Eclipse's build tools to compile your Java code into bytecode.

Debugging: Eclipse offers debugging tools to identify and fix issues in your code. You can set breakpoints, inspect variables, and step through your code line by line.

Running: You can run your Java applications from within Eclipse, and it provides various run configurations

for different types of applications.

Testing: You can integrate testing frameworks like JUnit within Eclipse for automated testing of your code.

Version Control: Eclipse supports version control systems like Git, allowing you to manage and collaborate on your projects more effectively.

Deployment: Once your Java application is ready, you can package it for deployment or distribution. Eclipse offers options for exporting your projects into JAR files or other executable formats.

Overall, Eclipse with Java is a powerful combination for developing Java applications, and it provides a comprehensive environment for writing, testing, and debugging code

Code

```
//Main Class

public class Main {

 public static void main(String[] args) {

        TicTacToe tictactoe = new TicTacToe();

  }
 }


//Tic Tac Toe Class

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

public class TicTacToe implements ActionListener{

    Random random = new Random();
    JFrame frame = new JFrame();
    JPanel title_panel = new JPanel();
    JPanel button_panel = new JPanel();
    JLabel textfield = new JLabel();
    JButton[] buttons = new JButton[9];
    boolean player1_turn;

    TicTacToe(){

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800,800);
        frame.getContentPane().setBackground(new Color(50,50,50));
        frame.setLayout(new BorderLayout());
        frame.setVisible(true);

        textfield.setBackground(new Color(25,25,25));
        textfield.setForeground(new Color(25,255,0));
        textfield.setFont(new Font("Ink Free",Font.BOLD,75));
        textfield.setHorizontalAlignment(JLabel.CENTER);
        textfield.setText("Tic-Tac-Toe");
        textfield.setOpaque(true);

        title_panel.setLayout(new BorderLayout());
        title_panel.setBounds(0,0,800,100);

        button_panel.setLayout(new GridLayout(3,3));
        button_panel.setBackground(new Color(150,150,150));

        for(int i=0;i<9;i++) {
            buttons[i] = new JButton();
            button_panel.add(buttons[i]);
            buttons[i].setFont(new Font("MV Boli",Font.BOLD,120));
            buttons[i].setFocusable(false);
            buttons[i].addActionListener(this);
        }

        title_panel.add(textfield);
```

```
		frame.add(title_panel,BorderLayout.NORTH);
		frame.add(button_panel);

		firstTurn();
	}

	@Override
	public void actionPerformed(ActionEvent e) {

		for(int i=0;i<9;i++) {
			if(e.getSource()==buttons[i]) {
				if(player1_turn) {
					if(buttons[i].getText()=="") {
						buttons[i].setForeground(new Color(255,0,0));
						buttons[i].setText("X");
						player1_turn=false;
						textfield.setText("O turn");
						check();
					}
				}
				else {
					if(buttons[i].getText()=="") {
						buttons[i].setForeground(new Color(0,0,255));
						buttons[i].setText("O");
						player1_turn=true;
						textfield.setText("X turn");
						check();
					}
				}
			}
		}
	}

	public void firstTurn() {

		try {
			Thread.sleep(2000);
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		if(random.nextInt(2)==0) {
			player1_turn=true;
			textfield.setText("X turn");
		}
		else {
			player1_turn=false;
			textfield.setText("O turn");
		}
	}

	public void check() {
		//check X win conditions
		if(
				(buttons[0].getText()=="X") &&
				(buttons[1].getText()=="X") &&
				(buttons[2].getText()=="X")
				) {
			xWins(0,1,2);
		}
		if(
```

```
                (buttons[3].getText()=="X") &&
                (buttons[4].getText()=="X") &&
                (buttons[5].getText()=="X")
                ) {
        xWins(3,4,5);
}
if(
                (buttons[6].getText()=="X") &&
                (buttons[7].getText()=="X") &&
                (buttons[8].getText()=="X")
                ) {
        xWins(6,7,8);
}
if(
                (buttons[0].getText()=="X") &&
                (buttons[3].getText()=="X") &&
                (buttons[6].getText()=="X")
                ) {
        xWins(0,3,6);
}
if(
                (buttons[1].getText()=="X") &&
                (buttons[4].getText()=="X") &&
                (buttons[7].getText()=="X")
                ) {
        xWins(1,4,7);
}
if(
                (buttons[2].getText()=="X") &&
                (buttons[5].getText()=="X") &&
                (buttons[8].getText()=="X")
                ) {
        xWins(2,5,8);
}
if(
                (buttons[0].getText()=="X") &&
                (buttons[4].getText()=="X") &&
                (buttons[8].getText()=="X")
                ) {
        xWins(0,4,8);
}
if(
                (buttons[2].getText()=="X") &&
                (buttons[4].getText()=="X") &&
                (buttons[6].getText()=="X")
                ) {
        xWins(2,4,6);
}
//check O win conditions
if(
                (buttons[0].getText()=="O") &&
                (buttons[1].getText()=="O") &&
                (buttons[2].getText()=="O")
                ) {
        oWins(0,1,2);
}
if(
                (buttons[3].getText()=="O") &&
                (buttons[4].getText()=="O") &&
                (buttons[5].getText()=="O")
                ) {
        oWins(3,4,5);
```

```
		}
		if(
				(buttons[6].getText()=="O") &&
				(buttons[7].getText()=="O") &&
				(buttons[8].getText()=="O")
				) {
			oWins(6,7,8);
		}
		if(
				(buttons[0].getText()=="O") &&
				(buttons[3].getText()=="O") &&
				(buttons[6].getText()=="O")
				) {
			oWins(0,3,6);
		}
		if(
				(buttons[1].getText()=="O") &&
				(buttons[4].getText()=="O") &&
				(buttons[7].getText()=="O")
				) {
			oWins(1,4,7);
		}
		if(
				(buttons[2].getText()=="O") &&
				(buttons[5].getText()=="O") &&
				(buttons[8].getText()=="O")
				) {
			oWins(2,5,8);
		}
		if(
				(buttons[0].getText()=="O") &&
				(buttons[4].getText()=="O") &&
				(buttons[8].getText()=="O")
				) {
			oWins(0,4,8);
		}
		if(
				(buttons[2].getText()=="O") &&
				(buttons[4].getText()=="O") &&
				(buttons[6].getText()=="O")
				) {
			oWins(2,4,6);
		}
	}

	public void xWins(int a,int b,int c) {
		buttons[a].setBackground(Color.GREEN);
		buttons[b].setBackground(Color.GREEN);
		buttons[c].setBackground(Color.GREEN);

		for(int i=0;i<9;i++) {
			buttons[i].setEnabled(false);
		}
		textfield.setText("X wins");
	}
	public void oWins(int a,int b,int c) {
		buttons[a].setBackground(Color.GREEN);
		buttons[b].setBackground(Color.GREEN);
		buttons[c].setBackground(Color.GREEN);

		for(int i=0;i<9;i++) {
			buttons[i].setEnabled(false);
```

```
        }

        textfield.setText("O wins");

    }
```

Results and conclusion

The combination of Java programming with the Eclipse Integrated Development Environment (IDE) is a highly effective and widely used approach for software development. Java, as a programming language, offers portability, security, and a rich set of libraries, making it suitable for a wide range of applications. Eclipse, as an IDE, provides a comprehensive set of features for Java development, including code editing, debugging, project management, and support for various plugins and extensions.

Hardware requirements for Java programming with Eclipse typically include a modern multi-core processor, a sufficient amount of RAM (4GB minimum, but 8GB or more recommended for larger projects), adequate storage space, and a comfortable display for coding and debugging.

The programming process with Java and Eclipse involves steps such as installation, project creation, writing and editing code, building, debugging, running, testing, version control, and deployment. These integrated tools and features make Java development in Eclipse efficient and productive.

Conclusion:

Java programming with Eclipse is a robust and popular combination for software development. The key advantages of this approach include:

Platform Independence: Java's "write once, run anywhere" capability allows developers to create applications that work on different platforms without modification.

Extensive Libraries: Java offers a vast array of libraries and frameworks, making it easier to develop a wide range of applications, from desktop to web and mobile.

Productivity: Eclipse IDE streamlines the development process with features like code completion, syntax highlighting, and debugging tools, enhancing developer productivity.

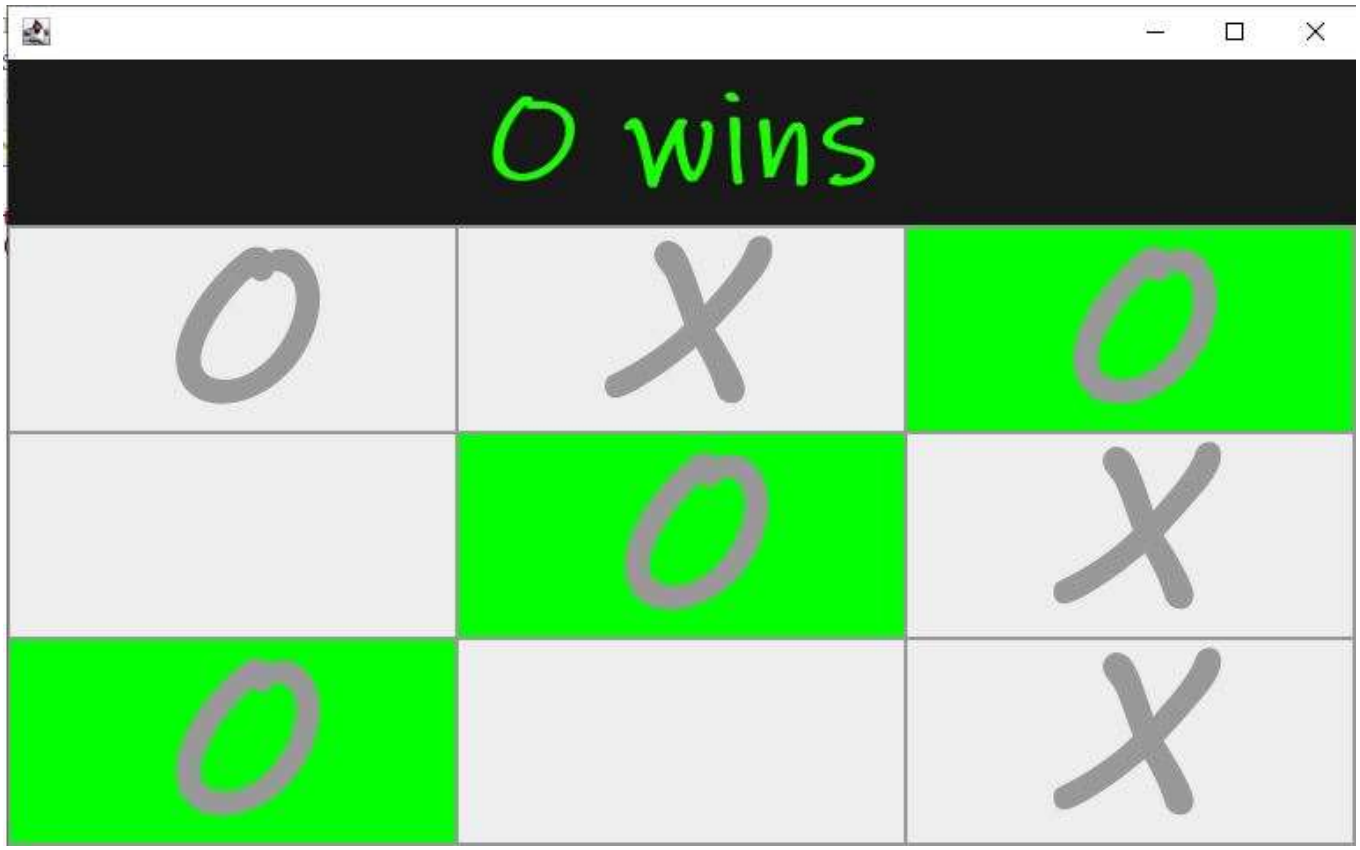
Collaboration: Eclipse's support for version control systems like Git facilitates collaboration among team members and helps manage code repositories efficiently.

Testing: Integration with testing frameworks like JUnit ensures code quality and reliability.

Deployment: Eclipse provides options for packaging and deploying Java applications in various formats.

In summary, Java programming with Eclipse is a powerful and versatile combination for software development, offering developers a comprehensive environment for building, testing, and deploying Java applications. It's a preferred choice for many developers and organizations due to its ease of use, robustness, and extensive ecosystem.

OUTPUT:



References

[1] Video: "Java Game Programming - Develop a “ Tic Tac Toe Game."
[2] https://www.youtube.com/watch?v=rA7tfvpkw0I Last Accessed: 10th October 2023.