SINGLE PLAYER TIC TAC TOE

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

VIKRAM KALLAKRINDA
[RA2111003010945]
YAMINI PRASANNA REDDY
[RA2111003010944]
R GOPIKRISHNAN
[RA2111003010960]

Under the guidance of

Mrs. DIVYA MOHAN

Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"SINGLE PLAYER TIC TAC TOE"** is the bona fide work of VIKRAM KALLAKRINDA (RA2111003010945), R GOPIKRISHNAN (RA2111003010960), YAMINI PRASANNA REDDY (RA2111003010944) who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mrs. DIVYA MOHAN
GUIDE
Assistant Professor
Department of Computing Technologies

SIGNATURE

Dr. M. Pushpalatha
HEAD OF THE DEPARTMENT
Professor & Head
Department of Computing Technologies

ABSTRACT

This project explores the implementation of a single-player Tic Tac Toe game with an AI opponent using the Minimax algorithm. The game aims to provide an interactive experience where a human player can challenge an intelligent computer opponent. The project involves developing a graphical user interface (GUI) for the game, allowing the player to interact with the board through mouse clicks or touchscreen inputs. The AI opponent uses the Minimax algorithm to make optimal moves, ensuring a challenging gameplay experience. Integration of the Minimax algorithm to enable the AI opponent to make strategic decisions by simulating future game states was done. Designed the game primarily for a single-player experience against the AI. `Implementation of adjustable difficulty levels by modifying the depth of the Minimax search tree. Recording and displaying game outcomes (win, lose, or draw) to evaluate player performance. The project aims to demonstrate the effectiveness of the Minimax algorithm in creating an AI opponent capable of playing Tic Tac Toe at different difficulty levels. The GUI enhances the overall user experience, making the game accessible and enjoyable for players of all skill levels. In conclusion, this project serves as an educational tool for understanding AI concepts in game development, showcasing how algorithms like Minimax can be applied to classic board games to create challenging and engaging experiences.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Single Player Tic Tac Toe with project is a fascinating endeavor aimed at enhancing the classic game of Tic Tac Toe by integrating an AI opponent powered by the Minimax algorithm. Tic Tac Toe, known for its simplicity, gains a new level of complexity and challenge with the addition of this AI. Players can now engage in strategic battles against a computer opponent that analyzes the game state and selects the best possible moves, making each game a thrilling intellectual challenge. Using Java and JavaFX, this project creates a seamless and enjoyable user experience. The AI opponent's moves are calculated in real-time, providing players with a challenging and dynamic gaming experience. This project not only showcases the application of AI algorithms in game development but also demonstrates the potential for enhancing traditional games to provide a more engaging and stimulating experience for players. This project aims to deliver a user-friendly interface that enhances the overall gaming experience. The game's interface will be designed to be visually appealing and intuitive, allowing players to easily understand the game state and make their moves. Additionally, clear feedback will be provided to the player about the current game state, including whose turn it is and the outcome of the game, ensuring a smooth and enjoyable gameplay experience. Future enhancements for this project could include implementing different difficulty levels for the AI opponent, providing players with the option to customize the challenge level to suit their skill level. Another potential enhancement could be improving the user interface with animations and sound effects to make the game more visually appealing and engaging. Additionally, adding a multiplayer mode where players can compete against each other online or locally could further enhance the game's replay value and appeal to a wider audience.

In conclusion, the Single Player Tic Tac Toe with AI project is an exciting opportunity to explore AI concepts in the context of a classic game. By implementing the Minimax algorithm, players can enjoy a challenging game of Tic Tac Toe against an intelligent AI opponent. This project aims to deliver a high-quality gaming experience that is both engaging and enjoyable for players of all ages, showcasing the potential of AI algorithms to enhance traditional games and provide a more immersive gaming experience.

CHAPTER 2

LITERATURE SURVEY

Minimax Algorithm in Game Development:

The Minimax algorithm is a fundamental concept in game theory and artificial intelligence. It is commonly used in two-player games to determine the optimal move for a player, assuming that the opponent is also playing optimally.
In the context of Tic Tac Toe, the Minimax algorithm can be used to create an AI opponent that can analyze the game state and select the best possible move, making the game more challenging and engaging for players.

User Experience in Game Development:

Several studies have explored the use of AI techniques in Tic Tac Toe to create intelligent opponents. These studies often focus on algorithms like Minimax, Alpha-Beta Pruning, and Monte Carlo Tree Search to improve the AI's decision-making process.
Researchers have also investigated the use of machine learning algorithms, such as neural networks, to train AI agents to play Tic Tac Toe at a high level. User experience (UX) is a critical aspect of game development, as it directly impacts how players interact with and enjoy the game. Studies have shown that factors such as interface design, feedback mechanisms, and game difficulty can significantly influence the overall user experience.

Java and JavaFX for Game Development:

Java is a popular programming language for game development due to its platform independence and extensive libraries for graphics and user interface development.
JavaFX is a user interface toolkit for Java applications, providing developers with tools to create rich, interactive user interfaces for their games.

Enhancing Traditional Games with AI:

There is a growing trend in game development to enhance traditional games with AI capabilities to provide players with a more challenging and dynamic gaming experience.
Studies have shown that integrating AI opponents in traditional games can increase player engagement and satisfaction, leading to a more enjoyable gaming experience.

CHAPTER 3

3.1 SYSTEM ARCHITECTURE AND DESIGN

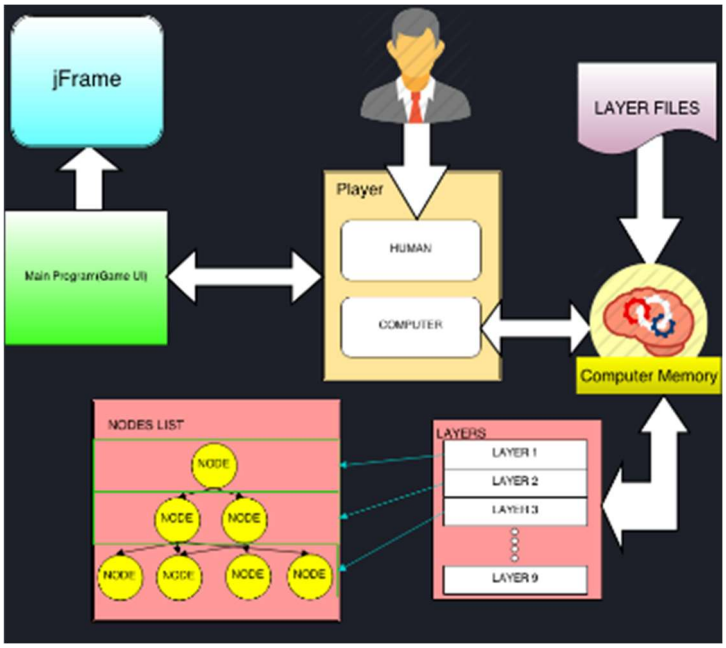

Fig. Architecture Model

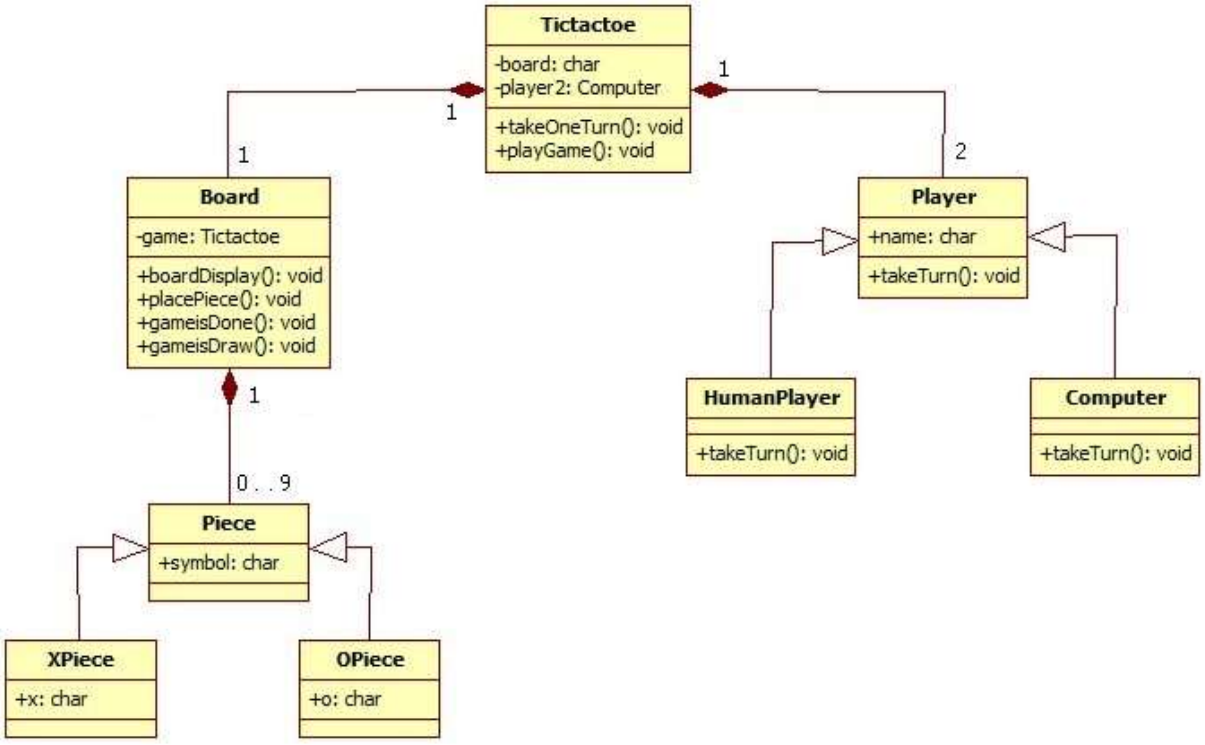

Fig. Class Diagram UML

3.2 Description of Module and components

The Single Player Tic Tac Toe with AI project consists of several modules and components that work together to create a seamless gaming experience. Here is a breakdown of the main modules and components used in the project:

Game Engine Module:

The Game Engine module is responsible for managing the game state, including the board state and player moves.

It contains the logic for validating moves, checking for a win or draw condition, and updating the board after each move.

This module also interfaces with the AI module to determine the AI opponent's moves.

AI Module:

The AI module implements the Minimax algorithm to calculate the optimal move for the AI opponent.

It analyzes the current game state to determine the best move that minimizes the maximum possible loss (hence the name "Minimax").

The AI module's output is a move recommendation that is passed back to the Game Engine module for execution.

User Interface Module:

The User Interface (UI) module is responsible for displaying the game board and user interface elements to the player.

It receives input from the player, such as clicking on a cell to make a move, and updates the UI to reflect the current game state.

This module also handles displaying messages to the player, such as whose turn it is and the outcome of the game.

Main Application Module:

The Main Application module is the entry point for the application and is responsible for initializing the game and managing the overall flow of the program.

It initializes the Game Engine, AI, and UI modules and coordinates their interactions.

This module also handles starting a new game, replaying a game, and exiting the application.

Additional Components:

The project may include additional components, such as classes for managing game settings (e.g., difficulty level) and game statistics (e.g., win/loss record).

Depending on the implementation, the project may also include classes for managing animations, sound effects, and other multimedia elements to enhance the user experience.

CHAPTER 4

METHODOLOGY

The project began with a thorough planning phase, where the requirements for the game were defined. This included determining the features of the game, such as the ability to play against an AI opponent using the Minimax algorithm, as well as the user interface design and overall game flow. The design phase involved creating mockups and wireframes to visualize the game's layout and functionality, ensuring that the final product would meet the project goals. The development phase of the project focused on implementing the planned features and design elements. This included writing the code for the game engine, AI module, user interface, and any additional components. Java and JavaFX were used for development, leveraging their capabilities for creating interactive user interfaces and implementing the Minimax algorithm for the AI opponent. The code was tested regularly to identify and fix any bugs or issues, ensuring that the game would function as intended. Once the game was developed, it underwent rigorous testing to ensure its functionality and stability. This involved testing various aspects of the game, such as the AI opponent's behavior, user interface interactions, and overall gameplay experience. Feedback from testers was used to refine the game, making improvements to enhance its playability and enjoyment. This iterative process of testing and refinement continued until the game met the project requirements and was ready for release. The project was developed using a modular approach, with distinct modules for the game engine, AI, user interface, and main application. This modular design allowed for easier debugging, maintenance, and future enhancements. was used for version control, allowing for easy tracking of changes and collaboration among team members. This ensured that changes could be easily managed and rolled back if needed, maintaining the integrity of the project. Continuous integration and deployment practices were followed to automate the build and deployment process. This allowed for faster and more efficient development cycles, ensuring that changes could be quickly tested and deployed. user testing was conducted throughout the development process to gather feedback on the game's usability and gameplay experience. This feedback was used to make iterative improvements to the game, ensuring that it met the needs and expectations of its target audience.

Comprehensive documentation was created for the project, including design documents, user manuals, and code documentation. This documentation provided a valuable resource for developers and users alike, helping to understand the game's functionality and how to use it effectively.

CHAPTER 5

CODING AND TESTING

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.ButtonType;
import javafx.scene.control.ButtonBar;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import java.util.Optional;
import java.util.Random;
import java.util.Stack;

public class TicTacToe extends Application {

    private Button[][] board;
    private char currentPlayer;
    private boolean singlePlayer;
    private char computerSymbol;
    private char playerSymbol;
    private Label turnLabel;

    private Stack<int[]> playerMoveStack = new Stack<>();
    private Stack<int[]> computerMoveStack = new Stack<>();

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Tic Tac Toe");

        Button singlePlayerButton = new Button("Single Player");
        singlePlayerButton.setStyle("-fx-font-size: 20px;");
        Button multiplayerButton = new Button("Multiplayer");
        multiplayerButton.setStyle("-fx-font-size: 20px;");

        VBox vbox = new VBox();
        vbox.setAlignment(Pos.CENTER);
        vbox.setSpacing(20); // Set vertical space between nodes

        Label titleLabel = new Label("Tic Tac Toe");
        titleLabel.setStyle("-fx-font-size: 36px;"); // Set font size for the heading
        vbox.getChildren().add(titleLabel);
```

```
        vbox.getChildren().addAll(singlePlayerButton, multiplayerButton);

        singlePlayerButton.setOnAction(event -> {
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Choose Difficulty");
            alert.setHeaderText("Select the difficulty level");
            alert.setContentText("Choose your option:");

            ButtonType beginnerButton = new ButtonType("Beginner");
            ButtonType advancedButton = new ButtonType("Advanced");
            ButtonType cancelButton = new ButtonType("Cancel",
ButtonBar.ButtonData.CANCEL_CLOSE);

            alert.getButtonTypes().setAll(beginnerButton, advancedButton, cancelButton);

            Optional<ButtonType> result = alert.showAndWait();
            if (result.isPresent() && result.get() == beginnerButton) {
                startGame(true, 'X'); // Single player, player is X
            } else if (result.isPresent() && result.get() == advancedButton) {
                startGame(true, 'O'); // Single player, player is O
            }
        });

        multiplayerButton.setOnAction(event -> startGame(false, 'X'));

        Scene scene = new Scene(vbox, 400, 400); // Increased width and height
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void startGame(boolean singlePlayer, char firstPlayer) {
        this.singlePlayer = singlePlayer;
        currentPlayer = firstPlayer;
        computerSymbol = (firstPlayer == 'X') ? 'O' : 'X';
        playerSymbol = (firstPlayer == 'X') ? 'O' : 'X'; // Set player symbol based on first
player
        board = new Button[3][3];

        BorderPane borderPane = new BorderPane();

        GridPane gridPane = new GridPane();
        gridPane.setAlignment(Pos.CENTER);
        gridPane.setVgap(10);
        gridPane.setHgap(10);

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                Button button = new Button();
                button.setPrefSize(100, 100);
                button.setStyle("-fx-font-size: 20px;");
```

```
                int finalI = i;
                int finalJ = j;
                button.setOnAction(event -> {
                    makeMove(finalI, finalJ);
                    animateButton(button);
                });
                board[i][j] = button;
                gridPane.add(button, j, i);
            }
        }

        turnLabel = new Label("Turn: " + currentPlayer);
        turnLabel.setStyle("-fx-font-size: 20px;");
        turnLabel.setAlignment(Pos.CENTER);

        HBox buttonBox = new HBox();
        buttonBox.setAlignment(Pos.CENTER);
        buttonBox.setSpacing(10);

        Button undoButton = new Button("Undo");
        undoButton.setOnAction(event -> undoMove());
        undoButton.setStyle("-fx-font-size: 16px;");

        Button resetButton = new Button("Reset");
        resetButton.setOnAction(event -> resetGame());
        resetButton.setStyle("-fx-font-size: 16px;");

        buttonBox.getChildren().addAll(undoButton, resetButton);

        borderPane.setTop(turnLabel);
        BorderPane.setAlignment(turnLabel, Pos.CENTER);
        borderPane.setCenter(gridPane);
        borderPane.setBottom(buttonBox);

        Scene scene = new Scene(borderPane, 400, 400);
        Stage stage = new Stage();
        stage.setScene(scene);
        stage.setTitle("Tic Tac Toe - Game");
        stage.show();

        if (singlePlayer && currentPlayer == computerSymbol) {
            makeComputerMove();
        }
    }

    private void makeMove(int row, int col) {
        if (board[row][col].getText().isEmpty()) {
            board[row][col].setText(String.valueOf(currentPlayer));
            if (currentPlayer == playerSymbol) {
                playerMoveStack.push(new int[]{row, col});
```

```
            } else {
                computerMoveStack.push(new int[]{row, col});
            }
            if (checkWin()) {
                endGame(currentPlayer + " wins!");
            } else if (isBoardFull()) {
                endGame("It's a tie!");
            } else {
                currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
                turnLabel.setText("Turn: " + currentPlayer);
                if (singlePlayer && currentPlayer == computerSymbol) {
                    makeComputerMove();
                }
            }
        }
    }

    private void undoMove() {
        if (!playerMoveStack.isEmpty()) {
            int[] playerLastMove = playerMoveStack.pop();
            board[playerLastMove[0]][playerLastMove[1]].setText("");
        }
        if (!computerMoveStack.isEmpty()) {
            int[] computerLastMove = computerMoveStack.pop();
            board[computerLastMove[0]][computerLastMove[1]].setText("");
        }
        currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
        turnLabel.setText("Turn: " + currentPlayer);
    }
    private void resetGame() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                board[i][j].setText("");
            }
        }
        currentPlayer = 'X'; // Reset to player X
        turnLabel.setText("Turn: " + currentPlayer);
    }
    private void makeComputerMove() {
        if (singlePlayer) {
            if (currentPlayer == computerSymbol) {
                // Beginner level: Random move
                Random random = new Random();
                int row, col;
                do {
                    row = random.nextInt(3);
                    col = random.nextInt(3);
                } while (!board[row][col].getText().isEmpty());
                board[row][col].setText(String.valueOf(currentPlayer));
            } else {
```

```
                int[] move = minimax(board, computerSymbol);
                board[move[0]][move[1]].setText(String.valueOf(computerSymbol));
                currentPlayer = playerSymbol;
            }
            if (checkWin()) {
                endGame(currentPlayer + " wins!");
            } else if (isBoardFull()) {
                endGame("It's a tie!");
            } else {
                currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
                turnLabel.setText("Turn: " + currentPlayer);
            }
        }
    }
    private int[] minimax(Button[][] board, char player) {
        int bestScore;
        int[] bestMove = new int[] { -1, -1 };

        if (checkWin()) {
            bestScore = (player == computerSymbol) ? 10 : -10;
        } else if (isBoardFull()) {
            bestScore = 0;
        } else {
            bestScore = (player == computerSymbol) ? Integer.MIN_VALUE : Integer.MAX_VALUE;

            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    if (board[i][j].getText().isEmpty()) {
                        board[i][j].setText(String.valueOf(player));
                        int score = minimax(board, (player == computerSymbol) ? playerSymbol
: computerSymbol)[0];
                        board[i][j].setText("");
                        if ((player == computerSymbol && score > bestScore)
                                || (player == playerSymbol && score < bestScore)) {
                            bestScore = score;
                            bestMove = new int[] { i, j };
                        }
                    }
                }
            }
        }

        return new int[] { bestScore, bestMove[0], bestMove[1] };
    }

    private boolean checkWin() {
        // Check for win logic
        for (int i = 0; i < 3; i++) {
            if (board[i][0].getText().equals(board[i][1].getText())
                    && board[i][1].getText().equals(board[i][2].getText())
```

```
                    && !board[i][0].getText().isEmpty()) {
                return true; // Check rows
            }
            if (board[0][i].getText().equals(board[1][i].getText())
                    && board[1][i].getText().equals(board[2][i].getText())
                    && !board[0][i].getText().isEmpty()) {
                return true; // Check columns
            }
        }
        if (board[0][0].getText().equals(board[1][1].getText())
                && board[1][1].getText().equals(board[2][2].getText())
                && !board[0][0].getText().isEmpty()) {
            return true; // Check diagonal from top-left to bottom-right
        }
        if (board[0][2].getText().equals(board[1][1].getText())
                && board[1][1].getText().equals(board[2][0].getText())
                && !board[0][2].getText().isEmpty()) {
            return true; // Check diagonal from top-right to bottom-left
        }
        return false;
    }

    private boolean isBoardFull() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j].getText().isEmpty()) {
                    return false; // If any button is empty, board is not full
                }
            }
        }
        return true; // All buttons are filled
    }

    private void endGame(String message) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Game Over");
        alert.setHeaderText(message);
        alert.showAndWait();
        Stage stage = (Stage) turnLabel.getScene().getWindow();
        stage.close();}

    private void animateButton(Button button) {
        button.setOpacity(0.5); // Set initial opacity
        button.setOnMousePressed(event -> button.setOpacity(0.5));
        button.setOnMouseReleased(event -> button.setOpacity(1.0));

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

CHAPTER 6

SCREENSHOTS AND RESULTS

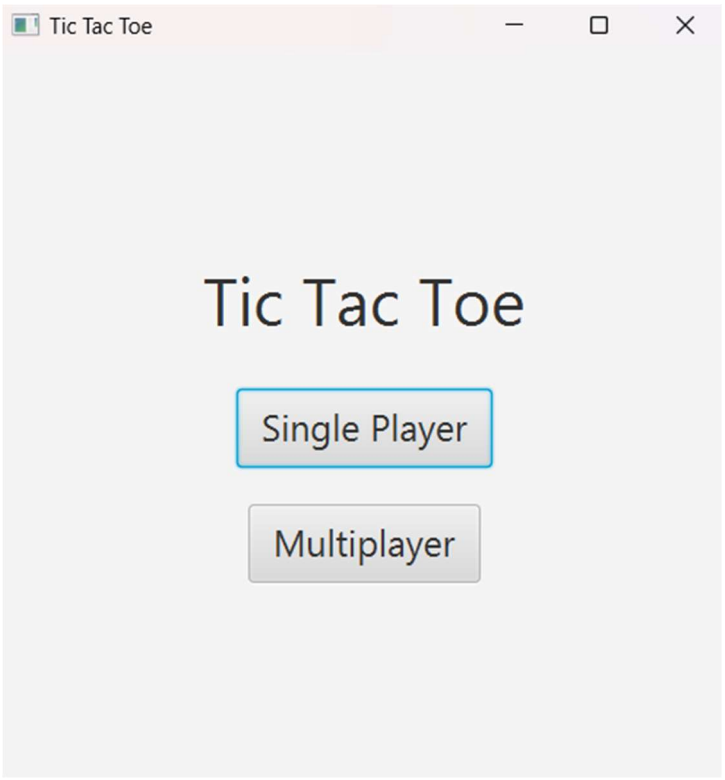

Fig. Opening Screen

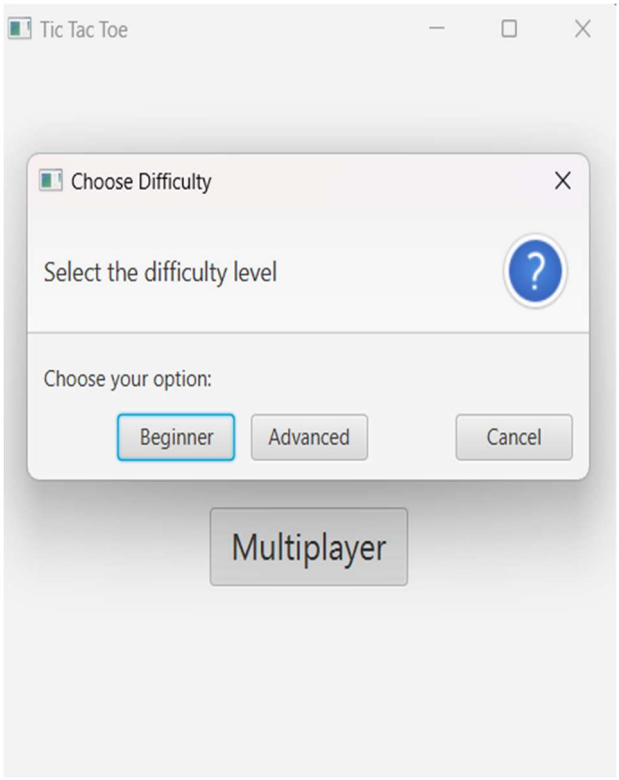

Fig. Single Player Screen

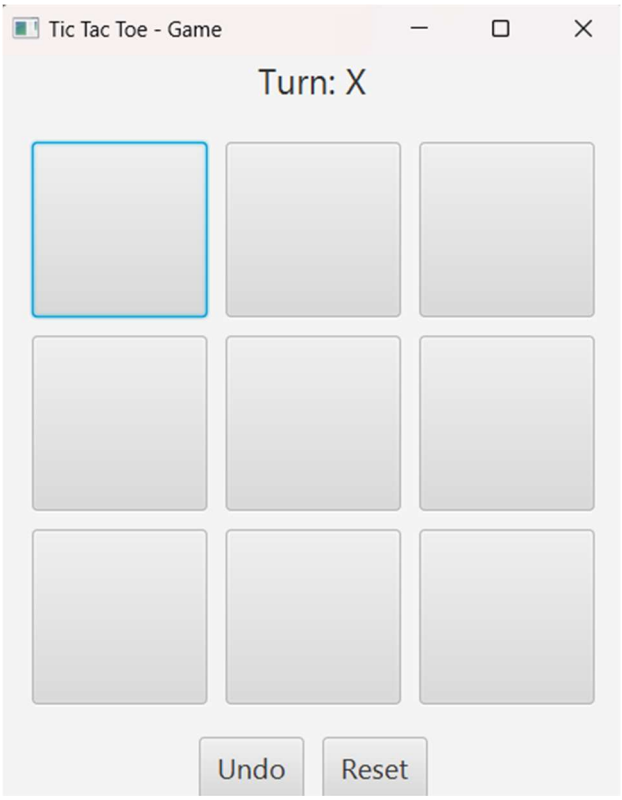

Fig. Game Board

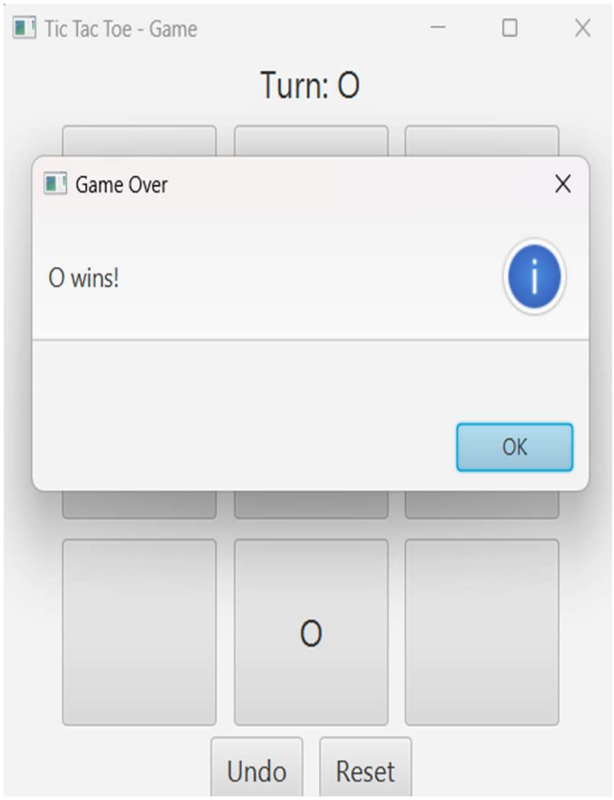

Fig. Game Result Window

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION :

In conclusion, the Single Player Tic Tac Toe with AI project has been a significant endeavor that has successfully integrated the Minimax algorithm to create a challenging and enjoyable gaming experience. The project aimed to enhance the classic game of Tic Tac Toe by providing players with the opportunity to play against an intelligent AI opponent. Through careful planning, design, and development, the project has achieved its goal of creating a game that offers strategic gameplay and a dynamic opponent. The use of Java and JavaFX allowed for the implementation of a user-friendly interface that enhances the overall gaming experience. The AI opponent's moves are calculated in real-time, providing players with a challenging and dynamic gaming experience. Additionally, the project followed best practices in software development, including modular design, version control, and continuous integration and deployment, ensuring a high-quality end product. Overall, the Single Player Tic Tac Toe with AI project has demonstrated the application of AI algorithms in game development and the potential for enhancing traditional games to provide a more engaging and stimulating experience for players. The project has provided valuable insights into the development process and the integration of AI in games, paving the way for future projects in this exciting field. Furthermore, the project's methodology emphasized user experience, leading to the creation of a visually appealing and intuitive interface. Clear feedback mechanisms were implemented to keep players informed about the game state, enhancing their overall engagement. The modular development approach facilitated efficient debugging, maintenance, and future enhancements, ensuring the project's scalability and adaptability. Through continuous testing and refinement, the project was able to deliver a high-quality gaming experience that met the project's goals and requirements. User feedback played a crucial role in shaping the game's development, highlighting the importance of user-centered design in creating successful gaming experiences.

In conclusion, the Single Player Tic Tac Toe with AI project has not only demonstrated the successful integration of the Minimax algorithm but also showcased best practices in software development and user experience design. The project has set a strong foundation for future projects in game development and AI integration, showcasing the potential for creating immersive and engaging gaming experiences.

7.2. FUTURE ENHANCEMENTS :

- **Improved AI Difficulty Levels:** Enhance the AI by adding different difficulty levels, allowing players to choose between easier and more challenging opponents. This could involve adjusting the depth of the Minimax algorithm's search or implementing additional strategies for the AI to consider.
- **Enhanced User Interface:** Improve the user interface by adding animations, sound effects, and visual cues to make the game more engaging and visually appealing. Consider adding customizable themes or graphics to allow players to personalize their gaming experience.
- **Multiplayer Mode:** Implement a multiplayer mode where players can compete against each other online or locally. This could involve adding networking capabilities to the game to allow players to connect and play against friends or other online opponents.
- **Additional Game Modes:** Introduce additional game modes to add variety and replay value. For example, a timed mode where players have a limited amount of time to make each move, or a "survival" mode where players must win multiple games in a row against increasingly difficult opponents.
- **AI Learning and Adaptation:** Implement machine learning algorithms to allow the AI to learn from its mistakes and adapt its strategy over time. This could create a more dynamic and challenging opponent that can provide a unique experience for each player.
- **Statistics and Leaderboards**: Add features to track player statistics, such as win-loss records and game completion times. Implement leaderboards to allow players to compare their performance with others and compete for high scores.
- **Cross-Platform Compatibility:** Ensure the game is compatible with multiple platforms, such as desktop, mobile, and web browsers, to reach a wider audience and allow players to enjoy the game on their preferred device.
- **Accessibility Features:** Include accessibility features such as customizable controls, color-blind mode, and text-to-speech capabilities to make the game more accessible to a wider range of players.

REFERENCES

1. Russell, S., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach. Pearson.
    - This textbook provides a comprehensive overview of artificial intelligence concepts, including the Minimax algorithm.
2. JavaFX Documentation: https://openjfx.io/javadoc/16/
    - Official documentation for JavaFX, which provides guidance on creating user interfaces and graphics for Java applications.
3. "Minimax Algorithm Explained" by Sebastian Lague: https://www.youtube.com/watch?v=l-hh51ncgDI
    - A video tutorial that explains the Minimax algorithm and how it can be used in game development.
4. "Tic Tac Toe AI with Minimax Algorithm" by Sebastian Lague: https://www.youtube.com/watch?v=trKjYdBASyQ
    - Another video tutorial that demonstrates how to implement the Minimax algorithm in a Tic Tac Toe game.
5. "Building Tic Tac Toe with JavaFX" by Marco Jakob: https://www.genuinecoder.com/javafx-tic-tac-toe-tutorial/
    - A tutorial on building a Tic Tac Toe game with JavaFX, which can provide insights into creating the user interface for your project.
6. "Creating a Simple Tic Tac Toe Game with Java" by Codota: https://www.codota.com/code/java/methods/javax.swing.JButton/setText
    - An example of how to create a simple Tic Tac Toe game in Java, which can serve as a reference for implementing game logic and user interactions.
7. "JavaFX Scene Builder" by Gluon: https://gluonhq.com/products/scene-builder/
    - Scene Builder is a visual layout tool for JavaFX applications, which can be helpful for designing the user interface for your game
8. CHATGPT.
9. Google and Bing AI.
10. STACK OVERFLOW website for the Architecture Diagrams.