University of Glasgow | School of Computing Science

Course : CSC 1103 Programming Methodology Mini-Project 2023

Title : Tic-Tac-Toe Mini-Project Report

Lab Session : P1

Group : 08

Name	GUID	SIT ID	Scope & Contribution
Ryan Oh Tian Guan	2957948O	2300916	• Extract tic-tac-toe dataset, randomise it and store it into training set and testing set • Training the AI based on dataset and predicting the best move • Naive Bayes Classifier AI
Elroy Lee	2957894L	2300950	• Basic Tic-tac-toe functions • Minimax Perfect • Naive Bayes Classifier Testing
Felix Chang	2957851C	2301105	• Implementation of header file and splitting of codebase • Two-Player Mode • Game GUI (RayLib)
Jiang Weimin	2957883J	2301083	• Restructure Codebase using structures & pointers • Minimax Imperfect • Game GUI (RayLib) • BlackBox Testing
Lim Jing Chuan Jonathan	2957906L	2300923	• Game GUI (RayLib), Visual & Auditory enhancement • BlackBox Testing • Gameplay comparison comments

Table of Contents

Tic-Tac-Toe Game Report

Abstract

The main purpose of this project is to develop a 3x3 Tic-Tac-Toe game in the C programming language. The game will be made for Internet of Things (IoT) tablets that have limited memory and processing power. The objectives of this game are to enhance motor skills, social skills, and the early stages of left brain development within young children.

Our game features a two-player mode and one-player mode. The two-player mode provides a platform for two kids to engage in interactive play. This fosters their social skills and motor skill development. The one-player mode enables a child to play against the AI computer. This helps to stimulate their initial stages of analytical thinking.

Additionally, to ensure accessibility and engagement, the game also features a user-friendly graphical user interface (GUI). The design and functionality of the Tic-Tac-Toe game offers an interactive and educational gaming experience, helping the developmental needs of children.

The end-product of this project will address the challenge of developing a 3x3 Tic-Tac-Toe game. Offering entertainment while contributing to the holistic development of children within IoT tablet constraints. The project holds potential as a valuable tool for early childhood education and skill enhancement.

How to run the program

NOTE: Our application was programmed, built and tested on Windows. The following instructions are based on the user being in a Windows platform.

Requirements:

- RayLib (Graphics library for C)
    - Our application uses RayLib for its GUI portion (https://github.com/raysan5/raylib)
    - To install on Windows:
        - Open MSYS2 (https://www.msys2.org/)
        - Run: **“pacman -S mingw-w64-x86_64-raylib”**
        - Alternatively, you can see https://github.com/raysan5/raylib/wiki/Working-on-Windows
    - To install on Mac: https://github.com/raysan5/raylib/wiki/Working-on-macOS

- A working C compiler (e.g. gcc for Windows)
    - You can install gcc using MSYS2 (https://www.msys2.org/)
        - Run: **“pacman -S mingw-w64-ucrt-x86_64-gcc”**

Building and Running the application:

1) Open a terminal
2) Build the app using: **“gcc -fcommon -o tictactoe tictactoeMain.c tictactoeGUI.c tictactoeLogic.c -lraylib -lopengl32 -lgdi32 -lwinmm”**
3) Run the generated exe file using: ./tictactoe

1. Problem Definition

The objective of this project is to implement a Tic-Tac-Toe game with various features, including a graphical user interface, a two-player mode, a single-player vs. AI mode, different AI difficulty levels and an AI that is based off machine learning (Naive Bayes Classifier).

The main challenge addressed of this project is the creation of a 3x3 Tic-Tac-Toe game using the C programming language specifically tailored for IoT tablets, which often operate with constraints such as limited memory and processing power. The primary motivation behind this endeavour is to provide an educational and entertaining platform for children, targeting the development of essential skills during their formative years.

1.1 Educational Objectives

- **Holistic & Motor Skill Child Development:**
    - Provide an interactive learning experience.
    - Develop a user-friendly GUI suitable for young children.
    - Enhancing hand-eye coordination in children through interactive gameplay.
- **Social Skill Facilitation:**
    - Foster cooperation, turn-taking, and communication in the two-player mode.
- **Analytical Thinking Skills:**
    - Stimulate analytical thinking through strategic decision-making against the AI in the single-player mode.

1.2 Challenges and Improvements

- **Memory and Processing Constraints:**
    - Optimise code for smooth gameplay within IoT tablet limitations.
- **User-Friendly GUI:**
    - Design an intuitive and visually appealing interface for young users to keep them engaged.
- **Interactive Learning Experience:**
    - Ensure the game contributes to skill development and education.
- **Accessibility:**
    - Cater to both individual engagement and social interaction scenarios.

2. Problem Analysis

2.1 User Engagement

- Problem: To enhance user engagement and create an intuitive interface.
- Solution: Implement an interactive and visually appealing GUI.

2.2 Game Logic

- Problem: To ensure accurate implementation of game logic for a fair Tic-Tac-Toe gameplay.
- Solution: Regularly test and validate the game logic for moves, win conditions, and turn transitions.

2.3 AI Performance

- Problem: To optimise AI algorithms for decision-making speed and difficulty levels.
- Solution: Optimise AI performance, implement different difficulty levels and verifying that the AI has made the correct decisions for a Tic-Tac-Toe game.

2.4 User Input Validation

- Problem: To handle invalid user inputs to prevent disruptions.
- Solution: Implement user input validation with clear error messages to notify the user of the invalid input.

2.5 Game State Management

- Problem: To efficiently manage game state transitions and restarts.
- Solution: Implement clear state transitions and ensure smooth game restarts.

2.6 Code Structure and Modularity

- Problem: To ensure a clear and maintainable code structure that is readable.
- Solution: Follow best coding practices, use meaningful names, and modularise code for better readability to other developers.

2.7 Scalability and Extensibility

- Problem: To enable scalability and extensibility for future enhancements.
- Solution: Design the codebase with modularity to allow for easy additions or modifications.

3. Tic-Tac-Toe

An overview of our Tic-Tac-Toe game program features.

3.1 Game Features

- **Single-player vs Perfect AI:**
    - Overview:
        - Utilises the Minimax algorithm for optimal decision-making.
        - Offers a challenging & strategic gameplay experience.
    - Gameplay:
        - Players face a flawless AI that makes the best possible moves.
        - Enhances analytical thinking & decision-making skills.
        - Impossible win, at most draw.
- **Single-player vs Imperfect AI:**
    - Overview:
        - Modified Minimax algorithm with depth limitations for imperfection.
        - Balances challenge & accessibility for varying skill levels.
    - Gameplay:
        - Provides a reasonably competitive AI with a forgiving edge.
        - Suitable for players seeking a less intense gaming session.
        - 2 Modes: Easy & Medium.
- **Single-player vs Naive Bayes AI:**
    - Overview:
        - AI's performance is dependent on the training data set
        - AI's algorithm is based on the predicted probability of each move on the board after every opponent's move
    - Gameplay:
        - Provides a reasonably competitive AI based on dataset, but difficulty is still manageable

- **Two-player mode:**
    - Game program provides an option for the user to play in Two-Player mode.
    - An offline/manual Two-Player mode where the players take turns in making their moves.
    - Player 1 (“X”) makes his turn first, followed by Player 2 (“O”). Player 1 (“X”) will always make the first move.
    - Players make their move by clicking on the tiles in the graphical interface of the game program.
    - Player who gets the 3-in-a-row pattern wins the game. Patterns such as any row, column or diagonal 3-in-a-row. A draw is also possible where there is no winner.
- **User-friendly graphical interface (Using RayLib):**
    - The game has an interactive graphical user interface with buttons and text rendered on the screen.
    - It includes a home screen with buttons for selecting game mode (two-player or vs AI) and an exit option.
    - There is a difficulty selection screen for choosing the AI level (easy, medium, hard).
    - The main game screen displays the Tic-Tac-Toe-board, results messages, and buttons for restarting or going back to the home screen.
    - The game incorporates various audio elements, including sound effects accompanying player actions and game restarts, as well as a background musical score.
    - Visual effect displaying a strike-through effect for the winning combination.
    - Scoreboard used to track and display the number of wins for Player1 (“X”), Player 2 (“O”), and draws. It then updates the scoreboard based on the outcome of each game.

3.2 AI Difficulty Levels

Perfect AI using Minimax algorithm

The Minimax Perfect AI is designed to play optimally and never lose. It operates on the principle of exploring all possible moves to determine the best one. In the context of your game:

- **Algorithm Depth:**
    - The minimax algorithm searches through the game tree to a certain depth, evaluating each possible move's outcome.
    - A higher depth allows the AI to look further ahead, making it more challenging for players to defeat.

- **Optimal Decision-Making:**
    - The AI, assuming the role of 'O,' systematically evaluates potential moves and chooses the one that leads to the highest likelihood of winning or a draw.

Imperfect AI with varying levels of difficulty using Minimax algorithm

The Minimax Imperfect AI introduces variability in decision-making, simulating an opponent with different skill levels.

- **Difficulty Levels:**
    - The AI has multiple difficulty levels, such as 'Easy' and 'Medium,' each affecting how deep the minimax algorithm explores the game tree.
    - 'Easy' might represent a less strategic opponent, while 'Medium' introduces more complexity.
- **Heuristic Evaluation:**
    - The Imperfect AI includes a heuristic evaluation at a certain depth, allowing it to make decisions based on a simplified assessment of the game state.
    - This heuristic evaluation can represent the AI's ability to make strategic decisions without exploring the entire game tree.
- **Adaptability:**
    - The Imperfect AI adapts to the player's skill level by adjusting the depth of the minimax search.
    - This adaptability provides a more engaging and balanced experience for players of varying skill levels.

Naive Bayes AI

- **Difficulty Levels:**
    - The AI’s difficulty is solely based on its training data. Since the training data has more positive outcomes than negative, the winning outcome is skewed towards the player rather than the AI
- **Adaptability:**
    - Based on each move that the player makes, the AI will calculate the predicted probability of positive and negative outcomes of the game by considering all possible moves on the board wherever it’s empty. After which, it will decide and play the move with the probability of a negative outcome being the highest.
    - However that being said, the training data set has a much higher probability of positive outcomes, thus making the AI much manageable to defeat

Overall Impact

- Perfect AI Challenge:
    - The Minimax Perfect AI serves as a challenging opponent, encouraging players to think strategically and plan their moves carefully.
- Imperfect AI Variety:
    - The Minimax Imperfect AI adds variety to the gaming experience, catering to a broader audience with different skill levels.
- Naive Bayes AI:
    - The performance of the Naive Bayes AI is similar to the lower levels of difficulty of an imperfect AI, in a sense that it isn’t as competitive. This allows it to cater to a wide range of audiences with different skill levels
- Educational Value:
    - All AI types contribute to the educational aspect of the game by stimulating analytical thinking and strategic decision-making, aligning with the project's educational objectives.

4. Pseudocode

Provide a high-level overview of the logic using pseudocode.

tictactoeMain.c

```
BEGIN
    initialiseNaiveBayes()
    initializeGame(&player.gamemodeChoice)
    UnloadSound(gameSounds.moveSound)
    UnloadSound(gameSounds.restartSound)
    UnloadSound(gameSounds.backgroundSound)
    CloseAudioDevice()
END

FUNCTION initialiseNaiveBayes()
    Example* allExamples
    numExamples ← readExamplesFromFile("tictactoe.data", &allExamples)
    IF (numExamples == 0)
        PRINT "Failed to read file or file is empty."
        RETURN 1
    ELSE
        PRINT "Successfully read " + numExamples + " examples from the file."
    END IF
    splitRatio ← 0.8
    splitDataset(allExamples, numExamples, &trainingSet, &numTraining, &testingSet,
    &numTesting, splitRatio)
    PRINT "Number of training examples: " + numTraining
    PRINT "Number of testing examples: " + numTesting
    PRINT "Total number of examples: " + numExamples
    IF (numTraining + numTesting == numExamples)
        PRINT "The splitExamples function works correctly."
    ELSE
        PRINT "The splitExamples function does not work correctly."
    END IF
    FOR (int k = 0 TO 8)
        FOR (int i = 0 TO sizeof(features) / sizeof(features[0]) - 1)
            FOR int j = 0 TO sizeof(targets) / sizeof(targets[0]) - 1
                counts[k][countIndex].feature ← features[i]
                counts[k][countIndex].target ← targets[j]
                counts[k][countIndex].count ← 0
                countIndex++
            END FOR
        END FOR
        countIndex ← 0
    END FOR
    FOR (int k = 0 TO 8)
        FOR (int i = 0 TO sizeof(features) / sizeof(features[0]) - 1)
            FOR (int j = 0 TO sizeof(targets) / sizeof(targets[0]) - 1)
                probability[k][countIndex].feature ← features[i]
                probability[k][countIndex].target ← targets[j]
                probability[k][countIndex].probability ← 0
                countIndex++
```

```
                END FOR
            END FOR
            countIndex ← 0
        END FOR
        train(trainingSet, numTraining, counts, label)
        learn(counts, probability)
        PRINT "Training Accuracy:"
        testingAccuracy(testingSet, numTesting, probability, countIndex)
        PRINT "Test Accuracy:"
        testingAccuracy(trainingSet, numTraining, probability, countIndex)
END FUNCTION
```

tictactoeGUI.c

```
FUNCTION initializeGame(gamemodeChoice)
        gamemodeChoice      refToInt → &gamemodeChoice
        InitAudioDevice()
        gameSounds.moveSound←LoadSound("...")
        gameSounds.restartSound←LoadSound("...")
        gameSounds.backgroundtSound←LoadSound("...")
        PlaySound(gameSounds.backgroundSound)
        player.whoWon ← 0
        gameLoop(*gamemodeChoice)
        CloseWindow()
END FUNCTION

FUNCTION restartGame()
        FOR (int i = 0; i < 9; i++)
            gameBoard.board[i]←0
        ENDFOR
        player.turn←1
        player.numOfTurns←0
        player.whoWon←0
        gameStatus.showInvalidMsg←0
        gameStatus.aiMadeMove←0
        gameStatus.scoreboardUpdated←false
END FUNCTION

FUNCTION gameLoop()
        InitWindow(screenProps.screenWidth, screenProps.screenHeight, "TIC TAC TOE")
        SetTargetFPS(60)
        RenderTexture2D screen ←
        LoadRenderTexture(screenProps.screenWidth,screenProps.screenHeight)
        RenderTexture2D X ←
LoadRenderTexture(screenProps.squareSize,screenProps.squareSize)
        RenderTexture2D O ←
LoadRenderTexture(screenProps.squareSize,screenProps.squareSize)
        volume ← 1.0f
        SetSoundVolume(gameSounds.backgroundSound, volume)
        WHILE (!WindowShouldClose())
            handleInput()
            Rectangle buttons[6] ←
                (Rectangle){250, 260, 300, 100},
                (Rectangle){250, 380, 300, 100},
                (Rectangle){250, 500, 300, 100},
                (Rectangle){250, 610, 300, 100},
                backButton,
```

```
        restartButton
IF (IsKeyPressed(KEY_M))
        IF (volume != 0.0f)
                volume ← 0.0f
        ELSE
                volume ← 1.0f
        END IF
        SetSoundVolume(gameSounds.backgroundSound, volume)
END IF
IF (!IsSoundPlaying(gameSounds.backgroundSound))
        PlaySound(gameSounds.backgroundSound)
END IF
SWITCH player.loopState
        CASE 0:
                pressedButton ← checkButton(buttons)
                IF ((pressedButton == 0 OR pressedButton == 1) AND
                IsMouseButtonPressed(0))
                        player.loopState ← (pressedButton == 0) ? 3 : 1
                        player.gamemodeChoice ← (pressedButton == 0) ? 2 : 1
                ELSE IF (pressedButton == 2 AND IsMouseButtonPressed(0))
                        player.gamemodeChoice ← 1
                        player.aiType ← 4
                        player.loopState ← 3
                ELSE IF (pressedButton == 3 AND IsMouseButtonPressed(0))
                        CloseWindow()
                        BREAK
                END IF
                BeginDrawing()
                BeginTextureMode(screen)
                ClearBackground(GRAY)
                DrawText("Tic-Tac-Toe", 100, 60, 90, BLACK)
                FOR (i FROM 0 TO 3)
                        DrawRectangleRounded(buttons[i], 0.2, 5, (pressedButton
                        == i) ? BLACK:DARKGRAY)
                        DrawRectangleRounded((Rectangle){buttons[i].x+10,
                        buttons[i].y+10, 280, 80}, 0.2, 5, WHITE)
                END FOR
                DrawText("Two Player", 275, 288, 45, (pressedButton == 0) ?
                BLACK:DARKGRAY)
                DrawText("Vs AI", 330, 408, 45, (pressedButton == 1) ?
                BLACK:DARKGRAY)
                DrawText("Vs AI (ML)", 290, 528, 45, (pressedButton == 2) ?
                BLACK:DARKGRAY)
                DrawText("Exit", 360, 648, 45, (pressedButton == 3) ?
                BLACK:DARKGRAY)
                EndTextureMode()
                DrawTextureRec(screen.texture, (Rectangle){0, 0,
                screenProps.screenWidth,   -screenProps.screenHeight},
                (Vector2){0, 0}, WHITE)
                EndDrawing()
                BREAK
        CASE 1:
                pressedButton ← checkButton(buttons)
                IF ((pressedButton == 0 OR pressedButton == 1 OR
                pressedButton == 2) && IsMouseButtonPressed(0))
                        player.aiType ← (pressedButton == 0) ? 3 : (pressedButton
                        == 1) ? 2 : 1
```

```
                player.loopState ← 3
            ELSE IF (pressedButton == 3 AND IsMouseButtonPressed(0))
                player.loopState ← 0
            END IF
            BeginDrawing()
            BeginTextureMode(screen)
            ClearBackground(GRAY)
            DrawText("Select Difficulty", 85, 100, 75, BLACK)
            FOR (i FROM 0 TO 3)
                DrawRectangleRounded(buttons[i], 0.2, 5, (pressedButton
                == i) ? BLACK:DARKGRAY)
                DrawRectangleRounded((Rectangle){buttons[i].x+10,
                buttons[i].y+10, 280, 80}, 0.2, 5, WHITE)
            END FOR
            DrawText("Easy", 345, 288, 45, (pressedButton == 0) ?
            BLACK:DARKGRAY)
            DrawText("Medium", 330, 408, 45, (pressedButton == 1) ?
            BLACK:DARKGRAY)
            DrawText("Hard", 350, 528, 45, (pressedButton == 2) ?
            BLACK:DARKGRAY)
            DrawText("Home", 350, 638, 45, (pressedButton == 3) ?
            BLACK:DARKGRAY)
            EndTextureMode()
            DrawTextureRec(screen.texture, (Rectangle){0, 0,
            screenProps.screenWidth, -screenProps.screenHeight},
            (Vector2){0, 0}, WHITE)
            EndDrawing()
            BREAK
        CASE 3:
            STATIC int wasInHomeMenu ← 0
            IF (wasInHomeMenu == 1)
                restartGame()
                wasInHomeMenu ← 0
                gameStatus.scoreboardUpdated ← false
                scoreboard.player1Wins ← 0
                scoreboard.player2Wins ← 0
                scoreboard.draws ← 0
            END IF
            drawGame()
            IF (IsMouseButtonPressed(MOUSE_LEFT_BUTTON))
                Vector2 mousePosition ← GetMousePosition()
                IF (CheckCollisionPointTriangle(mousePosition,
                    (Vector2){25, 55}, (Vector2){140, 55},
                    (Vector2){82.5, 15}))
                    player.loopState ← 0
                    wasInHomeMenu ← 1
                END IF
            END IF
            IF (!gameStatus.scoreboardUpdated AND (player.whoWon != 0
            OR isBoardFull(gameBoard.board))) THEN
                updateScoreboard(&player.whoWon)
                gameStatus.scoreboardUpdated ← true
            END IF
            BREAK
    END SWITCH
  END WHILE
END FUNCTION
```

```
FUNCTION handleInput() {
    IF (MouseButtonPressed(LEFT_MOUSE_BUTTON) && player.whoWon == &&
    player.loopState == 3) THEN
        mousePosition ← GetMousePosition()
        x ← (int)mousePosition.x
        y ← (int)mousePosition.y
        IF (x >= screenProps.boardX AND x <= (screenProps.boardX + 3 *
        screenProps.squareSize) AND y >= screenProps.boardY AND y <=
        (screenProps.boardY + 3 * screenProps.squareSize)) THEN
            col ← (x - screenProps.boardX) / screenProps.squareSize
            row ← (y - screenProps.boardY) / screenProps.squareSize
            choice ← row * 3 + col + 1
            IF (choice >= 1 AND choice <= 9 AND gameBoard.board[choice - 1] == 0)
            choice - ← 1 THEN
                insertChoice(&choice)
                PlaySound(gameSounds.moveSound)
            ELSE
                gameStatus.showInvalidMsg ← 1
            END IF
        END IF
    END IF
END FUNCTION

FUNCTION drawGame()
    InitiateDrawing()
    ClearBackground(GRAY)
    screenProps.boardX ← (screenProps.screenWidth - 3 * screenProps.squareSize) / 2
    screenProps.boardY ← (screenProps.screenHeight - 3 * screenProps.squareSize) / 2
    resultMessage ← " "
    winningCombo[3] ← { -1, -1, -1 }
    player.whoWon ← win(gameBoard.board)
    DrawTriangle((Vector2){25, 55}, (Vector2){140, 55}, (Vector2){82.5, 15}, WHITE)
    DrawText("BACK", 55, 35, 20, BLACK)
    DrawRectangleRounded(restartButton, 0.2, 5, WHITE)
    DrawText("RESTART", screenProps.screenWidth - 98, 35, 15, BLACK)
    IF (IsMouseButtonPressed(MOUSE_LEFT_BUTTON))
        Vector2 mousePosition = GetMousePosition()
        IF CheckCollisionPointTriangle(mousePosition,
            (Vector2){25, 55}, (Vector2){140, 55}, (Vector2){82.5, 15}) THEN
            player.loopState ← 0
            restartGame()
        END IF
    END IF
    IF (IsMouseButtonPressed(MOUSE_LEFT_BUTTON))
        Vector2 mousePosition = GetMousePosition()
        IF (CheckCollisionPointTriangle(mousePosition,
        (Vector2){25, 55}, (Vector2){140, 55}, (Vector2){82.5, 15}) THEN
            player.loopState ← 0
            restartGame()
        ELSE IF (CheckCollisionPointRec(mousePosition, restartButton))
            restartGame()
            PlaySound(gameSounds.restartSound)
        END IF
    END IF
    DrawText(TextFormat("Player 1 Wins: %d", scoreboard.player1Wins), 50, 650, 20,
    WHITE)
    DrawText(TextFormat("Player 2 Wins: %d", scoreboard.player2Wins), 300, 650, 20,
```

```
WHITE)
DrawText(TextFormat("Draws: %d", scoreboard.draws), 600, 650, 20, WHITE)
IF (player.whoWon != 0)
        FOR (i FROM 0 TO 7)
                IF (gameBoard.board[gameBoard.winningCombos[i][0]] ==
                player.whoWon AND
                gameBoard.board[gameBoard.winningCombos[i][1]] == player.whoWon
                AND gameBoard.board[gameBoard.winningCombos[i][2]] ==
                player.whoWon) THEN
                        winningCombo[0] ← gameBoard.winningCombos[i][0]
                        winningCombo[1] ← gameBoard.winningCombos[i][1]
                        winningCombo[2] ← gameBoard.winningCombos[i][2]
                        BREAK
                END IF
        END FOR
END IF
FOR (i FROM 0 TO 8)
        int row ← i / 3
        int col ← i % 3
        int cellX ← screenProps.boardX + col * screenProps.squareSize
        int cellY ← screenProps.boardY + row * screenProps.squareSize
        IF (gameBoard.board[i] == 1)
                DrawCircle(cellX + screenProps.squareSize / 2, cellY +
                screenProps.squareSize / 2, screenProps.squareSize / 2 - 20, WHITE)
        ELSE IF (gameBoard.board[i] == -1)
                DrawLineEx((Vector2){cellX + 20, cellY + 20}, (Vector2){cellX +
                screenProps.squareSize - 20, cellY + screenProps.squareSize - 20}, 10,
                WHITE)
                DrawLineEx((Vector2){cellX + screenProps.squareSize - 20, cellY + 20},
                (Vector2){cellX + 20, cellY + screenProps.squareSize - 20}, 10, WHITE)
        END IF
END FOR
player.whoWon ← win(gameBoard.board)
IF (player.whoWon == -1)
        resultMessage ← "PLAYER 1 (X) WINS, PRESS R TO RESTART!"
        gameStatus.showRestartMessage ← 1
ELSE IF (player.whoWon == 1 AND player.gamemodeChoice == 2)
        resultMessage ← "PLAYER 2 (O) WINS, PRESS R TO RESTART!"
        gameStatus.showRestartMessage ← 1
ELSE IF (player.whoWon == 1 AND player.gamemodeChoice == 1)
        IF (player.aiType == 1)
                resultMessage ← "PERFECT MINIMAX AI WINS, PRESS R TO
                RESTART!"
        ELSE IF (player.aiType == 2)
                resultMessage ← "IMPERFECT MINIMAX AI WINS, PRESS R TO
                RESTART!"
        ELSE IF (player.aiType == 3)
                resultMessage ← "EASY MODE AI WINS, PRESS R TO RESTART!"
        ELSE IF (player.aiType == 4)
                resultMessage ← "MACHINE LEARNING AI WINS, PRESS R TO
                RESTART!"
        END IF
        gameStatus.showRestartMessage = 1
ELSE IF (isBoardFull(gameBoard.board))
        resultMessage ← "DRAW, PRESS R TO RESTART!!!"
        gameStatus.showRestartMessage ← 1
        player.whoWon ← 2
```

```
        END IF
        IF (gameStatus.showInvalidMsg == 1)
                resultMessage ← "Please select an empty box!"
        END IF
        int messageWidth ← MeasureText(resultMessage, 25)
        int centerX ← (screenProps.screenWidth - messageWidth) / 2
        int bottomY ← screenProps.screenHeight - 50
        DrawText(resultMessage, centerX, bottomY, 25, WHITE)
        IF (player.turn == 1)
                DrawText("Player 1 (X)'s turn...", 250, 50, 30, WHITE)
        ELSE IF (player.turn == 2 AND player.gamemodeChoice == 2)
                DrawText("Player 2 (O)'s turn...", 250, 50, 30, WHITE)
        ELSE
                DrawText("Computer is thinking...", 250, 50, 30, WHITE)
                IF (player.whoWon == 0)
                        gameStatus.aiMadeMove = makeAIMove(&player.aiType)
                END IF
        END IF
        IF (gameStatus.showRestartMessage == 1)
                IF (IsKeyPressed(KEY_R))
                        restartGame()
                        PlaySound(gameSounds.restartSound)
                ELSE IF (IsKeyPressed(KEY_ESCAPE))
                CloseWindow()
                END IF
        END IF
        int lineThickness ← 5
        IF (player.whoWon != 0)
                int cellSize ← screenProps.squareSize - 20
                IF (winningCombo[0] != -1 AND winningCombo[1] != -1 AND winningCombo[2] !=
                -1)
                        int startX ← screenProps.boardX + (winningCombo[0] % 3) *
                        screenProps.squareSize + screenProps.squareSize / 2
                        int startY ← screenProps.boardY + (winningCombo[0] / 3) *
                        screenProps.squareSize + screenProps.squareSize / 2
                        int endX ← screenProps.boardX + (winningCombo[2] % 3) *
                        screenProps.squareSize + screenProps.squareSize / 2
                        int endY ← screenProps.boardY + (winningCombo[2] / 3) *
                        screenProps.squareSize + screenProps.squareSize / 2
                        Color lineColor ← BLACK
                        DrawLineEx((Vector2){startX, startY}, (Vector2){endX, endY},
                        lineThickness, lineColor)
                END IF
        END IF
        EndDrawing()
END FUNCTION

FUNCTION updateScoreboard(winner)
        winner refToInt → &winner
        IF (*winner == 1)
                scoreboard.player2Wins++
        ELSEIF (*winner == -1)
                scoreboard.player1Wins++
        ELSEIF (*winner == 2)
                scoreboard.draws++
        gameStatus.scoreboardUpdated←true
END FUNCTION
```

tictactoeLogic.c

```
FUNCTION minimaxImperfect(board, comp, depth)
        board refToInt → &board
        winner ← win(board)
        IF (winner != 0)
                RETURN winner * comp
        END IF
        IF (depth == 0)
                RETURN 0
        END IF
        move ← -1
        score ← -2
        FOR (i = 0 TO 8)
                IF (board[i] == 0)
                        board[i] ← comp
                        thisScore ← -minimaxImperfect(board, comp * -1, depth - 1)
                        board[i] ← 0
                        IF (thisScore > score)
                                score ← thisScore
                                move ← i
                        END IF
                END IF
        END FOR
        IF (move == -1)
                RETURN 0
        END IF
        RETURN score
END FUNCTION

FUNCTION minimax(board, comp)
        board refToInt → &board
        winner ← win(board)
        IF (winner != 0)
                RETURN winner * comp
        END IF
        move ← -1
        score ← -2
        FOR i ← 0 TO 8
                IF (board[i] == 0)
                        board[i] ← comp
                        thisScore ← -minimax(board, comp * -1)
                        board[i] ← 0
                        IF (thisScore > score)
                                score ← thisScore
                                move ← i
                        END IF
                END IF
        END FOR
        IF (move == -1)
                RETURN 0
        END IF
        RETURN score
END FUNCTION
```

```
FUNCTION checkButton(buttons)
        buttons refToRectangle &buttons
        mousePos ← getMousePosition()
        FOR (int i = 0; i < 5; i++){
                IF (CheckCollisionPointRec(mousePos, buttons[i]) )
                        RETURN i
                ENDIF
        ENDFOR
        RETURN -1
END FUNCTION

FUNCTION train(examples, numExamples, counts, label)
        examples refToExample &examples
        FOR (int i ← 0; i < numExamples; i++)
                IF (examples[i].target[0] == 'p')
                        label[0] ← label[0] + 1
                ELSE IF (examples[i].target[0] == 'n')
                        label[1] ← label[1] + 1
                ENDIF
                FOR (int j ← 0; j < 9; j++)
                        feature ← examples[i].features[j]
                        target ← examples[i].target[0]
                        FOR (int k ← 0; k < 6; k++)
                                IF (counts[j][k].feature == feature && counts[j][k].target == target)
                                        counts[j][k].count ← counts[j][k].count + 1
                                ENDIF
                                BREAK
                        ENDFOR
                ENDFOR
        ENDFOR
END FUNCTION

FUNCTION learn(counts, probability, label)
        FOR (int k ← 0; k < 9; k++)
                FOR (int i ← 0; i < 6; i++)
                        IF (probability[k][i].target == 'p)
                                probability[k][i].probability ← (counts[k][i].count + 1.0) / (label[0] +
                                3.0)
                        ELSE IF (probability[k][i].target == 'n'){
                                probability[k][i].probability ← (counts[k][i].count + 1.0) / (label[1] +
                                3.0)
                        ENDIF
                        PRINT("Grid: ", k + 1, ", Feature: ", probability[k][i].feature, ", Target: ",
                        probability[k][i].target, ", Probability: ", probability[k][i].probability)
                ENDFOR
        ENDFOR
END FUNCTION

FUNCTION currentfeatures(features, numcounts)
        predict(features,probability,numcounts)
END FUNCTION

FUNCTION predict(features, probability, numcounts)
        features refTochar → &features
        possibility[2] ←
{(double)(label[0])/(double)(label[0]+label[1]),(double)(label[1])/(double)(label[0]+label[1])}
        FOR (int k ← 0; k < 9; k++)
```

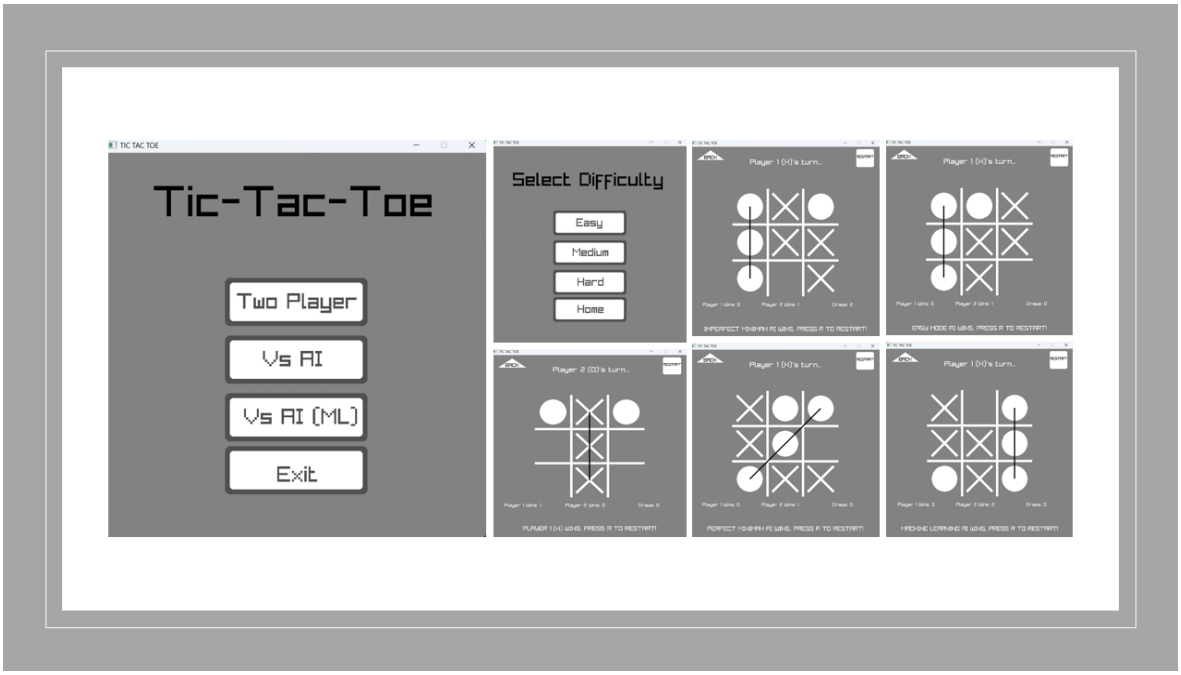
```
            FOR (int i ← 0; i < 6; i++)
                IF (features[k] == probability[k][i].feature && probability[k][i].target == 'p')
                    possibility[0] ← possibility[0] * probability[k][i].probability
                ENDIF
            ENDFOR
        ENDFOR
        FOR (int k ← 0; k < 9; k++)
            FOR (int i ← 0; i < 6; i++)
                IF (features[k] == probability[k][i].feature && probability[k][i].target == 'n')
                    possibility[1] ← possibility[1] * probability[k][i].probability
                ENDIF
            ENDFOR
        ENDFOR
        IF (possibility[0] > maxPossibilityP)
            maxPossibilityP ← possibility[0]
        ENDIF
        IF (possibility[1] > maxPossibilityN)
            maxPossibilityN ← possibility[1]
        ENDIF
        IF (possibility[0] > possibility[1] && possibility[1] == maxPossibilityN)
            bestmoveindex ← numcounts
        ELSE IF (possibility[1] > possibility[0])
            random ← 1;
        ENDIF
        RETURN (possibility[0] > possibility[1]) ? 'p' : 'n'
END FUNCTION

FUNCTION naiveBayes(board, counts, numCounts, numExamples)
        FOR (int i ← 0; i < 3; ++i)
            FOR (int j ← 0; j < 3; ++j)
                IF (board[i * 3 + j] == 1)
                    features[i * 3 + j] ← 'o'
                ELSE IF (board[i * 3 + j] == 0)
                    features[i * 3 + j] ← 'b'
                ELSE IF (board[i * 3 + j] == -1)
                    features[i * 3 + j] ← 'x'
                ENDIF
            ENDFOR
        ENDFOR
        currentfeatures(features,numCounts)
        FOR (int i ← 0; I < 9; ++i)
            IF (gameBoard.board[I] == 0)
                emptyCells[numEmptyCells++] ← i
            ENDIF
        ENDFOR
        IF (numEmptyCells > 0)
            FOR (int i = 0; i < numEmptyCells; ++i)
                IF (emptyCells[i] == bestmoveindex)
                    move ← bestmoveindex
                    BREAK
                ELSE IF (random == 1)
                    move ← emptyCells[rand() % numEmptyCells]
                ENDIF
            ENDFOR
        ENDIF
        RETURN move;
END FUNCTION
```

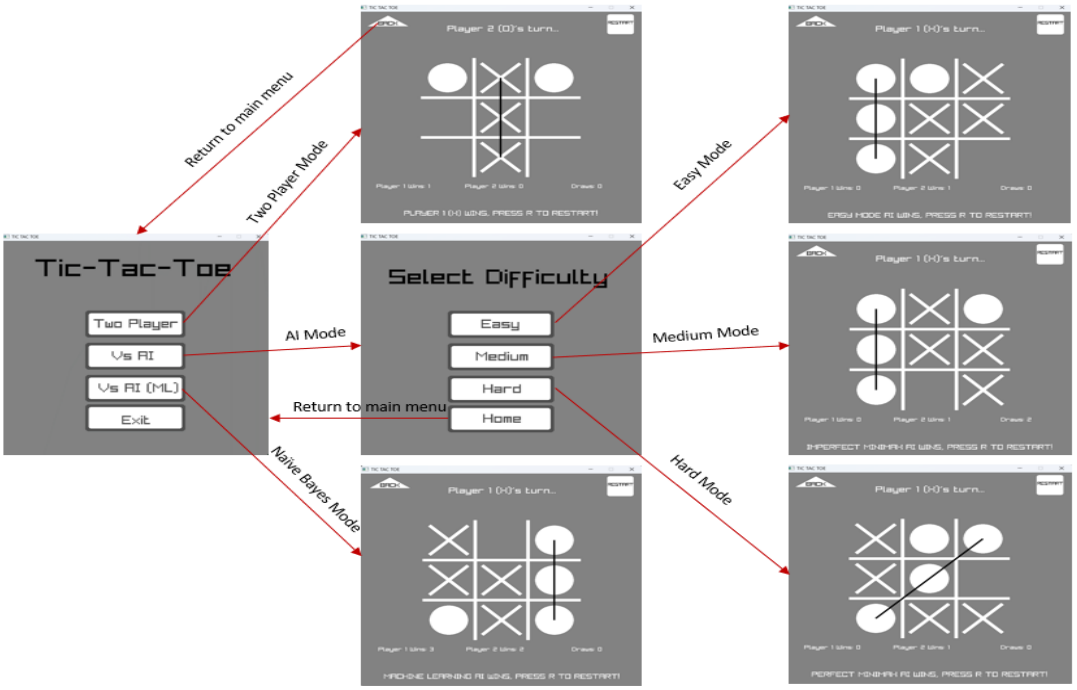
```
FUNCTION readExamplesFromFile(filename, examples)
        filename refToChar &filename
        Examples refToExample &examples
        file = fopen(“tic-tac-toe.data”, "r")
        IF (file == NULL)
                PRINT "Error opening file: filename"
                RETURN 0
        ENDIF
        numExamples ← 0
        line[100] ← 0
        WHILE fgets(line, sizeof(line), file != NULL
                numExamples++
        ENDWHILE
        *examples ← malloc(numExamples * sizeof(Example))
        rewind(file)
        FOR i FROM 0 TO numExamples - 1
                READ_VALUES_FROM_FILE(file, examples[i])
                IF READ_OPERATION_FAILED
                        PRINT "Error reading from file:”, filename
                        free(*examples)
                        fclose(file)
                        RETURN 0
                ENDIF
        ENDFOR
        fclose(file)
        RETURN numExamples
END FUNCTION

FUNCTION splitDataset(allExamples, numExamples, trainingSet, numTraining, testingSet,
numTesting, splitRatio)
        allExamples refToExample → &allExamples
        trainingSet refToExample → &trainingSet
        numTraining refToInt → &numTraining
        testingSet refToExample → &testingSet
        numTesting refToInt → &numTesting
        FOR(numExamples - 1; i > 0; i--)
                j ← rand() % (i +1)
                temp ← allExamples[i]
                allExamples[i] ← allExamples[j]
                allExamples[j] ← temp
        ENDFOR
        numTrainingExamples ← int(numExamples * splitRatio)
        numTestingExamples ← numExamples - numTrainingExamples
        *trainingSet ← malloc(numTrainingExamples * sizeof(Example))
        FOR ( i = 0; i < numTrainingExamples; i++)
                (*trainingSet)[i] ←  allExamples[i]
        ENDFOR
        *testingSet ← malloc(numTestingExamples *sizeof(Example))
        FOR (numTrainingExamples; i < numExamples; i++)
                (*trainingSet)[i - numTrainingExamples] ← allExamples[i]
        ENDFOR
        *numTraining ← numTrainingExamples
        *numTesting ← numTestingExamples
END FUNCTION
```

```
FUNCTION win(board)
        board refToInt → &board
        wins ← {{0,1,2},{3,4,5},{6,7,8},{0,3,6},{1,4,7},{2,5,8},{0,4,8},{2,4,6}}
        FOR (i = 0; i < 8; i++)
                IF board[wins[i][0]] != 0 AND
                        board[wins[i][0]] == board[wins[i][1]] AND
                        board[wins[i][0]] == board[wins[i][2]] THEN
                        RETURN board[wins[i][0]]
                ENDIF
        ENDFOR
        RETURN 0
END FUNCTION

FUNCTION swapTurn(currentTurn)
        currentTurn refToInt → &currentTurn
        IF (*currentTurn == 1)
                *currentTurn ← 2
        ELSE
                *currentTurn ← 1
        END IF
        RETURN 0
END FUNCTION

FUNCTION checkInput(choice)
        choice refToInt → &choice
        IF (gameBoard.board[*choice] != 0)
                RETURN 1
        ELSE
                RETURN 0
        END IF
END FUNCTION

FUNCTION insertChoice(choice)
        choice refToint→&choice
        gameStatus.validInputFlag ← checkInput(choice)
        IF gameStatus.validInputFlag == 1
                gameStatus.showInvalidMsg ← 1
        ELSE
        gameStatus.showInvalidMsg  ← 0
                IF (player.turn == )
                        gameBoard.board[*choice] ← -1
                        swapTurn(&player.turn)
                        player.numOfTurns ← player.numOfTurns + 1
                ELSE
                        gameBoard.board[*choice] ← 1
                        swapTurn(&player.turn)
                        player.numOfTurns ← player.numOfTurns + 1
                END IF
        END IF
        RETURN 0
END FUNCTION

FUNCTION makeAIMove(aiType)
        aiType refToInt → &aiType
        move ← -1
        bestScore ← -2
        FOR (i=0; i<9; i++)
```

```
            IF (gameBoard.board[i] == 0)
                gameBoard.board[i] ← 1
                IF (*aiType == 1)
                    score ←minimax(gameBoard.board, -1)
                    IF (score > bestScore)
                        bestScore ← score
                        move ← i
                    END IF
                ELSE IF (*aiType == 2)
                    score ← minimaxImperfect(gameBoard.board, -1, 2)
                    IF (score > bestScore)
                        bestScore ← score
                        move ← i
                    END IF
                ELSE IF (*aiType == 3)
                    score ← minimaxImperfect(gameBoard.board, -1, 0)
                    IF score > bestScore
                        bestScore ← score
                        move ← i
                    END IF
                ELSE IF (*aiType == 4)
                    move ← naiveBayes(gameBoard.board, counts, i, numTraining)
                END IF
                gameBoard.board[i] ← 0
            END IF
        END FOR
        insertChoice(&move)
END FUNCTION
```

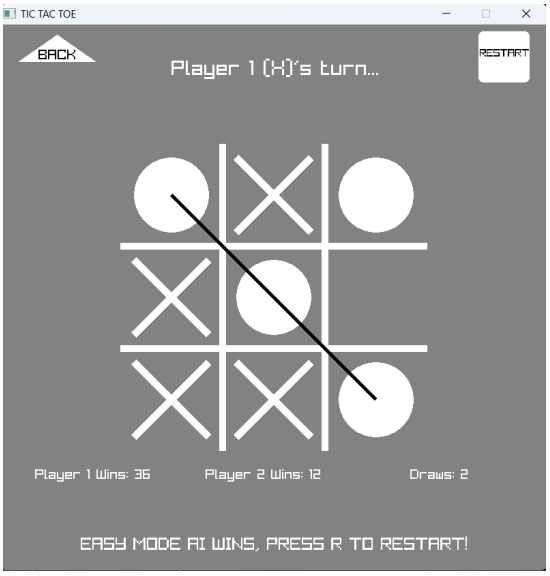
5. Plots and Results

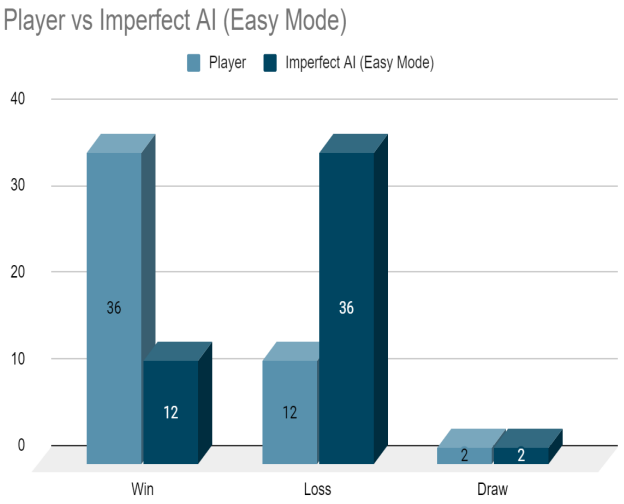




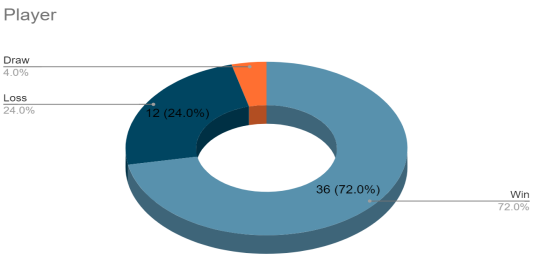
- The figures above show the two player, vs AI mode. If the player selects “Vs AI” mode, it will prompt to another page where players are able to select the difficulty or return back to the main menu. The game ends when either one player wins vice versa in AI mode with a strikethrough effect. The result will be tabulated with the scores as shown in the Tic-Tac-Toe grid.

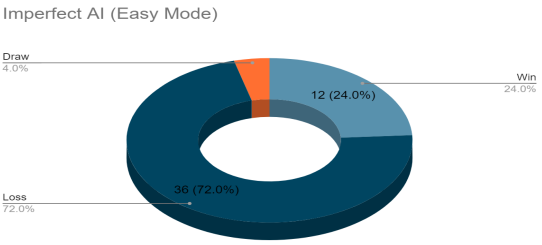
- **Player vs Imperfect AI (Easy Mode):**





"Out of 50 game sets, Player 1 won 36, Player 2 (Imperfect AI, Easy Mode) won 12, and there were 2 draws"





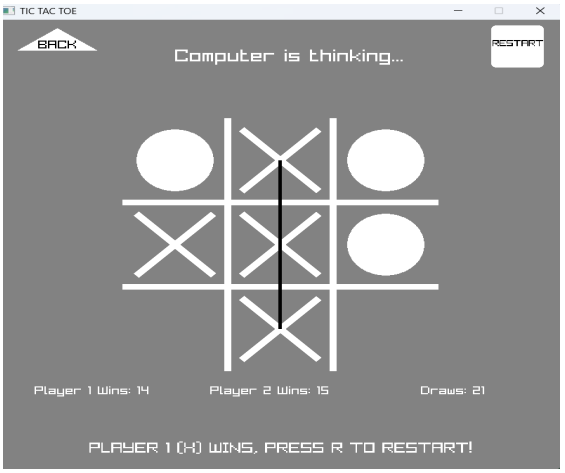
https://youtu.be/H2uWiUfWU5o?si=C8Qlk0CXWi1sjqpB

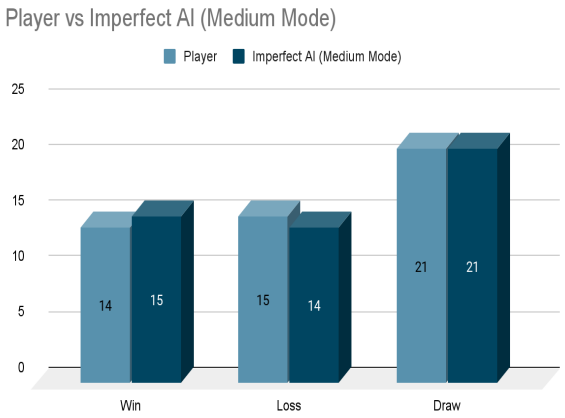
- The information generated shows that the total percentile of win rate, lose rate, and draw were 72.0%, 24.0%, and 4.0% respectively for players.

- Based on the information generated we can also see that the total percentile of win rate, lose rate, and draw were 24.0%, 72.0%, and 4.0% respectively for Imperfect AI (Easy Mode).

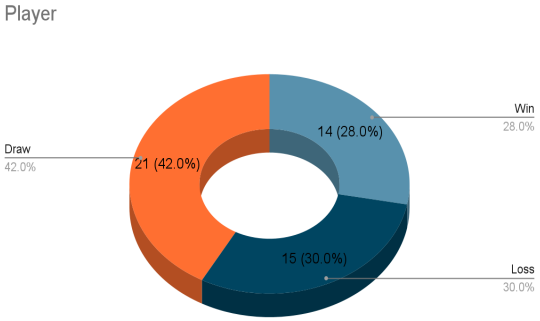
- From this comparison data, we can conclude that indeed the imperfect AI minimax algorithm (Easy Mode) was implemented successfully which allows the player to have a higher chance to win the AI out of the 50 sets of gameplay as tested.

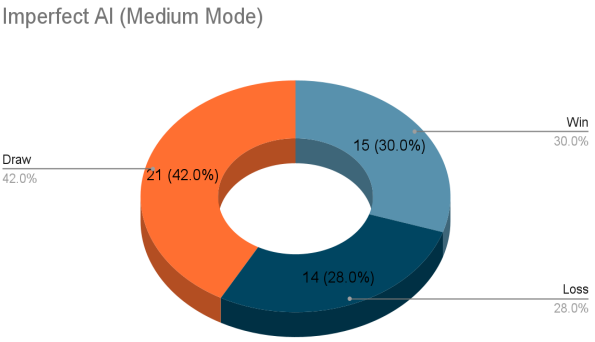
- **Player vs Imperfect AI (Medium Mode):**





"Out of 50 game sets, Player 1 won 14, Player 2 (Imperfect AI, Normal Mode) won 15, and there were 21 draws"





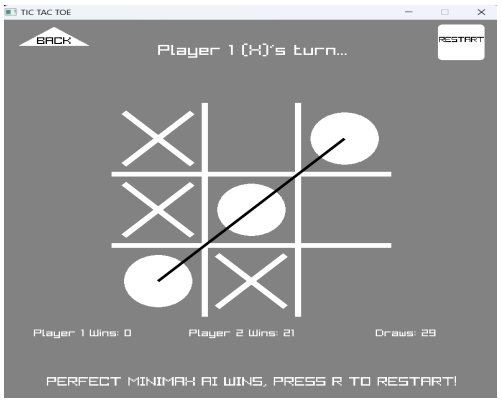
https://youtu.be/j6J30WmPfVg?si=JEKuaAwUDOOolPEz

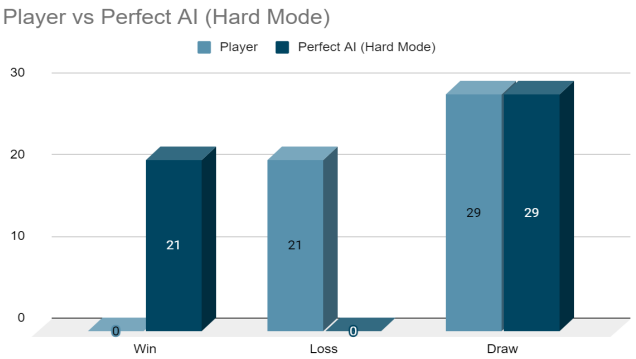
- The information generated shows that the total percentile of win rate, lose rate, and draw were 28.0%, 42.0%, and 42.0% respectively for players.

- Based on the information generated we can also see that the total percentile of win rate, lose rate, and draw were 30.0%, 28.0%, and 42.0% respectively for Imperfect AI (Medium Mode).

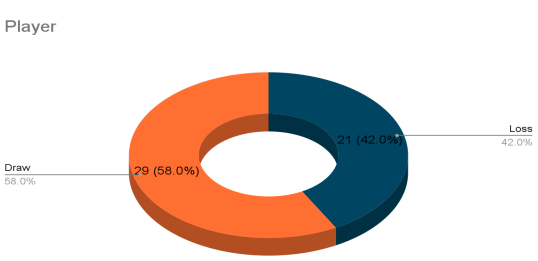
- From this comparison data, we can conclude that indeed the imperfect AI minimax algorithm (Medium Mode) was implemented successfully which allows the player to have a fair chance to win the AI out of the 50 sets of gameplay as tested.

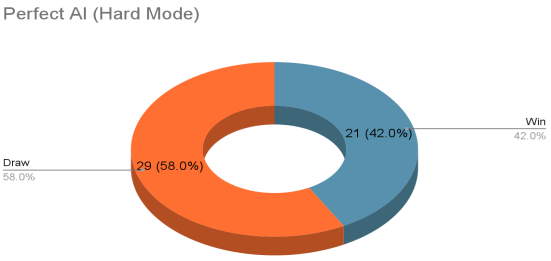
- **Player vs Perfect AI (Hard Mode):**





"Out of 50 game sets, Player 1 won 0, Player 2 (Perfect AI, Hard Mode) won 21, and there were 29 draws"

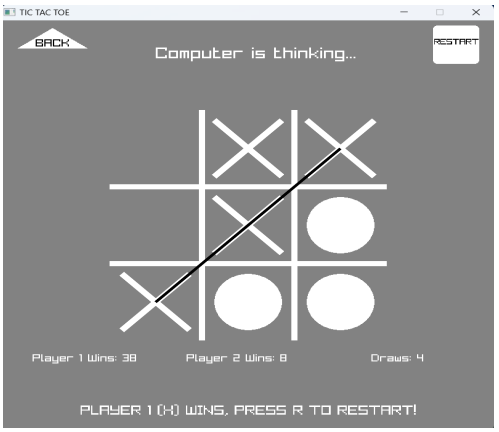


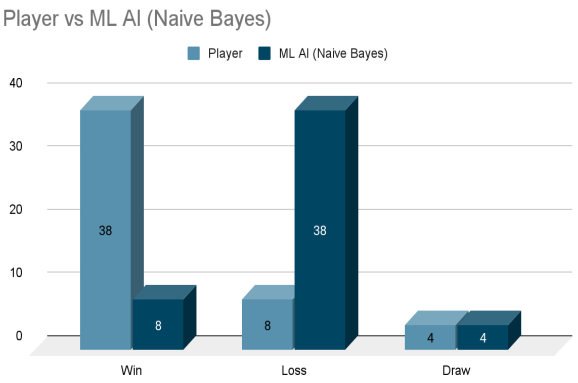


https://youtu.be/HhWwnIEbrzo?si=hKAP33i-pwL3RtNc

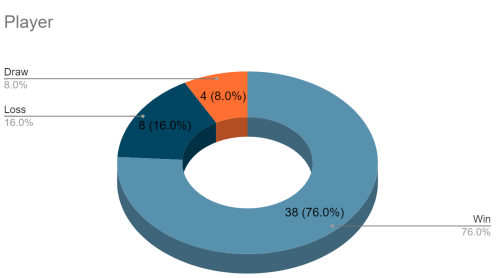
- The information generated shows that the total percentile of win rate, lose rate, and draw were 0.0%, 42.0%, and 58.0% respectively for players.
- Based on the information generated we can also see that the total percentile of win rate, lose rate, and draw were 42.0%, 58.0%, and 0.0% respectively for Perfect AI (Hard Mode).
- From this comparison data, we can conclude that indeed the Perfect AI minimax algorithm (Hard Mode) was implemented successfully, making it almost impossible for players to have a chance to win the AI out of the 50 sets of gameplay as tested.

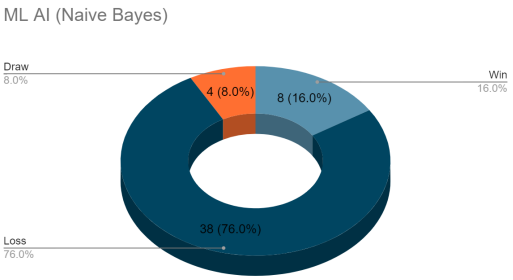
- **Player vs ML AI (Naive Bayes):**





"Out of 50 game sets, Player 1 won 38, Player 2 (ML AI, Easy Mode) won 8, and there were 4 draws"



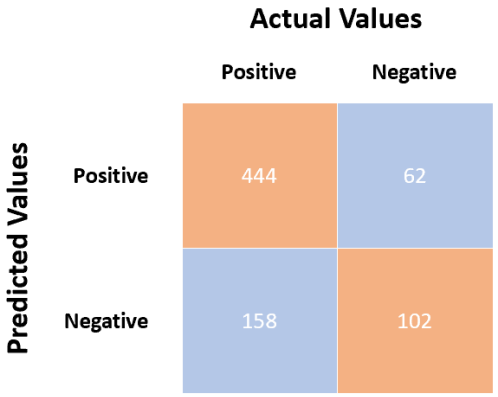


https://youtu.be/9aTYWm7W6Oc

- The information generated shows that the total percentile of win rate, lose rate, and draw were 76.0%, 16.0%, and 8.0% respectively for players.
- Based on the information generated we can also see that the total percentile of win rate, lose rate, and draw were 16.0%, 76.0%, and 8.0% respectively for ML AI (Naive Bayes).
- From this comparison data, we can conclude that the Naive Bayes algorithm, as implemented in the context of the tic-tac-toe game, exhibits a contrasting performance pattern to player pattern.
- The players achieved a win rate of 76.0% while the ML AI (Naive Bayes) had a win rate of only 16.0%. Conversely, the player had a loss rate of 16.0%, whereas the ML AI (Naive Bayes) had a much higher loss rate of 76.0%. This stark difference suggests that Naive Bayes may not effectively capture the strategies employed by the player in the game.

- **ML AI Accuracy and Confusion Matrix**

```
Training Accuracy:
Accuracy: 0.723958
Confusion Matrix:
                Predicted: YES  Predicted: NO
Actual: YES     105             15
Actual: NO      38              34

Test Accuracy:
Accuracy: 0.712794
Confusion Matrix:
                Predicted: YES  Predicted: NO
Actual: YES     444             62
Actual: NO      158             102
```



- For the accuracy of the Naive Bayes algorithm, we have split up our dataset into training and testing data into a ratio of 80:20. After the model was trained, predictions were made on the testing set to evaluate its performance.

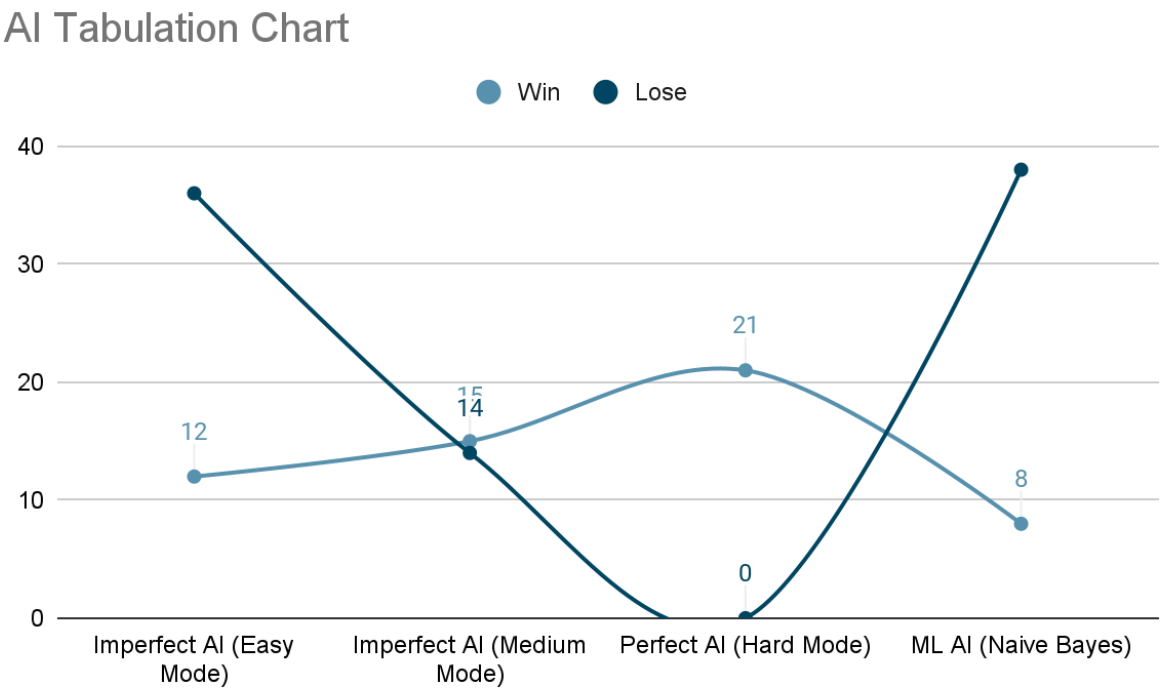
- Accuracy was then calculated by checking through the values, to gather how many predicted values matched actual values.

- A confusion matrix was also tabulated as an evaluation tool of the Naive Bayes performance, to judge the model's predictions compared to actual values. It displays the counts of True Positive (444), True Negative (62), False Positive (158), and False Negative (102) predictions.

- In this context, this confusion matrix represents predictions of wins and losses. Performance of the Naive Bayes classifier can be seen by evaluating its ability to differentiate between wins and losses.

- However, the Naive Bayes was shown to have a relatively low accuracy, especially when compared with the Minimax AI. Furthermore, the confusion matrix also shows numeral instances of False Negatives and False Positives, which basically leads to wrong conclusions. This suggests that Naive Bayes might not be suited for the tic-tac-toe game.

- **AI Comparison**



- In examining the tabulated results of the tic-tac-toe game, a noticeable trend emerges when selections are made from the easiest to the hardest mode. The slope gradually increases in correlation with wins and decreases in correlation with losses, indicating a logical progression of difficulty.

- However, a noteworthy deviation is observed when players opt for the ML AI with Naive Bayes. The steepness of the slope aligns more closely with that of an Imperfect AI (Easy Mode) or worse. This suggests that Naive Bayes performs at a level similar to, or even inferior to the easy mode. Consequently, it can be concluded that the implementation of Naive Bayes is not well-suited for the complexities of the tic-tac-toe game.

- In conclusion, the analysis obtained from those above such as the data chart, accuracy and confusion matrix reveals that Naive Bayes do not excel in capturing the strategic nuances of the game, demonstrating performance comparable to an easy mode. This underscores the importance of exploring alternative machine learning algorithms better equipped to handle the complexities and decision making challenges inherent in tic-tac-toe.

6. Comparison Comments

In this section, we compare the performance and gameplay experience of the different modes and AI difficulty levels.

- **User engagement in two-player mode vs AI mode:**

Two-Player Mode	AI Mode
High engagement due to human interaction	May lack the social and dynamic aspects
Dependant on the availability of players	Consistent availability, anytime play
Good for learning game mechanics	Challenging and rewarding experience

- **Impact of AI difficulty on the user experience:**

Perfect AI	Imperfect AI
Extremely high, optimal decision-making	Varied, offers room for human-like mistakes
Might be intimidating for some players	Provides a challenge without being too rigid
Encourages refinement of skills	Provides a more relatable and enjoyable feel

- The choice of game mode and AI difficulty levels can have both positive and negative impacts on children. Thus with the implementation, it strikes a balance to provide an engaging and educational experience that caters to a diverse range of skill levels while promoting skill development and enjoyment during their formative years.

7. Interesting Aspects

Computer AI

When comparing the performance of the Minimax algorithm and Naive Bayes in tic-tac-toe, their roles and characteristics must be considered.

Minimax Algorithm:

- For our Minimax algorithm, it exhaustively searches through all possible options to make the best move, to minimise the potential loss. All moves are evaluated recursively to ensure that all future possibilities are considered, before making a decision.

- Due to it being a deterministic algorithm, the AI guarantees an **optimal** move, leading to a win or a draw. However, due to it needing to explore the entire game tree first, this can be quite extensive, with a **substantial memory demand**. For more complex games, this could require an exponentially high amount of computer resources.

Naive Bayes:

- Naive Bayes is a more probabilistic machine learning algorithm, operating by assuming independence between features. In our tic-tac-toe context, It is used to predict the likelihood of winning, losing and or drawing.

- Probabilities considered are based only on the observed board, there is no accounting for the opponents moves, or future game states. In tic-tac-toe, thinking ahead is essential, and thus moves made are **not always optimal**, giving a lot of leeway for a loss.

- Due to the simple algorithm, it is more **memory efficient** compared to the Minimax algorithm. It has a more fixed and predictable approach, and does not need to store the entire game tree to work.

Code Structure and Modularity

- Utilising structured data type for defining our data variables and employing pointers for enhanced memory efficiency contributes to improved code modularity and readability, aligning with the considerations outlined in our program analysis 2.6.

Header File Implementation

As seen throughout this report, our team has utilised a header file for our game program. Previously, our code was all in 1 file which we found quite disorganised. Hence, we decided to split the codes and utilise a header file, organising them based on their usage.

The following are our files:

- tictactoe.h
  - The header file where our variables, structures and functions are defined.
- tictactoeMain.c
  - The “main” file where our main() function exists and the “start” of our program.
  - Also contains our initialisation of variables and structures.
- tictactoeLogic.c
  - The file that houses our functions regarding the Tic-Tac-Toe game logic.
- tictactoeGUI.c
  - The file that houses our functions regarding the game program GUI.
  - All of the RayLib function calls are housed here.

By utilising a header file and splitting our codes based on purpose, we have greatly improved the organisation and modularity of our codes. It also helps us to debug our codes as we are able to make changes without affecting other parts of the program.

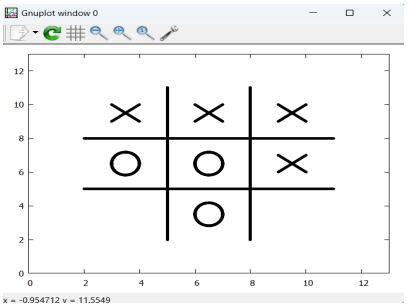
RayLib GUI Implementation

As GUI is a requirement of the project, we initially began using the suggested gnuplot GUI tool to help plot our game. However, we soon found that gnuplot has its limitations and would not be able to accomplish our vision of the end product.

Limitations such as:

- Non-interactive or not being able to “Click” on the GUI
- Having to ‘replot’ the grid every time a move was made
- Either only showing the game GUI at the end of the game where all points are plotted, or only having the gnuplot output a new window every time a move is made.
- Additional overhead of having an extra file. A csv file which has the plot points for gnuplot.

We decided to migrate the GUI portion of the project to use RayLib. RayLib is a library for the C language where it specialises in the creation of a game GUI. We are able to overcome the limitations of gnuplot using RayLib and create an actual Tic-Tac-Toe game program.



https://youtu.be/cFCO1AV1B-I

https://youtu.be/tJXzzxpCqGA

Visual and Auditory Enhancement

We added a strikethrough effect for winning combinations visually emphasising the victorious moment. Additionally, sound effects, like a satisfying click for selections, celebratory sounds for restarts, and background music, provided a multisensory layer to our Tic-Tac-Toe interactions. It elevated the overall GUI outlook, making our Tic-Tac-Toe game not only functional but also visually appealing.

In conclusion, the tic-tac-toe project evolved with the combined strengths of GNU Plot and RayLib from superior data visualisation and customization with GNUPlot to the interactive GUI, visual enhancement, and sound effect with RayLib, our project became a comprehensive exploration of these powerful tools.

---

8. Conclusion

We summarise our key findings and challenges of our Tic-Tac-Toe implementation. We also discuss the area for potential future improvement for our game program.

Key Findings

Educational Focus:

- Successful incorporation of educational objectives, including motor skill development and analytical thinking through our GUI and single-player vs AI mode.
- Enhanced motor skills, fostered social skills, stimulated analytical thinking, and contributed to left brain development.

Adaptation to IoT Environment:

- Effective optimization for memory and processing constraints in IoT tablets.
- User-friendly GUI designed for intuitive navigation by young children through our usage of RayLib.
- Seamless functionality within limited memory and processing capabilities.

Holistic Child Development:

- Achieved an interactive learning experience through our interactive/clickable game program.
- Accessibility for both single and multiple users through the different One-Player and Two-Player modes.
- The Tic-Tac-Toe game serves as a valuable tool for early childhood education and skill enhancement.

Challenges

- Understanding a whole new library such as GNUPlot, RayLib from scratch and integrating into the existing code.
- GNUPlot was non interactive. It only provides a visual presentation of the tic-tac-toe game set. Thus, we look into another alternative using RayLib which provides interactive tools and functions such that it is able to mimic an actual retro tic-tac-toe gameplay set.
- Originally when our code was coded in a single file it was quite out of place but now it is splitted with a few different files with logic, GUI, main, and header file for better code structure modularity and readability.

Areas for Future Improvement/Expansion

AI Enhancement:

- Further refinement of AI algorithms for increased difficulty levels.

Additional Features:

- Integration of more educational elements to expand learning opportunities.

Multi-Device Compatibility:

- Explore options for making the game compatible with a broader range of devices.

Enhanced GUI Features:

- Continuous improvement of the GUI for an even more engaging and visually appealing experience.

Collaborative Learning:

- Implementation of features that encourage collaborative learning among children.

Data Collection and Analysis:

- Incorporation of mechanisms for collecting gameplay data to assess learning outcomes.

Overall, the implementation successfully addressed initial goals, providing a foundation for continuous improvement and expansion to further enrich the educational and gaming experience for children.

9. Appendices

tictactoe.h

```
/*
TO INSTALL RAYLIB:
1) OPEN MSYS2
2) Run: pacman -S mingw-w64-x86_64-raylib
3) DONE

TO RUN PROGRAM:
1) In terminal: gcc -fcommon -o tictactoe tictactoeMain.c tictactoeGUI.c
tictactoeLogic.c -lraylib -lopengl32 -lgdi32 -lwinmm
2) Then: .\tictactoe
*/

// Libraries
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <raylib.h>
#include <string.h>
#include <math.h>

// Structure to represent each example in the Naive Bayes model
typedef struct {
    char features[9]; // 9 features
    char target[10];  // 10th feature is the target
} Example;

// Structure to store counts for each feature and target combination
typedef struct {
    char feature;  // Feature
    char target;   // Target
    int count;     // Count
} FeatureCount;

typedef struct {
    char feature;  // Feature
    char target;   // Target
    double probability;     // Probability
} ProbabilityCount;
```

```
int countIndex;  // Index for the current count in the counts array
FeatureCount counts[9][6];  // Array to store the counts of features
int numTraining;  // Number of training examples
extern int label[2];
ProbabilityCount probability[9][6];

// Structure definition for player settings
typedef struct {
    int aiType;         // 1 for perfect AI, 2 for imperfect AI (default to
perfect)
    int turn;           // 1 for X, 2 for O (default to X)
    int gamemodeChoice; // 1 for 1P, 2 for 2P (default to 1P)
    int numOfTurns;     // Number of turns taken
    int whoWon;         // -1 = X win, 0 = Draw, 1 = O win
    int loopState;      // 0 for main menu, 1 for game screen
} PlayerSettings;
PlayerSettings player;

// Structure to manage game status flags
typedef struct {
    int showRestartMessage;     // Flag for showing restart message
    int validInputFlag;         // Flag for valid input
    int showInvalidMsg;         // Flag for showing invalid message
    int aiMadeMove;             // Flag indicating whether AI made a move
    int scoreboardUpdated;      // Flag indicating whether the scoreboard is
updated
} GameStatus;
GameStatus gameStatus;

// Rectangle buttons for UI interaction
Rectangle backButton;         // Back button
Rectangle restartButton;      // Restart button

// Structure to store game scoreboard
typedef struct {
    int player2Wins;   // O wins
    int player1Wins;   // X wins
    int draws;         // Draws
} Scoreboard;
Scoreboard scoreboard;

// Structure to store properties of the game screen
typedef struct {
```

```
    const int screenWidth;    // Set the screen width
    const int screenHeight;   // Set the screen height
    const int squareSize;     // Set the square size for the Tic-Tac-Toe grid
    int boardX;               // Set the initial X-coordinate
    int boardY;               // Set the initial Y-coordinate
} ScreenProperties;
ScreenProperties screenProps;

// Structure to represent the game board
typedef struct {
    int winningCombos[8][3];  // Winning combinations
    int board[9];             // Initialize the board with empty spaces (0)
} GameBoard;
GameBoard gameBoard;

// Structure to store game sounds
typedef struct {
    Sound moveSound;        // Sound for when a move is made
    Sound restartSound;     // Sound for when the restart button is pressed
    Sound backgroundSound;  // Background music
} GameSounds;
GameSounds gameSounds;

// Function prototypes
int minimaxImperfect(int* board, int comp, int depth);
int minimax(int* board, int comp);
int win(const int* board);
int swapTurn(int* currentTurn);
int checkInput(int* choice);
int insertChoice(int* choice);
int makeAIMove(int* aiType);
void drawGame();
void restartGame();
void initializeGame(int* gamemodeChoice);
void gameLoop();
void handleInput();
int isBoardFull(const int* board);
int checkButton(Rectangle* buttons);
void updateScoreboard(int* winner);

// Naive Bayes
int initialiseNaiveBayes();
void train(Example* examples, int numExamples, FeatureCount counts[9][6], int
label[2]);
```

```
char predict(char* features, ProbabilityCount probability[9][6], int
bestmoveindex);
int readExamplesFromFile(const char* filename, Example** examples);
void splitDataset(Example* allExamples, int numExamples, Example** trainingSet,
int* numTraining, Example** testingSet, int* numTesting, double splitRatio);
int naiveBayes(int board[9], FeatureCount counts[9][6], int numCounts, int
numExamples);
double learn(FeatureCount counts[9][6], ProbabilityCount probability[9][6]);
void currentfeatures(char features[9], int bestmoveindex);
void testingAccuracy(Example* testingSet, int numTesting, ProbabilityCount
probability[9][6], int numCounts);
```

tictactoeMain.c

```c
#include "tictactoe.h"

// Initialize player settings
PlayerSettings player = {
  1, // aiType
  1, // turn
  0, // gamemodeChoice
  0, // numOfTurns
  0, // whoWon
  0  // loopState
};

// Initialize scoreboard
Scoreboard scoreboard = {
  0, // player2Wins
  0, // player1Wins
  0  // draws
};

// Initialize game board with winning combinations and an empty board
GameBoard gameBoard = {
    .winningCombos = {
        {0, 1, 2}, {3, 4, 5}, {6, 7, 8},  // Horizontal
        {0, 3, 6}, {1, 4, 7}, {2, 5, 8},  // Vertical
        {0, 4, 8}, {2, 4, 6}              // Diagonal
    },
    .board = {0, 0, 0, 0, 0, 0, 0, 0, 0}  // Initialize the board with empty spaces (0)
};

// Initialize game status flags
GameStatus gameStatus = {
    .showRestartMessage = 1,  // Show restart message initially
    .validInputFlag = 0,      // Flag for valid input (initialize to 0)
    .showInvalidMsg = 0,      // Flag for showing invalid message (initialize to 0)
    .aiMadeMove = 0,          // Flag indicating whether AI made a move (initialize to 0)
    .scoreboardUpdated = 0    // Flag indicating whether the scoreboard is updated (initialize to 0)
};
```

```
// Initialize screen properties
ScreenProperties screenProps = {
    .screenWidth = 800,   // Set the screen width
    .screenHeight = 800,  // Set the screen height
    .squareSize = 150,    // Set the square size for the Tic-Tac-Toe grid
    .boardX = 0,          // Set the initial X-coordinate
    .boardY = 0           // Set the initial Y-coordinate
};

// Initialize rectangles for UI interaction
Rectangle backButton = {20, 20, 100, 45};      // Define the position and size
of the back button
Rectangle restartButton = {700, 10, 75, 75 };  // Define the position and size
of the restart button

ProbabilityCount probability[9][6];
FeatureCount counts[9][6];  // Array to hold feature counts
int countIndex = 0;  // Index for the counts array
int label[2] = {0,0};

char features[] = {'x', 'o', 'b'};  // Array to hold the different features (x,
o, b)
char targets[] = {'p', 'n'};  // Array to hold the different targets (p, n)

int main(){
    // Initialise the Naive Bayes ML
    initialiseNaiveBayes();
    // Initialize the game
    initializeGame(&player.gamemodeChoice);

    // Unload sounds and close audio device
    UnloadSound(gameSounds.moveSound);
    UnloadSound(gameSounds.restartSound);
    UnloadSound(gameSounds.backgroundSound);
    CloseAudioDevice();

  return 0;
}

int initialiseNaiveBayes(){
    // Read examples from file
    Example* allExamples;
    int numExamples = readExamplesFromFile("tictactoe.data", &allExamples);
    if (numExamples == 0){
```

```
      printf("Failed to read file or file is empty.\n");
      return 1;
    } else {
      printf("Successfully read %d examples from the file.\n", numExamples);
    }

    // Split data into training and testing sets
    Example* trainingSet;
    Example* testingSet;
    extern int numTraining;
    int numTesting;
    double splitRatio = 0.8;  // 80% of the data for training
    splitDataset(allExamples, numExamples, &trainingSet, &numTraining,
&testingSet, &numTesting, splitRatio);

    // Print the number of examples in each set
    printf("Number of training examples: %d\n", numTraining);
    printf("Number of testing examples: %d\n", numTesting);
    printf("Total number of examples: %d\n", numExamples);

    // Verify the split function
    if (numTraining + numTesting == numExamples) {
      printf("The splitExamples function works correctly.\n");
    } else {
      printf("The splitExamples function does not work correctly.\n");
    }



    // Initialize feature counts
    for (int k = 0; k < 9; k++)
    {
      for (int i = 0; i < sizeof(features) / sizeof(features[0]); i++) {
          for (int j = 0; j < sizeof(targets) / sizeof(targets[0]); j++) {
              counts[k][countIndex].feature = features[i];
              counts[k][countIndex].target = targets[j];
              counts[k][countIndex].count = 0;
              countIndex++;
          }
      }
      countIndex = 0;
    }

    // Initialize probability counts
    for (int k = 0; k < 9; k++)
```

```
    {
      for (int i = 0; i < sizeof(features) / sizeof(features[0]); i++) {
          for (int j = 0; j < sizeof(targets) / sizeof(targets[0]); j++) {
              probability[k][countIndex].feature = features[i];
              probability[k][countIndex].target = targets[j];
              probability[k][countIndex].probability = 0;
              countIndex++;
          }
      }
      countIndex = 0;
    }


    // Train the model
    train(trainingSet, numTraining, counts, label);

    learn(counts,probability);

    printf("\nTraining Accuracy:");
    testingAccuracy(testingSet, numTesting, probability, countIndex);
    printf("\nTest Accuracy:");
    testingAccuracy(trainingSet, numTraining, probability, countIndex);

}
```

tictactoeGUI.c

```c
#include "tictactoe.h"

// This function initializes the game
void initializeGame(int* gamemodeChoice) {

    InitAudioDevice(); // Initialize the audio device

    // Load the sounds
    gameSounds.moveSound = LoadSound("mixkit-retro-game-notification-212.wav");
// Sound for when a move is made
    gameSounds.restartSound = LoadSound("mixkit-arcade-bonus-alert-767.wav"); //
Sound for when the game is restarted
    gameSounds.backgroundSound = LoadSound("pop.mp3"); // Background music

    PlaySound(gameSounds.backgroundSound); // Start playing the background music

    // Initialization code (window, AI type, etc.)
    player.whoWon = 0; // -1 = X win, 0 = Draw, 1 = O win

    // Start the game loop, passing the game mode choice
    gameLoop(*gamemodeChoice);

    CloseWindow(); // Close the window when the game is over
}

// Main game loop
void gameLoop() {

    // Initialize window and textures
    InitWindow(screenProps.screenWidth, screenProps.screenHeight, "TIC TAC
TOE");
    SetTargetFPS(60);
    RenderTexture2D screen = LoadRenderTexture(screenProps.screenWidth,
screenProps.screenHeight);
    RenderTexture2D X = LoadRenderTexture(screenProps.squareSize,
screenProps.squareSize);
    RenderTexture2D O = LoadRenderTexture(screenProps.squareSize,
screenProps.squareSize);

    // Initialize volume for background sound
    float volume = 1.0f;
    SetSoundVolume(gameSounds.backgroundSound, volume);
```

```
    // Main game loop
    while (!WindowShouldClose()) {
        handleInput(); // Handle player's input
        int pressedButton;

        // Define buttons
        Rectangle buttons[6] = {
        (Rectangle){250, 260, 300, 100},
        (Rectangle){250, 380, 300, 100},
        (Rectangle){250, 500, 300, 100},
        (Rectangle){250, 610, 300, 100},
        backButton,
        restartButton
        };

        // Check if the M key is pressed to mute/unmute the background music
        if (IsKeyPressed(KEY_M)) { // If M key is pressed
            if (volume != 0.0f) {
                volume = 0.0f; // Mute
            } else {
                volume = 1.0f; // Unmute
            }
            SetSoundVolume(gameSounds.backgroundSound, volume);
        }

        // Check if the background music is playing, if not, play it
        if (!IsSoundPlaying(gameSounds.backgroundSound)) {
            PlaySound(gameSounds.backgroundSound);
            }

        // Game states
        switch (player.loopState)
        {
            case 0:
                // Home screen
                pressedButton = checkButton(buttons);
                if ((pressedButton == 0 || pressedButton == 1) &&
IsMouseButtonPressed(0)){
                    // Check if Multiplayer = 0 or Single player = 1
                    player.loopState = (pressedButton == 0) ? 3 : 1;
                    player.gamemodeChoice = (pressedButton == 0) ? 2 : 1;
                } else if (pressedButton == 2 && IsMouseButtonPressed(0)) {
                    player.gamemodeChoice = 1;
```

```
                    player.aiType = 4; // Naive Bayes AI
                    player.loopState = 3; // Move to the game state
                } else if (pressedButton == 3 && IsMouseButtonPressed(0)) {
                    // Exit the game
                    CloseWindow();
                    break;
                }

                BeginDrawing();
                BeginTextureMode(screen);
                ClearBackground(GRAY);
                DrawText("Tic-Tac-Toe", 100, 60, 90, BLACK);

                // Draw buttons
                for (int i = 0; i < 4; i++)
                {
                    DrawRectangleRounded(buttons[i], 0.2, 5, (pressedButton ==
i) ? BLACK:DARKGRAY);
                    DrawRectangleRounded((Rectangle){buttons[i].x+10,
buttons[i].y+10, 280, 80}, 0.2, 5, WHITE);
                }

                // Draw button labels
                DrawText("Two Player", 275, 288, 45, (pressedButton == 0) ?
BLACK:DARKGRAY);
                DrawText("Vs AI", 330, 408, 45, (pressedButton == 1) ?
BLACK:DARKGRAY);
                DrawText("Vs AI (ML)", 290, 528, 45, (pressedButton == 2) ?
BLACK:DARKGRAY);
                DrawText("Exit", 360, 648, 45, (pressedButton == 3) ?
BLACK:DARKGRAY);

                EndTextureMode();
                DrawTextureRec(screen.texture, (Rectangle){0, 0,
screenProps.screenWidth, -screenProps.screenHeight}, (Vector2){0, 0}, WHITE);
                EndDrawing();
                break;
            case 1:
                // AI type selection menu
                pressedButton = checkButton(buttons);

                // Check which AI difficulty was selected
                if ((pressedButton == 0 || pressedButton == 1 || pressedButton
== 2) && IsMouseButtonPressed(0)) {
```

```
                    // Assign AI type based on button pressed (0 = Easy, 1 =
Medium, 2 = Hard)
                    player.aiType = (pressedButton == 0) ? 3 : (pressedButton ==
1) ? 2 : 1;
                    player.loopState = 3; // Move to the game state
                } else if (pressedButton == 3 && IsMouseButtonPressed(0)) {
                    // Go back to the home screen
                    player.loopState = 0;
                }

                BeginDrawing();
                BeginTextureMode(screen);
                ClearBackground(GRAY);
                DrawText("Select Difficulty", 85, 100, 75, BLACK);

                // Draw the buttons
                for (int i = 0; i < 4; i++)
                {
                    DrawRectangleRounded(buttons[i], 0.2, 5, (pressedButton ==
i) ? BLACK:DARKGRAY);
                    DrawRectangleRounded((Rectangle){buttons[i].x+10,
buttons[i].y+10, 280, 80}, 0.2, 5, WHITE);
                }

                // Draw button labels
                DrawText("Easy", 345, 288, 45, (pressedButton == 0) ?
BLACK:DARKGRAY);
                DrawText("Medium", 330, 408, 45, (pressedButton == 1) ?
BLACK:DARKGRAY);
                DrawText("Hard", 350, 528, 45, (pressedButton == 2) ?
BLACK:DARKGRAY);
                DrawText("Home", 350, 638, 45, (pressedButton == 3) ?
BLACK:DARKGRAY);

                EndTextureMode();
                DrawTextureRec(screen.texture, (Rectangle){0, 0,
screenProps.screenWidth, -screenProps.screenHeight}, (Vector2){0, 0}, WHITE);
                EndDrawing();
                break;

            case 3:
                // Game state
                static int wasInHomeMenu = 0;
```

```
                // If transitioning from the home menu, reset the game state
                if (wasInHomeMenu == 1) {
                    restartGame(); // Reset the game
                    wasInHomeMenu = 0;
                    gameStatus.scoreboardUpdated = false;
                    scoreboard.player1Wins = 0;  // Reset player 1 wins
                    scoreboard.player2Wins = 0;  // Reset player 2 wins
                    scoreboard.draws = 0;        // Reset draws
                }

                drawGame(); // Draw the game board

                // Check if the back button was pressed
                if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
                    Vector2 mousePosition = GetMousePosition();
                    if (CheckCollisionPointTriangle(mousePosition,
                        (Vector2){25, 55}, (Vector2){140, 55}, (Vector2){82.5,
15})) {
                        player.loopState = 0; // Go back to the home screen
                        wasInHomeMenu = 1;
                    }
                }

                // Update the scoreboard if the game has ended
                if (!gameStatus.scoreboardUpdated && (player.whoWon != 0 ||
isBoardFull(gameBoard.board))) {
                    updateScoreboard(&player.whoWon);
                    gameStatus.scoreboardUpdated = true;
                }
                break;
        }
    }
}

void restartGame() {
    // Reset game board to empty state
    for (int i = 0; i < 9; i++) {
        gameBoard.board[i] = 0;
    }

    // Reset game state variables
    player.turn = 1; // Set turn to player 1
    player.numOfTurns = 0; // Reset turn count
    player.whoWon = 0; // Reset winner status
```

```
    gameStatus.showInvalidMsg = 0; // Reset invalid move message
    gameStatus.aiMadeMove = 0; // Reset AI move status

    gameStatus.scoreboardUpdated = false; // Reset scoreboard update status
    // Reset AI-related variables if applicable
}

void handleInput() {
    // Check if left mouse button is pressed, game is not over, and game state
is in play mode
    if (IsMouseButtonPressed(MOUSE_BUTTON_LEFT) && player.whoWon == 0 &&
player.loopState == 3) {
        // Get mouse position
        Vector2 mousePosition = GetMousePosition();
        int x = (int)mousePosition.x;
        int y = (int)mousePosition.y;

        // Check if click is within the game board
        if (x >= screenProps.boardX && x <= (screenProps.boardX + 3 *
screenProps.squareSize) &&
            y >= screenProps.boardY && y <= (screenProps.boardY + 3 *
screenProps.squareSize)) {
            // Calculate the grid cell clicked based on mouse position
            int col = (x - screenProps.boardX) / screenProps.squareSize;
            int row = (y - screenProps.boardY) / screenProps.squareSize;

            // Convert grid cell to game choice (1-9)
            int choice = row * 3 + col + 1;

            // Check if choice is valid and empty, then make the move
            if (choice >= 1 && choice <= 9 && gameBoard.board[choice - 1] == 0)
{
                choice -= 1;
                insertChoice(&choice);
                PlaySound(gameSounds.moveSound); // Play move sound
            } else {
                gameStatus.showInvalidMsg = 1;  // Show invalid move message
            }
        }
    }
}

// Function to update scoreboard based on the winner of the game
void updateScoreboard(int* winner) {
```

```
    // Increment player 2 wins if O won
    if (*winner == 1) {
        scoreboard.player2Wins++;
    }
    // Increment player 1 wins if X won
    else if (*winner == -1) {
        scoreboard.player1Wins++;
    }
    // Increment draws if it's a draw
    else if (*winner == 2) {
        scoreboard.draws++;
    }
    // Mark scoreboard as updated
    gameStatus.scoreboardUpdated = true;
}




// Function to draw the game GUI
void drawGame() {
    BeginDrawing();
    ClearBackground(GRAY);
    // Calculate the board position
    screenProps.boardX = (screenProps.screenWidth - 3 * screenProps.squareSize)
/ 2;
    screenProps.boardY = (screenProps.screenHeight - 3 * screenProps.squareSize)
/ 2;
    const char* resultMessage = "";

    int winningCombo[3] = { -1, -1, -1 };
    player.whoWon = win(gameBoard.board);

    // Draw the back button
    DrawTriangle((Vector2){25, 55}, (Vector2){140, 55}, (Vector2){82.5, 15},
WHITE);
    DrawText("BACK", 55, 35, 20, BLACK);

    // Draw the restart button
    DrawRectangleRounded(restartButton, 0.2, 5, WHITE);
    DrawText("RESTART", screenProps.screenWidth - 98, 35, 15, BLACK);

    // Handle the back button press
    if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
```

```
        Vector2 mousePosition = GetMousePosition();
        if (CheckCollisionPointTriangle(mousePosition,
            (Vector2){25, 55}, (Vector2){140, 55}, (Vector2){82.5, 15})) {
            player.loopState = 0; // Go back to the home screen
            restartGame(); // Reset the game state
        }
    }

    // Handle the restart button press
    if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
        Vector2 mousePosition = GetMousePosition();
        if (CheckCollisionPointTriangle(mousePosition,
            (Vector2){25, 55}, (Vector2){140, 55}, (Vector2){82.5, 15})) {
            player.loopState = 0; // Go back to the home screen
            restartGame(); // Reset the game state
        } else if (CheckCollisionPointRec(mousePosition, restartButton)) {
            restartGame(); // Restart the game
            PlaySound(gameSounds.restartSound); // Play restart sound
        }
    }

    // Draw the scoreboard
    DrawText(TextFormat("Player 1 Wins: %d", scoreboard.player1Wins), 50, 650,
20, WHITE);
    DrawText(TextFormat("Player 2 Wins: %d", scoreboard.player2Wins), 300, 650,
20, WHITE);
    DrawText(TextFormat("Draws: %d", scoreboard.draws), 600, 650, 20, WHITE);



    if (player.whoWon != 0) {
        // Identify the winning combination
        for (int i = 0; i < 8; i++) {
            if (gameBoard.board[gameBoard.winningCombos[i][0]] == player.whoWon
&&
                gameBoard.board[gameBoard.winningCombos[i][1]] == player.whoWon
&&
                gameBoard.board[gameBoard.winningCombos[i][2]] == player.whoWon)
{
                // Store the winning combination
                winningCombo[0] = gameBoard.winningCombos[i][0];
                winningCombo[1] = gameBoard.winningCombos[i][1];
                winningCombo[2] = gameBoard.winningCombos[i][2];
                break;
```

```
            }
        }
    }

    // Draw grid lines
    DrawLineEx((Vector2){screenProps.boardX + screenProps.squareSize,
screenProps.boardY}, (Vector2){screenProps.boardX + screenProps.squareSize,
screenProps.boardY + 3 * screenProps.squareSize}, 10, WHITE);
    DrawLineEx((Vector2){screenProps.boardX + 2 * screenProps.squareSize,
screenProps.boardY}, (Vector2){screenProps.boardX + 2 * screenProps.squareSize,
screenProps.boardY + 3 * screenProps.squareSize}, 10, WHITE);
    DrawLineEx((Vector2){screenProps.boardX, screenProps.boardY +
screenProps.squareSize}, (Vector2){screenProps.boardX + 3 *
screenProps.squareSize, screenProps.boardY + screenProps.squareSize}, 10,
WHITE);
    DrawLineEx((Vector2){screenProps.boardX, screenProps.boardY + 2 *
screenProps.squareSize}, (Vector2){screenProps.boardX + 3 *
screenProps.squareSize, screenProps.boardY + 2 * screenProps.squareSize}, 10,
WHITE);

    // Adjust the positions for drawing X and O symbols within the cells
    for (int i = 0; i < 9; i++) {
        int row = i / 3;
        int col = i % 3;
        int cellX = screenProps.boardX + col * screenProps.squareSize;
        int cellY = screenProps.boardY + row * screenProps.squareSize;
        if (gameBoard.board[i] == 1) {
            // Draw O in the cell
            DrawCircle(cellX + screenProps.squareSize / 2, cellY +
screenProps.squareSize / 2, screenProps.squareSize / 2 - 20, WHITE);
        } else if (gameBoard.board[i] == -1) {
            // Draw X in the cell
            DrawLineEx((Vector2){cellX + 20, cellY + 20}, (Vector2){cellX +
screenProps.squareSize - 20, cellY + screenProps.squareSize - 20}, 10, WHITE);
            DrawLineEx((Vector2){cellX + screenProps.squareSize - 20, cellY +
20}, (Vector2){cellX + 20, cellY + screenProps.squareSize - 20}, 10, WHITE);
        }
    }



    // Determine game result and set appropriate message
    player.whoWon = win(gameBoard.board);
    if (player.whoWon == -1) {
        resultMessage = "PLAYER 1 (X) WINS, PRESS R TO RESTART!";
```

```
        gameStatus.showRestartMessage = 1;
    } else if (player.whoWon == 1 && player.gamemodeChoice == 2) {
        resultMessage = "PLAYER 2 (O) WINS, PRESS R TO RESTART!";
        gameStatus.showRestartMessage = 1;
    } else if (player.whoWon == 1 && player.gamemodeChoice == 1) {
        // Different messages for different AI types
        if (player.aiType == 1) {
            resultMessage = "PERFECT MINIMAX AI WINS, PRESS R TO RESTART!";
        } else if (player.aiType == 2) {
            resultMessage = "IMPERFECT MINIMAX AI WINS, PRESS R TO RESTART!";
        } else if (player.aiType == 3) {
            resultMessage = "EASY MODE AI WINS, PRESS R TO RESTART!";
        } else if (player.aiType == 4) {
            resultMessage = "MACHINE LEARNING AI WINS, PRESS R TO RESTART!";
        }
        gameStatus.showRestartMessage = 1;
    } else if (isBoardFull(gameBoard.board)) {
        // Message for draw
        resultMessage = "DRAW, PRESS R TO RESTART!!!";
        gameStatus.showRestartMessage = 1;
        player.whoWon = 2; // Signify Draw
    }

    // Show invalid move message if needed
    if (gameStatus.showInvalidMsg == 1) {
        resultMessage = "Please select an empty box!";
    }
    // Calculate the width and center position of the result message
    int messageWidth = MeasureText(resultMessage, 25);
    int centerX = (screenProps.screenWidth - messageWidth) / 2;
    int bottomY = screenProps.screenHeight - 50; // Adjust this value as needed

    // Draw the result message centered at the bottom
    DrawText(resultMessage, centerX, bottomY, 25, WHITE);



    // Display player's turn or AI thinking message
    if (player.turn == 1) {
        DrawText("Player 1 (X)'s turn...", 250, 50, 30, WHITE);
    } else if (player.turn == 2 && player.gamemodeChoice == 2) {
        DrawText("Player 2 (O)'s turn...", 250, 50, 30, WHITE);
    } else {
        DrawText("Computer is thinking...", 250, 50, 30, WHITE);
```

```
        // AI's turn (if no winner yet, then AI makes a move)
        if (player.whoWon == 0) {
            gameStatus.aiMadeMove = makeAIMove(&player.aiType);


        }
    }

    // Check for user input to restart or exit the game only when the game has
ended
    if (gameStatus.showRestartMessage == 1) {
        if (IsKeyPressed(KEY_R)) {
            restartGame(); // Restart the game
            PlaySound(gameSounds.restartSound); // Play restart sound
        } else if (IsKeyPressed(KEY_ESCAPE)) {
            CloseWindow(); // Close the game window
        }
    }

    int lineThickness = 5; // Set the thickness of the strike-through line

    // Draw the strike-through effect if there is a win
    if (player.whoWon != 0) {
        int cellSize = screenProps.squareSize - 20; // Adjust the size of the
strike-through line
        if (winningCombo[0] != -1 && winningCombo[1] != -1 && winningCombo[2] !=
-1) {
            // Calculate the start and end points of the strike-through line
            int startX = screenProps.boardX + (winningCombo[0] % 3) *
screenProps.squareSize + screenProps.squareSize / 2;
            int startY = screenProps.boardY + (winningCombo[0] / 3) *
screenProps.squareSize + screenProps.squareSize / 2;
            int endX = screenProps.boardX + (winningCombo[2] % 3) *
screenProps.squareSize + screenProps.squareSize / 2;
            int endY = screenProps.boardY + (winningCombo[2] / 3) *
screenProps.squareSize + screenProps.squareSize / 2;
            Color lineColor = BLACK; // Set the color for the line
            // Draw the strike-through line
            DrawLineEx((Vector2){startX, startY}, (Vector2){endX, endY},
lineThickness, lineColor);
        }
    }

    EndDrawing(); // End the drawing
```

```
}
```

tictactoeLogic.c

```c
#include "tictactoe.h"
#include <math.h>
#include <string.h>

int bestmoveindex;
int random;

// Function to check if a button has been clicked
int checkButton(Rectangle* buttons) {
    // Get the current mouse position
    Vector2 mousePos = GetMousePosition();

    // Loop through each button
    for (int i = 0; i < 5; i++)
    {
        // Check if the mouse position collides with the button rectangle
        if (CheckCollisionPointRec(mousePos, buttons[i]))
            return i;  // Return the index of the button that was clicked
    }

    // If no button was clicked, return -1
    return -1;
}

// Function to check if the board is full
int isBoardFull(const int* board) {
    for (int i = 0; i < 9; i++) {
        if (board[i] == 0) {
            return 0; // There's an empty space, the board is not full
        }
    }
    return 1; // All spaces are filled
}


// Function to implement the minimax algorithm with imperfect decision making
int minimaxImperfect(int* board, int comp, int depth) {
    // Check if the game has been won
    int winner = win(board);

    // If the game has been won, return the winner
    if (winner != 0) {
```

```
        return winner * comp;
    }

    // If the maximum depth has been reached, return 0
    if (depth == 0) {
        return 0;  // Heuristic evaluation here
    }

    // Initialize the best move and the best score
    int move = -1;
    int score = -2; // Losing moves are preferred to no move

    // Loop through each position on the board
    for (int i = 0; i < 9; ++i) {
        // If the position is empty
        if (board[i] == 0) {
            // Try the move
            board[i] = comp;
            // Recursively call minimaxImperfect to evaluate the move
            int thisScore = -minimaxImperfect(board, comp * -1, depth - 1);
            // Undo the move
            board[i] = 0;

            // If the move is better than the current best move, update the best
move and the best score
            if (thisScore > score) {
                score = thisScore;
                move = i;
            }
        }
    }

    // If no move has been found, return 0
    if (move == -1) {
        return 0;
    }

    // Return the score of the best move
    return score;
}

int minimax(int* board, int comp) {
    int winner = win(board);  // Check for winner
    if(winner != 0) return winner*comp;  // If game over, return winner
```

```
    int move = -1;
    int score = -2;  // Losing moves are preferred to no move
    int i;
    for(i = 0; i < 9; ++i) {  // For all moves,
        if(board[i] == 0) {  // If legal,
            board[i] = comp;  // Try the move
            int thisScore = -minimax(board, comp*-1);  // Recurse
            if(thisScore > score) {  // Pick the one that's worst for the
opponent
                score = thisScore;
                move = i;
            }
            board[i] = 0;  // Reset board after try
        }
    }
    if(move == -1) return 0;  // If no move, return 0
    return score;  // Return score of best move
}

// Function to train the model
void train(Example* examples, int numExamples, FeatureCount counts[9][6], int
label[2]) {
    // Loop through each example
    for (int i = 0; i < numExamples; i++) {
        if (examples[i].target[0] == 'p')
        {
            label[0] = label[0] + 1;
        }
        else if (examples[i].target[0] == 'n')
        {
            label[1] = label[1] + 1;
        }
        // Loop through each feature in the example
        for (int j = 0; j < 9; j++) {
            char feature = examples[i].features[j];
            char target = examples[i].target[0];

            // Update the count of each feature-target pair
            for (int k = 0; k < 6; k++) {
                if (counts[j][k].feature == feature && counts[j][k].target ==
target) {
                    counts[j][k].count++;
                    break;
```

```
                }
            }
        }
    }
}

double learn(FeatureCount counts[9][6], ProbabilityCount probability[9][6]){

    for (int k = 0; k < 9; k++)
    {
        for (int i = 0; i < 6; i++)
        {
            if (probability[k][i].target == 'p')
            {
                probability[k][i].probability = (counts[k][i].count +
1.0)/(label[0]+3.0) ;
            }
            else if (probability[k][i].target == 'n')
            {
                probability[k][i].probability = (counts[k][i].count +
1.0)/(label[1]+3.0) ;
            }

            printf("Grid: %d, Feature: %c, Target: %c, Probability: %lf\n", k+1,
probability[k][i].feature, probability[k][i].target,
probability[k][i].probability);
        }
    }

}

int readExamplesFromFile(const char* filename, Example** examples) {
    FILE* file = fopen("tic-tac-toe.data", "r");  // Open the file
    if (file == NULL) {  // Check if file opened successfully
        fprintf(stderr, "Error opening file: %s\n", filename);
        return 0;
    }

    int numExamples = 0;
    char line[100];
    while (fgets(line, sizeof(line), file) != NULL) {  // Count lines in the
file
        numExamples++;
    }
```

```
    *examples = malloc(numExamples * sizeof(Example));  // Allocate memory for
examples

    rewind(file);  // Rewind the file to the beginning

    // Read examples from the file
    for (int i = 0; i < numExamples; i++) {
        if (fscanf(file, "%c,%c,%c,%c,%c,%c,%c,%c,%c,%s\n",
                   &(*examples)[i].features[0], &(*examples)[i].features[1],
&(*examples)[i].features[2],
                   &(*examples)[i].features[3], &(*examples)[i].features[4],
&(*examples)[i].features[5],
                   &(*examples)[i].features[6], &(*examples)[i].features[7],
&(*examples)[i].features[8],
                   (*examples)[i].target) != 10) {  // Check if reading was
successful
            fprintf(stderr, "Error reading from file: %s\n", filename);
            free(*examples);  // Free allocated memory
            fclose(file);  // Close the file
            return 0;
        }
    }

    fclose(file);  // Close the file
    return numExamples;  // Return the number of examples
}

void splitDataset(Example* allExamples, int numExamples, Example** trainingSet,
int* numTraining, Example** testingSet, int* numTesting, double splitRatio) {
    // Shuffle the examples randomly
    for (int i = numExamples - 1; i > 0; i--) {
        int j = rand() % (i + 1);

        // Swap examples[i] and examples[j]
        Example temp = allExamples[i];
        allExamples[i] = allExamples[j];
        allExamples[j] = temp;
    }

    // Calculate the number of examples for training and testing
    int numTrainingExamples = (int)(numExamples * splitRatio);
    int numTestingExamples = numExamples - numTrainingExamples;
```

```
    // Allocate memory for training set
    *trainingSet = malloc(numTrainingExamples * sizeof(Example));

    // Copy examples for training set
    for (int i = 0; i < numTrainingExamples; i++) {
        (*trainingSet)[i] = allExamples[i];
    }

    // Allocate memory for testing set
    *testingSet = malloc(numTestingExamples * sizeof(Example));

    // Copy examples for testing set
    for (int i = numTrainingExamples; i < numExamples; i++) {
        (*testingSet)[i - numTrainingExamples] = allExamples[i];
    }

    // Set the number of training and testing examples
    *numTraining = numTrainingExamples;
    *numTesting = numTestingExamples;
}


// Function to check win status
int win(const int* board) {
    // Winning combinations
    unsigned wins[8][3] =
{{0,1,2},{3,4,5},{6,7,8},{0,3,6},{1,4,7},{2,5,8},{0,4,8},{2,4,6}};
    int i;
    for(i = 0; i < 8; ++i) {
        // If any winning combination is found, return the winner (1 or 2)
        if(board[wins[i][0]] != 0 &&
           board[wins[i][0]] == board[wins[i][1]] &&
           board[wins[i][0]] == board[wins[i][2]])
            return board[wins[i][0]];
    }
    // If no winner, return 0
    return 0;
}

// Function to swap turns
int swapTurn(int* currentTurn){
    if(*currentTurn == 1){
        *currentTurn = 2;   // If it's X's turn, make it O's turn
    }else{
```

```
        *currentTurn = 1;   // If it's O's turn, make it X's turn
    }

    return 0;
}

// Function to check if a grid cell already has input
int checkInput(int* choice){
    if (gameBoard.board[*choice] != 0) {
        return 1;   // If cell is not empty, return 1
    } else {
        return 0;   // If cell is empty, return 0
    }
}

// Function to insert player's choice onto the game board
int insertChoice(int* choice){
    gameStatus.validInputFlag = checkInput(choice);  // Check if the chosen cell
is empty

    if(gameStatus.validInputFlag == 1){
        gameStatus.showInvalidMsg = 1;  // If cell is not empty, set flag to
show invalid message
    }else{
        gameStatus.showInvalidMsg = 0;  // If cell is empty, reset flag to not
show invalid message
        if(player.turn == 1){
            gameBoard.board[*choice] = -1;  // If it's player 1's turn, mark the
cell with -1
            swapTurn(&player.turn);  // Swap turns
            player.numOfTurns += 1;  // Increment the number of turns
        }else{
            gameBoard.board[*choice]= 1;  // If it's player 2's turn, mark the
cell with 1
            swapTurn(&player.turn);  // Swap turns
            player.numOfTurns += 1;  // Increment the number of turns
        }
    }
    return 0;
}

void currentfeatures(char features[9], int numcounts)
{
    predict(features,probability,numcounts);
```

```
}

char predict(char* features, ProbabilityCount probability[9][6], int numcounts)
{
    double maxPossibilityP;
    double maxPossibilityN;
    double possibility[2] =
{(double)(label[0])/(double)(label[0]+label[1]),(double)(label[1])/(double)(labe
l[0]+label[1])};

    //printf("\nPossibility of p\n");
    for (int k = 0; k < 9; k++)
    {
        for (int i = 0; i < 6; i++)
        {
            if (features[k] == probability[k][i].feature &&
probability[k][i].target == 'p')
            {
                //printf("\nFeature of current state: %c, Feature in db: %c,
Probability is: %lf, possibility is: %lf\n",
features[k],probability[k][i].feature,probability[k][i].probability,
possibility[0]);
                possibility[0] = possibility[0] * probability[k][i].probability;
            }
        }
    }

    //printf("\n\n");
    //printf("\nPossibility of n\n");

    for (int k = 0; k < 9; k++)
    {
        for (int i = 0; i < 6; i++)
        {
            if (features[k] == probability[k][i].feature &&
probability[k][i].target == 'n')
            {
                //printf("\nFeature of current state: %c, Feature in db: %c,
Probability is: %lf, possibility is: %lf\n",
features[k],probability[k][i].feature,probability[k][i].probability,
possibility[1]);
                possibility[1] = possibility[1] * probability[k][i].probability;
            }
        }
```

```
    }

    //printf("\n\n");



    if (possibility[0] > maxPossibilityP)
    {
        maxPossibilityP = possibility[0];
        //printf("The features is: %s",features);
        //printf("The index is: %d",bestmoveindex);
    }

    if (possibility[1] > maxPossibilityN)
    {
        maxPossibilityN = possibility[1];
    }

    if (possibility[0] > possibility[1] && possibility[1] == maxPossibilityN) //
Choosing best move
    {
        //printf("\nAI is able to choose from:");
        //printf("\nThe features that AI can choose: %s",features);
        bestmoveindex = numcounts;
        //printf("\nThe index that AI can choose: %d",bestmoveindex);
    }
    else if (possibility[1] > possibility[0])
    {
        random = 1;
        //printf("\nAI is making a random move");
    }

    //printf("\npossibility of p = %lf\n", possibility[0]);
    //printf("\npossibility of n = %lf\n", possibility[1]);

    //printf("\nThe greatest possibility for p: %lf\n",maxPossibilityP);
    //printf("\nThe greatest possibility for n: %lf\n",maxPossibilityN);

    return (possibility[0] > possibility[1]) ? 'p' : 'n';
}

int naiveBayes(int board[9], FeatureCount counts[9][6], int numCounts, int
numExamples) {
    // Generate features from the current state of the board
```

```
//int row, col;
char features[9];

for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
        if (board[i * 3 + j] == 1)
        {
            features[i * 3 + j] = 'o';
        }
        else if (board[i * 3 + j] == 0)
        {
            features[i * 3 + j] = 'b';
        }
        else if (board[i * 3 + j] == -1)
        {
            features[i * 3 + j] = 'x';
        }
        //printf("%d ", board[i * 3 + j]);
    }
    printf("\n");
}
printf("\n\n");
currentfeatures(features,numCounts);

int move;
int emptyCells[9];  // Array to store indices of empty cells
int numEmptyCells = 0;  // Initialize the number of empty cells

// Iterate through each cell on the game board
for (int l = 0; l < 9; ++l) {
    if (gameBoard.board[l] == 0) {
        // If the cell is empty, add its index to the array of empty cells
        emptyCells[numEmptyCells++] = l;
    }
}

// Check if there are any empty cells
if (numEmptyCells > 0) {
    // If there are empty cells, choose a random index from the array
    for (int i = 0; i < numEmptyCells; ++i) {
    if (emptyCells[i] == bestmoveindex) {
        // If bestmoveindex is in the array, set move to bestmoveindex
        move = bestmoveindex;
```

```
            break; // Exit the loop once the move is set
        }
        else if (random == 1)
        {
            move = emptyCells[rand() % numEmptyCells];
        }
    }
    }
    return move;
}

// Function for AI to make a move
int makeAIMove(int* aiType){
    int move = -1;  // Initialize best move
    int bestScore = -2;  // Initialize best score
    for (int i = 0; i < 9; ++i) {  // For each cell on the board
        if (gameBoard.board[i] == 0) {  // If the cell is empty
            gameBoard.board[i] = 1;  // Assume AI is 'O'

            // Calculate score based on AI type
            if (*aiType == 1) {  // Perfect AI
                int score = -minimax(gameBoard.board, -1);
                if (score > bestScore) {  // If score is better than best score
                    bestScore = score;  // Update best score
                    move = i;  // Update best move
                }
            } else if (*aiType == 2) {  // Imperfect AI Medium
                int score = -minimaxImperfect(gameBoard.board, -1, 2);
                if (score > bestScore) {
                    bestScore = score;
                    move = i;
                }
            } else if (*aiType == 3) {  // Imperfect AI Easy
                int score = -minimaxImperfect(gameBoard.board, -1, 0);
                if (score > bestScore) {
                    bestScore = score;
                    move = i;
                }
            } else if (*aiType == 4) {  // Naive Bayes AI
                move = naiveBayes(gameBoard.board, counts, i, numTraining);
            }

            gameBoard.board[i] = 0;  // Reset cell after game
        }
```

```
    }
    insertChoice(&move);  // Make the best move
}

void testingAccuracy(Example* testingSet, int numTesting, ProbabilityCount
probability[9][6], int numCounts) {
    int correct = 0;

    Example* p_testingSet = testingSet;
    int confusionMatrix[2][2] = {{0, 0}, {0, 0}};

    char* predictions = malloc(numTesting * sizeof(int));
    char* actual = malloc(numTesting * sizeof(int));

    for (int i = 0; i < numTesting; i++) {
        char predicted = predict(p_testingSet[i].features, probability, 0);
        if (predicted == p_testingSet[i].target[0]) {
            correct++;
        }
        predictions[i] = predicted;
        actual[i] = p_testingSet[i].target[0];

    }

    for (int i = 0; i < numTesting; i++) {
    if (actual[i] == 'p' && predictions[i] == 'p') {
        confusionMatrix[0][0]++; // TP
    } else if (actual[i] == 'p' && predictions[i] == 'n') {
        confusionMatrix[0][1]++; // FN
    } else if (actual[i] == 'n' && predictions[i] == 'p') {
        confusionMatrix[1][0]++; // FP
    } else if (actual[i] == 'n' && predictions[i] == 'n') {
        confusionMatrix[1][1]++; // TN
    }

    }

    printf("\nAccuracy: %f", (double)correct / numTesting);
    printf("\nConfusion Matrix:\n");
    printf("\t\tPredicted: YES\tPredicted: NO\n");
    printf("Actual: YES\t%d\t\t%d\n", confusionMatrix[0][0],
confusionMatrix[0][1]);
    printf("Actual: NO \t%d\t\t%d\n", confusionMatrix[1][0],
confusionMatrix[1][1]);
```

```
    free(predictions);
    free(actual);

}
```

10. References

[1]I. Rish, “An empirical study of the naive Bayes classifier.” Available: https://sites.cc.gatech.edu/home/isbell/classes/reading/papers/Rish.pdf

[2]“A Novel Bayes Model: Hidden Naive Bayes | IEEE Journals & Magazine | IEEE Xplore,” *ieeexplore.ieee.org*. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4721435&casa_token=pF1DBw0DQAoAAAAA:pTq31lukCt0wXYx81Leymu3-DUhWNqLkoO51XfaXiR7OELgxE1ZLmCT2GMlYMvGtUdlN7z2vTI5m&tag=1

[3]C.-H. Chou, “Using Tic-Tac-Toe for Learning Data Mining Classifications and Evaluations,” *International Journal of Information and Education Technology*, pp. 437–441, 2013, doi: https://doi.org/10.7763/ijiet.2013.v3.314.

[4]O. Caelen, “A Bayesian interpretation of the confusion matrix,” *Annals of Mathematics and Artificial Intelligence*, vol. 81, no. 3–4, pp. 429–450, Sep. 2017, doi: https://doi.org/10.1007/s10472-017-9564-8.

[5]“Opleiding Informatica.” Accessed: Nov. 25, 2023. [Online]. Available: https://theses.liacs.nl/pdf/2022-2023-WongKM.pdf

[6]S Akshay, B. Shreyas Bhargav, and J. Amudha, “iGAME: Cognitive Game Analysis Through Eye Movements of the Player,” *Lecture Notes in Computer Science*, pp. 279–288, Jan. 2023, doi: https://doi.org/10.1007/978-3-031-45170-6_29.

[7]“Shakow 1 An investigation into the relationship between cognitive abilities and the theoretical accuracy of a prompted second move choice in Tic-Tac-Toe measured by the Trail-Making test Matthew Shakow Saratoga Springs High School.” Accessed: Nov. 25, 2023. [Online]. Available: https://www.gcrsef.org/wp-content/uploads/formidable/13/SaratogaSpringsHS.MatthewShakow.LohnesRobinson-1.pdf

11. Resources

[1]“Making Tic-Tac-Toe in Raylib with C,” *www.youtube.com*. https://www.youtube.com/watch?v=ki5l_It9Q9Y

[2]“C Tic Tac Toe game ⭕,” *www.youtube.com*. https://www.youtube.com/watch?v=_889aB2D1KI&t=564s

[3]“Naive Bayes, Clearly Explained!!!,” *www.youtube.com*. https://www.youtube.com/watch?v=O2L2Uv9pdDA&t=820s

[4]“Pointer to Structure Variable,” *www.youtube.com*. https://www.youtube.com/watch?v=VsnXsfNstVw

[5]“Confusion Matrix Solved Example Accuracy Precision Recall F1 Score Prevalence by Mahesh Huddar,” *www.youtube.com*. https://www.youtube.com/watch?v=_CGTbkHwUHQ

[6]“Getting Started with Raylib - C Tutorial,” *www.youtube.com*. https://www.youtube.com/watch?v=-F6THkPkF2I

[7]“Using gnuplot from C,” *Stack Overflow*. https://stackoverflow.com/questions/44839126/using-gnuplot-from-c

[8]“How to add Raylib to VS code?,” *Stack Overflow*. https://stackoverflow.com/questions/63791456/how-to-add-raylib-to-vs-code/75957745#75957745

[9]“Tic Tac Toe in C,” *Stack Overflow*. https://stackoverflow.com/questions/75076628/tic-tac-toe-in-c

[10]“raylib - cheatsheet,” *raylib*. https://www.raylib.com/cheatsheet/cheatsheet.html

[11]T. Williams *et al.*, “gnuplot 5.4 An Interactive Plotting Program,” 1993. Available: http://www.gnuplot.info/docs_5.4/Gnuplot_5_4.pdf

--- END ---