University of Glasgow | School of Computing Science

Course : CSC 1103 Programming Methodology Mini-Project 2023

Title : xxx

Lab Session : P2

Group : 03

Name	GUID	SIT ID	Scope	Contribution %
CHRISTIAN ANGELO LAU CANDELARIO	2957847C	2303383	GUI (Task 1) 2 Player Mode (Task 2) ML Algo (Task 4b) Documentation	100
DAN LAI KAI YI	2957891L	2300875	2 Player Mode (Task 2) Minimax (Perfect) (Task 3) Minimax (Imperfect) (Task 4a) Documentation	100
LOH QINGKAI, BENJAMIN	2957936L	2301019	GUI (Task 1) Minimax (Perfect) (Task 3) ML Algo (Task 4b) Documentation	100
MUHAMMAD SYAHMI BIN SAMRI	2957843B	2301125	GUI (Task 1) 2 Player Mode (Task 2) Minimax (Imperfect) (Task 4a) Documentation	100
TAY YONG JUN	2957980T	2300993	Minimax (Perfect) (Task 3) Minimax (Imperfect) (Task 4a) ML Algo (Task 4b) Documentation	100

Problem Definition

Develop a tic tac toe C application with the primary goal of creating an interesting platform for youngsters to develop critical thinking abilities through gameplay. This application has two separate modes: a two-player version that encourages interactive participation among participants, and a single-player game with three difficulty levels—easy, medium, and hard. The application uses GTK with recursive algorithms and array structures to create an immersive and educational Tic-Tac-Toe experience for children. The ultimate goal is to promote cognitive development while keeping a professional and pedagogically focused software design.

Problem Analysis

The program is designed to give users the choice of partaking in a multiplayer mode, where they can compete against another player, or a single-player variant, where the challenge is presented by the computer. When you pick multiplayer mode, the game starts right away. In contrast, before starting the game in single-player mode, users are given the option to pick the preferred difficulty level.

In the context of playing against a computer opponent, two distinct strategies are under consideration: the Minimax algorithm and Machine Learning (ML). As a result of its recursive structure, the Minimax algorithm carefully analyzes every potential outcome in order to identify the best move. It assesses whether a winner has already emerged and, if so, returns a score, considering the depth of the game. In the absence of a clear winner, the algorithm strategically makes a move and recursively calls itself. The returned scores are then compared within the current depth, and the optimal move for that depth is selected.

The exciting possibility of revolutionizing the game is presented by the Machine Learning subset of artificial intelligence. Supervised learning is a key paradigm within this approach, where algorithms get trained on labelled datasets to establish correlations between input data and output labels. Consequently, these algorithms acquire knowledge to make predictions about novel and unseen data. Machine Learning is a cutting-edge technology that has been successfully applied in various domains like natural language processing, image and speech recognition, and recommendation systems.

Input Variable

I. Game mode (Singleplayer or Multiplayer)
II. Difficulty (Easy, Medium or Hard) (Only available for Singleplayer)
III. Algorithm (Minimax or Machine Learning)
IV. Player’s move

Process Variable

I. Whose turn (player 1 or player 2/Computer)
II. Player’s move (To check if it is valid and empty)
III. Winning condition
IV. Minimax’ score (Which move has the best score will be selected as computer move)
V. ML

Output Variable

I. User move on the 3x3 grid
II. Computer’s move on the 3x3 grid
III. Board’s result (Win, Lose, Tie)

Algorithm

Algorithm for main()
1. Set board_grid[9] = {‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’}
2. Set MAX_LINES = 958
3. Set positive = 0
4. Set negative = 0
5. Initialise count_array[9][6]
6. Initialise probability_array[9][6]
7. Initialise gamemode
8. Read user input
9. Set gamemode from user input
10. IF (gamemode == “Singleplayer”)
    - 10.1. Initialise difficulty
    - 10.2. Initialise algorithm
    - 10.3. Read user input
    - 10.4. Set difficulty from user input
    - 10.5. Set algorithm from user input
    - 10.6. Run singleplayer(difficulty, algorithm)
11. ELSE IF (gamemode == “Multiplayer”)
    - 11.1. Run multiplayer()

Algorithm for printboard(board)
1. Prints 3x3 tic tac toe board with number 1 to 9, or ‘X’, or ‘O’

Algorithm for winning_condition(board)
1. Check if there is a winner
2. IF winner == ‘X’
    - 2.1. Return -1
3. ELSE
    - 3.1. Return 1
4. IF there is no winner
    - 4.1. Return 0

Algorithm for player_move(player)
1. Initialise board
2. Point board to board_grid[9] address
3. Initialise playerMove
4. Read user input
5. Set playerMove from user input
6. IF (board[playerMove] != ‘X’ or board[playerMove] != ‘O’)
    - 6.1. IF player == 1
        - 6.1.1. Set board[playerMove] == ‘X’
    - 6.2. ELSE
        - 6.2.1. Set board[playerMove] == ‘O’
7. ELSE
    - 7.1. Print the string “Slot is taken! Please select another number!”

Algorithm for multiplayer()
1. Initialise board
2. Point board to board_grid[9] address
3. Initialise win
4. Set player = 1
5. Set count = 0
6. WHILE (count < 9)
    - 6.1. Run player_move(player)
    - 6.2. Run printboard(board)
    - 6.3. Run winning_condition(board)
    - 6.4. Set win from result of winning_condition
    - 6.5. IF (win == -1)

```
            6.5.1.  Print the string “Player 1 Wins!”
            6.5.2.  Break the loop
        6.6.    ELSE IF (win == 1)
            6.6.1.  Print the string “Player 2 Wins!”
            6.6.2.  Break the loop
        6.7.    count++
        6.8.    player *= -1
    7.  IF (count == 8)
        7.1.    Print the string “It’s a tie!”
        7.2.    IF (algorithm == “ML” && difficulty == “MEDIUM)
            7.2.1.  negative++
            7.2.2.  For i = 0 to 9 do the following
                7.2.2.1.    IF (board[i] == ‘X’)
                    7.2.2.1.1.  count[i][3] += 1
                7.2.2.2.    ELSE IF (board[i] == ‘O’)
                    7.2.2.2.1.  count[i][4] += 1
                7.2.2.3.    ELSE
                    7.2.2.3.1.  count[i][5] += 1
```

```
Algorithm for singleplayer(difficulty, algorithm)
  1.  Initialise board
  2.  Point board to board_grid[9] address
  3.  Set count = 0
  4.  WHILE (count < 9)
      4.1.  IF (count % 2 == 0)
          4.1.1.  Run player_move(1)
          4.1.2.  Run printboard(board)
          4.1.3.  count++
      4.2.  ELSE
          4.2.1.  IF ((difficulty == “EASY”) || (difficulty == “MEDIUM” && count < 2 &&
                  algorithm == “Minimax”))
              4.2.1.1.    WHILE (TRUE)
                  4.2.1.1.1.  Set computerMove to random integer 0 to 8
                  4.2.1.1.2.  IF (board[computerMove] != ‘X’ &&
                              board[computerMove] != ‘O’ && computerMove >= 0
                              && computerMove < 9)
                          4.2.1.1.2.1.    Set board[computerMove] = ‘O’
                          4.2.1.1.2.2.    Break the loop
          4.2.2.  ELSE
              4.2.2.1.    IF (algorithm == “ML”)
                  4.2.2.1.1.  Set total_pprobability = 1
                  4.2.2.1.2.  Set total_nprobability = 1
                  4.2.2.1.3.  Set current_probability = -1
                  4.2.2.1.4.  Set move = 0
                  4.2.2.1.5.  Initialise sign
                  4.2.2.1.6.  Initialise int_str[5]
                  4.2.2.1.7.  Initialise symbol
                  4.2.2.1.8.
                  4.2.2.1.9.  For i = 0 to 9 do the following
                          4.2.2.1.9.1.    IF (board[i] == ‘X’)
                                  4.2.2.1.9.1.1.  total_pprobability *=
                                                  probability_array[i][0]
                          4.2.2.1.9.2.    ELSE IF (board[i] == ‘O’)
                                  4.2.2.1.9.2.1.  total_nprobability *=
                                                  probability_array[i][4]
                  4.2.2.1.10.  IF (total_pprobability > total_nprobability)
                          4.2.2.1.10.1.   Set symbol = ‘X’
                          4.2.2.1.10.2.   Set sign = -1
                  4.2.2.1.11.  ELSE
                          4.2.2.1.11.1.   Set symbol = ‘O’
                          4.2.2.1.11.2.   Set sign = 1
```

4.2.2.1.12. For i = 0 to 9 do the following
4.2.2.1.12.1. Set probability = sign
4.2.2.1.12.2. IF (board[i] != 'X' && board[i] != 'O')
4.2.2.1.12.2.1. Set board[i] = symbol
4.2.2.1.12.2.2. probability *= probability_array[i][0] IF symbol == 'X' ELSE probability_array[i][4]
4.2.2.1.12.2.3. probability *= total_pprobability IF symbol == 'X' ELSE total_nprobability
4.2.2.1.12.2.4. IF (symbol == 'X')
4.2.2.1.12.2.4.1. IF (probability < current_probability)
4.2.2.1.12.2.4.1.1. Set current_probability = probability
4.2.2.1.12.2.4.1.2. Set move = i
4.2.2.1.12.2.5. ELSE
4.2.2.1.12.2.5.1. IF (probability > current_probability)
4.2.2.1.12.2.5.1.1. Set current_probability = probability
4.2.2.1.12.2.5.1.2. Set move = i
4.2.2.1.13. Set board[move] = 'O'
4.2.2.2. ELSE
4.2.2.2.1. Set move = -1
4.2.2.2.2. Set score = -100
4.2.2.2.3. For i = 0 to 9 do the following
4.2.2.2.3.1. IF (board[i] != 'X' && board[i] != 'O')
4.2.2.2.3.1.1. Initialise tempScore
4.2.2.2.3.1.2. Set board[i] = 'O'
4.2.2.2.3.1.3. Set tempScore from result of -minimax(board, -1, 100 - count)
4.2.2.2.3.1.4. Reset the board to original state
4.2.2.2.3.1.5. IF (tempScore > score)
4.2.2.2.3.1.5.1. Set score = tempScore
4.2.2.2.3.1.5.2. Set move = i
4.2.2.2.4. Set board[move] = 'O'
4.2.2.3. Run printboard(board)
4.2.2.4. count++
4.2.2.5. Set win from result of winning_condition(board)
4.2.2.6. IF (win == -1)
4.2.2.6.1. Print the string "Player 1 Wins!"
4.2.2.6.2. IF (algorithm == "ML" && difficulty == "MEDIUM)
4.2.2.6.2.1. positive++
4.2.2.6.2.2. For i = 0 to 9 do the following
4.2.2.6.2.2.1. IF (board[i] == 'X')
4.2.2.6.2.2.1.1. count[i][0] += 1
4.2.2.6.2.2.2. ELSE IF (board[i] == 'O')
4.2.2.6.2.2.2.1. count[i][1] += 1
4.2.2.6.2.2.3. ELSE
4.2.2.6.2.2.3.1. count[i][2] += 1
4.2.2.7. ELSE IF (win == 1)
4.2.2.7.1. Print the string "Computer Wins!"
4.2.2.7.2. IF (algorithm == "ML" && difficulty == "MEDIUM)
4.2.2.7.2.1. negative++
4.2.2.7.2.2. For i = 0 to 9 do the following
4.2.2.7.2.2.1. IF (board[i] == 'X')
4.2.2.7.2.2.1.1. count[i][3] += 1
4.2.2.7.2.2.2. ELSE IF (board[i] == 'O')
4.2.2.7.2.2.2.1. count[i][4] += 1
4.2.2.7.2.2.3. ELSE
4.2.2.7.2.2.3.1. count[i][5] += 1

Algorithm for minimax(board. player, depth)

1. Set move = -1
2. Set score = -100
3. Set win from result of winning_condition(board)
4. IF (winner != 0)
    4.1. Return winner * player * depth
5. IF (player == -1)
    5.1. Set symbol = ‘X’
6. ELSE
    6.1. Set symbol = ‘O’
7. For i = 0 to 9 do the following
    7.1. Set currentDepth = depth
    7.2. IF (board[i] != ‘X’ && board[i] != ‘O’)
        7.2.1. Set board[i] = symbol
        7.2.2. Set thisScore = -minimax(board, player * -1), currentDepth - 1)
        7.2.3. IF (thisScore > score)
            7.2.3.1. Set score = thisScore
            7.2.3.2. Set move = i
        7.2.4. Reset the board to original state
8. IF (move == -1)
    8.1. Return 0
9. Return score

Algorithm for swap_lines(data)

1. Set seed for Random Number Generator
2. For x = 0 to MAX_LINES do the following
    2.1. Set i to random integer 0 to x + 1
    2.2. Set temp to lines[x]
    2.3. Set lines[x] to lines[i]
    2.4. Set lines[i] to temp

Algorithm for readFile(data)

1. Open file
2. IF (file == NULL)
    2.1. Print the string “Error opening file”
    2.2. Exit code
3. For i = 0 to MAX_LINES do the following
    3.1. IF (Set line in file to data[i])
        3.1.1. Replace “\n” of each line to “\0” in data[i]
4. Close file

Algorithm for training_data(data, training_dataset)

1. For i = 0 to training_dataset do the following
    1.1. Set current_line = data[i]
    1.2. Set label to end = current_line
    1.3. For x = 0 to 17 do the following with increment of x by 2
        1.3.1. IF (label == “positive”)
            1.3.1.1. positive++
            1.3.1.2. IF (current_line[x] == ‘x’)
                1.3.1.2.1. count_array[x / 2][0] += 1
            1.3.1.3. ELSE IF (current_line[x] == ‘o’)
                1.3.1.3.1. count_array[x / 2][1] += 1
            1.3.1.4. ELSE IF (current_line[x] == ‘b’)
                1.3.1.4.1. count_array[x / 2][2] += 1
        1.3.2. ELSE IF (label == “negative”)
            1.3.2.1. negative++
            1.3.2.2. IF (current_line[x] == ‘x’)
                1.3.2.2.1. count_array[x / 2][3] += 1
            1.3.2.3. ELSE IF (current_line[x] == ‘o’)
                1.3.2.3.1. count_array[x / 2][4] += 1
            1.3.2.4. ELSE IF (current_line[x] == ‘b’)

```
                    1.3.2.4.1.    count_array[x / 2][5] += 1
2.    For x = 0 to 9 do the following
      2.1.    For y = 0 to 6 do the following
          2.1.1.    IF (y < 3)
              2.1.1.1.    Set probability_array[x][y] = count_array[x][y] / positive
          2.1.2.    ELSE
              2.1.2.1.    Set probability_array[x][y] = count_array[x][y] / negative
```

```
Algorithm for testing_data(data, testing_dataset)
1.    Set tp = 0
2.    Set fp = 0
3.    Set tn = 0
4.    Set fn = 0
5.    For i = (MAX_LINES - testing_dataset) to MAX_LINES do the following
      5.1.    Set pprobability = positive / (positive + negative)
      5.2.    Set nprobability = negative / (positive + negative)
      5.3.    Set current_line = data[i]
      5.4.    Set label = end of current_line
      5.5.    For x = 0 to 17 do the following with increment of x by 2
          5.5.1.    IF (current_line[x] == 'x')
              5.5.1.1.    pprobability *= probability_array[x / 2][0]
              5.5.1.2.    pprobability *= probability_array[x / 2][3]
          5.5.2.    ELSE IF (current_line[x] == 'o')
                  5.5.2.1.1.    pprobability *= probability_array[x / 2][1]
                  5.5.2.1.2.    pprobability *= probability_array[x / 2][4]
          5.5.3.    IF (current_line[x] == 'b')
              5.5.3.1.    pprobability *= probability_array[x / 2][2]
              5.5.3.2.    pprobability *= probability_array[x / 2][5]
      5.6.    IF (pprobability > nprobability)
          5.6.1.    IF (label == "positive)
              5.6.1.1.    tp++
          5.6.2.    ELSE
              5.6.2.1.    fp++
      5.7.    ELSE
          5.7.1.    IF (label == "negative")
              5.7.1.1.    tn++
          5.7.2.    ELSE
              5.7.2.1.    fn++
6.    Print the string "Error: ", ((fp + fn) / (tp + fp + tn + fn)) * 100
7.    Print the string "True Positive: ", tp
8.    Print the string "False Positive: ", fp
9.    Print the string "True Negative: ", np
10.   Print the string "False Negative: ", nf
11.   Print the string "Accuracy: ", ((tp + tn) / (tp + fp + tn + fn)) * 100
```

Pseudocode

```
BEGIN
   MAX_LINES ← 958
   positive ← 0
   negative ← 0
   count_array[9][6]
   probability_array[9][6]
   board_grid[9] ← {'', '', '', '', '', '', '', '', ''}
   READ gamemode
   IF gamemode == "Singleplayer"
      READ difficulty
      singleplayer(difficulty)
```

```
   ELSE IF gamemode == "Multiplayer"
      multiplayer()
   END IF
END

FUNCTION printboard(board)
   board       refTochar → &board_grid[9]

   PRINT "\n     |     |     "
   PRINT "\n   board[0]  |  board[1]  |  board[2]  "
   PRINT "\n_______|_______|_______"
   PRINT "\n     |     |     "
   PRINT "\n   board[3]  |  board[4]  |  board[5]  "
   PRINT "\n_______|_______|_______"
   PRINT "\n     |     |     "
   PRINT "\n   board[6]  |  board[7]  |  board[8]  "
   PRINT "\n     |     |     "

END FUNCTION

FUNCTION winning_condition(board)
   win[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}}

   FOR i = 0 TO 8 DO
      IF ((board[win[i][0]] == 'X' || board[win[i][0]] == 'O') &&
         board[win[i][0]] == board[win[i][1]] &&
         board[win[i][1]] == board[win[i][2]])
         IF (board[win[i][0]] == 'X')
            RETURN -1
         ELSE IF (board[win[i][0]] == 'O')
            RETURN 1
         END IF
      END IF
   END FOR

   RETURN 0
END FUNCTION

FUNCTION player_move(player)
   board       refTochar → &board_grid[9]
   PRINT "\nPlease select [1-9]"
   READ playerMove
   IF (board[playerMove] != 'X' || board[playerMove] != 'O')

      IF (player == 1)
         symbol ← 'X'
      ELSE
         symbol ← 'O'
      END IF
      board[playerMove] ← symbol
   ELSE
      PRINT "\nSlot is taken! Please select another number!"
   END IF
END FUNCTION

FUNCTION multiplayer()
   board       refTochar → &board_grid[9]
   player ← 1
   count ← 0

   WHILE (count < 9)
```

```
      player_move(player)
      printboard(board)
      win ← winning_condition(board)
      IF (win == 1)
         PRINT "\nPlayer 1 Wins!"
         BREAK
      ELSE IF (win == -1)
         PRINT "\nPlayer 2 Wins!"
         BREAK
      END IF
      count ← count + 1
      player ← player * -1
   END WHILE

   IF (count == 8)
      PRINT "\nIt's a tie!"
   END IF
END FUNCTION

FUNCTION singleplayer(difficulty)
   board      refTochar → &board_grid[9]
   count ← 0
   WHILE (count < 9)
      IF (count % 2 == 0)
         player_move(player)
         printboard(board)
         count ← count + 1
      ELSE
         IF ((difficulty == "EASY") || (difficulty == "MEDIUM" && count < 2 && algorithm ==
"MINIMAX"))
            WHILE TRUE
               computerMove ← RANDOM_INT(0, 8)
               IF (board[computerMove] != 'X' && board[computerMove] != 'O' &&
computerMove >= 0 && computerMove < 9)
                  board[computerMove] ← 'O'
                  BREAK
               END IF
            END WHILE
         ELSE
            IF (algorithm == "ML")
               total_pprobability ← 1
               total_nprobability ← 1
               current_probability ← 1

               FOR i = 0 TO 9 DO
                  IF (board[i] == 'X')
                     total_pprobability ← total_pprobability * probability_array[i][0]
                  ELSE IF (board[i] == 'O')
                     total_nprobability ← total_nprobability * probability_array[i][4]
                  END IF
               END FOR

               IF (total_pprobability > total_nprobability)
                  symbol ← 'X'
                  sign ← -1
               ELSE
                  symbol ← 'O'
                  sign ← 1
               END IF

               FOR i = 0 TO 9 DO
```

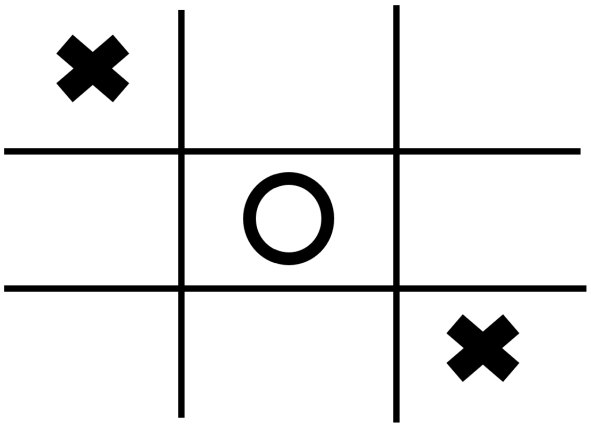
```
                probability ← sign
                IF (board[i] != 'X' && board[i] != 'O')
                   board[i] ← symbol
                   probability ← symbol == 'X' ? probability * probability_array[i][0] :
probability * probability_array[i][4]
                   probability ← symbol == 'X' ? probability * total_pprobability : probability *
total_nprobability
                IF (symbol == 'X')
                   IF (probability < current_probability)
                      current_probability ← probability
                      move ← i
                   END IF
                ELSE
                   IF (probability > current_probability)
                      current_probability ← probability
                      move ← i
                   END IF
                END IF
                   PRINT "Probability for move : ", i + 1, probability
                   board[i] ← string(i)
                END IF
             END FOR

             PRINT "Best move: ", move + 1
             board[move] ← 'O'
          ELSE
             move ← -1
             score ← -100
             FOR i = 0 TO 9 DO
                IF (board[i] != 'X' && board[i] != 'O')
                   board[i] ← 'O'
                   tempScore ← -minimax(board, -1, 100 - count)
                   board[i] ← string(i)
                   IF (tempScore > score)
                      score ← tempScore
                      move ← i
                   END IF
                END IF
             END FOR

             board[move] ← 'O'
          END IF
       END IF
    END IF
    printboard(board)
    count ← count + 1
    win ← winning_condition(board)
    IF (win == 1)
       PRINT "\nPlayer 1 Wins!"
       IF (algorithm == “ML” && difficulty == “MEDIUM”)
          positive ← positive + 1
          FOR i = 0 TO 9 DO
             IF (board[i] == ‘X’)
                count_array[i][0] ← count_array[i][0] + 1
             ELSE IF (board[i] == ‘O’)
                count_array[i][1] ← count_array[i][1] + 1
             ELSE
                count_array[i][2] ← count_array[i][2] + 1
             END IF
          END FOR
       END IF
```

```
            BREAK
        ELSE IF (win == -1)
            PRINT "\nComputer Wins!"
            IF (algorithm == “ML” && difficulty == “MEDIUM”)
                negative ← negative + 1
                FOR i = 0 TO 9 DO
                    IF (board[i] == ‘X’)
                        count_array[i][0] ← count_array[i][3] + 1
                    ELSE IF (board[i] == ‘O’)
                        count_array[i][1] ← count_array[i][4] + 1
                    ELSE
                        count_array[i][2] ← count_array[i][5] + 1
                    END IF
                END FOR
            END IF
            BREAK
        END IF
    END IF
END WHILE

FUNCTION minimax(board, player, depth)
    move ← -1
    score ← -100
    win ← winning_condition(board)
    IF (winner != 0)
        RETURN winner * player * depth
    END IF

    IF (player == -1)
        symbol ← 'X'
    ELSE
        symbol ← 'O'
    END IF

    FOR i = 0 TO 9 DO
        currentDepth ← depth
        IF (board[i] != 'X' && board[i] != 'O')
            board[i] ← symbol
            thisScore ← -minimax(board, player * -1, currentDepth - 1)
            IF (thisScore > score)
                score ← thisScore
                move ← i
            END IF
            board[i] ← string(i)
        END IF
    END FOR

    IF (move == -1)
        RETURN 0
    END IF

    RETURN score
END FUNCTION

FUNCTION swap_lines(lines)
    SET seed for Random Number Generator

    FOR x = 0 TO MAX_LINES DO
        i ← RANDOM_INT(0, x + 1)
        temp ← lines[x]
        lines[x] ← line[i]
```

```
      line[i] ← temp
   END FOR
END FUNCTION

FUNCTION readFile(data)
   OPEN file

   IF (file == NULL)
      PRINT "Error opening file"
      EXIT CODE
   END IF

   FOR i = 0 TO MAX_LINES DO
      IF (SET line in file to data[i])
         REPLACE "\n" TO "\0" in data[i]
      END IF
   END FOR

   CLOSE file
END FUNCTION

FUNCTION training_data(data, training_dataset)
   FOR i = 0 TO training_dataset DO
      current_line ← data[i]

      label ← GET end of current_line

      FOR x = 0 TO 17 INCREMENT BY 2 AND DO
         IF (label == "positive")
            positive ← positive + 1
            IF (current_line[x] == 'x')
               count_array[x / 2][0] ← count_array[x / 2][0] + 1
            ELSE IF (current_line[x] == 'o')
               count_array[x / 2][1] ← count_array[x / 2][1] + 1
            ELSE IF (current_line[x] == 'b')
               count_array[x / 2][2] ← count_array[x / 2][2] + 1
            END IF
         ELSE IF (label == "negative")
            negative ← negative + 1
            IF (current_line[x] == 'x')
               count_array[x / 2][3] ← count_array[x / 2][3] + 1
            ELSE IF (current_line[x] == 'o')
               count_array[x / 2][4] ← count_array[x / 2][4] + 1
            ELSE IF (current_line[x] == 'b')
               count_array[x / 2][5] ← count_array[x / 2][5] + 1
            END IF
         END IF
      END FOR
   END FOR

   FOR x = 0 TO 9 DO
      FOR y = 0 TO 6 DO
         IF (y < 3)
            probability_array[x][y] ← count_array[x][y] / positive
         ELSE
            probability_array[x][y] ← count_array[x][y] / negative
         END IF
      END FOR
   END FOR
END FUNCTION
```

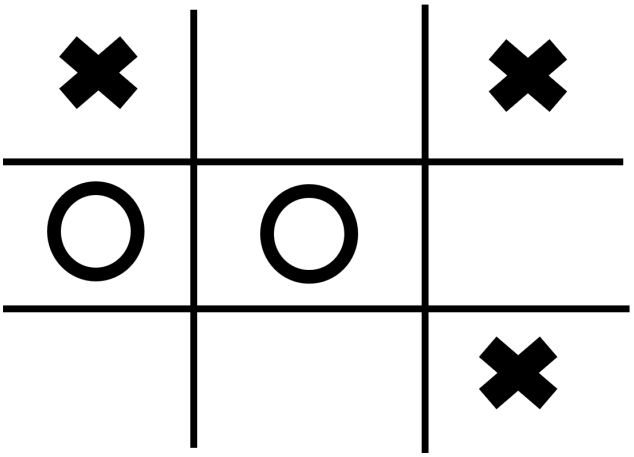
```
FUNCTION testing_data(data, testing_dataset)
   tp ← 0
   fp ← 0
   tn ← 0
   fn ← 0

   FOR i = (MAX_LINES - testing_dataset) TO MAX_LINES DO
      pprobability ← positive / (positive + negative)
      nprobability ← negative / (positive + negative)

      current_line ← data[i]

      label ← GET end of current_line

      FOR x = 0 TO 17 INCREMENT BY 2 DO
         IF (current_line[x] == 'x')
            pprobability ← pprobability * probability_array[x / 2][0]
            nprobability ← nprobability * probability_array[x / 2][3]
         ELSE IF (current_line[x] == 'o')
            pprobability ← pprobability * probability_array[x / 2][1]
            nprobability ← nprobability * probability_array[x / 2][4]
         ELSE IF (current_line[x] == 'b')
            pprobability ← pprobability * probability_array[x / 2][2]
            nprobability ← nprobability * probability_array[x / 2][5]
         END IF
      END FOR

      IF (pprobability > nprobability)
         IF (label == "positive")
            tp ← tp + 1
         ELSE
            fp ← fp + 1
         END IF
      ELSE
         IF (label == "negative")
            tn ← tn + 1
         ELSE
            fn ← fn + 1
         END IF
      END IF
   END FOR

   PRINT "Error: ", ((fp + fn) / (tp + fp + tn + fn)) * 100
   PRINT "True Positive: ", tp
   PRINT "False Positive: ", fp
   PRINT "True Negative: ", np
   PRINT "False Negative: ", nf
   PRINT "Accuracy: ", ((tp + tn) / (tp + fp + tn + fn)) * 100
END FUNCTION
```

Plots and Results



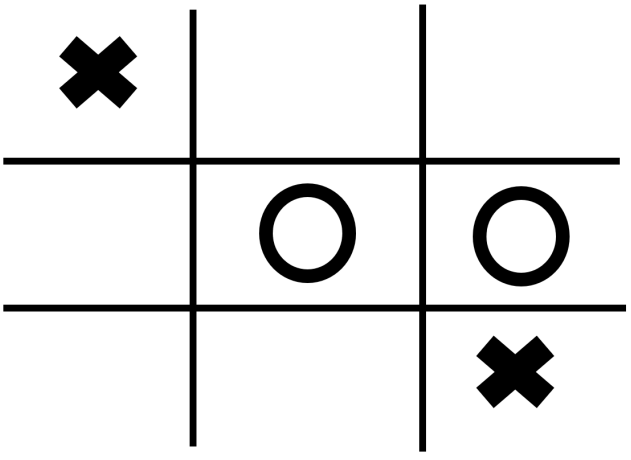
```
True Positive: 105
False Positive: 35
True Negative: 27
False Negative: 25
Accuracy: 68.75
Probability for move 2: -2.68809e-06
Probability for move 3: -3.67681e-06
Probability for move 4: -2.70354e-06
Probability for move 6: -2.67264e-06
Probability for move 7: -3.69226e-06
Probability for move 8: -2.85803e-06
Best move: 6
C:\Users\04jay\OneDrive - Singapore I
True Positive: 98
False Positive: 37
True Negative: 32
False Negative: 25
Accuracy: 67.7083
Probability for move 2: -2.93658e-06
Probability for move 3: -3.9984e-06
Probability for move 4: -2.88681e-06
Probability for move 6: -2.91999e-06
Probability for move 7: -3.89885e-06
Probability for move 8: -3.05272e-06
Best move: 4
C:\Users\04jay\OneDrive - Singapore I
True Positive: 100
False Positive: 42
True Negative: 33
False Negative: 17
Accuracy: 69.2708
Probability for move 2: -2.92588e-06
Probability for move 3: -3.68847e-06
Probability for move 4: -2.8325e-06
Probability for move 6: -2.86362e-06
Probability for move 7: -3.75072e-06
Probability for move 8: -2.73912e-06
Best move: 8
```

Optimized best move is grid 2, 4, 6, or 8



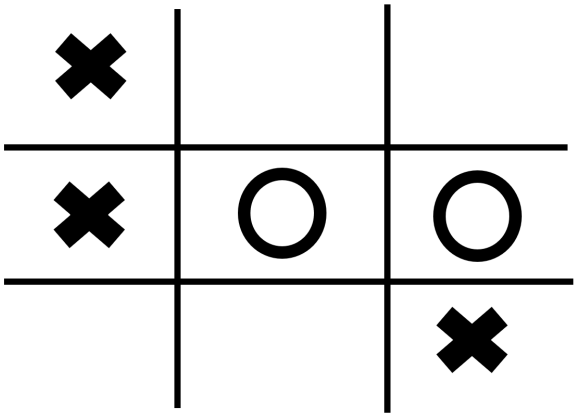
```
True Positive: 106
False Positive: 45
True Negative: 26
False Negative: 15
Accuracy: 68.75
Probability for move 2: -6.39531e-09
Probability for move 6: -6.42969e-09
Probability for move 7: -8.04571e-09
Probability for move 8: -6.25777e-09
Best move: 8
C:\Users\04jay\OneDrive - Singapore I
True Positive: 109
False Positive: 42
True Negative: 33
False Negative: 8
Accuracy: 73.9583
Probability for move 2: -5.84128e-09
Probability for move 6: -6.46831e-09
Probability for move 7: -7.78837e-09
Probability for move 8: -6.03929e-09
Best move: 2
C:\Users\04jay\OneDrive - Singapore I
True Positive: 115
False Positive: 33
True Negative: 27
False Negative: 17
Accuracy: 73.9583
Probability for move 2: -5.86755e-09
Probability for move 6: -5.67304e-09
Probability for move 7: -7.35875e-09
Probability for move 8: -5.89996e-09
Best move: 6
```

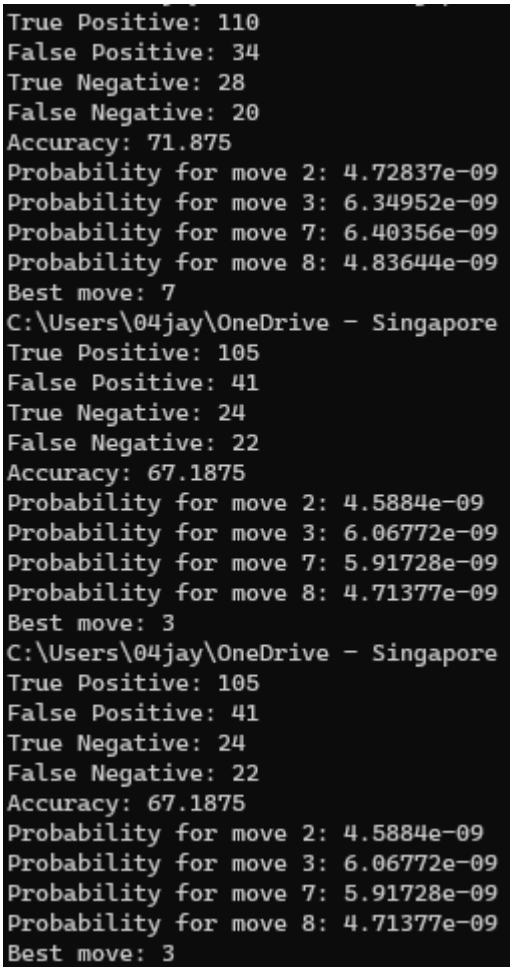
Optimized best move is grid 2, or 6



```
True Positive: 104
False Positive: 43
True Negative: 28
False Negative: 17
Accuracy: 68.75
Probability for move 2: -1.19105e-07
Probability for move 3: -1.44807e-07
Probability for move 4: -1.10329e-07
Probability for move 7: -1.46061e-07
Probability for move 8: -1.1409e-07
Best move: 4
C:\Users\04jay\OneDrive - Singapore I
True Positive: 98
False Positive: 50
True Negative: 31
False Negative: 13
Accuracy: 67.1875
Probability for move 2: -1.14995e-07
Probability for move 3: -1.53122e-07
Probability for move 4: -1.11306e-07
Probability for move 7: -1.48818e-07
Probability for move 8: -1.11921e-07
Best move: 4
C:\Users\04jay\OneDrive - Singapore I
True Positive: 111
False Positive: 39
True Negative: 31
False Negative: 11
Accuracy: 73.9583
Probability for move 2: -1.31688e-07
Probability for move 3: -1.7484e-07
Probability for move 4: -1.39872e-07
Probability for move 7: -1.80792e-07
Probability for move 8: -1.36896e-07
Best move: 2
```

Optimized best move is grid 4, or 7



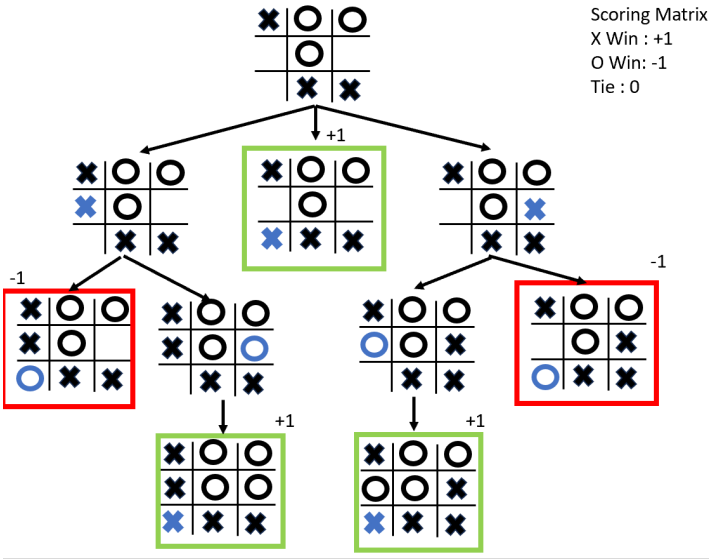


Optimized best move is grid 3, or 7

Minimax

Minimax is a self-calling function used in multiple fields like AI, game theory, etc to reduce the odds of losing for the worst case while maximizing its own odds to win.

For Tic tac toe, the AI will be the maximizer while the human will be the minimizer. The maximizer will aim to achieve the highest score while the minimizer will aim to do the opposite and get the lowest score. Each possible move has a value assigned to it which will lean towards either positive or negative depending on whoever is having the upper hand.



To evaluate how each move scores, we can give each possible scenario a score, e.g. X Win (AI) : +1, O Win (Human) : -1 , Tie : 0. We also can add in depth as another factor to give more weightage if Minimax chooses a move that will allow it to win as fast as possible.

Machine Learning

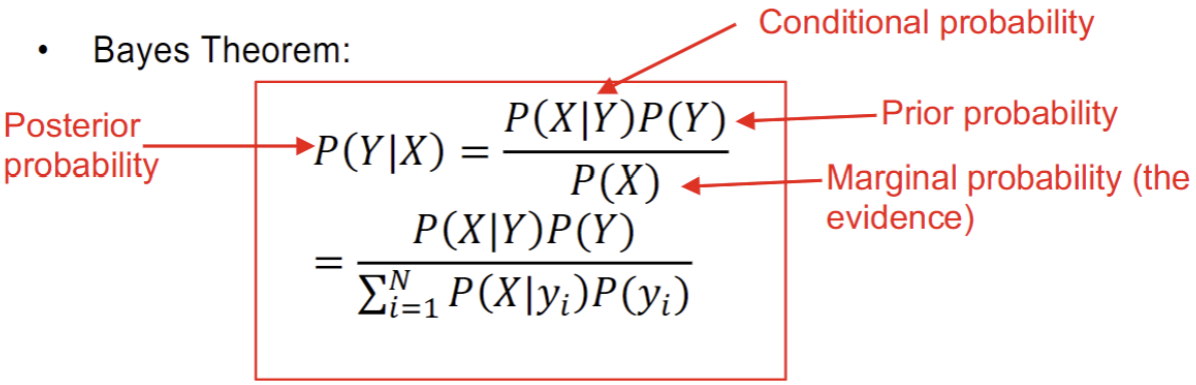
Gradually, machine learning aims to improve its accuracy by utilizing data and algorithms to mimic human learning. A combination of computer science and artificial intelligence, this field centers on the replication of human learning.

The model was trained using a supervised machine learning approach for the tic tac toe, which taught the AI to recognize patterns and make accurate predictions. In order to properly handle classification tasks, the Naïve Bayes algorithm was used due to its ability to quickly adapt to changing datasets and efficiently manage high-dimensional feature spaces. By simplifying the training process and minimizing the risk of overfitting, this algorithm proved to be both effective and straightforward.

- Conditional Probability:

$$P(Y|X) = \frac{P(X,Y)}{P(X)} \qquad P(X|Y) = \frac{P(X,Y)}{P(Y)}$$

- Bayes Theorem:



$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{P(X|Y)P(Y)}{\sum_{i=1}^{N} P(X|y_i)P(y_i)}$$

Minimax vs ML

When comparing the differences between Minimax and ML in terms of execution speed during run time, ML is faster than Minimax by 5.84% on average after multiple rounds of testing.

```
Total time for minimax: 0.001797
Total time for ml: 0.001692
```

Comparison between each level and minimax (perfect vs imperfect) and ML

Two Player Mode:

In two-player mode in Tic Tac Toe, two players take turns placing their assigned symbols ("X" and "O") on a 3x3 grid. The objective is to form a line of three of their symbols horizontally, vertically, or diagonally. Players alternate turns, and the game ends when one player wins or when it's a draw.

Easy (Random):

In the Easy difficulty level, the CPU just chooses a slot for its move using a random number generator regardless of algorithm (Minimax or ML). This method makes the CPU's motions unexpected and requires no strategic thinking or planning. Because of the randomness, both strong and weak plays can occur, making it simpler for a human player to exploit mistakes.

Medium (Imperfect Minimax):

The medium difficulty level introduces a bit of strategy by having the CPU start with a random move before employing the Minimax algorithm. The CPU's initial random move introduces an element of imperfection, making its overall strategy less optimal compared to perfect Minimax. The imperfections may allow human players to capitalize on strategic errors made by the CPU during the game.

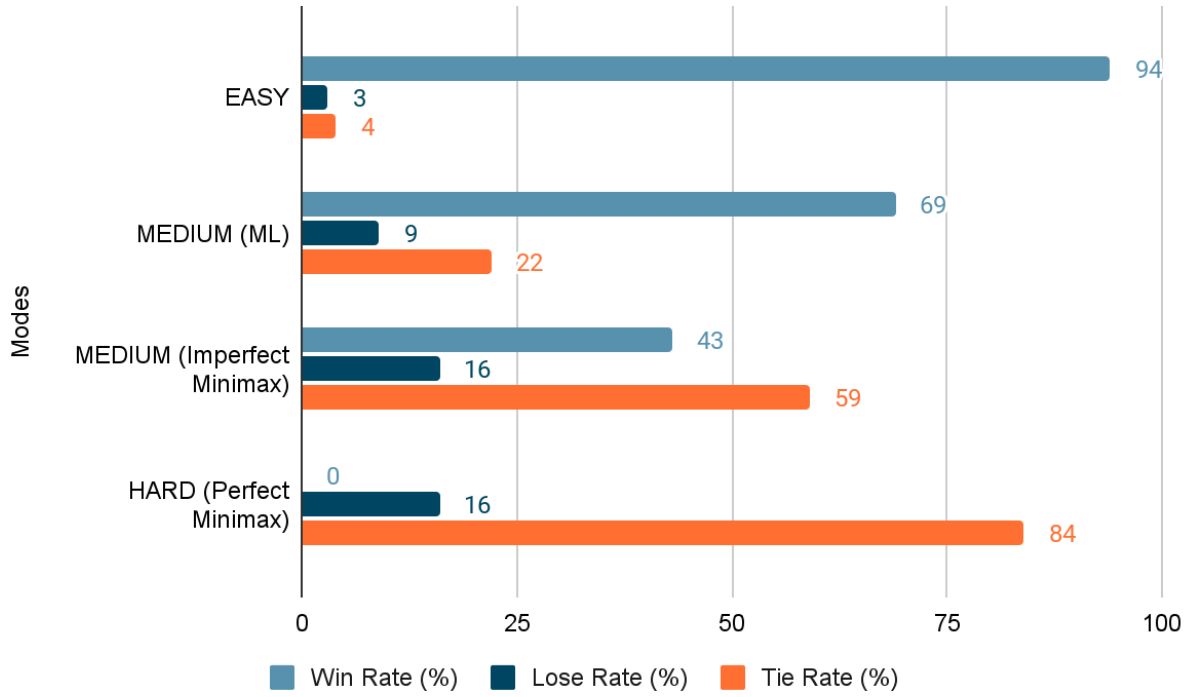
Medium (ML):

The utilization of a machine learning algorithm to train the CPU is suggested by a medium difficulty rating in the realm of machine learning. This may entail gradually enhancing the CPU's capacity through reinforcement learning or educating it on historical game data. The application of ML-based methods may lead to adaptive gaming, in which the CPU modifies its approach in response to its mistakes.

Hard (Perfect Minimax):

The hard difficulty level involves the CPU playing with a perfect implementation of the Minimax algorithm from the start to the end of the game. Perfect Minimax means the CPU will make the most optimal move at every turn, assuming the opponent also plays perfectly. This level of play is difficult for human players since it necessitates a thorough understanding of the game's mechanics and ideal techniques. It minimizes the odds of human players taking advantage of mistakes, making it a difficult challenge.

MODE / PERCENTAGE	EASY	MEDIUM (ML)	MEDIUM (Imperfect Minimax)	HARD (Perfect Minimax)
Win Rate (%)	94 (94/100)	69 (69/100)	43 (43/100)	0 (0/100)
Lose Rate (%)	3 (3/100)	9 (9/100)	16 (16/100)	16 (16/100)
Tie Rate (%)	4 (4/100)	22 (22/100)	59 (59/100)	84 (84/100)

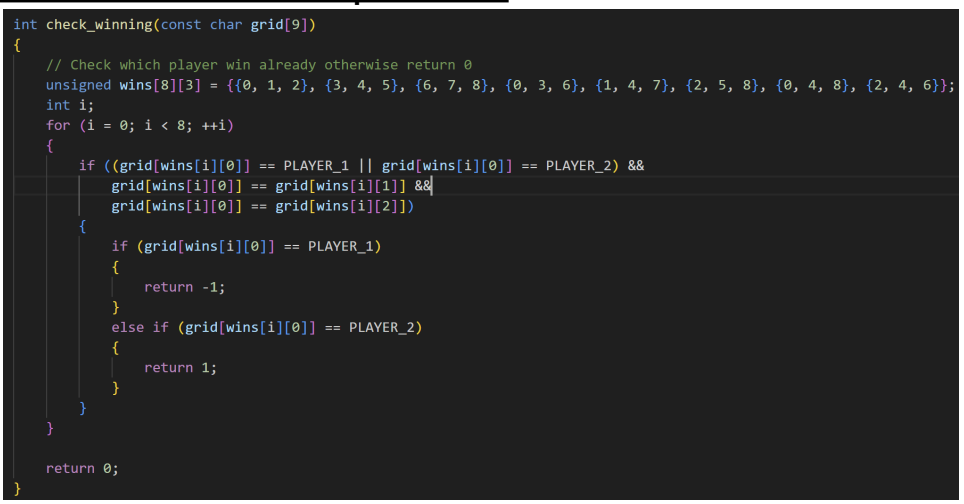


We performed the program modes of Easy, Medium (ML), Medium (imperfect minimax), and Hard (Perfect Minimax) 100 times each in total and tested the individual mode with different persons. The graph above illustrates that the Easy mode has a high victory rate with a very low possibility of computer winning or tying. While Medium (ML) has a lower likelihood of a user winning in comparison to Easy. Following that, we have Medium (Imperfect Minimax). From the graph, you can notice a significant increase in the computer win rate between Medium (ML), indicating that Medium (Imperfect Minimax) is far superior to Medium (ML). Last we have the Hard (Perfect Minimax) which shows that the user is impossible to win and at most the user will only be able to tie.

In summary, the progressive difficulty levels accommodate different phases of learning. The simple and medium levels offer a fun and approachable starting point, but the addition of Minimax and machine learning adds layers of complexity as children develop. The instructional usefulness is in gradually teaching key ideas like randomness, strategy, and artificial intelligence.

Interesting aspects of solution (AKA assumptions, optimisations)

1. **Win Condition Checker Optimisation**



```
int check_winning(const char grid[9])
{
    // Check which player win already otherwise return 0
    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 ((grid[wins[i][0]] == PLAYER_1 || grid[wins[i][0]] == PLAYER_2) &&
            grid[wins[i][0]] == grid[wins[i][1]] &&
            grid[wins[i][0]] == grid[wins[i][2]])
        {
            if (grid[wins[i][0]] == PLAYER_1)
            {
                return -1;
            }
            else if (grid[wins[i][0]] == PLAYER_2)
            {
                return 1;
            }
        }
    }

    return 0;
}
```

- Compact Representation:
  - The use of a 2D array (wins) to store the winning combinations makes the code more compact and easier to read.
- Easy to Extend:
  - Adding or modifying winning conditions is straightforward with this approach. You can easily expand the wins array to accommodate additional winning combinations without significantly altering the existing code.
- Readability:
  - The code benefits from readability due to the clear representation of winning combinations in the wins array. This makes it easier for developers to understand and maintain the code.
- Avoids Redundant Checks:
  - The code checks for a win only if the first cell in a winning combination is occupied by a player (either PLAYER_1 or PLAYER_2). This avoids unnecessary comparisons and improves the efficiency of the win-checking process.
- Scalability:
  - This method scales well with the size of the grid. If you were to use a larger grid (e.g., 4x4 or 5x5), you could extend the wins array accordingly, and the logic would remain concise and manageable.

2. **Minimax (Perfect) Scoring Matrix Assumption**
   - We decided to set the initial score to -100 to ensure that the best move can be chosen.
   - We also decided to incorporate depth as part of the calculation if the game will end with the particular move:
     - Score = winner * player * depth;
     - player = -1 (Player 1), player = 1 (Player 2/CPU)
   - The above allowed us to compare each possibility with a clearer difference and prompt the AI to try to win with as little moves as possible (lower depth).

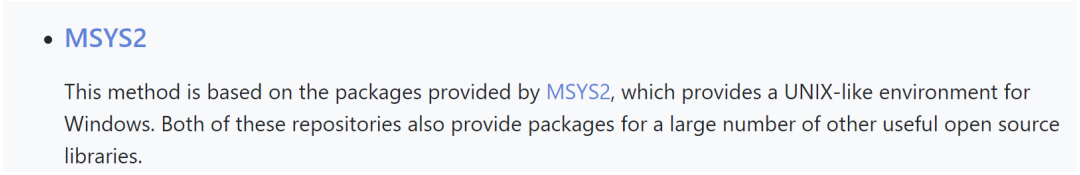
3. **Minimax (Imperfect) Assumption**
   - We decided for the medium difficulty of Minimax (Imperfect) to make the AI

play randomly for the first move before using Minimax.
- This results in the human player to have a higher chance of Winning/tying the AI instead of impossible to win against the AI using Minimax (Perfect).
- This will lead the human player to better develop their skills on Tic Tac Toe.

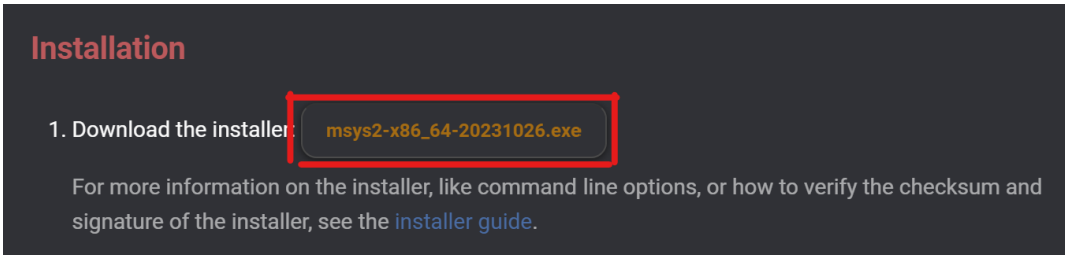
4. **ML Optimisation & Assumption**
   - ML now sees both sides of the game; X's turn and O's turn and predict who has a higher chance of winning the game. If X has a higher chance, ML will choose the move best suitable to stop X from winning. However if O has a higher chance, ML will choose the move that will boost O winning chances.
   - This allows a good increment of difficulty among all available difficulties.
   - Probability of current state are affected by Xs and Os. Blanks are not considered which will result in the ML sometimes not making the optimal moves due to the training data.

How to install GTK 3

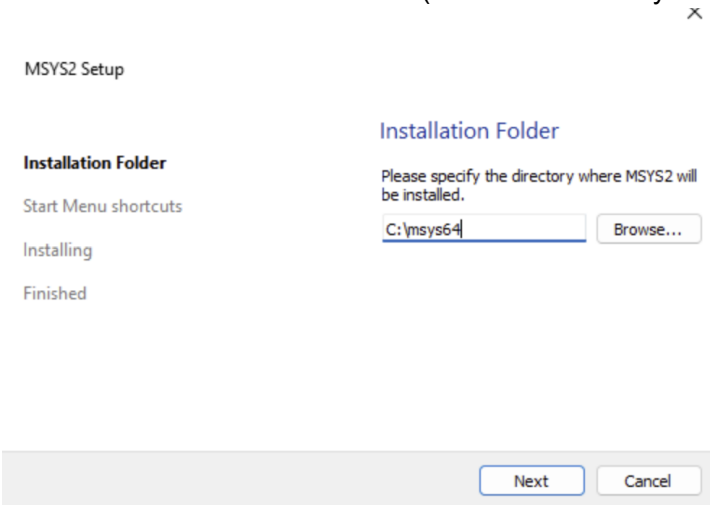
1. We will be using the MSYS2 method to install GTK 3.



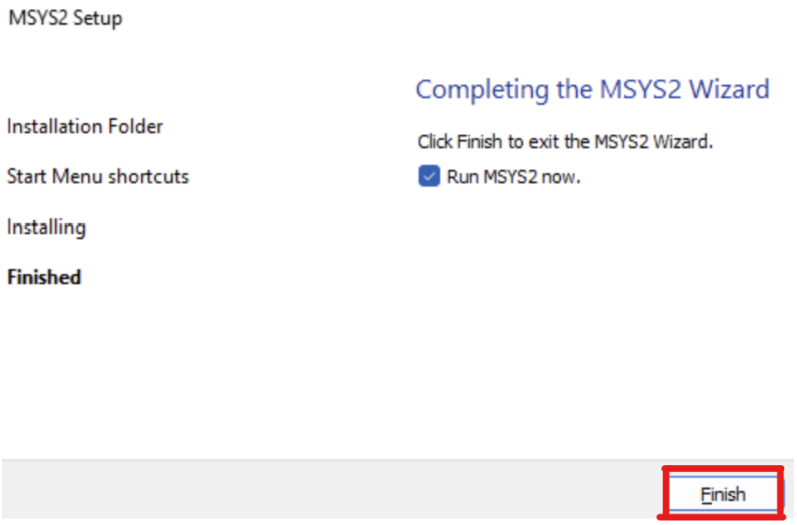
2. Go to MSYS2 website to download the installer.



3. Run the installer. MSYS2 requires 64-bit Windows 8.1 or newer.
4. Enter desired Installation Folder (the default C:\msys64 is good enough)



5. When done, click Finish.



6. In MSYS2, use the command “PACMAN -Syuu” to upgrade all out-of-date packages and refresh all package databases.

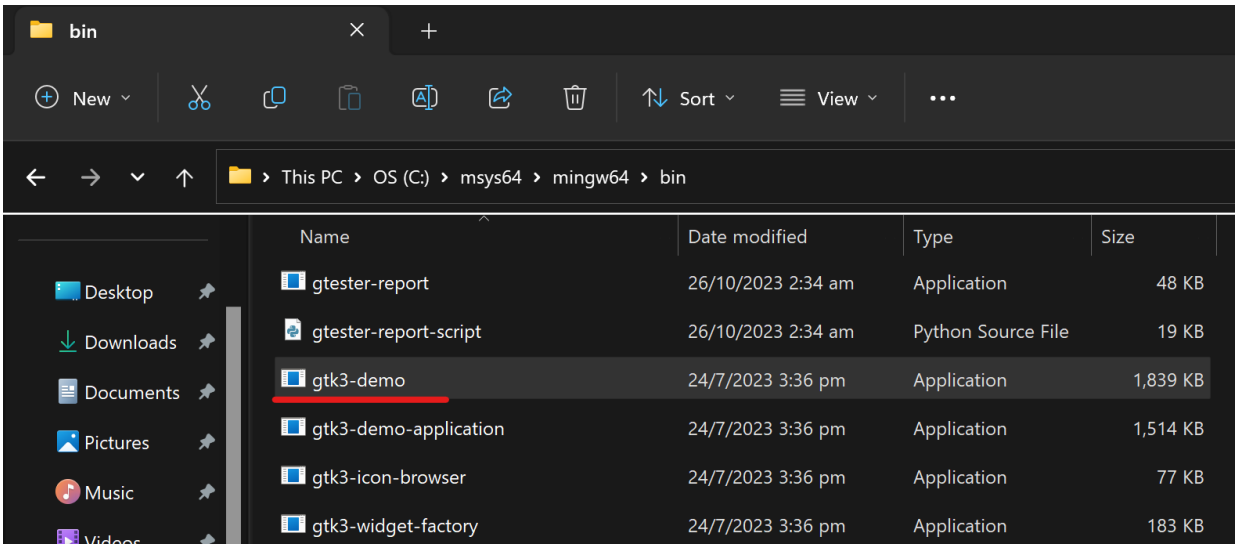
```
User@LAPTOP-JPCH2QLP MSYS ~
$ pacman -Syuu
:: Synchronizing package databases...
 clangarm64 is up to date
 mingw32 is up to date
 mingw64 is up to date
 ucrt64 is up to date
 clang32 is up to date
 clang64 is up to date
 msys is up to date
:: Starting core system upgrade...
 there is nothing to do
:: Starting full system upgrade...
resolving dependencies...
looking for conflicting packages...

Packages (40) gawk-5.3.0-1  info-7.1-1  libgcrypt-1.10.3-1  libgnutls-3.8.2-1  libgpgme-1.23.1-1
              libksba-1.6.5-1  liblzma-5.4.5-1  libnghttp2-1.58.0-1  libopenssl-3.1.4-1
              libp11-kit-0.25.3-1  libsqlite-3.44.0-1  libxml2-2.12.0-1  libxslt-1.1.39-1
              mingw-w64-x86_64-crt-git-11.0.0.r404.g3a137bd87-1  mingw-w64-x86_64-gettext-0.22.3-2
              mingw-w64-x86_64-glib2-2.78.1-1  mingw-w64-x86_64-gst-plugins-bad-libs-1.22.7-1
              mingw-w64-x86_64-gst-plugins-base-1.22.7-1  mingw-w64-x86_64-gstreamer-1.22.7-1
              mingw-w64-x86_64-gtk4-4.12.4-1  mingw-w64-x86_64-harfbuzz-8.3.0-1
              mingw-w64-x86_64-headers-git-11.0.0.r404.g3a137bd87-1
              mingw-w64-x86_64-libmangle-git-11.0.0.r404.g3a137bd87-1
              mingw-w64-x86_64-libwinpthread-git-11.0.0.r404.g3a137bd87-1
              mingw-w64-x86_64-libxml2-2.12.0-1  mingw-w64-x86_64-openssl-3.1.4-1
              mingw-w64-x86_64-python-3.11.6-2  mingw-w64-x86_64-shared-mime-info-2.4-1
              mingw-w64-x86_64-sqlite3-3.44.0-1
              mingw-w64-x86_64-tools-git-11.0.0.r404.g3a137bd87-1
              mingw-w64-x86_64-vulkan-headers-1.3.268-1  mingw-w64-x86_64-vulkan-loader-1.3.268-1
              mingw-w64-x86_64-winpthreads-git-11.0.0.r404.g3a137bd87-1
              mingw-w64-x86_64-winstorecompat-git-11.0.0.r404.g3a137bd87-1
              mingw-w64-x86_64-xz-5.4.5-1  openssl-3.1.4-1  p11-kit-0.25.3-1  texinfo-7.1-1
              texinfo-tex-7.1-1  xz-5.4.5-1

Total Download Size:    78.55 MiB
Total Installed Size:  648.09 MiB
Net Upgrade Size:        9.68 MiB

:: Proceed with installation? [Y/n] y
:: Retrieving packages...
 mingw-w64-x86_64-headers-git-11.0.0.r404.g3a137bd87-1-any                    6.0 MiB
 mingw-w64-x86_64-glib2-2.78.1-1-any                                          3.8 MiB
 mingw-w64-x86_64-crt-git-11.0.0.r404.g3a137bd87-1-any                        3.5 MiB
 mingw-w64-x86_64-gettext-0.22.3-2-any                                        3.3 MiB
 mingw-w64-x86_64-gtk4-4.12.4-1-any                                           9.1 MiB
 mingw-w64-x86_64-sqlite3-3.44.0-1-any                                     1675.6 KiB
 mingw-w64-x86_64-python-3.11.6-2-any                                        23.1 MiB
 libopenssl-3.1.4-1-x86_64                                                 1632.3 KiB
 mingw-w64-x86_64-harfbuzz-8.3.0-1-any                                     1611.1 KiB
```

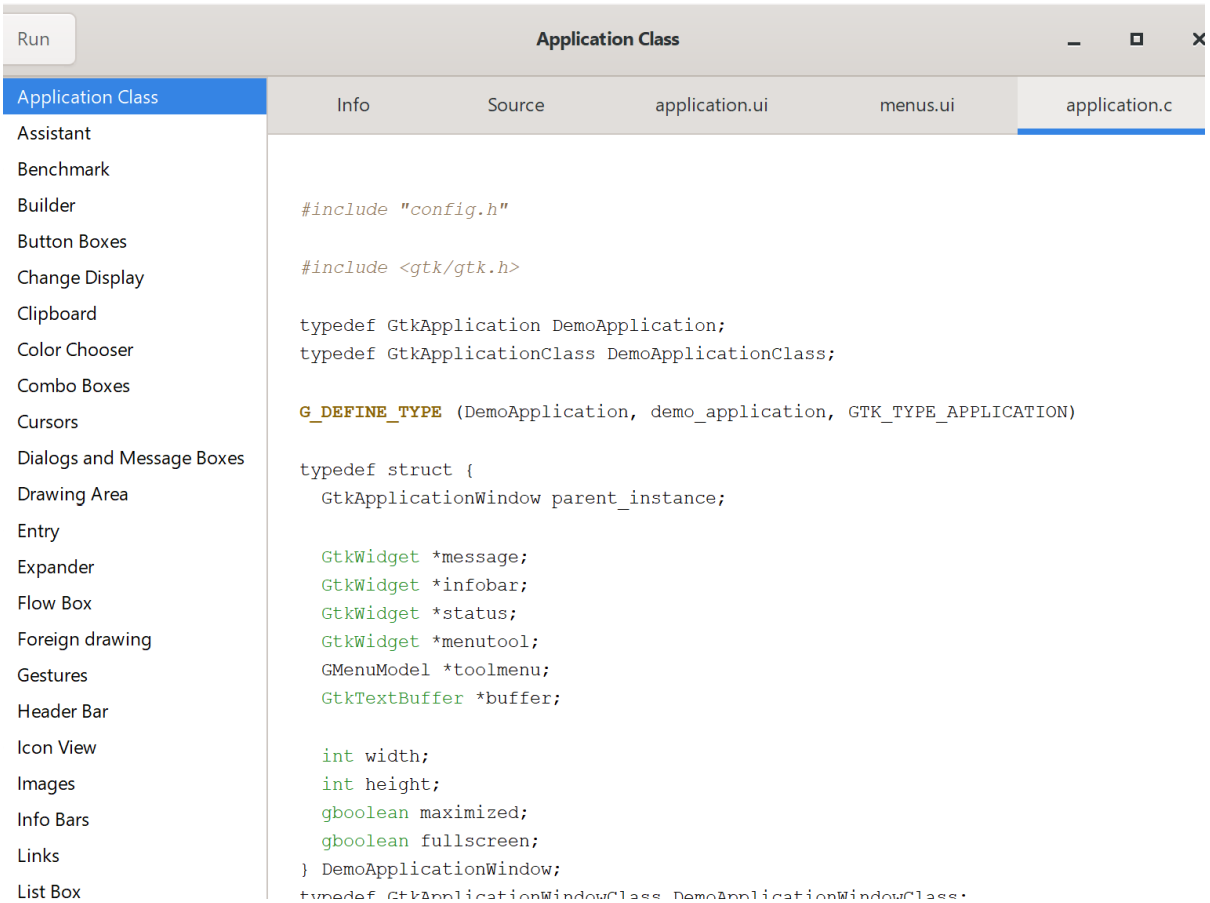
```
:: Processing package changes...
( 1/40) upgrading gawk
( 2/40) upgrading info
( 3/40) upgrading libgcrypt
( 4/40) upgrading libp11-kit
( 5/40) upgrading libgnutls
( 6/40) upgrading libksba
( 7/40) upgrading libsqlite
( 8/40) upgrading libnghttp2
( 9/40) upgrading libopenssl
(10/40) upgrading openssl
(11/40) upgrading p11-kit
(12/40) upgrading liblzma
(13/40) upgrading libxml2
(14/40) upgrading libxslt
(15/40) upgrading libgpgme
(16/40) upgrading mingw-w64-x86_64-headers-git
(17/40) upgrading mingw-w64-x86_64-crt-git
(18/40) upgrading mingw-w64-x86_64-libwinpthread-git
(19/40) upgrading mingw-w64-x86_64-gettext
(20/40) upgrading mingw-w64-x86_64-openssl
(21/40) upgrading mingw-w64-x86_64-sqlite3
(22/40) upgrading mingw-w64-x86_64-xz
(23/40) upgrading mingw-w64-x86_64-python
(24/40) upgrading mingw-w64-x86_64-glib2
(25/40) upgrading mingw-w64-x86_64-libxml2
(26/40) upgrading mingw-w64-x86_64-gstreamer
(27/40) upgrading mingw-w64-x86_64-harfbuzz
(28/40) upgrading mingw-w64-x86_64-gst-plugins-base
(29/40) upgrading mingw-w64-x86_64-vulkan-headers
(30/40) upgrading mingw-w64-x86_64-vulkan-loader
(31/40) upgrading mingw-w64-x86_64-gst-plugins-bad-libs
(32/40) upgrading mingw-w64-x86_64-shared-mime-info
(33/40) upgrading mingw-w64-x86_64-gtk4
(34/40) upgrading mingw-w64-x86_64-libmangle-git
(35/40) upgrading mingw-w64-x86_64-tools-git
(36/40) upgrading mingw-w64-x86_64-winpthreads-git
(37/40) upgrading mingw-w64-x86_64-winstorecompat-git
(38/40) upgrading texinfo
(39/40) upgrading texinfo-tex
(40/40) upgrading xz
:: Running post-transaction hooks...
(1/4) Compiling GSettings XML schema files...
(2/4) Updating icon theme caches...
(3/4) Updating the MIME type database...

Note that 'C:/msys64/mingw64/share' is not in the search path
set by the XDG_DATA_HOME and XDG_DATA_DIRS
environment variables, so applications may not
be able to find it until you set them. The
directories currently searched are:

- C:\msys64\home\User\.local\share
- /usr/local/share/
- /usr/share/

(4/4) Updating the info directory file...
```

7. Run “pacman -S mingw-w64-x86_64-gtk3” to install GTK3

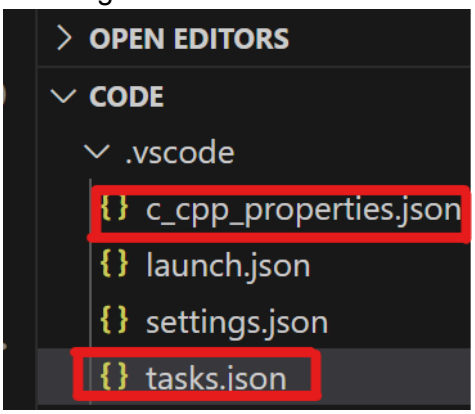
```
User@LAPTOP-JPCH2QLP MSYS ~
$ pacman -S mingw-w64-x86_64-gtk3
warning: mingw-w64-x86_64-gtk3-3.24.38.r43.g0f717ca-1 is up to date -- reinstalling
resolving dependencies...
looking for conflicting packages...

Packages (1) mingw-w64-x86_64-gtk3-3.24.38.r43.g0f717ca-1

Total Installed Size:  69.34 MiB
Net Upgrade Size:       0.00 MiB

:: Proceed with installation? [Y/n]
(1/1) checking keys in keyring                     [############################] 100%
(1/1) checking package integrity                   [############################] 100%
(1/1) loading package files                        [############################] 100%
(1/1) checking for file conflicts                  [############################] 100%
(1/1) checking available disk space                [############################] 100%
:: Processing package changes...
(1/1) reinstalling mingw-w64-x86_64-gtk3           [############################] 100%
:: Running post-transaction hooks...
(1/2) Compiling GSettings XML schema files...
(2/2) Updating icon theme caches...
```

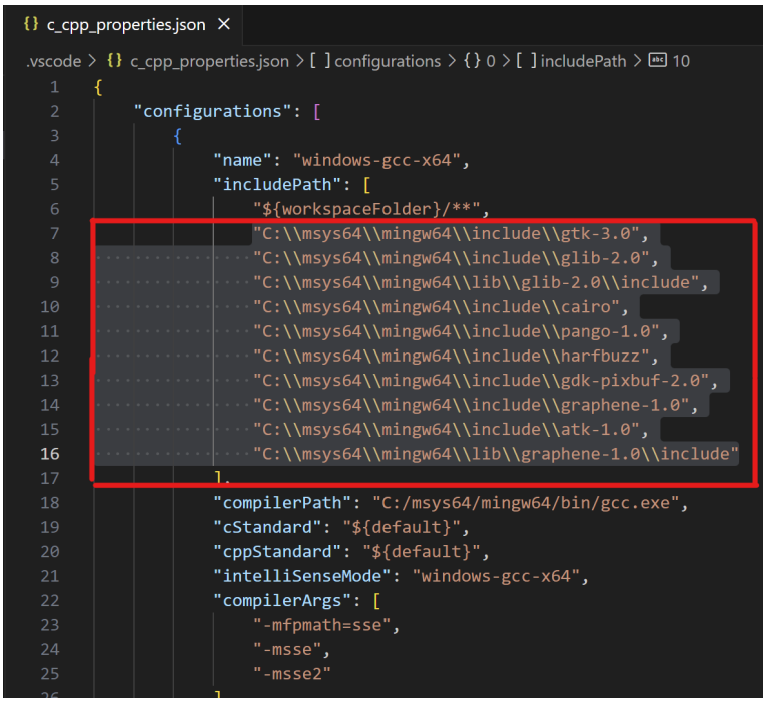
8. Once installed, go to File Explorer path “C:\msys64\mingw64\bin” to find the gtk3 files. Open “gtk3-demo” to test if gtk3 is properly installed.





9. Open Visual Studio Code. (At this point GTK3 is already installed, the following steps is to optimize compiling via “Run Build Task”).
10. Open up “c_cpp_properties.json” & “tasks.json” under .vscode folder and add the following to each file.





11. You have successfully installed gtk3 and optimized for “Run Build Task”.

Appendix

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <string.h>
#include <sys/time.h>
#include <math.h>

// Define constants
#define PLAYER_1 'X'
#define PLAYER_2 'O'
#define FILE_NAME "tic-tac-toe.data"
#define MAX_LINES 958
#define MAX_SIZE 30

// Define functions GTK GUI
void marking_player_move(gpointer ptr);
void quit_game(gpointer ptr);
void start_game(gpointer ptr);
void reset_board();
void change_gamemode(gpointer ptr);
void change_difficulty(gpointer ptr);
void change_algo(gpointer ptr);

// Define functions for game flow
void main_page(GtkApplication *app);
void computer_move();
void marking_computer_move(int move);
int check_winning(const char grid[9]);

// Define functions for minimax
int minimax(char grid[9], int player, int depth);

// Define functions for ML
void swap_lines(char lines[MAX_LINES][MAX_SIZE]);
void readFile(char data[MAX_LINES][MAX_SIZE]);
void training_data(char data[MAX_LINES][MAX_SIZE], int training_dataset);
void testing_data(char datap[MAX_LINES][MAX_SIZE], int testing_dataset);

// Define global variables and global pointers for GTK widgets
GtkWidget *board[3][3];
// 3x3 grid buttons
GtkWidget *status, *gamemode, *difficulty, *algorithm, *quit, *text_box;
// Setting buttons
const gchar *button_label, *current_status, *current_gamemode, *current_difficulty,
*current_algo; // Current settings
char grid[9] = {'1', '2', '3', '4', '5', '6', '7', '8', '9'};
bool start_flag = false, win_flag = false;
int player = -1, count = 0;
double positive = 0, negative = 0;
char data[MAX_LINES][MAX_SIZE];
double count_array[9][6];
```

```
double total_time_minimax = 0;
double total_time_ml = 0;

void marking_player_move(gpointer ptr)
{
    struct timeval start, end;
    button_label = gtk_button_get_label(GTK_BUTTON(ptr)); // Reading the button
label

    // Initializing local variables
    char symbol[5], player_symbol;

    // Checks if the board has already over and whether the button is empty as well
as whether the start button is pressed
    if (start_flag && strncmp(button_label, "\0", 2) == 0 && !win_flag)
    {
        // Assigning X or O according to player
        if (player == -1)
        {
            sprintf(symbol, "%s", "X");
            player_symbol = PLAYER_1;
            gtk_button_set_label(GTK_BUTTON(text_box), "Player 2's Turn!");
        }
        else
        {
            sprintf(symbol, "%s", "O");
            player_symbol = PLAYER_2;
            gtk_button_set_label(GTK_BUTTON(text_box), "Player 1's Turn!");
        }

        gtk_button_set_label(GTK_BUTTON(ptr), symbol); // Setting the clicked
button to the player's symbol
        ++count;                                       // Increases count to 9 to
prevent
        player *= -1;                                  // Switches player

        // Marking player's move to check winning conditions and for computer's
minimax
        if (ptr == board[0][0])
        {
            grid[0] = player_symbol;
        }
        else if (ptr == board[0][1])
        {
            grid[3] = player_symbol;
        }
        else if (ptr == board[0][2])
        {
            grid[6] = player_symbol;
        }
        else if (ptr == board[1][0])
        {
            grid[1] = player_symbol;
        }
        else if (ptr == board[1][1])
```

```
        {
            grid[4] = player_symbol;
        }
        else if (ptr == board[1][2])
        {
            grid[7] = player_symbol;
        }
        else if (ptr == board[2][0])
        {
            grid[2] = player_symbol;
        }
        else if (ptr == board[2][1])
        {
            grid[5] = player_symbol;
        }
        else if (ptr == board[2][2])
        {
            grid[8] = player_symbol;
        }

        // Check if there is winner
        int winner = check_winning(grid);
        if (winner == -1)
        {
            win_flag = true;
            gtk_button_set_label(GTK_BUTTON(text_box), "Player 1 Wins!");
            // printf("\nTotal time for minimax: %f", total_time_minimax);
            // printf("\nTotal time for ml: %f", total_time_ml);
        }
        else if (winner == 1)
        {
            win_flag = true;
            gtk_button_set_label(GTK_BUTTON(text_box), "Player 2 Wins!");
        }
        else if (count == 9)
        {
            gtk_button_set_label(GTK_BUTTON(text_box), "It's a Tie!");
            // printf("\nTotal time for minimax: %f", total_time_minimax);
            // printf("\nTotal time for ml: %f", total_time_ml);
        }
        else if (strncmp(current_gamemode, "Singleplayer", 12) == 0 && player == 1)
// If its Singleplayer, switches to computer immediately after player make a move
        {
            // gettimeofday(&start, NULL);
            computer_move();
            // gettimeofday(&end, NULL);
            // if (strncmp(current_algo, "MINIMAX", 7) == 0)
            // {
            //     total_time_minimax += end.tv_sec + end.tv_usec / 1e6 -
start.tv_sec - start.tv_usec / 1e6;
            // }
            // else
            // {
            //     total_time_ml += end.tv_sec + end.tv_usec / 1e6 - start.tv_sec -
start.tv_usec / 1e6;
```

```
            // }
        }
    }
    else if (!start_flag)
    {
        gtk_button_set_label(GTK_BUTTON(text_box), "Please click 'START' to start
the game."); // Outputting error if start button not pressed
    }
    else if (strncmp(button_label, "\0", 2) != 0 && count < 9)
    {
        gtk_button_set_label(GTK_BUTTON(text_box), "Please choose another box.");
// Outputting error if board button is selected
    }
}

// This function is called when the quit button is clicked
void quit_game(gpointer ptr)
{
    exit(0); // Stop the code and close the window.
}

void reset_board()
{
    // Initializing local variable
    char current_number[5];

    // Clear entire board
    for (int i = 0; i < 3; ++i)
    {
        for (int j = 0; j < 3; ++j)
        {
            gtk_button_set_label(GTK_BUTTON(board[i][j]), "\0");
            sprintf(current_number, "%d", (i * 3) + (j + 1));
            grid[i * 3 + j] = current_number[0];
        }
    }

    // Reset certain global variable and
    gtk_button_set_label(GTK_BUTTON(text_box), "\0");
    win_flag = false;
    player = -1;
    count = 0;
}

// This function is called when the start or restart button is clicked
void start_game(gpointer ptr)
{
    // Getting the current label to differentiate between start and restart button
    current_status = gtk_button_get_label(GTK_BUTTON(ptr));

    // If is start button, initiate all setting buttons and their current settings.
Otherwise, clear the entire board however remain the current settings
    if (strncmp(current_status, "START", 5) == 0)
    {
        start_flag = true;
```

```
        gtk_button_set_label(GTK_BUTTON(ptr), "RESTART");
        current_status = gtk_button_get_label(GTK_BUTTON(ptr));

        gtk_button_set_label(GTK_BUTTON(gamemode), "Multiplayer");
        current_gamemode = gtk_button_get_label(GTK_BUTTON(gamemode));

        gtk_button_set_label(GTK_BUTTON(text_box), "Player 1's Turn!");
    }
    else
    {
        if (strncmp(current_algo, "ML", 2) == 0 && strncmp(current_difficulty,
"MEDIUM", 6) == 0)
        {
            int training_dataset = floor(MAX_LINES * 0.8);
            int testing_dataset = MAX_LINES - training_dataset;
            swap_lines(data);
            training_data(data, training_dataset);
            testing_data(data, testing_dataset);
        }
        reset_board();
    }
}

void change_gamemode(gpointer ptr)
{
    // Getting the current label to differentiate between Multiplayer and
Singleplayer
    current_gamemode = gtk_button_get_label(GTK_BUTTON(ptr));

    // Toggles between Multiplayer and Singleplayer
    if (strncmp(current_gamemode, "Multiplayer", 11) == 0)
    {
        gtk_button_set_label(GTK_BUTTON(ptr), "Singleplayer");
        current_gamemode = gtk_button_get_label(GTK_BUTTON(ptr));

        gtk_button_set_label(GTK_BUTTON(difficulty), "EASY");
        current_difficulty = gtk_button_get_label(GTK_BUTTON(difficulty));

        gtk_button_set_label(GTK_BUTTON(algorithm), "MINIMAX");
        current_algo = gtk_button_get_label(GTK_BUTTON(algorithm));

        reset_board();
    }
    else if (strncmp(current_gamemode, "Singleplayer", 12) == 0)
    {
        gtk_button_set_label(GTK_BUTTON(ptr), "Multiplayer");
        current_gamemode = gtk_button_get_label(GTK_BUTTON(ptr));

        gtk_button_set_label(GTK_BUTTON(difficulty), "\0");
        current_difficulty = gtk_button_get_label(GTK_BUTTON(difficulty));

        gtk_button_set_label(GTK_BUTTON(algorithm), "\0");
        current_algo = gtk_button_get_label(GTK_BUTTON(algorithm));

        gtk_button_set_label(GTK_BUTTON(text_box), "Player 1's Turn!");
```

```
        reset_board();
    }
}

// This function is called when the difficulty button is pressed to change
difficulty; only applicable to Singleplayer
void change_difficulty(gpointer ptr)
{
    // Getting the current label to differentiate between EASY, MEDIUM and HARD
    current_difficulty = gtk_button_get_label(GTK_BUTTON(ptr));

    // Toggle between the three difficulties
    if (strncmp(current_difficulty, "EASY", 4) == 0)
    {
        gtk_button_set_label(GTK_BUTTON(ptr), "MEDIUM");
        current_difficulty = gtk_button_get_label(GTK_BUTTON(ptr));

        reset_board();
    }
    else if (strncmp(current_difficulty, "MEDIUM", 6) == 0)
    {
        if (strncmp(current_algo, "ML", 2) == 0)
        {
            gtk_button_set_label(GTK_BUTTON(ptr), "EASY");
            current_difficulty = gtk_button_get_label(GTK_BUTTON(ptr));
        }
        else
        {
            gtk_button_set_label(GTK_BUTTON(ptr), "HARD");
            current_difficulty = gtk_button_get_label(GTK_BUTTON(ptr));
        }

        reset_board();
    }
    else if (strncmp(current_difficulty, "HARD", 4) == 0)
    {
        gtk_button_set_label(GTK_BUTTON(ptr), "EASY");
        current_difficulty = gtk_button_get_label(GTK_BUTTON(ptr));

        reset_board();
    }
}

// This function is called when the algorithm option button is pressed
void change_algo(gpointer ptr)
{
    // Getting the current label to differentiate between MINIMAX and ML
    current_algo = gtk_button_get_label(GTK_BUTTON(ptr));

    // Toggle between the two algorithms
    if (strncmp(current_algo, "MINIMAX", 7) == 0)
    {
        gtk_button_set_label(GTK_BUTTON(ptr), "ML");
        gtk_button_set_label(GTK_BUTTON(difficulty), "EASY");
```

```
        current_algo = gtk_button_get_label(GTK_BUTTON(ptr));
        current_difficulty = gtk_button_get_label(GTK_BUTTON(difficulty));

        reset_board();
    }
    else if (strncmp(current_algo, "ML", 2) == 0)
    {
        gtk_button_set_label(GTK_BUTTON(ptr), "MINIMAX");
        gtk_button_set_label(GTK_BUTTON(difficulty), "EASY");
        current_algo = gtk_button_get_label(GTK_BUTTON(ptr));
        current_difficulty = gtk_button_get_label(GTK_BUTTON(difficulty));

        reset_board();
    }
}

// This function is called to create the windows and the buttons
void main_page(GtkApplication *app)
{
    // Creates GTK widget pointer for user interface
    GtkWidget *window;
    GtkWidget *board_grid, *main_grid, *settings_grid;

    // Create the pop up window which will contain all other GTK widget
    window = gtk_application_window_new(app);
    gtk_window_set_default_size(GTK_WINDOW(window), 600, 500);       // Set window
size
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER); // Set window
position to be in the center of the screen
    gtk_window_set_title(GTK_WINDOW(window), "Tic Tac Toe");         // Set
window's title

    // Creates all the grid for difference purposes
    main_grid = gtk_grid_new();     // Main grid holds both 3x3 grid and setting
grid
    board_grid = gtk_grid_new();    // Board grid hold only 3x3
    settings_grid = gtk_grid_new(); // Setting grid hold all buttons to set the
settings

    // Add main grid to window
    gtk_container_add(GTK_CONTAINER(window), main_grid);

    // Creating setting buttons and giving them the signal for onClicked to do
their respective jobs
    status = gtk_button_new_with_label("START");
    g_signal_connect(status, "clicked", G_CALLBACK(start_game), status);

    gamemode = gtk_button_new_with_label("\0");
    g_signal_connect(gamemode, "clicked", G_CALLBACK(change_gamemode), gamemode);

    difficulty = gtk_button_new_with_label("\0");
    g_signal_connect(difficulty, "clicked", G_CALLBACK(change_difficulty),
difficulty);

    algorithm = gtk_button_new_with_label("\0");
```

```
    g_signal_connect(algorithm, "clicked", G_CALLBACK(change_algo), algorithm);

    quit = gtk_button_new_with_label("QUIT");
    g_signal_connect(quit, "clicked", G_CALLBACK(quit_game), NULL);

    // Creating text box for information
    text_box = gtk_button_new_with_label("\0");
    gtk_widget_set_hexpand(text_box, TRUE);

    // Make the button expand to fill the grid horizontally
    gtk_widget_set_hexpand(status, TRUE);
    gtk_widget_set_hexpand(gamemode, TRUE);
    gtk_widget_set_hexpand(difficulty, TRUE);
    gtk_widget_set_hexpand(algorithm, TRUE);
    gtk_widget_set_hexpand(quit, TRUE);

    // Attach the buttons to the position of the grid
    gtk_grid_attach(GTK_GRID(settings_grid), status, 0, 0, 1, 1);
    gtk_grid_attach(GTK_GRID(settings_grid), gamemode, 1, 0, 1, 1);
    gtk_grid_attach(GTK_GRID(settings_grid), difficulty, 2, 0, 1, 1);
    gtk_grid_attach(GTK_GRID(settings_grid), algorithm, 3, 0, 1, 1);
    gtk_grid_attach(GTK_GRID(settings_grid), quit, 4, 0, 1, 1);

    // Creating 3x3 grid buttons
    for (int i = 0; i < 3; ++i)
    {
        for (int j = 0; j < 3; ++j)
        {
            board[i][j] = gtk_button_new_with_label("\0");
            gtk_widget_set_hexpand(board[i][j], TRUE);
// Make the button expand to fill the grid horizontally
            gtk_widget_set_vexpand(board[i][j], TRUE);
// Make the button expand to fill the grid vertically
            gtk_grid_attach(GTK_GRID(board_grid), board[i][j], i, j + 1, 1, 1);
// Attach the board buttons to the board grid
            g_signal_connect(board[i][j], "clicked",
G_CALLBACK(marking_player_move), board[i][j]); // Giving each button a signal for
onClicked to set the symbol
        }
    }

    // Attaching setting grid and board grid in the main grid
    gtk_grid_attach(GTK_GRID(main_grid), settings_grid, 0, 0, 3, 3);
    gtk_grid_attach(GTK_GRID(main_grid), text_box, 0, 3, 3, 3);
    gtk_grid_attach(GTK_GRID(main_grid), board_grid, 0, 6, 3, 3);

    // Show all widgets
    gtk_widget_show_all(window);
}

// This function is called only during Singleplayer and immediately after player
make his/her move
// For this function only, PLAYER_2 refers to COMPUTER
void computer_move()
{
```

```
    // Setting seed for random number generator
    srand(time(NULL));

    // Initializing local variable
    int cpu_move;
    if ((strncmp(current_difficulty, "EASY", 4) == 0) ||
// Easy difficulty --> Using completely random number generator to determine
Computer's move
        (strncmp(current_difficulty, "MEDIUM", 6) == 0 && count < 2 &&
strncmp(current_algo, "MINIMAX", 7) == 0)) // First step of medium for minimax only
difficulty --> Using completely random number generator to determine Computer's
move
    {
        while (1)
        {
            // Assign computer move to a random number generated
            cpu_move = rand() % 9;
            if (grid[cpu_move] != PLAYER_1 && grid[cpu_move] != PLAYER_2 &&
cpu_move >= 0 && cpu_move < 9) // Checks if slot is taken and whether number
generated is between 0 to 8
            {
                grid[cpu_move] = PLAYER_2; // Register Computer's move in array
                ++count;                   // Increases count for determining
whether it's tie a not
                player = -1;               // Switches to player's turn

                // Setting GTK button to computer's move
                marking_computer_move(cpu_move);
                break; // To get out of the while loop when computer make it's move
            }
        }
    }
    else // Hard difficulty --> Using completely minimax algorithm to determine
Computer's move / Medium difficulty --> Use minimax from Computer's second move and
above / Algo is ML then use ML to play
    {
        if (strncmp(current_algo, "ML", 2) == 0)
        {
            // Initialize local variables for this if condition
            double total_pprobability = 1;
            double total_nprobability = 1;
            double current_probability = 0;
            int move = 0, sign;
            char int_str[5];
            char symbol;

            // Calculate the probability of X winning and O winning
            for (int i = 0; i < 9; i++)
            {
                if (grid[i] == PLAYER_1)
                {
                    total_pprobability *= count_array[i][0];
                }
                else if (grid[i] == PLAYER_2)
                {
```

```
                total_nprobability *= count_array[i][1];
            }
        }

        // Compare to see if computer is going to play defensive or offensive
        if (total_pprobability > total_nprobability)
        {
            symbol = PLAYER_1;
            sign = -1;
        }
        else
        {
            symbol = PLAYER_2;
            sign = 1;
        }

        // Base on whether is it defensive or offensive, choose a suitable move
relative to respective playing style
        for (int i = 0; i < 9; i++)
        {
            double probability = sign;
            if (grid[i] != PLAYER_1 && grid[i] != PLAYER_2)
            {
                grid[i] = symbol;
                probability *= (symbol == PLAYER_1 ? count_array[i][0] :
count_array[i][4]);
                probability *= (symbol == PLAYER_1 ? total_pprobability :
total_nprobability);
                if (symbol == 'x')
                {
                    if (probability < current_probability)
                    {
                        current_probability = probability;
                        move = i;
                    }
                }
                else
                {
                    if (probability > current_probability)
                    {
                        current_probability = probability;
                        move = i;
                    }
                }
                printf("Probability for move %d is %g\n", i + 1, probability);
                sprintf(int_str, "%d", i + 1);
                grid[i] = int_str[0];
            }
        }

        printf("Best move: %d\n", move + 1);
        grid[move] = PLAYER_2; // Register Computer's move in array
        ++count;               // Increases count for determining whether it's
tie a not
        player = -1;           // Switches to player's turn
```

```
            // Setting GTK button to computer's move
            marking_computer_move(move);
        }
        else
        {
            // Computer's move and minimax algorithm
            int move = -1, score = -100, i;
            char int_str[5];
            for (i = 0; i < 9; ++i)
            {
                if (grid[i] != PLAYER_1 && grid[i] != PLAYER_2)
                {
                    grid[i] = PLAYER_2; // Start the first move

                    int tempScore = -minimax(grid, -1, 100 - count); // Call
minimax as opponent's move
                    // Reset the backend grid
                    sprintf(int_str, "%d", i + 1);
                    grid[i] = int_str[0];

                    // If returned score from minimax is higher than score, store
the score value as returned score to compare if there is a better option
                    if (tempScore > score)
                    {
                        score = tempScore;
                        move = i;
                    }
                }
            }

            grid[move] = PLAYER_2; // Register Computer's move in array
            ++count;               // Increases count for determining whether it's
tie a not
            player = -1;           // Switches to player's turn

            // Setting GTK button to computer's move
            marking_computer_move(move);
        }
    }
}

void marking_computer_move(int move)
{
    // Initializing local variable
    char string[100];

    // Setting GTK button to computer's move
    if (move == 0)
    {
        gtk_button_set_label(GTK_BUTTON(board[0][0]), "O");
    }
    else if (move == 1)
    {
        gtk_button_set_label(GTK_BUTTON(board[1][0]), "O");
```

```
    }
    else if (move == 2)
    {
        gtk_button_set_label(GTK_BUTTON(board[2][0]), "O");
    }
    else if (move == 3)
    {
        gtk_button_set_label(GTK_BUTTON(board[0][1]), "O");
    }
    else if (move == 4)
    {
        gtk_button_set_label(GTK_BUTTON(board[1][1]), "O");
    }
    else if (move == 5)
    {
        gtk_button_set_label(GTK_BUTTON(board[2][1]), "O");
    }
    else if (move == 6)
    {
        gtk_button_set_label(GTK_BUTTON(board[0][2]), "O");
    }
    else if (move == 7)
    {
        gtk_button_set_label(GTK_BUTTON(board[1][2]), "O");
    }
    else if (move == 8)
    {
        gtk_button_set_label(GTK_BUTTON(board[2][2]), "O");
    }

    // Check if there is winner
    int winner = check_winning(grid);
    if (winner == 1)
    {
        win_flag = true;
        gtk_button_set_label(GTK_BUTTON(text_box), "Computer Wins!");
    }
    else if (winner == 0)
    {
        // Setting information text on GUI
        sprintf(string, "COMPUTER chose grid %d!", move + 1);
        gtk_button_set_label(GTK_BUTTON(text_box), string);
    }
}

int check_winning(const char grid[9])
{
    // Check which player win already otherwise return 0
    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 ((grid[wins[i][0]] == PLAYER_1 || grid[wins[i][0]] == PLAYER_2) &&
            grid[wins[i][0]] == grid[wins[i][1]] &&
```

```
            grid[wins[i][0]] == grid[wins[i][2]])
        {
            if (grid[wins[i][0]] == PLAYER_1)
            {
                return -1;
            }
            else if (grid[wins[i][0]] == PLAYER_2)
            {
                return 1;
            }
        }
    }

    return 0;
}

int minimax(char grid[9], int player, int depth)
{
    // Initializing local variables
    char shape, int_str[5];
    int move = -1, score = -100, i;

    // Checks if the game has already ended
    int winner = check_winning(grid);

    // If game ended, return which player won
    if (winner != 0)
        return winner * player * depth;

    // Sorting player's move shape
    if (player == -1)
    {
        shape = PLAYER_1;
    }
    else if (player == 1)
    {
        shape = PLAYER_2;
    }

    // Start of minimax algorithm
    for (i = 0; i < 9; ++i)
    {
        int current_depth = depth;
        if (grid[i] != PLAYER_1 && grid[i] != PLAYER_2)
        {
            grid[i] = shape; // Try next move

            // Run minimax function for next step as next player
            int thisScore = -minimax(grid, player * -1, current_depth - 1);
            if (thisScore > score)
            {
                score = thisScore;
                move = i;
            }
```

```
            // Reset the backend grid
            sprintf(int_str, "%d", i + 1);
            grid[i] = int_str[0];
        }
    }

    // If there are no more moves, return 0
    if (move == -1)
        return 0;

    return score;
}

void swap_lines(char lines[MAX_LINES][MAX_SIZE])
{
    // Setting seed for random value
    srand(time(NULL));

    // Swap all the lines starting from the first line till the last
    for (int x = 0; x < MAX_LINES; x++)
    {
        // Randomly generate a number from 0 to MAX_SIZE to swap x with
        int i = rand() % (x + 1);
        char temp[MAX_SIZE];
        strcpy(temp, lines[x]);
        strcpy(lines[x], lines[i]);
        strcpy(lines[i], temp);
    }
}

void readFile(char data[MAX_LINES][MAX_SIZE])
{
    // Read dataset file
    FILE *file = fopen(FILE_NAME, "r");

    // Check if can open file
    if (file == NULL)
    {
        printf("Error opening file");
        exit(1);
    }

    // Change newline flag to end string flag for each row
    for (int i = 0; i < MAX_LINES; i++)
    {
        if (fgets(data[i], sizeof(data[0]), file))
        {
            data[i][strcspn(data[i], "\n")] = '\0';
        }
    }

    // Close file
    fclose(file);
}
```

```
void training_data(char data[MAX_LINES][MAX_SIZE], int training_dataset)
{
    // Iterate each row to count number of x, o, and b and the positive and
negative outcome to calculate probability
    for (int i = 0; i < training_dataset; i++)
    {
        // Get current line
        char *current_line = data[i];

        // Get either positive or negative at the end of current line
        char *label = strrchr(current_line, ',') + 1;

        // Iterate through current line with increment of x by 2 to skip commas
        for (int x = 0; x < 17; x += 2)
        {
            if (strcmp(label, "positive") == 0)
            {
                positive++;
                if (current_line[x] == 'x')
                {
                    count_array[x / 2][0] += 1;
                }
                else if (current_line[x] == 'o')
                {
                    count_array[x / 2][1] += 1;
                }
                else if (current_line[x] == 'b')
                {
                    count_array[x / 2][2] += 1;
                }
            }
            else if (strcmp(label, "negative") == 0)
            {
                negative++;
                if (current_line[x] == 'x')
                {
                    count_array[x / 2][3] += 1;
                }
                else if (current_line[x] == 'o')
                {
                    count_array[x / 2][4] += 1;
                }
                else if (current_line[x] == 'b')
                {
                    count_array[x / 2][5] += 1;
                }
            }
        }
    }

    // Calculate the probability of each move which result in winning or losing
    for (int x = 0; x < 9; x++)
    {
        for (int y = 0; y < 6; y++)
        {
```

```
            if (y < 3)
            {
                count_array[x][y] /= positive;
            }
            else
            {
                count_array[x][y] /= negative;
            }
        }
    }
}

void testing_data(char data[MAX_LINES][MAX_SIZE], int testing_dataset)
{
    // Initialize truePositive, falsePositive, trueNegative, falseNegative
    double tp = 0, fp = 0, tn = 0, fn = 0;

    // Iterate through the rest of the dataset (20%) for testing
    for (int i = MAX_LINES - testing_dataset; i < MAX_LINES; i++)
    {
        // Calculate the positive and negative probability of 80%
        double pprobability = positive / (positive + negative);
        double nprobability = negative / (positive + negative);

        // Get current line
        char *current_line = data[i];

        // Get either positive or negative at the end of current line
        char *label = strrchr(current_line, ',') + 1;

        // Iterate through current line with increment of x by 2 to skip commas and
calculate the current probability of that state
        for (int x = 0; x < 17; x += 2)
        {
            if (current_line[x] == 'x')
            {
                pprobability *= count_array[x / 2][0];
                nprobability *= count_array[x / 2][3];
            }
            else if (current_line[x] == 'o')
            {
                pprobability *= count_array[x / 2][1];
                nprobability *= count_array[x / 2][4];
            }
            else if (current_line[x] == 'b')
            {
                pprobability *= count_array[x / 2][2];
                nprobability *= count_array[x / 2][5];
            }
        }

        // Check if positive probability is higher than negative probability, if
positive is higher, predicted result is positive else otherwise
        if (pprobability > nprobability)
        {
```

```
            if (strcmp(label, "positive") == 0)
            {
                tp += 1;
            }
            else
            {
                fp += 1;
            }
        }
        else
        {
            if (strcmp(label, "negative") == 0)
            {
                tn += 1;
            }
            else
            {
                fn += 1;
            }
        }
    }

    // Output the accuracy of the test data
    printf("Error: %g\n", ((fp + fn) / (tp + fp + tn + fn)) * 100);
    printf("True Positive: %g\n", tp);
    printf("False Positive: %g\n", fp);
    printf("True Negative: %g\n", tn);
    printf("False Negative: %g\n", fn);
    printf("Accuracy: %g\n", ((tp + tn) / (tp + fp + tn + fn)) * 100);
}

int main(int argc, char **argv)
{
    GtkApplication *app;
    int training_dataset = floor(MAX_LINES * 0.8);
    int testing_dataset = MAX_LINES - training_dataset;
    int status;

    // Start of ML to prepare for ML mode during the game
    readFile(data);
    swap_lines(data);
    training_data(data, training_dataset);
    testing_data(data, testing_dataset);

    app = gtk_application_new("com.p2t3.csc1103_project",
G_APPLICATION_DEFAULT_FLAGS);
    g_signal_connect(app, "activate", G_CALLBACK(main_page), NULL);
    status = g_application_run(G_APPLICATION(app), argc, argv);
    g_object_unref(app);

    return status;
}
```

References

1. Team, Gtk. “The GTK Project - a Free and Open-source Cross-platform Widget Toolkit.” *The GTK Team*, www.gtk.org/docs/installations/windows.
2. “MSYS2.” *MSYS2*, www.msys2.org
3. GTK World. “How to Install GTK3/GTK4 on Windows 10.” *YouTube*, 14 June 2020, www.youtube.com/watch?v=rUJFYOCbuDg
4. *Tic Tac Toe*. tictactoe.com.
5. “Tic-tac-toe.” *Wikipedia*, 26 Oct. 2023, en.wikipedia.org/wiki/Tic-tac-toe.
6. “Minimax.” *Wikipedia*, 18 Nov. 2023, en.wikipedia.org/wiki/Minimax.
7. The Coding Train. “Coding Challenge #149: Tic Tac Toe.” *YouTube*, 24 July 2019, www.youtube.com/watch?v=GTWrWM1UsnA.
8. The Coding Train. “Coding Challenge 154: Tic Tac Toe AI With Minimax Algorithm.” *YouTube*, 11 Dec. 2019, www.youtube.com/watch?v=trKjYdBASyQ.
9. “Machine Learning.” *Wikipedia*, 14 Nov. 2023, en.wikipedia.org/wiki/Machine_learning.
10. Zhang, Jeremy. “Reinforcement Learning — Implement TicTacToe - Towards Data Science.” *Medium*, 10 Dec. 2021, towardsdatascience.com/reinforcement-learning-implement-tictactoe-189582bea542.
11. Indriyani, Indriyani, and M. Ihsan Alfani Putera. “Web-based Application for Classification Using Naïve Bayes and K-means Clustering (Case Study: Tic-tac-toe Game).” *International Journal of Engineering and Emerging Technology*, vol. 5, no. 1, Universitas Udayana, July 2020, p. 8. *Crossref*, https://doi.org/10.24843/ijeet.2020.v05.i01.p04.
12. “Naive Bayes Classifier.” *Wikipedia*, 20 Nov. 2023, en.wikipedia.org/wiki/Naive_Bayes_classifier.
13. *What Are Naive Bayes Classifiers? | IBM*. www.ibm.com/topics/naive-bayes.
14. GokulVSD. “GitHub - GokulVSD/tiCtactoe: Tic Tac Toe Game With GUI. PvP as Well as PvC With Rudimentary AI. Written in C Using GTK Lib 3.0 and Glade.” *GitHub*, github.com/GokulVSD/tiCtactoe.