Programming Data Structures and Algorithms – 2

BSc (Hons)Computing 23.1P

Hansi Weerathunga - COCOBSCCOMP231P-037

Lecturer – Mr.Samira Dayan Jayasekara

School of Computing & Engineering

National Institute of Business Management

Colombo

Table of Contents

1 Acknowledgement

First of all, I would like to thank National Institute of Business Management and Coventry university for giving this great opportunity to us for making games using Programming Data Structures and Algorithms.

I would like to express my gratitude to Mr. Samira Dayan Jayasekara for the source of knowledge, support and guidance given throughout our lectures.

Finally, I would like to Thank My parents and everyone who supported to do this project successfully.

2 Abstract

The report briefly contains the Problem-solving logic for each game, UI screenshot for allowing game player to provide answers and when the game players to provide correct answer and incorrect answers, Exception handling and code segment unit testing. Normalized database table structure screenshots and gaming playing experience, game functionalities, user interfaces, data management and quality assurance procedures.

3 Introduction

3.1.1 Overview of the project

As part of our BSc (Hons) Computing degree program, there is a subject called Programming Data Structure and Algorithms. So, I am assigned to do a project using our theoretical knowledge to implement a game application. There for, I have created five real-world games to represent My theoretical knowledge of data structures and algorithms for real-world scenarios. I created these games to improve the players' problem-solving thinking patterns.

3.1.2 Importance of data structure and algorithm in Game developments

The importance of data structures and algorithms in the industry of game creation cannot be highlighted. They represent the basis on which the logic, mechanics, and connection of games are developed. My project is a great example of how theoretical understanding in these fields is essential for creating interesting and challenging games. With the help of these ideas, we can design effective games, sort out challenging problems, helping both players and developers.

3.1.3 Introduction to game menu option

The game application includes five games and, all created to show how data structures and algorithms are used in real-world environments:

I have created the Knight's tour problem game to identify the navigate a Knight across a chessboard. So that player must understand the correct sequence of moves.

Then I created the Identify the longest common sequence game to find the longest common sequence between randomly selected two strings.

Then I created the Eight Queens' Puzzle game to place the movements of eight queens on an 8*8 chessboard without threatening each other.

Then I created a Tic-Tac-Tac-Tac-Teo game to play with the computer to identify the Min-Max theory.

Finally, I created the Identify Shortest Path game to identify the short distance needed to navigate between randomly selected cities.

So, My Game Application not only fully entertains but also players can educate themselves on the algorithm theories. Also, players can improve their problem-solving thinking pattern. Also, as a developer, I have team used JavaScript to develop the backend of this game application. Then, to design the front pages, I used HTML and CSS languages. Then I used MYSQL to develop the database side. Also, from that, I could successfully develop our theoretical knowledge through practical implementation.

3.1.4 Chapter Summary

I provided the five game options that best represent my knowledge of data structures and algorithms. These games not only entertain players but also provide users and improve their problem-solving abilities. I also pointed out the value of these concepts in game design, focusing on how important they are. My project is an example of how academic theory and real-world application may merge, offering an environment for both education and entertainment.

4 Implementation

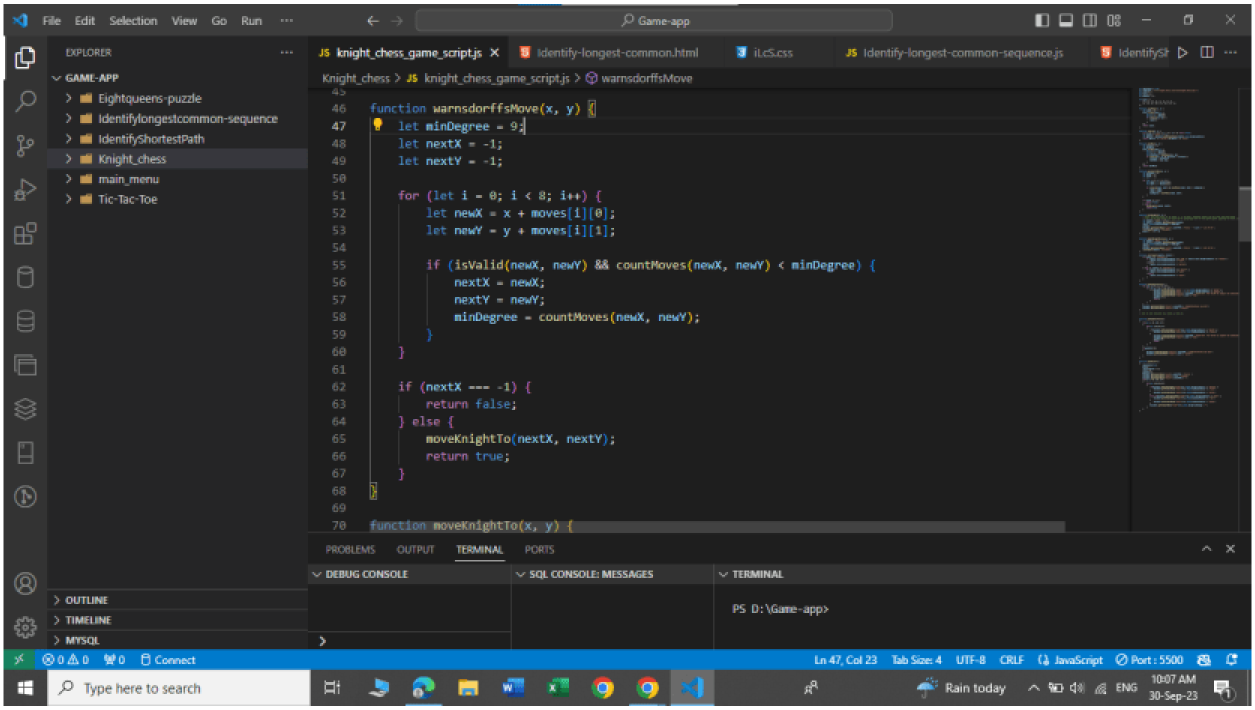
4.1 knight's tour Problem

4.1.1 Program Logic used to solve the problem

Knight’s tour problem is a game that only use knights to the game and it improves the chess knowledge and skills. Knight can only move the way of letter “L”.

The program used two by two array and eight by eight chessboard for the game development. The algorithm used for the game development is “Warns Dorf’s Rule”

Here is the logic used to the solve the problem:



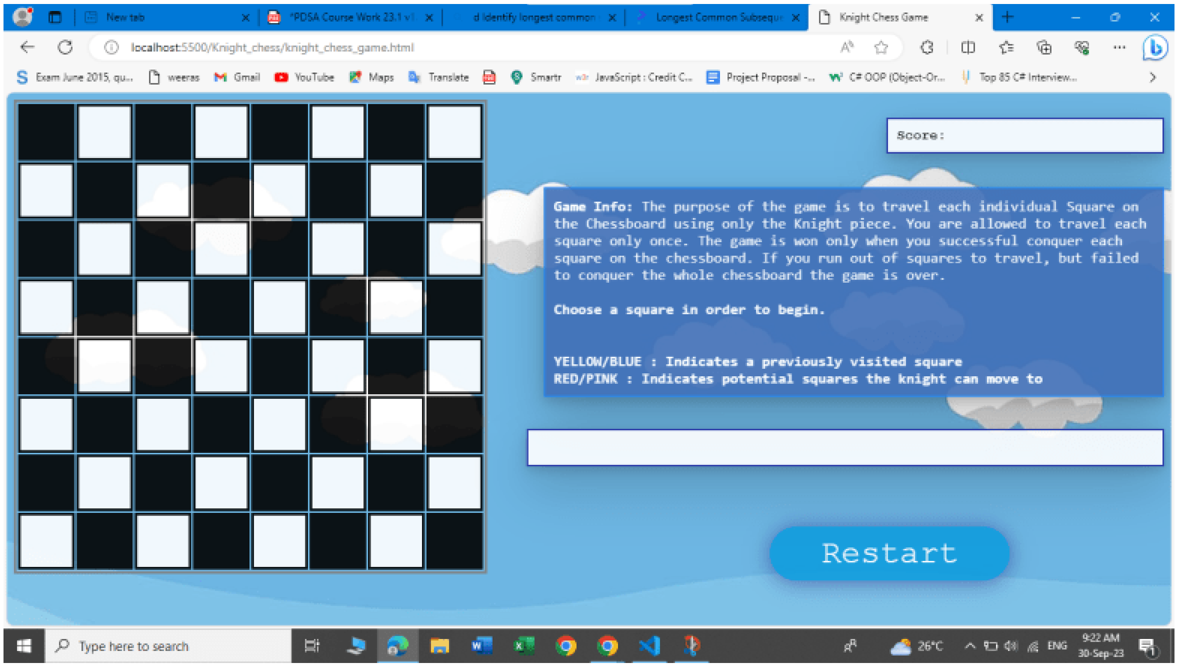
File Edit Selection View Go Run ···
Game-app
EXPLORER
GAME-APP
Eightqueens-puzzle
Identifylongestcommon-sequence
IdentifyShortestPath
Knight_chess
main_menu
Tic-Tac-Toe
knight_chess_game_script.js
Identify-longest-common.html
iLcS.css
Identify-longest-common-sequence.js
IdentifySh
Knight_chess > knight_chess_game_script.js > warnsdorffsMove
PROBLEMS OUTPUT TERMINAL PORTS
DEBUG CONSOLE
SQL CONSOLE: MESSAGES
TERMINAL
OUTLINE
TIMELINE
MYSQL
Connect
Ln 47, Col 23 Tab Size: 4 UTF-8 CRLF JavaScript Port : 5500
Type here to search
Rain today
ENG
10:07 AM
30-Sep-23

```
function warnsdorffsMove(x, y) {
    let minDegree = 9;
    let nextX = -1;
    let nextY = -1;

    for (let i = 0; i < 8; i++) {
        let newX = x + moves[i][0];
        let newY = y + moves[i][1];

        if (isValid(newX, newY) && countMoves(newX, newY) < minDegree) {
            nextX = newX;
            nextY = newY;
            minDegree = countMoves(newX, newY);
        }
    }

    if (nextX === -1) {
        return false;
    } else {
        moveKnightTo(nextX, nextY);
        return true;
    }
}

function moveKnightTo(x, y) {
```

```
PS D:\Game-app>
```

Figure 01

4.1.2 User Interfaces



Score:

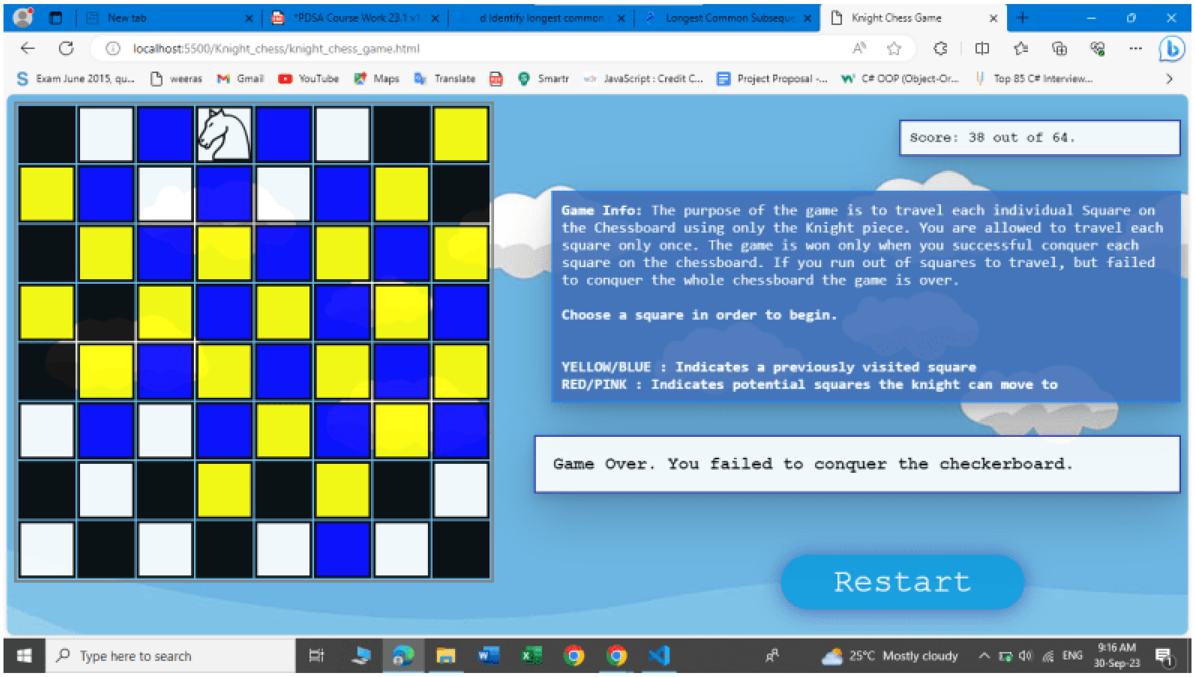
Game Info: The purpose of the game is to travel each individual Square on the Chessboard using only the Knight piece. You are allowed to travel each square only once. The game is won only when you successful conquer each square on the chessboard. If you run out of squares to travel, but failed to conquer the whole chessboard the game is over.

Choose a square in order to begin.

YELLOW/BLUE : Indicates a previously visited square
RED/PINK : Indicates potential squares the knight can move to

Restart

Figure 02



Score: 38 out of 64.

Game Info: The purpose of the game is to travel each individual Square on the Chessboard using only the Knight piece. You are allowed to travel each square only once. The game is won only when you successful conquer each square on the chessboard. If you run out of squares to travel, but failed to conquer the whole chessboard the game is over.

Choose a square in order to begin.

YELLOW/BLUE : Indicates a previously visited square
RED/PINK : Indicates potential squares the knight can move to

Game Over. You failed to conquer the checkerboard.

Restart

Figure 03

4.1.3 Validations and Exception Handling in this application

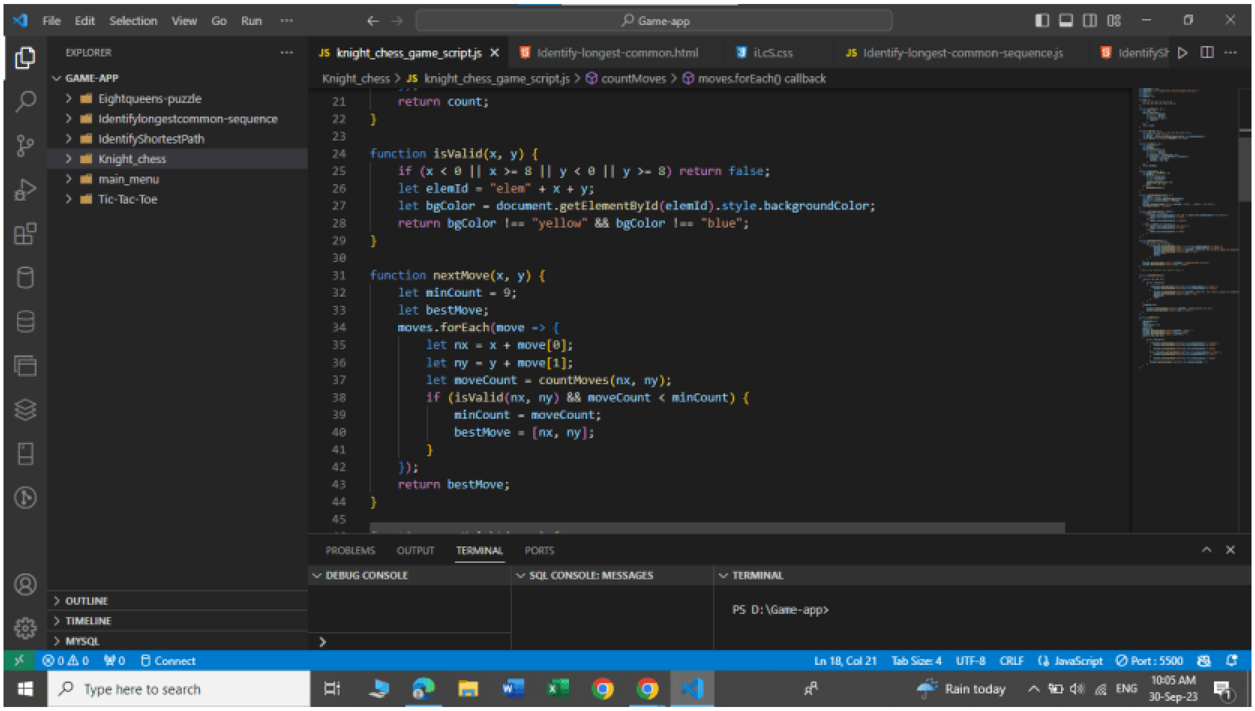
Input Validation:

- The code doesn’t check specifically for user input validity as the game depends on user interaction with the chessboard. However, it ensures or the certifies that valid moves are calculated based on the position of knight on the chessboard.

Validation of the Results:

- getGameOverStatus() function checks the output of the game by checking the chessboard status to decide the player has won or not.

Code:


File Edit Selection View Go Run ···
Game-app
EXPLORER
GAME-APP
Eightqueens-puzzle
Identifylongestcommon-sequence
IdentifyShortestPath
Knight_chess
main_menu
Tic-Tac-Toe
knight_chess_game_script.js
Identify-longest-common.html
iLcS.css
Identify-longest-common-sequence.js
IdentifySh
Knight_chess > knight_chess_game_script.js > countMoves > moves.forEach() callback
PROBLEMS OUTPUT TERMINAL PORTS
DEBUG CONSOLE
SQL CONSOLE: MESSAGES
TERMINAL
OUTLINE
TIMELINE
MYSQL
Connect
Ln 18, Col 21 Tab Size: 4 UTF-8 CRLF JavaScript Port : 5500
Type here to search
Rain today
ENG
10:05 AM
30-Sep-23

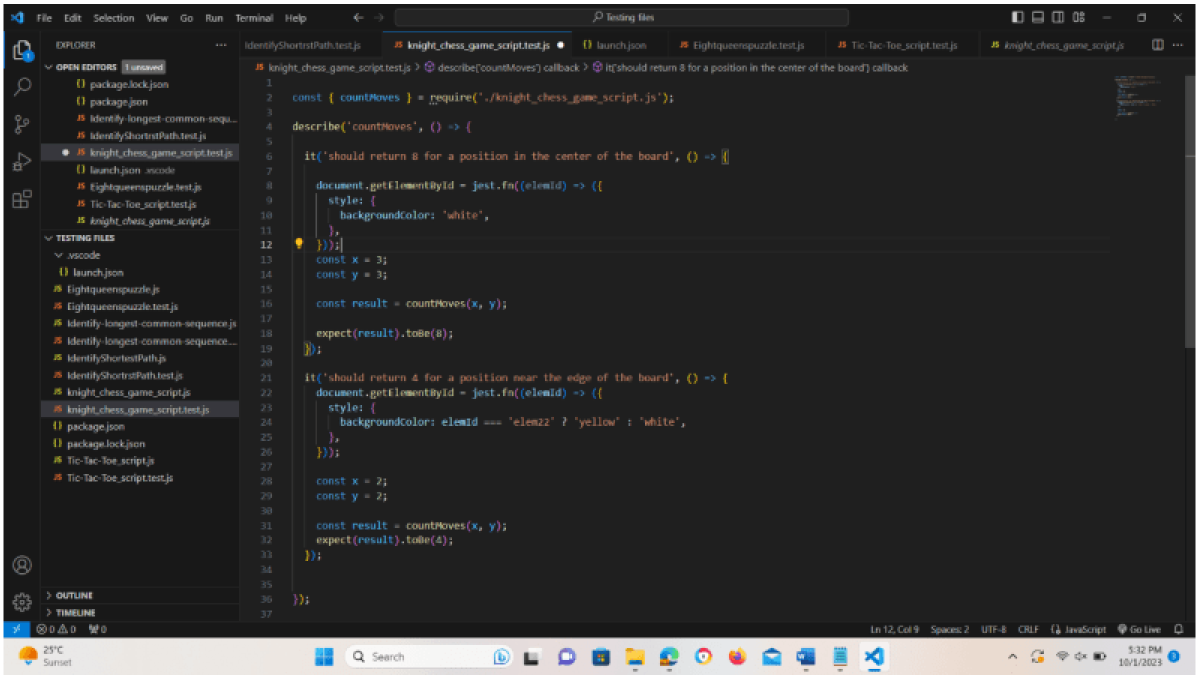
```javascript
    return count;
}

function isValid(x, y) {
    if (x < 0 || x >= 8 || y < 0 || y >= 8) return false;
    let elemId = "elem" + x + y;
    let bgColor = document.getElementById(elemId).style.backgroundColor;
    return bgColor !== "yellow" && bgColor !== "blue";
}

function nextMove(x, y) {
    let minCount = 9;
    let bestMove;
    moves.forEach(move => {
        let nx = x + move[0];
        let ny = y + move[1];
        let moveCount = countMoves(nx, ny);
        if (isValid(nx, ny) && moveCount < minCount) {
            minCount = moveCount;
            bestMove = [nx, ny];
        }
    });
    return bestMove;
}
```

```
PS D:\Game-app>
```

Figure 04

4.1.4 Unit Testing

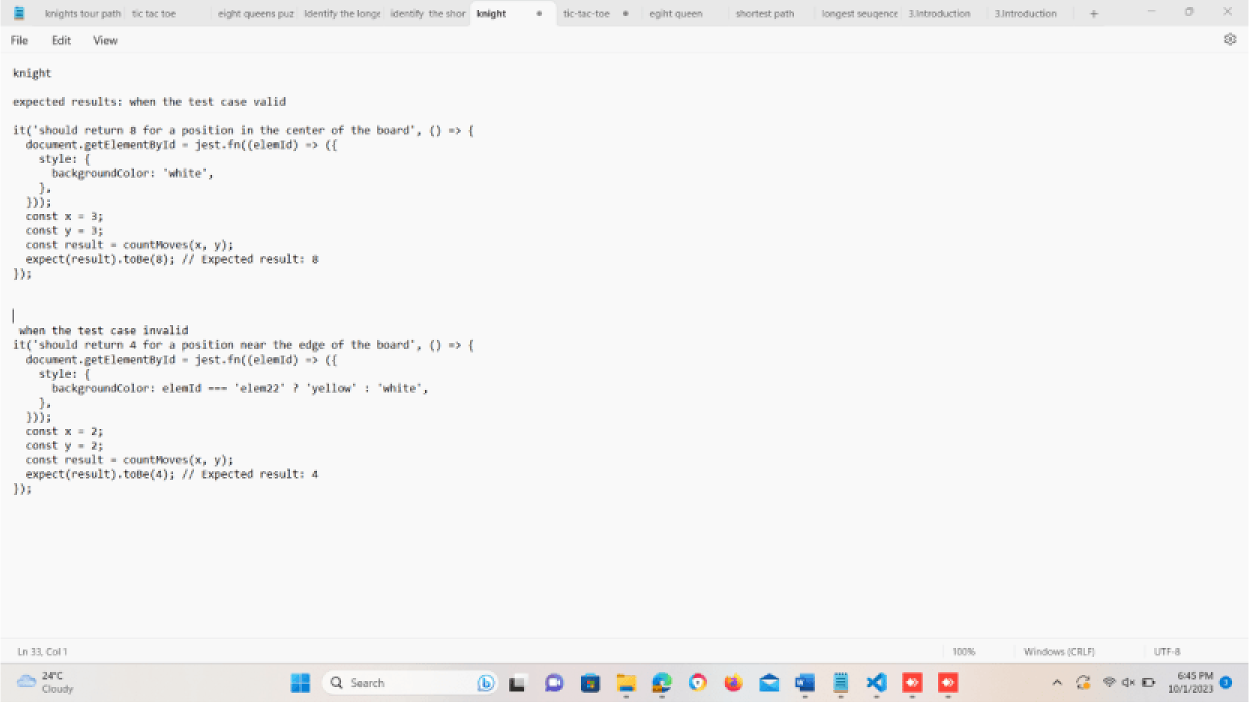
Testing code:



```javascript
const { countMoves } = require('./knight_chess_game_script.js');

describe('countMoves', () => {

  it('should return 8 for a position in the center of the board', () => {

    document.getElementById = jest.fn((elemId) => ({
      style: {
        backgroundColor: 'white',
      },
    }));
    const x = 3;
    const y = 3;

    const result = countMoves(x, y);

    expect(result).toBe(8);
  });

  it('should return 4 for a position near the edge of the board', () => {
    document.getElementById = jest.fn((elemId) => ({
      style: {
        backgroundColor: elemId === 'elem22' ? 'yellow' : 'white',
      },
    }));

    const x = 2;
    const y = 2;

    const result = countMoves(x, y);
    expect(result).toBe(4);
  });


});
```

Figure 05

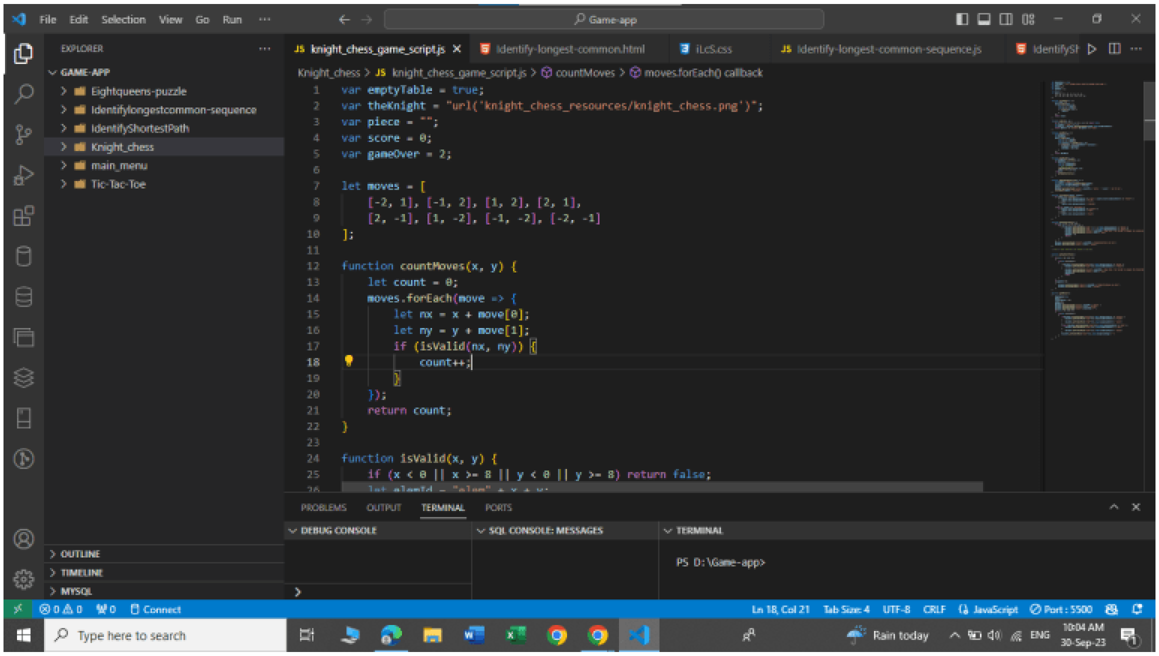
Results:



```
knight

expected results: when the test case valid

it('should return 8 for a position in the center of the board', () => {
  document.getElementById = jest.fn((elemId) => ({
    style: {
      backgroundColor: 'white',
    },
  }));
  const x = 3;
  const y = 3;
  const result = countMoves(x, y);
  expect(result).toBe(8); // Expected result: 8
});


 when the test case invalid
it('should return 4 for a position near the edge of the board', () => {
  document.getElementById = jest.fn((elemId) => ({
    style: {
      backgroundColor: elemId === 'elem22' ? 'yellow' : 'white',
    },
  }));
  const x = 2;
  const y = 2;
  const result = countMoves(x, y);
  expect(result).toBe(4); // Expected result: 4
});
```

Figure 06

4.1.5 Indicate the Data Structures used with its purpose

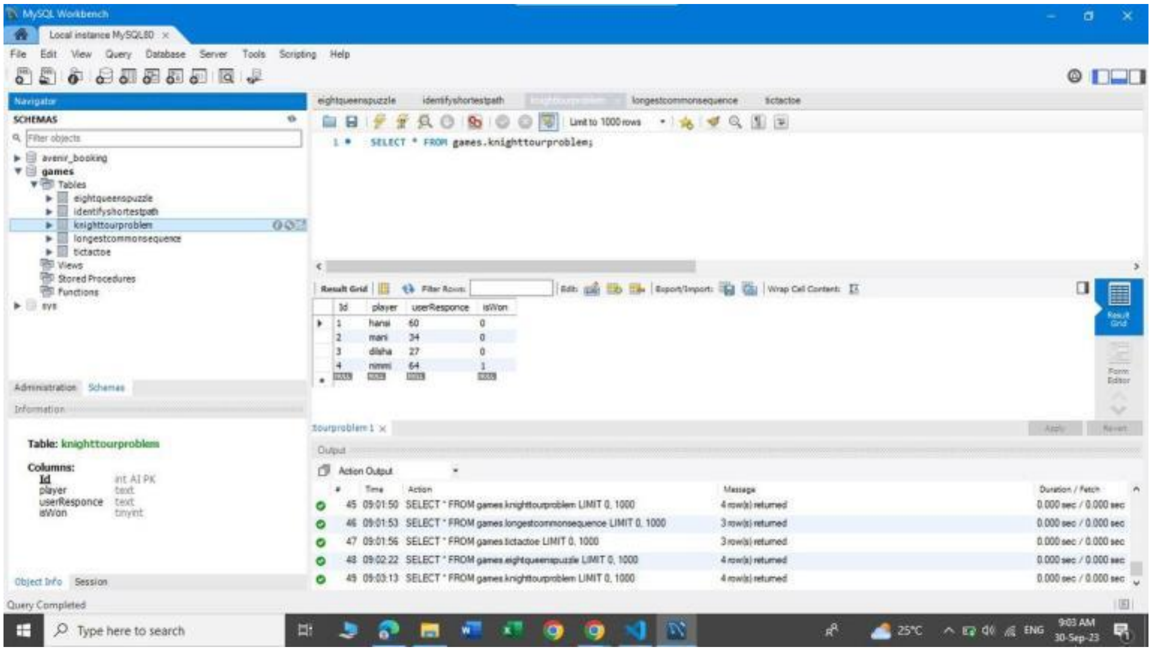

File Edit Selection View Go Run ...
Game-app
EXPLORER
GAME-APP
Eightqueens-puzzle
Identifylongestcommon-sequence
IdentifyShortestPath
Knight_chess
main_menu
Tic-Tac-Toe
knight_chess_game_script.js
Identify-longest-common.html
iLcS.css
Identify-longest-common-sequence.js
Knight_chess > knight_chess_game_script.js > countMoves > moves.forEach() callback
PROBLEMS OUTPUT TERMINAL PORTS
DEBUG CONSOLE
SQL CONSOLE: MESSAGES
TERMINAL
OUTLINE
TIMELINE
MYSQL
Ln 18, Col 21 Tab Size: 4 UTF-8 CRLF JavaScript Port : 5500
Type here to search
Rain today
ENG
10:04 AM
30-Sep-23

```
var emptyTable = true;
var theKnight = "url('knight_chess_resources/knight_chess.png')";
var piece = "";
var score = 0;
var gameOver = 2;

let moves = [
    [-2, 1], [-1, 2], [1, 2], [2, 1],
    [2, -1], [1, -2], [-1, -2], [-2, -1]
];

function countMoves(x, y) {
    let count = 0;
    moves.forEach(move => {
        let nx = x + move[0];
        let ny = y + move[1];
        if (isValid(nx, ny)) {
            count++;
        }
    });
    return count;
}

function isValid(x, y) {
    if (x < 0 || x >= 8 || y < 0 || y >= 8) return false;
```

```
PS D:\Game-app>
```

Figure 07

4.1.6 Normalized DB Table Structure



```
SELECT * FROM games.knighttourproblem;
```

Id	player	userResponce	isWon
1	hansi	60	0
2	mani	34	0
3	dilsha	27	0
4	nimmi	64	1
NULL	NULL	NULL	NULL

Table: knighttourproblem

Columns:

Id	int AI PK
player	text
userResponce	text
isWon	tinyint

```
#   Time      Action                                                     Message            Duration / Fetch
45  09:01:50  SELECT * FROM games.knighttourproblem LIMIT 0, 1000        4 row(s) returned  0.000 sec / 0.000 sec
46  09:01:53  SELECT * FROM games.longestcommonsequence LIMIT 0, 1000    3 row(s) returned  0.000 sec / 0.000 sec
47  09:01:56  SELECT * FROM games.tictactoe LIMIT 0, 1000                3 row(s) returned  0.000 sec / 0.000 sec
48  09:02:22  SELECT * FROM games.eightqueenspuzzle LIMIT 0, 1000        4 row(s) returned  0.000 sec / 0.000 sec
49  09:03:13  SELECT * FROM games.knighttourproblem LIMIT 0, 1000        4 row(s) returned  0.000 sec / 0.000 sec
```

Figure 08

4.2 Identify longest common sequence

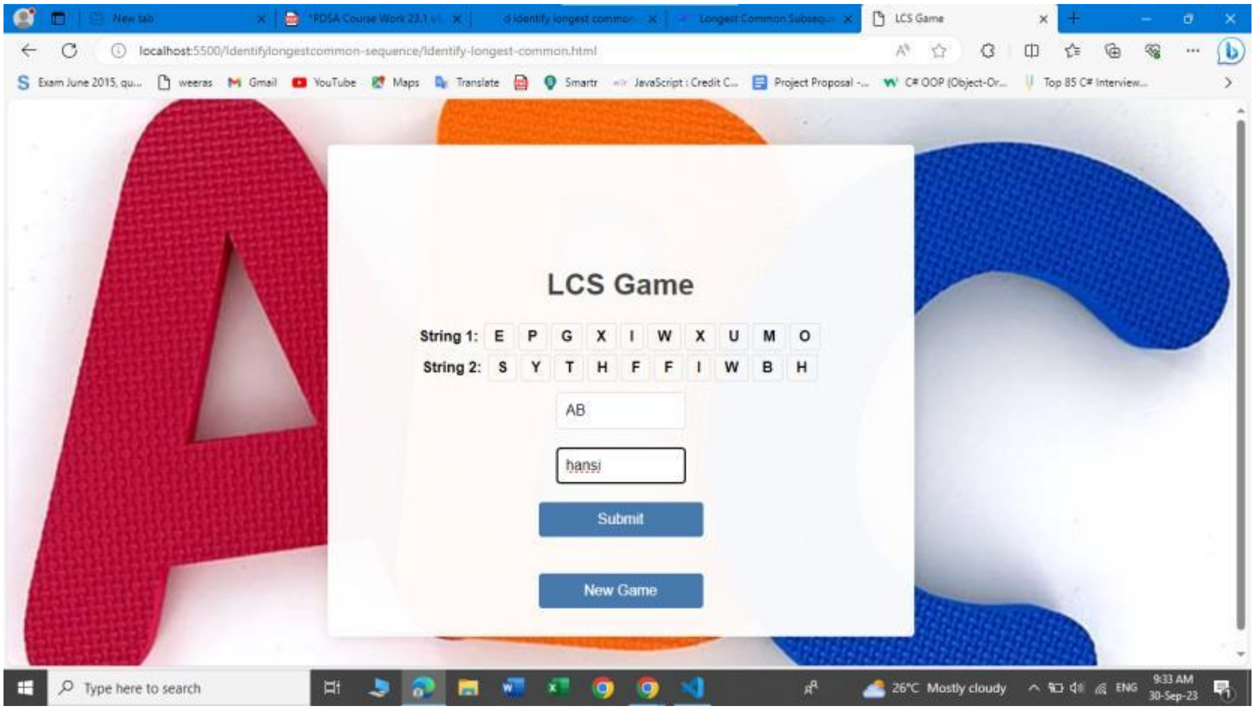
4.2.1 Program Logic used to solve the problem

In the application, the game's logic is built up by generating 2 random strings (s1 and s2), which is used to compare and find the longest common sequence where each character is displayed in separate boxes. An interface is implemented for the players to enter the answer and that string will be comparatively check with he previously generated longest common sequence.

If the answer matches among the strings, a response message will be displayed as "Correct!" and the data of the player with the response will be saved to the database. If the answer mismatches, a response message will be displayed as "Wrong. Try Again!". Once the "New Game" button is clicked it allows the player to restart the game by regenerating a random couple of new strings again. Therefore, a user interactive interface has been designed to overcome the boredom of the players with an efficient calculation by storing all the data to the database.

Here is the logic used to the solve the problem:

4.2.2 User Interfaces

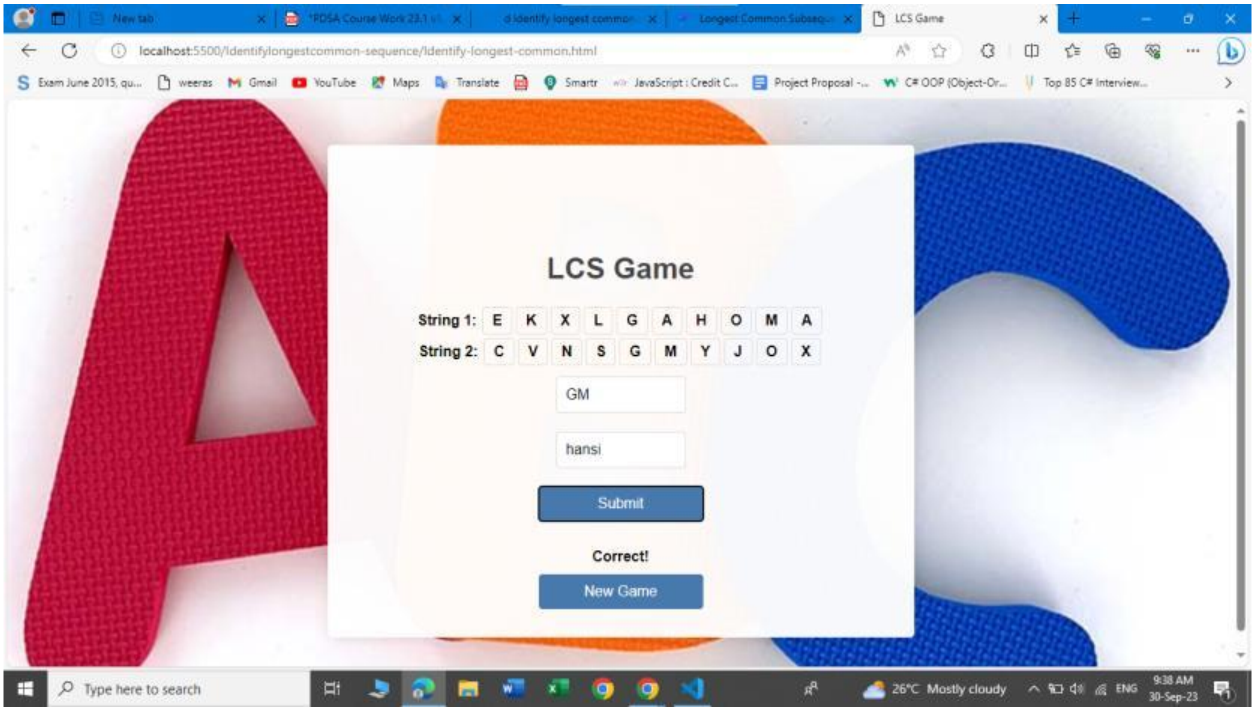


String 1: E P G X I W X U M O
String 2: S Y T H F F I W B H

AB

hansi

Figure 09



LCS Game

String 1: E K X L G A H O M A

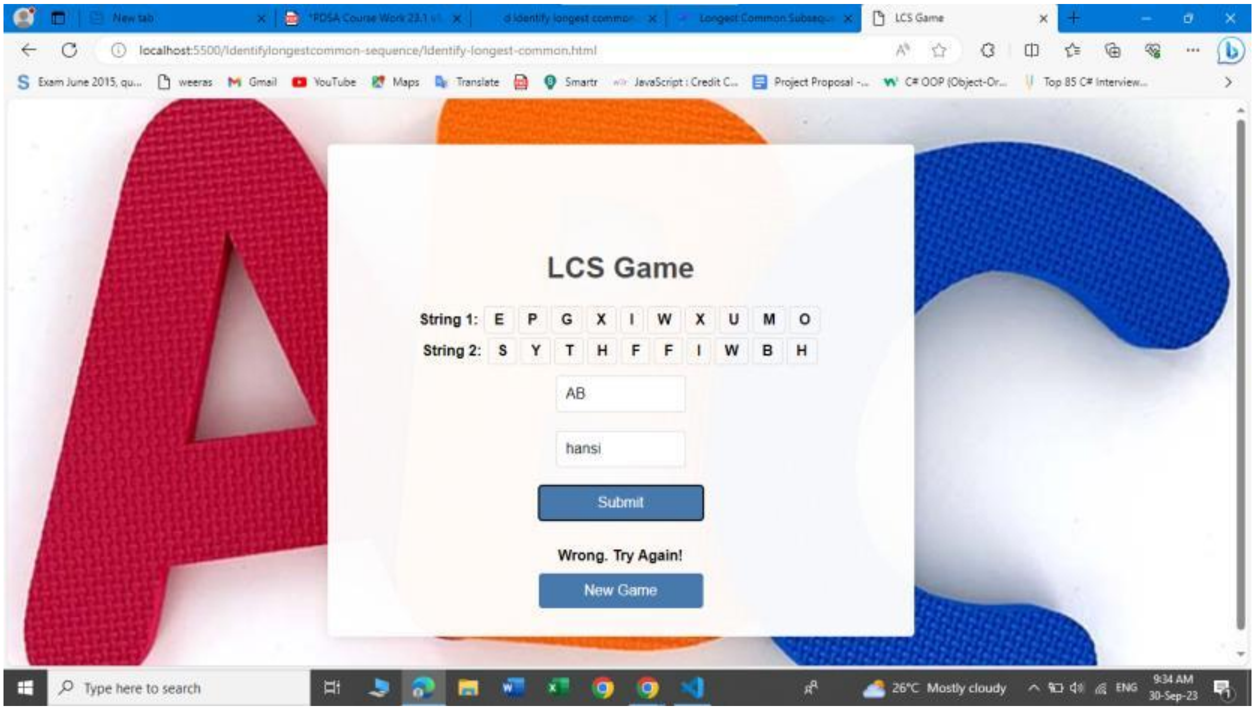
String 2: C V N S G M Y J O X

GM

hansi

Correct!

Figure 10



LCS Game

String 1: E P G X I W X U M O

String 2: S Y T H F F I W B H

AB

hansi

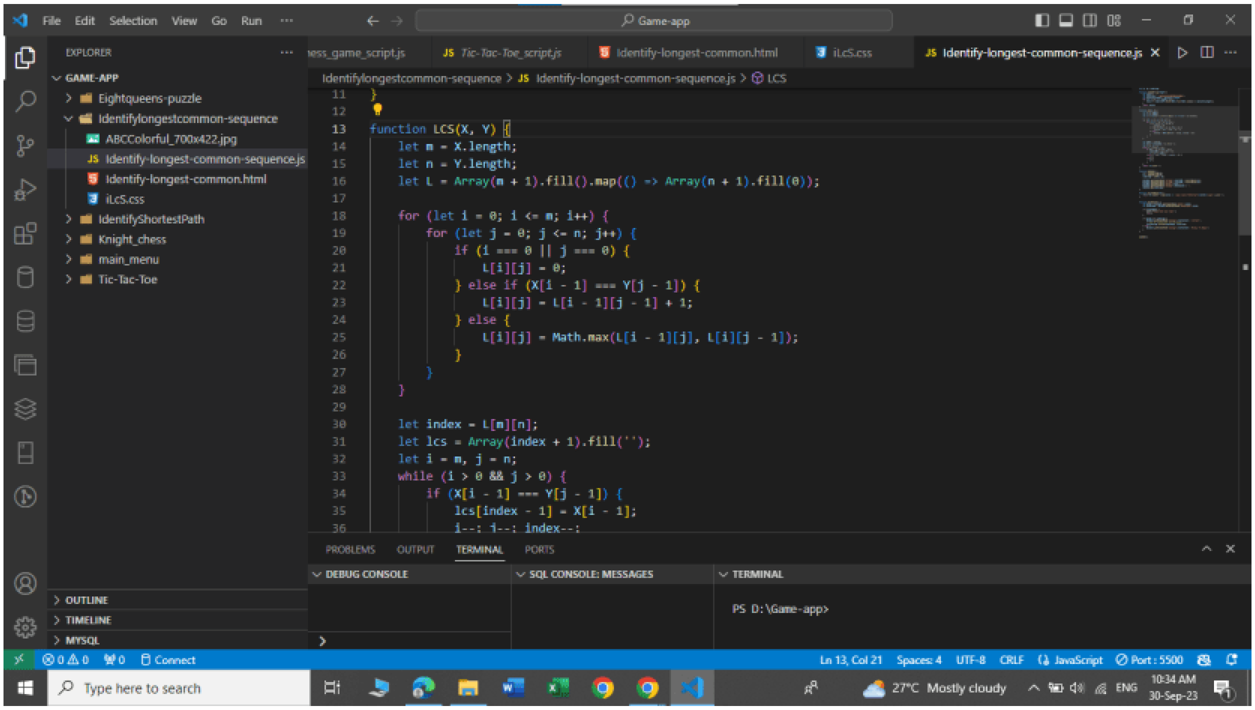
Wrong. Try Again!

Figure 11

4.2.3 Validations and Exception Handling in this application

The algorithm used to calculate the Longest Common Sequence (LCS) was “Dynamic Programming”. Between two strings which were randomly generated (s1 and s2) with a 2D dimensional array, as the data structure to store and retrieve data efficiently.

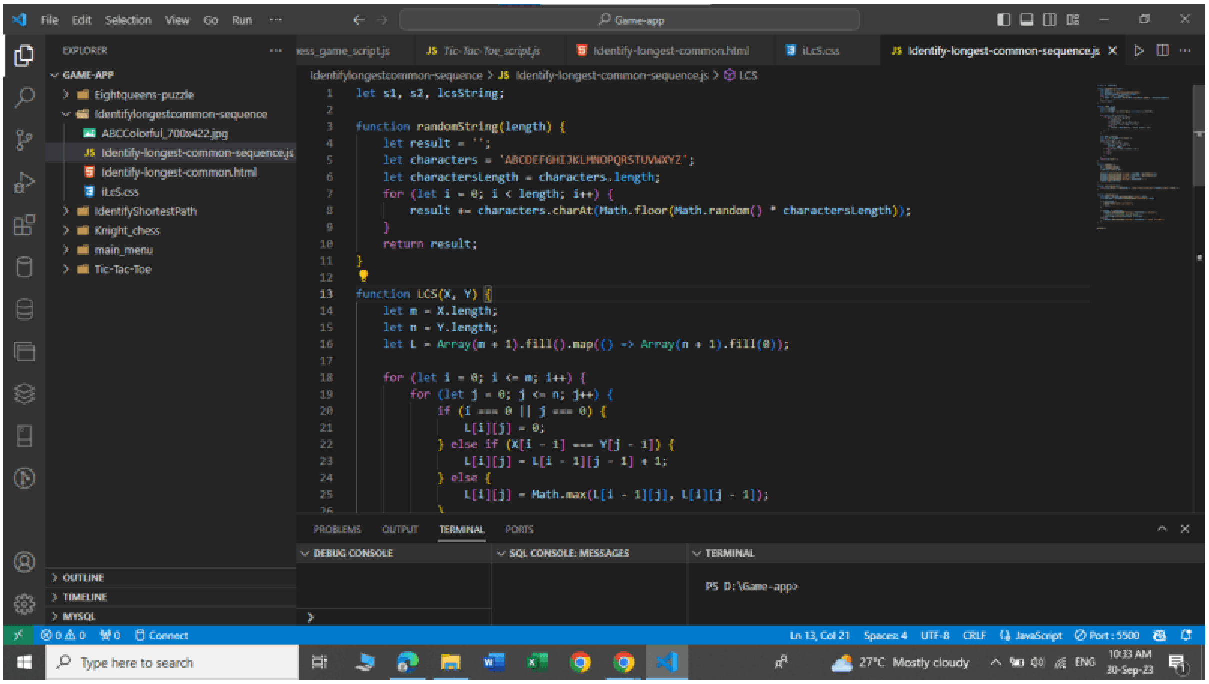
Its purpose of using the mentioned algorithm is to let players engage actively to find the LCS between the two strings



```
}

function LCS(X, Y) {
    let m = X.length;
    let n = Y.length;
    let L = Array(m + 1).fill().map(() => Array(n + 1).fill(0));

    for (let i = 0; i <= m; i++) {
        for (let j = 0; j <= n; j++) {
            if (i === 0 || j === 0) {
                L[i][j] = 0;
            } else if (X[i - 1] === Y[j - 1]) {
                L[i][j] = L[i - 1][j - 1] + 1;
            } else {
                L[i][j] = Math.max(L[i - 1][j], L[i][j - 1]);
            }
        }
    }

    let index = L[m][n];
    let lcs = Array(index + 1).fill('');
    let i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i - 1] === Y[j - 1]) {
            lcs[index - 1] = X[i - 1];
            i--; j--; index--;
```

```
PS D:\Game-app>
```

Figure 12



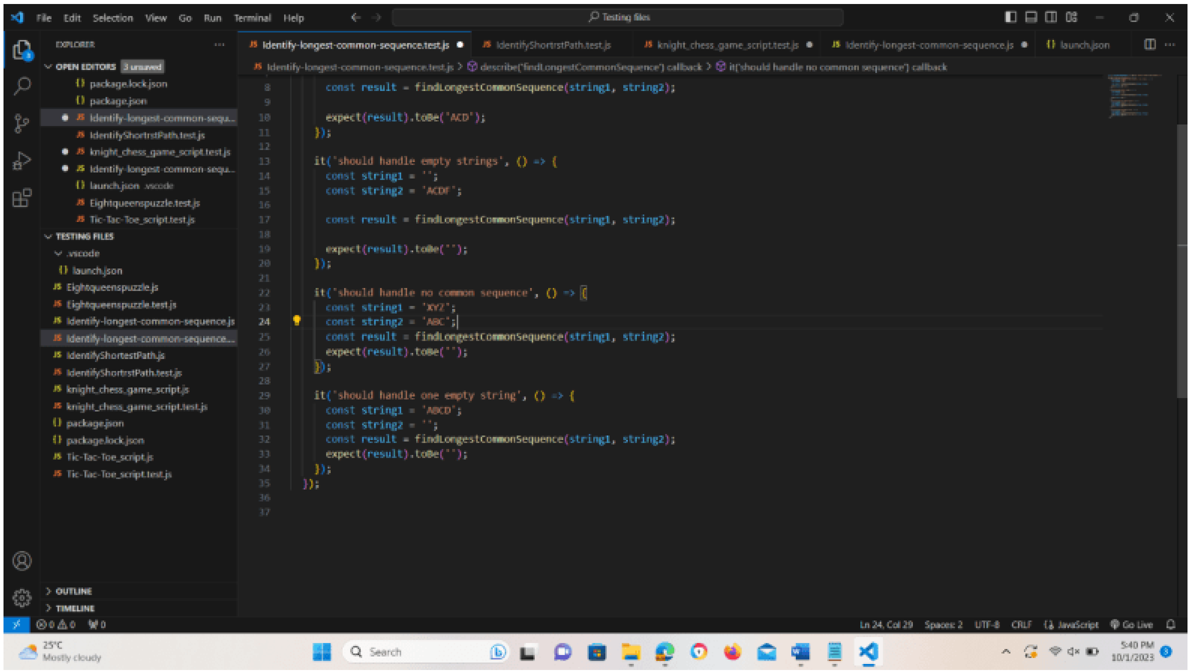
```
let s1, s2, lcsString;

function randomString(length) {
    let result = '';
    let characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    let charactersLength = characters.length;
    for (let i = 0; i < length; i++) {
        result += characters.charAt(Math.floor(Math.random() * charactersLength));
    }
    return result;
}

function LCS(X, Y) {
    let m = X.length;
    let n = Y.length;
    let L = Array(m + 1).fill().map(() => Array(n + 1).fill(0));

    for (let i = 0; i <= m; i++) {
        for (let j = 0; j <= n; j++) {
            if (i === 0 || j === 0) {
                L[i][j] = 0;
            } else if (X[i - 1] === Y[j - 1]) {
                L[i][j] = L[i - 1][j - 1] + 1;
            } else {
                L[i][j] = Math.max(L[i - 1][j], L[i][j - 1]);
            }
```

```
PS D:\Game-app>
```

Figure 13

4.2.4 Unit Testing

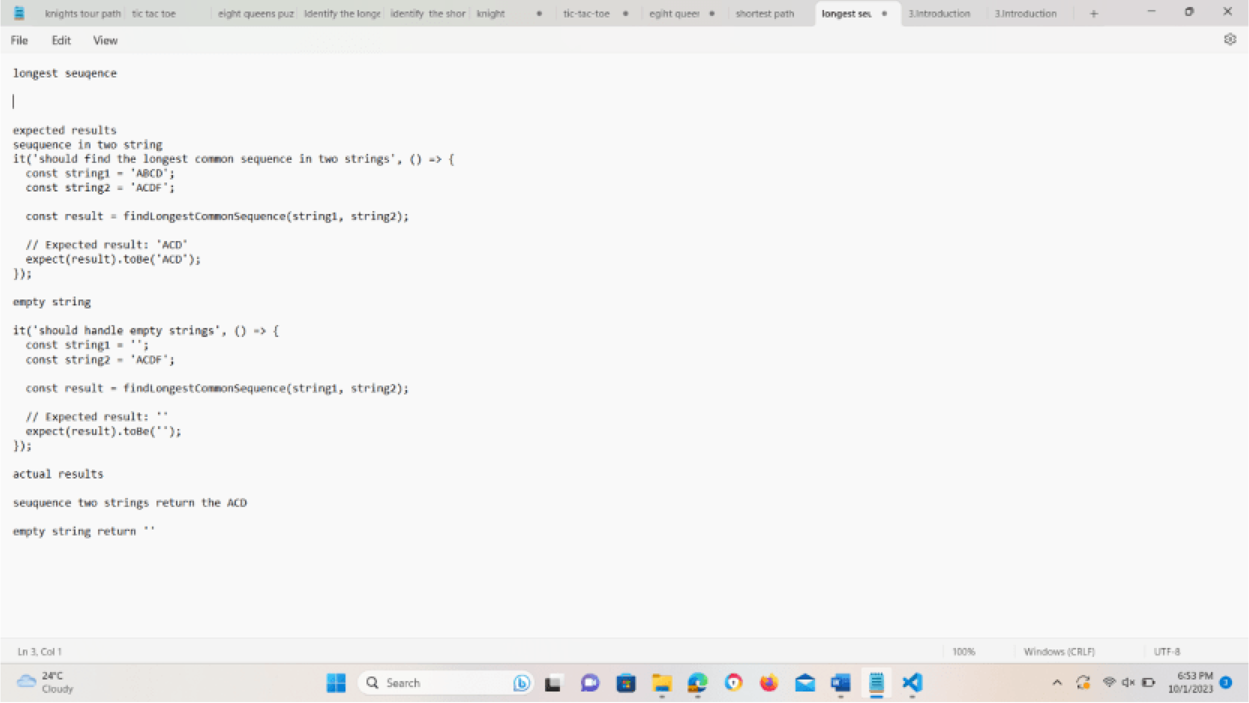
Testing code:



```
      const result = findLongestCommonSequence(string1, string2);

      expect(result).toBe('ACD');
    });

    it('should handle empty strings', () => {
      const string1 = '';
      const string2 = 'ACDF';

      const result = findLongestCommonSequence(string1, string2);

      expect(result).toBe('');
    });

    it('should handle no common sequence', () => {
      const string1 = 'XYZ';
      const string2 = 'ABC';
      const result = findLongestCommonSequence(string1, string2);
      expect(result).toBe('');
    });

    it('should handle one empty string', () => {
      const string1 = 'ABCD';
      const string2 = '';
      const result = findLongestCommonSequence(string1, string2);
      expect(result).toBe('');
    });
  });
```

Figure 14

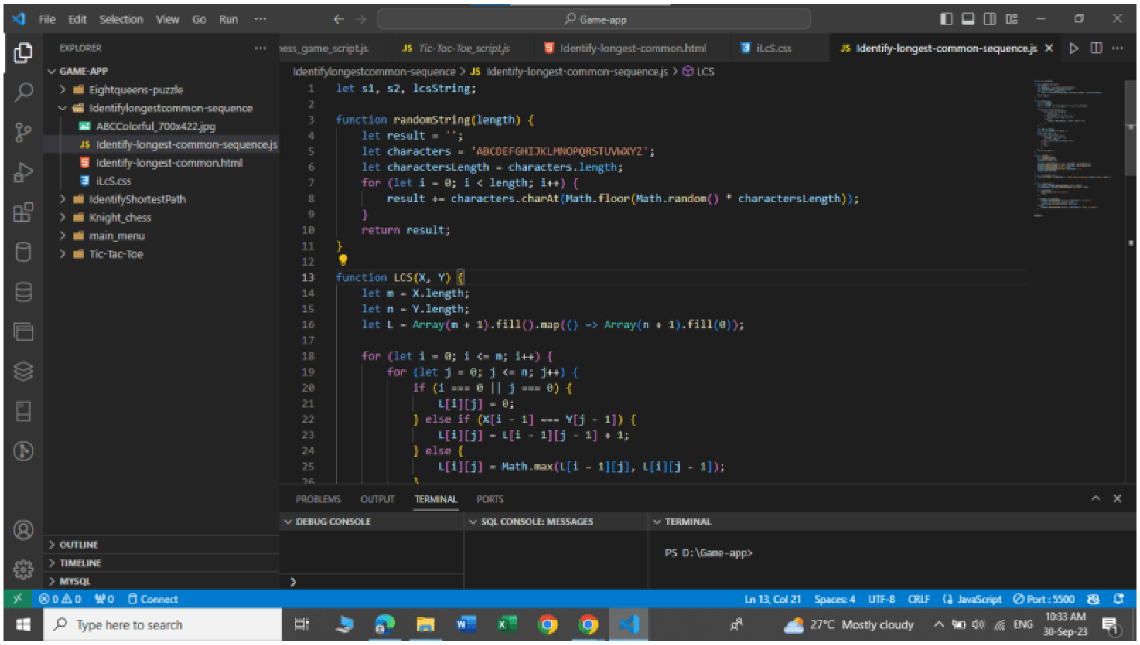
Results:



```
longest seuqence

|

expected results
seuquence in two string
it('should find the longest common sequence in two strings', () => {
  const string1 = 'ABCD';
  const string2 = 'ACDF';

  const result = findLongestCommonSequence(string1, string2);

  // Expected result: 'ACD'
  expect(result).toBe('ACD');
});

empty string

it('should handle empty strings', () => {
  const string1 = '';
  const string2 = 'ACDF';

  const result = findLongestCommonSequence(string1, string2);

  // Expected result: ''
  expect(result).toBe('');
});

actual results

seuquence two strings return the ACD

empty string return ''
```

Figure 15

4.2.5 Indicate the Data Structures used with its purpose

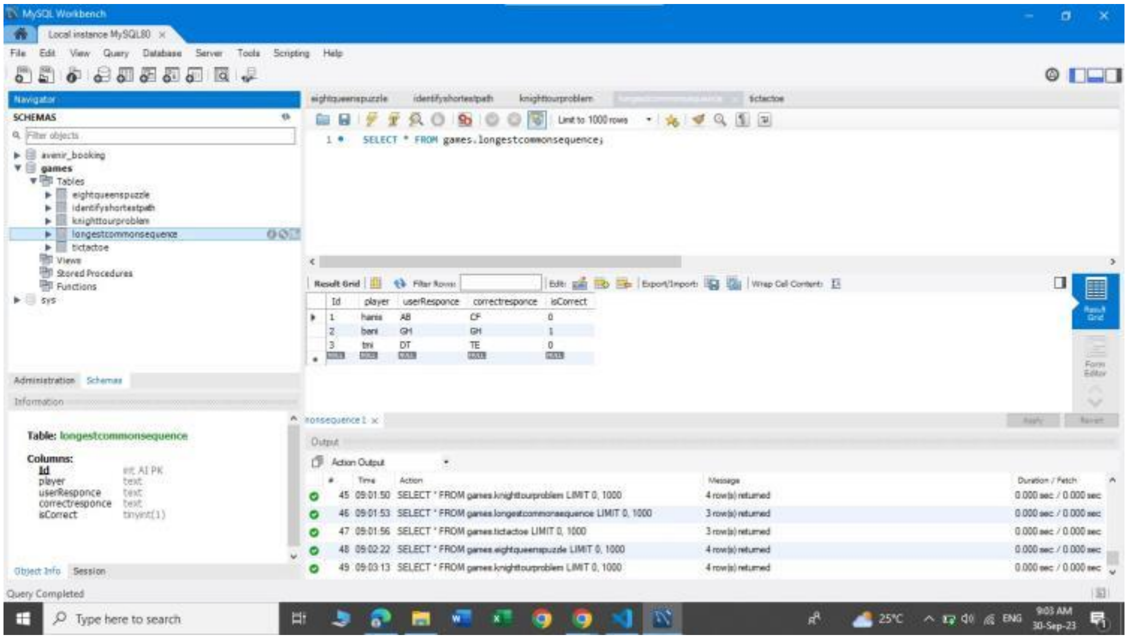


```
let s1, s2, lcsString;

function randomString(length) {
    let result = '';
    let characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    let charactersLength = characters.length;
    for (let i = 0; i < length; i++) {
        result += characters.charAt(Math.floor(Math.random() * charactersLength));
    }
    return result;
}

function LCS(X, Y) {
    let m = X.length;
    let n = Y.length;
    let L = Array(m + 1).fill().map(() => Array(n + 1).fill(0));

    for (let i = 0; i <= m; i++) {
        for (let j = 0; j <= n; j++) {
            if (i === 0 || j === 0) {
                L[i][j] = 0;
            } else if (X[i - 1] === Y[j - 1]) {
                L[i][j] = L[i - 1][j - 1] + 1;
            } else {
                L[i][j] = Math.max(L[i - 1][j], L[i][j - 1]);
            }
```

```
PS D:\Game-app>
```

Figure 16

4.2.6 Normalized DB Table Structure



```
SELECT * FROM games.longestcommonsequence;
```

Id	player	userResponce	correctresponce	isCorrect
1	haris	AB	CF	0
2	bani	GH	GH	1
3	tni	DT	TE	0
NULL	NULL	NULL	NULL	NULL

Table: longestcommonsequence

Columns:

Id	int AI PK
player	text
userResponce	text
correctresponce	text
isCorrect	tinyint(1)

#	Time	Action	Message	Duration / Fetch
45	09:01:50	SELECT * FROM games.knighttourproblem LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
46	09:01:53	SELECT * FROM games.longestcommonsequence LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
47	09:01:56	SELECT * FROM games.tictactoe LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
48	09:02:22	SELECT * FROM games.eightqueenspuzzle LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
49	09:03:13	SELECT * FROM games.knighttourproblem LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec

Figure 17

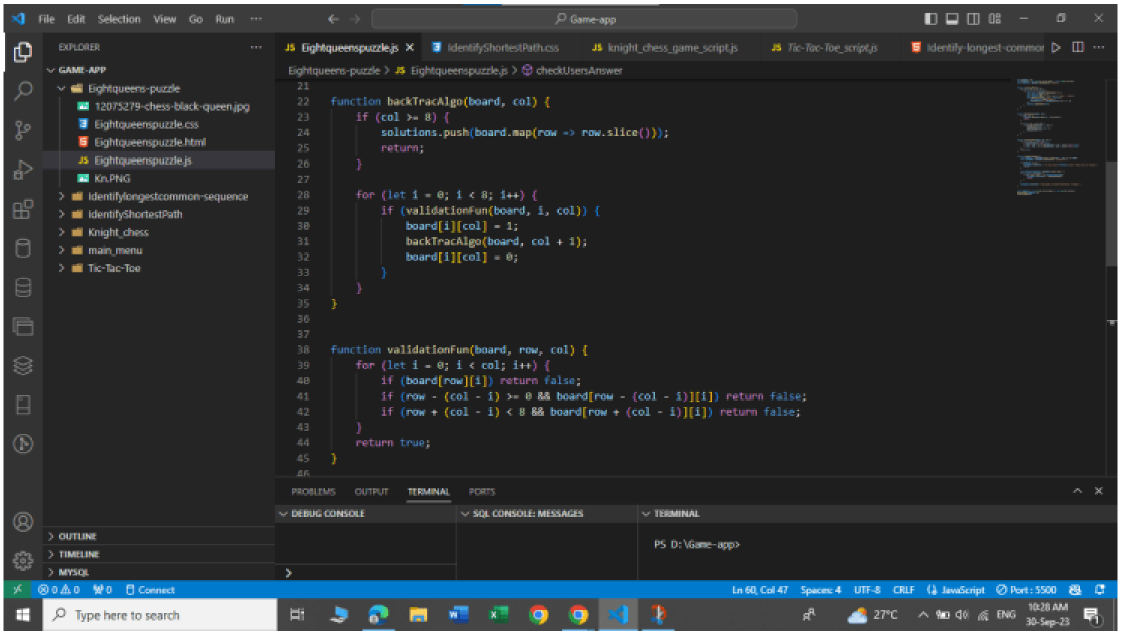
4.3 Eight queens' puzzle

4.3.1 Program Logic used to solve the problem

In basically on this game use chessboard as well as the eight queens. That's the reason game name became "Eight queen's puzzle". Here we have used data structures and algorithms to develop this game and solve the problem in the game.

The algorithm used for the solve problem is "Recursive backtracking algorithm" and used eight by eight chessboard for develop the game.

Here is the logic used to the solve the problem:

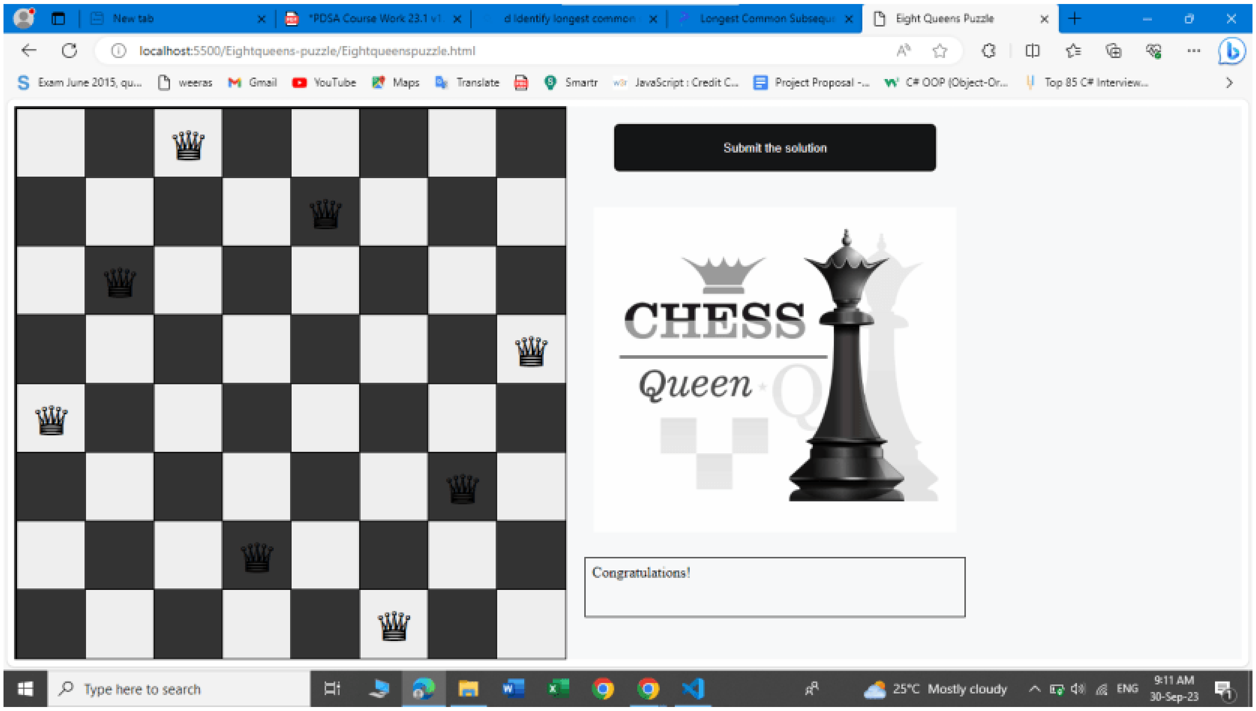


```javascript

function backTracAlgo(board, col) {
    if (col >= 8) {
        solutions.push(board.map(row => row.slice()));
        return;
    }

    for (let i = 0; i < 8; i++) {
        if (validationFun(board, i, col)) {
            board[i][col] = 1;
            backTracAlgo(board, col + 1);
            board[i][col] = 0;
        }
    }
}


function validationFun(board, row, col) {
    for (let i = 0; i < col; i++) {
        if (board[row][i]) return false;
        if (row - (col - i) >= 0 && board[row - (col - i)][i]) return false;
        if (row + (col - i) < 8 && board[row + (col - i)][i]) return false;
    }
    return true;
}

```

```
PS D:\Game-app>
```

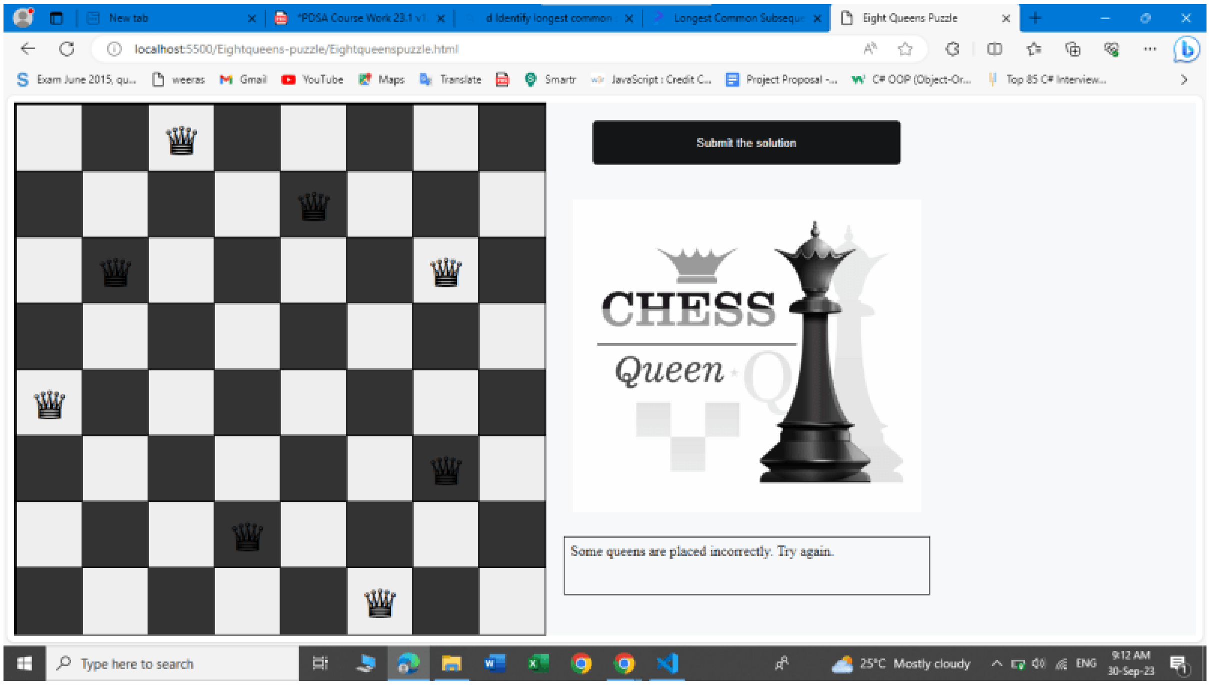
Figure 18

4.3.2 User Interfaces



Congratulations!

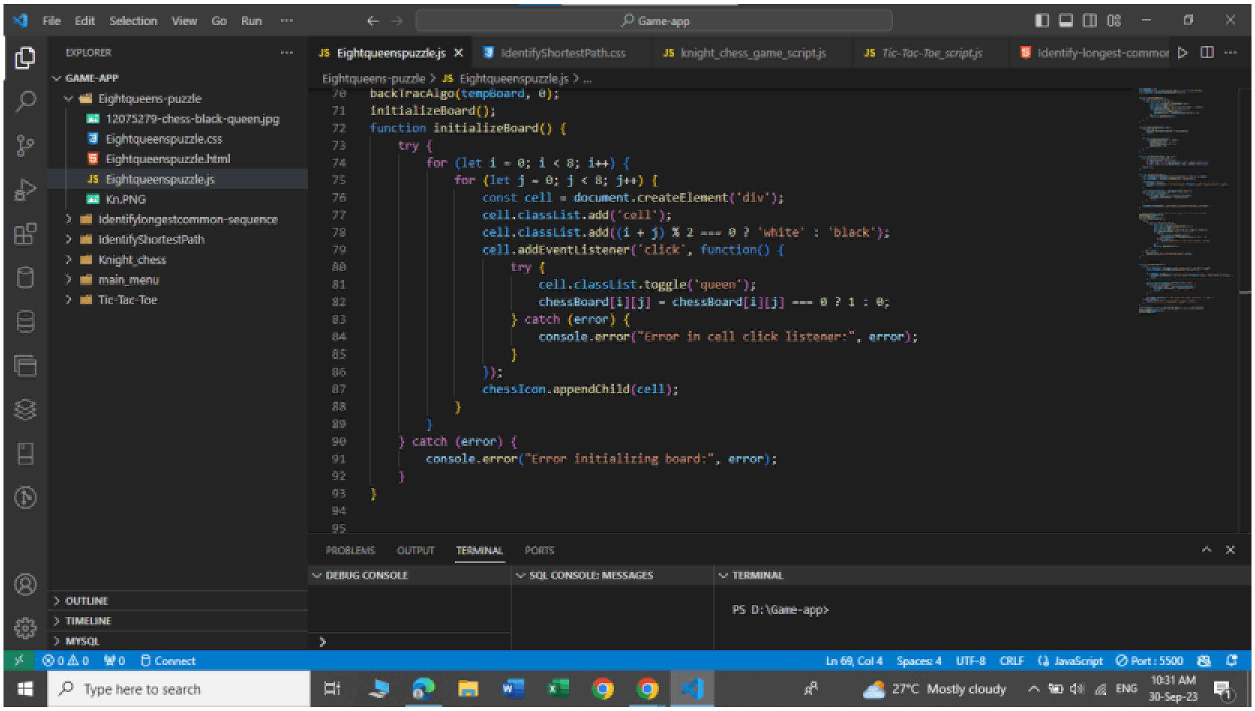
Figure



Some queens are placed incorrectly. Try again.

Figure 19

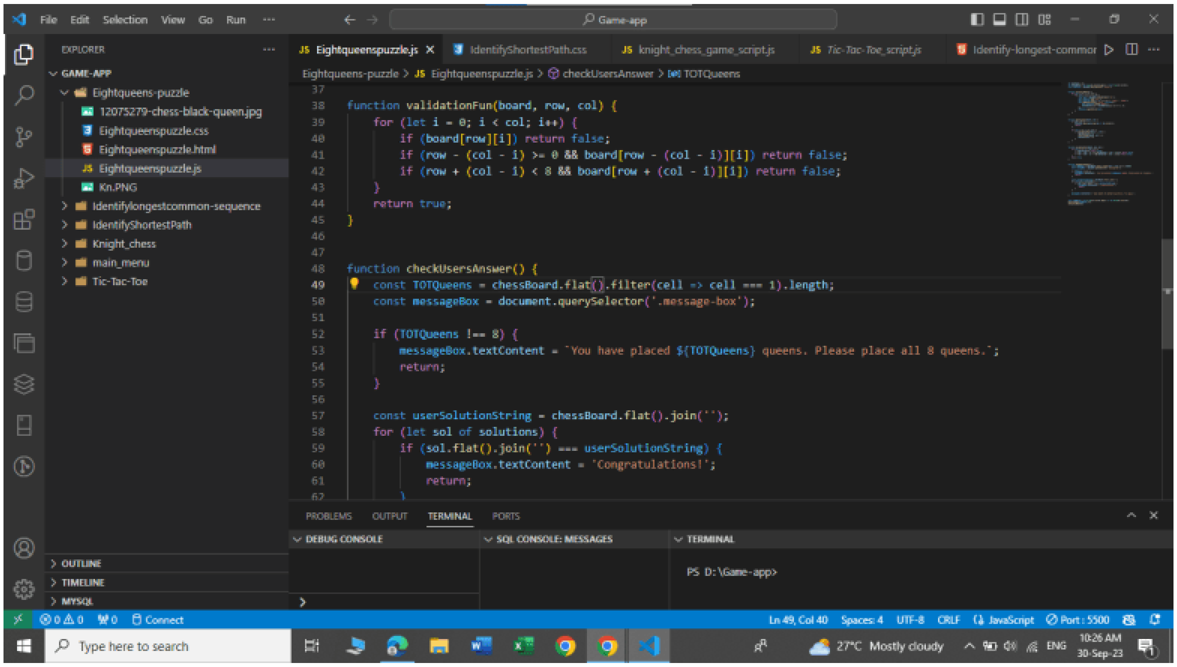
4.3.3 Validations and Exception Handling in this application



```
backTracAlgo(tempBoard, 0);
initializeBoard();
function initializeBoard() {
    try {
        for (let i = 0; i < 8; i++) {
            for (let j = 0; j < 8; j++) {
                const cell = document.createElement('div');
                cell.classList.add('cell');
                cell.classList.add((i + j) % 2 === 0 ? 'white' : 'black');
                cell.addEventListener('click', function() {
                    try {
                        cell.classList.toggle('queen');
                        chessBoard[i][j] = chessBoard[i][j] === 0 ? 1 : 0;
                    } catch (error) {
                        console.error("Error in cell click listener:", error);
                    }
                });
                chessIcon.appendChild(cell);
            }
        }
    } catch (error) {
        console.error("Error initializing board:", error);
    }
}
```

```
PS D:\Game-app>
```

Figure 20



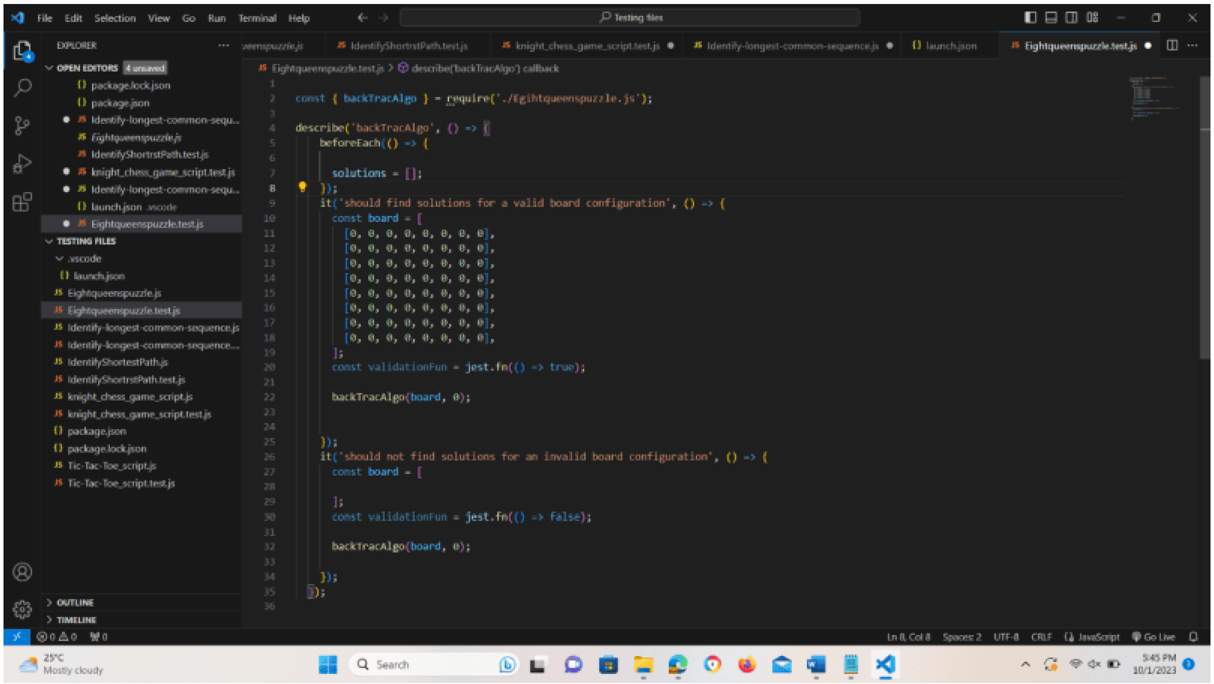
```

function validationFun(board, row, col) {
    for (let i = 0; i < col; i++) {
        if (board[row][i]) return false;
        if (row - (col - i) >= 0 && board[row - (col - i)][i]) return false;
        if (row + (col - i) < 8 && board[row + (col - i)][i]) return false;
    }
    return true;
}


function checkUsersAnswer() {
    const TOTQueens = chessBoard.flat().filter(cell => cell === 1).length;
    const messageBox = document.querySelector('.message-box');

    if (TOTQueens !== 8) {
        messageBox.textContent = `You have placed ${TOTQueens} queens. Please place all 8 queens.`;
        return;
    }

    const userSolutionString = chessBoard.flat().join('');
    for (let sol of solutions) {
        if (sol.flat().join('') === userSolutionString) {
            messageBox.textContent = 'Congratulations!';
            return;
        }
```

```
PS D:\Game-app>
```

Figure 21

4.3.4 Unit Testing

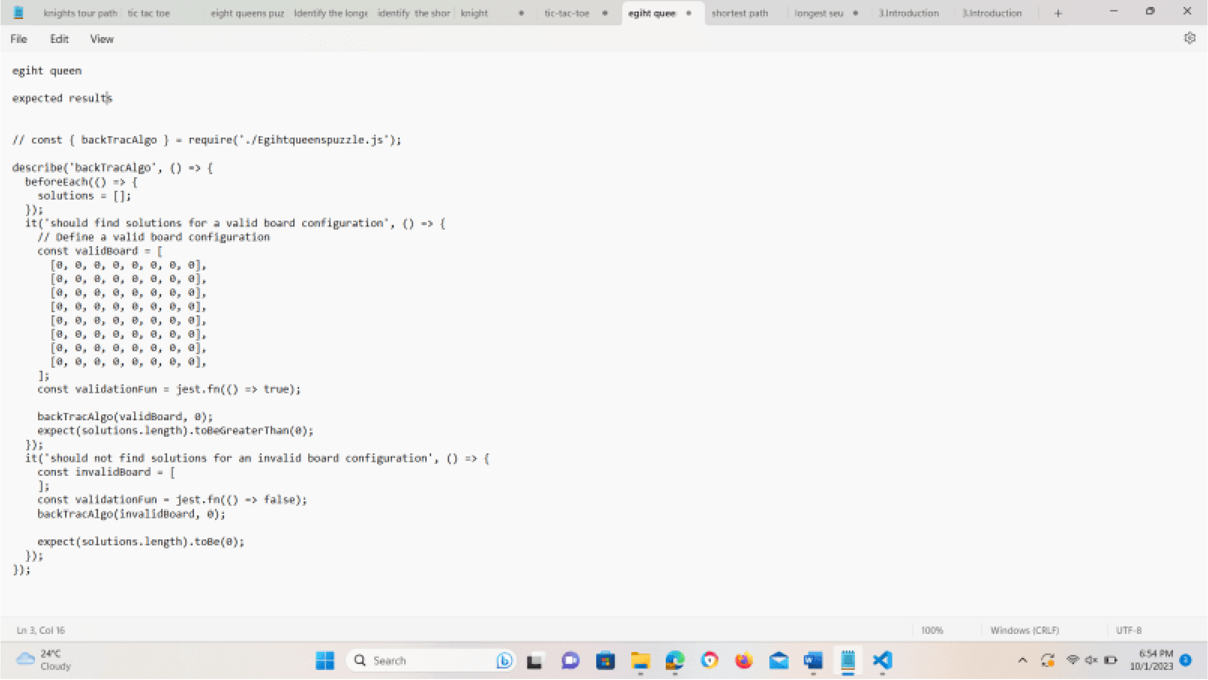
Testing code:



```
const { backTracAlgo } = require('./Egihtqueenspuzzle.js');

describe('backTracAlgo', () => {
  beforeEach(() => {

    solutions = [];
  });
  it('should find solutions for a valid board configuration', () => {
    const board = [
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
    ];
    const validationFun = jest.fn(() => true);

    backTracAlgo(board, 0);


  });
  it('should not find solutions for an invalid board configuration', () => {
    const board = [

    ];
    const validationFun = jest.fn(() => false);

    backTracAlgo(board, 0);

  });
});
```

Figure 22

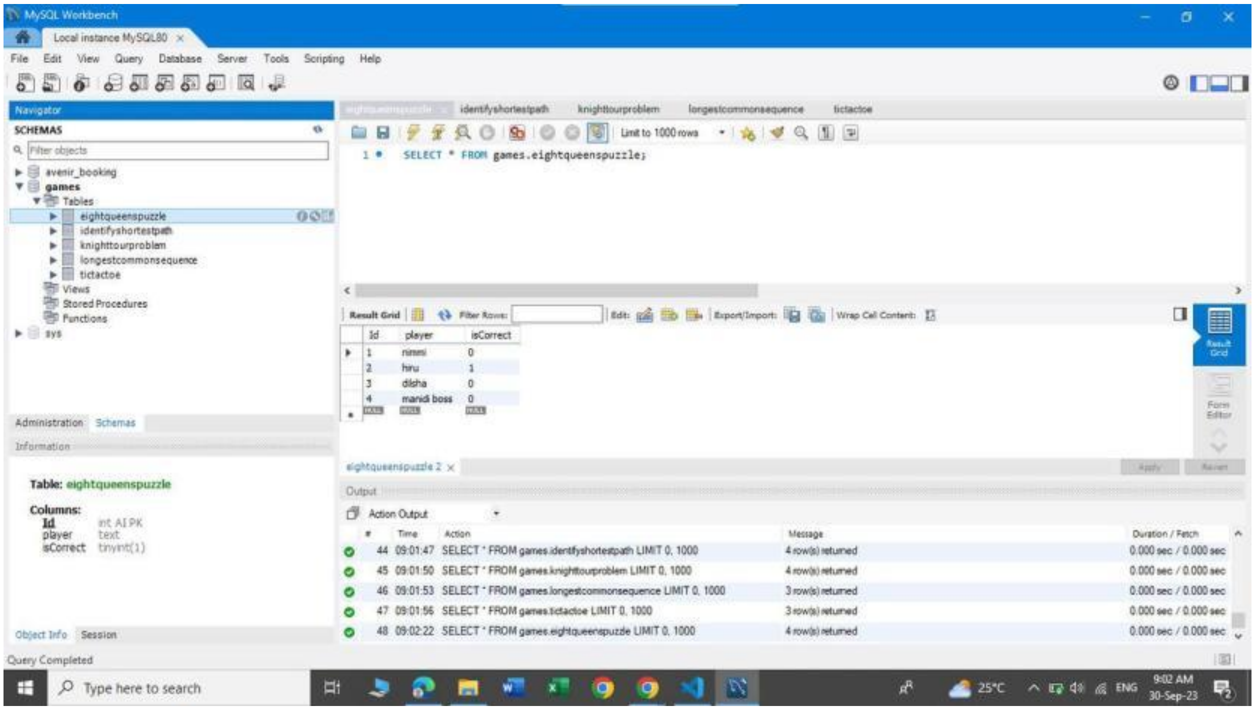
Results:



```
egiht queen

expected results


// const { backTracAlgo } = require('./Egihtqueenspuzzle.js');

describe('backTracAlgo', () => {
  beforeEach(() => {
    solutions = [];
  });
  it('should find solutions for a valid board configuration', () => {
    // Define a valid board configuration
    const validBoard = [
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
    ];
    const validationFun = jest.fn(() => true);

    backTracAlgo(validBoard, 0);
    expect(solutions.length).toBeGreaterThan(0);
  });
  it('should not find solutions for an invalid board configuration', () => {
    const invalidBoard = [
    ];
    const validationFun = jest.fn(() => false);
    backTracAlgo(invalidBoard, 0);

    expect(solutions.length).toBe(0);
  });
});
```

Figure 23

4.3.5 Normalized DB Table Structure



```
1 ● SELECT * FROM games.eightqueenspuzzle;
```

Id	player	isCorrect
1	nimmi	0
2	hiru	1
3	dilsha	0
4	manidi boss	0
NULL	NULL	NULL

Table: eightqueenspuzzle

Columns:

Id	int AI PK
player	text
isCorrect	tinyint(1)

Output

Action Output

#	Time	Action	Message	Duration / Fetch
44	09:01:47	SELECT * FROM games.identifyshortestpath LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
45	09:01:50	SELECT * FROM games.knighttourproblem LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
46	09:01:53	SELECT * FROM games.longestcommonsequence LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
47	09:01:56	SELECT * FROM games.tictactoe LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
48	09:02:22	SELECT * FROM games.eightqueenspuzzle LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec

Figure 24

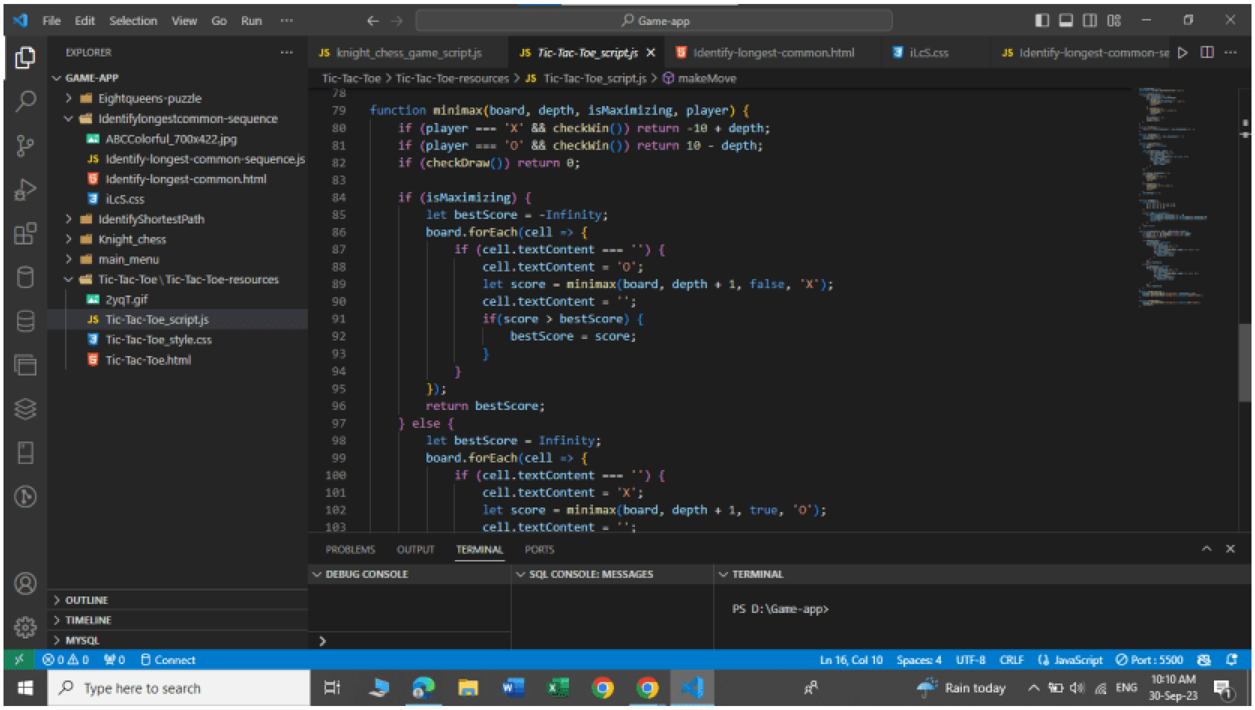
4.4 Tic-Tac-Toe

4.4.1 Program Logic used to solve the problem

On this game I have used min max algorithm to use the problem. There I was designed optimal moves that could have a single player. And I can assume to get win that when I can assume the optimal move by using min max algorithm.

Firstly, I must create a function as a make move. And then I consider which is the best path to defeat the opponent by studying the square where he moves.in min max algorithm its recursively call the next players move. Function will be return a negative value or positive value to check whether winning status achieved or not. If it will be a draw. I would return zero.

Here is the logic used to the solve the problem:



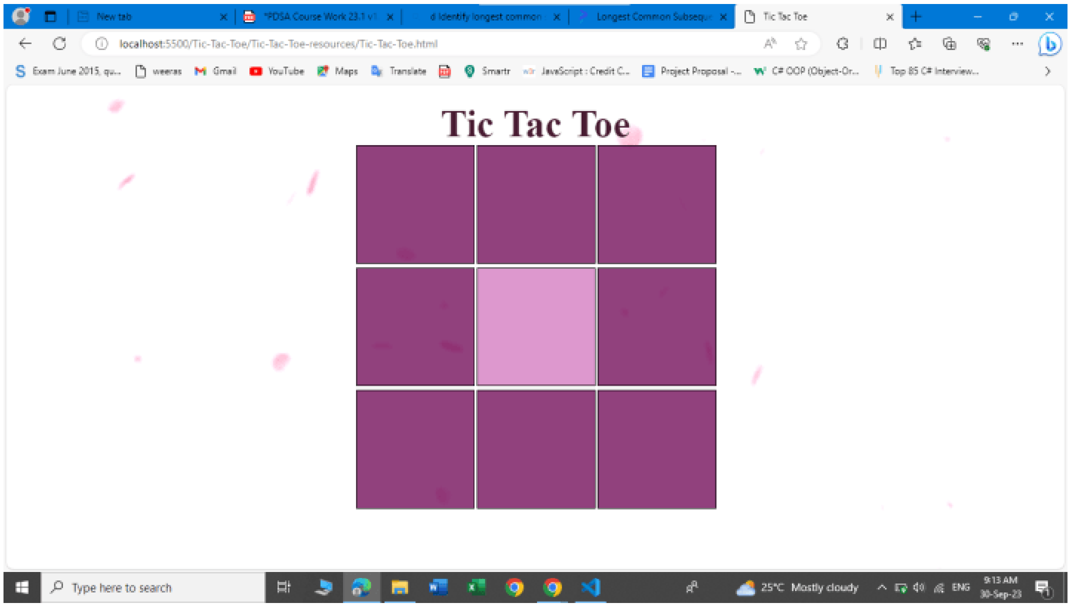
File Edit Selection View Go Run
Game-app
EXPLORER
GAME-APP
Eightqueens-puzzle
Identifylongestcommon-sequence
ABCColorful_700x422.jpg
Identify-longest-common-sequence.js
Identify-longest-common.html
iLcS.css
IdentifyShortestPath
Knight_chess
main_menu
Tic-Tac-Toe\Tic-Tac-Toe-resources
2yqT.gif
Tic-Tac-Toe_script.js
Tic-Tac-Toe_style.css
Tic-Tac-Toe.html
OUTLINE
TIMELINE
MYSQL
knight_chess_game_script.js
Tic-Tac-Toe_script.js
Identify-longest-common.html
iLcS.css
Identify-longest-common-se
Tic-Tac-Toe > Tic-Tac-Toe-resources > Tic-Tac-Toe_script.js > makeMove
PROBLEMS OUTPUT TERMINAL PORTS
DEBUG CONSOLE
SQL CONSOLE: MESSAGES
TERMINAL
Ln 16, Col 10 Spaces: 4 UTF-8 CRLF JavaScript Port : 5500
Type here to search
Rain today
ENG
10:10 AM
30-Sep-23

```
function minimax(board, depth, isMaximizing, player) {
    if (player === 'X' && checkWin()) return -10 + depth;
    if (player === 'O' && checkWin()) return 10 - depth;
    if (checkDraw()) return 0;

    if (isMaximizing) {
        let bestScore = -Infinity;
        board.forEach(cell => {
            if (cell.textContent === '') {
                cell.textContent = 'O';
                let score = minimax(board, depth + 1, false, 'X');
                cell.textContent = '';
                if(score > bestScore) {
                    bestScore = score;
                }
            }
        });
        return bestScore;
    } else {
        let bestScore = Infinity;
        board.forEach(cell => {
            if (cell.textContent === '') {
                cell.textContent = 'X';
                let score = minimax(board, depth + 1, true, 'O');
                cell.textContent = '';
```

```
PS D:\Game-app>
```

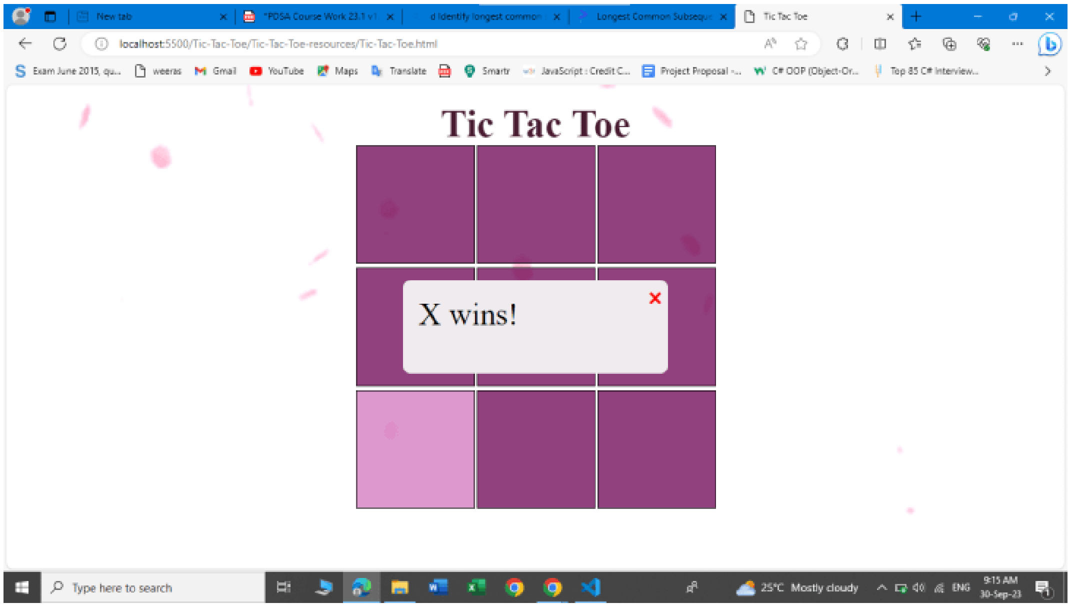
Figure 24

4.4.2 User Interfaces


localhost:5500/Tic-Tac-Toe/Tic-Tac-Toe-resources/Tic-Tac-Toe.html
Tic Tac Toe
Type here to search
25°C Mostly cloudy
9:13 AM
30-Sep-23

Tic Tac Toe

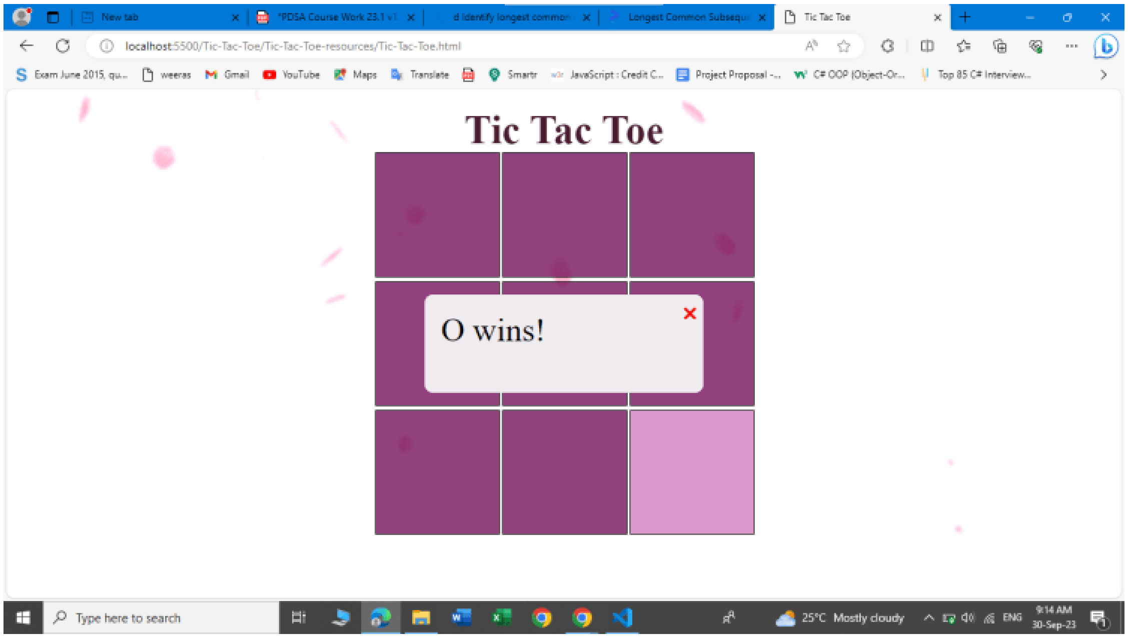
Figure 25


localhost:5500/Tic-Tac-Toe/Tic-Tac-Toe-resources/Tic-Tac-Toe.html
Tic Tac Toe
×
Type here to search
25°C Mostly cloudy
9:15 AM
30-Sep-23

Tic Tac Toe

X wins!

Figure 26



Tic Tac Toe

O wins!

Figure 27

4.4.3 Validations and Exception Handling in this application

1. Input Validation:

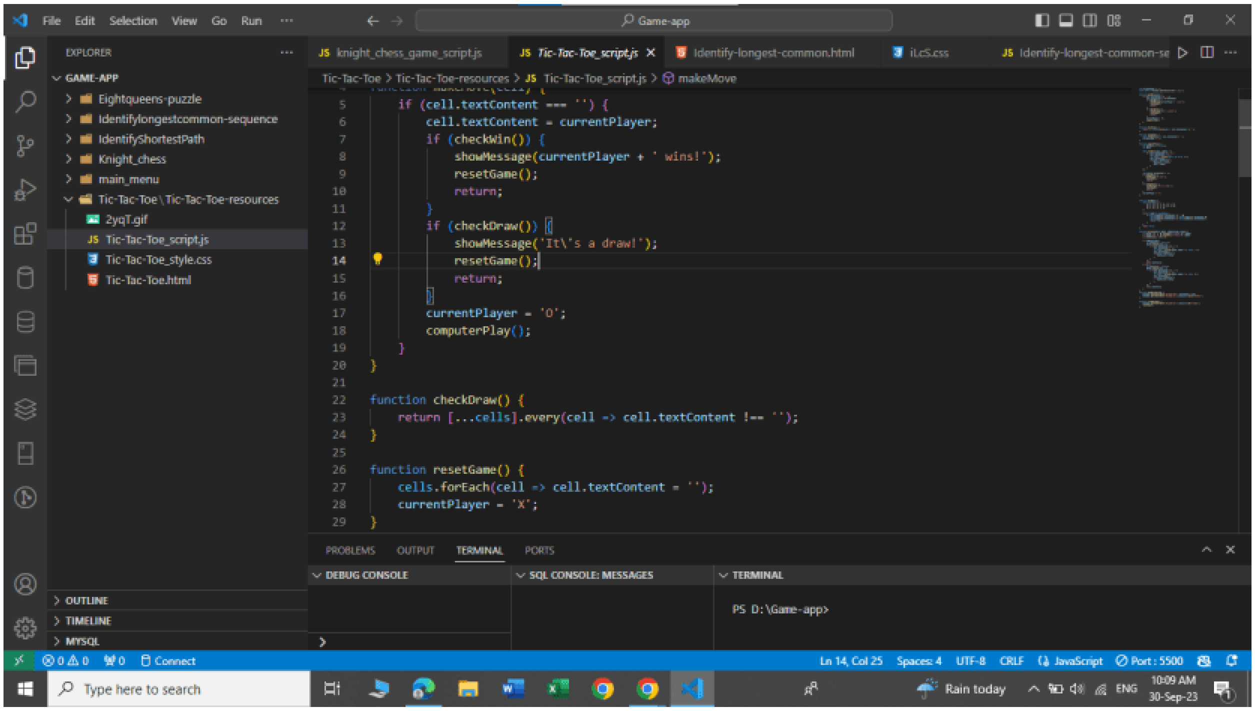
- I codded validation part to ensure that users next move is for an empty cell *(“`cell.textContent === ''`)*

2. Winning and Draw Validation:

- I added another function called `checkWin()`. That function will return +1 for a win -1 for lose and zero for the draw.

4. Error Handling in `computerPlay()`:

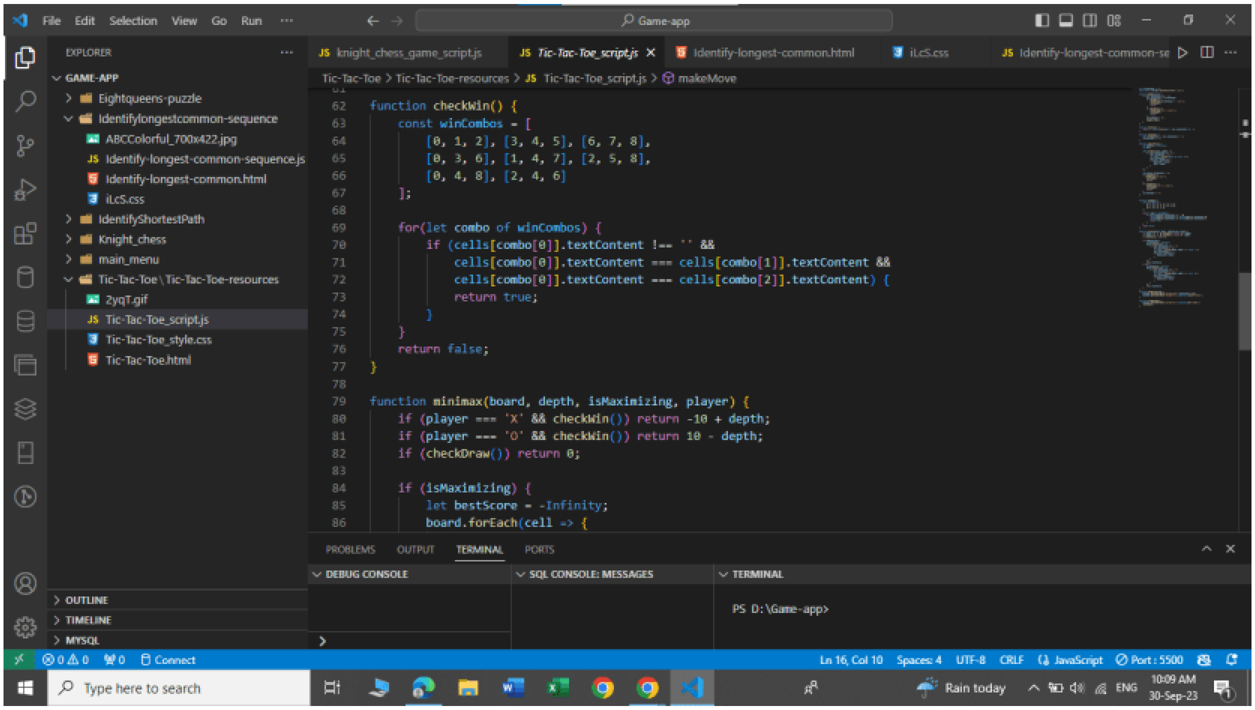
- In `computerPlay()` function I added exception handling to ensure that computers next move is a valid one and make sure to handle if an error occurs in minmax algorithm.


Tic-Tac-Toe > Tic-Tac-Toe-resources > Tic-Tac-Toe_script.js > makeMove
Ln 14, Col 25 Spaces: 4 UTF-8 CRLF JavaScript Port : 5500

```javascript
    if (cell.textContent === '') {
        cell.textContent = currentPlayer;
        if (checkWin()) {
            showMessage(currentPlayer + ' wins!');
            resetGame();
            return;
        }
        if (checkDraw()) {
            showMessage('It\'s a draw!');
            resetGame();
            return;
        }
        currentPlayer = 'O';
        computerPlay();
    }
}

function checkDraw() {
    return [...cells].every(cell => cell.textContent !== '');
}

function resetGame() {
    cells.forEach(cell => cell.textContent = '');
    currentPlayer = 'X';
}
```

```
PS D:\Game-app>
```

Figure 28


Tic-Tac-Toe > Tic-Tac-Toe-resources > Tic-Tac-Toe_script.js > makeMove
Ln 16, Col 10 Spaces: 4 UTF-8 CRLF JavaScript Port : 5500

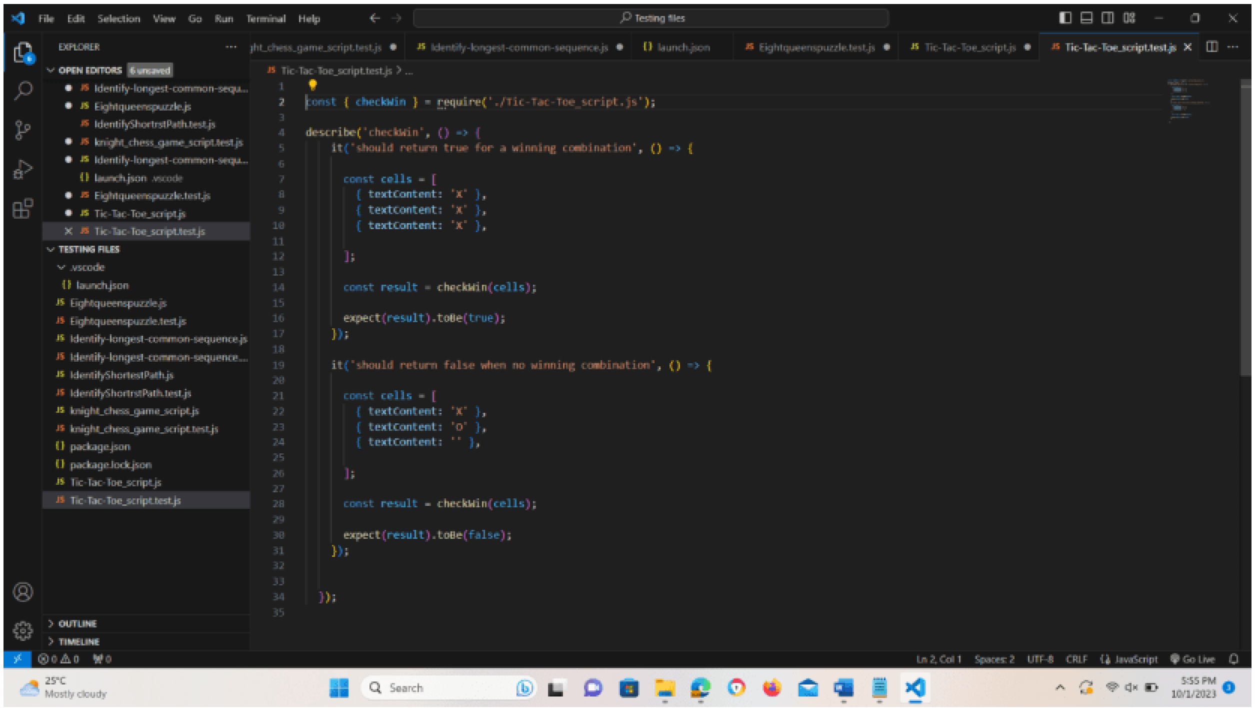
```javascript
function checkWin() {
    const winCombos = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8],
        [0, 3, 6], [1, 4, 7], [2, 5, 8],
        [0, 4, 8], [2, 4, 6]
    ];

    for(let combo of winCombos) {
        if (cells[combo[0]].textContent !== '' &&
            cells[combo[0]].textContent === cells[combo[1]].textContent &&
            cells[combo[0]].textContent === cells[combo[2]].textContent) {
            return true;
        }
    }
    return false;
}

function minimax(board, depth, isMaximizing, player) {
    if (player === 'X' && checkWin()) return -10 + depth;
    if (player === 'O' && checkWin()) return 10 - depth;
    if (checkDraw()) return 0;

    if (isMaximizing) {
        let bestScore = -Infinity;
        board.forEach(cell => {
```

```
PS D:\Game-app>
```

Figure 29

4.4.4 Unit Testing

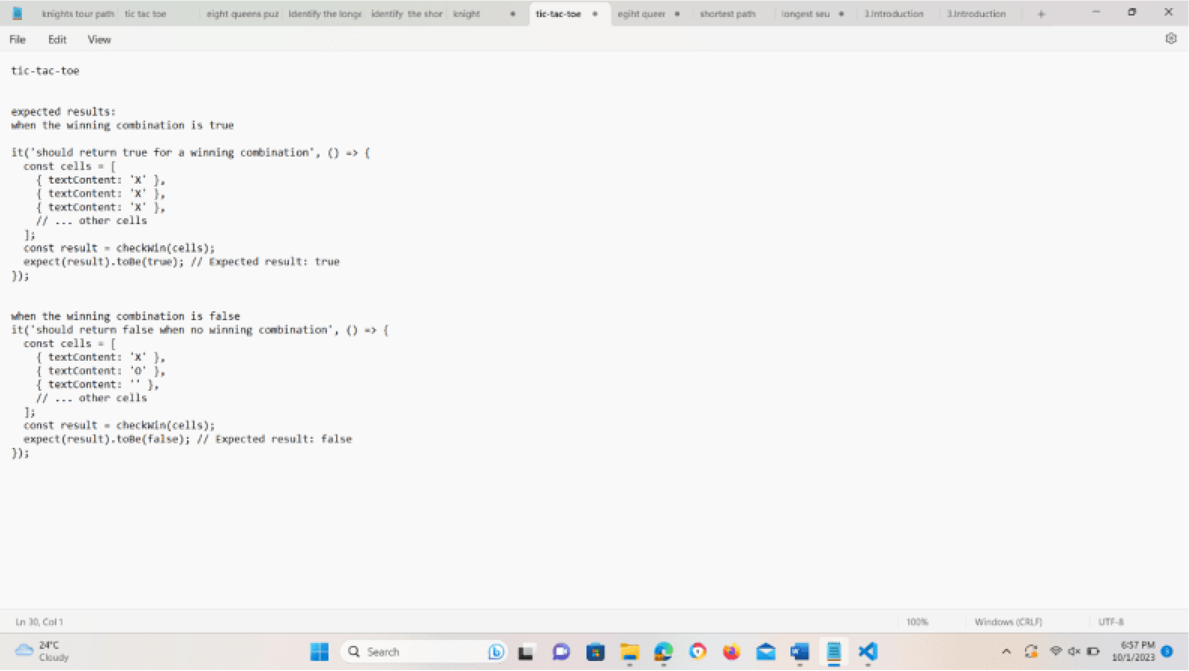
Testing code:



```
const { checkWin } = require('./Tic-Tac-Toe_script.js');

describe('checkWin', () => {
    it('should return true for a winning combination', () => {

      const cells = [
        { textContent: 'X' },
        { textContent: 'X' },
        { textContent: 'X' },

      ];

      const result = checkWin(cells);

      expect(result).toBe(true);
    });

    it('should return false when no winning combination', () => {

      const cells = [
        { textContent: 'X' },
        { textContent: 'O' },
        { textContent: '' },

      ];

      const result = checkWin(cells);

      expect(result).toBe(false);
    });


  });
```

Figure 30

Results:



```
tic-tac-toe


expected results:
when the winning combination is true

it('should return true for a winning combination', () => {
  const cells = [
    { textContent: 'X' },
    { textContent: 'X' },
    { textContent: 'X' },
    // ... other cells
  ];
  const result = checkWin(cells);
  expect(result).toBe(true); // Expected result: true
});


when the winning combination is false
it('should return false when no winning combination', () => {
  const cells = [
    { textContent: 'X' },
    { textContent: 'O' },
    { textContent: '' },
    // ... other cells
  ];
  const result = checkWin(cells);
  expect(result).toBe(false); // Expected result: false
});
```

Figure 31

4.4.5 Indicate the Data Structures used with its purpose

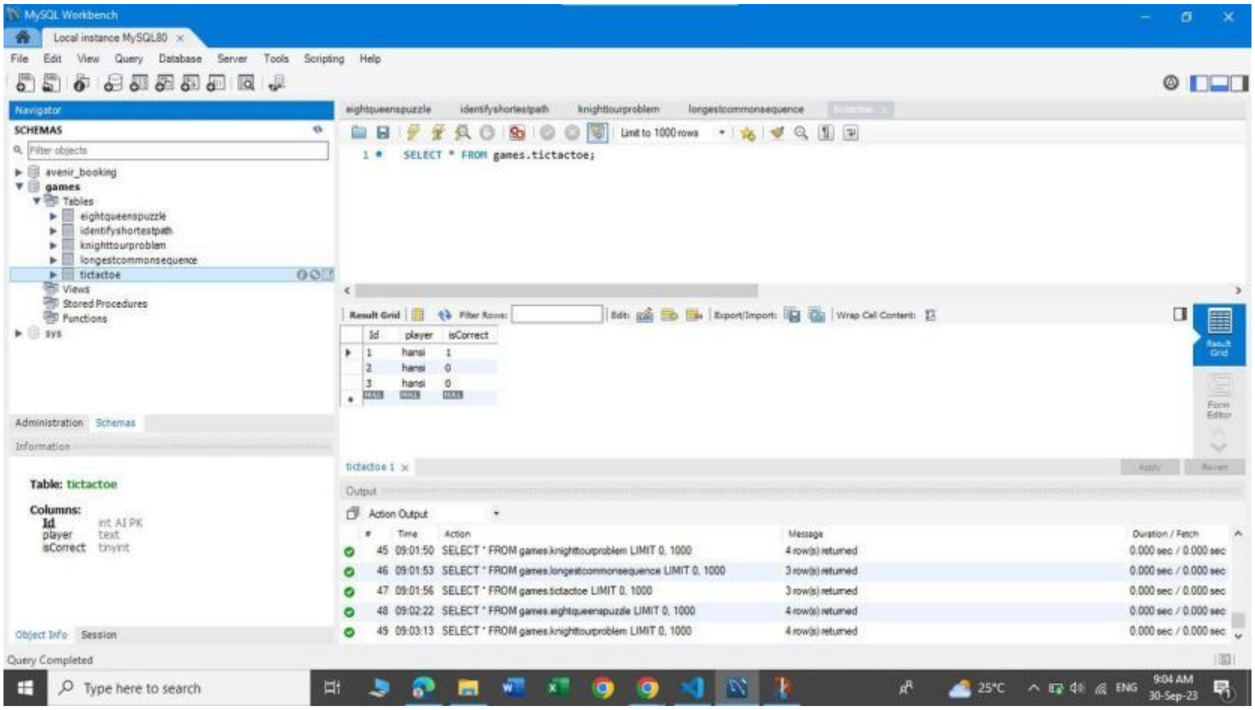
In this case I have used the 9 elements array as a structure. I have used tree data

Structure in minmax algorithm to visualize the game state and represent the node.

There I have 2D array which known as graph.it represents the distance of two cities .and there is a list which consists of 10 cities. And there is a function which is use to randomly assign distances to the graph. And another function used to create a distance table. And there is a function that is used to users for giving there inputs to the server.startGame() function is use to select two cities randomly and put the distance table the distance of two cities randomly .

There is a function for the bellman ford algorithm on that algorithm calculate shortest distance and return the value. Besides that, handle the cell which consists negative value by using digit trace algorithm .it there values is correct those values assign to the database.

4.4.6 Normalized DB Table Structure



```
SELECT * FROM games.tictactoe;
```

Table: tictactoe

Columns:

Id	int AI PK
player	text
isCorrect	tinyint

Id	player	isCorrect
1	hansi	1
2	hansi	0
3	hansi	0
NULL	NULL	NULL

```
#   Time      Action                                                       Message            Duration / Fetch
45  09:01:50  SELECT * FROM games.knighttourproblem LIMIT 0, 1000          4 row(s) returned  0.000 sec / 0.000 sec
46  09:01:53  SELECT * FROM games.longestcommonsequence LIMIT 0, 1000      3 row(s) returned  0.000 sec / 0.000 sec
47  09:01:56  SELECT * FROM games.tictactoe LIMIT 0, 1000                  3 row(s) returned  0.000 sec / 0.000 sec
48  09:02:22  SELECT * FROM games.eightqueenspuzzle LIMIT 0, 1000          4 row(s) returned  0.000 sec / 0.000 sec
49  09:03:13  SELECT * FROM games.knighttourproblem LIMIT 0, 1000          4 row(s) returned  0.000 sec / 0.000 sec
```

Figure 33

4.5 Identify Shortest Path

4.5.1 Program Logic used to solve the problem

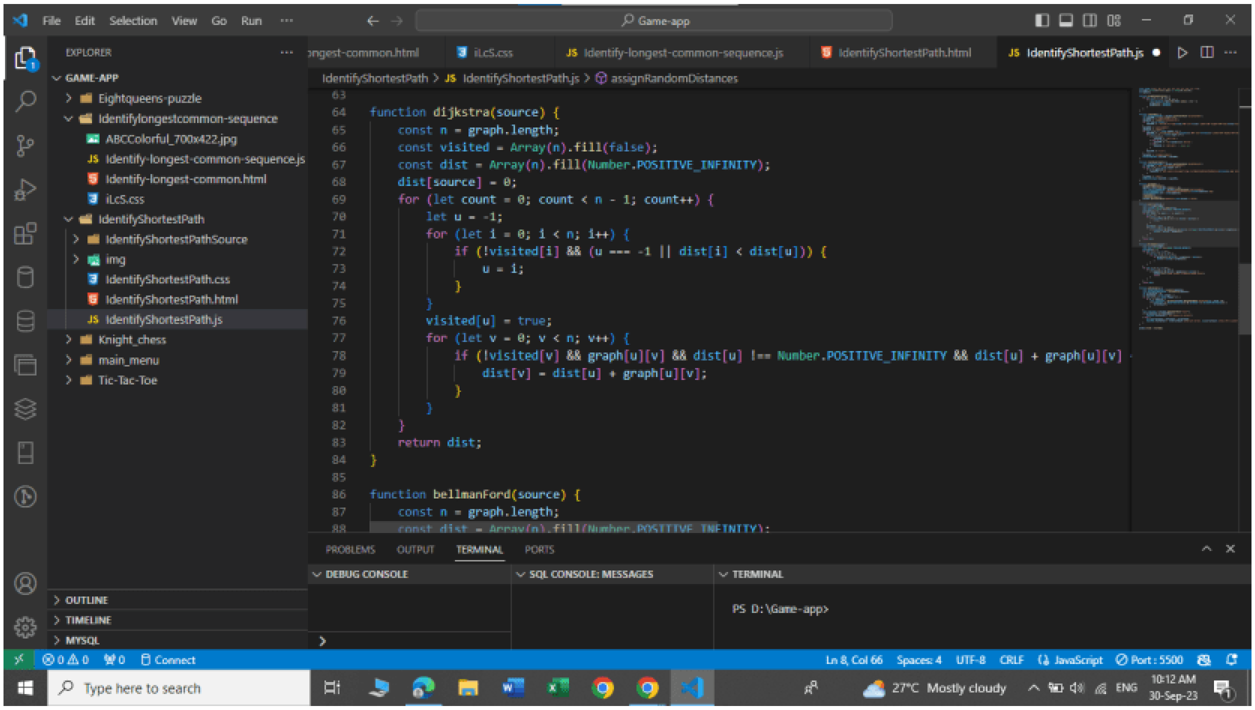
In this case I have used the 9 elements array as a structure. I have used tree data

Structure in minmax algorithm to visualize the game state and represent the node.

There I have 2D array which known as graph.it represents the distance of two cities .and there is a list which consists of 10 cities. And there is a function which is use to randomly assign distances to the graph. And another function used to create a distance table. And there is a function that is used to users for giving there inputs to the server.startGame() function is use to select two cities randomly and put the distance table the distance of two cities randomly .

There is a function for the bellman ford algorithm on that algorithm calculate shortest distance and return the value. Besides that, handle the cell which consists negative value by using digit trace algorithm .it their values is correct those values assign to the database.

Here is the logic used to the solve the problem:

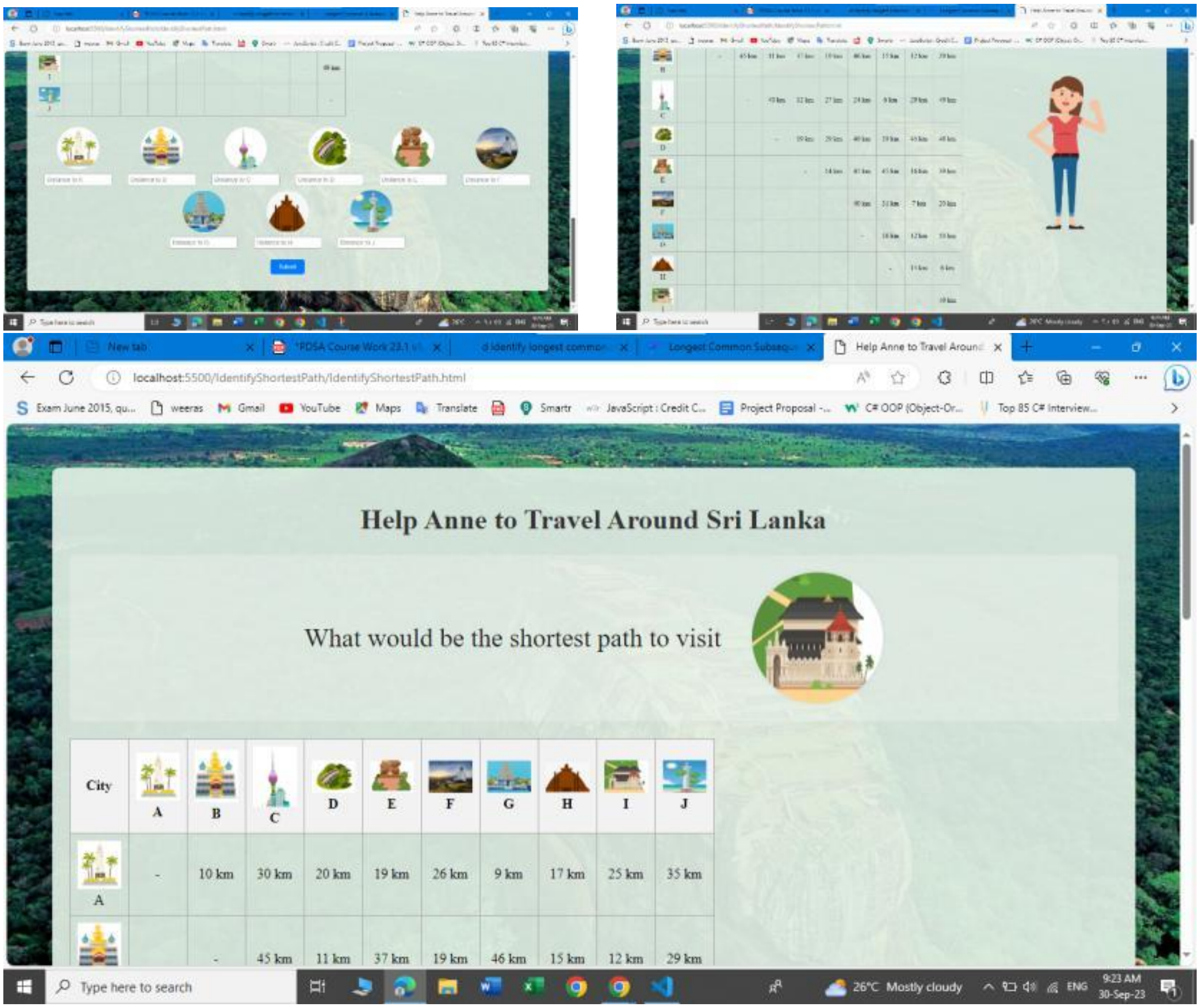


```javascript

function dijkstra(source) {
    const n = graph.length;
    const visited = Array(n).fill(false);
    const dist = Array(n).fill(Number.POSITIVE_INFINITY);
    dist[source] = 0;
    for (let count = 0; count < n - 1; count++) {
        let u = -1;
        for (let i = 0; i < n; i++) {
            if (!visited[i] && (u === -1 || dist[i] < dist[u])) {
                u = i;
            }
        }
        visited[u] = true;
        for (let v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] && dist[u] !== Number.POSITIVE_INFINITY && dist[u] + graph[u][v]
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
    return dist;
}

function bellmanFord(source) {
    const n = graph.length;
    const dist = Array(n).fill(Number.POSITIVE_INFINITY);
```

```
PS D:\Game-app>
```

Figure 34

4.5.2 User Interfaces



Help Anne to Travel Around Sri Lanka

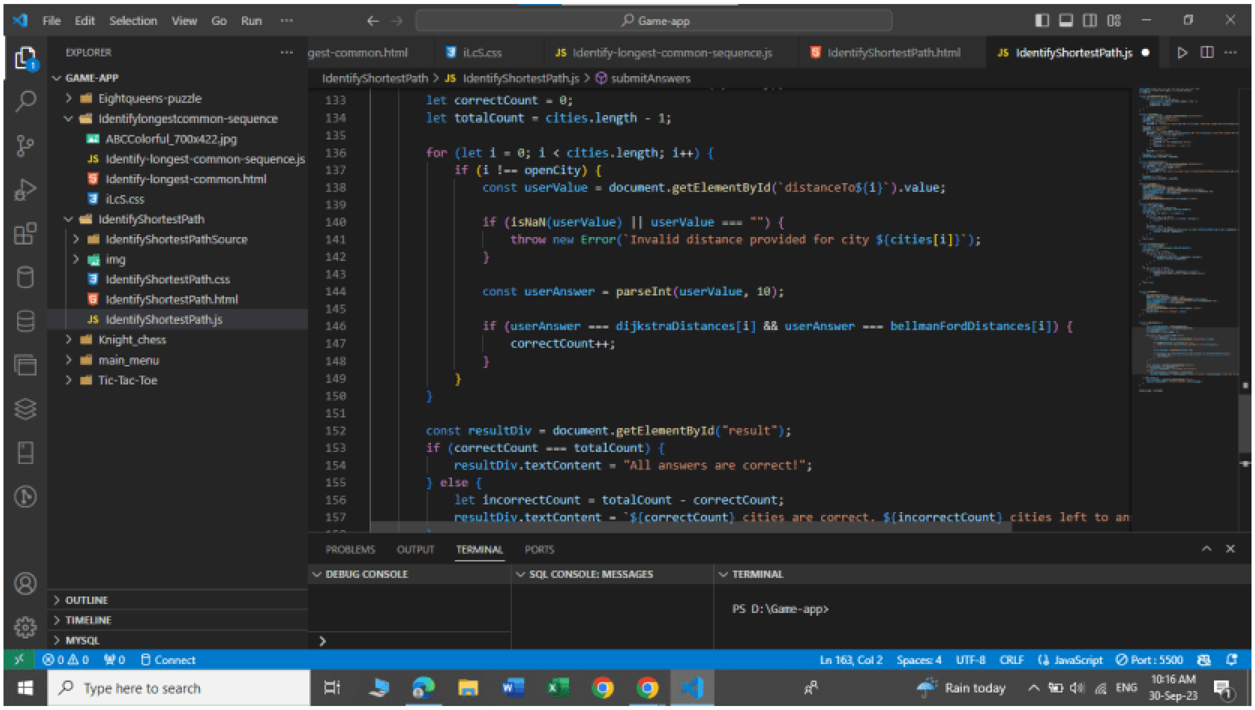
What would be the shortest path to visit

City	A	B	C	D	E	F	G	H	I	J
A	-	10 km	30 km	20 km	19 km	26 km	9 km	17 km	25 km	35 km
		-	45 km	11 km	37 km	19 km	46 km	15 km	12 km	29 km

localhost:5500/IdentifyShortestPath/IdentifyShortestPath.html
Help Anne to Travel Around
Type here to search
26°C Mostly cloudy
9:23 AM
30-Sep-23

Figure 35

4.5.3 Validations and Exception Handling in this application



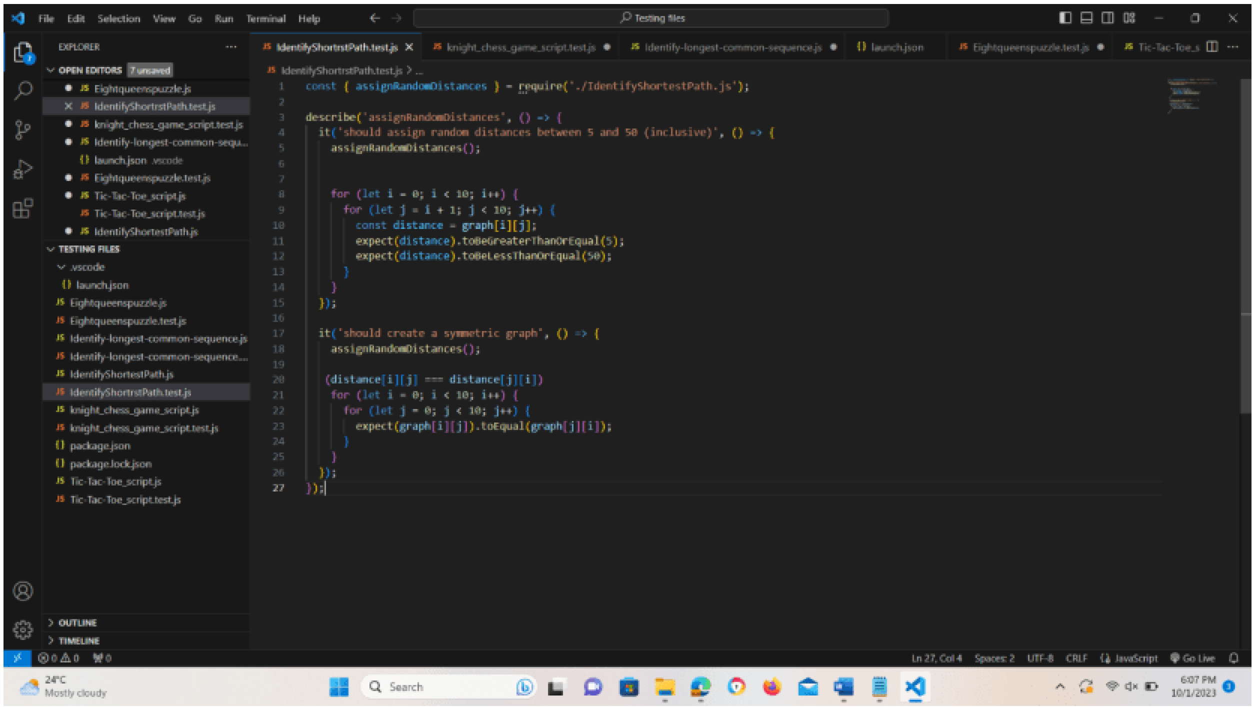
```
let correctCount = 0;
let totalCount = cities.length - 1;

for (let i = 0; i < cities.length; i++) {
    if (i !== openCity) {
        const userValue = document.getElementById(`distanceTo${i}`).value;

        if (isNaN(userValue) || userValue === "") {
            throw new Error(`Invalid distance provided for city ${cities[i]}`);
        }

        const userAnswer = parseInt(userValue, 10);

        if (userAnswer === dijkstraDistances[i] && userAnswer === bellmanFordDistances[i]) {
            correctCount++;
        }
    }
}

const resultDiv = document.getElementById("result");
if (correctCount === totalCount) {
    resultDiv.textContent = "All answers are correct!";
} else {
    let incorrectCount = totalCount - correctCount;
    resultDiv.textContent = `${correctCount} cities are correct. ${incorrectCount} cities left to an
```

```
PS D:\Game-app>
```

Figure 36

4.5.4 Unit Testing

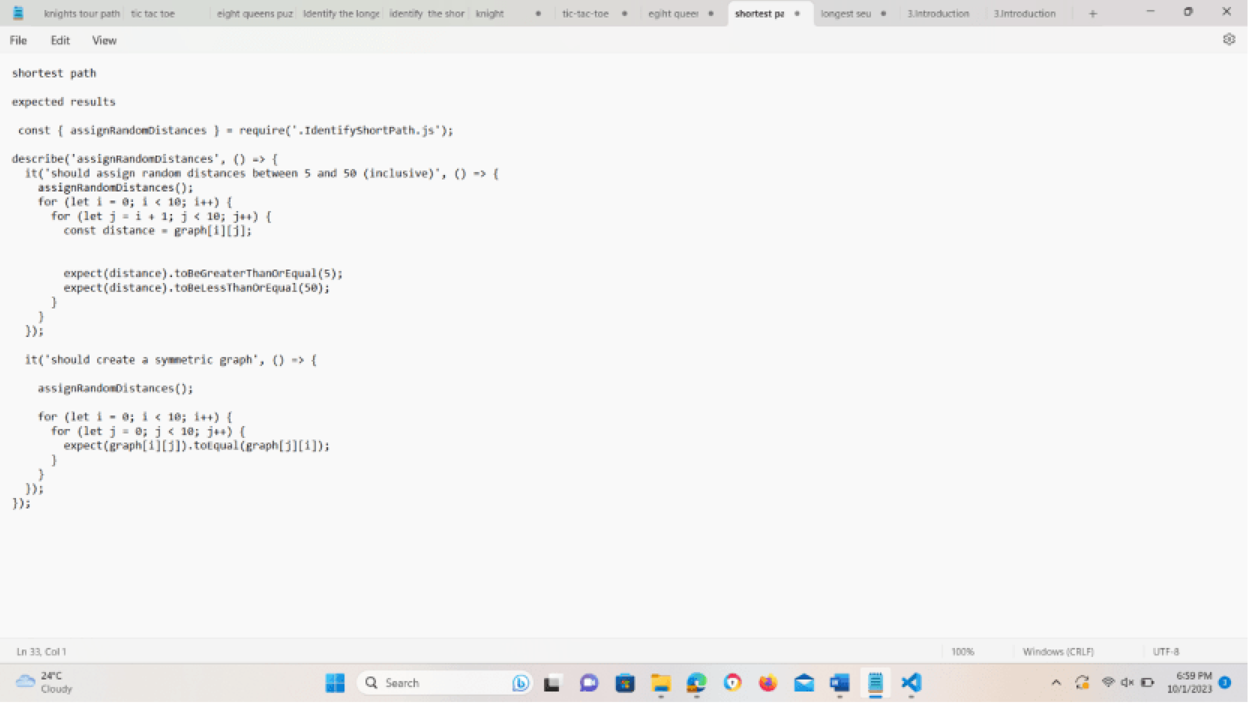
Testing code:



```
const { assignRandomDistances } = require('./IdentifyShortestPath.js');

describe('assignRandomDistances', () => {
  it('should assign random distances between 5 and 50 (inclusive)', () => {
    assignRandomDistances();


    for (let i = 0; i < 10; i++) {
      for (let j = i + 1; j < 10; j++) {
        const distance = graph[i][j];
        expect(distance).toBeGreaterThanOrEqual(5);
        expect(distance).toBeLessThanOrEqual(50);
      }
    }
  });

  it('should create a symmetric graph', () => {
    assignRandomDistances();

   (distance[i][j] === distance[j][i])
    for (let i = 0; i < 10; i++) {
      for (let j = 0; j < 10; j++) {
        expect(graph[i][j]).toEqual(graph[j][i]);
      }
    }
  });
});
```

Figure 37

Results:



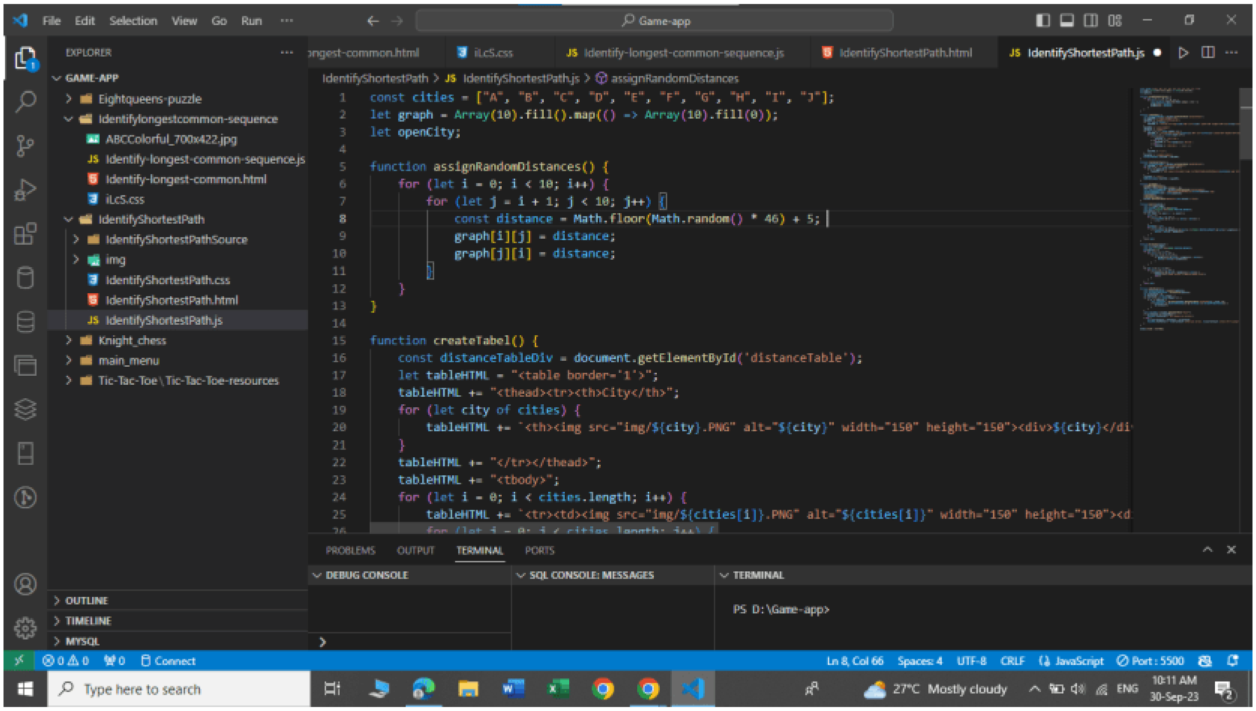
```
shortest path

expected results

 const { assignRandomDistances } = require('.IdentifyShortPath.js');

describe('assignRandomDistances', () => {
  it('should assign random distances between 5 and 50 (inclusive)', () => {
    assignRandomDistances();
    for (let i = 0; i < 10; i++) {
      for (let j = i + 1; j < 10; j++) {
        const distance = graph[i][j];


        expect(distance).toBeGreaterThanOrEqual(5);
        expect(distance).toBeLessThanOrEqual(50);
      }
    }
  });

  it('should create a symmetric graph', () => {

    assignRandomDistances();

    for (let i = 0; i < 10; i++) {
      for (let j = 0; j < 10; j++) {
        expect(graph[i][j]).toEqual(graph[j][i]);
      }
    }
  });
});
```

Figure 38

4.5.5 Indicate the Data Structures used with its purpose

And there is a function which is use to randomly assign distances to the graph. And another function used to create a distance table. And there is a function that is used to users for giving there inputs to the server.startGame() function is use to select two cities randomly and put the distance table the distance of two cities randomly .

There is a function for the bellman ford algorithm on that algorithm calculate shortest distance and return the value. Besides that, handle the cell which consists negative value by using digit trace algorithm .it there values is correct those values assign to the database.

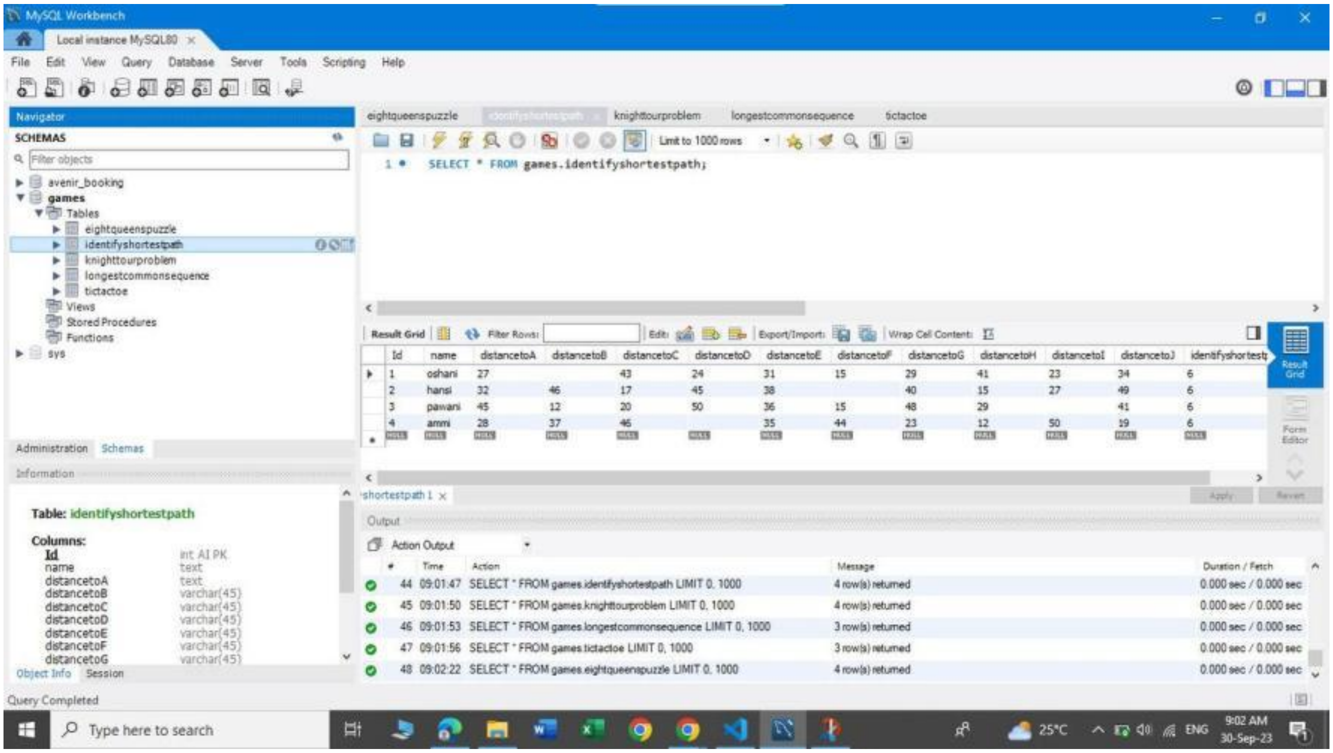


```
const cities = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"];
let graph = Array(10).fill().map(() => Array(10).fill(0));
let openCity;

function assignRandomDistances() {
    for (let i = 0; i < 10; i++) {
        for (let j = i + 1; j < 10; j++) {
            const distance = Math.floor(Math.random() * 46) + 5;
            graph[i][j] = distance;
            graph[j][i] = distance;
        }
    }
}

function createTabel() {
    const distanceTableDiv = document.getElementById('distanceTable');
    let tableHTML = "<table border='1'>";
    tableHTML += "<thead><tr><th>City</th>";
    for (let city of cities) {
        tableHTML += `<th><img src="img/${city}.PNG" alt="${city}" width="150" height="150"><div>${city}</di
    }
    tableHTML += "</tr></thead>";
    tableHTML += "<tbody>";
    for (let i = 0; i < cities.length; i++) {
        tableHTML += `<tr><td><img src="img/${cities[i]}.PNG" alt="${cities[i]}" width="150" height="150"><d
```

```
PS D:\Game-app>
```

Figure 39

4.5.6 Normalized DB Table Structure



```
SELECT * FROM games.identifyshortestpath;
```

Id	name	distancetoA	distancetoB	distancetoC	distancetoD	distancetoE	distancetoF	distancetoG	distancetoH	distancetoI	distancetoJ	identifyshortestp
1	oshani	27		43	24	31	15	29	41	23	34	6
2	hansi	32	46	17	45	38		40	15	27	49	6
3	pawani	45	12	20	50	36	15	48	29		41	6
4	ammi	28	37	46		35	44	23	12	50	19	6
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Table: identifyshortestpath

Columns:

Id	int AI PK
name	text
distancetoA	text
distancetoB	varchar(45)
distancetoC	varchar(45)
distancetoD	varchar(45)
distancetoE	varchar(45)
distancetoF	varchar(45)
distancetoG	varchar(45)

```
#   Time      Action                                                      Message            Duration / Fetch
44  09:01:47  SELECT * FROM games.identifyshortestpath LIMIT 0, 1000      4 row(s) returned  0.000 sec / 0.000 sec
45  09:01:50  SELECT * FROM games.knighttourproblem LIMIT 0, 1000         4 row(s) returned  0.000 sec / 0.000 sec
46  09:01:53  SELECT * FROM games.longestcommonsequence LIMIT 0, 1000     3 row(s) returned  0.000 sec / 0.000 sec
47  09:01:56  SELECT * FROM games.tictactoe LIMIT 0, 1000                 3 row(s) returned  0.000 sec / 0.000 sec
48  09:02:22  SELECT * FROM games.eightqueenspuzzle LIMIT 0, 1000         4 row(s) returned  0.000 sec / 0.000 sec
```

Query Completed

Figure 39

5 Conclusion

5.1 Summary of the Project

The project's goal is to create a game store which includes 5 gaming applications that uses Data Structures and Algorithm theory to play and solve problem in a practical setting. Randomly chosen beginning places, a user interface for player interaction, and database integration to preserve successful player responses and unsuccessful player responses are some of the main components added. The project is based on below algorithms with the below features.
Algorithm used:
Warns Dorf's Rule
Dynamic Programming
Backtracking
Minimax Algorithm
Bellman-Ford Algorithm/Dijkstra's Algorithm

To determine how to move a Knight around a chessboard, I developed the Knight's Tour Problem game. Therefore, the player must comprehend the proper order of moves.

Then, to discover the longest common sequence between two strings that were chosen at random, I developed the game Identify the Longest Common Sequence.

Then, to arrange eight queens' motions on an 8 by 8 chessboard without endangering one another, I invented the Eight Queens' Puzzle game.

Then, to find the Min-Max theory, I made a Tic-Tac-Toe game to play with the computer.

To determine the shortest route between two randomly chosen cities, I lastly developed the game Identify Shortest Path.

As mentioned before the project makes use of a variety of data structures, including dynamic arrays for computational speed, databases for player records, and arrays for representing chessboards. I also focused on providing a user-friendly and interesting gaming experience is ensured by the user interface's UX design with both pleasure and education by applying algorithmic techniques. Thorough unit testing confirms the game's dependability and usefulness. In the end I were able to successfully combines theory and practice in game production, giving players a unique challenge and an enjoyable gaming experience.

5.2 References

https://youtu.be/kRs3aTi3pzU?si=8SJ4CTmaDQ81te18

https://saturncloud.io/blog/minimax-algorithm-for-tic-tac-toe-understanding-and-overcoming-failure/#:~:text=The%20MiniMax%20algorithm%20is%20a,will%20also%20make%20optimal%20moves.

https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/#:~:text=Dijkstra's%20Algorithm%20finds%20the%20shortest,node%20and%20all%20other%20nodes.

https://favtutor.com/blogs/knight-tour-problem#:~:text=The%20Knight's%20Tour%20problem%20is,the%20edges%20of%20the%20graph.

https://www.geeksforgeeks.org/8-queen-problem/

https://www.javatpoint.com/longest-common-subsequence

5.3 Appendices

5.3.1 Timeline

Activity	**Week -01**							**Week -02**							**Week -03**							**Week -04**						
	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Planning	■	■	■																									
Gathering the requirements		■	■	■	■	■	■																					
Research stage						■	■	■	■																			
Analyzing data										■	■																	
Designing the system												■	■	■	■	■	■											
Implementing the system																		■	■	■	■	■	■					
Testing errors																								■	■			
Documentation																							■	■	■	■	■	