Programming Data Structures and Algorithms – 2

BSc(Hons)Computing 23.1P

Group course work

Group Members :

Name		**Index Number**
W.M.P.Maheshika Bandara	-	COCOBSCCOMP231P-020
Dilsha Mallawarachchi	-	COCOBSCCOMP231P-029
Hansi Weerathunga	-	COCOBSCCOMP231P-037
Manidi Kumarasinghe	-	COCOBSCCOMP231P-016
B.O.Vidurangi	-	COCOBSCCOMP231P-025

Lecturer – Mr.Samira Dayan Jayasekara

School of Computing & Engineering

National Institute of Business Management

Colombo

Table of Contents

Table of Contents 1

1 **Acknowledgement** 3

2 **Abstract** 4

3 **Introduction** 5

3.1.1 Overview of the project 5

3.1.2 Importance of data structure and algorithm in Game developments 5

3.1.3 Introduction to game menu option 5

3.1.4 Chapter Summary 6

4 **Implementation** 7

4.1 **knight's tour Problem** 7

4.1.1 Program Logic used to solve the problem 7

4.1.2 User Interfaces 8

4.1.3 Validations and Exception Handling in this application 9

4.1.4 Unit Testing 10

4.1.5 Indicate the Data Structures used with its purpose 11

4.1.6 Normalized DB Table Structure 11

4.2 Identify longest common sequence 12

4.2.1 Program Logic used to solve the problem 12

4.2.2 User Interfaces 12

4.2.3 Validations and Exception Handling in this application 14

4.2.4 Unit Testing 15

4.2.5 Indicate the Data Structures used with its purpose 16

4.2.6 Normalized DB Table Structure 17

4.3 **Eight queens' puzzle** 18

4.3.1 Program Logic used to solve the problem 18

4.3.2 User Interfaces 19

4.3.3 Validations and Exception Handling in this application 20

4.3.4 Unit Testing 21

4.3.5 Normalized DB Table Structure 22

4.4 **Tic-Tac-Toe** 23

4.4.1 Program Logic used to solve the problem 23
4.4.2 User Interfaces 24
4.4.3 Validations and Exception Handling in this application 25
4.4.4 Unit Testing 27
4.4.5 Indicate the Data Structures used with its purpose 28
4.4.6 Normalized DB Table Structure 28
4.5 **Identify Shortest Path** 29
4.5.1 Program Logic used to solve the problem 29
4.5.2 User Interfaces 30
4.5.3 Validations and Exception Handling in this application 31
4.5.4 Unit Testing 31
4.5.5 Indicate the Data Structures used with its purpose 33
4.5.6 Normalized DB Table Structure 34
5 **Conclusion** 35
5.1 Summary of the Project 35
5.2 References 36
5.3 Appendices 37
5.3.1 Timeline 37
5.3.2 Contribution 38

1 Acknowledgement

First of all, we 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.

We would like to express our gratitude for Mr. Samira Dayan Jayasekara for the source of knowledge, support and guidance given throughout our lectures. also thanking for all our teams for their hard work.

Finally, We would like to Thank our 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, we are assigned to do a project using our theoretical knowledge to implement the game application. There, we created five real-world games to represent our theoretical knowledge of data structures and algorithm for real-world scenarios. We 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. Our 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

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

We 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 we created the Identify the longest common sequence game to find the longest common sequence between randomly selected two strings.

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

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

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

So our 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, our team used JavaScript to develop the backend of this game application. Then, to design the front pages, we used HTML and CSS languages. Then we used MYSQL to develop the database side. Also, from that, we could successfully develop our theoretical knowledge through practical implementation.

3.1.4 Chapter Summary

We provided the five game options that best represent our knowledge of data structures and algorithms. These games not only entertain players but also provide users and improve their problem-solving abilities. We also pointed out the value of these concepts in game design, focusing on how important they are. Our 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:

Figure 01

4.1.2 User Interfaces

Figure 02

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 check the output of the game by checking the chessboard status to decide the player has won or not.

Code:

Figure 04

4.1.4 Unit Testing

Testing code:

```
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

Figure 07

4.1.6 Normalized DB Table Structure

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

Figure 09

Figure 10

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

Figure 12

Figure 13

4.2.4 Unit Testing

Testing code:

Figure 14

Results:

```
longest seuqence

expected results
seuqence 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

seuqence 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]);
```

Figure 16

4.2.6 Normalized DB Table Structure

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:

Figure 18

4.3.2 User Interfaces

Figure

Figure 19

4.3.3 Validations and Exception Handling in this application

Figure 20

Figure 21

4.3.4 Unit Testing

Testing code:

Figure 22

Results:

Figure 23

4.3.5 Normalized DB Table Structure

Figure 24

4.4 Tic-Tac-Toe

4.4.1 Program Logic used to solve the problem

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

Firstly, we have to create function as a make move. And then we 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. We would return zero.

Here is the logic used to the solve the problem:

Figure 24

4.4.2 User Interfaces

Figure 25

Figure 26

Figure 27

4.4.3 Validations and Exception Handling in this application

1. Input Validation:

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

2. Winning and Draw Validation:

- We added another function call `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 we 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.

Figure 28

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 we have used the 9 elements array as a structure. We have used tree data

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

There we have 2D array which known as graph.it represents the distance of two cities .and there is a list which consists 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

Figure 33

4.5 Identify Shortest Path

4.5.1 Program Logic used to solve the problem

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

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

There we have 2D array which known as graph.it represents the distance of two cities .and there is a list which consists 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.

Here is the logic used to the solve the problem:

Figure 34

4.5.2 User Interfaces

Figure 35

4.5.3 Validations and Exception Handling in this application

Figure 36

4.5.4 Unit Testing

Testing code:

Figure 37

Results:

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.

Figure 39

4.5.6 Normalized DB Table Structure

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:
Warnsdorf's Rule
Dynamic Programming
Backtracking
Minimax Algorithm
Bellman-Ford Algorithm/Dijkstra's Algorithm

To determine how to move a Knight around a chessboard, we 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, we developed the game Identify the Longest Common Sequence.

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

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

To determine the shortest route between two randomly chosen cities, we 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. We 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 we 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																												

5.3.2 Contribution

COCOBSCCOMP231P-020	Conducted the unit testing and contribute to the report making and finalizing the report
COCOBSCCOMP231P-029	Contribute with Identify longest common sequence game implementation and helping for report making process
COCOBSCCOMP231P-037	Contribute in knight's tour Problem implementation and Eight queens' puzzle, implement data base structure and contribute with validations and exceptions handling
COCOBSCCOMP231P-016	Contribute with Tic-Tac-Toe game implementation and report making process
COCOBSCCOMP231P-025	Did the Identify Shortest Path game implementation and took the responsibilities for creating game menu and main page development.