

PDSA COURSE WORK

GROUP No 09

A project report submitted by

Hadaragama H.P

(cobsccomp222p-026)

Perera I.A.C.D

(cobsccomp222p-017)

Nuwanthi B.G.A

(cobsccomp222p-015)

National Institute of Business Management

BSc(Hons) Computing

2023

Table of Contents

Chapter 1 : Eight Queen's Puzzle----- 4

1.1 Program Logic used to Identify maximum number of solutions for Eight queen's puzzle.	4
1.2 UI screenshot allowing game players to provide answers.	5
1.3 UI screenshots when game players to provide correct answer & incorrect answers.	6
1.4 .Code Segment: When a game player correctly identifies an answer, save that person's name along with the correct response in the database.	9
1.5 .Code Segment: If another game player provides the same right response, indicate that the solution has already been recognized.	10
1.6 Code Segment: When all the solutions have been identified by game players, the system should clear the flag that indicate solution has already been recognized.	11
1.7 Indicate the Data Structures used with its purpose.	12
1.8 Validations and Exception Handling in this application.	12
1.9 Screenshot of the Normalized DB Table Structure used for this Game Option.	13

Chapter 2 : Encode /Decode using Huffman.-----14

2.1 .Program Logic used to implement Encode /Decode using Huffman Coding Algorithm.	14
2.2 UI screenshot allowing game players to provide answers.	15
2.3 .Code Segment: for Option 1	17
2.4 Code Segment: for Option 2	18
2.5 UI screenshots when game players to provide correct answer & incorrect answers.	19
2.6 Indicate the Data Structures used with its purpose.	23
2.7 .Validations and Exception Handling in this application.	23

Chapter 3 : Tic-Tac-Toe -----25

3.1 .Program Logic used to implement Tic-Tac-Toe.....	25
3.2 .UI screenshot allowing game players to provide answers.	26
3.3 Code Segment: Determining the optimal Tic-Tac-Toe move for a computer player using the Minimax Algorithm in Game Theory.	27
3.4 Screenshots of the user interface when players win, lose, or draw a game.	28

3.5 Indicate the Data Structures used with its purpose.	30
3.6 Specify using Code Segments or Screenshots the Validations and Exception Handling in this application.....	30

Chapter 4 : Identify Shortest Path-----31

4.1 Program Logic used to implement Identify Shortest Path	31
4.2 Code Segment used to set random distance.	31
4.3 UI screenshot allowing game players to provide answers.	32
4.4 Code Segment: find the shortest path and distance for other cities from the system's randomly selected city.....	33
4.5 Code Segment used to save person's name along with the correct answer.	34
4.6 Code Segment used to save distance between cities when they correctly identify an answer....	35
4.7 UI screenshots when game players to provide correct answer & incorrect answers.	35
4.8 .Indicate the Data Structures used with its purpose.	37
4.9 Specify using Code Segments or Screenshots the Validations and Exception Handling in this game option.....	37
4.10 Screenshot of the Normalized DB Table Structure used for this Game Option.....	38

Chapter 5 : Identify minimum connectors. -----39

5.1 Program Logic used to Identify minimum connectors.	39
5.2 Code Segment used to set random distance.	39
5.3 UI screenshot allowing game players to provide answers.	40
5.4 UI screenshots when game players to provide correct answer & incorrect answers.	41
5.5 .Code Segment used to save person's name along with the correct answer.	43
5.6 Code Segment used to save distance between cities when they correctly identify an answer....	43
5.7 Indicate the Data Structures used with its purpose.	44
5.8 Specify using Code Segments or Screenshots the Validations and Exception Handling in this game option.....	44
5.9 .Screenshot of the Normalized DB Table Structure used for this Game Option.....	45

Chapter 1 : Eight Queen's Puzzle

1.1 Program Logic used to Identify maximum number of solutions for Eight queen's puzzle.

In this case, the Eight Queens puzzle was resolved using the backtracking approach. For the queens, we have developed a system that recursively try several positions and reverse course when a disagreement arises. This method enables the investigation of all potential problem solutions.

The backtracking technique reduces the search space by removing search tree branches that are certain to result in incorrect results. It efficiently discovers all feasible answers to the challenge by methodically examining the remaining branches.

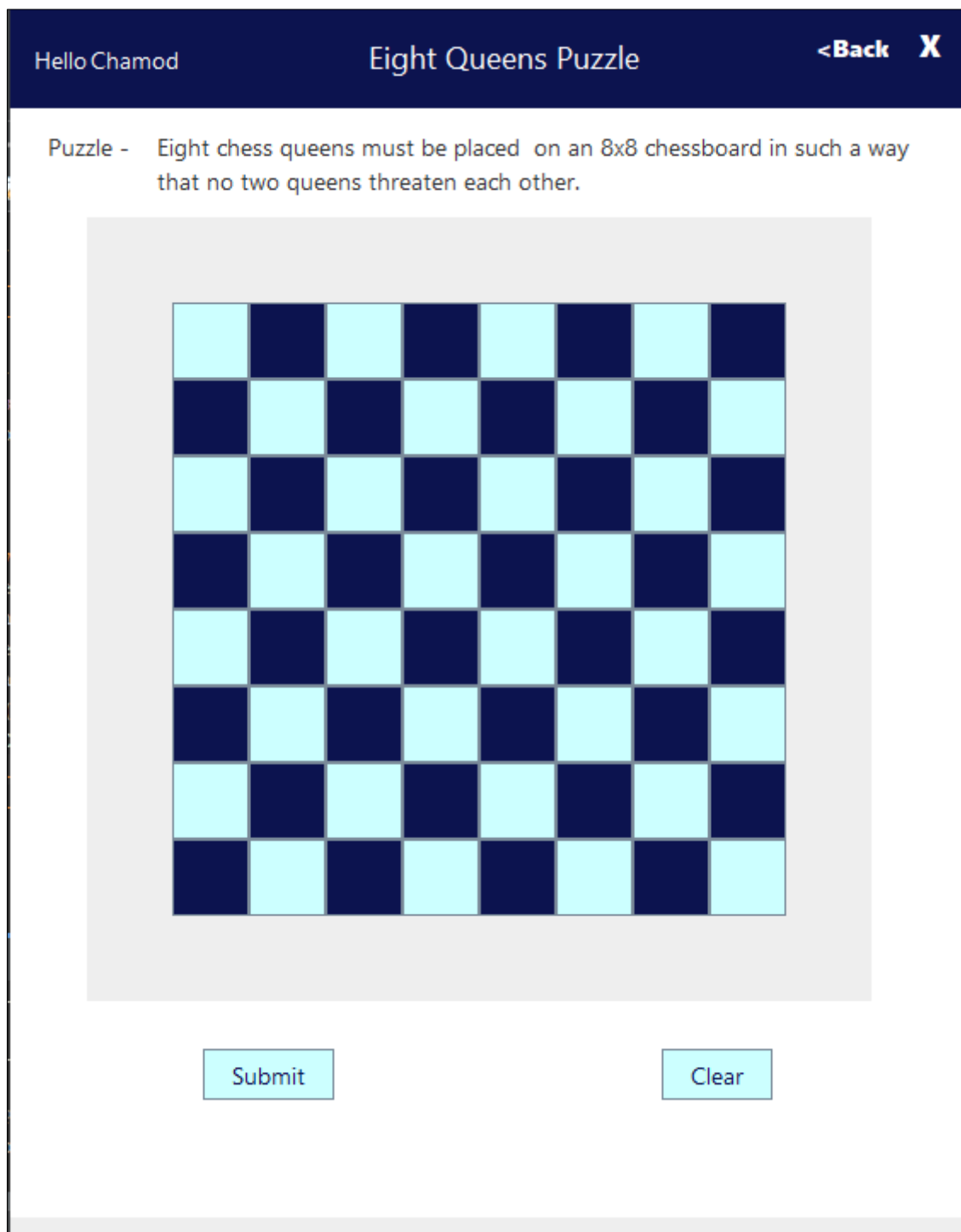
It should be noted that methods like constraint propagation and intelligent variable ordering can be used to further optimize the backtracking process. These methods can enhance the algorithm's performance by lowering the number of recursive calls.

Backtracking technique, as previously indicated, was used to determine how to keep the eight queens in a way that prevents them from endangering one another. We then took those viable solutions and converted them into a 2D array before saving them as a 2D array List. The answer given by the game player is likewise saved as a 2D array with the aid of the user interface we developed, and we first check to see if it is in the 2D array list above. It is a bad response if it is not included in that 2D array List. The player is then informed via a suitable message that is displayed. If the answer is in a 2D array list, we then search the database to see if it has already been given. If this is the case, I messaged the player to let them know. If an answer is given without assistance from anybody else, it is accepted as the correct response, and both the answer and the player's name are entered into the database.

1.2 UI screenshot allowing game players to provide answers.

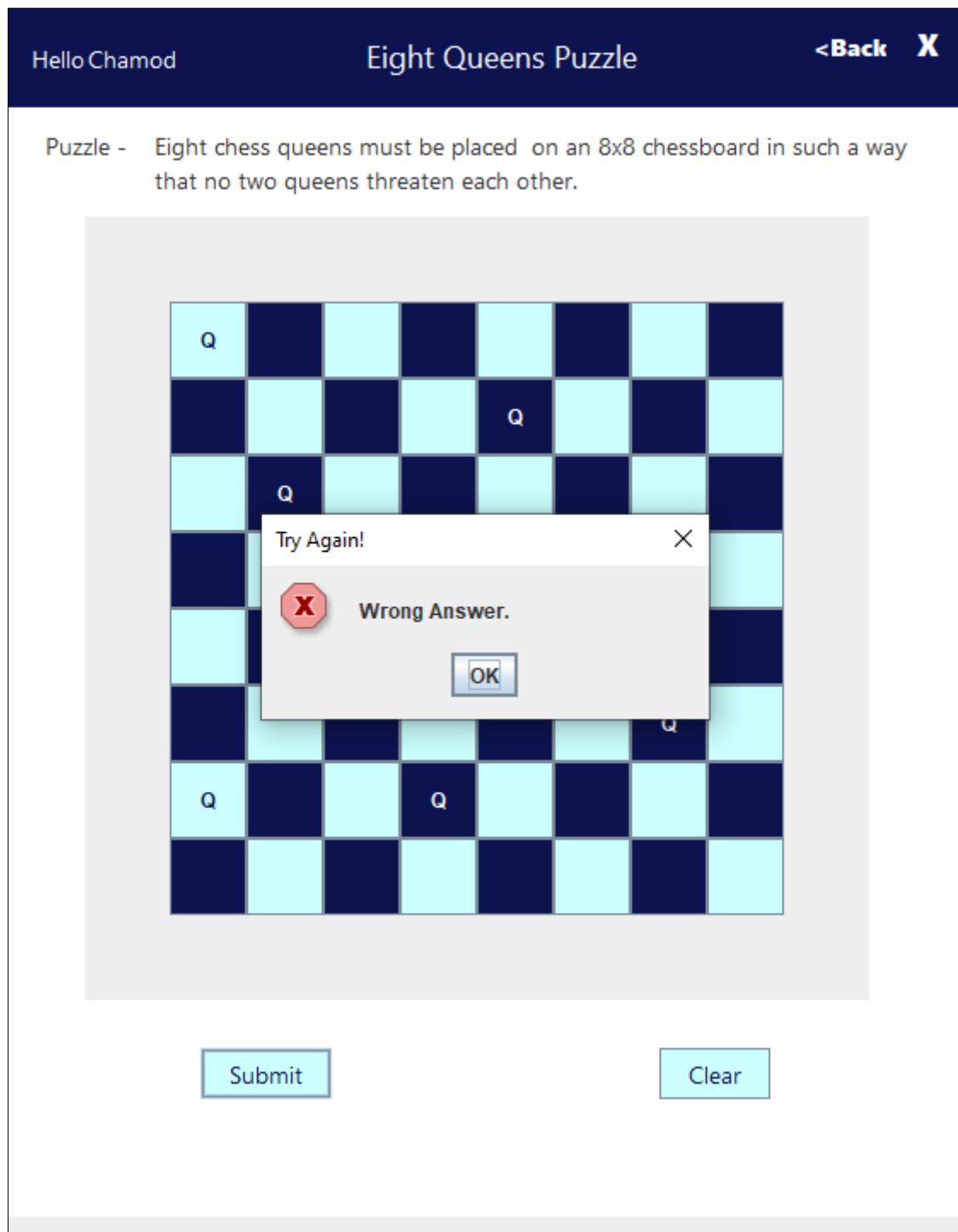
When user come to the queens' puzzle from the main menu of the game, the user interface appears as below. Here we have used a 560x700 JFrame, and basically, Two Jpanels have been added to it. The second Jpanel is a title bar. It has a button for the game player's name, the game's title, and main menu, and a button to close the game.

In the second Jpanel, we created a chess bord using Jbuttons. Also, added Two Jbutton to submit user answers and clear the user's answer.

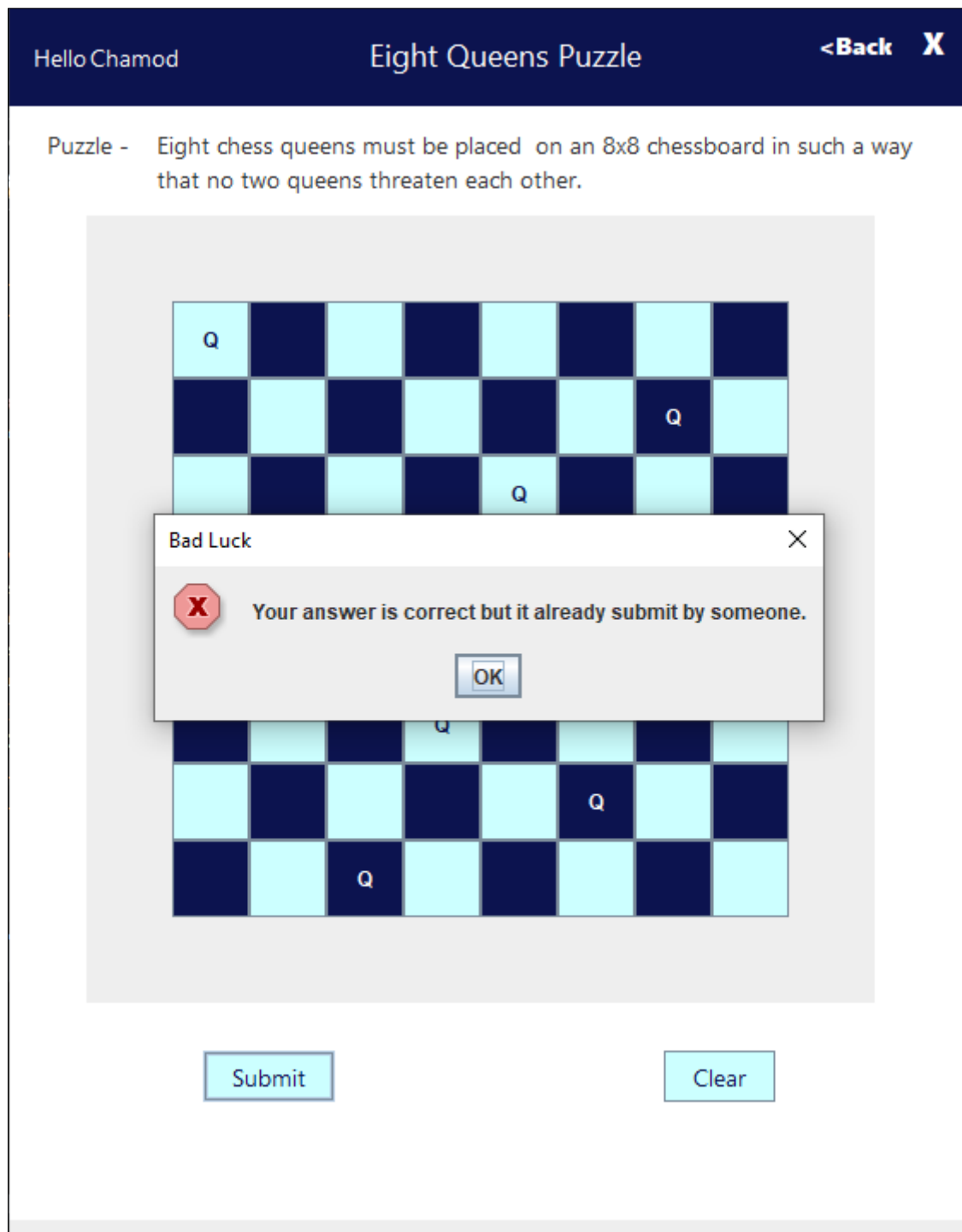


1.3 UI screenshots when game players to provide correct answer & incorrect answers.

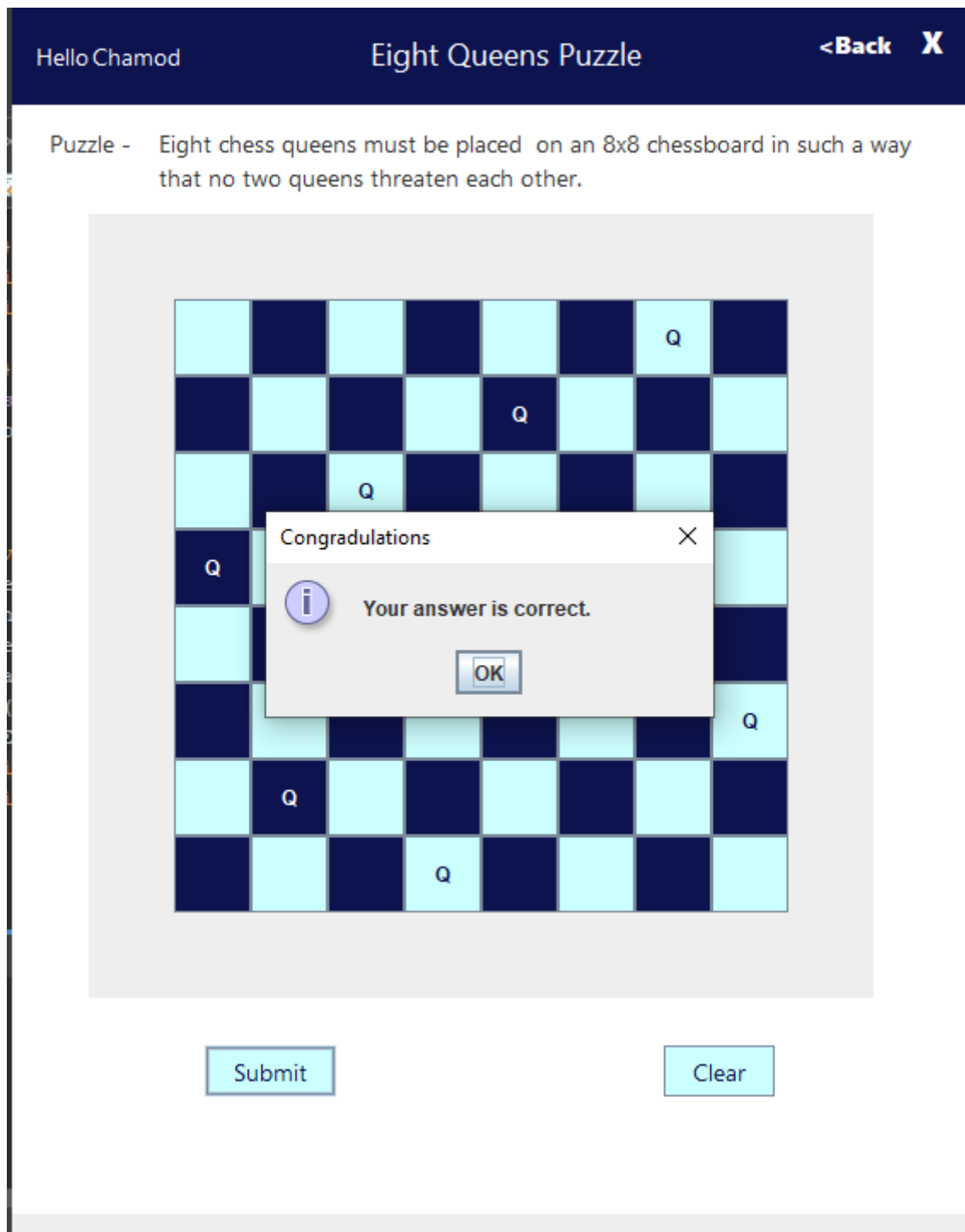
Scenario 1- When user insert a wrong answer.



Scenario 2- When user insert the correct answer, but it's already added by someone else.



Scenario 3- When user insert the correct answer for the first time.



1.4 .Code Segment: When a game player correctly identifies an answer, save that person's name along with the correct response in the database.

When game player provides an answer, first converted the answer into a 2D array and checked to see if it has been saved in the 2D ArrayList with the correct answers. A valid final answer is not obtained, and if it is not in the database, it is saved as a valid answer. Here, the game player's id number, player's name and answer are saved in the database.

- Code For, check the game player's answer, with answers 2D array List.

First, the gameplayer's answer gets into the 2D array by EightQueens.java JFrame in the View package and passes it to EightQueensLogics.java class which is in the Controller package to Compare the gameplayer's answer with Solution 2D array List.

```
public void checkSolution(int answerArray[][]) {  
  
    boolean found = false;  
    for (int[][] arr : solutions) {  
        if (Arrays.deepEquals(arr, answerArray)) {  
            found = true;  
            break;  
        }  
    }  
  
    // Print the result  
    if (found) {  
        EightQueensSql eqsql = new EightQueensSql(userid, answer, userName);  
        eqsql.checkCount();  
        eqsql.checkAnswer(answer);  
    } else {  
        JOptionPane.showMessageDialog(null, "Wrong Answer.", "Try Again!", JOptionPane.ERROR_MESSAGE);  
    }  
}
```

Here, the game player's answer passes to EightQueensSql. java class which is in the Model package by the EightQueensLogics.java class to check the game player's answer, In the database for check the answer submitted by another player.

- Save Game Players Answer.

```
public void inputAnswer(){  
    try{ String query = "INSERT INTO 'eightqueensanswers' ('uid','uname', 'answer') VALUES (?, ?, ?)";  
        pst = con.prepareStatement(query);  
        pst.setInt(1,userid);  
        pst.setString(2,uname);  
        pst.setString(3,answer);  
        pst.executeUpdate();  
        JOptionPane.showMessageDialog(null, "Your answer "  
            + "is correct.", "Congradulations", JOptionPane.INFORMATION_MESSAGE);  
    }catch(Exception ex){  
        JOptionPane.showMessageDialog(null, ex);  
    }  
}
```

1.5 .Code Segment: If another game player provides the same right response, indicate that the solution has already been recognized.

As mentioned in 3.1.4. If the gameplayer's answer is valid, then check whether the answer in the database. This process can be divided into two parts as follows.

- Code For, check the game player's answer, with answers 2D array List.
First, the gameplayer's answer gets into the 2D array by EightQueens.java JFrame in the View package and passes it to EightQueensLogics.java class which is in the Controller package to Compare the gameplayer's answer with Solution 2D array List.

```
public void checkSolution(int answerArray[][]) {  
  
    boolean found = false;  
    for (int[][] arr : solutions) {  
        if (Arrays.deepEquals(arr, answerArray)) {  
            found = true;  
            break;  
        }  
    }  
  
    // Print the result  
    if (found) {  
        EightQueensSql eqsql = new EightQueensSql(userid, answer, userName);  
        eqsql.checkCount();  
        eqsql.checkAnswer(answer);  
    } else {  
        JOptionPane.showMessageDialog(null, "Wrong Answer.", "Try Again!", JOptionPane.ERROR_MESSAGE);  
    }  
}
```

- Code For, check the game player's answer, In the database for check the answer submit by another player.

Here, the game player's answer passes to EightQueensSql. java class which is in the Model package by the EightQueensLogics.java class to check the game player's answer, In the database to check the answer submitted by another player. If it is, the game will show a proper message to inform it to gameplayer.

```

public void checkAnswer(String answer){
    this.answer=answer;
    try{String query = "select * from eightqueensanswers where answer = ?";
        pst=con.prepareStatement(query);
        pst.setString(1,answer);
        rs=pst.executeQuery();
        if(rs.next()){
            JOptionPane.showMessageDialog(null, "Your answer is correct but it already submit"
                + " by someone.", "Bad Luck", JOptionPane.ERROR_MESSAGE);
        }else{
            inputAnswer();
        }
    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(null, "Sql connection Error.", "Error", JOptionPane.ERROR_MESSAGE);
    }
}

```

```

public void inputAnswer(){
    try{ String query = "INSERT INTO `eightqueensanswers` (`uid`,`uname`, `answer`) VALUES (?, ?, ?) ";
        pst = con.prepareStatement(query);
        pst.setInt(1,userid);
        pst.setString(2,uname);
        pst.setString(3,answer);
        pst.executeUpdate();
        JOptionPane.showMessageDialog(null, "Your answer "
            + "is correct.", "Congradulations", JOptionPane.INFORMATION_MESSAGE);
    }catch(Exception ex){
        JOptionPane.showMessageDialog(null, ex);
    }
}

```

1.6 Code Segment: When all the solutions have been identified by game players, the system should clear the flag that indicate solution has already been recognized.

The maximum number of solutions for the 8-queen puzzle is 92. The game system checks the records count in the database after each new answer is submitted to the database's "queenspuzzleanswer" table. When the count is equal to 92, the data is deleted from the table and reset. This process is carried out by EightQueensSql. java class which is in the Model package.

```

public void restAnswers(){
    try{ String query = "DELETE FROM `eightqueensanswers`";
        pst = con.prepareStatement(query);
        pst.executeUpdate();
        System.out.println("Reseted Successfully");
    }catch(SQLException | HeadlessException ex){
        JOptionPane.showMessageDialog(null, ex);
    }
}

```

1.7 Indicate the Data Structures used with its purpose.

- **2D Arrays:**

The puzzle board is represented by a 2D array (board) with a size of 8 by 8. Each cell of the array represents a position on the board, and the value 1 is used to indicate the presence of a queen at that position.

- **List:**

The code uses a `List<int[][]>` (implemented as an `ArrayList`) to store the solutions to the Eight Queens puzzle. The list allows for dynamically adding solutions as they are found during the backtracking process.

1.8 Validations and Exception Handling in this application.

In this game, the game player provides answers by clicking the buttons on the chess board. Here, the game player must place eight queens on the chess board. There is a validation method so that the player cannot be placed more than eight or less than eight queens on the chess board. Also, exceptions have been used when saving a definite answer to the database. Also, exceptions have been used for the 2D arrays and 2D ArrayList used here. The codes for all of these are given below.

```
try{String query = "SELECT COUNT(*) FROM eightqueensanswers";
    pst=con.prepareStatement(query);
    rs=pst.executeQuery();
    if(rs.next()){
        count = rs.getInt(1);
        if(count==92){
            restAnswers();
        }
    }
}
catch (SQLException ex) {
    JOptionPane.showMessageDialog(null, "Sql connection Error.", "Error", JOptionPane.ERROR_MESSAGE);
}
```

```
private void checkQueens() {
    int counter = 0;
    if (btn_cell1.getText().equals("Q")) {
        counter = counter+1;
    } if (btn_cell2.getText().equals("Q")) {
        counter = counter+1;
    } if (btn_cell3.getText().equals("Q")) {
        counter = counter+1;
    } if (btn_cell4.getText().equals("Q")) {
        counter = counter+1;
    } if (btn_cell5.getText().equals("Q")) {
        counter = counter+1;
    } if (btn_cell6.getText().equals("Q")) {
        counter = counter+1;
    } if (btn_cell7.getText().equals("Q")) {
        counter = counter+1;
    } if (btn_cell8.getText().equals("Q")) {
        counter = counter+1;
    }
}
```


Chapter 2 : Encode /Decode using Huffman.

2.1 .Program Logic used to implement Encode /Decode using Huffman Coding Algorithm.

The idea behind Huffman coding is to produce a variable-length prefix code in such a way that the characters that occur the most frequently are given shorter codes and the characters that occur the least frequently are given longer codes. Because the most frequent characters only need a small number of bits to represent them, this enables efficient encoding and decoding of messages.

According to the question, first we measured the frequency of individual character in the input. Then we map those values inside a frequency map with the corresponding frequency. Then we created an instance of priority queue and added those values. We used a min heap priority queue with this question. Because of that we were able to store node which having the lowest frequency at the front of the queue. Then we removed the top two value which contain the lowest frequencies from the priority queue. After that we created a new internal node and set the value. For the value we got the sum of the frequency of removed two nodes. Then to create a tree we added removed two nodes as the left and right child of the newly created node. Then we added that newly created node into the priority queue again. At the end of the algorithm, last remaining node in queue will be the root node of the Huffman tree.

Encoding – To do the encoding, first we looped through the Huffman tree form the root to last leaf node and assigned “0” to the left branch and “1” to the right branch. Then at each node assigned the binary code that represent the path from root node to that corresponding node. At the end of encoding, we can get the binary codes that can replace all the characters. Finally, we merge them all together and get the encoded message.

Decoding- To do the decoding, first we started form the root of the Huffman tree and read the first bit from the encoded message. If the first bit is 0 we select the left child of the current node. otherwise, we need to move to the right child of the current node of the tree. We need to repeat these until we reach to the leaf node. Then append the character represented by that node to the decoded message. Then we read the next bit from the encoded message and to the above step. After the last bit the decoded message will be fully reconstructed.

2.2 UI screenshot allowing game players to provide answers.

Encoding option –

Hello Chamod

Encode Puzzle

<Back X

Puzzle - Generate a word and encode it to binary.

iHEDL

Generate String

SubmitClear

Decoding option

Hello Chamod

Decode Puzzle

<Back X

Puzzle - Generate a Encoded word and Decode it.

000111111010

Generate Encoded Word

SubmitClear

2.3 .Code Segment: for Option 1

```
public static HashMap<Character, String> encode(String input) {
    HashMap<Character, Integer> frequencyMap = new HashMap<>();
    for (char c : input.toCharArray()) {
        frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
    }
    PriorityQueue<HuffmanCode> priorityQueue = new PriorityQueue<>();
    for (char c : frequencyMap.keySet()) {
        priorityQueue.offer(new HuffmanCode(c, frequencyMap.get(c)));
    }
    while (priorityQueue.size() > 1) {
        HuffmanCode leftChild = priorityQueue.poll();
        HuffmanCode rightChild = priorityQueue.poll();

        HuffmanCode parent = new HuffmanCode('\0', leftChild.frequency + rightChild.frequency);
        parent.left = leftChild;
        parent.right = rightChild;

        priorityQueue.offer(parent);
    }
    HuffmanCode root = priorityQueue.poll();
    HashMap<Character, String> encodingMap = new HashMap<>();
    buildEncodingMap(root, "", encodingMap);

    return encodingMap;
}
```

```
private static void buildEncodingMap(HuffmanCode node, String code, HashMap<Character, String> encodingMap) {
    if (node.isLeafNode()) {
        encodingMap.put(node.character, code);
        return;
    }

    buildEncodingMap(node.left, code + "0", encodingMap);
    buildEncodingMap(node.right, code + "1", encodingMap);
}
```

```
int uid;
String uname;
Random random = new Random();
int length = 10;

public EncodeHuffman(int uid, String uname) {
    this.uname = uname;
    this.uid = uid;
    System.out.println("" + uid + uname);
    initComponents();
    lbl_userName.setText(uname);
}
```

```

private String generateRandomString(int length) {
    // Define the characters that can be used in the random string
    String characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ123456780";

    StringBuilder sb = new StringBuilder(length);
    for (int i = 0; i < length; i++) {
        // Generate a random index within the range of available characters
        int randomIndex = random.nextInt(characters.length());

        // Append the character at the random index to the string
        sb.append(characters.charAt(randomIndex));
    }

    return sb.toString();
}

```

2.4 Code Segment: for Option 2

```

public static String decode(String encodedMessage, HashMap<Character, String> encodingMap) {

    StringBuilder decodedMessage = new StringBuilder();
    StringBuilder currentCode = new StringBuilder();

    for (char c : encodedMessage.toCharArray()) {
        currentCode.append(c);

        for (char character : encodingMap.keySet()) {
            if (encodingMap.get(character).equals(currentCode.toString())) {
                decodedMessage.append(character);
                currentCode.setLength(0);
                break;
            }
        }
    }
    return decodedMessage.toString();
}

```

```

private void btn_generateWordActionPerformed(java.awt.event.ActionEvent evt) {
    String input = generateRandomString(5);

    HashMap<Character, String> encodingMap = HuffmanEncoder.encode(input);

    StringBuilder encodedMessage = new StringBuilder();

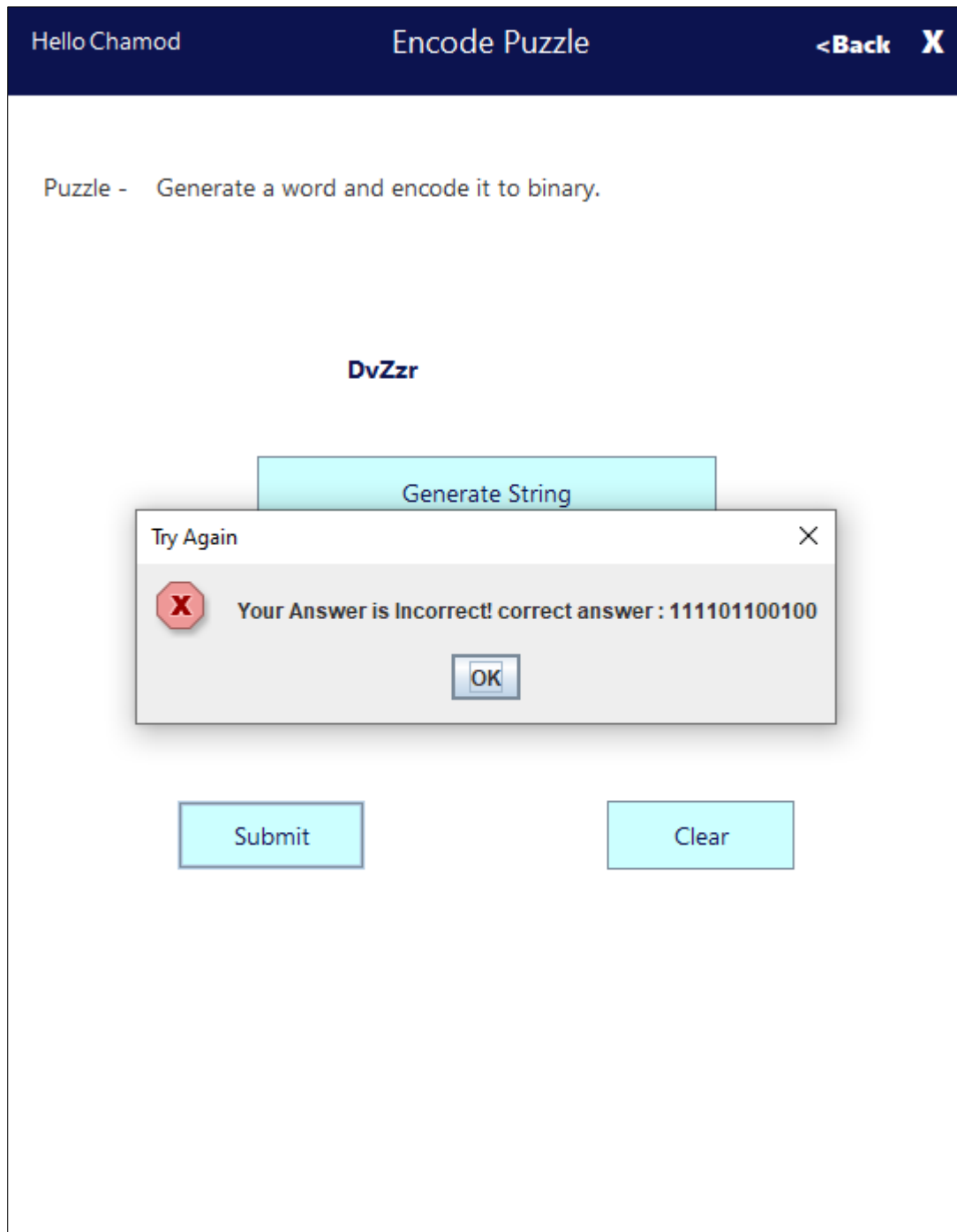
    for (char c : input.toCharArray()) {
        encodedMessage.append(encodingMap.get(c));
    }
    lbl_string.setText(encodedMessage.toString());

    decodedMessage = HuffmanDecoder.decode(encodedMessage.toString(), encodingMap);
}

```

2.5 UI screenshots when game players to provide correct answer & incorrect answers.

Option 1 - Scenario 1 – When user insert the wrong answer.



Option 1 - Scenario 2 – When user insert the correct answer.

Hello Chamod

Encode Puzzle

<Back X

Puzzle - Generate a word and encode it to binary.

DvZzr

Generate String

Congragulations! X

i

Your Answer is Correct!

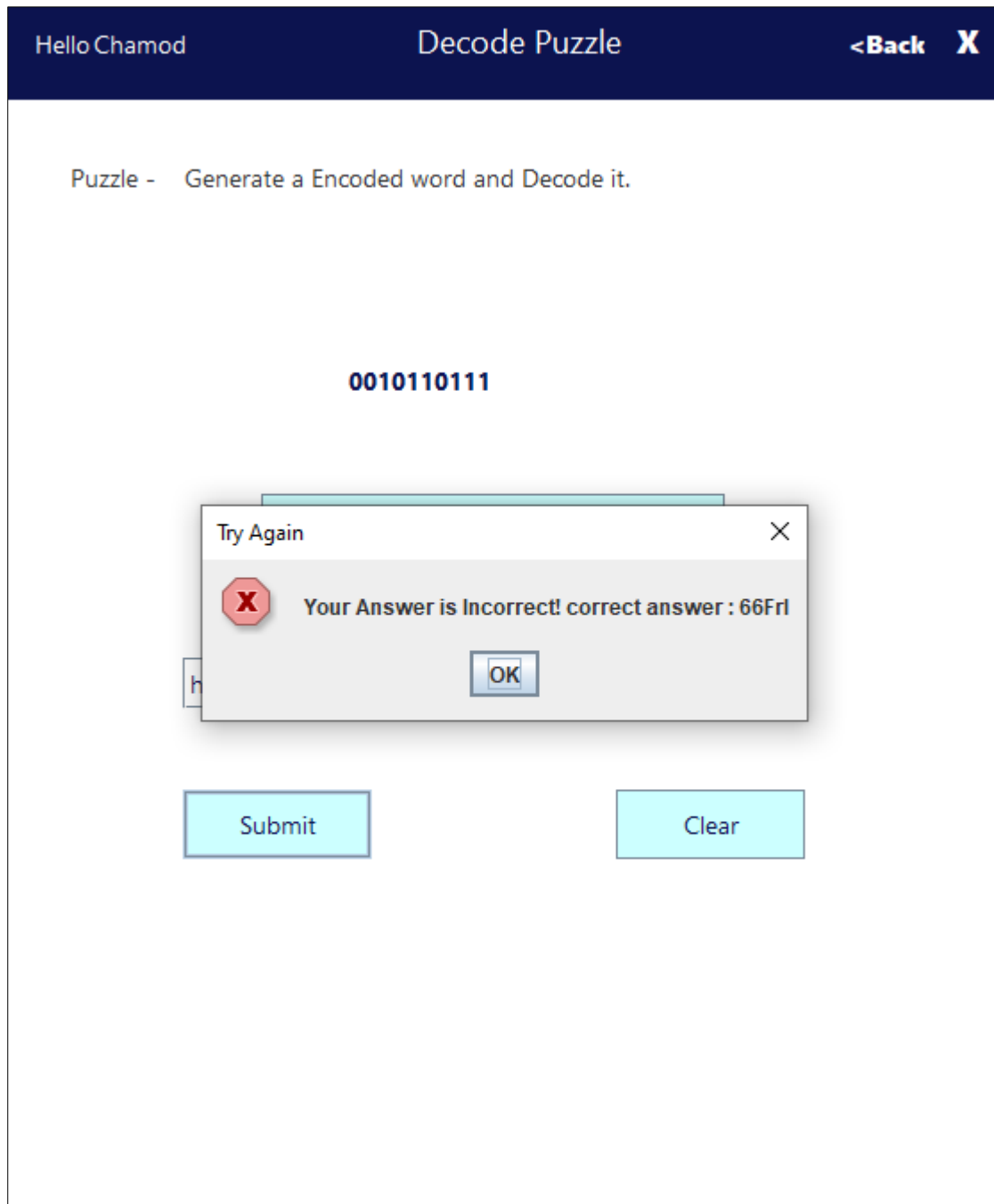
OK

11110

Submit

Clear

Option 2 - Scenario 1 – When user insert the wrong answer.



Option 2 - Scenario 2 – When user insert the correct answer.

Hello Chamod


Decode Puzzle

<Back X

Puzzle - Generate a Encoded word and Decode it.

0010110111

Congragulations!

 Your Answer is Correct!

OK

66FrI

Submit

Clear

2.6 Indicate the Data Structures used with its purpose.

- Priority Queue

An abstract data type called a priority queue resembles a queue or a stack but has an additional priority assigned to each piece. The entries in a priority queue are given priorities, and the higher-priority elements are dequeued before the lower-priority elements.

- Map

A map data structure, sometimes referred to as a dictionary, associative array, or hash map, is a type of database that contains key-value pairs in which each key corresponds to a single value.

2.7 .Validations and Exception Handling in this application.

```
try{
    for (char c : encodedMessage.toCharArray()) {
        currentCode.append(c);

        for (char character : encodingMap.keySet()) {
            if (encodingMap.get(character).equals(currentCode.toString())) {
                decodedMessage.append(character);
                currentCode.setLength(0);
                break;
            }
        }
    }
} catch (Exception e) {
    System.out.println(e);
}

return decodedMessage.toString();
```

```
if (input.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please Generate"
        + " a Encoded word First !", "Error", JOptionPane.ERROR_MESSAGE);
} else {
    if (answer.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Please"
            + " enter answer !", "Error", JOptionPane.ERROR_MESSAGE);
    } else {
        if (answer.equals(decodedMessage.toString())) {
            JOptionPane.showMessageDialog(this, "Your"
                + " Answer is Correct!", "Congragulations!", JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(this, "Your"
                + " Answer is Incorrect! correct answer"
                + " : " + decodedMessage.toString(), "Try Again"
                + " !", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

```

private void txt_userAnswerKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    char c = evt.getKeyChar();
    if (evt.getKeyChar() >= '0' && evt.getKeyChar() <= '1') {
        txt_userAnswer.setEditable(true);
    } else {
        if (evt.getExtendedKeyCode() == KeyEvent.VK_BACK_SPACE) {
            txt_userAnswer.setEditable(true);
        } else {
            JOptionPane.showMessageDialog(this, "Please enter "
                + "Binary Value !", "Information", JOptionPane.ERROR_MESSAGE);

            txt_userAnswer.setText("");
        }
    }
}

```


Chapter 3 : Tic-Tac-Toe

3.1 .Program Logic used to implement Tic-Tac-Toe

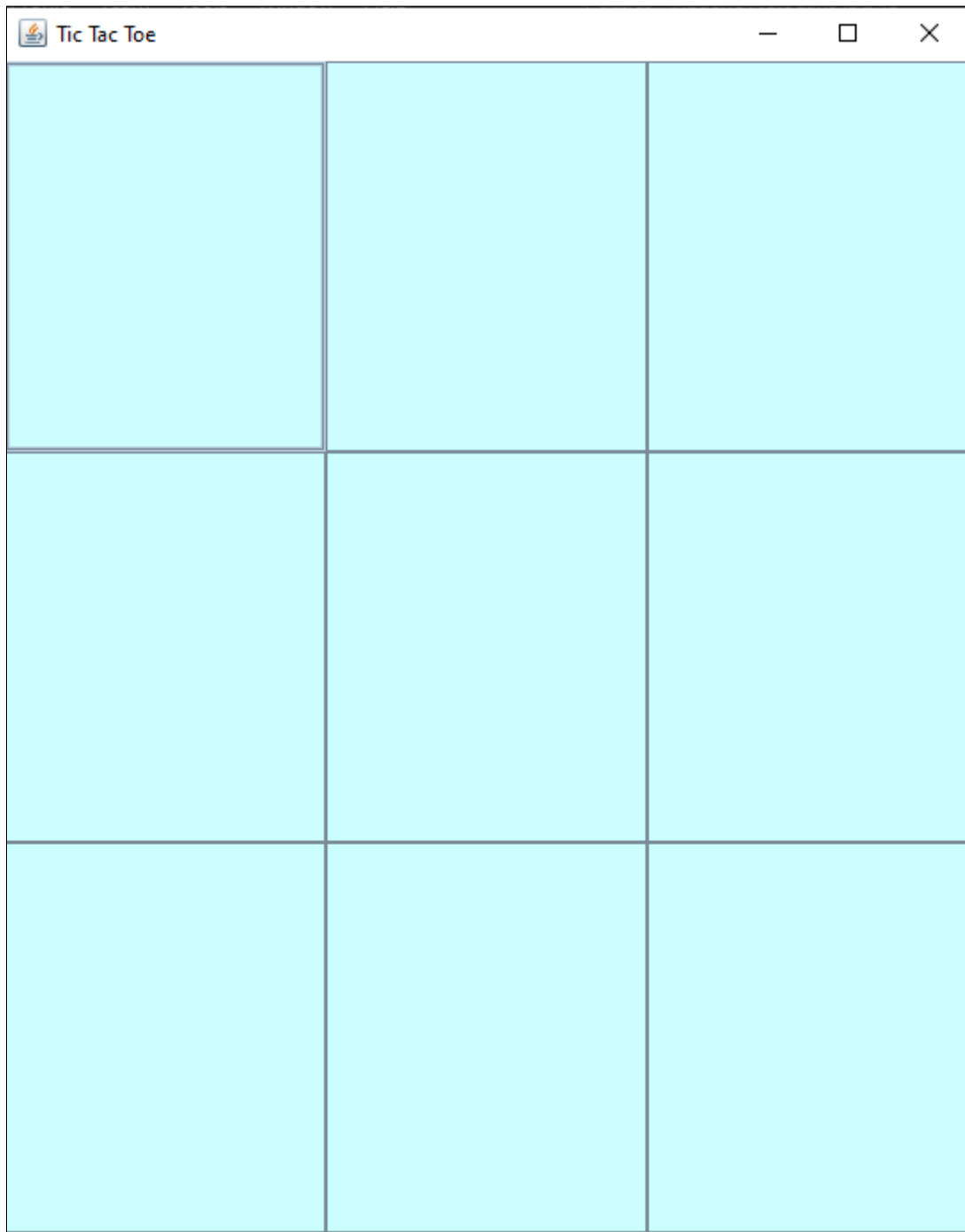
There is a 2D Array of characters that serves as the game board's representation. The answers for each cell on the board can be empty, the player's answer ('X'), or the game system's answer ('O').

The game board is initialized by setting all cells to their empty states. then confirms that the given cell is empty before determining if a move is valid.

Also, we created methods for performing following features.

- verifies whether the chosen player has won the match. It looks for lucrative alignments in the rows, columns, and diagonals.
- To determines if either player has won or the board is full to see if the game is over.
- To move the corresponding cell to the player's symbol for the selected player on the board.
- A method uses the minimax algorithm to decide what the AI player's best move should be. The best score is returned after recursively evaluating every move that might be made. The maximizing player (AI) and the minimizing player (user) are both considered by the method.
- A method calculates a score by assessing the board's current condition. It returns 1 if the AI player has triumphed. It returns -1 if the user has triumphed. If a draw occurs, it gives back 0.

3.2 .UI screenshot allowing game players to provide answers.



3.3 Code Segment: Determining the optimal Tic-Tac-Toe move for a computer player using the Minimax Algorithm in Game Theory.

```
private int minimax(int depth, boolean isMaximizingPlayer) {
    int score = evaluateBoard();

    if (score == 1 || score == -1) {
        return score;
    }

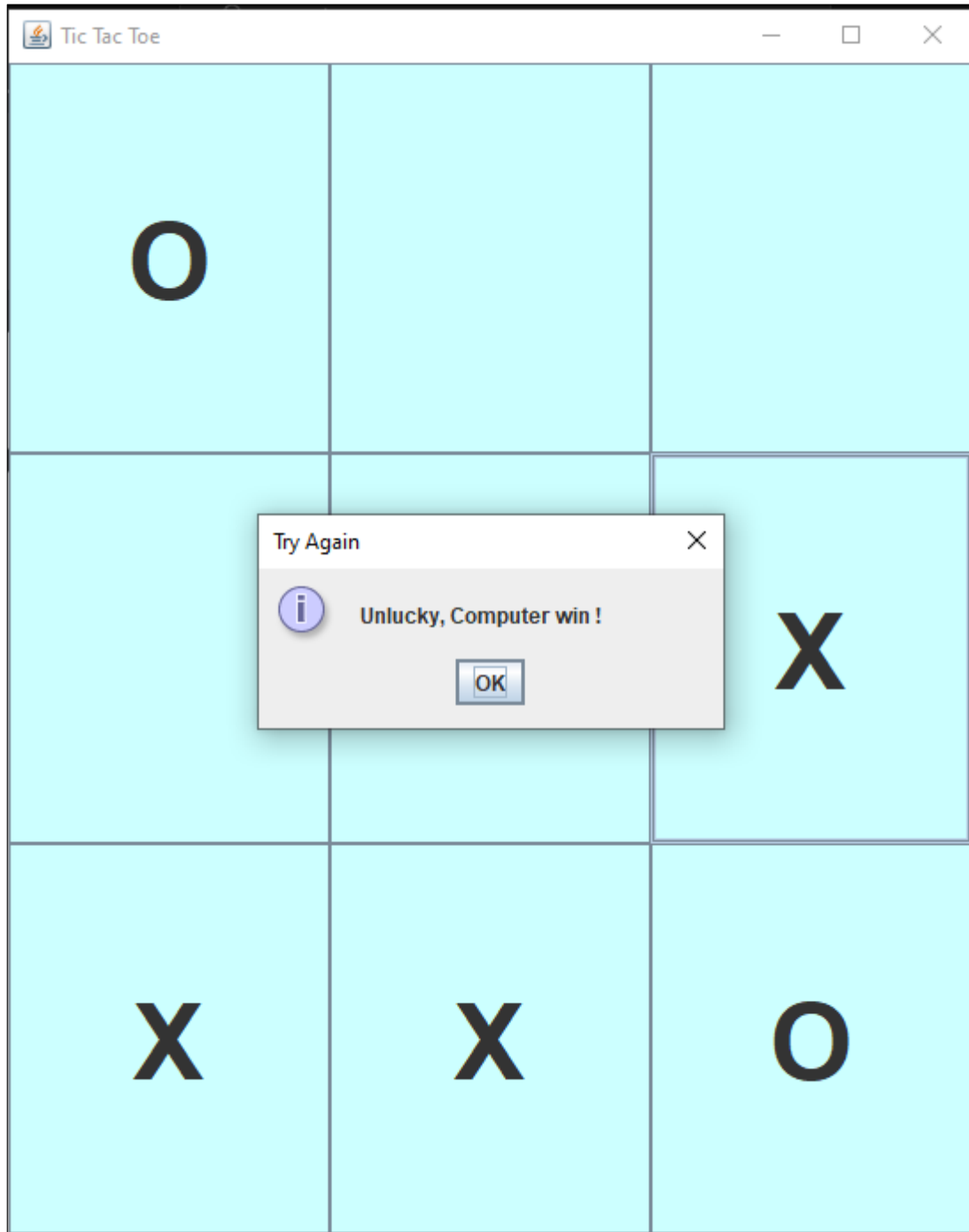
    if (isBoardFull()) {
        return 0;
    }

    if (isMaximizingPlayer) {
        int bestScore = Integer.MIN_VALUE;
        for (int i = 0; i < sizeLayout; i++) {
            for (int j = 0; j < sizeLayout; j++) {
                if (board[i][j] == Enull) {
                    board[i][j] = aiComputer;
                    int currentScore = minimax(depth + 1, false);
                    board[i][j] = Enull;
                    bestScore = Math.max(bestScore, currentScore);
                }
            }
        }
        return bestScore;
    }
```

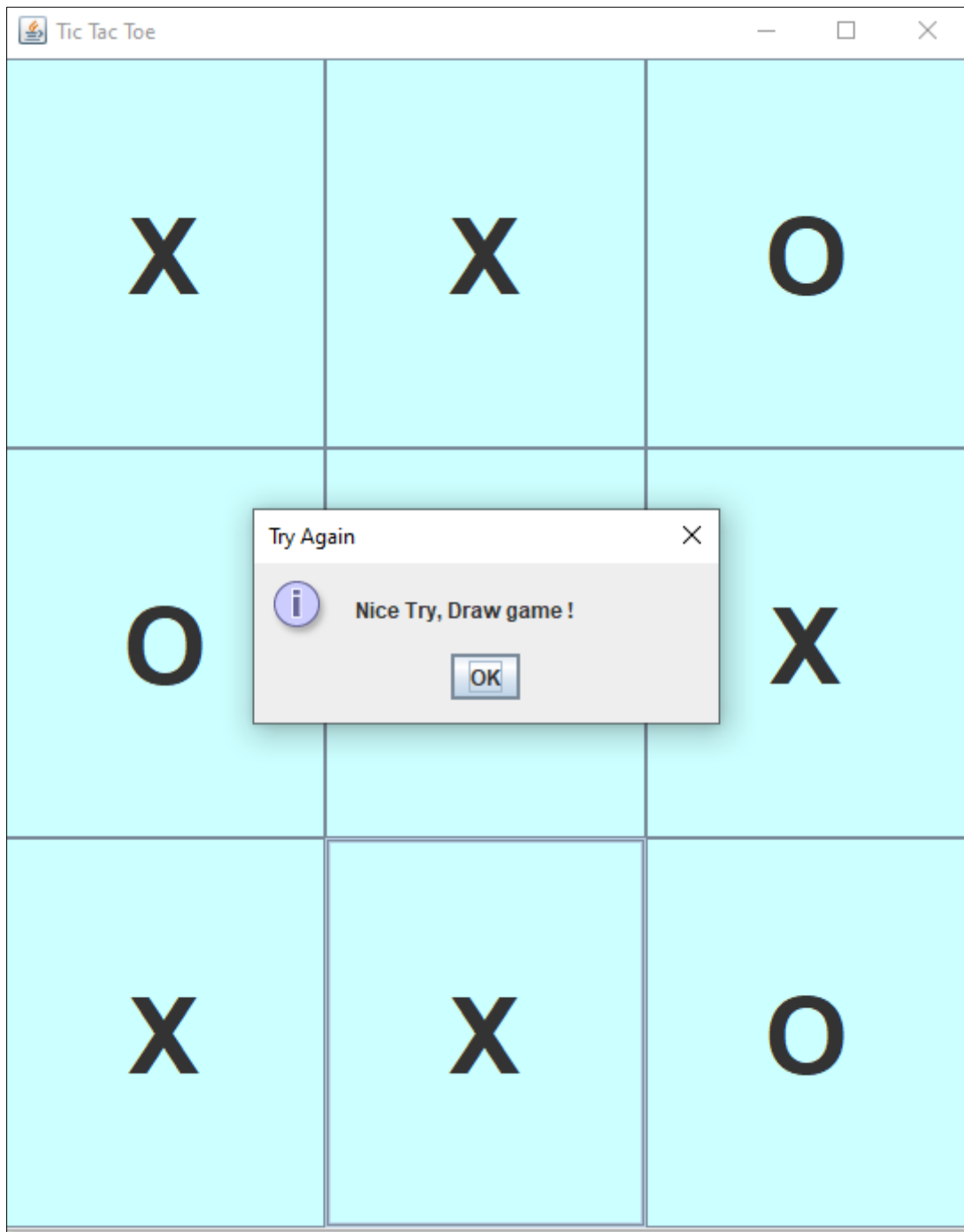
```
    } else {
        int bestScore = Integer.MAX_VALUE;
        for (int i = 0; i < sizeLayout; i++) {
            for (int j = 0; j < sizeLayout; j++) {
                if (board[i][j] == Enull) {
                    board[i][j] = userInput;
                    int currentScore = minimax(depth + 1, true);
                    board[i][j] = Enull;
                    bestScore = Math.min(bestScore, currentScore);
                }
            }
        }
        return bestScore;
    }
}
```

3.4 Screenshots of the user interface when players win, lose, or draw a game.

Scenario 1 – When computer wins.



Scenario 2 – When game draw.



Scenario 3 – When user wins.

3.5 Indicate the Data Structures used with its purpose.

- **2D Arrays:**

The puzzle board is represented by a 2D array (board) with a size of 8 by 8. Each cell of the array represents a position on the board, and the value 1 is used to indicate the presence of a queen at that position.

3.6 Specify using Code Segments or Screenshots the Validations and Exception Handling in this application.

```
try {  
    for (int i = 0; i < sizeLayout; i++) {  
        for (int j = 0; j < sizeLayout; j++) {  
            board[i][j] = Enull;  
        }  
    }  
} catch (Exception e) {  
    System.out.println(e);  
}
```

Chapter 4 : Identify Shortest Path

4.1 Program Logic used to implement Identify Shortest Path

Algorithms are widely used in programming. We utilized the Dijiskal algorithm to determine the shortest distance between two points. We first built a graph data structure to represent the connections between various vertices in order to answer the query. To initialize the vertices and edges, we developed two distinct classes. Every vertex must have neighbours and the corresponding edge weights.

Then we created an array to store the shortest path to each vertex from the source and assign the distance of the source to be 0 and distance of other vertex to infinity. After that we created a instance of a priority que to keep the vertices based on their current distance to the source vertex. In this case distance with the shortest distance will be always on the top of the queue.

As the next step we perform the dijiskal algorithm. There are few steps in this algorithm. As a first step of this algorithm, we had to get the top value from the priority queue. It's also known as the smallest distance. Then we marked that vertex as a visited one. After that we found the neighbour vertices of the smallest vertex and calculate the distance of the neighbour using the following logic.

```
If ( Distance(Vertex 1) + Distance(Vertex 1 , Vertex 2) < Distance(Vertex 2){  
    Distance(Vertex2) = Distance(Vertex 1) + Distance(Vertex 1 , Vertex 2)  
}
```

As a next step we added the neighbour into priority queue. We recursively did above steps until the priority queue is empty. At the completion of the algorithm, we were able to get the shortest path form source to every vertex.

4.2 Code Segment used to set random distance.

Using a call called “Random” we were able to generate values. We only need values between 5 and 50. Because of that we had to set a limit. Then we assign those random values into an array using a for loop. After that we assign those values for the distance

```

Random random = new Random();
int[] val = new int[13];

for (int i = 0; i < 13; i++) {
    val[i] = random.nextInt(45) + 5;
}

String cities = "ABCDEFGH I";
int i = random.nextInt(9);
String point = String.valueOf((cities.charAt(i)));

System.out.println("Distance AB = " + val[0] + "km AI = " + val[1] + "km BI = " + val[2] + "km BC = " + val[3]
    + "km CD = " + val[4] + "km DI = " + val[5] + "km DE = " + val[6] + "km EH = " + val[7] + "km EF = " + val[8]
    + "km FH = " + val[9] + "km FG = " + val[10] + "km HG = " + val[11] + "km HI = " + val[12] + "km");
System.out.println("Find the shortest path to each city from " + point);

```

4.3 UI screenshot allowing game players to provide answers.

When user come to the shortest path game from the selection menu, user interface appears as below. Here we have used a 560x700 JFrame,

User will be able to see the question with randomly generated source. Also, user can see each city along with the randomly generated distance to other cities. User can calculate the shortest distance to each city from the source and insert into text boxes and click the submit button. If all the answer are correct user will see a success message.

Hello Chamod
Find Minimum Connectors
<Back X

Puzzle - Question : Calculate the shortest path to each city from city C

From C to A : <input type="text"/> km	From C to B : <input type="text"/> km
From C to C : <input type="text"/> km	From C to D : <input type="text"/> km
From C to E : <input type="text"/> km	From C to F : <input type="text"/> km
From C to G : <input type="text"/> km	From C to H : <input type="text"/> km
From C to I : <input type="text"/> km	

4.4 Code Segment: find the shortest path and distance for other cities from the system's randomly selected city.

- Following code will randomly select a city as a source.

```
String cities = "ABCDEFGHI";  
int i = random.nextInt(9);  
String point = String.valueOf((cities.charAt(i)));
```

- Then we called the following method to find the index of the source. For that we parsed the source as a parameter .

```
public void dijkstraHeap(String source) {  
    int src = -1;  
    try {  
        for (int i = 0; i < numVertices; i++) {  
            if (vertices[i].id.toLowerCase().equals(source.toLowerCase())) {  
                src = i;  
            }  
        }  
    } catch (Exception e) {  
        System.out.println(e);  
    }  
  
    if (src == -1) {  
        return;  
    }  
    dijkstraHeapHelper(src);  
}
```

- Then we called the following method. We sent the source index as a parameter. This function will assign the source vertex distance as 0 and other vertex distance as infinity. Also, it set all vertices as unvisited and then add the source to priority queue.

```
private void dijkstraHeapHelper(int source) {  
    int numSet = 0;  
    try {  
        for (int i = 0; i < numVertices; i++) {  
            dist[i] = Integer.MAX_VALUE;  
            set[i] = false;  
        }  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.out.println(e);  
    }  
  
    pq.add(new Edge(source, 0));  
    dist[source] = 0;  
    numSet++;  
    while (numSet < numVertices) {  
        if (pq.isEmpty()) {  
            return;  
        }  
        int index = pq.remove().toIndex;  
        if (set[index]) {  
            continue;  
        }  
        set[index] = true;  
        processNeighbors(index);  
    }  
}
```

- This function will allow us to find the neighbours of the given vertex and find the shortest path using the logic mentioned in section 4.1.

```
private void processNeighbors(int x) {
    int edgeDistance = -1;
    int newDistance = -1;
    Vertex v = vertices[x];
    Iterator<Edge> itr;
    for (itr = v.edges.iterator(); itr.hasNext();) {
        Edge e = itr.next();
        int index = e.toIndex;
        if (!set[index]) {
            edgeDistance = (int) e.cost;
            newDistance = dist[x] + edgeDistance;
            if (newDistance < dist[index]) {
                dist[index] = newDistance;
            }
            pq.add(new Edge(index, dist[index]));
        }
    }
}
```

4.5 Code Segment used to save person's name along with the correct answer.

```
public void saveAnswers() {
    try {
        String query = "INSERT INTO `shortestpathanswers`"
            + "(`uid`, `uname`, `disBetCities`, `startPoint`, "
            + "`uanswer`) VALUES (?,?,?,2,2)";
        pst = con.prepareStatement(query);
        pst.setInt(1, userid);
        pst.setString(2, userName);
        pst.setString(3, disBetCities);
        pst.setString(4, startpoint);
        pst.setString(5, userAnswer);
        pst.executeUpdate();
        JOptionPane.showMessageDialog(null, "Your Answer is "
            + "Correct.", "Congratulations!", JOptionPane.INFORMATION_MESSAGE);
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

4.6 Code Segment used to save distance between cities when they correctly identify an answer.

```
public void saveAnswers(){
    try{ String query = "INSERT INTO `shortestpathanswers`"
        + "("uid`, `uname`, `disBetCities`, `startPoint`,`
        + " `uanswer`) VALUES  (?, ?, ?, ?, ?)";
        pst = con.prepareStatement(query);
        pst.setInt(1,userid);
        pst.setString(2,userName);
        pst.setString(3,disBetCities);
        pst.setString(4,startpoint);
        pst.setString(5,uswerAnswer);
        pst.executeUpdate();
        JOptionPane.showMessageDialog(null, "Your Answer is "
            + "[Correct].","Congragulations!",JOptionPane.INFORMATION_MESSAGE);
    }catch(Exception ex){
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

4.7 UI screenshots when game players to provide correct answer & incorrect answers.

Scenario 1 – If user insert the wrong answer

Hello Chamod

Find Minimum Connectors

<Back X

Puzzle -

Question : Calculate the shortest path to each city from city C

Try Again!

Wrong Answer.

OK

From C to A: km

From C to B: km

From C to C: km

From C to D: km

From C to E: km

From C to F: km

From C to G: km

From C to H: km

From C to I: km

Submit

Clear

scenario 2 – If user insert the correct answer

Hello Chamod

Find Minimum Connectors

<Back X

Puzzle -

Question : Calculate the shortest path to each city from city C

Congragulations! Your Answer is Correct.

OK

From **C** to **A**: km

From **C** to **B**: km

From **C** to **C**: km

From **C** to **D**: km

From **C** to **E**: km

From **C** to **F**: km

From **C** to **G**: km

From **C** to **H**: km

From **C** to **I**: km

Submit

Clear

4.8 .Indicate the Data Structures used with its purpose.

- **Array**
An array holds a fixed-length sequence of identical-type elements. It is a container that houses a group of values, each of which is designated by an index or key. Programming languages frequently use arrays to arrange and work with sets of data. In this example we used array to store vertices, distance of vertices, etc.
- **Priority Queue**
An abstract data type called a priority queue resembles a queue or a stack but has an additional priority assigned to each piece. The entries in a priority queue are given priorities, and the higher-priority elements are dequeued before the lower-priority elements.

4.9 Specify using Code Segments or Screenshots the Validations and Exception Handling in this game option.

```
try {
    for (int i = 0; i < numVertices; i++) {
        dist[i] = Integer.MAX_VALUE;
        set[i] = false;
    }
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println(e);
}
```

```
private void resultCKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    char c = evt.getKeyChar();
    if (evt.getKeyChar() >= '0' && evt.getKeyChar() <= '9') {
        resultC.setEditable(true);
    } else {
        if (evt.getExtendedKeyCode() == KeyEvent.VK_BACK_SPACE) {
            resultC.setEditable(true);
        } else {
            JOptionPane.showMessageDialog(null, "Please insert integer values");
            resultC.setText("");
        }
    }
}
```

```

if (resultA.getText().equals("") || resultB.getText().equals("") ||
    resultC.getText().equals("") || resultD.getText().equals("") ||
    resultE.getText().equals("") || resultF.getText().equals("") ||
    resultG.getText().equals("") || resultH.getText().equals("") ||
    resultI.getText().equals("")) {
    JOptionPane.showMessageDialog(null, "Please insert values");
} else {

```

4.10 Screenshot of the Normalized DB Table Structure used for this Game Option.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 said	int(11)			No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2 uid	int(11)			No	None			Change Drop More
<input type="checkbox"/>	3 uname	varchar(30)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/>	4 disBetCities	varchar(400)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/>	5 startPoint	varchar(10)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/>	6 uanswer	varchar(400)	latin1_swedish_ci		No	None			Change Drop More

+ Options

			said	uid	uname	disBetCities	startPoint	uanswer
<input type="checkbox"/>	Edit Copy Delete	3	3	Amasha	[A to B, 5, A to I, 38, B to I, 35, B to C, 38...	E	[E, A, 63, E, B, 60, E, C, 49, E, D, 9, E, E, 0, E...	
<input type="checkbox"/>	Edit Copy Delete	4	2	Chamod	[A to B, 32, A to I, 14, B to I, 37, B to C, 4...	C	[C, A, 58, C, B, 46, C, C, 0, C, D, 14, C, E, 54, ...	
		<input type="checkbox"/> Check all	With selected:		Edit Copy Delete Export			

Chapter 5 : Identify minimum connectors.

5.1 Program Logic used to Identify minimum connectors.

Here, we have taken five cities located at different distances from each other, select a certain city, and search for minimum connectors and minimum total distance according to all cities' connector order. For this we have used the prims algorithm.

In order to determine the minimal spanning tree (MST) of a linked weighted graph, Prim's approach is a well-known greedy algorithm. The MST is a subgraph that joins all of the original graph's vertices with the least amount of edge weight.

The subgraph that connects all of the vertices with the least overall edge weight is the resulting MST.

Prim's approach is effective and has an $O(E \log V)$ time complexity, where E is the number of edges in the graph and V is the number of vertices. The MST evolves from a single vertex to cover the complete graph using the technique, which guarantees that it is always linked.

5.2 Code Segment used to set random distance.

To connect two cities to each other, we use a 5 by 5 integer adjacency matrix. Here the four distances are kept between 10 and 100. The code is shown below. This process is carried out by Minimum Connectors.java class which is in the View package.

```
public static int[][] generateGraph(int numVertices, int minValue, int maxValue) {
    int[][] graph = new int[numVertices][numVertices];
    Random random = new Random();

    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            if (i == j) {
                graph[i][j] = 0; // No self-loops
            } else {
                int randomValue = random.nextInt(maxValue - minValue + 1) + minValue;
                graph[i][j] = randomValue;
                graph[j][i] = randomValue; // Graph is undirected
            }
        }
    }

    return graph;
}
```

5.3 UI screenshot allowing game players to provide answers.

When you come to the Find Minimum Connectors puzzle from the main menu of the game, the user interface appears as above. Here we have used a 560x700 JFrame, and basically, Two Jpanels have been added to it. The second Jpanel is a title bar. It has a button for the game player's name, the game's title, and main menu, and a button to close the game.

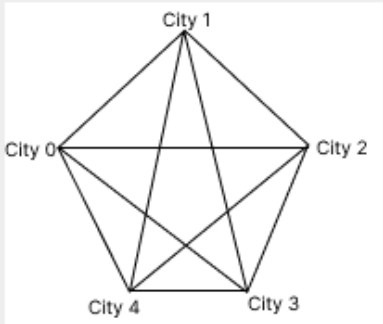
Then, the puzzle is explained with the help of a JLabel and the starting city is also shown here. Also, the connections of the cities and their distances are put in a JLabel. Also, a diagram is also put in a JLabel to make it easy to identify by the player. Here you can find the minimum connectors and minimum total distance and put them in the text boxes and check whether the answer is correct with the help of the submit button. Also, all the text boxes can be cleared by the clear button.

Hello ChamodFind Minimum Connectors<Back X

Puzzle - There are five cities connecting each other. Identify the minimum connectors & distances to connect all the cities & total distance starting from the city number : **3**

Distances between cities

0 -> 1 :38 Km || 0 -> 2 :76 Km || 0 -> 3 :13 Km || 0 -> 4 :63 Km || 1 -> 0 :38 Km || 1 -> 2 :70 Km || 1 -> 3 :10 Km || 1 -> 4 :82 Km || 2 -> 0 :76 Km || 2 -> 1 :70 Km || 2 -> 3 :84 Km || 2 -> 4 :79 Km || 3 -> 0 :13 Km || 3 -> 1 :10 Km || 3 -> 2 :84 Km || 3 -> 4 :89 Km || 4 -> 0 :63 Km || 4 -> 1 :82 Km || 4 -> 2 :79 Km || 4 -> 3 :89 Km ||



City		City	Distance
<input type="text"/>	to	1	<input type="text"/> Km
<input type="text"/>	to	2	<input type="text"/> Km
<input type="text"/>	to	3	<input type="text"/> Km
<input type="text"/>	to	4	<input type="text"/> Km
Total Distance		:	<input type="text"/> Km

Submit

Clear

5.4 UI screenshots when game players to provide correct answer & incorrect answers.

Scenario 1 – When user insert wrong answer.

Hello Chamod

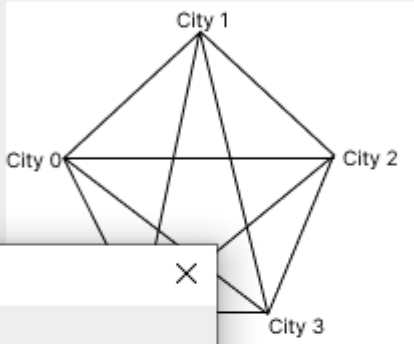
Find Minimum Connectors

<Back X


Puzzle - There are five cities connecting each other. Identify the minimum connectors & distances to connect all the cities & total distance starting from the city number : **3**

Distances between cities

0 -> 1 :38 Km || 0 -> 2 :76 Km || 0 -> 3 :13 Km || 0 -> 4 :63 Km || 1 -> 0 :38 Km || 1 -> 2 :70 Km || 1 -> 3 :10 Km || 1 -> 4 :82 Km || 2 -> 0 :76 Km || 2 -> 1 :70 Km || 2 -> 3 :84 Km || 2 -> 4 :79 Km || 3 -> 0 :13 Km || 3 -> 1 :10 Km || 3 -> 2 :84 Km || 3 -> 4 :89 Km || 4 -> 0 :63 Km || 4 -> 1 :82 Km || 4 -> 2 :79 Km || 4 -> 3 :89 Km ||



Try Again!

 Wrong Answer.

OK

City		City	Distance
0	to	1	34 Km
1	to	2	43 Km
2	to	3	11 Km
32	to	4	23 Km
Total Distance		:	111 Km

Submit

Clear

Scenario 2 – When user insert the correct answer.

Hello Chamod

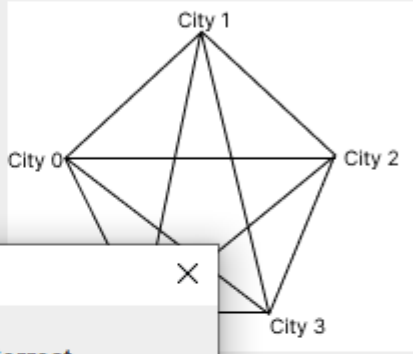
Find Minimum Connectors

<Back X


Puzzle - There are five cities connecting each other. Identify the minimum connectors & distances to connect all the cities & total distance starting from the city number : **3**

Distances between cities

0 -> 1 :39 Km || 0 -> 2 :85 Km || 0 -> 3 :23 Km || 0 -> 4 :40 Km || 1 -> 0 :39 Km || 1 -> 2 :47 Km || 1 -> 3 :25 Km || 1 -> 4 :69 Km || 2 -> 0 :85 Km || 2 -> 1 :47 Km || 2 -> 3 :52 Km || 2 -> 4 :80 Km || 3 -> 0 :23 Km || 3 -> 1 :25 Km || 3 -> 2 :52 Km || 3 -> 4 :85 Km || 4 -> 0 :40 Km || 4 -> 1 :69 Km || 4 -> 2 :80 Km || 4 -> 3 :85 Km



Congratulations!

 Your Answer is Correct.

OK

City		City	Distance	
3	to	1	25	Km
1	to	2	47	Km
0	to	3	23	Km
0	to	4	40	Km
Total Distance		:	135	Km

Submit

Clear

5.5 .Code Segment used to save person's name along with the correct answer.

Here, the MinimumConnectorsLogics.java class, which is in the Controller package, passes the user's information and the user's right response as a string to the MinimumConnectorsSql.java class, which is in the Model package. Then, add the right response, the gameplayer's ID, and the player's name to the "minimumconnectorsanswers" Table in Database. The code can be seen below.

```
public void saveAnswers(){
    try{ String query = "INSERT INTO "
        + "'minimumconnectorsanswers'('uid', 'uname', "
        + "'disBetCities', 'startPoint', 'uanswer', 'uTotDis') VALUES (?, ?, ?, ?, ?, ?)";
        pst = con.prepareStatement(query);
        pst.setInt(1,userid);
        pst.setString(2,userName);
        pst.setString(3,disBetCities);
        pst.setInt(4,startpoint);
        pst.setString(5,uswerAnswer);
        pst.setInt(6,totalDistance);
        pst.executeUpdate();
        JOptionPane.showMessageDialog(null, "Your Answer is Correct."
            + "\n", "Congragulations!", JOptionPane.INFORMATION_MESSAGE);
    }catch(Exception ex){
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

5.6 Code Segment used to save distance between cities when they correctly identify an answer.

What happens here is that the player's answer is obtained through the user interface, puts it in to integer array and compares it with the one with the correct answers array. Then can know whether the player's answer is right or wrong. Here player's answer get in to a array and pass by MinimumConnectors.Jframe which is in the View package to the MinimumConnectorsLogics.java class which is in the Controller package.

```
public void setAnswers(int []userAnswers,int totalDis,String disBetCities){
    this.userAnswers =userAnswers;
    userTotalDistance = totalDis;
    this.disBetCities = disBetCities;
    boolean result = compareArrays(validAnswers, userAnswers);
    boolean result1;
    if (userTotalDistance==totalDistanceT) {
        result1=true;
    } else {
        result1=false;
    }
    if (result == true && result1==true ) {
        MinimumConnectorsSql mcSql= new MinimumConnectorsSql(uid, uname,
            disBetCities, startPoint, Arrays.toString(userAnswers), totalDistanceT);
        mcSql.saveAnswers();
    } else {
        JOptionPane.showMessageDialog(null, "Wrong Answer.", "Try Again!", JOptionPane.ERROR_MESSAGE);
    }
}
```

5.7 Indicate the Data Structures used with its purpose.

- **2D Array graph:**

The graph is represented by a 2D array in the graph variable. The weights (distances) between the vertices are stored. The links between various graph vertices are represented by this data structure.

Array visited: During the MST construction, the visited array—a boolean array—is utilized to record the visited vertices. It aids in figuring out which vertices are part of the MST.

Arrays parent and key: Prim's technique stores the MST information in the parent and key arrays. The key array stores the lowest weight necessary to reach each vertex from the MST, while the parent array stores the parent vertex for each vertex in the MST. These arrays are utilized to build the MST and identify the edges with the least weight.

- **ArrayList mst:**

The minimal spanning tree (MST) edges are kept in the mst ArrayList. Initially blank, it gains edges as part of the MST assembly process.

5.8 Specify using Code Segments or Screenshots the Validations and Exception Handling in this game option.

To determine if the player's responses are correct or incomplete, a mechanism has been developed. The player will receive a notification if there is an error in the responses in this case. Additionally, the solutions can only be given as numerical data. As a result, we introduced a keypress event that, when one character key is pushed, shows an associated error message. Additionally, the data structures utilized here have been built with exception handling, and data entry into the database has also been done thus.

```
try{ String query = "INSERT INTO "
    + "minimumconnectorsanswers ('uid', 'uname', "
    + "disBetCities', 'startPoint', 'uanswer', 'uTotDis') VALUES (?, ?, ?, ?, ?, ?)";
    pst = con.prepareStatement(query);
    pst.setInt(1,userid);
    pst.setString(2,userName);
    pst.setString(3,disBetCities);
    pst.setInt(4,startpoint);
    pst.setString(5,uswerAnswer);
    pst.setInt(6,totalDistance);
    pst.executeUpdate();
    JOptionPane.showMessageDialog(null, "Your Answer is Correct."
        + " ", "Congratulations!", JOptionPane.INFORMATION_MESSAGE);
} catch (Exception ex) {
    JOptionPane.showMessageDialog(null, ex);
}
```

```

if (val1.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter city numbers", "Error", JOptionPane.ERROR_MESSAGE);
} else if (distance1.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter distance between cities", "Error", JOptionPane.ERROR_MESSAGE);
} else if (val2.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter city numbers", "Error", JOptionPane.ERROR_MESSAGE);
} else if (distance2.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter distance between cities", "Error", JOptionPane.ERROR_MESSAGE);
} else if (val3.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter city numbers", "Error", JOptionPane.ERROR_MESSAGE);
} else if (distance3.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter distance between cities", "Error", JOptionPane.ERROR_MESSAGE);
} else if (val4.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter city numbers", "Error", JOptionPane.ERROR_MESSAGE);
} else if (distance4.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter distance between cities", "Error", JOptionPane.ERROR_MESSAGE);
} else if (total.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please enter total distance of between cities", "Error", JOptionPane.ERROR_MESSAGE);
} else {
    getUserInput(total);
}

```

```

//Validations for event handling
private void txt_val2KeyPressed(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    if (evt.getKeyChar() >= '0' && evt.getKeyChar() <= '9') {
        txt_val2.setEditable(true);
    } else {
        if (evt.getExtendedKeyCode() == KeyEvent.VK_BACK_SPACE) {
            txt_val2.setEditable(true);
        } else {
            JOptionPane.showMessageDialog(this, "Please enter "
                + "Integer Value !", "Information", JOptionPane.ERROR_MESSAGE);
            txt_val2.setText("");
        }
    }
}
}

```

5.9 .Screenshot of the Normalized DB Table Structure used for this Game Option.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	caid	int(11)		No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2	uid	int(11)		No	None			Change Drop More
<input type="checkbox"/>	3	uname	varchar(30)	latin1_swedish_ci	No	None			Change Drop More
<input type="checkbox"/>	4	disBetCities	varchar(500)	latin1_swedish_ci	No	None			Change Drop More
<input type="checkbox"/>	5	startPoint	int(11)		No	None			Change Drop More
<input type="checkbox"/>	6	uanswer	varchar(150)	latin1_swedish_ci	No	None			Change Drop More
<input type="checkbox"/>	7	uTotDis	int(11)		No	None			Change Drop More

+ Options

			caid	uid	uname	disBetCities	startPoint	uanswer	uTotDis
<input type="checkbox"/>	Edit	Copy	Delete	1	1	Harith	0 -> 1 :19 Km 0 -> 2 :81 Km 0 -> 3 :74	0	[0, 1, 19, 3, 2, 12, 1, 3, 41, 0, 4, 23] 95
<input type="checkbox"/>	Edit	Copy	Delete	2	1	Harith	0 -> 1 :83 Km 0 -> 2 :26 Km 0 -> 3 :60	4	[4, 1, 42, 0, 2, 26, 2, 3, 47, 2, 4, 20] 135
<input type="checkbox"/>	Edit	Copy	Delete	3	1	Harith	0 -> 1 :54 Km 0 -> 2 :40 Km 0 -> 3 :71	2	[2, 1, 34, 4, 2, 21, 4, 3, 49, 0, 4, 28] 132
<input type="checkbox"/>	Edit	Copy	Delete	4	1	Harith	0 -> 1 :60 Km 0 -> 2 :37 Km 0 -> 3 :79	3	[3, 1, 13, 0, 2, 37, 2, 3, 26, 0, 4, 55] 131

☐ Check all
 ☐ With selected
 Edit
 Copy
 Delete
 Export

DB Link -

<https://drive.google.com/file/d/1sEkaBKweLZeuGeumMI6zkEkk4clkVySl/view?usp=sharing>

Game Play - [https://drive.google.com/file/d/1x-](https://drive.google.com/file/d/1x-OA0KMVqDLckvBTAAqCgqW5adH0FZl/view?usp=sharing)

[OA0KMVqDLckvBTAAqCgqW5adH0FZl/view?usp=sharing](https://drive.google.com/file/d/1x-OA0KMVqDLckvBTAAqCgqW5adH0FZl/view?usp=sharing)