

**Dharmsinh Desai University, Nadiad**  
**Faculty of Technology**  
**Department of Computer Engineering**



**B. Tech. CE Semester – VI**

**Subject: System Design Practice**

**Project Title:**  
**N-Queen Game**

**By:**

**Namrata P. Brahmhatt**  
**(Roll No: CE-012, ID: 16CEUBG001)**

**Krishna M. Mistry**  
**(Roll No: CE-069, ID: 16CEUBS063)**

**Guided by:**

**Prof. Mrudang Mehta**  
**Associate Professor**  
**Department of Computer Engineering**



**Faculty of Technology  
Department of Computer Engineering  
Dharmsinh Desai University**

## **CERTIFICATE**

This is to certify that the project carried out in the subject of System Design Practice titled **N-Queen Game** and recorded in this report is a bonafide work of

Ms. Namrata Brahmbhatt (CE-12, 16CEUBG001) and Ms. Krishna M. Mistry (CE-69, 16CEUBS063) of the degree of Bachelor of Technology semester VI in the branch of Computer Engineering during the academic year December 2018 to April 2019.

Prof. Mrudang Mehta  
Associate Professor.  
Dept. of Computer Engineering.  
Faculty of Technology  
Dharmsinh Desai University

Dr. C. K. Bhensdadia  
Head of Department  
Dept. of Computer Engineering  
Faculty of Technology  
Dharmsinh Desai University

# CONTENTS

## Table of Contents

<b>I. Abstract .....</b>	<b>iv</b>
<b>1. Introduction.....</b>	<b>1</b>
<b>2. System Requirements Specifications .....</b>	<b>3</b>
2.1 Introduction.....	4
2.2 Overall Description.....	5
2.3 Functional Requirements .....	6
2.4 Non-Functional Requirements .....	8
<b>3. Design .....</b>	<b>9</b>
3.1 System Architecture.....	10
3.2 Use Case Diagram .....	11
3.3 Class Diagram.....	12
3.4 Sequence Diagram .....	12
3.5 Activity Diagram .....	13
3.6 Algorithm.....	14
3.7 Flowchart of the algorithm.....	15
3.8 Time Complexity of the algorithm.....	16
<b>4. Implementation Details.....</b>	<b>17</b>
<b>5. Testing .....</b>	<b>20</b>
<b>6. Screenshots of the system.....</b>	<b>22</b>
<b>7. Conclusion .....</b>	<b>30</b>
<b>8. Limitations and Future Extensions .....</b>	<b>32</b>
<b>9. Bibliography.....</b>	<b>34</b>

# **I. ABSTRACT**

In n-Queen game, the goal is to place “n” queens such that no two queens should attack each other on the board of  $n \times n$  squares. The project focuses on solving n-queen problem using backtracking algorithm. Using backtracking algorithm, the idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of solution. If we do not find such a row due to clashes then we backtrack and return false. The user places queens after understanding rules of the game and try to solve the problem. The application will then compute the solution and checks whether user has placed queens correctly or not. If not, user can view the solution or retry the game. Otherwise, user can play the next level of the game where n is incremented by one.

# CH- 1

## INTRODUCTION

---

## **1.1 Brief Description:**

The project N-Queen Game mainly focuses on placing N queens on a board of  $n \times n$  squares such that no queens can attack and defeat any other queens on the board. The solution of the N queen problem is calculated using the recursive backtracking algorithm. The algorithm starts by placing the queen on the column 1 of the first row and advances further to check for the clashes between the queens. If any two queens are found clashing each other the algorithm backtracks and starts placing the queen in the next column. Thus, the solution of the N queen problem is obtained. Then the solution is compared with the positions of the queens placed by the user. If the solution matches, the user moves to the next level else, the user has to play the level again or s/he can also view the solution. The game is also provided with a number of themes which can be applied to the board according to the user's choice. At any time, the user can go to the help page.

## **1.2 Technology/Platform/Tools used:**

- Java Netbeans IDE for Swing Application: Using Swing application in Netbeans IDE, the project displays various GUI frames required for n-queen game.
- MySQL Workbench for managing database: The project has the user class which stores user details for authentication purpose.

## CH- 2

# SOFTWARE REQUIREMENTS SPECIFICATIONS

---

## **2.1 Introduction**

### **2.1.1 Purpose**

The purpose of brain games is to improve your memory, thinking, reaction time, cognitive ability. These games are especially important later in life to keep these factors strong. While it may seem hard to believe, by simply playing certain games you can simply avoid degeneration and memory loss by continuing to work your brain and keep it sharp. Brain games are usually simple in design but have a great impact.

### **2.1.2 Document Conventions**

Highlighted text are the headings for which description is provided. Functional requirements are intended on basis of its priorities. Priorities for higher-level requirements are assumed to be inherited by detailed requirements.

### **2.1.3 Intended Audience and Reading Suggestions**

Document is intended for different types of reader, such as developers, project managers, marketing staff, testers, and documentation writers. Rest of this SRS contains product perspective, product function, user characteristics, constraints, functional requirements and non-functional requirements.

### **2.1.4 Product Scope**

The system is operational any time. It is compatible with all kinds of web clients. The system is reliable, accurate, robust and secure.



### **2.1.5 References**

IEEE, IEEE-Std 830-1998 IEE Recommended practice for System Requirement Specifications.

## **2.2 Overall Description**

### **2.2.1 Product Perspective**

Software described in this SRS is the software for a complete Game. The system merges various hardware and software elements and further interfaces with external systems. Thus, while the software covers the majority of the system's functionality, it relies on a number of external interfaces for persistence and unhandled tasks, as well as physically interfacing with humans.

### **2.2.2 Product Functions**

The major functionalities of the system are storing user details, scores of the user based on number of moves and calculating best scores.

### **2.2.3 User Classes and Characteristics**

The customer is expected to be Windows or any OS literate and to be able to use button, pull-down menus and similar tools.

### **2.2.4 Operating Environment**

The application can be operated from a PC.

### **2.2.5 Design and Implementation Constraints**

The Software can work on any environment but it also has some minimum requirements.

Hardware:

- Processor: 800MHz Intel Pentium III or Equivalent
- Memory: 512MB
- Disk space: 750MB of free disk space.

Software:

- Netbeans IDE, JRE7 or JRE8, JDK7 or JDK8, Firefox, Chrome.

User do not feel any incontinency for the people playing outdoor games, and this may limit them to enjoy their sports.

### **2.2.6 User Documentation**

In our system we will provide various documents to help users know and understand how to play the game.

### **2.2.7 Assumptions and Dependencies**

The Software needs the user to be logged into their account for playing the Game.

## **2.3 System Features**

### **2.3.1 Gamer details**

I/P : User Name, Date of Birth, City

O/P : Redirected to Home page

Process : Gamer is registered if not

### **2.3.2 Selecting Level**

I/P : Selects level

O/P : Shows board of selected level

Process : Level applied

### **2.3.3 Choosing Theme**

I/P : Chooses theme  
O/P : Shows selected theme applied on board  
Process : Theme applied

### **2.3.4 Playing Game**

I/P : Clicks on any square on the board  
O/P : Shows queen placed on that square  
Process : Places queen on the board

### **2.3.5 Get Solution**

I/P : Selects get solution option  
O/P : Shows solution  
Process : Calculates solution and displays solution

### **2.3.6 Get Help**

I/P : Selects help option  
O/P : Shows help page  
Process : Redirecting to help page

### **2.3.7 Submitting Game**

I/P : Selects submit option  
O/P : Shows score page  
Process : Calculates moves and display score

### **2.3.8 Resetting Game**

I/P : Selects reset option

O/P : Shows new board game

Process : Removes all queens from the board and restarts game

## **2.4 Nonfunctional Requirements**

### **2.4.1 Reliability**

Our game will be able to recover from hardware failure. Failures and other natural catastrophes should roll back other databases to consistent state.

### **2.4.2 Security**

Our game will give access to authorized person only.

### **2.4.3 Scalability**

The system will be able to expand to meet future need.

### **2.4.4 Usability**

The user will be able to understand content of system. The system will provide an easy to use interface which will be very well understood by the user.

## CH- 3

### DESIGN

---

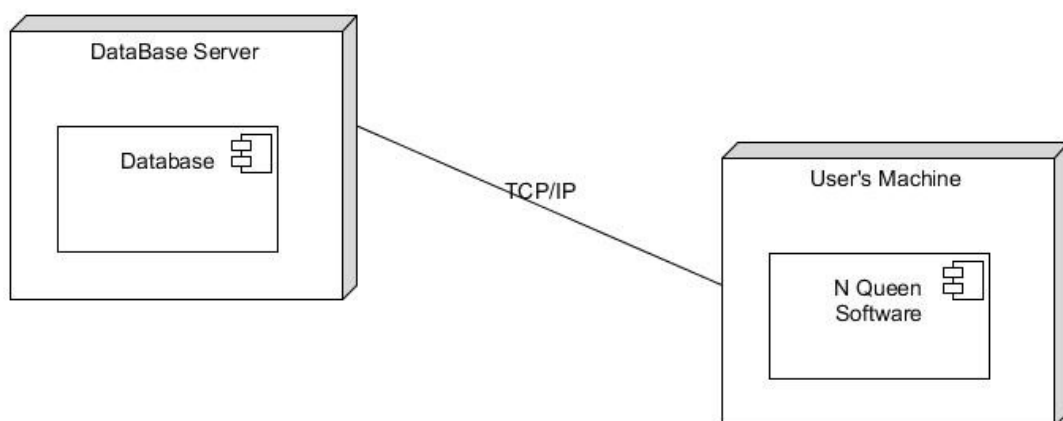
### 3.1 System Architecture:

MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts –

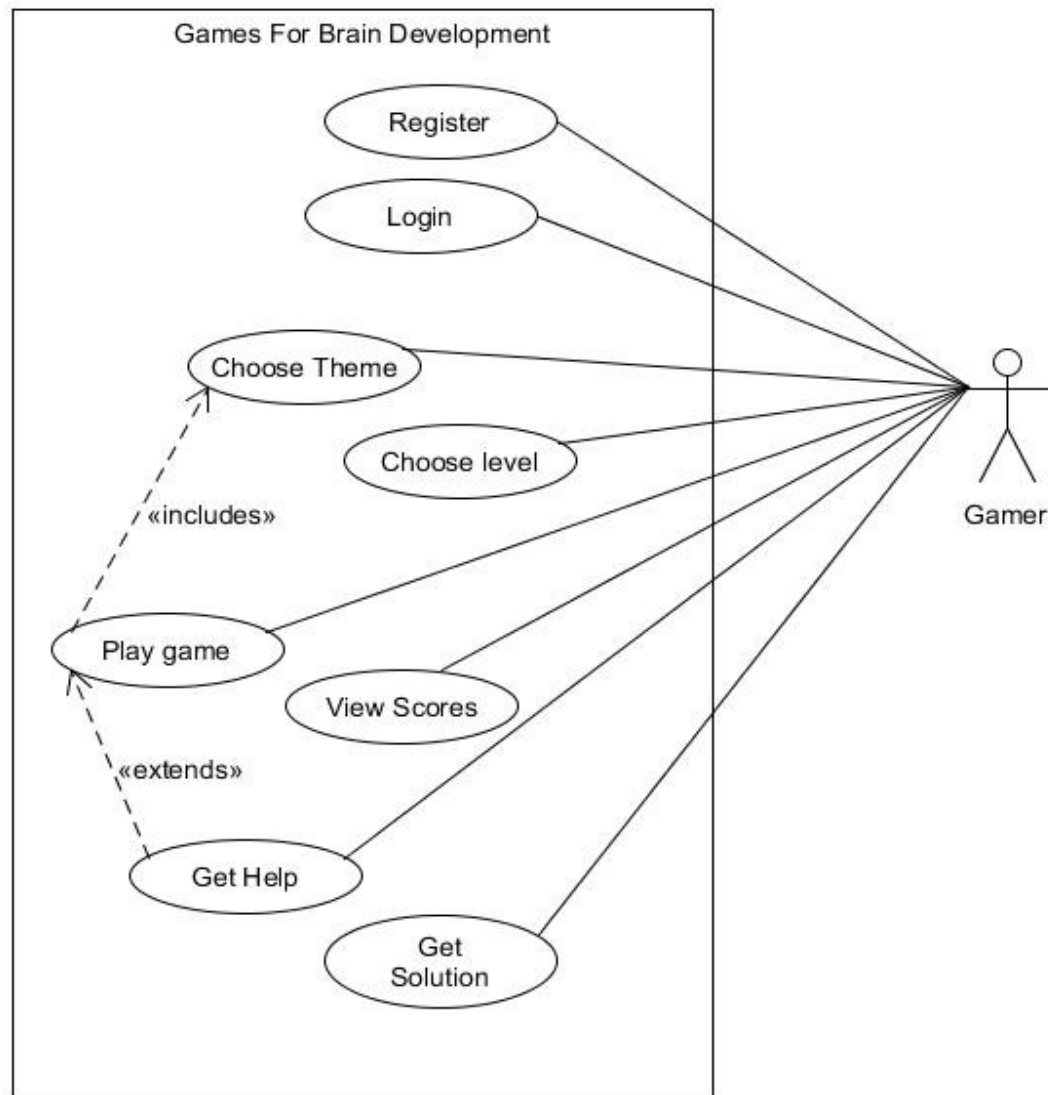
- **Model** – The lowest level of the pattern which is responsible for maintaining data.
- **View** – This is responsible for displaying all or a portion of the data to the user.
- **Controller** – Software Code that controls the interactions between the Model and View.

In the N- Queen Game, there is the Model named User for User details. The View contains the view for selecting theme, for selecting level, for playing game, to view help and to view best score. The Controller has to check whether the gamer wins or lose, to calculate the score of gamer and the algorithm for the solution of n-queen problem.

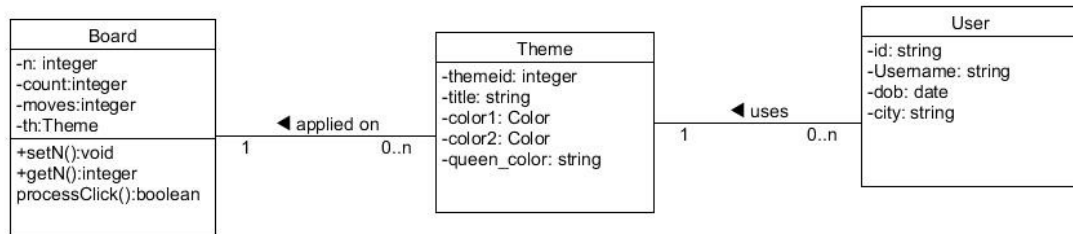
The deployment diagram for the project is as follows:



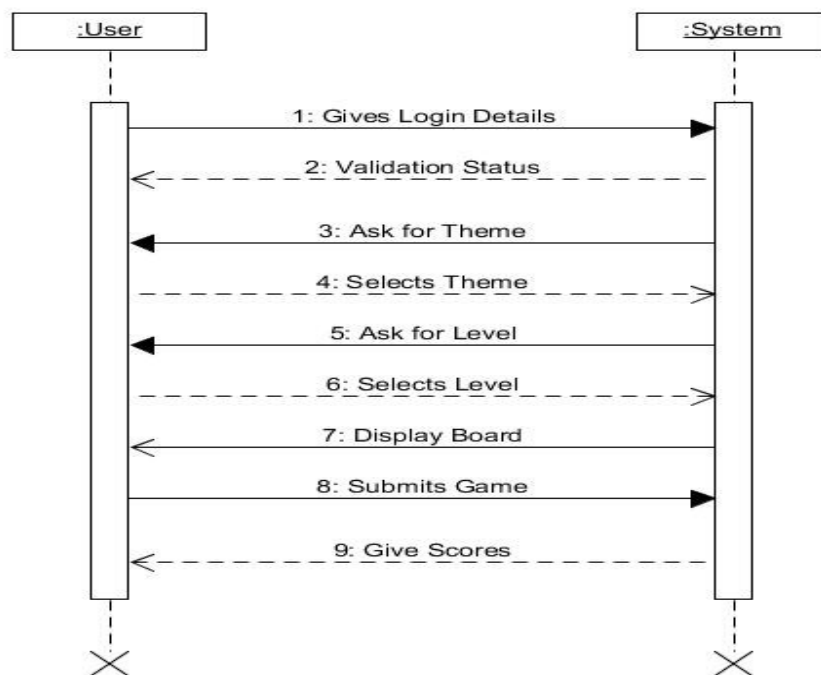
**3.2 Use Case Diagram:** The Gamers logs in to the system or register themselves if not registered. The Gamer plays game by placing queens on the board. The Gamers can choose theme or level of the game. The Gamers can view help or solution if they lose the game.



**3.3 Class Diagram:** To implement the n-queen game, following classes will be used by the system: Board, Theme and User. Board class to display board of  $n \times n$  squares, Theme class to be applied on the board and User class to manage user details.

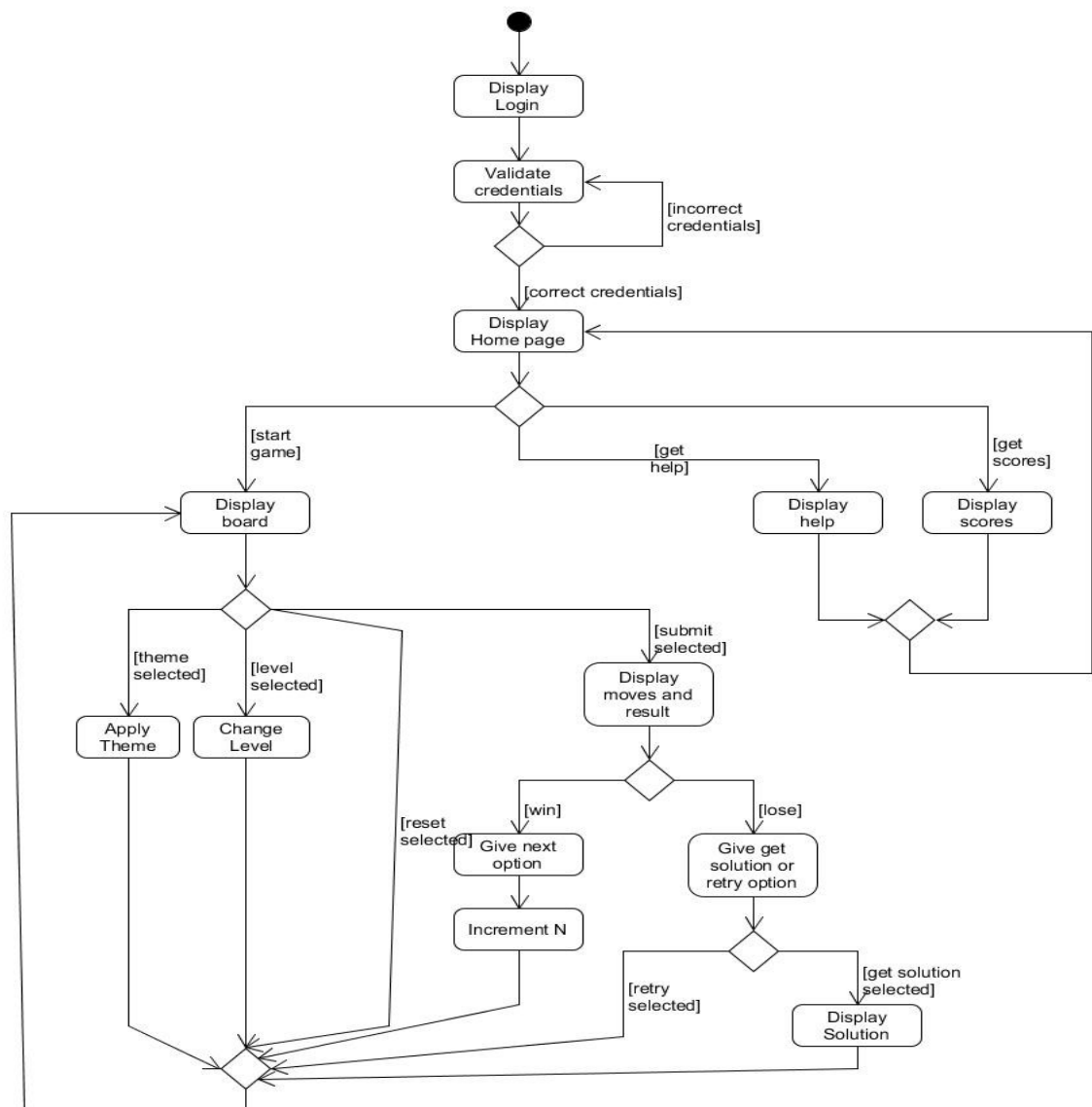


**3.4 Sequence Diagram:** The user first logs into the system by entering the credentials. The system validates the credentials and redirects to the home page. The user selects the level and plays the game and submits it according to which the system displays the result.





**3.5 Activity Diagram for the play game activity (system's perspective):** The system displays the login page and validates the entered credentials, if correct redirects to the home page. From home page the user can play the game or help. After playing the game the user submits it so s/he is redirected to the scoreboard where the moves and the result is displayed, and the next level option is provided. If the user loses the game then s/he can get the solution or play the same level again. The user can also change the theme or the level from the board.

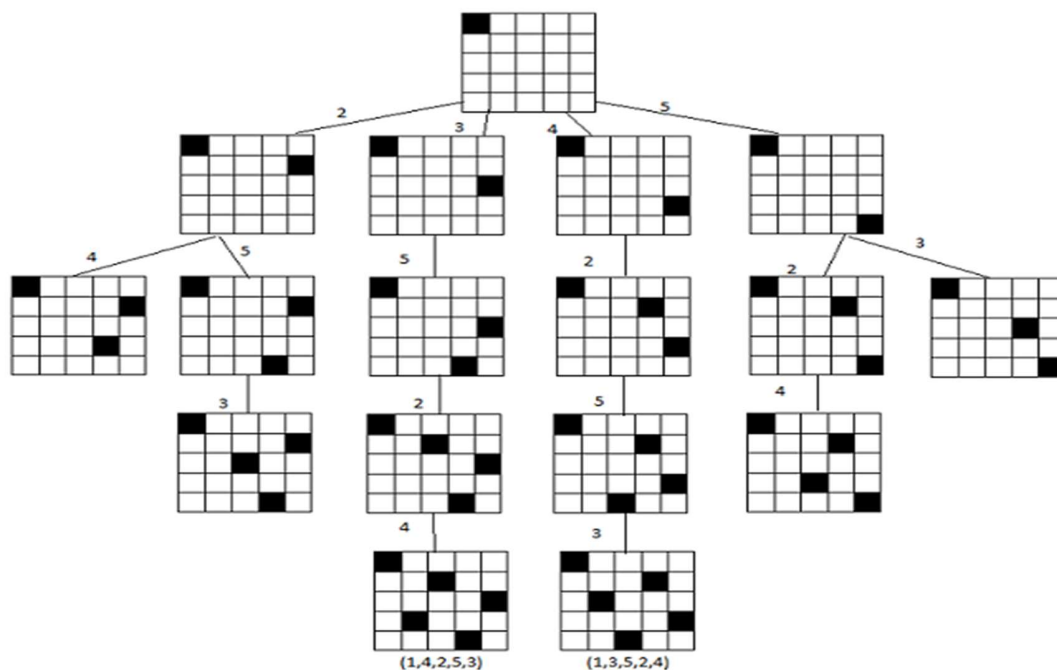


### 3.6 Algorithm:

The Game solves the n-queen problem using back tracking algorithm described below:

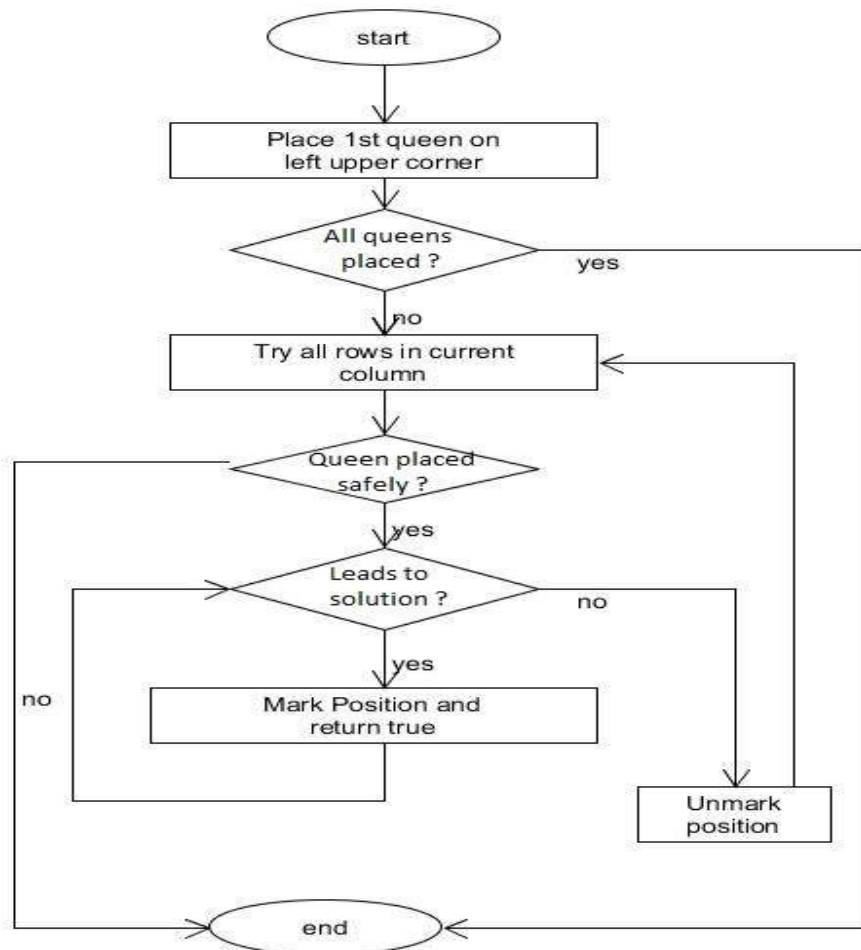
1. Start in the leftmost column.
2. If all queens are placed,  
return true
3. Try all rows in the current column. Do following for every tried row
  - a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if the placing queen here leads to a solution.
  - b) If placing the queen in [row, column] leads to a solution then return true.
  - c) If placing queen does not lead to a solution then mark this [row, column] (backtrack) and go to step (a) to try another row.
4. If all the rows have been tried and nothing worked, return false to trigger backtracking.

Example of 5-queen problem:



Since our board is  $5 \times 5$ , our recursion will be 5 levels deep. At the 0<sup>th</sup> level recursion, we place the 0<sup>th</sup> Queen on the 0<sup>th</sup> row. At the 1<sup>st</sup> level of recursion, we place the 1<sup>st</sup> queen on the 1<sup>st</sup> row such that she does not attack the 0<sup>th</sup> queen. At the 2<sup>nd</sup> level of recursion, we place 2<sup>nd</sup> queen on the 2<sup>nd</sup> row such that she does not attack the 1<sup>st</sup> and 0<sup>th</sup> queen and so on. At any point of time if we cannot find a cell for a queen on that row, we return false to the calling function and that level of recursion will then try to place queen on the next available cell of that row. If that doesn't work out then that function itself is going to return false to the calling function above it and so on.

### 3.7 Flowchart of the algorithm:



### **3.8 Time Complexity of the algorithm:**

The function for the time complexity to solve the queen problem is:

$$T(n) = n * T(n - 1) + O(1)$$

which gives the time complexity of  $O(n!)$ , where  $n$  is no queens.

Thus, the time complexity for this algorithm is  $O(n!)$  because of recursive calls and backtracking.

## CH- 4

# IMPLEMENTATION DETAILS

---

## Modules created and brief description of each modules:

- 1) **GUI Module:** This module initializes the GUI and displays it to the user.
- 2) **Solve N-Queen problem:** This module solves the n queen problem using the backtracking algorithm.

## Function prototypes which implement major functionality:

**compareMat:** Compares the matrices and return boolean values based on the comparison.

**printSolution:** Prints the matrix which is passed as an argument as the solution.

**isSafe:** Validating the positions of the queens placed on board and returns boolean values.

**solveNQUtil:** Calls the isSafe for validating the position, calls solveNQUtil recursively and compares the obtained solution matrix with the board matrix using compareMat.

**solveNQ:** Initializes the value of n and calls solveNQUtil with appropriate arguments and returns boolean values.

**processClick:** Calls solveNQ and returns boolean values according to the value returned by solveNQ.

**initializeGUI:** Creates the chess Board object and initializes the gui.

## To check whether placed queen is safe or not:

```
static boolean isSafe(int board[][], int row, int col)
{
    int i, j;
    for (i = 0; i < col; i++)
        if (board[row][i] == 1)
            return false;
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j] == 1)
            return false;
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j] == 1)
            return false;
    return true;
}
```

Let us assume the following condition:

	0	1	2	3	4
0		■			
1					
2				■	
3	■				
4					

Let queen to placed is on the square(3,2). From the first loop it will return false as the queen is placed on the square(3,0) at i=0 only. Let queen to placed is (1,2). From the second loop it will return false as queen is placed on the square(0,1) at i=0 and j=1. Let queen to be placed is (4,4). After iterating to all the loops, isSafe() returns true as queen can attack on queen placed at(4,4).

### Solving N-Queen problem:

```

boolean solveNQUtil(int board[][], int col)
{
    if (col >= N)
        return true;
    for (int i = 0; i < N; i++)
    {
        if (isSafe(board, i, col))
        {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1) == true)
                return true;
            board[i][col] = 0; // BACKTRACK
        }
    }
    return false;
}

boolean solveNQ()
{
    int board[][] = new int[N][N];
    if (solveNQUtil(board, 0) == false)
    {
        System.out.print("Solution does not exist");
        return false;
    }
    printSolution(board);
    return true;
}

```

## CH- 5

## TESTING

---

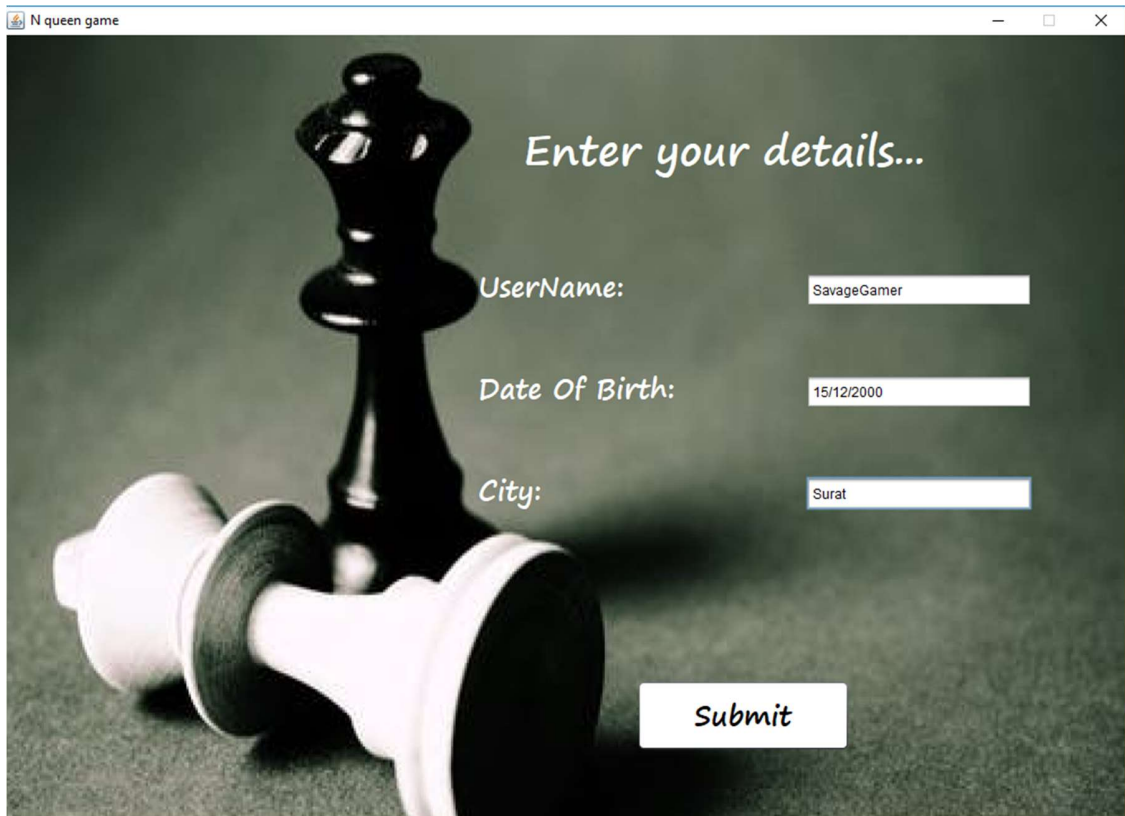


<b>Test case ID</b>	<b>Test Scenario</b>	<b>Test Steps</b>	<b>Tests Data</b>	<b>Expected Result(s)</b>	<b>Pass or Fail</b>
<b>T01</b>	Input non-existent username	Enter a new username	username	Registered successfully and redirect to home page	Pass
<b>T02</b>	Input existing username	Enter already existing username	username	Error Message	Pass
<b>T03</b>	Input username and date of birth	Enter correct username and date of birth	username and password	logged in successfully and redirect to home page	Pass
<b>T04</b>	Input Incorrect username and date of birth	Enter improper username and password	Incorrect username and password	Error Message	Pass
<b>T05</b>	Input correct solution on the board	Place queens on correct positions	place queens	Winning result on scoreboard	Pass
<b>T06</b>	Input incorrect solution on the board	Place queens on correct positions	place queens	Losing result on scoreboard	Pass

## CH- 6

### SCREEN-SHOTS

---



The image shows a web browser window titled "N queen game". The background is a dark green chessboard with a black king and a white queen. The text "Enter your details..." is displayed in a white, cursive font. Below this, there are three input fields: "UserName:" with the value "SavageGamer", "Date Of Birth:" with the value "15/12/2000", and "City:" with the value "Surat". A white "Submit" button is located at the bottom right.

Enter your details...

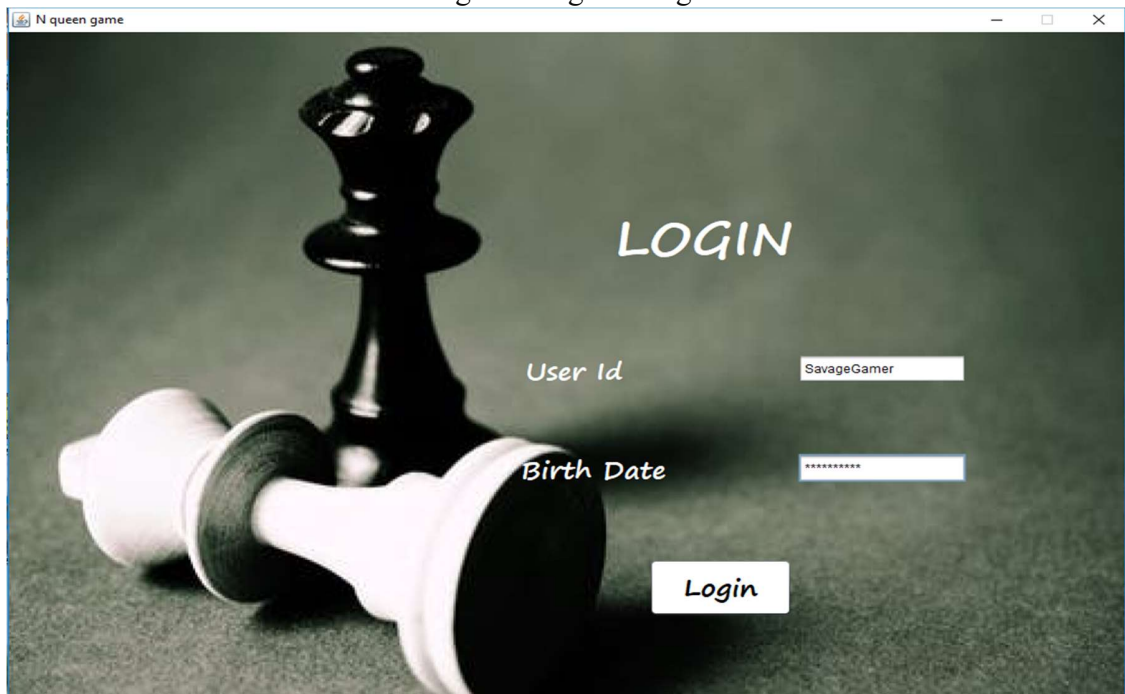
UserName: SavageGamer

Date Of Birth: 15/12/2000

City: Surat

Submit

Fig. 6.1 Register Page



The image shows a web browser window titled "N queen game". The background is a dark green chessboard with a black king and a white queen. The text "LOGIN" is displayed in a white, cursive font. Below this, there are two input fields: "User Id" with the value "SavageGamer" and "Birth Date" with the value "\*\*\*\*\*". A white "Login" button is located at the bottom right.

LOGIN

User Id SavageGamer

Birth Date \*\*\*\*\*

Login

Fig. 6.2 Login page

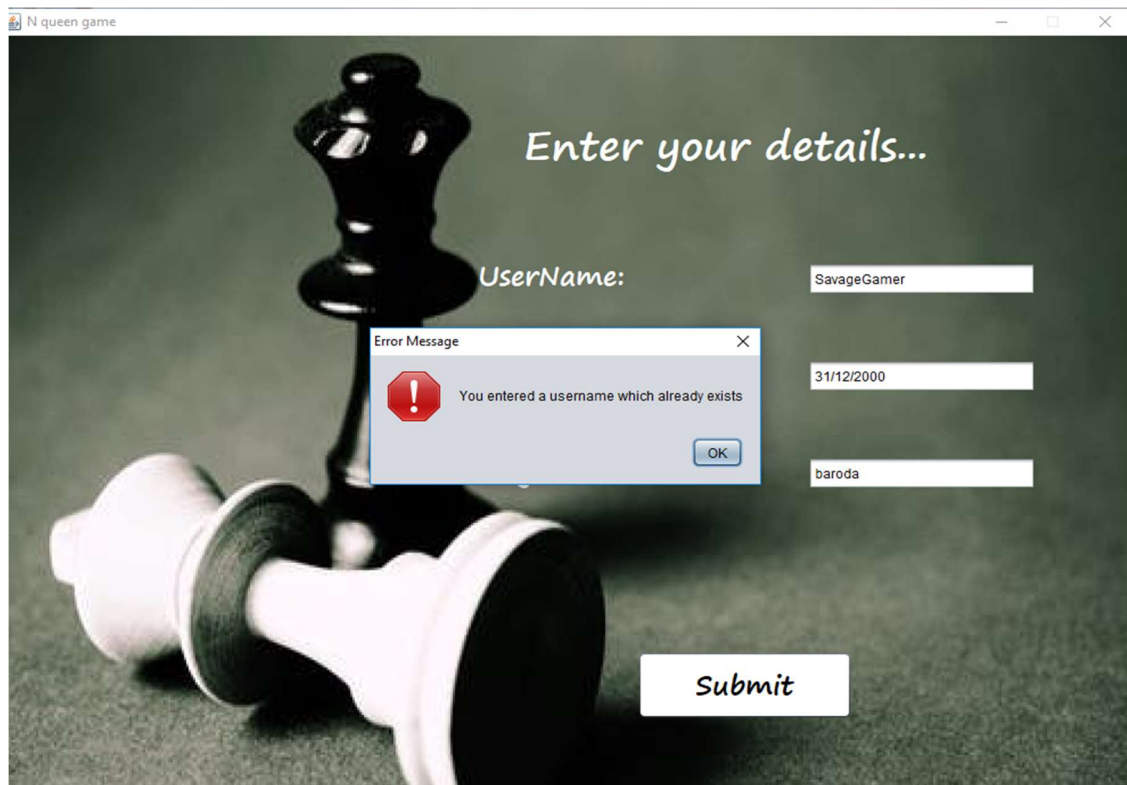


Fig. 6.3 Unique username required for Registration

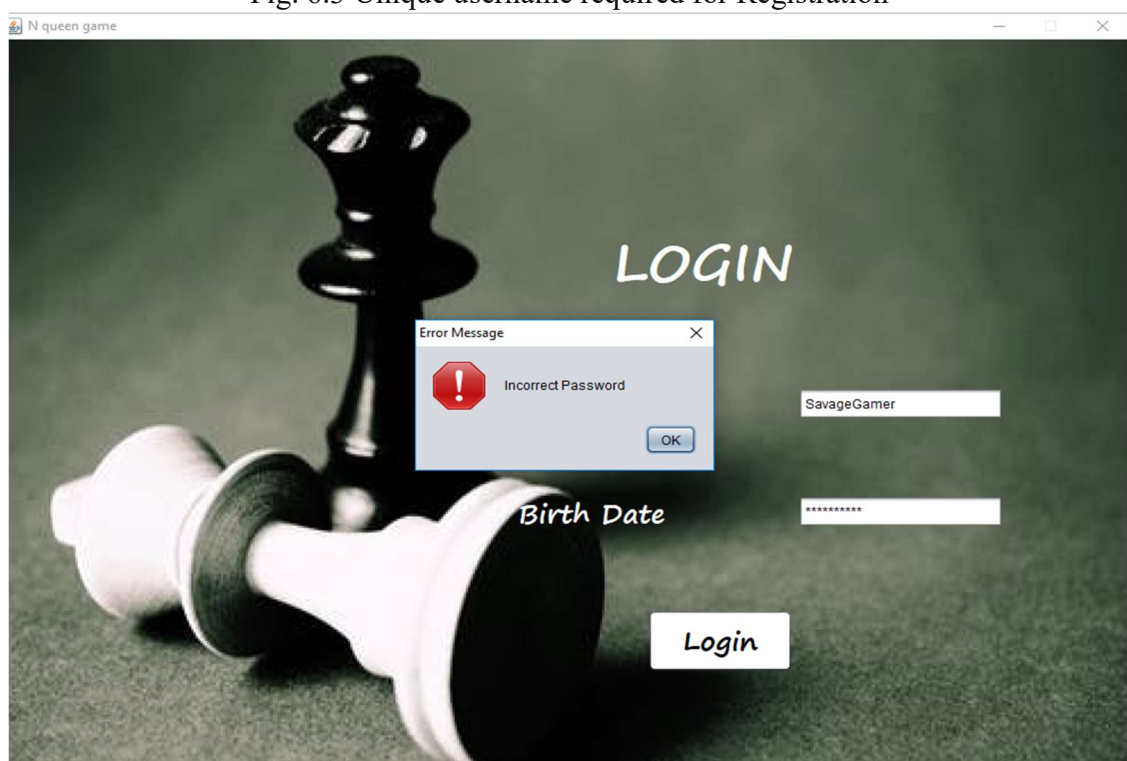


Fig. 6.4 Invalid password while logging in



Fig. 6.5 Invalid password while logging in

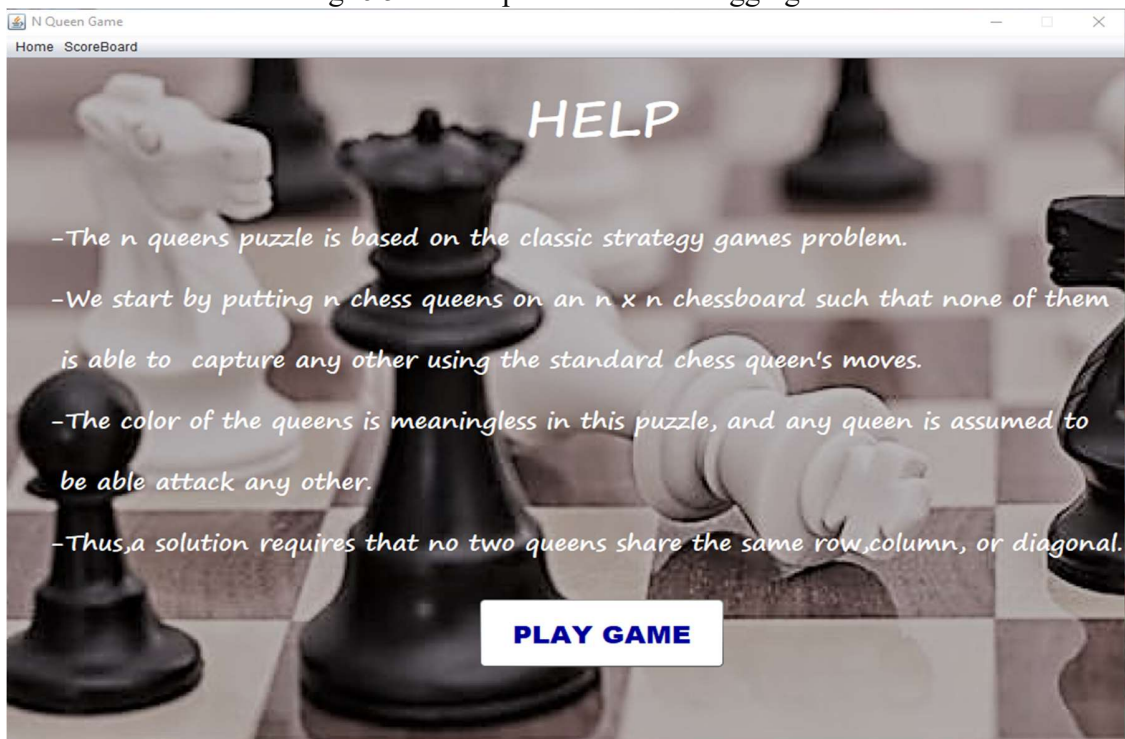


Fig. 6.6 Help Page

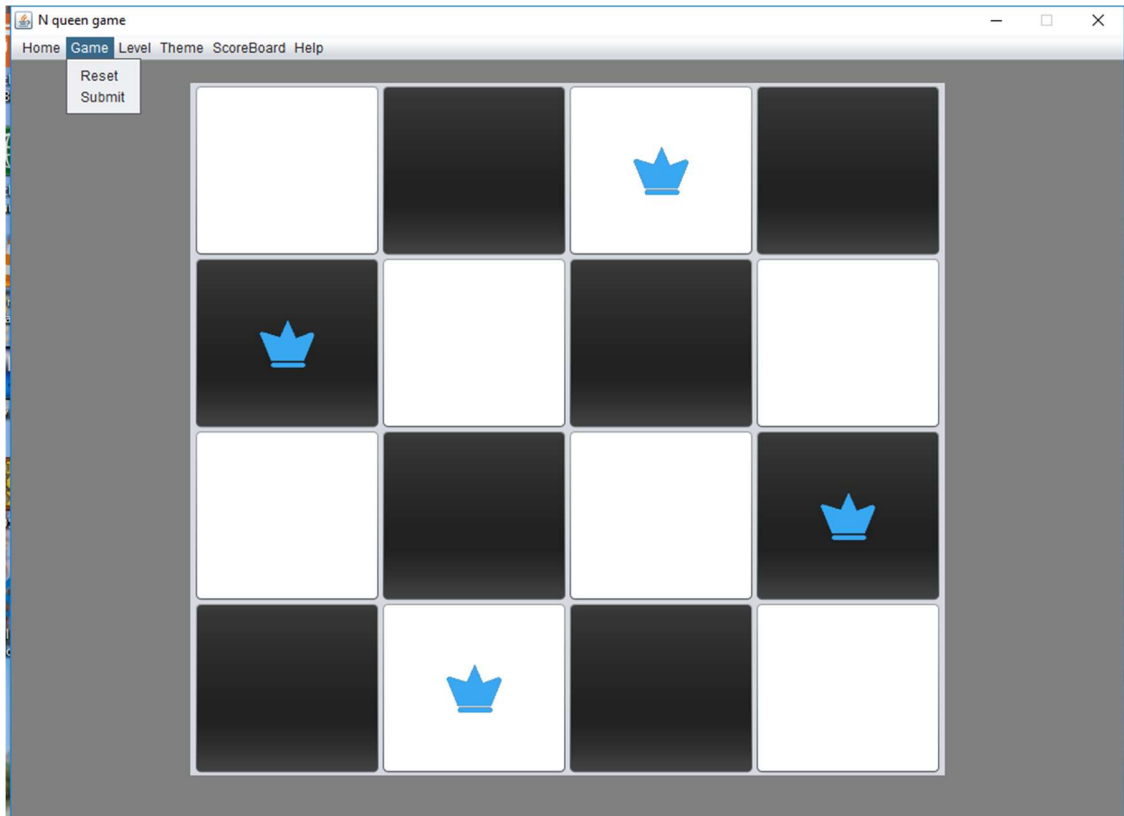


Fig. 6.7 Playing 4-queen on Board (Default)

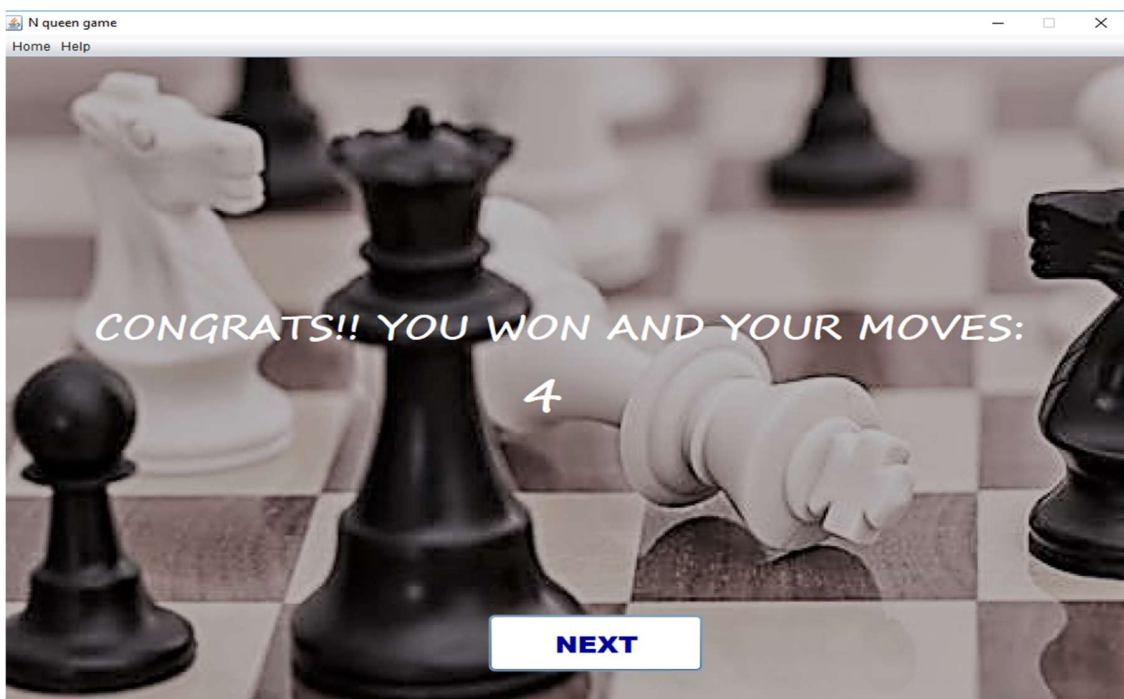


Fig. 6.8 Winning Result on Score Page

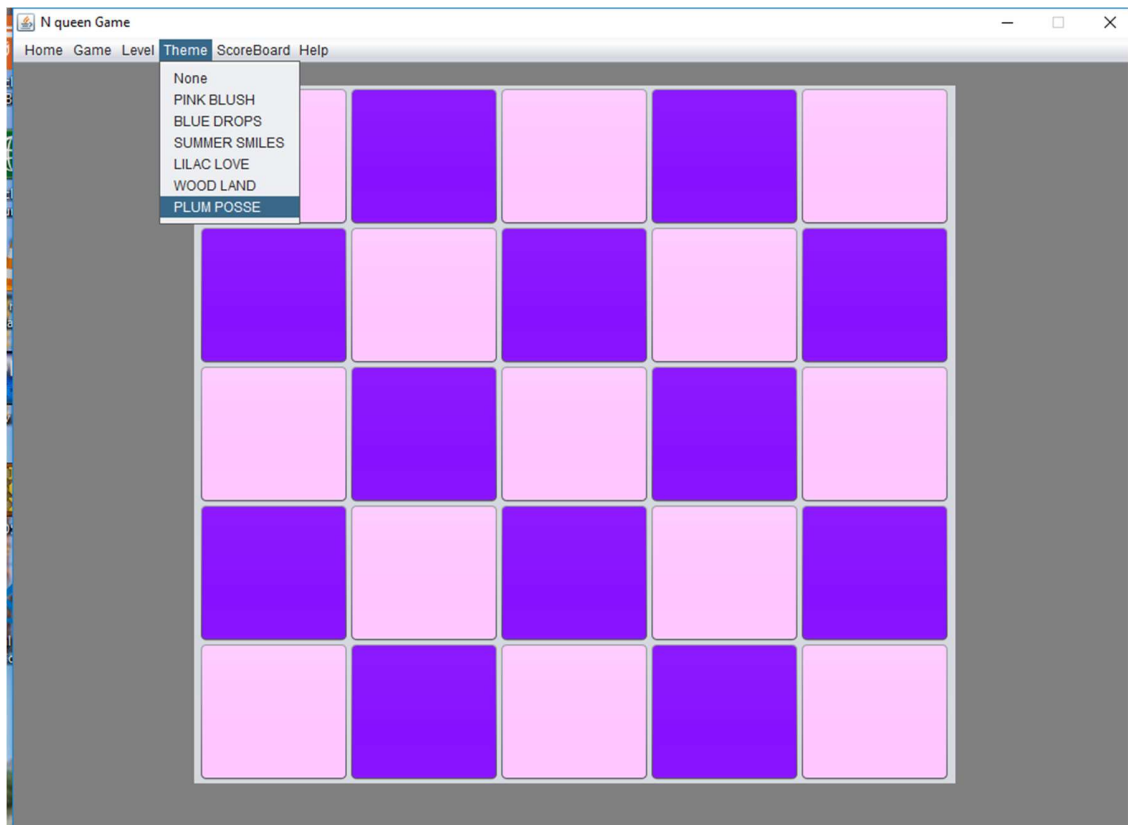


Fig. 6.9 Theme “PLUM POOSE” applied on board

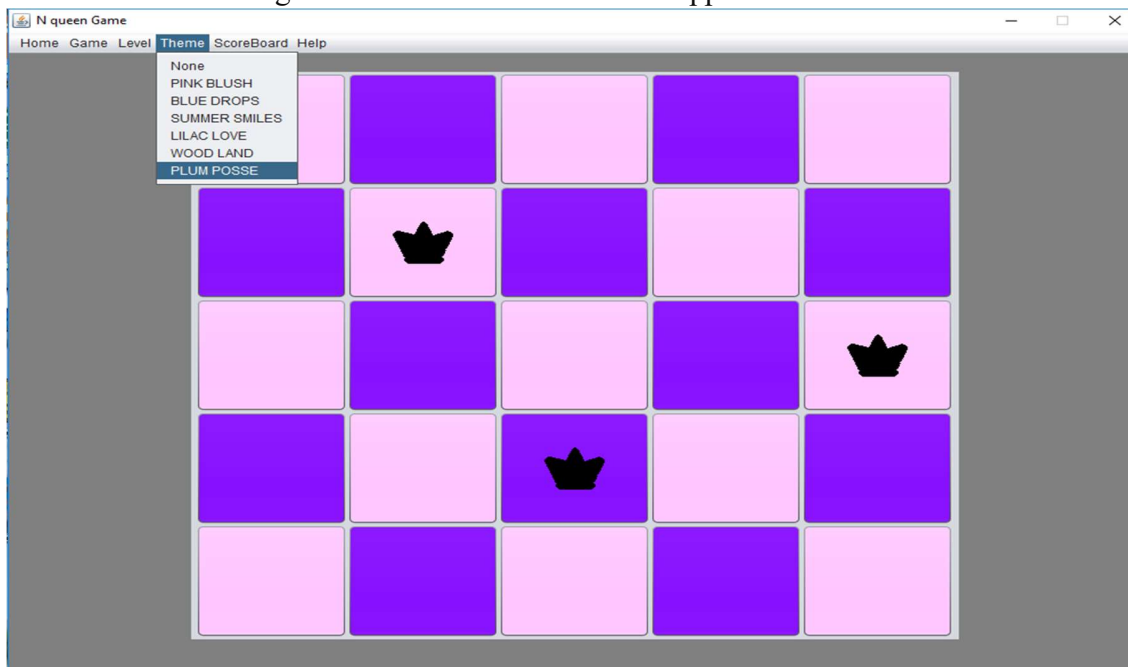


Fig. 6.10 Playing on Applied Theme Board



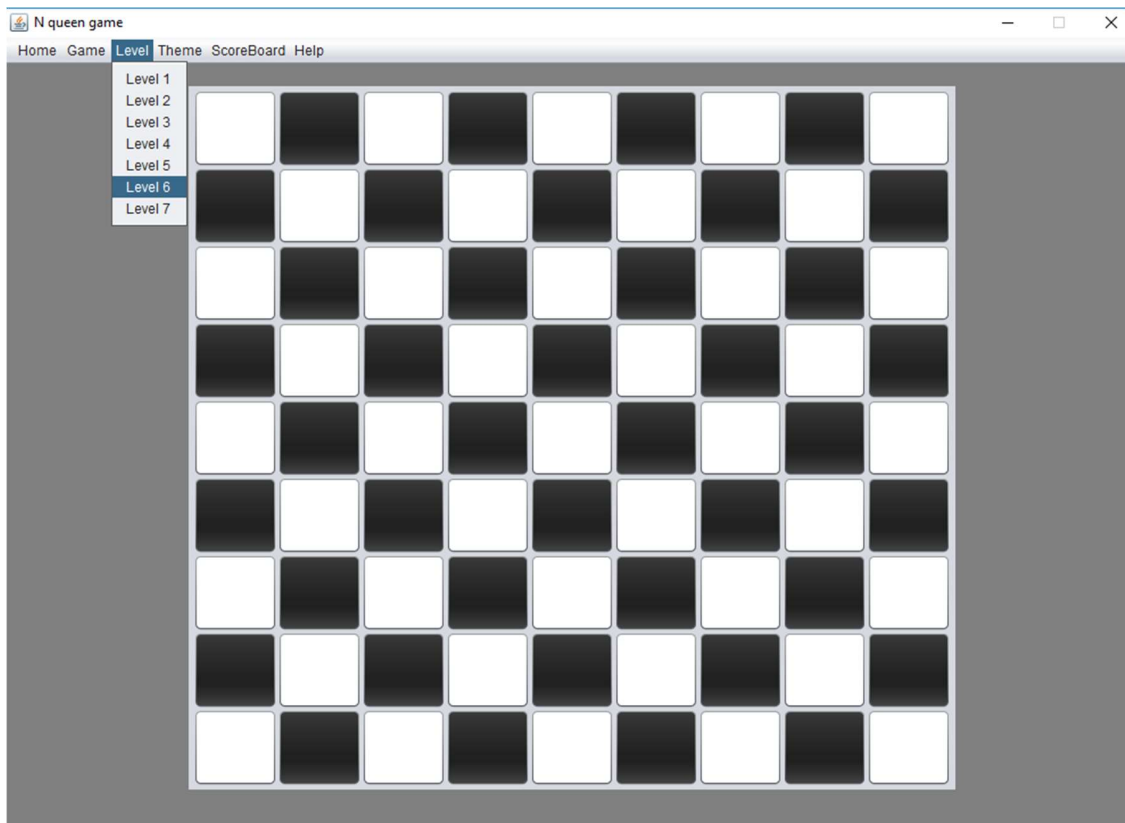


Fig. 6.11 Selecting Level

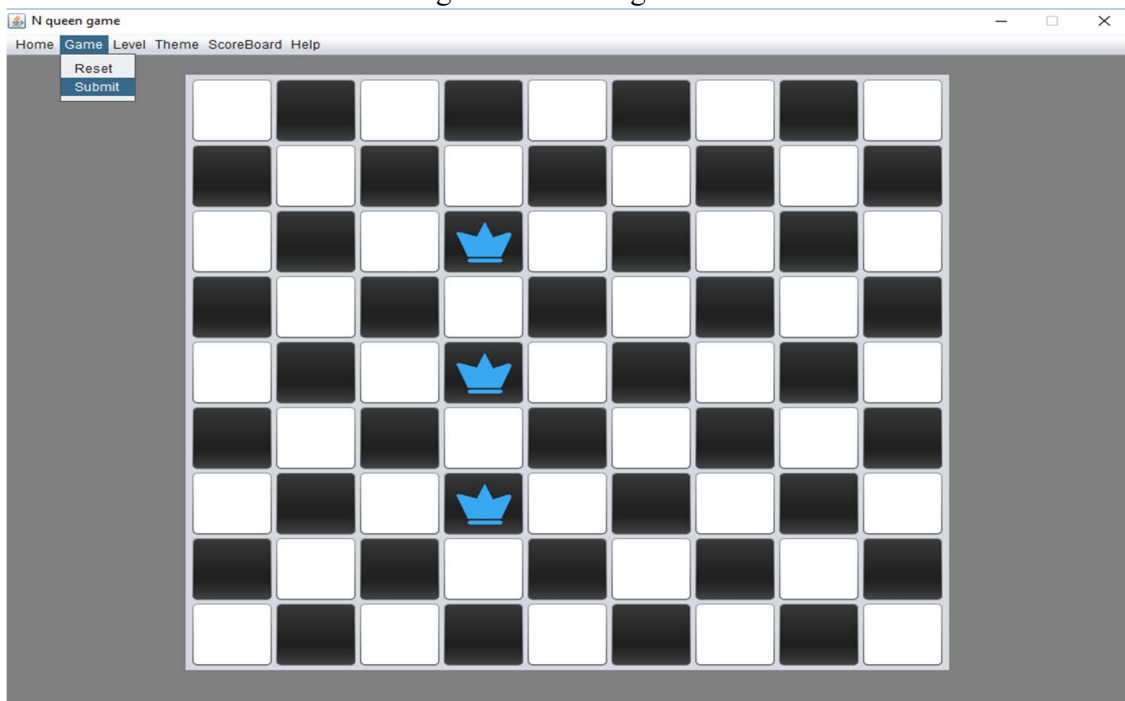


Fig. 6.12 Submitting wrong solution



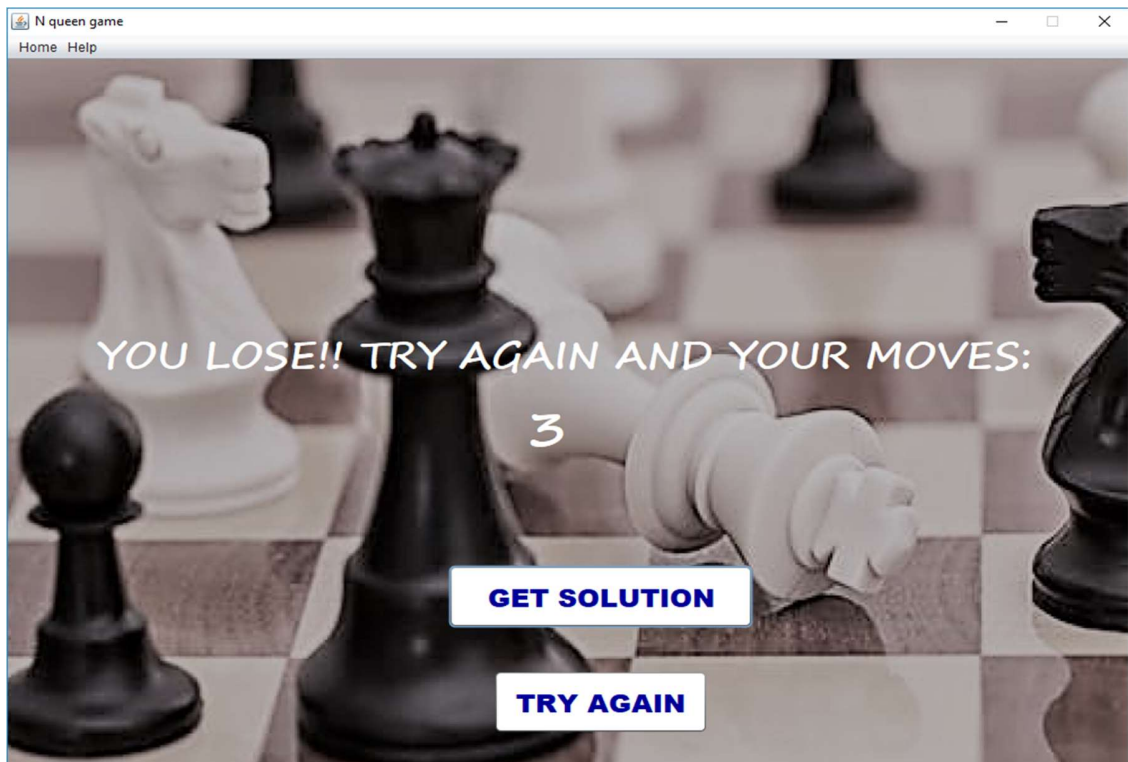


Fig. 6.13 Losing Result displayed on Score Page

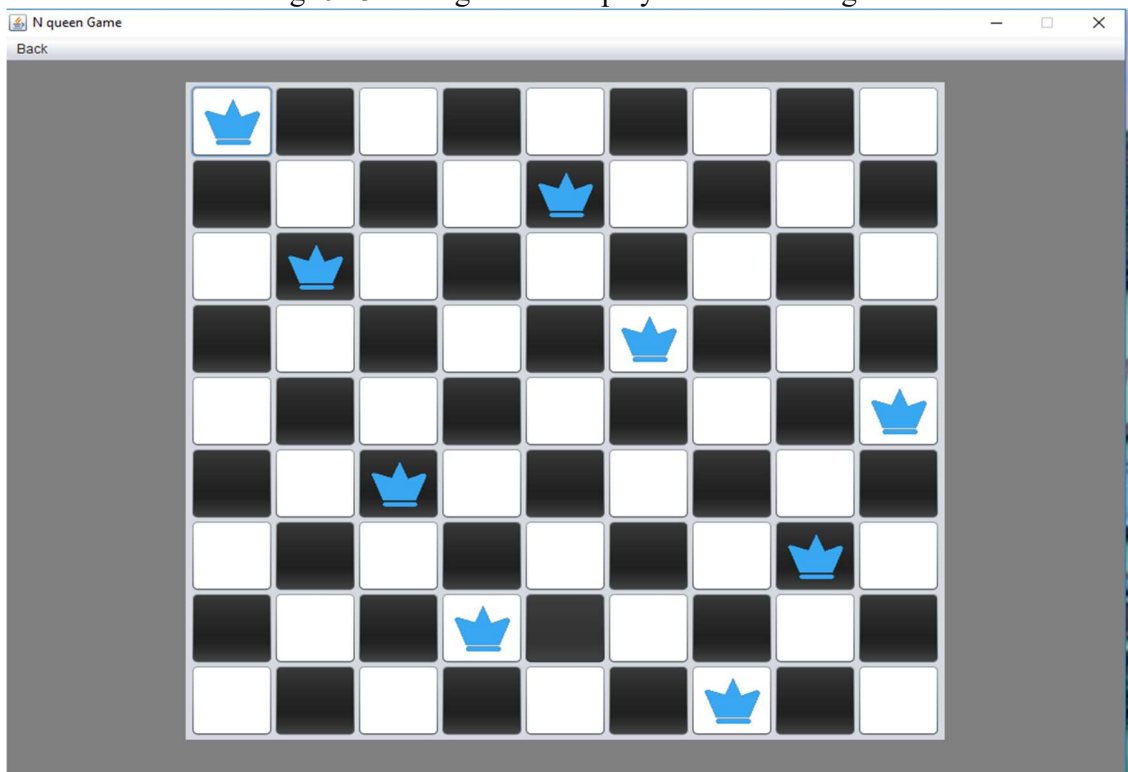


Fig. 6.14 Solution displayed on Board on selecting "Get Solution" button

## CH- 7

# CONCLUSION

---

To conclude, this project mainly deals with the N Queen problem which is demonstrated in a game. The project focuses on the brain development of the user by playing the game. The game has 7 levels varying from 4-queen game to 10 -queen game. The game is also provided with various themes which is applied on the board color and the queen color as well. If the user loses the game, for such cases the game is also provided with a view solution page which gives one of many solutions of the problem. Overall, this project solves N Queen puzzle effectively and is a fully-fledged Java application.

## **CH- 8**

# **LIMITATIONS AND FUTURE EXTENSIONS**

---

The limitations for the project are: The game displays only one solution on request of “Get solution”. The system does not check dynamically whether placed queen is safe to place on the board or not. We are not having functionality of getting best scores.

Further extensions possible are: The game can be made to play online between two players and the one who does it in lesser time wins the game. Presently for the game scores the moves are counted, instead we can calculate points based on moves and give the players find a hint at some minimum score. Other possible extension is to provide more levels and themes. Also, we can have life to play the game, once all lives are over the player can play the game after the lives are restored. The game can have functionality of getting best scores.

## CH- 9

# BIBLIOGRAPHY

---

- Tutorials from [www.youtube.com](http://www.youtube.com)
- Background images from [www.pexels.com](http://www.pexels.com)
- Algorithm reference from <https://www.geeksforgeeks.org/java-program-for-n-queen-problem-backtracking-3/amp/>