Pimpri Chinchwad College of Engineering
Computer Laboratory - IV

# Assignment No - B6

**AIM: -** 8-Queens Matrix is Stored using JSON/XML having first Queen placed, use back-tracking to place remaining Queens to generate final 8-queen's Matrix. Use suitable Software modeling , Design and testing methods. Justify the selection over other methods.

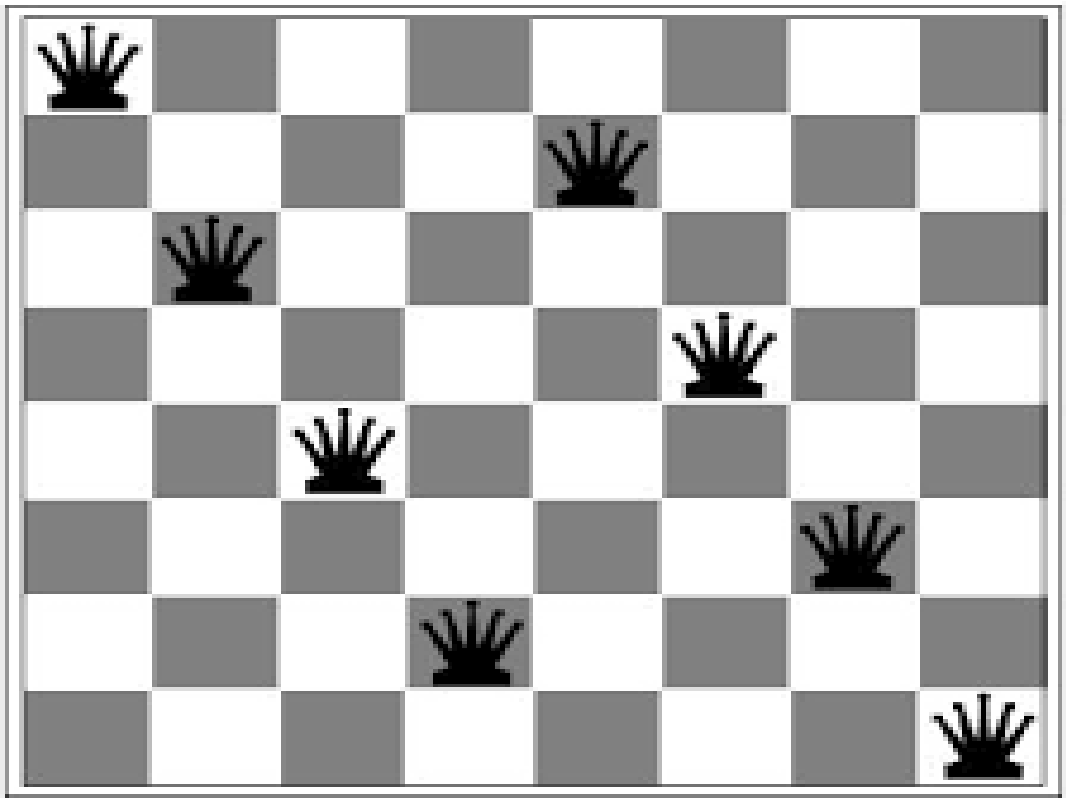
**THEORY:**

The **Eight queens puzzle** is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general ***n*-queens problem** of placing $n$ queens on an $n \times n$ chessboard, where solutions exist for all natural numbers $n$ with the exception of $n$=2 and $n$=3.

For example:

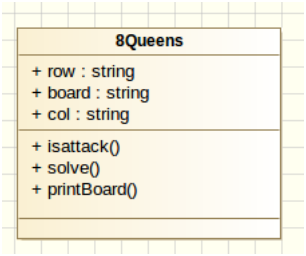
Q1 attacks some positions, therefore Q2 has to comply with these constraints and take its place, not directly attacked by Q1. Placing Q3 is harder, since we have to satisfy constraints of Q1 and Q2. Going the same way we may reach point, where the constraints make the placement of the next queen impossible. Therefore we need to relax the constraints and find new solution.

To do this we are going backwards and finding new admissible solution. To keep everything in order we keep the simple rule: last placed, first displaced. In other words if we place successfully queen on the ith column but cannot find solution for (i+1)th queen, then going backwards we will try to find other admissible solution for the ith queen first. This process is called backtracking.

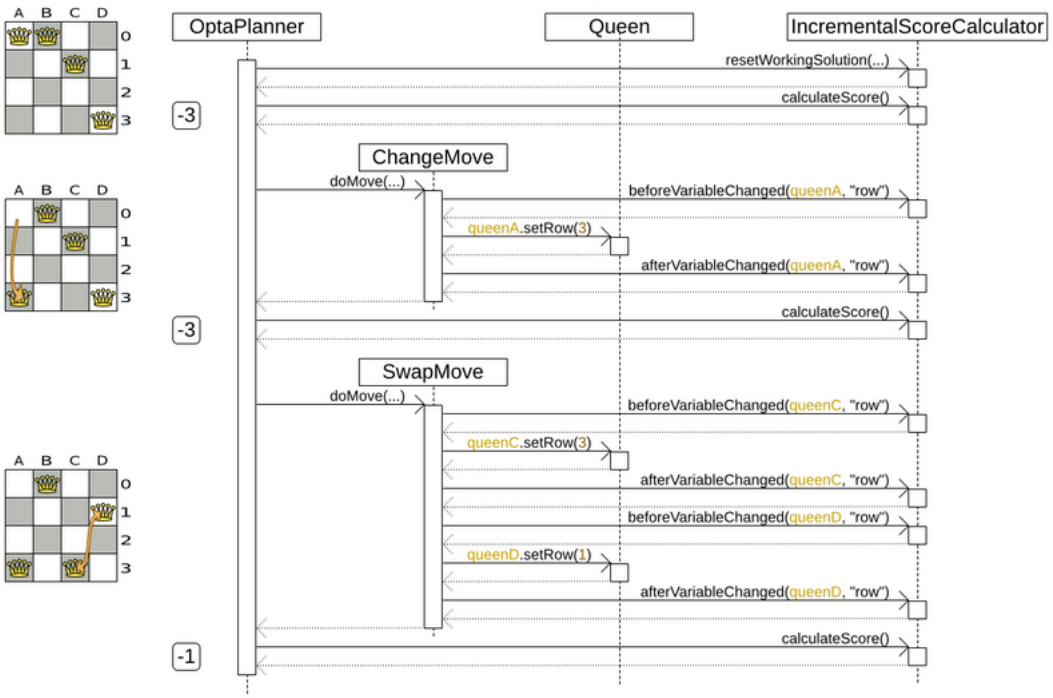


**UML diagrams :**

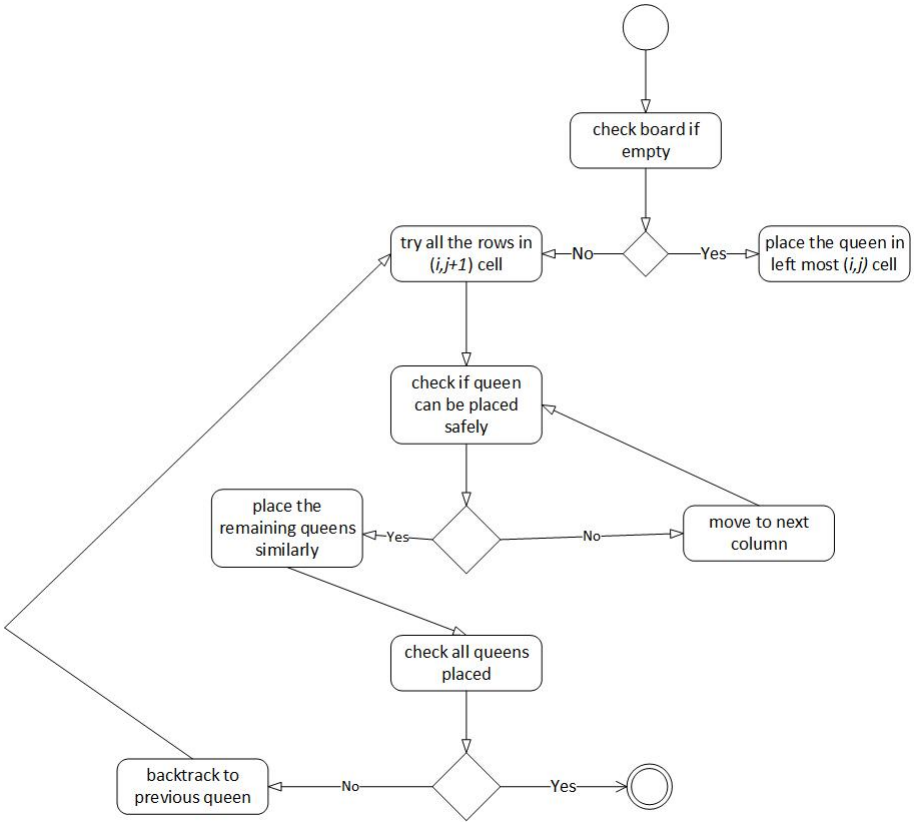
- **Class diagram :**




- **Sequence diagram :**




- **Activity diagram :**




**Mathematical Model:**

Consider a set S consisting of all the elements related to a program.The mathematical model is given as below,

**S= {s, e, X, Y, Fme, DD, NDD, Mem shared}**

**Where,**

**s = Initial State**

**e = End State**

**X = Input. Here it is map size, start point**

**Y = Output. Here output is time required to generate route, actual route and map.**

**Fme = Algorithm/Function used in program.**

For e.g. placequeen() etc.

**DD=Deterministic Data**

**NDD=Non deterministic Data**

**Mem shared=Memory shared by processor.**

**Algorithm:**

**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 the solution. If we do not find such a row due to clashes then we backtrack and return false.

**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 placing queen here leads to a solution.**
**b. If placing queen in [row, column] leads to a solution then return true.**
**c. If placing queen doesn't lead to a solution then umark this [row, Column] (Backtrack) and go to step (a) to try other rows.**
**4. If all rows have been tried and nothing worked, return false to trigger backtracking.**

**Testing:**

- **Positive Testing:**
  Positive Testing is testing process where the system validated against the valid input data. In this testing tester always check for only valid set of values and check if a application behaves as expected with its expected inputs. The main intention of this testing is to check whether software application not showing error when not supposed to and showing error when supposed to.

  The system will show the appropriate output when it is tested against correct mathematical values. For example, for trigonometric functions, the system will show appropriate output in floating point numbers. Also for basic mathematical operations like addition, subtraction, it will show appropriate output.

| Sr. No. | Test Condition | Steps to be executed | Expected Result | Actual Result |
|---|---|---|---|---|
| 1. | First queen is placed at appropriate position | Press Enter | First queen is placed successfully | Same as Expected |
| 2. | Accordingly, remaining queens are placed by avoiding conflicts | Press Enter | All queens are placed at appropriate position | Same as Expected |

- **Negative Testing:**
  Negative Testing is testing process where the system validated against the invalid input data. A negative test checks if a application behaves as expected with its negative inputs. The main intention of this testing is to check whether software application not showing error when supposed to and showing error when not supposed to.

  While performing the negative testing, the system will show incorrect output while performing the basic arithmetic operations if the user enters only a single

number and performs addition without entering the second number. The system will give an error when the user attempts a divide by zero operation.

| Sr. No. | Test Condition | Steps to be executed | Expected Result | Actual Result |
|---|---|---|---|---|
| 1. | Position given for placing first queen is between 0 to 7 or not | Press Enter | Yes | Yes |
| 3. | Remaining queens are placed with conflicts | Press Enter | All queens are not placed | Same as Expected Result |

- **Test cases using black box and white box testing:**

1. Test case:Part 1

| Test Case ID | Objective | Test Description | Test Data | Expected output | Actual output | Result | Method |
|---|---|---|---|---|---|---|---|
| 1 | Check valid column for placing first queen | We check whether user enters a valid row number between 0-7. | column number | If 0<=column number <=7 then place queen in first column | Queen placed in first row number and given column | Pass | Whitebox condition coverage |
| 2 | Check if the queen can be placed in next row and possible column number | Check if queen is being placed in same row of any of the already placed queens | Row number, column number where queen has to be placed. | If queen is being attacked by any of the already placed queens on the board, backtrack and check for next position. | If it passes the test, queen is placed, else other positions are evaluated by backtracking. | Pass | Whitebox condition coverage |
| 3 | Check if the queen can be placed in next row and possible column number | Check if queen is being placed in same column of any of the already placed queens | Row number, column number where queen has to be placed. | If queen is being attacked by any of the already placed queens on the board, backtrack and check for next position. | If it passes the test, queen is placed, else other positions are evaluated by backtracking. | Pass | Whitebox condition coverage |
| 4 | Check if the queen can be placed in next row and possible column number | Check if queen is being placed diagonally of any of the already placed queens | Row number, column number where queen has to be placed. | If queen is being attacked by any of the already placed queens on the board, backtrack and check for next position. | If it passes the test, queen is placed, else other positions are evaluated by backtracking. | Pass | Whitebox condition coverage |

1. Test case: Part 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | Check working of solveBoard() method | Checks if queen is being attacked for current position by calling isAttack() and places the queen if not being attacked. | Row number, column number. | Place the queen at correct position | Place the queen at correct position | Pass | Whitebox function coverage. |
| 6 | Check working of isAttack() method | Check if queen is being attacked by any of the already placed queens | Current board status, row and column number | Queen placed at position where it is not attacked | Queen placed at a safe position | Pass | Whitebox function coverage. |
| 7 | Check working of printBoard() method | Dump the board configuration in json file and print the board configuration | Board configuration | Data returns successfully to the json file and data is written in json output file. | Data printed and dumped into json file. | Pass | Whitebox function coverage. |
| 8 | Check JSON input. | Check if correct column number is given by JSON input file. | column number | If 0<=column number <=7 then board configuration is displayed else error message is displayed. | If 0<=column number <=7 then print board configuration else print "invalid json input". | Pass | Black box positive testing and negative testing. |

## CONCLUSION:

Hence we implemented 8 queen problem using backtracking in python and studied different design and testing methods.

| Roll No. | Name of Student | Date of Performance | Date of Submission | Sign. |
|---|---|---|---|---|
| 302 | Abhinav Bakshi | 21/03/16 | 30/03/16 | |

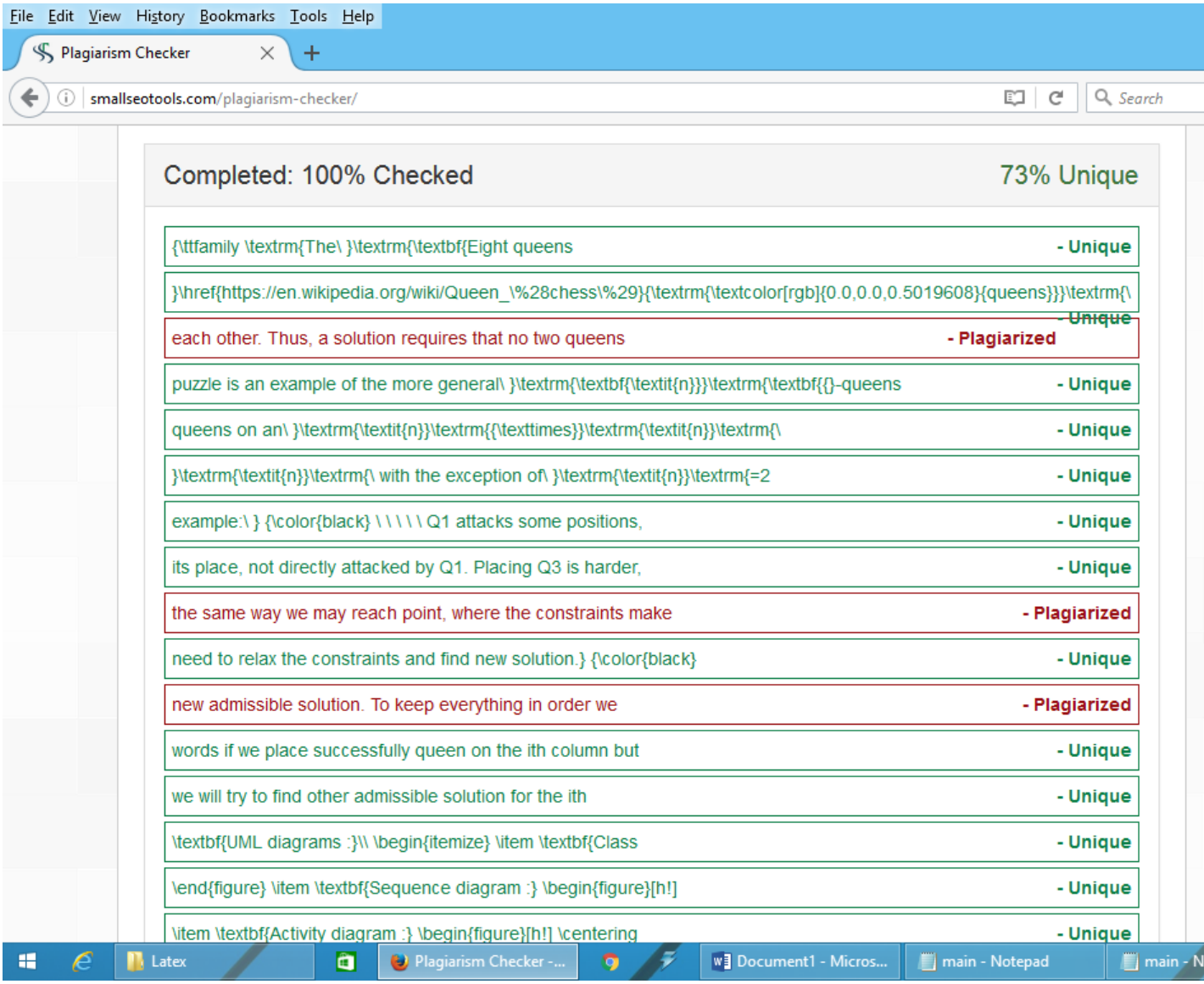



Figure 1: Plagiarism Report