Name: VISHALKUMAR PATEL (#1054821)

KISHAN GABANI (#1116169)

SNAKER LAD (#1131716)

Course: CSCI 621 (Programming Languages)

Instructor: Dr. Wenjia Li

Project: N-Qeen Riddle

Submission Date: 05-10-2016

N-Queen Game

The eight rulers riddle

Patel Vishalkumar B (1054821)
CSCI-621 Programming Language
NYIT
New York, USA
vpatel80@nyit.edu

Kishan Gabani (1116169)
CSCI-621 Programming Language
NYIT
New York, USA
kgabani@nyit.edu

Sanket Lad (1131716)
CSCI-621 Programming Language
NYIT
New York, USA
Slad01@nyit.edu

***Abstract* - The eight rulers riddle is the issue of putting eight chess rulers on 8×8 chessboard so that no two rulers debilitate each other.**

Keywords—8queen; nqueen; riddle; game; chessboard;

I. INTRODUCTION

The eight rulers riddle is the issue of putting eight chess rulers on a 8×8 chessboard so that no two rulers debilitate each other. Therefore, an answer requires that no two rulers have the same line, section, or slanting. The eight rulers riddle is a case of the more broad n-rulers issue of setting n rulers on a n×n chessboard, where arrangements exist for every single regular number n except for n=2 and n=3.

This riddle was published as a 8x8 matrics, but we are making it to be work as a dynamic. Where person has to enter the dimension and the queen numbers, and have to arrange that queens.

II. HISTORY

Chess arranger Max Bezzel[1] distributed the eight rulers riddle in 1848. Franz Nauck[2] distributed the principal arrangements in 1850. Nauck likewise extended the riddle to the n-rulers issue, with n rulers on a chessboard of n × n squares. And we are developing this riddle using java to solve n problems with n x n matrics. From that point forward, numerous mathematicians, including Carl Friedrich Gauss[3], have taken a shot at both the eight rulers riddle and it's summed up n-rulers form. In 1874, S. Gunther proposed a technique utilizing determinants to discover solutions. J.W.L. Glaisher refined Gunther's methodology.

In 1972 Edsger Dijkstra[4] utilized this issue to outline the force of what he called organized programming. He distributed a depth-first backtracking algorithm.

III. SOLUTION

This issue can be very computationally costly as there are 4,426,165,368 (i.e., 64C8) conceivable game plans of eight rulers on a 8×8 board, yet just 92 arrangements. It is possible to use shortcuts which reduce computational requirements or rules of thumb that avoids brute-force computational techniques For instance, just by applying a basic decide that obliges every ruler to a solitary segment (or line), however still thought to be beast drive, it is conceivable to diminish the quantity of potential outcomes to only 16,777,216 (that is, 88) conceivable mixes. Creating stages further lessens the potential outcomes to only 40,320 (that is, 8!), which are then checked for further.

Martin Richards distributed a project to number answers for the n-rulers issue utilizing bitwise operations.

As mentioned above, the eight rulers riddle has 92 particular arrangements. On the off chance that arrangements that contrast just by symmetry operations (revolutions and reflections) of the board are considered one, the riddle has 12 crucial arrangements.

A basic arrangement ordinarily has eight variations (counting its unique structure) acquired by turning 90, 180, or 270 degrees and afterward mirroring

Each of the four rotational variations in a mirror in a settled position. In any case, ought to an answer be equal to its own particular 90-degree pivot that essential arrangement will have just two variations (itself and its appearance). Should an answer be proportional to its own particular 180-degree turn (however not to its 90 degree pivot) it will have four variations (itself and its appearance, its 90 degree revolution and the impression of that). It is unrealistic for an answer for be proportional to its own appearance (aside from at n=1) in light of the fact that that would require two rulers to be confronting each other. (For n-ruler issue's answer for be comparable to its own mirror-picture arrangement, the arrangement should be symmetrical by the focal point of the board either on a level plane or vertically.

At that point, two rulers would be confronting each other, making it not an answer.) Of the 12 essential answers for the issue with eight rulers on a 8x8 board, precisely one is equivalent to its own particular 180 degree turn, and none are equivalent to their 90 degree revolution, along these lines the quantity of unmistakable arrangements is 11*8 + 1*4 = 92 (where the 8 is gotten from four 90-degree rotational positions

and their appearance, and the 4 is gotten from two 180-degree rotational positions and their appearance).

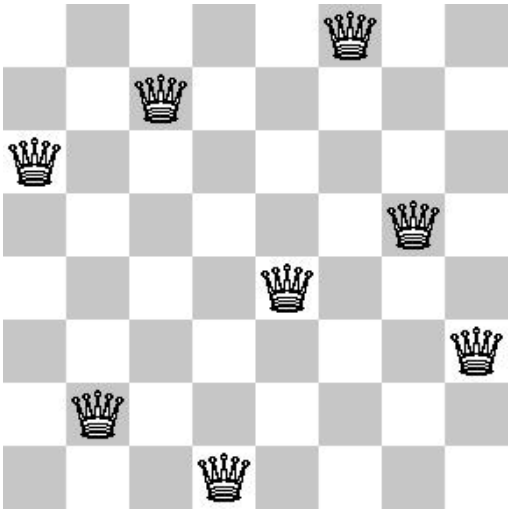


Figure 1.1

To the privilege is a table of the quantity of answers for various estimated n x n sheets. For every size board I've demonstrated the quantity of aggregate arrangements, furthermore the quantity of particular sorts of arrangements (extraordinary before pivots and reflections).

Inconsequentially, there is stand out answer for a 1 x 1 board, and it's not hard to see that there are no conceivable answers for a 2 x 2 or 3 x 3 measured board. It's fascinating that, whilst there are 10 answers for a 5 x 5 board, the quantity of arrangements drops to only 4 arrangements on a 6 x 6 board. There is (at present) no known equation for deciding the number conceivable answers for a n x n board, and a web look uncovers that the most astounding figured board size to-date is 26 x 26.

Board	Total Solutions	Unique Solutions
1 x 1	1	1
2 x 2	0	0
3 x 3	0	0
4 x 4	2	1
5 x 5	10	2
6 x 6	4	1
7 x 7	40	6
8 x 8	92	12
9 x 9	352	46
10 x 10	724	92
11 x 11	2,680	341
12 x 12	14,200	1,787
13 x 13	73,712	9,233
14 x 14	365,596	45,752
15 x 15	2,279,184	285,053
16 x 16	14,772,512	1,846,955
17 x 17	95,815,104	11,977,939
18 x 18	666,090,624	83,263,591
19 x 19	4,968,057,848	621,012,754
20 x 20	39,029,188,884	4,878,666,808
21 x 21	314,666,222,712	39,333,324,973
22 x 22	2,691,008,701,644	336,376,244,042
23 x 23	24,233,937,684,440	3,029,242,658,210
24 x 24	227,514,171,973,736	28,439,272,956,934
25 x 25	2,207,893,435,808,350	275,986,683,743,434
26 x 26	22,317,699,616,364,000	2,789,712,466,510,280

Table 1.1

Where the below table shows the chances of chessboard can be up to 26x26, and the solutions can be in creases till 22,317,699,616,364,000 and according to the board the unique solutions can be also increased till 2,789,712,466,510,280, where for 2x2 metrics the solution is not possible not even unique solutions.

IV. Approach

In past this was constrained was only 8 ruler issue, where we took it to n!, client can make his/her chess board with a craved quality.

This riddle is for 8x8 matric, we are building up a JAVA application in which we can choose the estimation of n, attempting to make it dynamic. Where n can be up to ser or player and from the estimation of n, the board will be made in jpanel. We are additionally giving an answer catch which demonstrates the all conceivable positions to the player, so player can take an assistance of PC to get the answer.

V. Used Algorithm

There are numerous conceivable calculations that can be utilized to discover answers for the eight ruler's issue, and a littler subset of calculations that can be utilized to identify every single conceivable arrangement.

The primary conceivable system is immaculate savage power; indiscriminately attempting the eight rulers in each conceivable area. This is a truly dumb thought, however would figure every single conceivable arrangement (We would need to look at 64!/56! = 178,462,987,637,760 conceivable positions, which is like impossible)

A more effective calculation (which can be actualized recursively or not) is to begin in the main section in the upper left, then it then places a ruler in the second segment and moves it until it finds a spot where it can't be hit by the ruler in the principal segment. It then places a ruler in the third segment and moves it until it can't be hit by both of the initial two rulers. In the event that the present section is the primary segment and its ruler is being gotten off the board then all conceivable arrangements have been inspected, the sum total of what arrangements have been found, and the calculation ends.

The thought is to place rulers one by one in various sections, beginning from the furthest left segment. When we put a ruler in a section, we check for conflicts with officially put rulers. In the present section, in the event that we discover a line for which there is no conflict, we check this line and segment as a feature of the arrangement. On the off chance that we don't discover such a line because of conflicts then we backtrack and return false. Showed as below algorithm.

1) Start in the furthest left segment
2) If all rulers are set
return true

3) Try all lines in the present segment. Do taking after for each attempted column.
a) If the ruler can be set securely in this line then stamp this [row, column] as a major aspect of the arrangement and recursively check if putting ruler here prompts an answer.
b) If setting ruler in [row, column] prompts an answer then return true.
c) If setting ruler doesn't prompt an answer then unmark this [row, column] (Backtrack) and go to step (a) to attempt different lines.

3) If the sum total of what lines have been attempted and nothing worked, return false to trigger backtracking.

Here Jeff Somer's calculation is utilized to take care of this issue! his calculation for N-Queen issue is considered as the speediest one, he utilized the idea of back following to explain this.

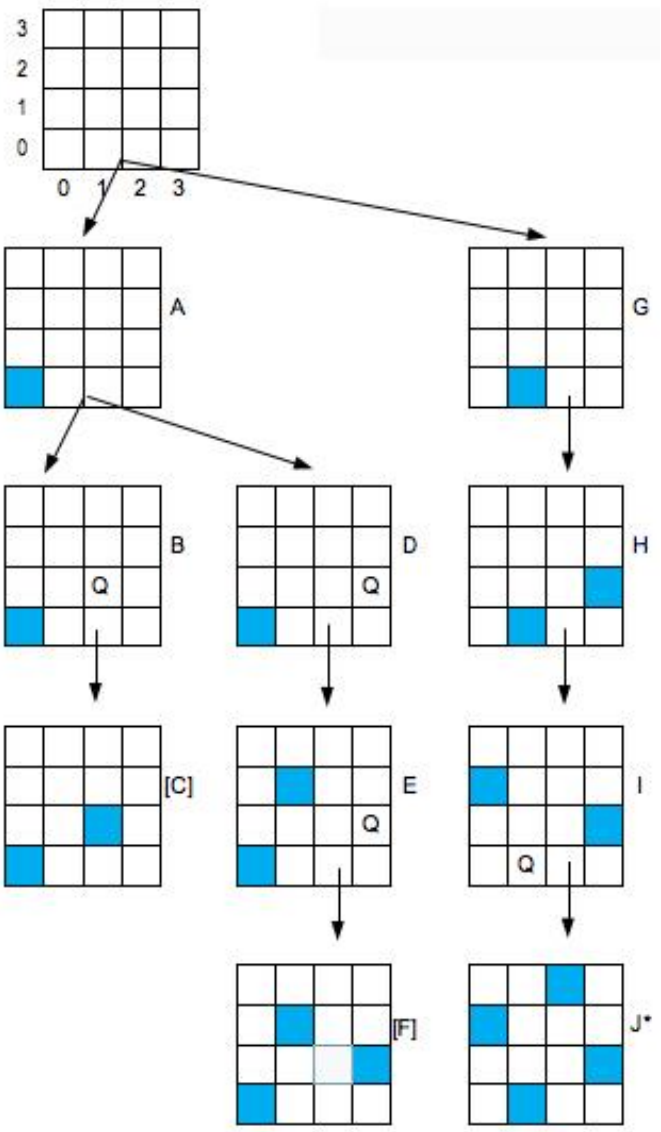
before hand the world speediest calculation used to explain this was Sylvain Pion and Joel-Yann Fourre. Be that as it may, his calculation was restricted to only 23 rulers and was utilizing bit field control backtracking

VI. How it Works

All answers for the n-rulers issue can in this manner be spoken to as ntuples (c1, c2, . . . cn), where ci is segment on which ruler i is put, and no two ciscan be the same.

Every ruler is doled out an alternate segment, so the main thing left to check is whether two rulers are in the same corner to corner. The lines are currently numbered from base to best and the segments are numbered from left to right Hence the base left corner has the directions (0,0) and the upper left corner (n-1,0).

With this numeration, two rulers with the directions (i, ci) and (j, cj) are in the same climbing askew if j −i == cjci. They are in the same diving askew if j −i == ci − cj. Hence, keeping in mind the end goal to check whether two rulers are in the same inclining, regardless of in which course, it is adequate to check whether |j − i| == |cj − ci| or not.



VII. Goal to Design

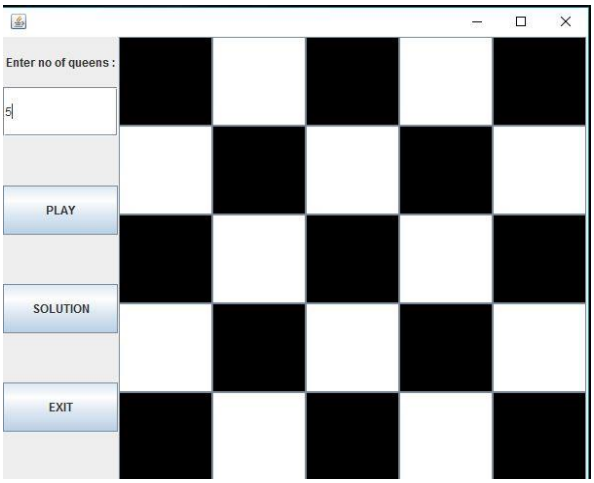


Figure 1.2

Our objective was to plan a design utilizing jpanel as displayed in above figure 1.2, that's a part of java as appeared with a textbox to enter the digit of ruler required with a Grid format to demonstrate all the positions player can put their ruler and attempt to get the answer.

VIII. SCREENSHOTS OF ACTUAL DESIGN

As appeared underneath figure 1.3 how an utilized can begin setting ruler on by one on the areas. Three rulers have been demonstrated the set

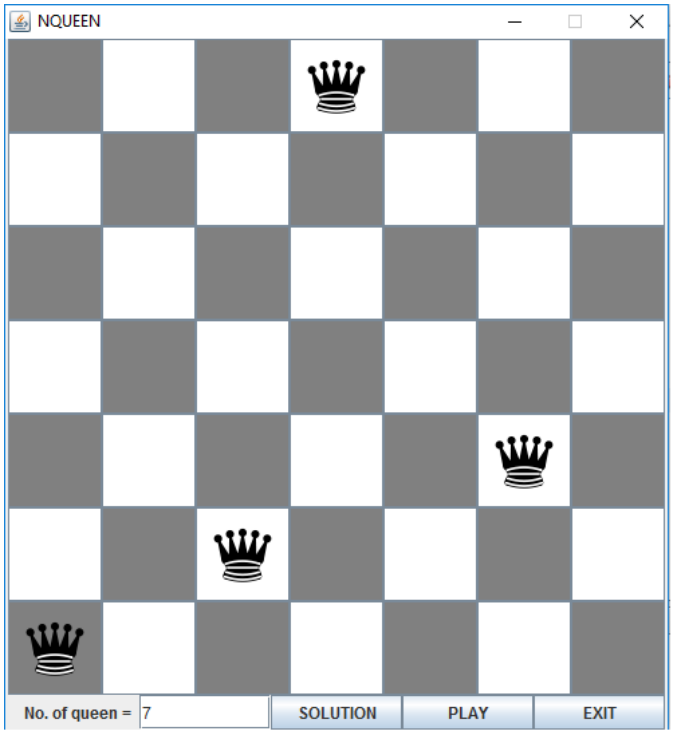


Figure 1.3

As appeared beneath in figure 1.4 subsequent to giving contribution as a 6 how the 6x6 chessboard is produced with 6 lines and 6 segments. Also, how the 6 rulers are set without in a same line or segments or get a corner to corner and specifically tapping the arrangement catch, ser can get the immediate yield by as of now 6 rulers set on an exceptional spot.

As appeared in figure 1.5 how the chessboard don't permit to put no more ruler in the event that you tries to place more than 7 ruler on the areas.

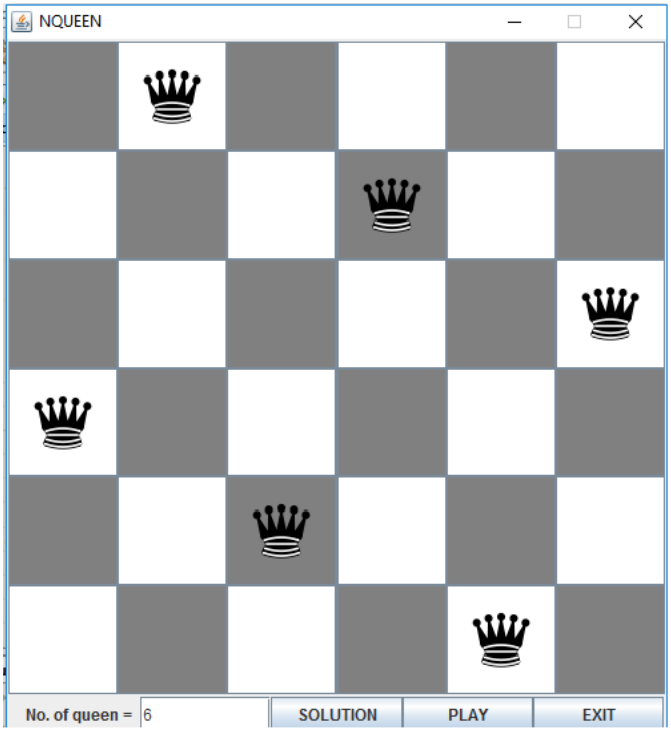


Figure 1.4

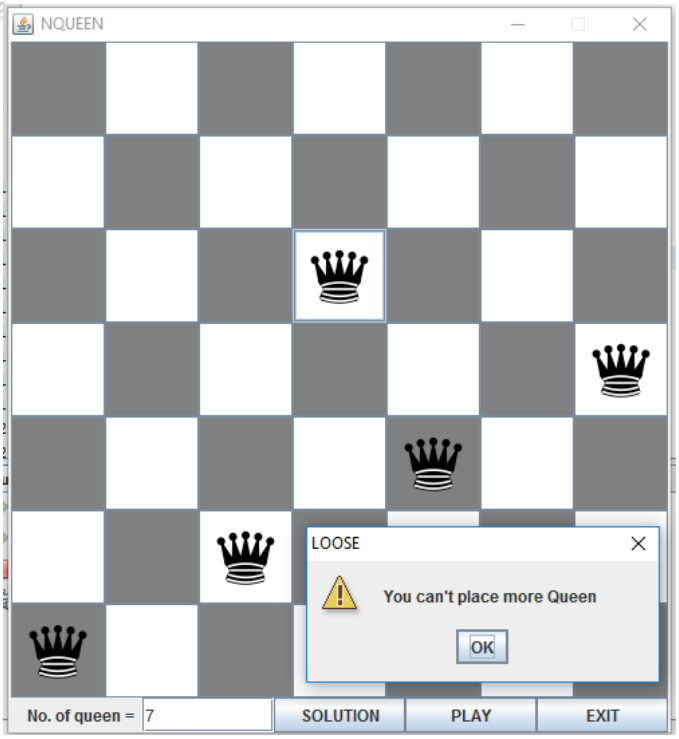


Figure 1.5

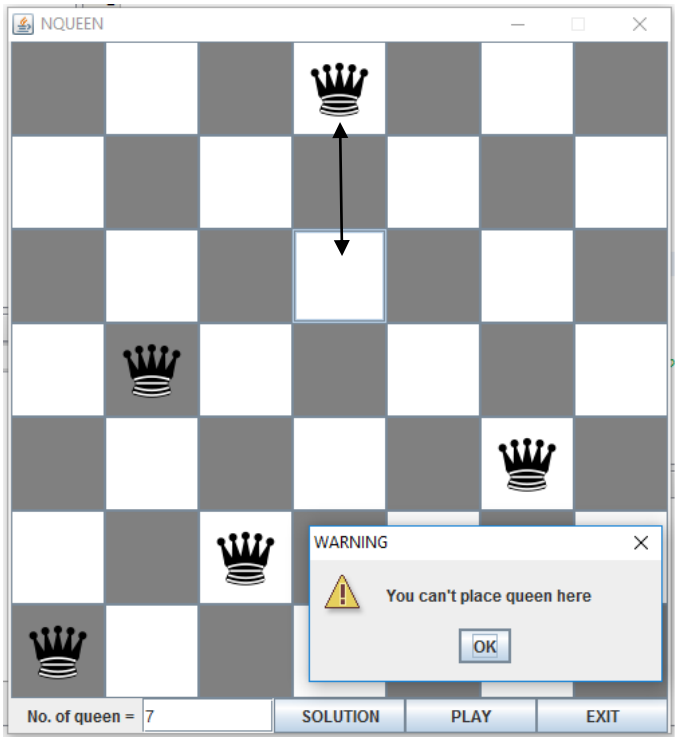


Figure 1.6

On the above figure 1.6 on ruler is set at row1 and segment 4, however another ruler is attempting to be set at row3 and column3 at the same time, that makes to be 2 rulers in one section, so it is ceased by an exchange box says that another ruler can't be set on this position

IX. Code

I. ButtonPanel.java :

```
package project;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class ButtonPanel extends JPanel {

    private JButton solBtn;
    private JButton playBtn;
    private JButton exitBtn;
    private NQueen nq;
    private JLabel no;
    private JTextField tf;
    Queen q;

    ButtonPanel(Queen q) {
        this.q = q;
        solBtn = new JButton("SOLUTION");
        playBtn = new JButton("PLAY");
        exitBtn = new JButton("EXIT");
        tf = new JTextField(2);
        no = new JLabel("    No. of queen = ");
        setLayout(new GridLayout(1, 5));
        add(no);
        add(tf);
        add(solBtn);
        add(playBtn);
        add(exitBtn);

        solBtn.addActionListener(new mybuttonlistener());
        playBtn.addActionListener(new mybuttonlistener());
        exitBtn.addActionListener(new mybuttonlistener());

    }

    class mybuttonlistener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {

            if (e.getSource() == solBtn) {
                System.out.println("Performed");
                String str1 = tf.getText();
                int int1 = Integer.parseInt(str1);
                q.solution = true;
                q.ncall(int1);
                q.dispose();
            } else if (e.getSource() == playBtn) {
                String str1 = tf.getText();
                int int1 = Integer.parseInt(str1);
                q.solution = false;
                q.ncall(int1);
                q.dispose();
            } else if (e.getSource() == exitBtn) {
                System.exit(0);
            }

        }

    }

}
```

II. NQueen.java

```
package project;

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import javax.swing.JButton;
import javax.swing.JPanel;
import java.awt.event.ActionListener;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;
```

```
public class NQueen extends JPanel implements ActionListener {

   JButton pos[][] = new JButton[30][30];
   int arr[][] = new int[30][30];
   int allPlace = 0, queen = 0, n = 1;
   Queen q;

   public NQueen(Queen q) {
      this.q = q;
   }

   public NQueen(int n, boolean sol, Queen q) {
      this.q = q;
      this.n = n;
      if (n != 0) {
         setLayout(new GridLayout(n, n));
         if (sol) {
            queenPlace(0, n);
         }
         for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
               pos[i][j] = (JButton) add(new JButton());
               if (!sol) {
                  pos[i][j].addActionListener(this);
               }

               if (i % 2 == 0 && j % 2 == 0) {
                  pos[i][j].setBackground(Color.gray);
               }
               if (i % 2 == 0 && j % 2 != 0) {
                  pos[i][j].setBackground(Color.white);
               }
               if (i % 2 != 0 && j % 2 == 0) {
                  pos[i][j].setBackground(Color.white);
               }
               if (i % 2 != 0 && j % 2 != 0) {
                  pos[i][j].setBackground(Color.gray);
               }
               if (sol) {
                  System.out.println("Yes " + sol);
                  if (arr[i][j] == 1) {
                     /*
                     ImageIcon imgicn= new ImageIcon("chess.jpg");
                     Image img=imgicn.getImage();
                     int h=pos[i][j].getHeight();
                     int w=pos[i][j].getWidth();

                     Image nwimg=img.getScaledInstance(w, h,Image.SCALE_SMOOTH);
                     imgicn= new ImageIcon(nwimg);
                     pos[i][j].setIcon(imgicn);
                      */
                     pos[i][j].setIcon(new ImageIcon("chess.jpg"));
                  }
               }
            }
         }
      }
   }

   public void actionPerformed(ActionEvent ae) {
      System.out.println("Hello, ");
      for (int i = 0; i < n; i++) {
         System.out.println("Row " + i);
         for (int j = 0; j < n; j++) {
            System.out.println("Col " + j);
            if (ae.getSource() == pos[i][j]) {
               System.out.println("Row Row Row " + j);
               if (arr[i][j] != 1) {
                  System.out.println("Col Col Col " + j);
                  /*
                  ImageIcon imgicn= new ImageIcon("chess.jpg");
                  Image img=imgicn.getImage();
                  int h=pos[i][j].getHeight();
                  int w=pos[i][j].getWidth();

                  Image nwimg=img.getScaledInstance(w, h,Image.SCALE_SMOOTH);
                  imgicn= new ImageIcon(nwimg);
                  pos[i][j].setIcon(imgicn);
                  */
                  pos[i][j].setIcon(new ImageIcon("chess.jpg"));

                  setarr(i, j);
                  queen++;
                  if (queen == n) {
                     JOptionPane.showMessageDialog(null, "Congratulation, You Won", "WINNER", JOptionPane.INFORMATION_MESSAGE);//destroy();

                     q.solution = false;
                     q.ncall(this.n);
                     q.dispose();
                     break;
                  }
                  int dead = 1;
                  for (int k = 0; k < n; k++) {
                     for (int l = 0; l < n; l++) {
                        if (arr[k][l] == 0) {
                           dead = 0;
                        }
                     }
                  }
                  if (dead == 1) {
                     JOptionPane.showMessageDialog(null, "You can't place more Queen", "LOOSE", JOptionPane.WARNING_MESSAGE);
```

```
                    q.solution = false;
                    q.ncall(this.n);
                    q.dispose();
                    break;
                }
            } else {
                JOptionPane.showMessageDialog(null, "You can't place queen here", "WARNING", JOptionPane.WARNING_MESSAGE);

            }
        }
    }
}

public void setarr(int x, int y) {
    int a, b;
    for (a = 0; a < n; a++) {
        arr[a][y] = 1;
    }
    for (b = 0; b < n; b++) {
        arr[x][b] = 1;
    }
    for (a = x, b = y; (a < n && b >= 0); a++, b--) {
        arr[a][b] = 1;
    }
    for (a = x, b = y; (a >= 0 && b < n); a--, b++) {
        arr[a][b] = 1;
    }
    for (a = x, b = y; (a >= 0 && b >= 0); a--, b--) {
        arr[a][b] = 1;
    }
    for (a = x, b = y; (a < n && b < n); a++, b++) {
        arr[a][b] = 1;
    }
}

public void queenPlace(int r, int n) {
    System.out.println("Queen place");

    for (int i = 0; i < n; i++) {
        if (allPlace == 0) {
            int tmp = 0, a, b;

            for (b = 0; b < n; b++) {
                arr[r][b] = 0;
            }
            for (a = 0; a < n; a++) {
                if (arr[a][i] == 1) {
                    tmp = 1;
                }
            }
            if (tmp == 0) {
                for (a = r, b = i; (a < n && b >= 0); a++, b--) {
                    if (arr[a][b] == 1) {
                        tmp = 1;
                    }
                }
            }
            if (tmp == 0) {
                for (a = r, b = i; (a >= 0 && b < n); a--, b++) {
                    if (arr[a][b] == 1) {
                        tmp = 1;
                    }
                }
            }
            if (tmp == 0) {
                for (a = r, b = i; (a >= 0 && b >= 0); a--, b--) {
                    if (arr[a][b] == 1) {
                        tmp = 1;
                    }
                }
            }
            if (tmp == 0) {
                for (a = r, b = i; (a < n && b < n); a++, b++) {
                    if (arr[a][b] == 1) {
                        tmp = 1;
                    }
                }
            }

            if (tmp == 0) {
                arr[r][i] = 1;
                if (r == n - 1) {
                    allPlace = 1;
                } else {
                    queenPlace(r + 1, n);
                }
            }
        }
    }
}

}
```

III. Qeen.java :

```
package project;

import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JFrame;

public class Queen extends JFrame {

    private final Container mainContainer;
    private final NQueen nq;
    private final ButtonPanel bp;
    protected static boolean solution;

    Queen(int n) {

        super("NQUEEN");
```

```
        System.out.println("Solution " + solution);
        if (solution) {
            nq = new NQueen(n, true, this);
        } else {
            nq = new NQueen(n, false, this);
        }
        bp = new ButtonPanel(this);
        setSize(500, 545);
        mainContainer = getContentPane();
        mainContainer.add(nq, BorderLayout.CENTER);
        mainContainer.add(bp, BorderLayout.SOUTH);
        this.setResizable(false);
    }

    public static void main(String arg[]) {
        ncall(0);
    }

    public static void ncall(int n) {
        System.out.println("Call " + solution);
        Queen q = new Queen(n);
        q.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        q.show();
    }

}
```

X. DISTRIBUTION OF WORK

'Talent wins games, but teamwork and intelligence win championships." --*Michael Jordan*[5], as the colossal individual and extraordinary achiever said all the accomplishment can be accomplished by just cooperation! Each of the gathering part strived to take this anticipate to it end. Where coding was taken care of by each of part! Kishan Gabani was better in planning so was centered on outlining. Sanket Lad was better in rationale operation, so he concentrated on the way how the ruler can be settled on its areas with every one of the conceivable outcomes. Patel Vishal was better in java programming so completely centered around how this riddle can be executed with better representation and from it's all rationale.

Report was taken care of by every individual from gathering, where every part made their own particular style of report with their own configuration and present before other. In conclusion, the last report made within the sight of all gathering individuals by taking the essential information from the majority of the reports!

XI. CONCLUSION

In this paper, a riddle fathoming thought named N-Queen is proposed to illuminate the riddle to pt 8 rulers on a chessboard without collaborating with each other in a solitary line, section or being in a corner to corner. What is this issue? , What was it before ? how it is unraveled, and how we tackled it utilizing java projects is appeared as above with all the screenshots, tables and all the conceivable chances it can be with 1 to N Queen.

XII. ACKNOWLEDGEMENT

This research was supported by Dr. Wenjia Li . We thank our colleagues from NYIT, who provided insight and expertise that greatly assisted the research, although they may not agree with all of the interpretations of this paper.

We say thanks to Dr. Wenjia Li for help with this specific subject and to learn all of us the systems utilized for a PL, additionally to give us an inside and out information in regards to every one of the ideas of PL and thought regarding utilizing this to make something new

Thank You,

REFERENCES

[1] Max Bezzel (4 February 1824 – 30 July 1871) was a German chess composer who created the eight queens puzzle in 1848. [2] Franz Nauck published the first solutions in 1850 [3] Carl Friedrich Gauss - Some of the time alluded to as the Princeps mathematicorum and most prominent mathematician since relic, Gauss had an extraordinary impact in numerous fields of arithmetic and science and is positioned as one of history's most powerful mathematicians. [4] Edsger Dijkstra- Edsger Wybe Dijkstra was a Dutch computer scientist. A theoretical physicist by training, he worked as a programmer at the Mathematisch Centrum from 1952 to 1962.[5] also known by his initials, MJ, is an American retired professional basketball player.

- ✓ http://en.literateprograms.org/Eight_queens_puzzle_%28C%29
- ✓ http://en.wikipedia.org/wiki/Eight_queens_puzzle
- ✓ Y.Daniel Liang "Introduction to Java Programming, 7th edition",Prentice Hall,2008.
- ✓ Big Java ,2nd edition, Cay Horstman, Wiley Student Edition , Wiely India Private Limited.
- ✓ Herbert Schildt "The Complete Reference Java, J2SE 5th edition", Tata McGraw Hill Edition,2007.
- ✓ Bitner, J.R. and E.M. Reingold (1975), "Backtracking programming techniques," Communications of the ACM, Vol. 18, No. 11, pp. 651-56.

- ✓ Bitner, J.R. and E.M. Reingold (1975), "Backtracking programmingtechniques," Communications of the ACM, Vol. 18, No.11, pp. 651-56. Herbert Schildt "The Complete Reference Java, J2SE 5th edition", Tata McGraw Hill Edition,2007.
- ✓ http://www.java.ahchuthan.org/2012/02/n-queens-problem-in-java.html