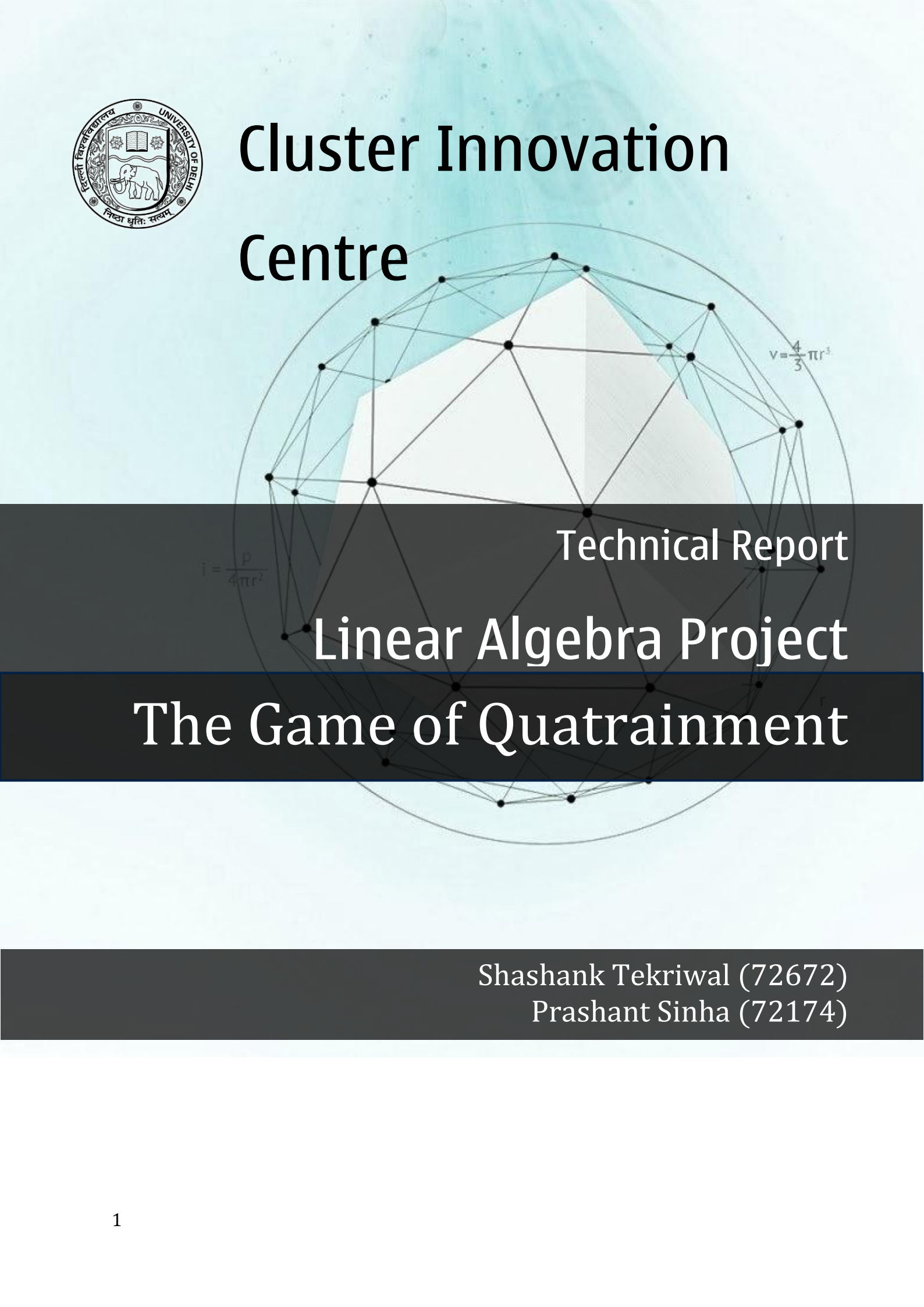


Cluster Innovation Centre

Technical Report

Linear Algebra Project

The Game of Quatrainment

Shashank Tekriwal (72672)
Prashant Sinha (72174)

Acknowledgement

Completion of this Project was a combined effort, and it received a lot of help from our teachers and our peers. We'd like to thank Ms. Sonam Singh for her guidance, and suggestion of topic of the project.

Further, we'd like to appreciate mutual effort all of us gave individually, and as a team.

Prashant Sinha,
Shashank Tekriwal,
22-04-2014

Contents

1 Abstract:

The game of Quatrainment is a simple game where the states of a starting four by four grid are manipulated to match a target four by four grid.

In this report we discuss the underlying Mathematical Principles that make this seemingly difficult game easy. We also built an implementation of this game.

2 The Game of Quatrainment

2.1 Introduction

The game of Quatrainment is a simple game where the states of a starting four by four grid are manipulated to match a target four by four grid.

2.2 Gameplay

The game Quatrainment is played by First Filling two four by four grids randomly with any number of X's, as displayed in Figure 1. The goal of the game is to manipulate the pattern of 1a so that it matches 1b by manipulating the states of certain cells. To understand which cells we are manipulating we will label them 1 through 16 as displayed in Figure 2. Selecting different cells has different effects on the grid.

	x		x
		x	
x	x		
x			x

(a) Starting Stage

x	x		x
x		x	
x	x		

(b) Target Stage

Figure 1

One must understand that there are three different types of cells: corner cells, edge cells, and centre cells, each with their own transformation rule when selected:

- If a corner cell is selected, then that cell and the six cells that form a triangle in the corner are reversed. If it was an X it is now blank, if it was blank, it now has an X (See Figure 3a).
- If a centre cell is selected, then that cell and the four adjacent (non-diagonal) cells to it are reversed (See Figure 3b).
- If an edge cell is selected, then that cell is left alone but the three cells adjacent to it are reversed (See Figure 3c). (See Section 2.3 for Illustrations)

Since the only option for each cell is either blank or an X, we change the notation to a zero for a blank cell and a one for a cell with an X as one can see in Figure 4 where the same grids from Figure 1 have been converted.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figure 2: Blank grid filled with numbers for reference.

The purpose of converting the grids to binary is that we will be using the properties of matrices further on in this paper, and this allows us to use those properties in a base-two system.

2.2.3 XOR operation

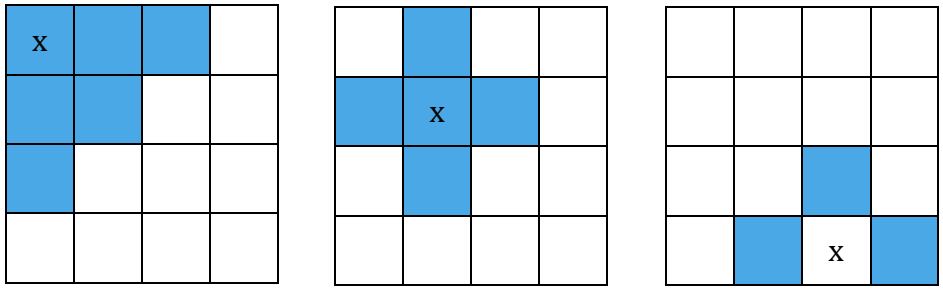
The XOR, or e**X**clusive**OR** operation has following set of rules. These are to be followed in the game solution step, as well. In our notation, a crossed cell is equivalent to 1 and an empty cell is 0, as mentioned in the previous section.

```
0 (+) 0 = 0
0 (+) 1 = 1
1 (+) 0 = 1
1 (+) 1 = 0
```

XOR Operations on a 2-bit system.

2.3 Moves

Let X represents currently selected cell in the matrix. Then, the manipulated (XOR'd) cells are shown below.



(c) A corner cell selected *(b) An inner cell selected* *(a) An edge cell selected*

Figure 3: The "X" represents the cell selected, and the Blue cells represents the cells that are manipulated.

2.3.1 Cell Manipulation: An Example

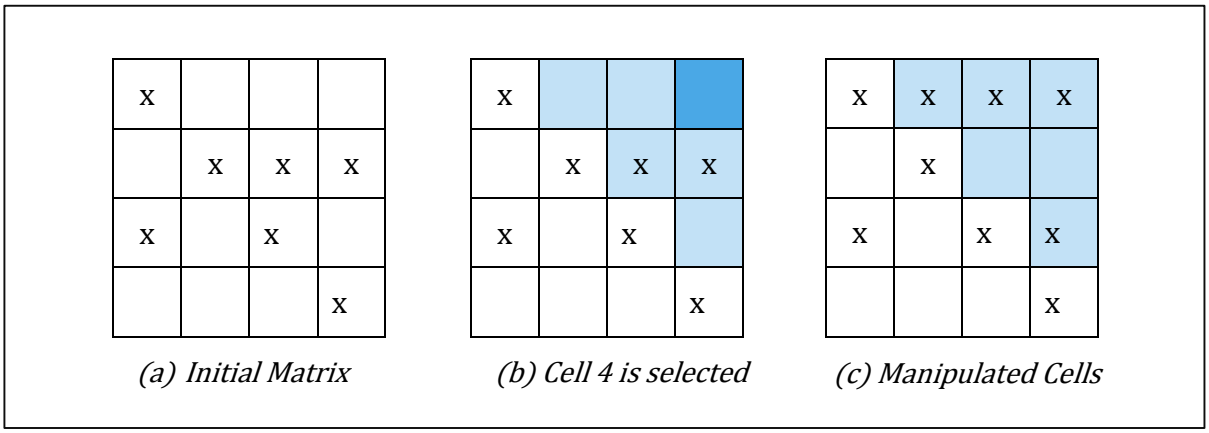


Figure 4

1. Let 4a be a given matrix.
2. Let say we select Cell 4. From Figure 3a, we have manipulation candidates as:

   `{(a → Cell Number) , {2, 3, 4, 7, 8, 12}}`

3. From Section 2.2.3, we know that Crossed Cells are to be replaced by Empty Cells, and vice-versa, in highlighted region. (Figure 4b)
4. Hence, Figure 4c is the result.

2.4 Sample gameplay:

Please see the Appendix 1.

3 Solution

3.1 Is a solution to a Quatrainment puzzle always possible?

Yes.

To show that there is always a solution, we need to show that it is possible to get from any starting combination of X's, to a blank grid.

Since each cell can take either of the value in the set: {0, 1}, we have a total of 2^{16} possible matrices of this kind.

But, we only have 16 possible Moves, that is, Input Cells. These are 16 Linearly Independent matrices, which can solve any given Puzzle. This is proved in the next Section.

They're as follows:

$$M_1 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, M_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, M_3 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, M_4 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$M_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, M_6 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, M_7 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, M_8 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$M_9 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, M_{10} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, M_{11} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, M_{12} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{13} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}, M_{14} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}, M_{15} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, M_{16} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Figure 5: Mapping resulting from a total of 16 possible inputs and the "1" representing manipulative cells.

3.2 Proof of Linear Independency

To show that all the Matrices, from M_1 to M_{16}, in Figure 5, are linearly independent.

Condition: The Matrices are linearly independent if and only if there are no linear combinations of the Matrices that add up to zero.

Above statement can be expressed as following:

$$\sum_{i=1}^{16} a_i M_i = 0 \qquad \dots (1)$$

In fact, the only solution to (1) is:

... (*To be proved*)

$$a_1 = a_2 = a_3 = \cdots = a_{16} = 0$$
$$\equiv a_i = 0\ , i\ \epsilon\ [1, 16]$$

Let's Expand (1) using Matrices, M.

We get:

$$a_1\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + a_2 \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \ldots + a_{16} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Further simplifying it, we get:

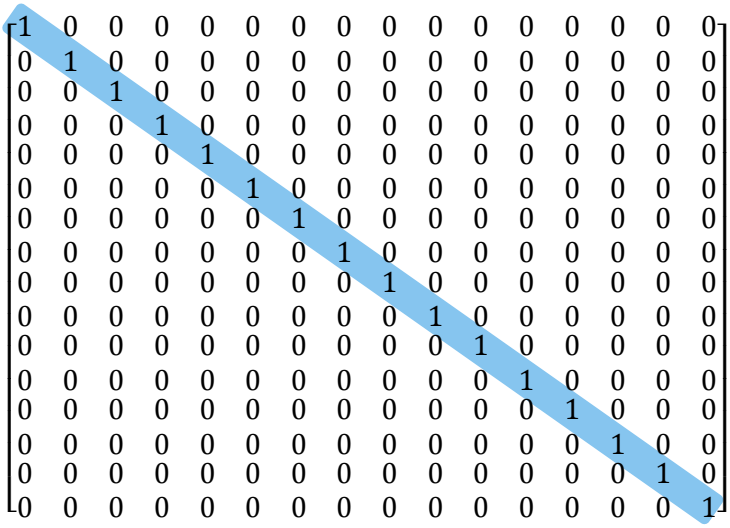
$$\begin{bmatrix} a_1+a_2+a_5 & a_1+a_6+a_9+a_{13} & a_1+a_2+a_4+a_7 & a_3+a_4+a_8 \\ a_1+a_6+a_9+a_{13} & a_1+a_2+a_5+a_6+a_7+a_{10} & a_3+a_4+a_6+a_7+a_8+a_{11} & a_4+a_7+a_{12}+a_{16} \\ a_1+a_5+a_{10}+a_{13} & a_6+a_9+a_{10}+a_{11}+a_{13}+a_{14} & a_7+a_{10}+a_{11}+a_{12}+a_{15}+a_{16} & a_4+a_8+a_{11}+a_{16} \\ a_9+a_{13}+a_{14} & a_{10}+a_{13}+a_{15}+a_{16} & a_{11}+a_{13}+a_{14}+a_{16} & a_{12}+a_{15}+a_{16} \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \ldots (2)$$

This gives us 16 linear equations. Solution of which can be found by Matrix-Method.

The Matrix representation of (2) above is as follows:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{15} \\ a_{16} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \ldots (3)$$

We can Row-Reduce the Matrix in (3) to get its Echelon form, as follows:



We see that there is a Pivot in every Row. (Blue Hightlighted)

$$\therefore \textit{The Matrix Set } M_i \textit{ is Linearly Independent.}$$

This Supports our Hypothesis above.

$$\textit{Hence Proved, that the Matrix Set } M \textit{ is Linearly Independent.}$$

3.3 Solution Steps

This section describes a method to find out the Solution of a given 4x4 puzzle.

Let $\boldsymbol{\mathcal{R}}$ represent our 16x16 input state matrix, in Equation (3), section 3.2, and $\boldsymbol{\alpha}$ represents the actual inputs required to get to the Target Matrix.

Also, let $\boldsymbol{\mathcal{P}}$ be the column Vector, representing the Target state.

Hence, we have:

$$\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\alpha} = \boldsymbol{\mathcal{P}}$$

$$\boldsymbol{\mathcal{R}^{-1}}\,\boldsymbol{\mathcal{R}} \cdot \boldsymbol{\alpha} = \boldsymbol{\mathcal{R}^{-1}} \cdot \boldsymbol{\mathcal{P}}$$

$$\boldsymbol{\alpha} = \boldsymbol{\mathcal{R}^{-1}} \cdot \boldsymbol{\mathcal{P}}$$

We may Use MATLAB or Python for finding out $\boldsymbol{\mathcal{R}^{-1}}$.

Following is the $\boldsymbol{\mathcal{R}^{-1}}$ we get when we run the Code provided in a Paper by Loran Briggs and Daniel McBride[1]. Please see the Appendix 2 for the Code.

[1] The Mathematics of Quatrainment (Loran Briggs and Daniel McBride) 20101217

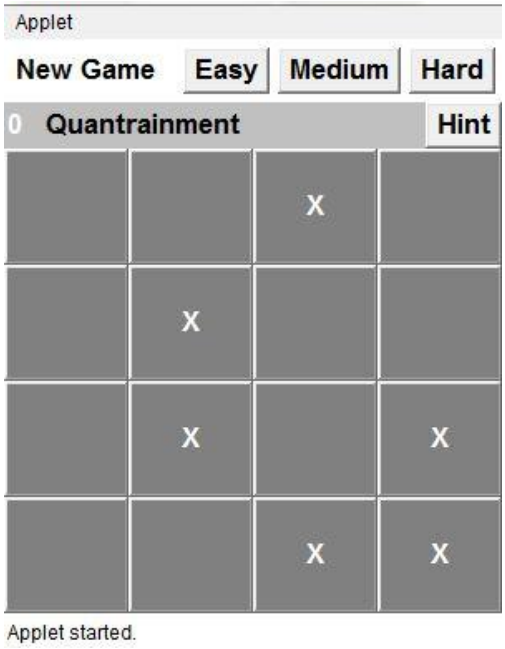
$$\boldsymbol{\mathcal{R}^{-1}} = \begin{bmatrix} 1&1&1&0&1&0&1&0&1&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1&0&1&0&0&0&1&1&1 \\ 0&1&0&0&1&0&0&1&0&0&1&0&1&1&1&0 \\ 0&1&1&1&0&1&0&1&1&0&1&1&0&1&0&0 \\ 0&1&0&0&0&0&1&1&1&0&0&1&0&1&0&1 \\ 1&0&0&1&0&1&1&0&0&1&0&0&1&0&0&0 \\ 1&0&0&1&0&1&1&0&0&0&1&0&0&0&0&1 \\ 0&0&1&0&1&1&0&0&1&0&0&1&1&0&1&0 \\ 0&1&0&1&1&0&0&1&0&0&1&1&0&1&0&0 \\ 1&0&0&0&0&1&0&0&0&1&1&0&1&0&0&1 \\ 0&0&0&1&0&0&1&0&0&1&1&0&1&0&0&1 \\ 1&0&1&0&1&0&0&1&1&1&0&0&0&0&1&0 \\ 0&0&1&0&1&1&0&1&1&0&1&0&1&1&1&0 \\ 0&1&1&1&0&1&0&0&1&0&0&1&0&0&1&0 \\ 1&1&1&0&0&0&1&0&1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&1&0&1&0&1&0&1&1&1 \end{bmatrix}$$

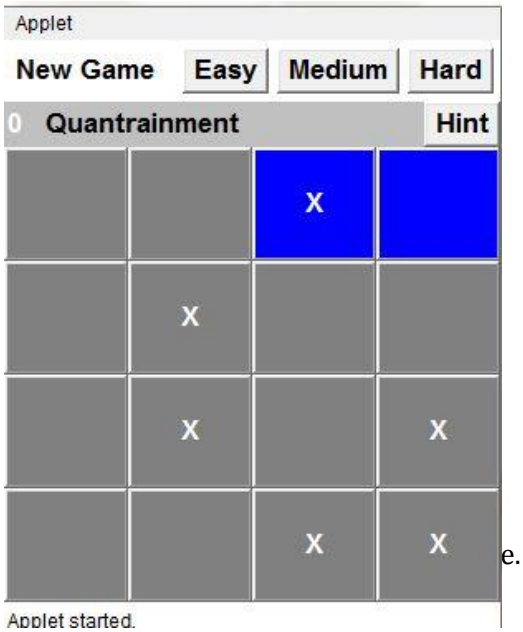
Since we have the $\boldsymbol{\mathcal{R}^{-1}}$ available, we can easily find a solution in form of Input Matrix, $\boldsymbol{\alpha}$.

4 Implementation in JAVA

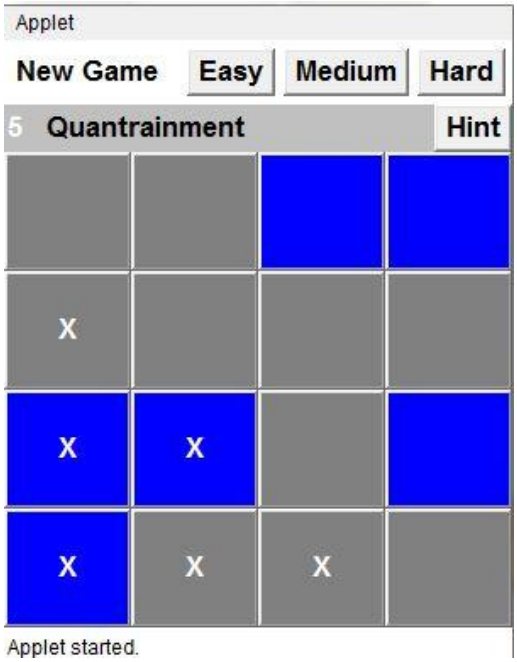
We implemented the Game in JAVA Language in form of an applet. Followings are the Screen-Shot of the applet.

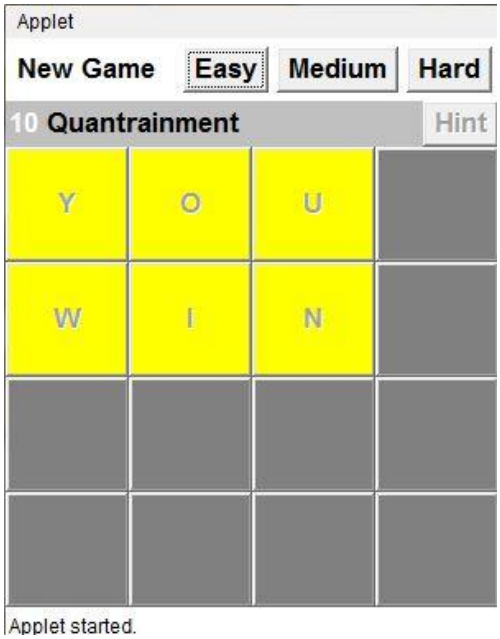
The Source Code is in Appendix 3.





e.





5 Conclusion

The Game of Quatrainment, with its finite number of moves, is a great example of how basic Linear Algebra can be applied to make an Engaging Puzzle.

We proved the solvability of the puzzle, and built it on Java Applet as a demonstration.

6 References

1. T. E. Gantner. 1988 The Game of Quatrainment.
2. J. Miller, K. Menkveld. 2000 The Game of Quatrainment Revisited
3. L. Briggs, D. McBride. 2010 The Mathematics of Quatrainment.
4. Various. 2013 http://www.tutorialspoint.com/Java

I Appendix 1: Sample Gameplay

X		X	
	X		X
X		X	
	X		X

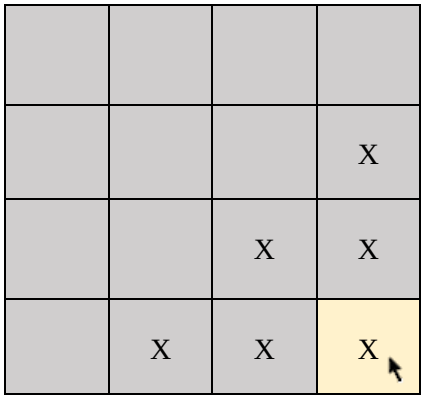
The following figures represents the solution matrix. All the cells marked yellow when clicked one by one lead the grid to the target state i.e. the blank matrix. This is how the game proceeds:

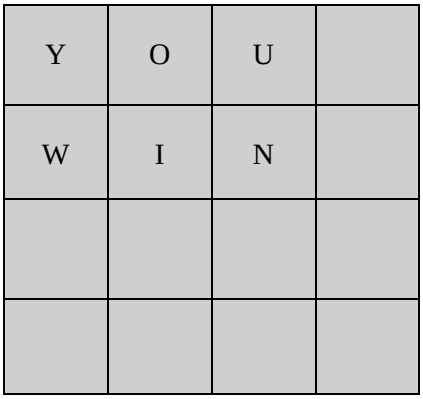
	X		
X			X
		X	
	X		X

			X
X		X	X
		X	
	X		X

X			X
		X	X
	X		X

			X
	X	X	X
X	X		X





Note: If any of the non-yellow cells are clicked in between then the state of the grid changes and hence the solution too. The yellow cells then will not lead to the solution. The user will then have to click on the hint button again. The hint function is dynamic and gives hints according to the current state of the grid.

II Appendix 2: MATLAB Code

```
function x = bin_solve(A,b)
% solver for systems of linear equations in binary arithmetic using
% Gaussian elimination algorithm.
	mat=[A,b];
	[m n]=size(A); % read the size of the original matrix A
	for i=1:n
		j= find(mat(i:m,i),1); % finds FIRST 1 in i-th column starting at i
		if isempty(j)
			error('More than one solution.');
		else
			j=j+i-1; % we need to add i-1 since j starts at i
			temp=mat(j,:); % swap rows
			mat(j,:)=mat(i,:);
			mat(i,:)=temp;
			% add i-th row to all rows that contain 1 in i-th column
			% starting at j+1 - remember up to j are zeros
			for k=find(mat((j+1):m,i))'
				mat(j+k,:)=bitxor(mat(j+k,:),mat(i,:));
		end
	end
end

% remove all-zero rows if there are some
mat=mat(sum(mat,2)>0,:);
if any(sum(mat(:,1:n),2)==0) % no solution because matrix A contains
	error('No solution.'); % all-zero row, but with nonzero RHS
end

%% Finding R inverse
R= [1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0;
	1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0;
	1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0;
	0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0;
	1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0;
	1 1 0 0 1 1 1 0 0 1 0 0 0 0 0 0;
	0 0 1 1 0 1 1 1 0 0 1 0 0 0 0 0;
	0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1;
	1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0;
	0 0 0 0 0 1 0 0 1 1 1 0 1 1 0 0;
	0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1;
	0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1;
	0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0;
	0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1;
	0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1;
	0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1];
Ri=zeros(16,16);
I=eye(16);
for k=1:16
	Ri(:,k)=bin_solve(R,I(:,k));
end
Ri


function x=q_solve(S,T)
P=mod(S+T,2);
r1=P(1,:); r2=P(2,:); r3=P(3,:); r4=P(4,:);
```

```
p=[r1 r2 r3 r4]';
Ri=[1 1 1 0 1 0 1 0 1 1 0 1 0 0 1 0;
      0 0 1 0 1 0 0 1 0 1 0 0 0 1 1 1;
      0 1 0 0 1 0 0 1 0 0 1 0 1 1 1 0;
      0 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0;
      0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1;
      1 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0;
      1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1;
      0 0 1 0 1 1 0 0 1 0 0 1 1 0 1 0;
      0 1 0 1 1 0 0 1 0 0 1 1 0 1 0 0;
      1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 1;
      0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1;
      1 0 1 0 1 0 0 1 1 1 0 0 0 0 1 0;
      0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 0;
      0 1 1 1 0 1 0 0 1 0 0 1 0 0 1 0;
      1 1 1 0 0 0 1 0 1 0 0 1 0 1 0 0;
      0 1 0 0 1 0 1 1 0 1 0 1 0 1 1 1];
a=mod(Ri*p,2);
a1=a(1:4,1); a2=a(5:8,1); a3=a(9:12,1); a4=a(13:16,1);
A=[a1';a2';a3';a4']
```

III Appendix 3: JAVA Applet Code

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Linear extends Applet implements MouseListener
{
    public Linear()
    {
    }

    int matrix[][]={{1,1,1,0,1,0,1,0,1,1,0,1,0,0,1,0},
                    {0,0,1,0,1,0,0,1,0,1,0,0,0,1,1,1},
                    {0,1,0,0,1,0,0,1,0,0,1,0,1,1,1,0},
                    {0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,0},
                    {0,1,0,0,0,0,1,1,1,0,0,1,0,1,0,1},
                    {1,0,0,1,0,1,1,0,0,1,0,0,1,0,0,0},
                    {1,0,0,1,0,1,1,0,0,0,1,0,0,0,0,1},
                    {0,0,1,0,1,1,0,0,1,0,0,1,1,0,1,0},
                    {0,1,0,1,1,0,0,1,0,0,1,1,0,1,0,0},
                    {1,0,0,0,0,1,0,0,0,1,1,0,1,0,0,1},
                    {0,0,0,1,0,0,1,0,0,1,1,0,1,0,0,1},
                    {1,0,1,0,1,0,0,1,1,1,0,0,0,0,1,0},
                    {0,0,1,0,1,1,0,1,1,0,1,0,1,1,1,0},
                    {0,1,1,1,0,1,0,0,1,0,0,1,0,0,1,0},
                    {1,1,1,0,0,0,1,0,1,0,0,1,0,1,0,0},
                    {0,1,0,0,1,0,1,1,0,1,0,1,0,1,1,1}};
    int sol[]=new int[16];
    int p[]=new int[16];
    Button table[][];
    boolean state[][];
    boolean ngame[][];
    int moves;
    int randx , randy;
    int sizex , sizey;
    Panel P , P2 , P3;
    Label Moves, Quant ;
    Button Restart, Solve;
    GridLayout gl;
    int row = 4, col = 4;
    Font font;
    String mark = "X";
    int Solve_counter = 2;
    Panel f;
    int target;

    public void start()
    {
        setLayout(new BorderLayout());
        gl = new GridLayout(row,col);
        P = new Panel(gl);
        font = new Font("ComicSans",Font.BOLD,17);
        setFont(font);
        P2 = new Panel(new BorderLayout());
        P3 = new Panel();
        moves = 0;
        Moves = new Label(Integer.toString(moves));
        Quant = new Label("Quantrainment");
```

```
        table = new Button[row][col];
        state = new boolean[row][col];
        ngame = new boolean[row][col];
        Restart = new Button("Restart");
        Restart.addMouseListener(this);
        Solve = new Button("Solve");
        Solve.addMouseListener(this);
        Restart = new Button("Restart");
        Restart.addMouseListener(this);
        int count=0;
        for(int x=0;x<row;x++)
        {
            for(int y=0;y<col;y++)
            {
                table[x][y]=new Button();
                table[x][y].addMouseListener(this);
                P.add(table[x][y]);
                state[x][y]=false;
                ngame[x][y]=false;
                p[count]=sol[count]=0;
                count++;
            }
        }
        add(P2, "North");
        add(P, "Center");
        P2.add(Moves, "West");
        P2.add(Quant, "Center");
        P2.add(P3, "North");

        P3.add(Restart);
        P2.add(Solve,"East");
        P2.setBackground(Color.lightGray);
        Moves.setBackground(Color.lightGray);
        Moves.setForeground(Color.white);
        Quant.setBackground(Color.lightGray);
        Quant.setForeground(Color.black);
        sizex = 300;
        sizey = 346;
        Restart_Game();
    }

    public void Restart_Game()
    {
        target = 0;
        setSize(sizex, sizey);
        invalidate();
        validate();
        gl.setRows(row);
        gl.setColumns(col);
        moves=0;
        Moves.setText(Integer.toString(moves));
        Solve.setEnabled(true);
        int count = 0;
        for(int x=0;x<row;x++)
        {
            for(int y=0;y<col;y++)
            {
                table[x][y].setEnabled(true);
                table[x][y].setLabel("");
```

```
                table[x][y].setBackground(Color.gray);
                table[x][y].setForeground(Color.white);
                P.add(table[x][y]);
                state[x][y]=false;
                ngame[x][y]=false;
                p[count]=0;
                sol[count]=0;
                count++;
            }
        }
        setSize(sizex,sizey);
        invalidate();
        validate();
        count = 0;
    }

    public void mouseClicked(MouseEvent e)
    {
        if(e.getSource()==Restart)
        {
            Restart_Game();
        }
        if(e.getSource()== Solve)
        {
            target = 1;
            show_Solves();
        }
        for(int x=0;x<row;x++)
        {
            for(int y=0;y<col;y++)
            {
                if(e.getSource()==table[x][y])
                {
                    table[x][y].setBackground(Color.gray);
                    if(target==0)
                    {
                        alter(x,y);
                    }
                    else
                    {
                        moves++;
                        Moves.setText(Integer.toString(moves));
                        if(x==0 && y==0 || x==3 && y==0 || x==0 && y==3 || x==3 &&
y==3)
                        {
                            change_state('c',x,y);
                        }
                        else if(x==0&&y>0&&y<3 || x==3&&y>0&&y<3 || y==0&&x>0&&x<3
|| y==3&&x>0&&x<3)
                        {
                            change_state('e',x,y);
                        }
                        else
                        {
                            change_state('i',x,y);
                        }
                        check_win();
                    }
                }
```

```
            }
        }
    }

    public void change_state(char ch, int m, int n)
    {
        switch(ch)
        {
            case 'c':
                alter(m,n);
                alter(m,Math.abs(n-1));
                alter(m,Math.abs(n-2));
                alter(Math.abs(m-1),n);
                alter(Math.abs(m-2),n);
                alter(Math.abs(m-1),Math.abs(n-1));
                break;
            case 'e':
                if(m==0 || m==3)
                {
                    alter(m,n-1);
                    alter(m,n+1);
                    alter(Math.abs(m-1),n);
                }
                else
                {
                    alter(m+1,n);
                    alter(m-1,n);
                    alter(m,Math.abs(n-1));
                }
                break;
            case 'i':
                alter(m,n);
                alter(m+1,n);
                alter(m-1,n);
                alter(m,n+1);
                alter(m,n-1);
                break;
            default:
                break;
        }
    }

    public void alter(int m , int n)
    {
        state[m][n]=!state[m][n];
        if(state[m][n])
        {
            table[m][n].setLabel(mark);
        }
        else
        {
            table[m][n].setLabel("");
        }
    }

    public void show_Solves()
    {
        int count=0;
        for(int x=0;x<row;x++)
```

```
        {
            for(int y=0;y<col;y++)
            {
                table[x][y].setBackground(Color.gray);
                if(state[x][y])
                {
                    p[count]=1;
                }
                else
                {
                    p[count]=0;
                }
                count++;
            }
        }
        multiply();
        count=0;
        outer:
        for(int x=0;x<row;x++)
        {
            inner:
            for(int y=0;y<col;y++)
            {
                if(sol[count]==1)
                {
                    blink(x,y);
                }
                count++;
            }
        }
    }

    public void blink(int m, int n)
    {
        try
        {
            table[m][n].setBackground(Color.blue);
            Thread.sleep(200);
        }
        catch(InterruptedException e)
        {
            //necessary for thread
        }
    }

    public void multiply()
    {
        int sum=0;
        for(int x=0;x<16;x++)
        {
            sum=0;
            for(int y=0;y<16;y++)
            {
                sum=sum + (matrix[x][y]*p[y]);
            }
            sol[x]=sum%2;
        }
    }
```

```
    public void check_win()
    {
        boolean temp = true;
        outer:
        for(int x=0;x<row;x++)
        {
            inner:
            for(int y=0;y<col;y++)
            {
                if(state[x][y]==true)
                {
                    temp = false;
                    break outer;
                }
            }
        }
        if(temp)
        {
            Solve.setEnabled(false);
            for(int x=0;x<row;x++)
            {
                for(int y=0;y<col;y++)
                {
                    table[x][y].setEnabled(false);
                    table[x][y].setBackground(Color.gray);
                }
            }
        }
    }
    //These are not used but are necessary for mouseListener
    public void mouseEntered (MouseEvent e)
    {
    }

    public void mouseExited (MouseEvent e)
    {
    }

    public void mousePressed (MouseEvent e)
    {
    }

    public void mouseReleased (MouseEvent e)
    {
    }
}
```