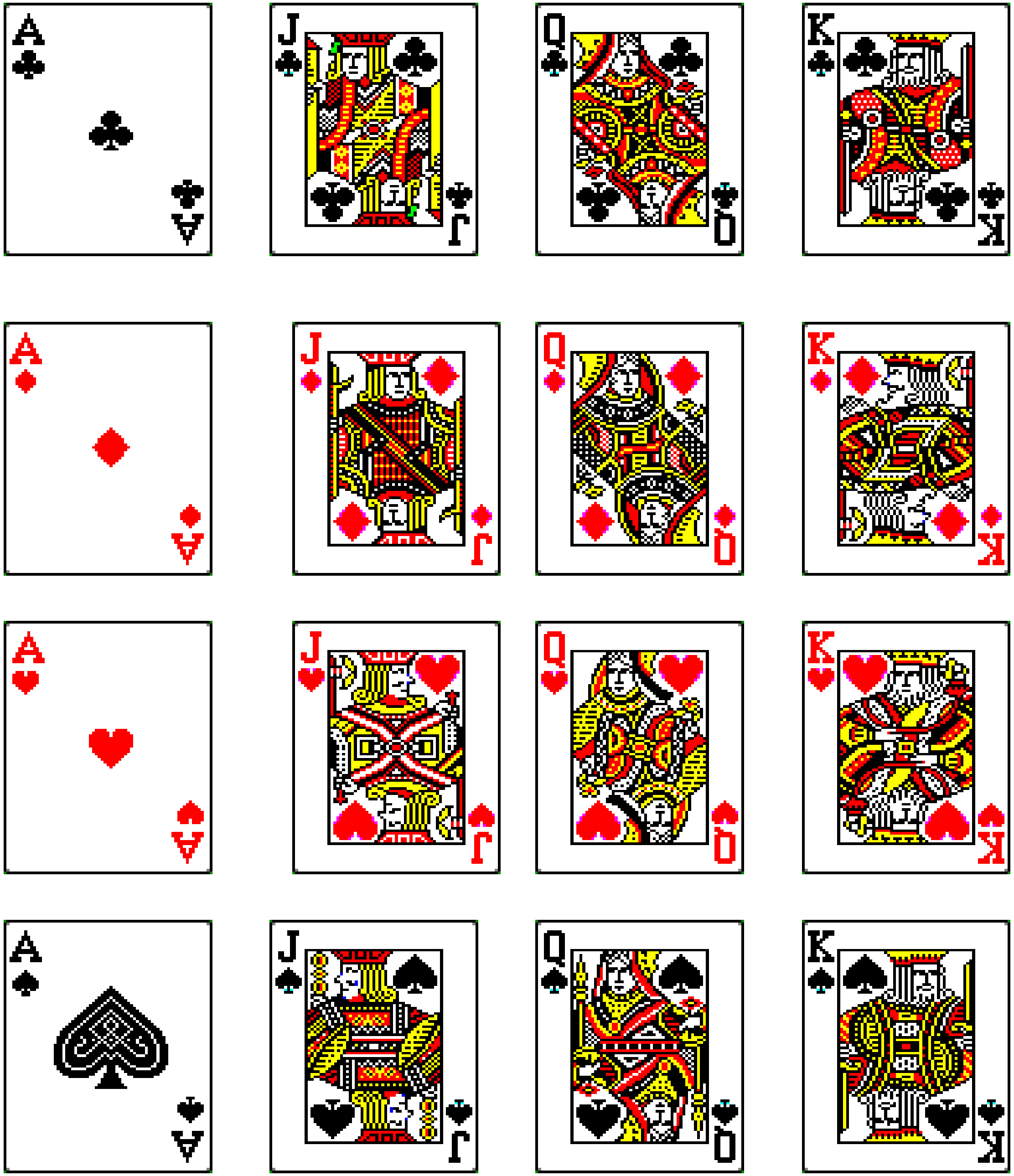
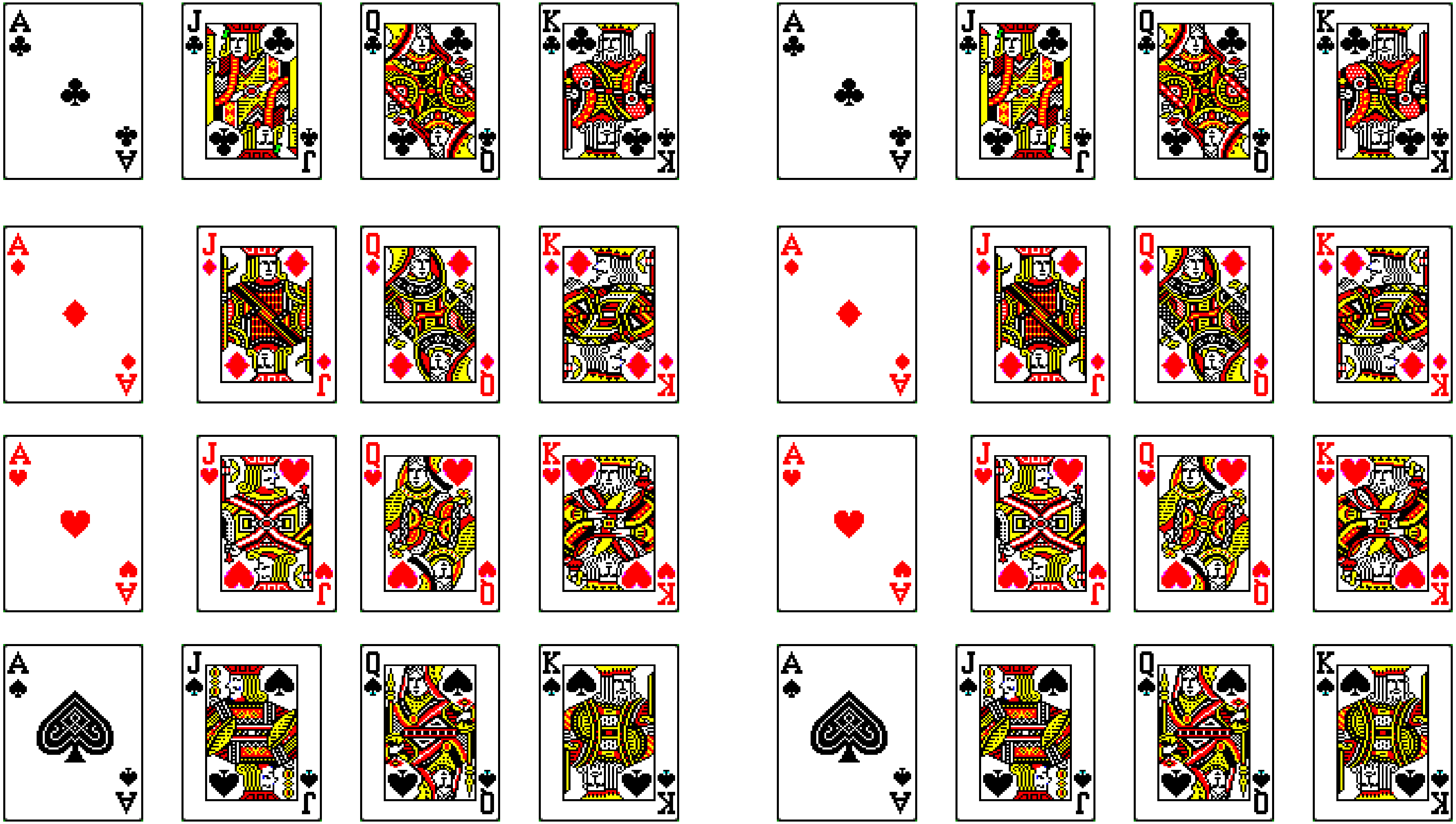
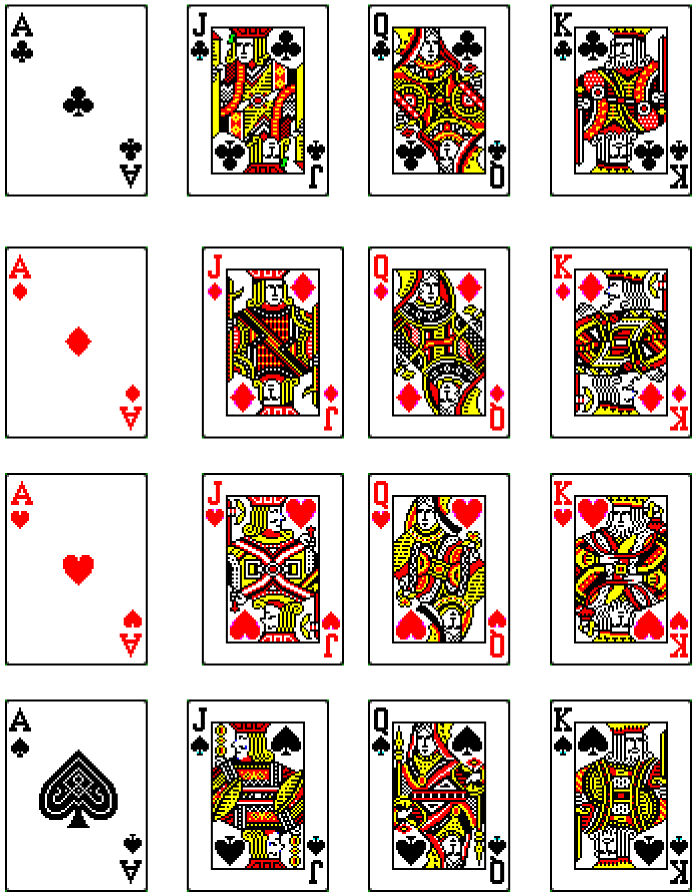
Graeko-Latin Squares







Arrange the playing cards in a 4 by 4 grid such that
each value occurs once in each row and once in each column
and
each suit occurs once in each row and once in each column
(problem is due to Jacques Ozanam 1725).

Graeco-Latin square

From Wikipedia, the free encyclopedia

This article includes a list of references, related reading or external links, but **its sources remain unclear because it lacks inline citations**. Please improve this article by introducing more precise citations. *(November 2010)*

In mathematics, a **Graeco-Latin square** or **Euler square** or **orthogonal Latin squares** of order *n* over two sets *S* and *T*, each consisting of *n* symbols, is an *n*×*n* arrangement of cells, each cell containing an ordered pair (*s*,*t*), where *s* is in *S* and *t* is in *T*, such that every row and every column contains each element of *S* and each element of *T* exactly once, and that no two cells contain the same ordered pair.

Aα	Bγ	Cβ
Bβ	Cα	Aγ
Cγ	Aβ	Bα

Orthogonal Latin squares of order 3

The arrangement of the *s*-coordinates by themselves (which may be thought of as Latin characters) and of the *t*-coordinates (the Greek characters) each forms a Latin square. A Graeco-Latin square can therefore be decomposed into two "orthogonal" Latin squares. Orthogonality here means that every pair (*s*, *t*) from the Cartesian product *S*×*T* occurs exactly once.

Contents [hide]

Aα	Bδ	Cβ	Dε	Eγ
Bβ	Cε	Dγ	Eα	Aδ
Cγ	Dα	Eδ	Aβ	Bε
Dδ	Eβ	Aε	Bγ	Cα
Eε	Aγ	Bα	Cδ	Dβ

History

[edit]

Orthogonal Latin squares have been known to predate Euler. As described by Donald Knuth in Volume 4A, p.3, of TAOCP, the construction of 4x4 set was published by Jacques Ozanam in 1725 (in *Recreation mathematiques et physiques*) as a puzzle involving playing cards. The problem was to take all aces, kings, queens and jacks from a standard deck of cards, and arrange them in a 4x4 grid such that each row and each column contained all four suits as well as one of each face value. This problem has several solutions.

A common variant of this problem was to arrange the 16 cards so that, in addition to the row and column constraints, each diagonal contains all four face values and all four suits as well. As described by Martin Gardner in *Gardner's Workout*, the number of distinct solutions to this problem was incorrectly estimated by Rouse Ball to be 72, and persisted many years before it was shown to be 144 by Kathleen Ollerenshaw. Each of the 144 solutions has 8 reflections and rotations, giving 1152 solutions in total. The 144x8 solutions can be categorized into the following two classes:

Solution	Normal form
Solution #1	A♠ K♥ Q♦ J♣ Q♣ J♦ A♥ K♠ J♥ Q♠ K♣ A♦ K♦ A♣ J♠ Q♥
Solution #2	A♠ K♥ Q♦ J♣ J♦ Q♣ K♠ A♥ K♣ A♦ J♥ Q♠ Q♥ J♠ A♣ K♦

For each of the two solutions, 24x24 = 576 solutions can be derived by permuting the four suits and the four face values independently. No permutation will convert the two solutions into each other.

The solution set can be seen to be complete through this proof outline:

1. Without loss of generality, let us choose the card in the top left corner to be A♠.
2. Now, in the second row, the first two squares can be neither ace nor spades, due to being on the same column or diagonal

In April 1959, Parker, Bose, and Shrikhande presented their paper showing Euler's conjecture to be false for all $n \geq 10$. Thus, Graeco-Latin squares exist for all orders $n \geq 3$ except $n = 6$.

Applications

[edit]

Graeco-Latin squares are used in the design of experiments, tournament scheduling and constructing magic squares. The French writer Georges Perec structured his 1978 novel Life: A User's Manual around a 10×10 orthogonal square.

Mutually orthogonal Latin squares

[edit]

Mutually orthogonal Latin squares arise in various problems. A set of Latin squares is called mutually orthogonal if every pair of its element Latin squares is orthogonal to each other.

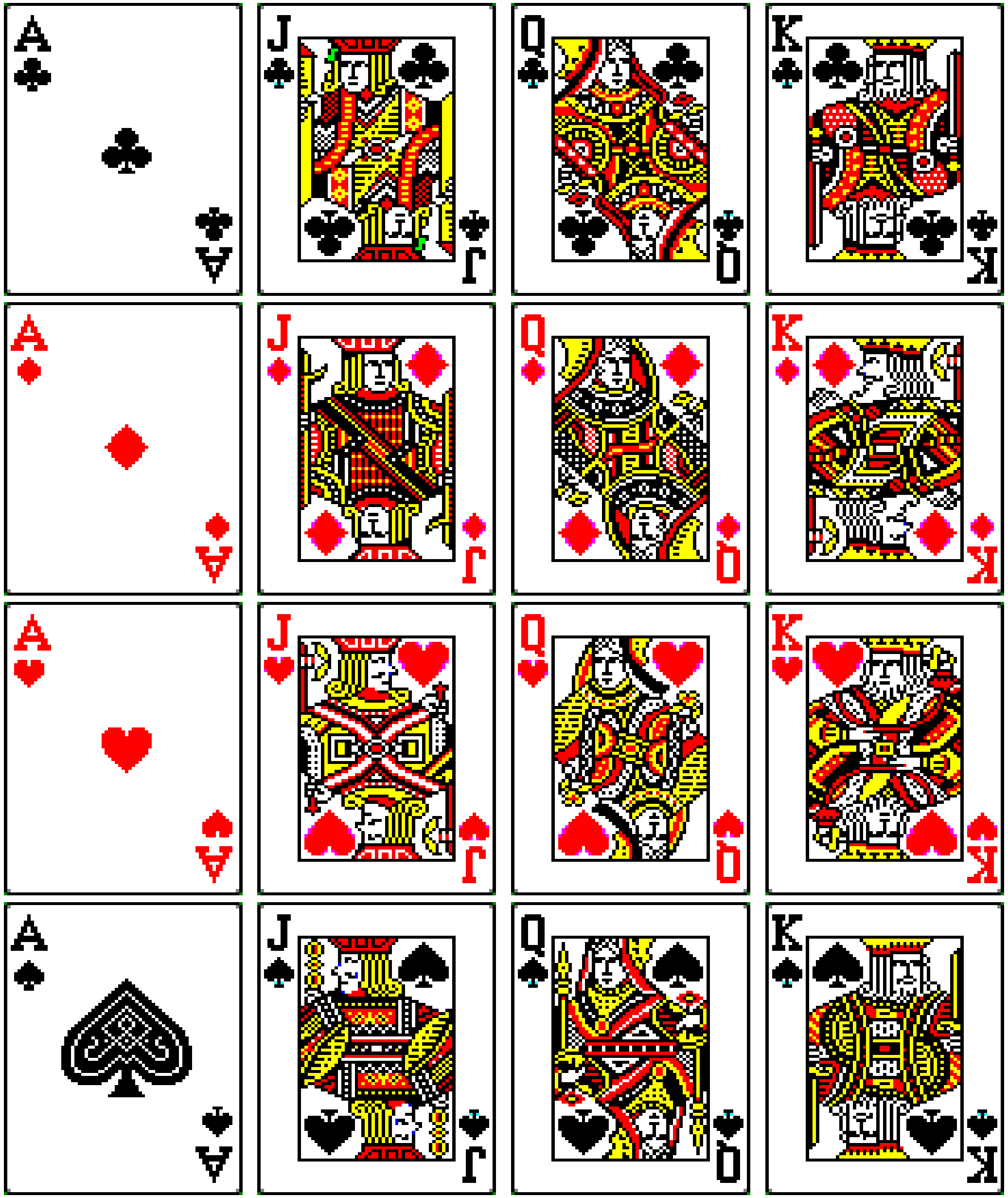
Any two of text, foreground color, background color and typeface form a pair of orthogonal Latin squares:

fjords	jawbox	phlegm	qiviut	zincky
zincky	fjords	jawbox	phlegm	qiviut
qiviut	zincky	fjords	jawbox	phlegm
phlegm	qiviut	zincky	fjords	jawbox
jawbox	phlegm	qiviut	zincky	fjords

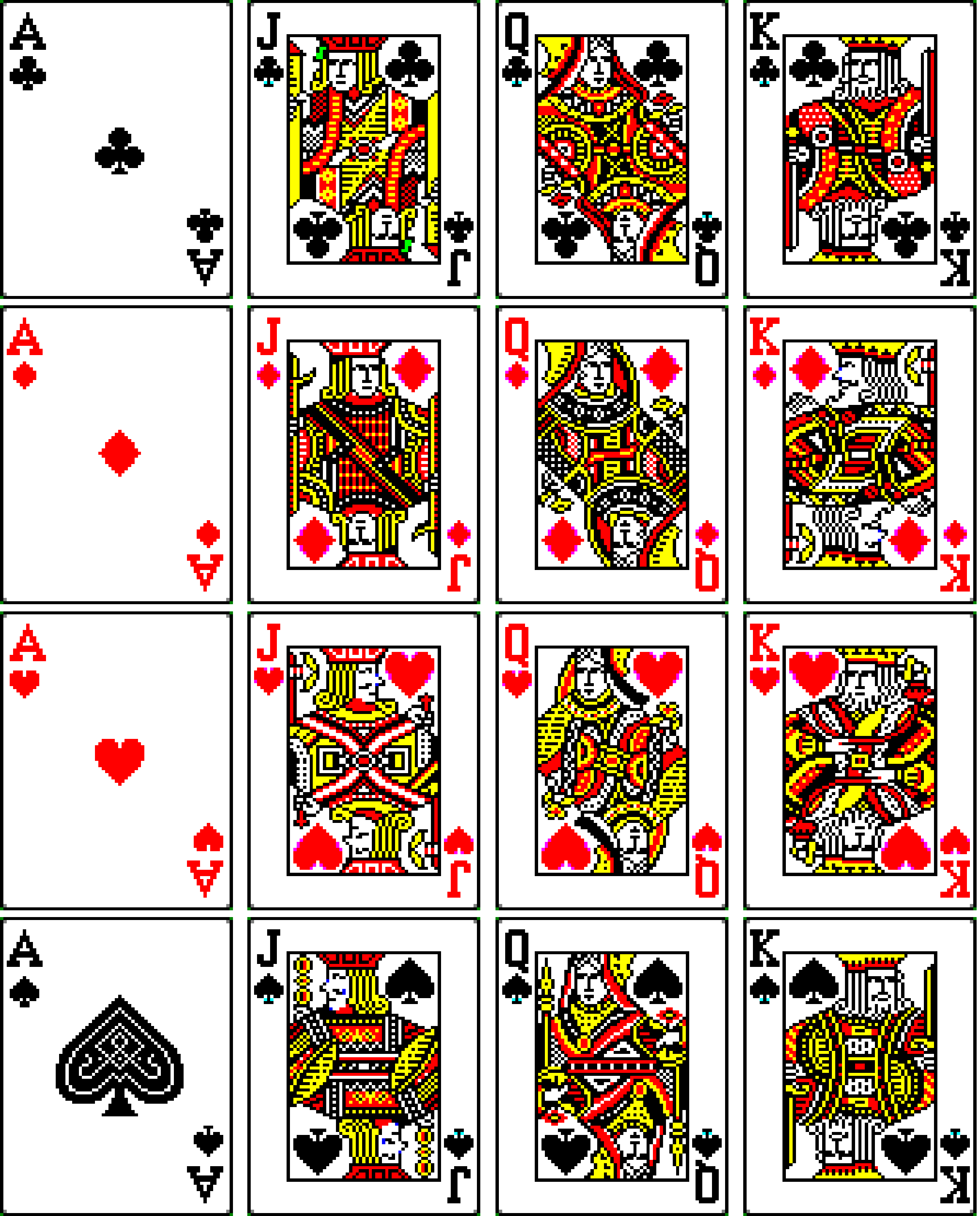
The above table shows 4 mutually orthogonal Latin squares of order 5, representing respectively:

- the text: *fjords*, *jawbox*, *phlegm*, *qiviut*, and *zincky*
- the foreground color: white, red, lime, blue, and yellow

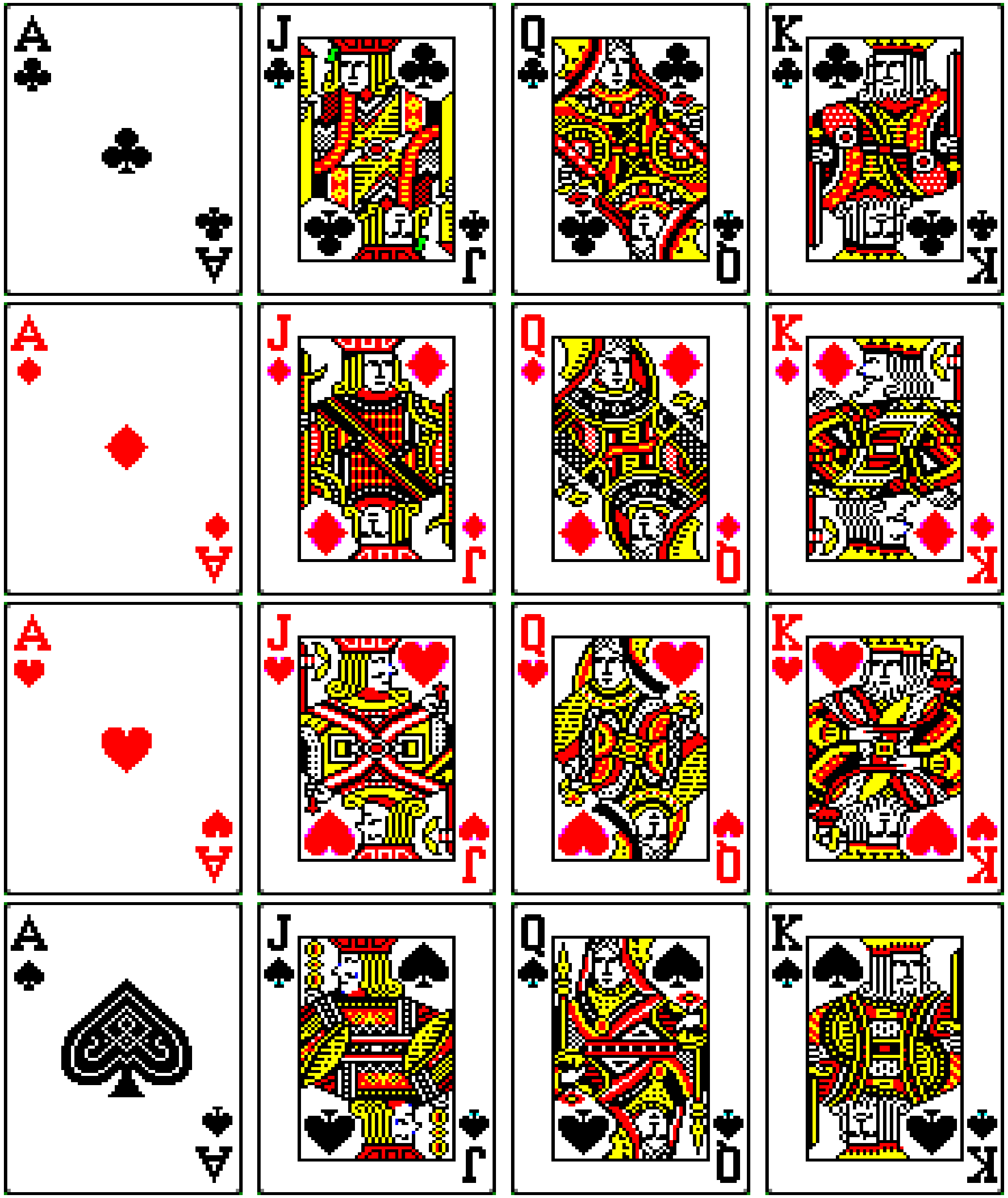
Numbering

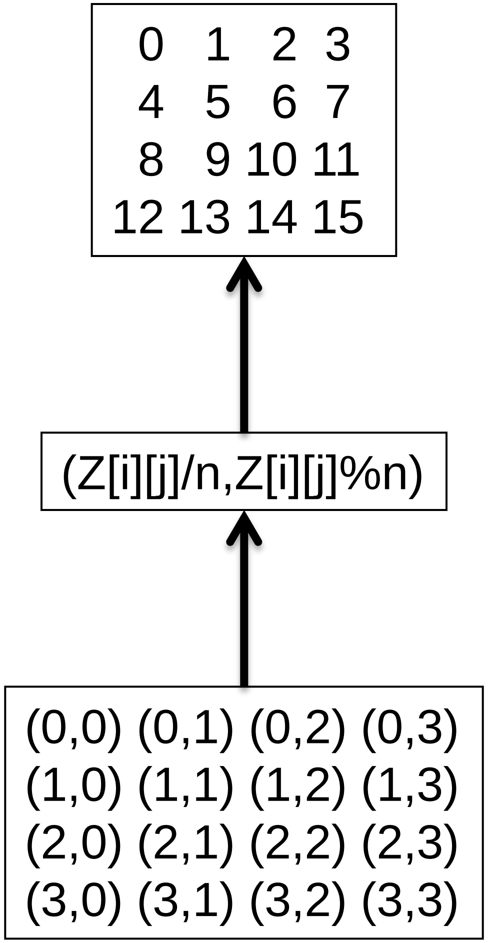


0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

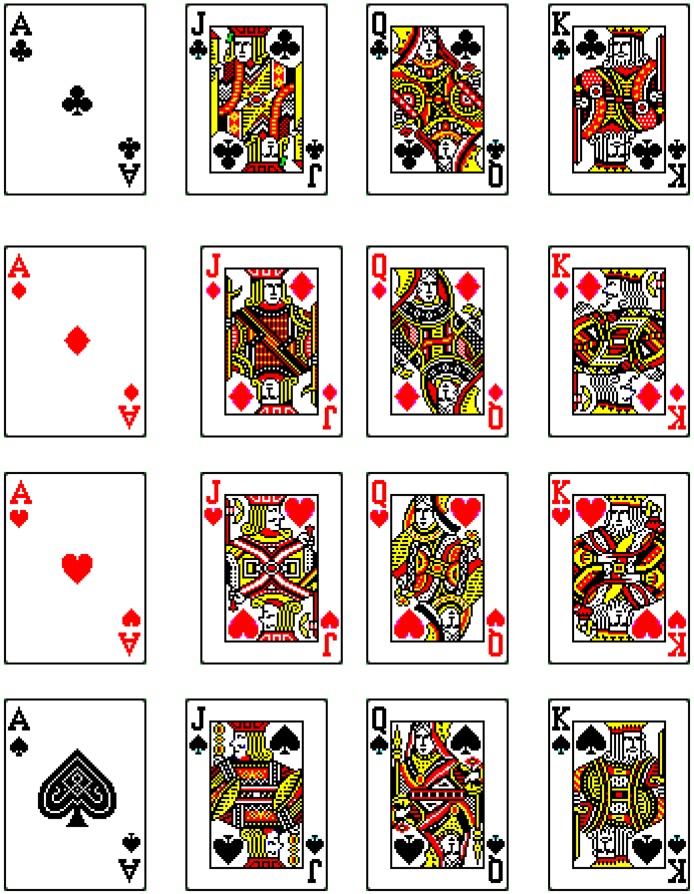


Numbering

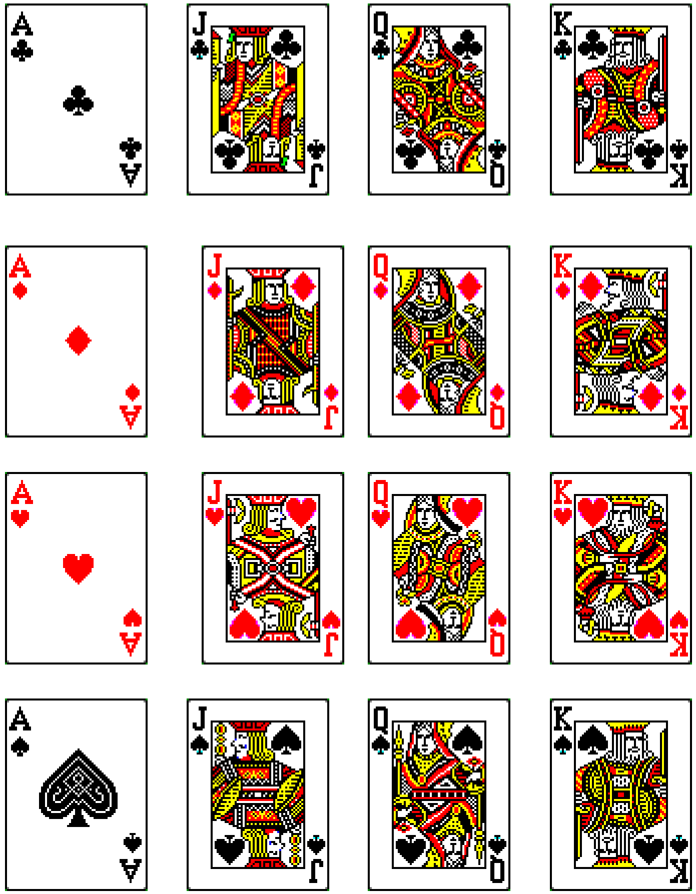



0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
(Z[i][j]/n,Z[i][j]%n)
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0) (2,1) (2,2) (2,3)
(3,0) (3,1) (3,2) (3,3)

Numbering

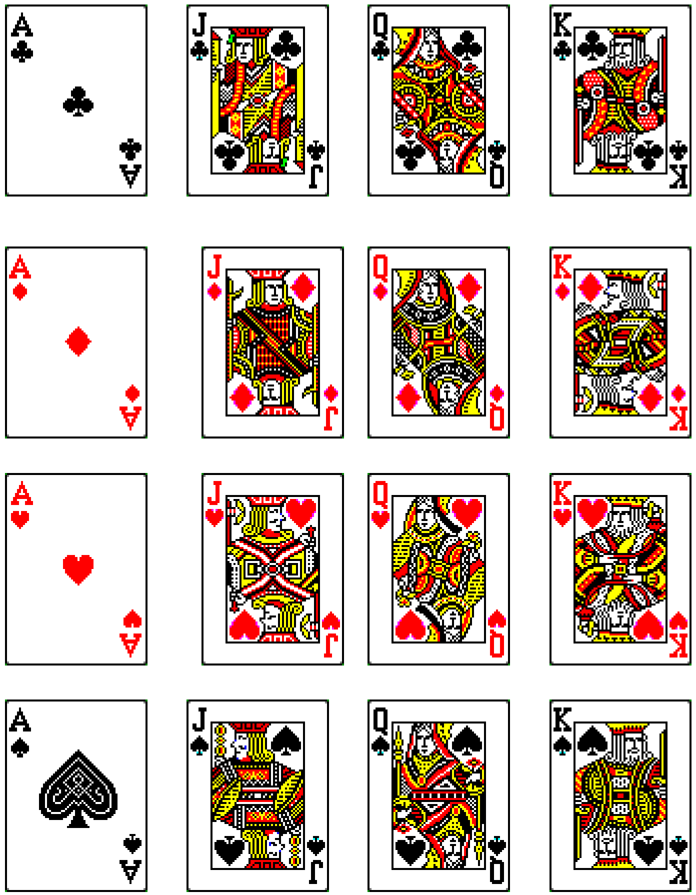


Solution n=4



0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

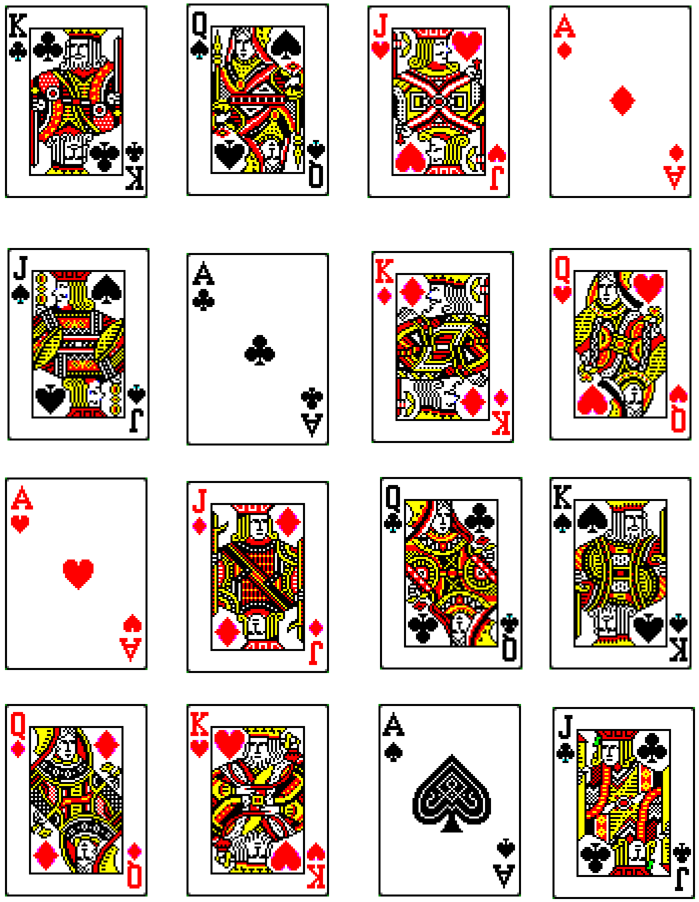
Solution n=4



0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

(0,0)	(1,1)	(2,2)	(3,3)
(1,2)	(0,3)	(3,0)	(2,1)
(2,3)	(3,2)	(0,1)	(1,0)
(3,1)	(2,0)	(1,3)	(0,2)

Solution n=4



0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

(0,0)	(1,1)	(2,2)	(3,3)
(1,2)	(0,3)	(3,0)	(2,1)
(2,3)	(3,2)	(0,1)	(1,0)
(3,1)	(2,0)	(1,3)	(0,2)

Model 0

Latin
All different in a row
All different in a column

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

X

Model 0

Greek
All different in a row
All different in a column

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

X

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

Y

Model 0

(0,0)	(1,1)	(2,2)	(3,3)
(1,2)	(0,3)	(3,0)	(2,1)
(2,3)	(3,2)	(0,1)	(1,0)
(3,1)	(2,0)	(1,3)	(0,2)

Graeco-Latin

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

X

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

Y

Model 0

Z

(0,0)	(1,1)	(2,2)	(3,3)
(1,2)	(0,3)	(3,0)	(2,1)
(2,3)	(3,2)	(0,1)	(1,0)
(3,1)	(2,0)	(1,3)	(0,2)

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

X

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

Y

Model 0

Z

0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

$Z[i][j] = 4.X[i][j] + Y[i][j]$

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

X

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

Y

```java
public class MOLS0  {

    public static void main(String args[]) {
        int n                          = Integer.parseInt(args[0]);
        Model model                    = new CPModel();
        IntegerVariable[][] Z    = makeIntVarArray("Z",n,n,0,n*n-1);
        IntegerVariable[][] Zt   = new IntegerVariable[n][n];
        IntegerVariable[][] X    = makeIntVarArray("X",n,n,0,n-1);
        IntegerVariable[][] Xt   = new IntegerVariable[n][n];
        IntegerVariable[][] Y    = makeIntVarArray("Y",n,n,0,n-1);
        IntegerVariable[][] Yt   = new IntegerVariable[n][n];
        IntegerVariable[] flatZ = new IntegerVariable[n*n];

        int k = 0;
        for (int i=0;i<n;i++)
            for (int j=0;j<n;j++){
                Zt[j][i] = Z[i][j];
                Xt[j][i] = X[i][j];
                Yt[j][i] = Y[i][j];
                flatZ[k] = Z[i][j];
                k++;
            }

        model.addConstraint(allDifferent(flatZ));
        for (int i=0;i<n;i++){
            model.addConstraint(allDifferent(X[i]));
            model.addConstraint(allDifferent(Xt[i]));
            model.addConstraint(allDifferent(Y[i]));
            model.addConstraint(allDifferent(Yt[i]));
        }

        // tie X and Y together
        for (int i=0;i<n;i++)
            for (int j=0;j<n;j++)
                model.addConstraint(eq(Z[i][j],plus(mult(X[i][j],n),Y[i][j])));
```

```
        Solver sol = new CPSolver();
        sol.read(model);
        IntDomainVar [] v = sol.getVar(flatZ);
        sol.setVarIntSelector(new MinDomain(sol,v));
        System.out.println("solved: " + sol.solve(false));
        for (int i=0;i<n;i++){
            for (int j=0;j<n;j++)
                System.out.print("(" + sol.getVar(X[i][j]).getVal()
                                 +","+ sol.getVar(Y[i][j]).getVal() +")");
            System.out.println();
        }
        sol.printRuntimeSatistics();

        // another view
        for (int i=0;i<n;i++){
            for (int j=0;j<n;j++)
                System.out.print(sol.getVar(Z[i][j]).getVal() +" ");
            System.out.println();
        }
    }
}
```

Symmetry

Model 1

Z

0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

Can fix first row

Z[i][j] = 4.X[i][j] + Y[i][j

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

X

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

Y

Symmetry

Model 1

Z

0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

Can fix first row

Z[i][j] = 4.X[i][j] + Y[i][j

Can fix first column

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

X

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

Y

```
// symmetry breaking
for (int j=0;j<n;j++){
    model.addConstraint(eq(X[0][j],j));
    model.addConstraint(eq(Y[0][j],j));
}
for (int i=1;i<n;i++) model.addConstraint(eq(X[i][0],i));
model.addConstraint(eq(Y[1][0],2));
```

Magic?

Model 2

Z

0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

Z[i][j] = 4.X[i][j] + Y[i][j]

Observe sum of each row and sum of each column

```
// magic square!
int sumZ = (n+1)*(n)*(n-1)/2;
for (int i=0;i<n;i++){
    model.addConstraint(eq(sum(Z[i]),sumZ));
    model.addConstraint(eq(sum(Zt[i]),sumZ));
}
```

Channeling

Model BMS

Z

0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

Flatten Z, Z’

0	5	10	15	6	3	12	9	11	14	1	4	13	8	7	2

Channeling

Z

Model BMS

0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

0	5	10	15	6	3	12	9	11	14	1	4	13	8	7	2

location

0	10	15	5	11	1	4	14	13	7	2	8	6	12	9	3

Channeling

Model BMS

Z

0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

0	5	10	15	6	3	12	9	11	14	1	4	13	8	7	2

$Z'[i] = x \leftrightarrow loc[x] = i$

Channeling constraint

0	10	15	5	11	1	4	14	13	7	2	8	6	12	9	3

Model BMS

Z

0	5	10	15
6	3	12	9
11	14	1	4
13	8	7	2

0	5	10	15	6	3	12	9	11	14	1	4	13	8	7	2

Z'[i] = x ↔ loc[x] = i

Channeling constraint

0	10	15	5	11	1	4	14	13	7	2	8	6	12	9	3

Can then do away with an allDiff on Z'