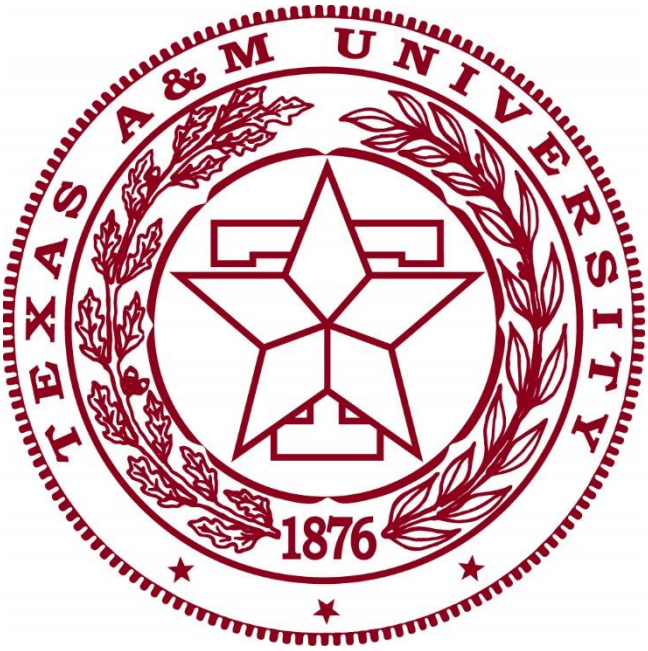
**ISEN 420 Project**

**for**

**Dr. Erick Moreno-Centeno**

**Phase 3 Final Report, Team 06**

**http://mcilvoy.org/420**




**Team Members:**

Zachary McIlvoy

Erik Villeda

****SPECIAL: Team 6 now has only two members, and thus you told us that our instruction manual would not be graded for looks and only for content****

# Table of Contents

## Executive Summary

This is the final report on the development of automated solvers for two BrainBashers.com puzzles. There are two different puzzles to be solved, both with unique challenges. We approached them in similar ways. First developed a model for solving the puzzles, then converted that to computer code to automate solving, then polished all of the information in to a convenient solver for the end user. Included in this report is the development of the model, the computer code, and a user guide on how to use the final product for solving puzzles. Examples are given for each, and any puzzle can easily input and solved with one simple command.

## Technical Report

### Problem #1 – "3-in-a-Row"

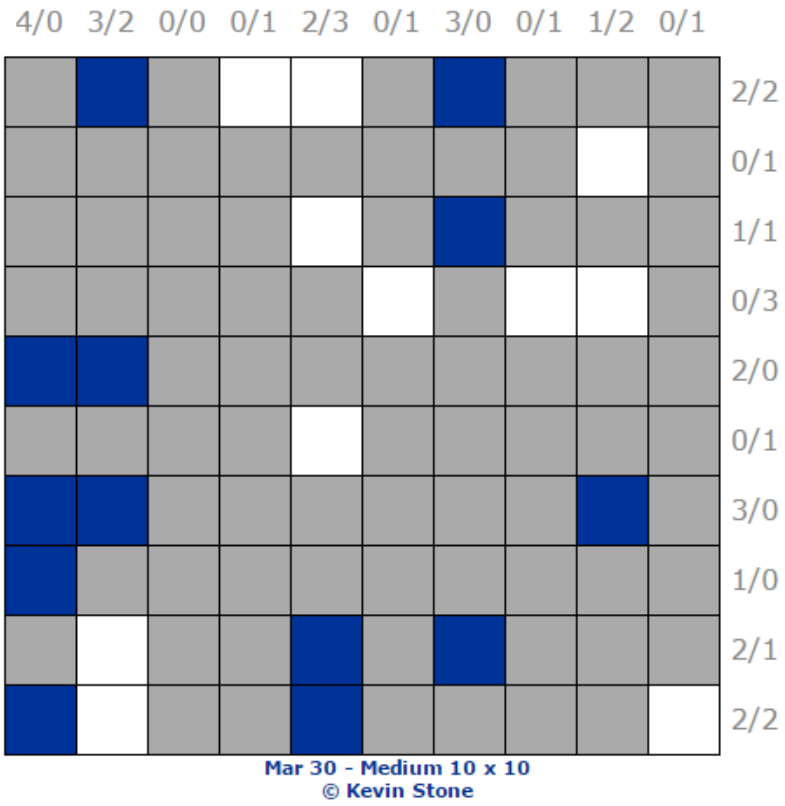
The puzzle can be found at *http://www.brainbashers.com/3inarow.asp*

Sources referenced:

http://gssf.gdansk.pl/upload/497_Olszowy_Wiktor_Sudoku.pdf

http://www.hakank.org/ampl/sudoku_cp.mod

Example puzzle:




---

Explanation of the Rules

There are only three rules to this puzzle:

1. The grid must be filled with blue and white squares
2. There may not be three in a row of white or blue in any row or column (diagonal is fine)
3. Each row and column must have an equal number of blue squares and white squares

## Overview of Problem Approach

**Objective function:** 0

There is no function being solved in this case, as we aren't minimizing or maximizing anything, just finding a solution that satisfies all constraints.

**Parameters:**

n (the length of one side of the given puzzle (in 'squares')

$p_{i,j,k}$ *i and j from 1 to n* (the given puzzle board)

P is a three-dimensional matrix representing the starting board, where i and j represent a given coordinate on the board, and k represents what value that square should take of 0, 1 or 2 for grey, blue, or white.

**Variables:**

$x_{i,j,k}$ *i and j go from 1 to n* (the board that will contain the answers)

*k goes from 1 to 2*

*Binary: x[i,j,k] = 1 if value k is found in cell [i,j], 0 otherwise*

The variable x will be the final answer grid, where again, i and j represent a given coordinate on the board, and k represents what value that square should take of 1 or 2 for blue or white, since the answer cannot contain grey squares. If the value of a given i,j pair has the value of k, then the value of x for that i,j,k becomes 1. The final answer outputs the x where all 1s represent white squares.

**Constraints:**

$x_{i,j,k} + x_{i+1,j,k} + x_{i+2,j,k} \leq 2$ (No three of the same color in any given row should be consecutive)

*i from 1 to n-2, j from 1 to n, k from 1 to 2*

Since the size of the game board doesn't affect the 3-in-a-row constraint, we iterate across each row, check the three contiguous squares, and ensure that the sum is less than or equal to 2. Since x is binary, anything more than 2 means that there are values in all three squares that are the same.

$x_{i,j,k} + x_{i,j+1,k} + x_{i,j+2,k} \leq 2$ (No three of the same color in any given column should be consecutive)

*i from 1 to n, j from 1 to n-2, k from 1 to 2*

Since the size of the game board doesn't affect the 3-in-a-row constraint, we iterate across each column, check the three contiguous squares, and ensure that the sum is less than or equal to 2. Since x is binary, anything more than 2 means that there are values in all three squares that are the same.

$\sum_{j=1}^{n} x_{i,j,1} = \sum_{j=1}^{n} x_{i,j,2}$ (Equal number of colors in a given row)

*i from 1 to n*

Again, since x is binary, we can sum x for the two values of k, and ensure that they are equal, ensuring that we have an equal amount of blue and white squares in a given row.

$\sum_{i=1}^{n} x_{i,j,1} = \sum_{i=1}^{m} x_{i,j,2}$ (Equal number of colors in a given column)

*j from 1 to n*

Here we flip the above constraint, summing across columns, ensuring that we have an equal amount of blue and white squares.

$\sum_{k=1}^{2} x_{i,j,k} = 1$ (Must solve all squares)

*i from 1 to n,* *j from 1 to n*

This constraint checks x for each cell for each value of k, that it exactly equals one, thus that it cannot still be grey, and it must either be only white or only blue.

$x_{i,j,p_{i,j}} = 1$ (Given points are fixed)

*i from 1 to n,* *j from 1 to n* | $p_{i,j} \neq 0$

For each coordinate square in x, set it's value equal to the value given in the initial board parameter p, for all p coordinates not set to 0. That is, make sure the solution includes the given constraint of the game board initial setup.

p is an integer (integer values represent colors)

**Problem #2 – "ABC Path"**

The puzzle can be found at https://www.brainbashers.com/showabcpath.asp

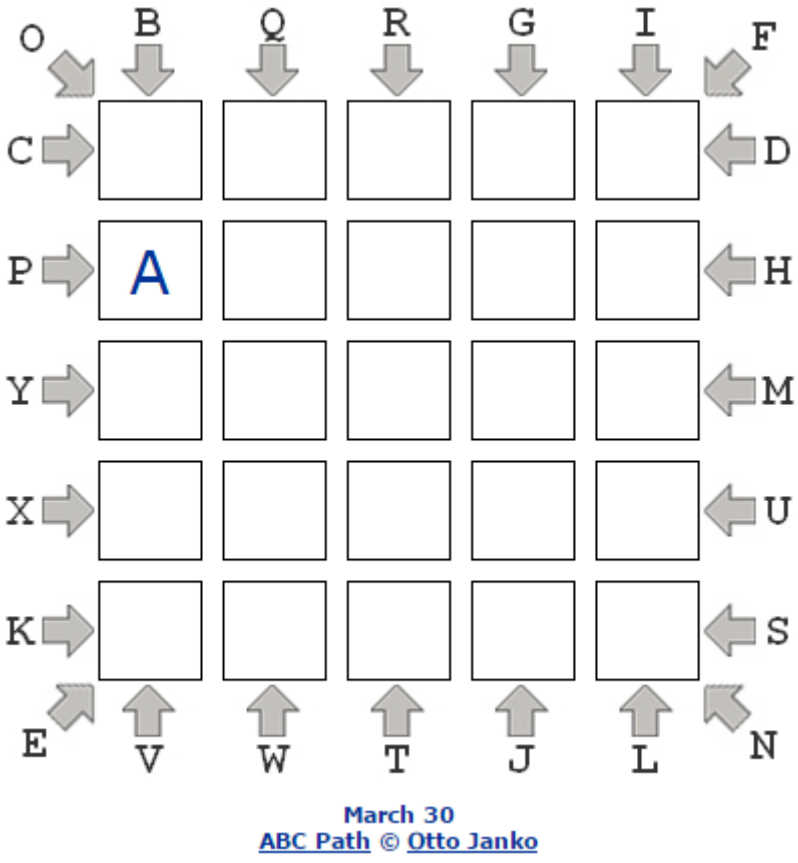
Sources referenced:

http://gssf.gdansk.pl/upload/497_Olszowy_Wiktor_Sudoku.pdf

http://www.hakank.org/ampl/sudoku_cp.mod

Dr. Erick Moreno-Centeno's ABC Path Solution from "ISEN 420 – Solution to Project Phase 1"

Example puzzle:



## Explanation of the Rules

Again, there are three rules to the puzzle:

1. Every letter from A to Y must be on the grid
2. Each letter must be next to the previous letter (in alphabetical order) either horizontally, vertically, or diagonally
3. The letters around the outside of the board constrain which row, column, or diagonal that letter must be in

## Overview of Problem Approach

**Objective Function:** 0

Again, there is no function being solved in this case, as we aren't minimizing or maximizing anything, just finding a solution that satisfies all constraints.

**Parameters:**

m (grid size)

m is the row/column size of the grid, in this case it is always fixed at 5

n=m*m (maximum path length)

A given path cannot be longer than the total number of available squares on the board.

M= {1..m} (set for row/column)

A row can have values stored anywhere between 1 and the board side length

N={1..n} (set for all path values)

A path can have values stored anywhere between 1 and the total board size.

Ar (given initial letter row location)

This is the row number of the initial placement of "A"

Ac (given initial letter column location)

This is the column number of the initial placement of "A"

$clue_{k,i,j}$ (given perimeter clues)

*k from 2..n, i from 0 to m+1, j from 0 to m+1*

*binary: take the value 1 if letter k can be in cell i,j*

Clue is a binary 3-dimensional matrix representing whether or not a given cell can contain a given letter, k

**Variables:**

$x_{k,i,j}$ (final solution grid)

*k from 1..n, i from 0 to m+1, j from 0 to m+1*

*Binary: takes the value 1 when letter k goes in cell i,j*

The variable x holds the final location of letters. Its size is the grid plus one to allow for checking of adjacency. It takes the value 1 if letter k goes in cell i,j

**Constraints:**

Ar & Ac are integers (must be integers to correspond to row/col.)

Clue & X are binary (must be binary)

$x_{k,0,j} = 0$ for j from 0 to m+1

$x_{k,m+1,j} = 0$ for j from 0 to m+1

$x_{k,i,0} = 0$ for i from 0 to m+1

$x_{k,i,m+1} = 0$ for i from 0 to m+1

Right, Left, Top, and Bottom border rows must be zero, as they are only for equation purposes and may not take actual values.

$x_{1,Ar,Ac,}$ (Initial A Placement)

The initial location of A placed in the answer variable

$\sum_{k=1}^{n} x_{k,i,j} = 1$ (Only one letter in each cell)

*i from 1 to m, c from 1 to m*

For each i,j cell location, only one letter may exist, so the binary variable must then equal exactly one

$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{k,i,j} = 1$ (every location must be filled)

*k from 1 to n*

The sum across the whole board (i,j), for each k value (letter) must equal 1. This ensures that there is one and only one of each letter on the board.

$(\sum_{g=-1}^{1} \sum_{h=-1}^{1} x_{k+1,i+g,j+h}) - x_{k+1,i,j} \geq x_{k,i,j}$ (Consecutive letters adjacent to each other)

*K from 1 to n-1, i from 1 to m, j from 1 to m*

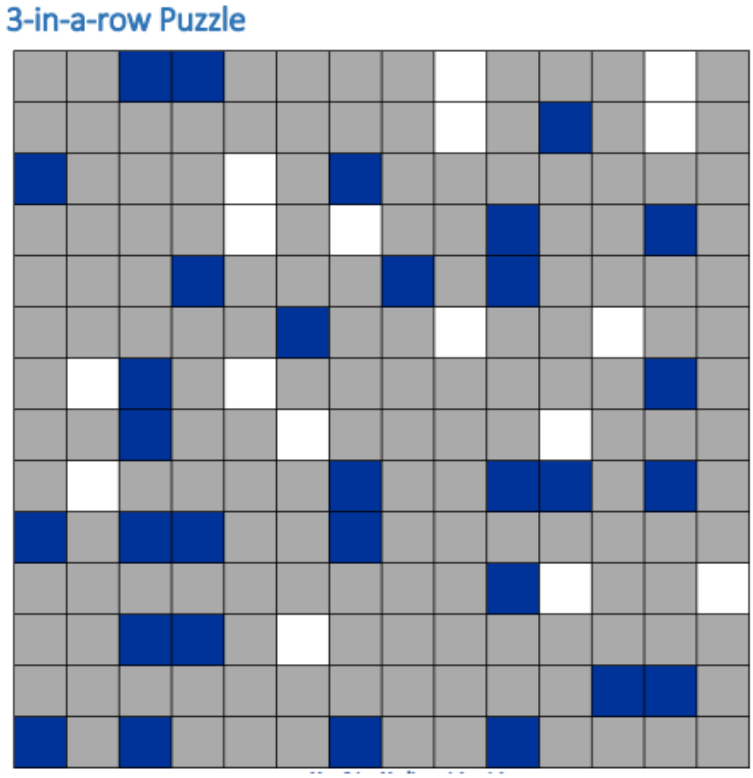
This constraint restricts the location of the next letter (k+1) to be adjacent to the location of the currently letter, k.

## Results

### Problem #1 – "3-in-a-Row"

AMPL

**For this report, we are solving the given test case:**



Model:

```
param n;
param p{i in 1..n, j in 1..n}default 0, integer, in 0..2,;
var x{i in 1..n, j in 1..n, k in 1..2}binary;       #=1 if value k is found in cell i,j
minimize nothing: 0;
subject to
NoThreeInARowROW{i in 1..n-2, j in 1..n, k in 1..2}: x[i,j,k] + x[i+1,j,k] + x[i+2,j,k] <= 2;
NoThreeInARowCOL{i in 1..n, j in 1..n-2, k in 1..2}: x[i,j,k] + x[i,j+1,k] + x[i,j+2,k] <= 2;
EqualNumbersROW{i in 1..n}: sum{j in 1..n}x[i,j,1] = sum{j in 1..n}x[i,j,2];
EqualNumbersCOL{j in 1..n}: sum{i in 1..n}x[i,j,1] = sum{i in 1..n}x[i,j,2];
MustSolveAllSquares{i in 1..n, j in 1..n}: sum{k in 1..2}x[i,j,k] = 1;
FixedDataPoints{i in 1..n, j in 1..n: p[i,j] <> 0}: x[i,j,p[i,j]]=1;
```

Data:

*param n:=14;*

*param p:  1 2 3 4 5 6 7 8 9 10 11 12 13 14:=*

*1 . . 2 2 . . . . 1 . . . 1 .*

*2 . . . . . . . . 1 . 2 . 1 .*

*3 2 . . . 1 . 2 . . . . . . .*

*4 . . . . 1 . 1 . . 2 . . 2 .*

*5 . . . 2 . . . 2 . 2 . . . .*

*6 . . . . . 2 . . 1 . . 1 . .*

*7 . 1 2 . 1 . . . . . . . 2 .*

*8 . . 2 . . 1 . . . . 1 . . .*

*9 . 1 . . . . 2 . . 2 2 . 2 .*

*10 2 . 2 2 . . 2 . . . . . . .*

*11 . . . . . . . . . 2 1 . . 1*

*12 . . 2 2 . 1 . . . . . . . .*

*13 . . . . . . . . . . . 2 2 .*

*14 2 . 2 . . . 2 . . 2 . . . .;*

This data file has periods as grey spaces, ones as white spaces, and twos as blue spaces.

To directly download copies of the files, click here:

Model File

Data File

The model is run in AMPL using the following commands:

ampl

option solver cplex;

model p2t06mod3.txt;

data p2t06dat3.txt;

solve;

display {i in 1..n, j in 1..n, k in 2..2} x[i,j,k];

## Results

This series of commands will solve then display the answer in the following format:

Ones are blue squares, zeroes are white squares

After running in AMPL, the output is the following variable(color coded for clarity):

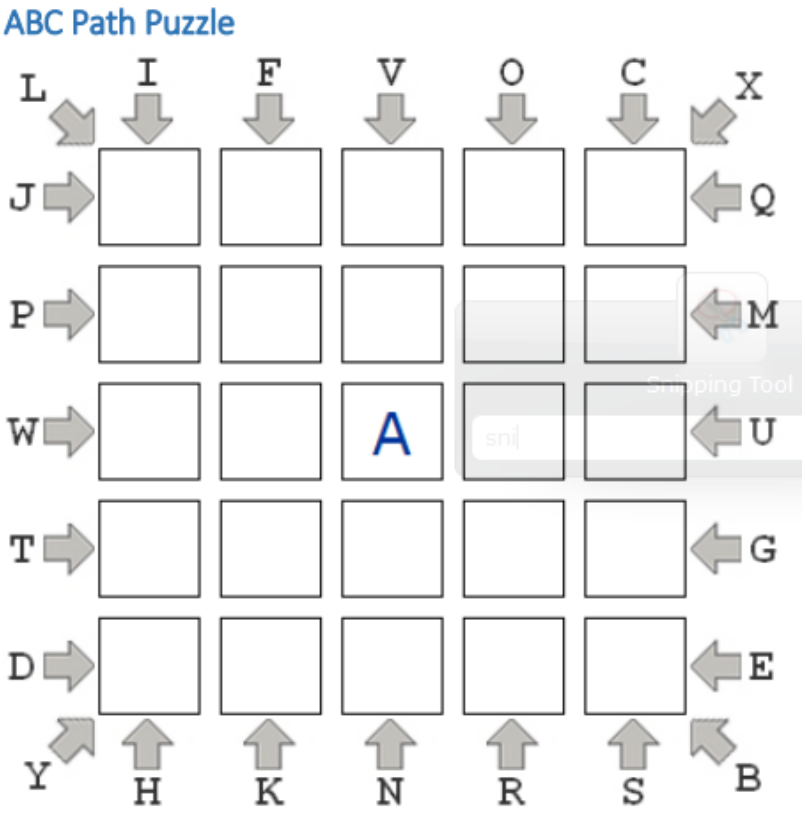
x[i,j,k] [*,*,2]

| : | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | := |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| **2** | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | |
| **3** | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| **4** | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | |
| **5** | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | |
| **6** | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | |
| **7** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | |
| **8** | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | |
| **9** | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | |
| **10** | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | |
| **11** | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | |
| **12** | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | |
| **13** | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | |
| **14** | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0; | |

**Problem #2 – "ABC Path"**

AMPL

**For this report we solved the given test case:**



Model:

*param m;  #number of rows/columns*

*param n:=m*m;  #number of letters*

*set M := {1..m};  #set for row/column*

*set N := {1..n};  #set for letters*

*param Ar;#Row*

*param Ac;#Column*

*param clue{k in 2..n, i in M,j in M}binary, default 0; #matrix w/ numbers N representing letters*

*var x{k in 1..n, i in 0..m+1,j in 0..m+1}binary; #1 if letter k is in cell (i,j)*

*#Binary definition shown above_____^^^^^^*

*minimize dummy: 0;*

*subject to*

*A_Placement: x[1,Ar,Ac] = 1;*

*LimitLetterPlacement{k in 2..n, i in M, j in M}: x[k,i,j] <= clue[k,i,j];*

*OnlyOneLetterPerCell{i in M, j in M}: sum{k in N} x[k,i,j] = 1;*

*EachLetterinGrid{k in N}: sum{i in M, j in M} x[k,i,j] = 1;*

*SnakePathway{k in 1..n-1, i in M, j in M}: (sum{g in -1..1, h in -1..1} x[k+1,i+g,j+h])-x[k+1,i,j] >= x[k,i,j];*

*LeftBorder{k in 1..n-1, i in 0..m+1}: x[k,i,0] = 0;*

*RightBorder{k in 1..n-1, i in 0..m+1}: x[k,i,m+1] = 0;*

*TopBorder{k in 1..n-1, j in 0..m+1}: x[k,0,j] = 0;*

*BottomBorder{k in 1..n-1,j in 0..m+1}: x[k,m+1,j] = 0;*

Data:

*param m:=5;*

*param Ar:= 3; #row Placeholder*

*param Ac:= 3; #column Placeholder*

*param clue:=*

*#B*
*[2,1,1] 1*
*[2,1,2] 0*
*[2,2,2] 1*

*[2,3,3] 1*
*[2,4,4] 1*

*[2,5,5] 1*
*#C*
*[3,1,5] 1*
*[3,2,5] 1*
*[3,3,5] 1*
*[3,4,5] 1*
*[3,5,5] 1*
*#D*
*[4,5,1] 1*
*[4,5,2] 1*
*[4,5,3] 1*
*[4,5,4] 1*
*[4,5,5] 1*
*#E*
*[5,5,1] 1*
*[5,5,2] 1*
*[5,5,3] 1*
*[5,5,4] 1*
*[5,5,5] 1*
*#F*
*[6,1,2] 1*
*[6,2,2] 1*
*[6,3,2] 1*
*[6,4,2] 1*
*[6,5,2] 1*
*#G*
*[7,4,1] 1*
*[7,4,2] 1*

*[7,4,3] 1*
*[7,4,4] 1*
*[7,4,5] 1*
*#H*
*[8,1,1] 1*
*[8,2,1] 1*
*[8,3,1] 1*
*[8,4,1] 1*
*[8,5,1] 1*
*#I*
*[9,1,1] 1*
*[9,2,1] 1*
*[9,3,1] 1*
*[9,4,1] 1*
*[9,5,1] 1*
*#J*
*[10,1,1] 1*
*[10,1,2] 1*
*[10,1,3] 1*
*[10,1,4] 1*
*[10,1,5] 1*
*#K*
*[11,1,2] 1*
*[11,2,2] 1*
*[11,3,2] 1*
*[11,4,2] 1*
*[11,5,2] 1*
*#L*
*[12,1,1] 1*
*[12,2,2] 1*
*[12,3,3] 1*
*[12,4,4] 1*
*[12,5,5] 1*
*#M*
*[13,2,1] 1*
*[13,2,2] 1*
*[13,2,3] 1*
*[13,2,4] 1*
*[13,2,5] 1*
*#N*
*[14,1,3] 1*
*[14,2,3] 1*
*[14,3,3] 1*
*[14,4,3] 1*
*[14,5,3] 1*
*#O*

*[15,1,4] 1*
*[15,2,4] 1*
*[15,3,4] 1*
*[15,4,4] 1*
*[15,5,4] 1*
*#P*
*[16,2,1] 1*
*[16,2,2] 1*
*[16,2,3] 1*
*[16,2,4] 1*
*[16,2,5] 1*
*#Q*
*[17,1,1] 1*
*[17,1,2] 1*
*[17,1,3] 1*
*[17,1,4] 1*
*[17,1,5] 1*
*#R*
*[18,1,4] 1*
*[18,2,4] 1*
*[18,3,4] 1*
*[18,4,4] 1*
*[18,5,4] 1*
*#S*
*[19,1,5] 1*
*[19,2,5] 1*
*[19,3,5] 1*
*[19,4,5] 1*
*[19,5,5] 1*
*#T*
*[20,4,1] 1*
*[20,4,2] 1*
*[20,4,3] 1*
*[20,4,4] 1*
*[20,4,5] 1*
*#U*
*[21,3,1] 1*
*[21,3,2] 1*
*[21,3,3] 1*
*[21,3,4] 1*
*[21,3,5] 1*
*#V*
*[22,1,3] 1*
*[22,2,3] 1*
*[22,3,3] 1*
*[22,4,3] 1*

*[22,5,3] 1*
*#W*
*[23,3,1] 1*
*[23,3,2] 1*
*[23,3,3] 1*
*[23,3,4] 1*
*[23,3,5] 1*
*#X*
*[24,1,5] 1*
*[24,2,4] 1*
*[24,3,3] 1*
*[24,4,2] 1*
*[24,5,1] 1*
*#Y*
*[25,1,5] 1*
*[25,2,4] 1*
*[25,3,3] 1*
*[25,4,2] 1*
*[25,5,1] 1;*

To directly download copies of the files, click here:

Model File

Data File

The model is run in AMPL using the following commands:

ampl

option solver cplex;

model p2t06modA.txt;

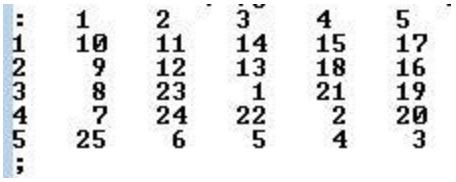
data p2t06datA.txt;

solve;

display x;

## Results:

The results of this approach are long and confusing. Using some simplifying display commands as seen in the user guide, the output is as follows, with each letter represented by a number:



```
:    1    2    3    4    5 :=
1   10   11   14   15   17
2    9   12   13   18   16
3    8   23    1   21   19
4    7   24   22    2   20
5   25    6    5    4    3
;
```

Given is the translation to a final grid from the AMPL output.

| | | | | |
|---|---|---|---|---|
| J | K | N | O | Q |
| I | L | M | R | P |
| H | W | A | U | S |
| G | X | V | B | T |
| Y | F | E | D | C |

## Conclusion

These models are able to take the correct inputs for the puzzle, in these cases in the form of matrices and board size, and return a filled matrix with the solution. In this way, we can take advantage of generalized IPs and computer solvers to find the solution to and variation of either of these puzzles. The user guide below simplifies this process for the user focused only on arriving at a solution.

## User Manual

So you need help solving some BrainBashers puzzles huh? Read on to find out how to use our helper which will solve them all for you!

STEP 1

First we need to download the software. Click HERE to download the folder you need.

STEP 2

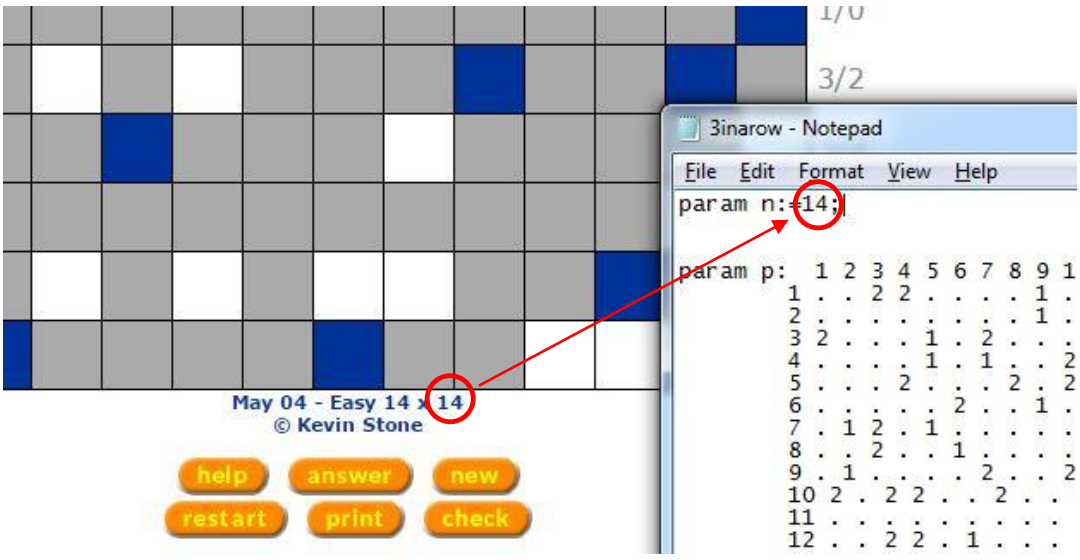
Now we need to tell the computer what your puzzle looks like. Open that folder that you just downloaded, then double click on the folder called "EDIT_ME". Here you will see two other files, one for 3-In-a-Row and one for ABC Path type puzzles.

Let's look at what you need to change to tell the computer what your puzzle looks like.

3-In-a-Row:

There are two parts you need to change:

The board size is the first and easiest. Change the part that says "CHANGE THIS BOARD SIZE" to the number shown from BrainBashers.com.




Now, you are going to have to input what the whole board looks like.

First change the numbers along the top and left side to only go as high as the first number that we just entered. For example, in this case the board size was 14, so our two sections go all the way to 14:

```
param n:=CHANGE THIS BOARDSIZE;

param p:  1 2 3 4 5 6 7 8 9 10 11 12 13 14:=
       1  ***CHANGE ALL OF THIS***
       2  . . . . . . . . 1 . 2 . 1 .
       3  2 . . . 1 . 2 . . . . . . .
       4  . . . . 1 . 1 . . 2 . . 2 .
       5  . . . 2 . . . 2 . 2 . . . .
       6  . . . . . 2 . . 1 . . 1 . .
       7  . 1 2 . 1 . . . . . . . 2 .
       8  . . 2 . . 1 . . . . 1 . . .
       9  . 1 . . . . 2 . . 2 2 . 2 .
       10 2 . 2 2 . . 2 . . . . . . .
       11 . . . . . . . . . 2 1 . . 1
       12 . . 2 2 . 1 . . . . . . . .
       13 . . . . . . . . . . . 2 2 .
       14 2 . 2 . . . 2 . . 2 . . . .;
```

Now comes the tricky part.

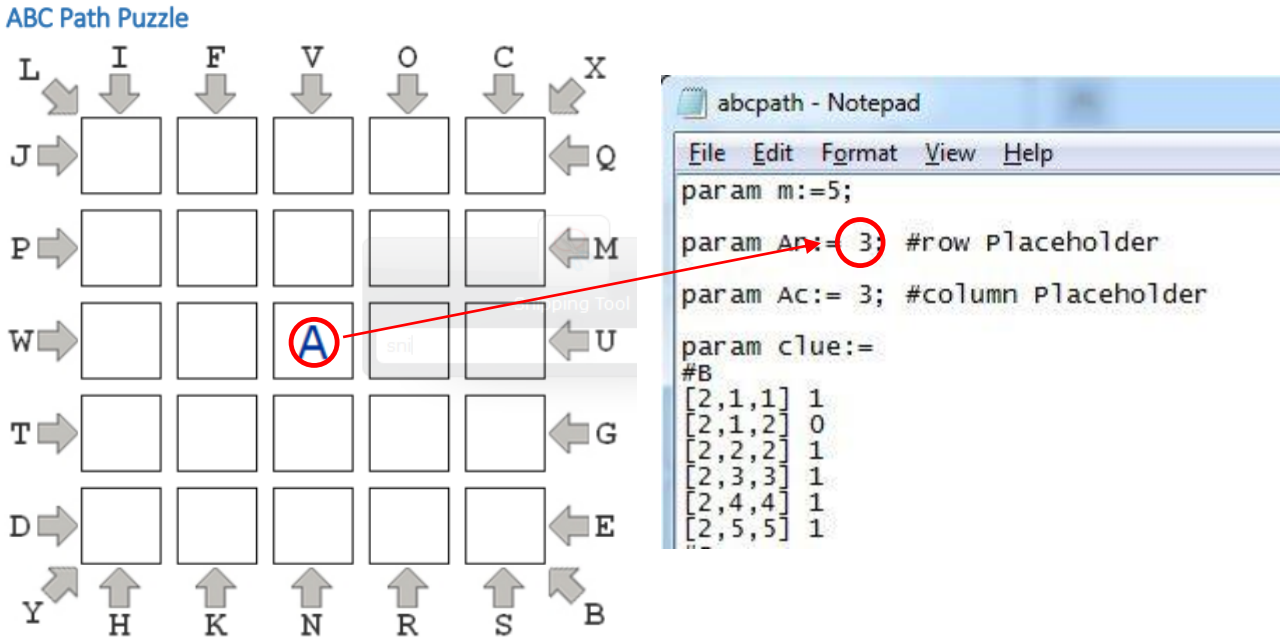
Next to each number on the left, you will need to put one symbol for every square on your puzzle. A period is for a grey square, and 1 is for a white square, and a 2 is for a blue square. Between each is a space. For example, if your puzzle on the first row has grey, grey, blue, white, next to the 1, you would type “period, SPACE, period, SPACE, 1, SPACE, 2” or “. . 1 2”

Make sure you have the same number of symbols as your board size.

Make sure after your very last one you leave that “;” there. It’s very important!

Most of your data is already there because they will be periods, so just replace the ones you need to and make sure to follow the pattern!

ABC Path:




First, change the two numbers of the initial location of A. The two locations are marked "AROW" and "ACOLUMN". Fill them with the corresponding values by counting down and right from the top left corner. See example above.

Now to input the clues.

Each letter is assigned a certain column or row or corner. For example, above, D is row 5, K is column 2, and B would be a corner point. Follow this template:

Row Clue:

[letter value, assigned row, 1-5(1)]1

[letter value, assigned row, 2]1

[letter value, assigned row, 3]1

[letter value, assigned row, 4]1

[letter value, assigned row, 5]1

Column Clue:

[letter value, 1-5(1), assigned column]1

[letter value, 2, assigned column]1

[letter value, 3, assigned column]1

[letter value, 4, assigned column]1

[letter value, 5, assigned column]1

Corner Clue:

Corner clues are marked in the file. Simply change out the letter value with the corresponding corner for your clue/letter.

Again, much is already typed in, just change the values you need for all constraints.

STEP 3

Time to get the answer! Open that folder and double click on icon called “RUN_ME”.

First type “ampl” and then press enter. Now type one of two things:

If you need to solve a “3-in-a-row” puzzle, type “include run3inarow.run;”, then press enter

If you need to solve an “ABC Path” puzzle, type “include runabc.run;”, then press enter

Those commands will do all of your hard work and output the answers you are looking for! Let look at how to understand what it is telling you.

(To see it run with data entered, do the same process but from within the SEEITWORK folder)

STEP 4

Confused? Here’s what those numbers mean if it’s still unclear.

For the 3-In-a-Row puzzle:

Ignore the number along the outside, they are the size of the grid just like before. We want to look at just the inside numbers where all the zeroes and ones are. When you go to put in your answer on BrainBashers, just look at this and remember that if there is a 1 there, make that square blue. If there is a 0 there, make that square white. It’s that simple.

ABC Path:

The output really only requires counting. Each letter is represented by a number from 1 to 25. A=1, B=2, C=3, etc. Y=25. Simply follow the numbers around the grid and replace with the sequential letter for that value.