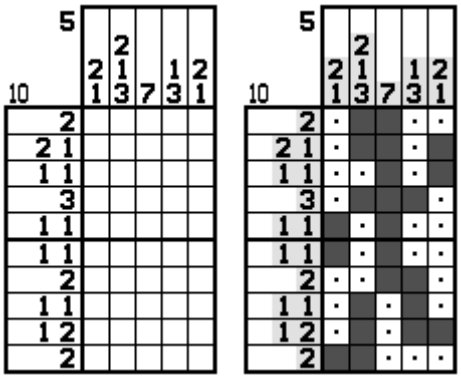
CSE 494 Jake Griffin

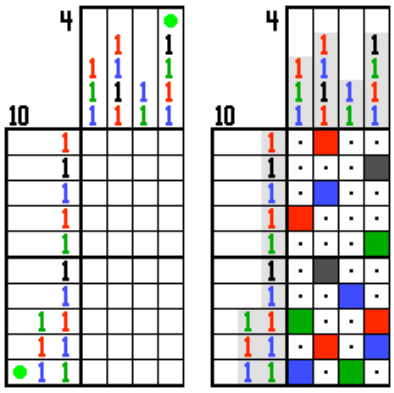
Programming Project 2008-12-10

Paint by Number

Project Description



There is a type of puzzle called Paint by Number. The "unsolved" state of this puzzle consists of a rectangular grid of cells that are to be either filled in with a color or left blank. To determine which cells need to be colored which colors, there is a set of "row clues" and "column clues" along the left and top edges of the grid respectively. Each row or column has a set of numbers associated with it and each number is a "clue" as to how to fill in the cells of that row or column. For example, if a row has the clues "3, 2", then in that row, there is a group of three consecutive black (filled in) cells, and somewhere to the right of that group of cells is another group of two consecutive black cells, and all other cells are white (unfilled). Note that there must be at least one white cell between these two groups of cells, or else the clue would be "5" instead.



In the most general type of these puzzles, the clues are all black; however there is a variation that allows clues to be colored. For example, if a column has the clues "6, 6, 2." This means that in that column, there is a group of 6 consecutive blue cells, and somewhere below that is a group of 6 consecutive red cells, and somewhere below that is a group of 2 consecutive red cells. Note that in this case, there is at least one white cell between the 6 red cells and 2 red cells, but not necessarily between the 6 blue cells and 6 red cells because the differing colors is enough to distinguish them. All other cells in the column are to be white. If the idea of how Paint by Number puzzles work is still not clear, there are some examples of unsolved Paint by Number puzzles and their corresponding solved puzzles in the Appendix.

Through repeated use of logic, some puzzles can be filled in uniquely, while others may have multiple solutions. My project is to determine whether a given puzzle has a unique solution. Given a filled in grid, my program will generate the "clues," then will use these clues to try to solve the puzzle in a different way than was originally given. If a different solution is found, then one can conclude that the puzzle does not have a unique solution.

Background

This is an interesting project for me because I have completed many puzzles on www.webpbn.com (currently over 3,500 puzzles available as of this writing) and found that quite a few are actually solvable in more than one way, and therefore, they require some guesswork in order to solve them. Often, I get near the end of a puzzle before I get stuck in a position and realize that there are actually multiple solutions, and that takes a lot of the fun out of the puzzle (working through logically is a lot more fun in my opinion).

Current Work

I started out writing a solver in ASP that solves black and white puzzles, then modified this code in order to handle the more complex multi-colored puzzles. My algorithm follows a sort of Generate-Define-Test structure.

The input to the solver program is a set of row blocks and column blocks given in the following form:

```
br(R, B). % There are B blocks (clues) for row R

bc(C, B). % There are B blocks (clues) for column C

lr(R, B, L). % The Bth block in the Rth row is of length L

lc(C, B, L). % The Bth block in the Cth column is of length L

cr(R, B, C). % The Bth block in the Rth row is of color C

cc(C, B, C). % The Bth block in the Cth column is of color C
```

To solve the puzzle, I first set up the domains for the problem:

```
startr(0..columns-1). % Starting point for row clues

startc(0..rows-1). % Starting point for column clues

row(1..rows). % Row indices

column(1..columns). % Column indices

color(1..colors). % Color indices

#domain row(Y). % Y denotes a row

#domain column(X). % X denotes a column

#domain color(C). % C denotes a color

cell(X, Y). % There is a cell at every (X, Y) coordinate
```

The **generate** part of the program enumerates all starting points for each row and column block (along with some **constraints** that keep them overlapping or hanging off of the grid) and also sets each cell to either filled (with some color) or dotted (white):

```
1 { dot(X, Y), fill(X, Y, N) : color(N) } 1. % Every cell is filled or dotted

1 { sr(Y, B, S) : startr(S) } 1 :- lr(Y, B, L). % Every given block in a row has some start cell

1 { sc(X, B, S) : startc(S) } 1 :- lc(X, B, L). % Every given block in a column has some start cell

:- sr(Y, B, S), lr(Y, B, L), S+L>columns, startr(S). % The end of a block in a row cannot hang off the end of the row

:- sc(X, B, S), lc(X, B, L), S+L>rows, startc(S). % The end of a block in a column cannot hang off the end of the column

:- sr(Y, B, S1), lr(Y, B, L), sr(Y, B+1, S2), S1+L>=S2, startr(S1;S2), cr(Y, B, C), cr(Y, B+1, C). % Each block in a row is separated by at least one space if they are the same color

:- sc(X, B, S1), lc(X, B, L), sc(X, B+1, S2), S1+L>=S2, startc(S1;S2), cc(X, B, C), cc(X, B+1, C). % Each block in a column is separated by at least one space if they are the same color

:- sr(Y, B, S1), lr(Y, B, L), sr(Y, B+1, S2), S1+L>S2, startr(S1;S2). % No two blocks in a row overlap

:- sc(X, B, S1), lc(X, B, L), sc(X, B+1, S2), S1+L>S2, startc(S1;S2). % No two blocks in a column overlap
```

A cell of the grid is **constrained** to be the same color of a block covering it in its row or column and white if there is not a block covering it in its row or column:

```
:- fill(X, Y, C), sr(Y, 1, S), S>=X, cell(X, Y), startr(S). % Cells before the first block in a row are not filled

:- fill(X, Y, C), sc(X, 1, S), S>=Y, cell(X, Y), startc(S). % Cells before the first block in a column are not filled

:- fill(X, Y, C), sr(Y, B, S), br(Y, B), lr(Y, B, L), S+L<X, cell(X, Y), startr(S). % Cells after the last block in a row are not filled

:- fill(X, Y, C), sc(X, B, S), bc(X, B), lc(X, B, L), S+L<Y, cell(X, Y), startc(S). % Cells after the last block in a column are not filled

:- fill(R, C), sr(R, B, S1), lr(R, B, L), sr(R, B+1, S2), S1+L<C, C<=S2, cell(R, C), startr(S1;S2). % Cells between blocks on a row are not filled

:- fill(X, Y, C), sc(X, B, S1), lc(X, B, L), sc(X, B+1, S2), S1+L<Y, Y<=S2, cell(X, Y), startc(S1;S2). % Cells between blocks on a column are not filled
```

```
fill(X, Y, C) :- sr(Y, B, S), lr(Y, B, L), S<X, X<=S+L, cell(X, Y),
startr(S), cr(Y, B, C). % Cells covered by blocks on a row are filled with
the same color

fill(X, Y, C) :- sc(X, B, S), lc(X, B, L), S<Y, Y<=S+L, cell(X, Y),
startc(S), cc(X, B, C). % Cells covered by blocks on a column are filled with
the same color

:- fill(X, Y, C), br(Y, 0), cell(X, Y). % Cells in a row with no clues are
are not filled

:- fill(X, Y, C), bc(X, 0), cell(X, Y). % Cells in a column with no clues are
are not filled
```

For the complete piece of ASP code that I developed, please see the Appendix. After the solver was completed, I realized that due to the complexity of input into the solver that testing it on larger puzzles would not be easy, as the input to the program is not elegant. I decided I needed to create a GUI that enables easier input of puzzles into the solver, so I wrote one in Visual Studio .NET using C#. Some screenshots of the final GUI are shown in the Appendix. The GUI I created allows the user to choose the size of the grid and the colors with which to "paint" the grid. The user can then simply left click on a cell to toggle the color to paint, then hover the cursor over the desired cells, and then right click to stop painting. When finished, clicking a button causes the program to generate the instance of the puzzle and send it on to the ASP solver. To program takes the output of the solver and parses through it and displays the results in a more user-friendly manner. If there is only the one solution, the program outputs a message notifying the user of this. If there is more than one solution, it displays a message notifying the user of how many there are, then allows the user to see all of them in a window vary similar to the one on which they "painted" the original puzzle. I was able to use this GUI to test my ASP program in a much less error prone and more time efficient manner. I found that there were few puzzles from webpbn that I input to my program that took longer than a couple seconds to determine uniqueness.

Future Work

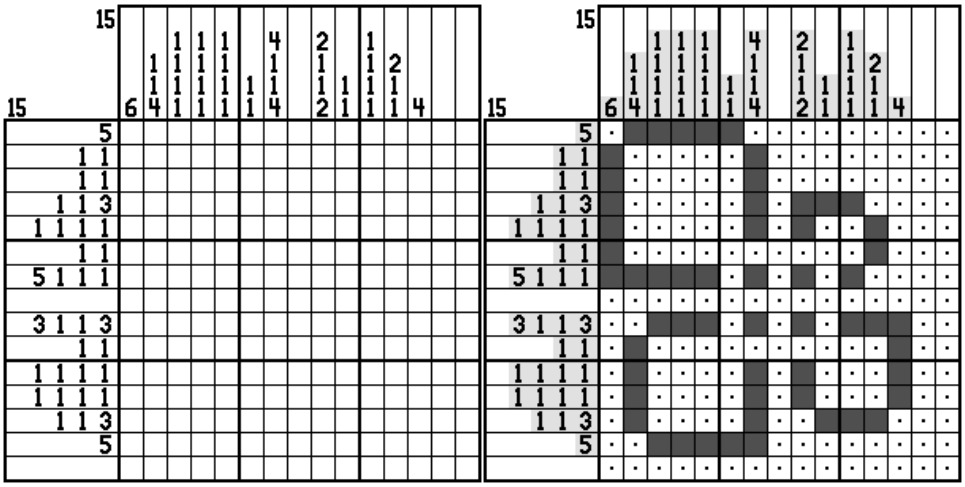
As I have already stated, my program determines uniqueness of most black and white as well as multicolored puzzles. Here is what I would still like to expand on my program to do:

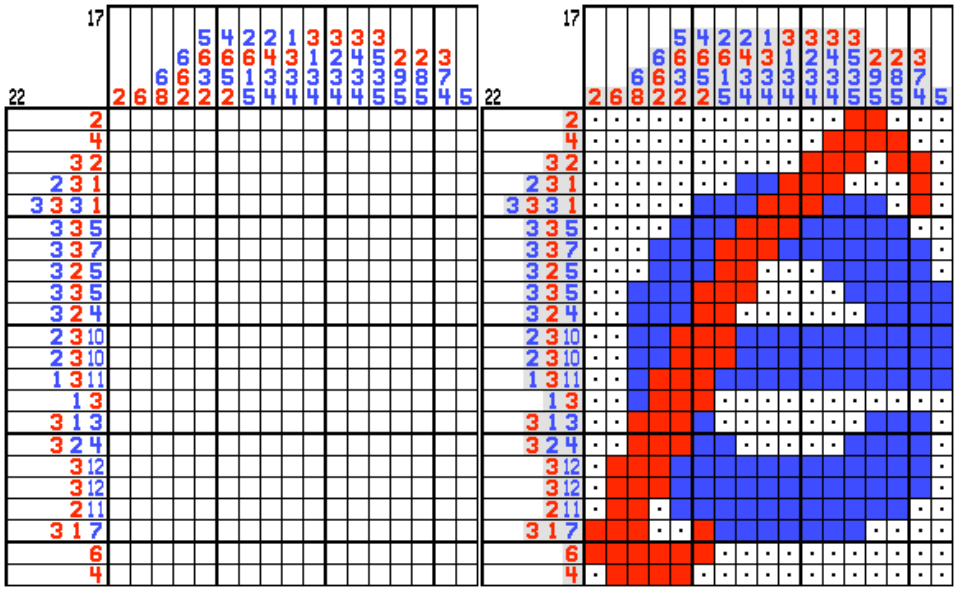
1. Webpbn has an "export" feature that outputs a puzzle in XML format. I would like to enable my program to take this output and automatically "paint" the initial grid to further speed up testing of webpbn puzzles.
2. Make the GUI a bit more user-friendly, with a palette to choose colors to paint rather than clicking on a cell to toggle colors. The current method that I use is not as intuitive as I would like.
3. Some puzzles still take very long for my ASP program to solve puzzles that I know have only one solution. I would like to come up with additional constraints that would assist the program in solving these puzzles.

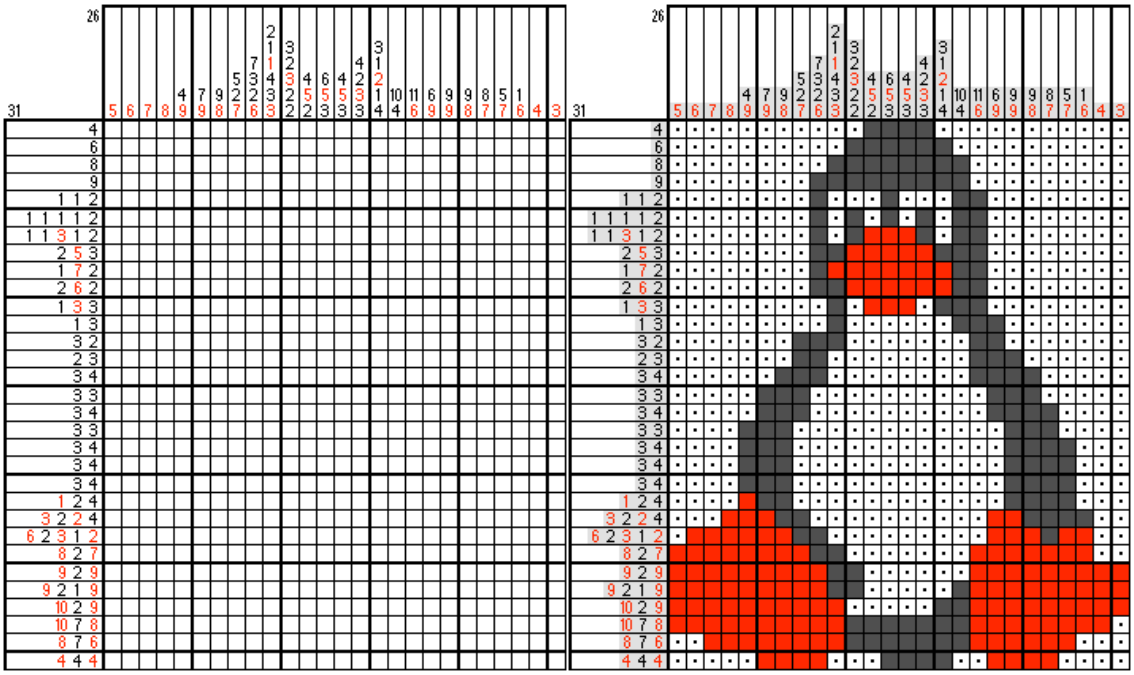
Appendix

Examples:

Here are a few more examples of unsolved puzzles and the corresponding solved puzzles. They may be used to aid in understanding the rules of the Paint by Number puzzle.





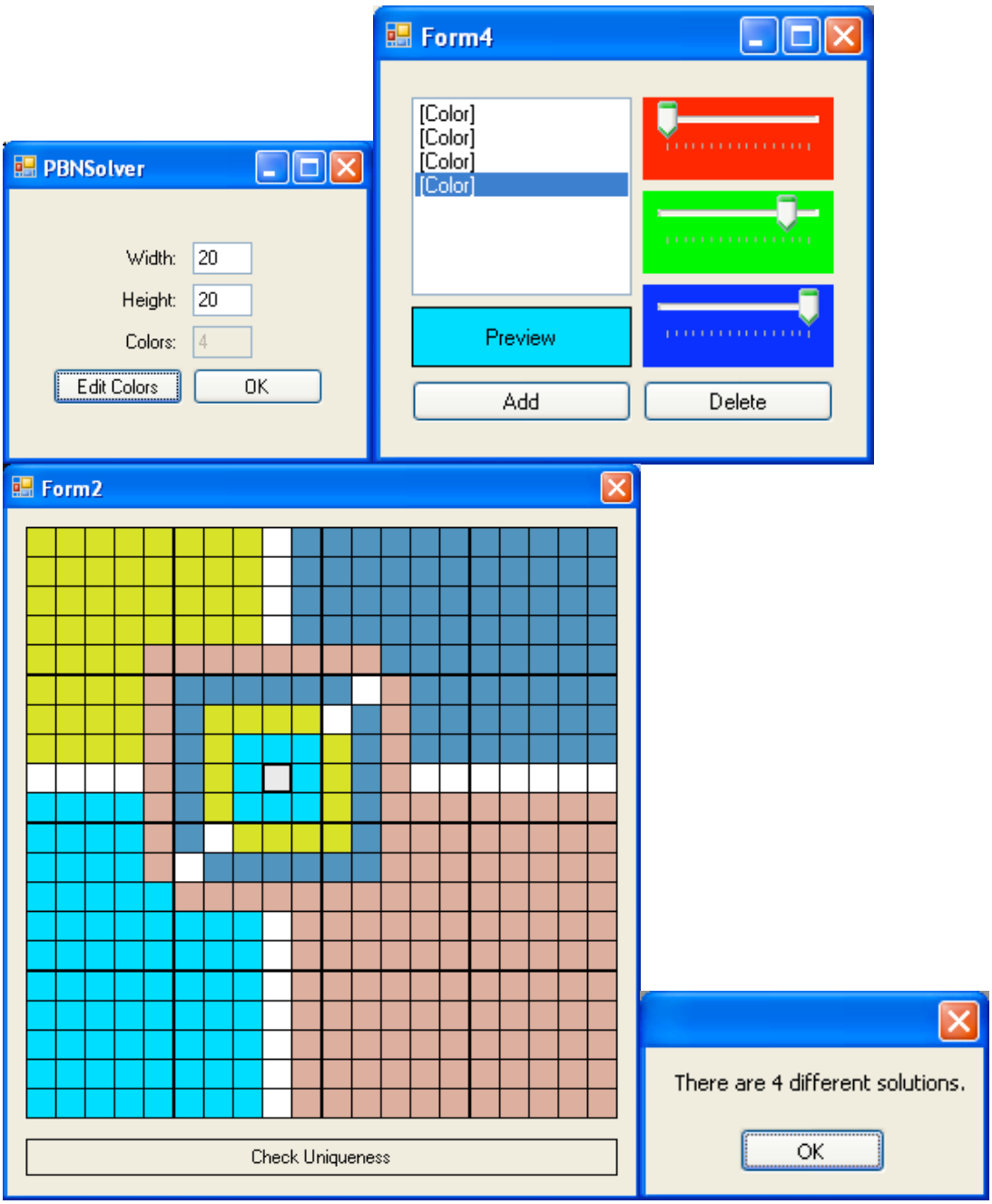


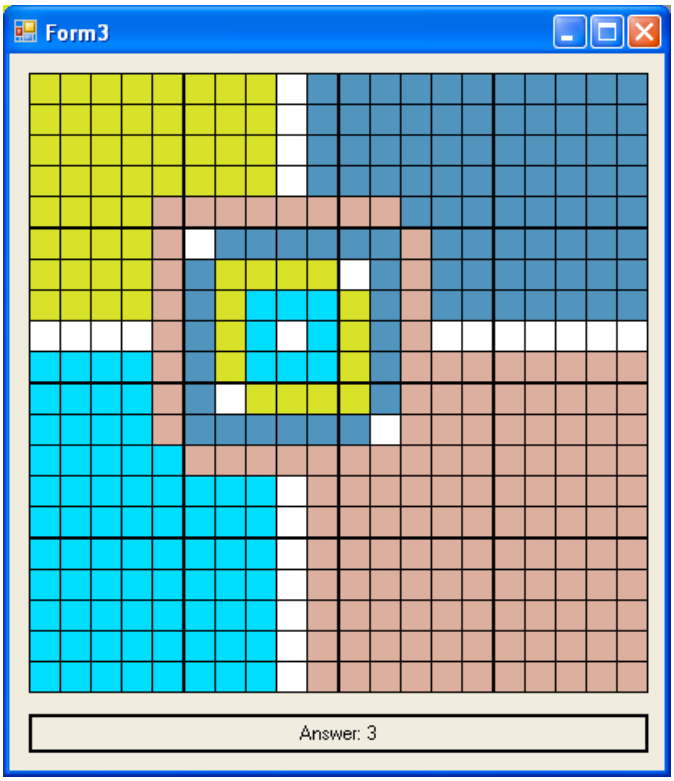
Complete code listing:

```
startr(0..columns-1).
startc(0..rows-1).
row(1..rows).
column(1..columns).
color(1..colors).

#domain row(Y).
#domain column(X).
#domain color(C).

%%% OVERALL RULES %%%
% There is a grid of cells
cell(X, Y).

% Every cell is filled with a color or dotted
1 { dot(X, Y), fill(X, Y, N) : color(N) } 1.

%%% ROW RULES %%%
% Every given block in a row has some start cell
1 { sr(Y, B, S) : startr(S) } 1 :- lr(Y, B, L).

% The end of a block in a row cannot hang off the end of the row
:- sr(Y, B, S), lr(Y, B, L), S+L>columns, startr(S).

% Each block in a row is separated by at least one space if they are the same
color
:- sr(Y, B, S1), lr(Y, B, L), sr(Y, B+1, S2), S1+L>=S2, startr(S1;S2), cr(Y,
B, C), cr(Y, B+1, C).
:- sr(Y, B, S1), lr(Y, B, L), sr(Y, B+1, S2), S1+L>S2, startr(S1;S2).

% Cells before the first block in a row are not filled
:- fill(X, Y, C), sr(Y, 1, S), S>=X, cell(X, Y), startr(S).
```

```
% Cells after the last block in a row are not filled
:- fill(X, Y, C), sr(Y, B, S), br(Y, B), lr(Y, B, L), S+L<X, cell(X, Y),
startr(S).

% Cells between blocks on a row are not filled
:- fill(X, Y, C), sr(Y, B, S1), lr(Y, B, L), sr(Y, B+1, S2), S1+L<X, X<=S2,
cell(X, Y), startr(S1;S2).

% Cells covered by blocks on a row are filled with the same color
fill(X, Y, C) :- sr(Y, B, S), lr(Y, B, L), S<X, X<=S+L, cell(X, Y),
startr(S), cr(Y, B, C).

% Cells in a row with no clues are are not filled
:- fill(X, Y, C), br(Y, 0), cell(X, Y).

%%% COLUMN RULES %%%
% Every given block in a column has some start cell
1 { sc(X, B, S) : startc(S) } 1 :- lc(X, B, L).

% The end of a block in a column cannot hang off the end of the column
:- sc(X, B, S), lc(X, B, L), S+L>rows, startc(S).

% Each block in a column is separated by at least one space
:- sc(X, B, S1), lc(X, B, L), sc(X, B+1, S2), S1+L>=S2, startc(S1;S2), cc(X,
B, C), cc(X, B+1, C).
:- sc(X, B, S1), lc(X, B, L), sc(X, B+1, S2), S1+L>S2, startc(S1;S2).

% Cells before the first block in a column are not filled
:- fill(X, Y, C), sc(X, 1, S), S>=Y, cell(X, Y), startc(S).

% Cells after the last block in a column are not filled
:- fill(X, Y, C), sc(X, B, S), bc(X, B), lc(X, B, L), S+L<Y, cell(X, Y),
startc(S).

% Cells between blocks on a column are not filled
:- fill(X, Y, C), sc(X, B, S1), lc(X, B, L), sc(X, B+1, S2), S1+L<Y, Y<=S2,
cell(X, Y), startc(S1;S2).

% Cells covered by blocks on a column are filled with the same color
fill(X, Y, C) :- sc(X, B, S), lc(X, B, L), S<Y, Y<=S+L, cell(X, Y),
startc(S), cc(X, B, C).

% Cells in a column with no clues are not filled
:- fill(X, Y, C), bc(X, 0), cell(X, Y).

hide.
show fill(_,_,_).
show dot(_,_).
```

Screenshots of GUI:





This is one of the four solutions to the entered puzzle. Note that it is NOT the same as the one entered, although it is very close. This is one puzzle that would nearly be solved by the time you realize there are multiple solutions.