Programming by Robert Massaioli

Learning through programming adventures.

GAMEDEV, **INTERESTING**

Solving Minesweeper with Matrices

Posted on January 12, 2013February 8, 2016 by Robert Massaioli in Gamedev, Interesting

What is your motivation for writing this?

Note: skip to the next section if you don't care about the back-story and want to get straight to the actual algorithm.

Back in 2008 I was starting Computer Science at UNSW. I was actually enrolled in the course that became those famous YouTube video lectures on Computer Science by Richard Buckland. I was also enrolled in your standard first year Maths course at the time and we were just learning Matrix mathematics. While in the computing lectures and in the grounds and basements around campus I had a friend that loved to play Minesweeper and boy were they fast. But as I watched them play I came to realise that it really was a simple game and probably something that would be better suited to a computer solving. And then, as I wrote out a simple game of minesweeper it hit me, you could solve Minesweeper with matrices I then proceeded to write program that did exactly that, it solved minesweeper as best as it could without probabilities.

Fast forward to just last month, just before Christmas, when I checked Reddit and saw the following blog post get released: http://luckytoilet.wordpress.com/2012/12/23/2125/

It was a pretty cool blog post and it explained a method of solving minesweeper and how you would go about doing that. I commend the author on writing it. However, one thing bugged me, nobody seemed to realise that you can actually solve Minesweeper by using Matrices (and one special lemma specific to minesweeper). So I made a comment to that affect on Reddit and I gained some interest from people that wanted to know how to do that and how it was possible. So I have decided to explain this method fully and provide a working implementation. It was a fair bit of work but I hope that you enjoy the end results.

Just as a side note: I want you to know that I am not unique in finding this method of solving Minesweeper. Here is a website of somebody that discovered it two years after I did. And I am sure that there are people that worked that out before we did again. I believe that Matrices are just the natural way to solve this kind of problem.

Quick Overview

This blog post is going to cover:

- A simple example of how this method works and can be used to find solutions to Minesweeper configurations.
- A robust and reasonably efficient general algorithm that explains how to apply this in the real world.
- A brief description of my implementation of this method which is available on BitBucket: http://bitbucket.org/robertmassaioli/minesweeper-and-matricies/overview
- Please note that I will try and provide code links, where possible, so that you can follow along in the code. If you are like me then you enjoy reading code more because it is more precise.

Prerequisites

You will need to have the following skills to read this blog post:

- Linear Algebra Knowledge that includes Matrices. If you don't know what matrices are then go lean about them, they are very useful tools in a programmers toolkit and you certainly need them for Video Game Development. Really go learn about it; it will take time but it is worth it.
- How to play Minesweeper. I don't explain the general rules of minesweeper, if you want to know how to play then go read the rules or, better yet, go play a game before reading this post. You can play Minesweeper on Windows, Linux and OSX; there are ports for every OS.

To read the code you will also need to understand C++; the coding could have been better, sorry. On the plus side, your C++ reading comprehension will improve.

The General Idea (aka How it works)

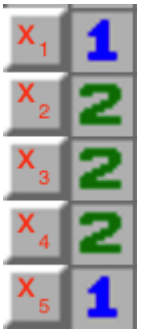
When dealing with a new problem it helps to first start with a simple example. You use a simple example because it is easier to conceptualise. Using the simple example you then develop a rigorous model for solving the problem in general. Once you have that model you then apply it to more complicated scenarios and problems and you discover, to your pleasure, that you did it. That is exactly the process that we are going to go through here.

A Simple Example

Here is a very small minesweeper configuration and we will be using this as our simple example:

This is a simple example of Minesweeper taken from http://minesweeperonline.com/

Those of you that have played minesweeper before should be able to solve this configuration as best as you can using the intuition that you have learned from playing many games. That is good, but I want a robust math-based solution to this problem. So lets look at what this Minesweeper configuration tells us. The first thing that I note here is that we have five squares that have not been clicked yet. They are our 'unknowns'; these squares either contain mines or they do not, there is no other alternative. So since they are our unknowns then lets label them and make them our variables. I have done that in the following image:



All of the unknown squares labeled from x_1 to x_5. Each of them either contains a mine or it does not.

Now for any unclicked square x_i look at the square x_1, lets say that if it is a mine then it has a value of 1 and if it is not then it has a value of 0. Therefore mines are ones and non-mines are zeros; simple.

Now take a look at the top right hand corner square of the previous image; it contains a 1 (from here on in we will call clicked squares that contain numbers 'numbered squares'). That means that it is adjacent to one, and only one, mine. So first we look to see which non-clicked squares are adjacent to it and we discover that x_1 and x_2 are the only squares adjacent to it. Therefore we know that the number of mines in both x_1 and x_2 must add up to equal 1. Another way of writing that is the following:

$$x_1 + x_2 = 1$$

Now we can come up with similar equations for the other numbered squares on the right hand side of the simple Minesweeper example. If we do that we get the following equations:

$$\begin{aligned} x_1 + x_2 &= 1 \\ x_1 + x_2 + x_3 &= 2 \\ x_2 + x_3 + x_4 &= 2 \\ x_3 + x_4 + x_5 &= 2 \\ x_4 + x_5 &= 1 \end{aligned}$$

You may not be able to see (just yet) why the previous set of equations is incredibly useful, but the key insight here is to realise that you now have a set of five linear equations with five variables. It will be even clearer if you let me add in the co-efficients and the non-clicked squares that had co-efficients of zero:

$$\begin{aligned} 1x_1 + 1x_2 + 0x_3 + 0x_4 + 0x_5 &= 1 \\ 1x_1 + 1x_2 + 1x_3 + 0x_4 + 0x_5 &= 2 \\ 0x_1 + 1x_2 + 1x_3 + 1x_4 + 0x_5 &= 2 \\ 0x_1 + 0x_2 + 1x_3 + 1x_4 + 1x_5 &= 2 \\ 0x_1 + 0x_2 + 0x_3 + 1x_4 + 1x_5 &= 1 \end{aligned}$$

The same equations as before, just written a little more clearly.

As you can see this looks exactly like it should be solved using a Matrix that is exactly what we are going to do. Here is the previous results in an augmented matrix:

$$\left(\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 2 \\ 0 & 1 & 1 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right)$$

At this point in time we want to get a solution to this matrix so, as usual, we Gaussian Eliminate to find a solution. The solution is on the next line but I recommend that you solve this yourself on a pen and paper if you have one handy. If you don't then you can take my word for it and just move to the next line:

$$\left(\begin{array}{ccccc|c} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

On first glance at this eliminated matrix you can immediately tell that there is no unique solution to the vector x (this is where I am relying upon your prerequisite knowledge). This may mislead you into thinking that the Gaussian Elimination failed but that would be incorrect; it worked perfectly and it has given us a partial solution to the vector x. To see why you need to remember that each non-clicked square in the minesweeper grid is either a mine underneath or it is not a mine (1 or 0). Therefore each value in the vector x has the following property:

$$x \in [1, 0]$$

Every value x is either a 1
or a zero.

This means that the matrix above has an extra property that we do not get when the expected values of the vector x could be anything in a set of infinite numbers, like the set of integers or reals. Remember that we are in the boolean set and this will all make sense.

To understand this property lets take a look at the third row of the eliminated matrix. As you can see x_3 is the only column of the matrix with a non-zero co-efficient and the row adds to give 1. Setting x_3 to be 1 is the only value that makes the row work (conversely, if it last value in the row was 0 then x_3 would have to be zero). This means that we can tell that x_3 is a mine even though we do not know what the other squares are. It is interesting to note that we can only tell that from the Gaussian eliminated matrix; not the original matrix. So even though the elimination does not find a complete solution it still simplifies the matrix and allows us to get partial solutions. But what is the general rule to get partial solutions from eliminated matricies?

A Special Rule

Lets see a few more example rows that can help us to intuitively derive that rule:

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 1 & 2 \end{array}\right)$$
$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 1 & 0 \end{array}\right)$$
$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & -1 & 1 \end{array}\right)$$
$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & -1 & -1 \end{array}\right)$$

These are some examples of some extra rows that may be possible end
results after gaussian elimination.

Pretend each of the rows in the above image are unique rows taken from unique matrices (what I am trying to say is that each row above is not correlated, they are all unique). Let me deal with each row in a dot point:

1. If we take a look at the first row you should be able to tell that x_1 and x_4 are both mines because that is the only way that they will equal 2.

2. If you look at the second row you can see that both x_1 and x_4 must not be mines because that is the only possible solution for $x_1 + x_4 = 0$ when the only potential values for any x are ones and zeroes.
3. The third row is interesting because it has a negative number, that means that the equation is $x_1 - x_4 = 1$. This can only be true if x_1 is a mine and x_4 is not. Now things start to get interesting, clearly we need some concept of a minimum and maximum bound for each equation. In this example the maximum value the equation could take is 1 and the minimum value that it could take is minus one. Since this row meets that upper bound we can solve for it.
4. This is the same as the previous example except that it meets the lower bound. This also means that we can solve for it.

As we can see the general solution to getting more information from each row is to to work out the lower and upper bounds and see if the value on the other side of the equality is the same as one of the bounds. If it is then you know that there is only one possible configuration of mines that will allow that to occur and you can quickly rattle that off. Because of that uniqueness property you can only apply this rule to a row if it is equal to an upper or lower bound; if it does not then multiple solutions are possible and you have strayed into the area of probabilistic analysis that this blog post will not attempt to cover. This grid shows what logic you use, on a per-square basis, to partially solve the matrix:

	Co-Efficient is Positive	Co-Efficient is Negative
Row meets lower bound	Not Mine	Mine
Row meets upper bound	Mine	Not Mine
Row meets neither bounds	Unsure	Unsure

You can use this rule on any Gaussian Eliminated Minesweeper matrix to get partial solutions from rows. Just so that you can really see how that works here is a rough algorithm (and here is a link to the actual C++ code):

```
Set the maximum bound and minimum bound to zero
For each column in the row (not including the augmented column of course) if the
If the augmented column value is equal to the minimum bound then
   All of the negative numbers in that row are mines and all of the positive va
else if the augmented column value is equal to the maximum bound then
   All of the negative numbers in that row are not mines and all of the positive
```

Finishing the Simple Example

So now lets wind back to the Gaussian Eliminated matrix. As we can see the only row that we can apply this rule to is row 3 which tells us that x_3 is a mine. Therefore we can flag that square:

The final result of our simple minesweeper configuration.

And that is it for our simple example, we have worked out as much as we possibly can without more information. The game is still in progress but if we want to move forward we would have to make a guess or some probability based decision that could fail. This method of solving Minesweeper only works for grids that are completely solvable without guesswork and it is my future plan to expand this method to include probabilistic analysis as well.

The Robust Algorithm

Taking what we have learned from the simple example we can create an algorithm that is a fair bit more robust:

1. Get a list of the squares that contain numbers AND are adjacent to at-least one square that has not been clicked or flagged. (code link)
2. For every numbered square in the list assign a unique matrix column number to that square. This is so that we can map our Matrix columns to Minesweeper squares. (code link)
3. For every numbered square in the list create a matrix row that represents the adjacent non-clicked squares and the number they touch. Don't forget to put zeroes in all of the matrix columns that are not adjacent. (code link)
4. Gaussian Eliminate the Matrix. (code link)
5. Attempt to use standard matrix reduction, and the special rule that we developed, to get a partial (or even full) solution to the the current Minesweeper configuration. Remember to tackle the matrix from the bottom row up so that you can make use of partial solutions as you go. (code link)
6. Use the (possibly partial) solution you worked out to generate the list of clicks that should be made: flagging known mines and clicking known empty squares. Leave everything else alone and wait for more information. (code link)
7. Keep running all of the previous steps in a loop until you either cannot make any moves (meaning that you cannot get further without guessing) or until the game is finished and won. (code link)

And that is all that there is to it. Writing those steps can get a little complicated at times but totally manageable with a decent working knowledge of Matrix mathematics. I have not explained those steps in really great detail because if you want that information then you have now got to the point where you should really check out my code and have a run and read.

The Implementation

All of this talk would mean nothing if I was not able to implement it and show you some working code. Therefore, I have implemented this algorithm from scratch and have provided the source code to everybody under the MIT license. If you use this code anywhere or use the idea, then I would really appreciate it if you mentioned my name or gave me attribution somehow; really it would make my day.

Go get the code from BitBucket: http://bitbucket.org/robertmassaioli/minesweeper-and-matricies/overview (By the way, while we are here, did I mention that I love pull requests)

How do I compile your code? Read the README.markdown file that is in the root directory of the project. It will always have up to date details.

What design choices did you make?

Haha, design choices, that's a good one. This code is not the most beautiful code that I have ever written. I wrote it all myself to avoid spurious dependencies in the hope of easy cross platform compilation. I have not even used RAII principles in this code and frankly that makes the delete's sprinkled all over the code quite ugly. If I was writing this without a care in the world for dependencies then I would have used Boost and gained a large amount of nice looking code for free. Also the solver looks like a bit of a monster method at the moment, sorry, it should be re-factored.

How fast is it?

Short answer: very fast and much faster than it needs to be to solve a single game of minesweeper. The longer answer is that his code was written in C++ and it is lightning fast even though it only uses a single core to do the processing and thus does no parallelism whatsoever. It is fast because working with Matrices is quick. I gain speed just by the fact that I used efficient mathematical constructs. To give you an example, it takes less than a minute and thirty seconds to play one hundred thousand minesweeper games on a single core on my computer (I have a machine that contains an "Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz"). I would expect you to see similar speed results on your own machine.

Results of playing many Games

My minesweeper implementation plays a great many games but here are the results of it attempting to play 100000 games of Beginner, Intermediate and Expert minesweeper. Please keep in mind that there are three states that the game can end up in:

- The Win state: we were able to completely solve the grid without guessing.
- The Progress state: we got to a point in the game where the only move we could make would have to be a guess. As a result we stopped making moves and left the game partially completed and still 'in progress'.
- The Lost state: this happens when you click on a mine. That should not be possible using our method and, if you see that you can consider it a bug and please report it to me.

Beginner

A 'Beginner' grid is 8×8 or 9×9 with only 10 mines. have chosen the easier of the two and gone for the 9×9 grid:

```
WINs: 74090 (74.09%)
PROGRESSes 25910 (25.91%)
ERRORS (losses) 0
./localbuild/src/mnm 8.68s user 0.00s system 99% cpu 8.697 total
```

As you can see a beginner grid is pretty easy, it only took ~9 seconds for my computer to do play 100000 games and it won ~74% of the time.

Intermediate

An 'Intermediate' grid is 16×16 squares with 40 mines and this is how my run performed:

```
WINs: 43432 (43.432%)
PROGRESSes 56568 (56.568%)
ERRORS (losses) 0
./localbuild/src/mnm 43.56s user 0.01s system 99% cpu 43.627 total
```

In an intermediate game there is less chance that you can win without guessing. I have run this a number of times and you have about a 45% chance that you will be given an intermediate grid that you can win without guessing. That makes the intermediate games a fair bit harder.

Expert

An 'Expert' grid is 30×16 squares with 99 mines and this is how this current run performed:

```
WINs: 1707 (1.707%)
PROGRESSes 98293 (98.293%)
ERRORS (losses) 0
./localbuild/src/mnm 83.87s user 0.00s system 99% cpu 1:23.98 total
```

As you can see from these combined results the solver is very fast and the difficulty levels of Microsoft's minesweeper are appropriately chosen; you have a very small chance of getting a grid in Expert mode that lets you win without making at-least one guess.

Next Steps

There are a few extra things that I would like to do to the codebase if I had some time:

- I would make a probabilistic solver to attempt to solve the majority of the games instead of leaving them in the 'Progress' state.
- I would make the code multi-threaded where I could. Specifically it would be good to run multiple test games in parallel because they are a great example of a 'painfully parallel' problem.
- The code should have better test cases. Currently only the matrix code is tested reasonably well. The game and the solver should have more test cases too.

So just to wrap it up quickly, this is how you solve Minesweeper with Matrices. Please feel free to ask any questions you like or make suggestions below.

MATHEMATICS **MATRICIES** **MINESWEEPER**

20

20 thoughts on "Solving Minesweeper with Matrices"

1. **Florian Kirmaier**
says:
January 20, 2013 at 6:42 pm
I don't think that your algorithm finds als solutions.
lets assume the following gausian eliminated Matrix:
e1: 1 1 1 | 1
e2: 0 1 1 | 1
e3: 0 0 0 | 0
you won't find the solution with your algorithm.
But it is solveable without guessing.
when you compute e1 – e2 you will get x1 = 0. So x1 cant be a mine.

   please correct me, if i am wrong.

   I dont think solving minesweeper is so easy. Minesweeper is still NP.

My Solver, which i wrote about 3 years ago, solves about every 10th game on hard (starting with 3×3 empty fields). It's using a similar technique, but without matricies. It combines equations with the same variables. It's using some heuristics to prefer the correct equations. I use it to messure the difficulty of minesweeperboards. So i can breed some super-hard games with an evolutionary algorithm. Sometimes they become realy hard.

At the moment im working on a web-version of these hard minesweeper problems.
Heres an example of a realy difficult problem with a solution: http://i.imgur.com/t9lT0tu.png?1

PS. im not a native speaker.

Reply

1. **Robert Massaioli**
says:
January 20, 2013 at 10:26 pm
That is a well thought out counter-example, I am honestly really happy that somebody has seen fit to respond to my post with such a good comment. However I am going to need to sit on that for a bit, because the counter example that you have given me looks something like this:

However maybe reduced-row-echelon form will prevent these types of problems…hmmm. I think what I really need to do now is grab all of the complicated minesweeper configurations and make test cases out of them. That way I know if my Minesweeper solver is upto scratch.

Wow; honestly, kudos to you and very well done.

Reply

1. **Florian Kirmaier**
says:
January 21, 2013 at 5:53 am
Yes, reduced-row-echelon form would solve this Problem. It will solve more Problems than the previous Algorithm, because the equations have got less variables. But i think, it solves only a subset of all minesweeper problems.

I've got a example of a reduced-row-echelon form where you wont find the solution without additional computations.

e1: 1 0 0 0 | 3
e2: 0 1 0 1 | 1
e3: 0 0 1 -1 | 0

e1 + e2 + e3 => x1 + x2 + x3 = 3 => all three are mines.
But i dont know how to prove, that this or a similar matrix represents a minesweeper problem.

2. **Florian Kirmaier**
says:
January 21, 2013 at 10:41 am
I have to correct myself.
my last example doesn't work. e1: x1 = 3 can not be a minesweeper problem, because this that there are 3 mines at x1.

2. **matt**

says:
May 11, 2013 at 11:42 am
Minesweeper has been proven np-complete so you're going to have to settle for a heuristic that won't find all solutions.

http://web.mat.bham.ac.uk/R.W.Kaye/minesw/ordmsw.htm

Reply

3. **Emilian**

says:
June 13, 2014 at 11:27 pm
I have a web implementation of a minesweeper solver which can handle any solvable problem, or so I think.

https://www.logigames.com/minesweepersolver

Reply

1. **Robert Massaioli**

says:
June 14, 2014 at 7:11 pm
Are you sure that this is correct in the event of symmetry? For example this is what I see:

It seems to me that you cannot say for certain that the red flags are mines. They are just potentially mines.

Reply

1. **Emilian**

says:
June 15, 2014 at 12:55 am
Maybe it's not obvious and I have to mention that the solver:
– doesn't pre-validate the input configuration, therefore for invalid configurations the results are unpredictable
– takes into account that the 9*9 board has a total of 10 mines, 16*16 has 40 mines and 16*30 has 99.

The configuration you show here is invalid because there is no solution for it.
? ? ? ? ?
? 1 2 1 ?
? 2 0 2 ?
? 1 2 1 ?
? ? ? ? ?

2. **Robert Massaioli**

says:
June 15, 2014 at 6:02 pm
That makes sense. Though if you are letting people input boards then it would be cool to let them know if there was an invalid board.

4. **vdpittman**

says:
September 30, 2015 at 6:27 am

Great piece! Inspired by your article, I'm trying a similar approach in C#, but with a significant difference: I consider each numbered square that abuts one or more unknown (unchecked) squares, with coefficients of 1 for each of the adjacent unknowns and the RHS being the number from the square less the count of adjacent marked squares, and if that equation is not already in the matrix, I add it to the matrix. However sometimes I wind up with more rows than columns, which my Gaussian Elimination routine doesn't know how to handle. Any suggestions for handling the case of having more equations than unknowns? Should I just limit the number of equations to be no more than the number of unknowns for each iteration?

Reply

1. **Robert Massaioli**
says:
September 30, 2015 at 9:17 am
How is that different from the way that I am doing it in my example? Could you show a full worked example of what you are trying to do and where the problem is?

Reply

5. **vdpittman**
says:
September 30, 2015 at 10:57 am
The way I read the "Robust Algorithm" section (perhaps wrongly), you add a column (Item 2) and a row (Item 3) for each numbered square that abuts an unknown square. I only add one column per unknown square instead of for every numbered square, and one for the equation for each numbered square, if a like-valued row is not already in the matrix (as determined by comparing the row elements one-by-one). Not a huge difference but it cuts down the matrix size.

I also added an aggressive "zero chaser".

The "problem" I run into, sometimes, is having more rows than non-RHS columns going into the Gaussian elimination. Right now I stop adding rows when the number of rows equals the number of non-RHS columns, but am concerned I might be leaving something important out.

Here's an example… Say the grid is as follows, where letters indicate an unknown square (there are no marked squares):

0 1 A C
0 1 B 1
0 1 1 1
D 0 E 0

My algorithm generates the following matrix for Gaussian elimination (after weeding out duplicated rows):

A B C D E | rhs
1 1 0 0 0 | 1
1 1 1 0 0 | 1
0 0 0 1 0 | 0
0 1 0 1 1 | 1
0 1 0 0 1 | 1
0 0 0 1 1 | 0
0 0 0 0 1 | 0

So that's 5 unknowns and 7 unique equations.

As I said, right now I clip the rows after the fifth one since there are 5 unknowns. But I realize now after typing this all in is that what I need to do is check each candidate row with the upper/lower bounds algorithm and cover or flag squares right then if possible instead of adding it to the matrix for Gaussian elimination. That'll make the overall run to completion all the faster.

Thanks for your attention, for replying, and for the great article! It's been great fun working on it.

Reply

6. **vdpittman**
says:
September 30, 2015 at 10:59 am
Make that "I only add one column per unknown square instead of for every numbered square, and one *row* for the equation for each numbered square"… (Arggh)

Reply

7. **LiquidM**
says:
December 13, 2015 at 7:56 pm
Just a thought for all you smart people:

Has anyone considered 3 dimensional mine sweeper? What about n-dimensional?

Reply

1. **Robert Massaioli** AUTHOR
says:
December 13, 2015 at 7:59 pm
What would a 3-d minesweeper provide that a 2-d minesweeper does not? I believe that the logic is the same.

But, at any rate, you can go and play 3D minesweeper if you like:
http://egraether.com/mine3d/ (People have thought of this before 🙂)

Reply

8. **Repcak**
says:
December 23, 2015 at 1:37 am
Is there any chance of an upgrade of the images in the article? Because alot of them dont display. I just started getting into the problem of solving minesweeper and want to write a program to it and you article is helping alot. 🙂 Thanks in advance.

Reply

1. **Robert Massaioli** AUTHOR
says:
February 10, 2016 at 3:08 am
The article has now been fixed. Sorry for the delay.

Reply

9. **David McDermott**
says:
February 10, 2016 at 2:50 am

Can you give an example of how your algorithm works on island tiles. such as the configuration below

[E][E][E][E][E][E][E][E][E]
[E][E][E][E][E][E][E][E][E]
[E][E][E][E][E][E][E][E][E]
[E][E][E][3][2][E][E][E][E]
[E][E][E][4][2][E][E][E][E]
[E][E][E][E][E][E][E][E][E]
[E][E][E][E][E][E][E][E][E]
[E][E][E][E][E][E][E][E][E]

3 2
4 2

with E being blank tiles.
This is the matrix I get with index left to right, top to bottom

3: 1 1 1 0 1 0 1 0 0 0 0 0
top 2: 0 1 1 1 0 1 0 1 0 0 0 0
bot 2: 0 0 0 0 0 1 0 1 0 1 0 1
4: 0 0 0 0 1 0 1 0 1 1 1 0

Reply

1. **Robert Massaioli** **AUTHOR**
says:
February 10, 2016 at 3:08 am
Quick answer: That is not a valid minesweeper configuration. However, if I spend some time soon then I might be able to generate that grid in the source code and just tell you what it generates. You can do the same if you have more time spare than I do.

Reply

10. **David McDermott**
says:
February 10, 2016 at 2:55 pm
What do you mean by invalid?

Solutions are:

. B
.B 3 2
.B 4 2 B
.B B

. B
.B 3 2 B
.B 4 2
.B B

.B B B
. 3 2
.B 4 2
.B B B

. B B
.B 3 2
.B 4 2
. B B

. B B
.B 3 2
.B 4 2
.B B B

. B B
.B 3 2
.B 4 2
. B B

. B B
.B 3 2
.B 4 2
.B B B

I believe that is all of them so following the same number convention.
0 or 1
0 or 1
0 or 1
0 or 1
0 or 1
#3
#2
0
1
#4
#2
0 or 1
0 or 1
1
0 or 1
0 or 1

Let me know if my calculations are incorrect.

Reply