Building Minesweeper in Minesweeper

Kirby703

2025-03-28ish

1 Introduction

You've heard of building things in Minecraft. Someone's probably already built Minesweeper in Minecraft![1] But, surely, you're not ready for building things in Minesweeper, a game where you start with an unrevealed board and gradually reveal cells that tell you how many of their 8 neighbors are hidden mines. Also, while I've got you here, I should note that you can actually play the final product of this paper in your browser.[2] Check it out!

2 Prior Work

Surprisingly, there is a little bit of prior work for this. Richard Kaye[3] made a few logic gates and showed that one could construct an arbitrarily large boolean circuit on the board in polynomial space. All this to prove that it's NP-hard to verify if a supposed screenshot of Minesweeper actually corresponds to a possible game state. Though, his gates lack a few properties that I want, so I didn't end up using any of them.

3 Components

Don't worry about dissecting these; they won't be on the exam.

3.1 Wires, splitters, and inverters

Any sufficiently avid Minesweeper player has probably run into a few of these. They can be in one of two states, and arbitrarily long. You can even wrangle them in all sorts of ways:

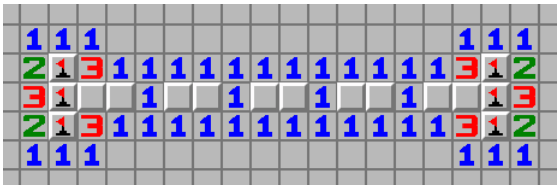

Figure 1: A wire, terminated at both ends.

[1] mattbatwings, apparently

[2] https://github.com/Kirby703/minesweeper-in-minesweeper

[3] https://web.archive.org/web/20231102222553/https://web.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.pdf

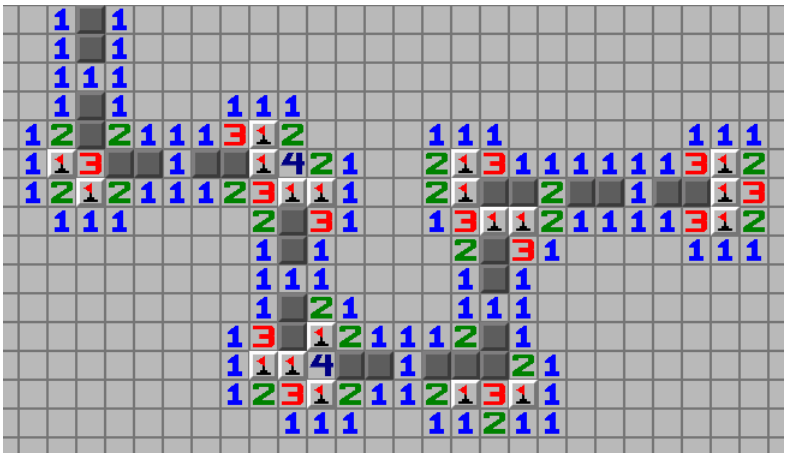

Figure 2: A wire (gray) taking all sorts of turns and snaking around the board.

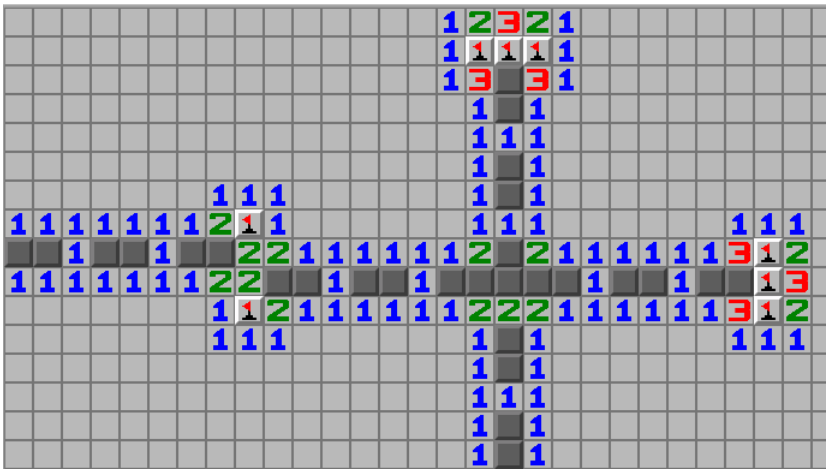

Figure 3: A wire being shifted and crossing over another wire without interfering.

Figure 4: Getting two wires side-by-side.

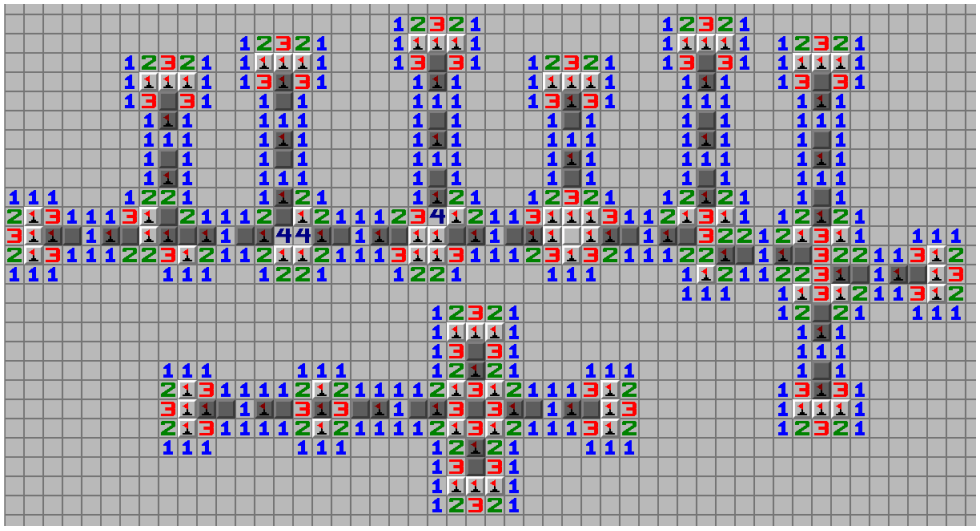

Figure 5: Various splitters and inverters.

3.2 Logic gate(s)

Minesweeper is a game of solving a board full of constraints (the visible numbers). When you're making a board, you get to choose what these constraints are. So, making a logic gate isn't really a matter of going through some process to generate the output you want. It's really about creating a design, and then eliminating all possible universes where it fails. Initially, I had wanted to create an AND, OR, and perhaps an XOR gate. As I assembled one, it dawned on me that I'd made all three:

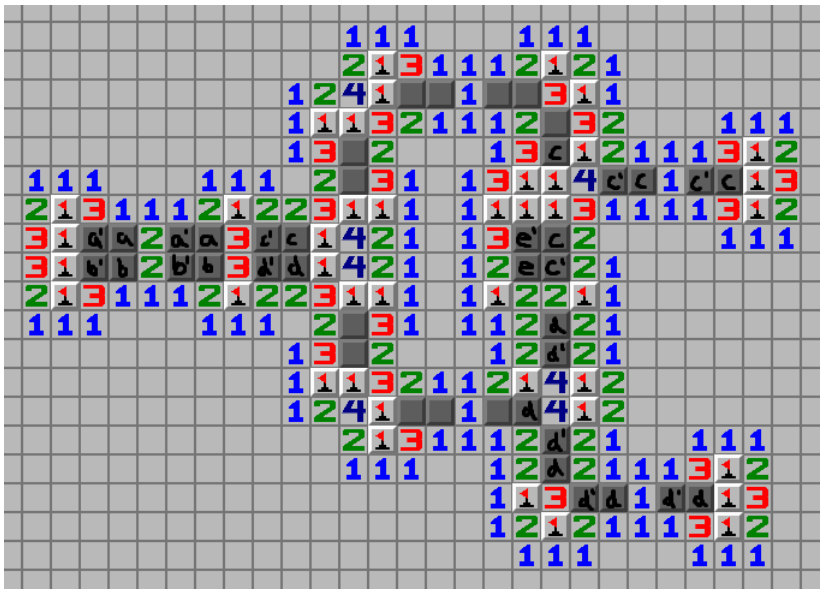

Figure 6: The gate in question, hastily annotated in Microsoft Paint.

On the left, $a + b + c' + d' = 2$, implying $a + b = c + d$. On the right, $c' + d + e = 1$. Note that c is a OR b, d is a AND b, and e is a XOR b, though it's directed straight into a wall.

4 Desirable Properties for Building Minesweeper

Richard Kaye concerned himself with verifying screenshots, so his components didn't have certain properties. You don't have to be concerned with the specifics, but here they are:

1. Components should be discoverable - that is, if you start a fresh game and only reveal one tile, you should be able to reveal all the constraints for each component.

2. Components should have a constant number of mines, no matter what state they're in. Minesweeper has a counter of remaining mines on the board, and I don't want to leak information through it. In fact, if you trust that I've filled all of my mines completely full of mines, you can count how many mines are remaining in the Minesweeper game that I've built. This is the only case where I require trust from the player.

3. Components with some inputs/outputs revealed should not leak information. For instance, something that guarantees that $a+b+c+d=2$ should not reveal $b+c$ after you know that $a=0$.

5 Construction

Minesweeper is a game made of squares, so I would like to make a grid of squares. Inside each of these squares should be either a constraint on how many of its 8 neighbors are mines, or a mine.

5.1 Cells with mines

I fill every last cell inside them with mines. If you click on it, you will lose the game, just as in regular Minesweeper.

5.2 Safe cells

To tell the player the sum of 8 neighbors, I'll have to get 8 inputs from the neighbors. By putting two of these inputs into a gate and taking the OR and AND as outputs, I can force 01 to become 10 while letting 00, 10, and 11 output themselves. By wiring up 16 instances of this 2-output gate, I can turn the 8 inputs into 8 outputs, where the sums are equal, and the inputs have been mostly sorted. I can then wire up the top 4 outputs to one constraint, and the bottom 4 outputs to another constraint. For cells with ≤ 4 neighboring mines, all mines will be in the highest 4 outputs. So, we can tell the player the sum of the top 4 outputs, and that the bottom 4 outputs sum to 0. For cells with a value ≥ 4, all safe cells will be in the lowest 4 outputs. So, we can set the sum of the top 4 outputs to 4, and the sum of the bottom 4 outputs to the number of neighboring mines minus 4. Since none of the gates leak information, this tells the player only the sum of the inputs, and it doesn't reveal anything about individual inputs.

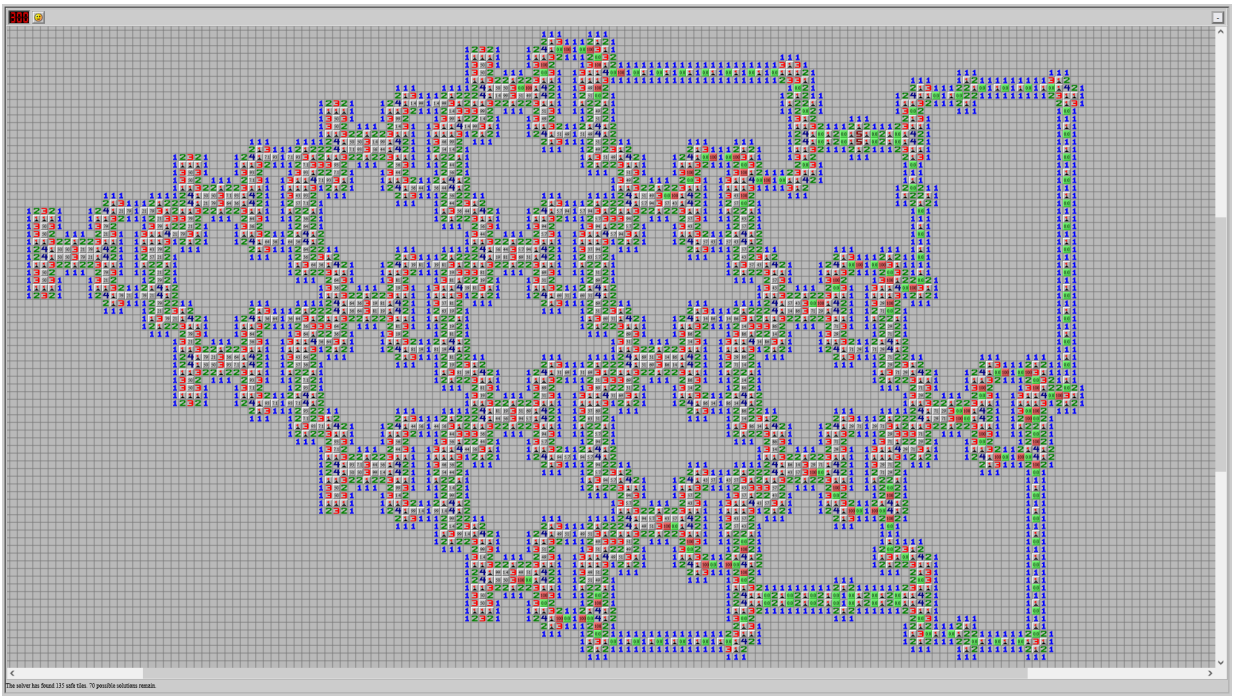

Figure 7: The inside of a safe cell, with the outputs wired up. You may be able to zoom in.

5.3 Inter-cell wiring

It's simple in theory! Each cell needs to have 8 inputs from its neighbors, and also broadcast its state to its neighbors. But, the player needs to know for certain that a cell is broadcasting 0 to every neighbor iff it's safe, and 1 to every neighbor iff it's a mine. To accomplish this, there's a wire surrounding every cell that gets split off to each neighbor. This ensures that all outputs are the same. This wire also connects to each cell, allowing the player to enter that cell if it's a 0, and stopping dead at the entrance to the cell if it's a 1.
As for the inputs, they can simply cross over the boundary of each cell. But wait! If they do that, one possible state of the wire will reveal that the interior of the cell isn't full of mines! This has the disastrous consequence of leaking information! To solve this, I made a very unfortunate choice to OR each input with the state of the cell. So, safe cells receive all 8 inputs as normal, and mines receive 8 1's. Then, 1's are flush with the boundary of the cell, and 0's reveal the interior. This has the slightly less unfortunate consequence of making a staggering 35-wide interstitial space, but... it's fine. Each cell is 142x142 anyways. I'll spare you the details of fitting all this together. There's a bunch of gates, and messy wiring, and the space at the corner where 4 cells meet has a 4-way junction as if it's an intersection of two roads that you can drive on.

6 Conclusion

You can make Minesweeper in Minesweeper! Who would have guessed? It looks like this:

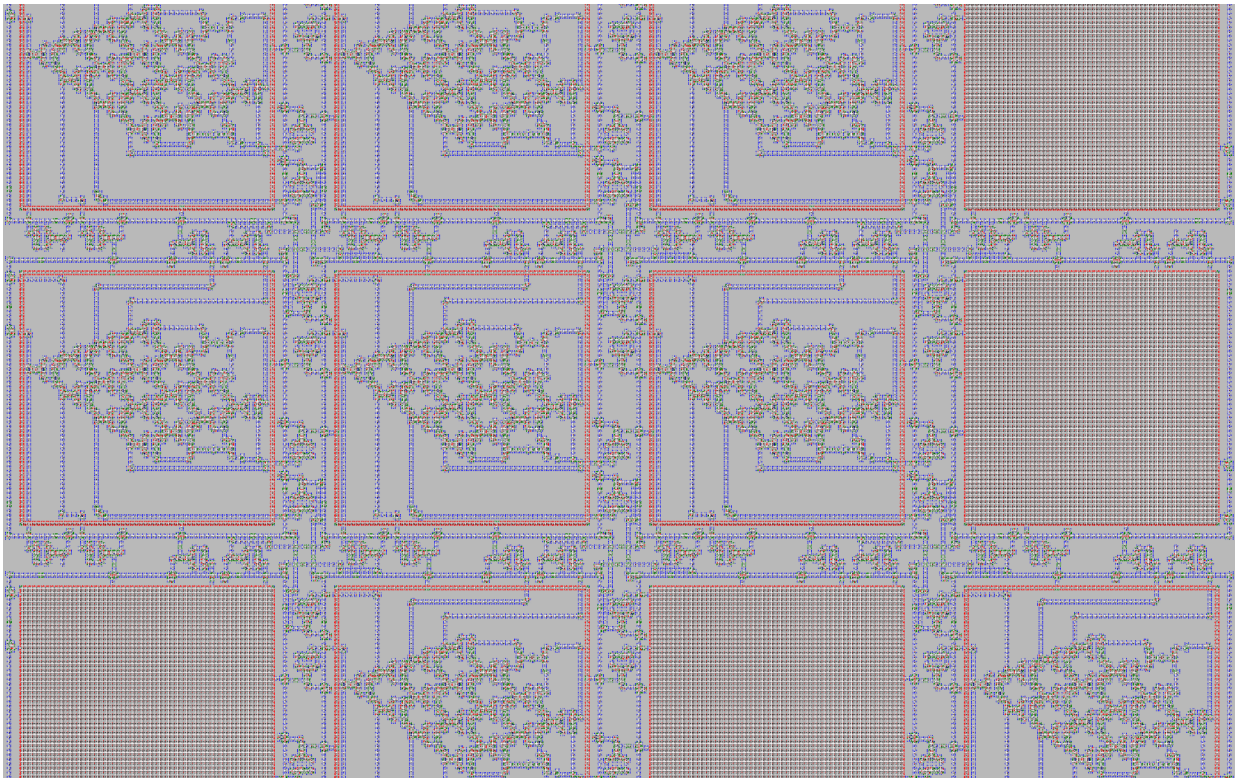

Figure 8: A snapshot of a 4x4 board, with 8 safe cells and 4 mines "visible".

7 Future Work

It's looking somewhat likely that I'll make a YouTube video about this, and put it on my channel.[4] Perhaps by the time you're reading this, it's already up! Maybe I'll have footage of subjecting some poor player to this board, if you'd rather watch someone suffer through it than go to my GitHub[5] and play it yourself. This whole design could also probably be made somewhat smaller. I didn't try too hard to optimize anything after I built functional components.

8 Acknowledgements

Many thanks to the creator of JSMinesweeper[6] for making the current highest-winrate solver for Minesweeper, making an online board editor, and recently implementing a few features that allow my board to be played by a human.
Thank you to SIGBOVIK for accepting my incredibly prestigious and very serious paper.
And you, the reader!

[4] https://youtube.com/@Kirby703/videos
[5] https://github.com/Kirby703/minesweeper-in-minesweeper
[6] https://davidnhill.github.io/JSMinesweeper