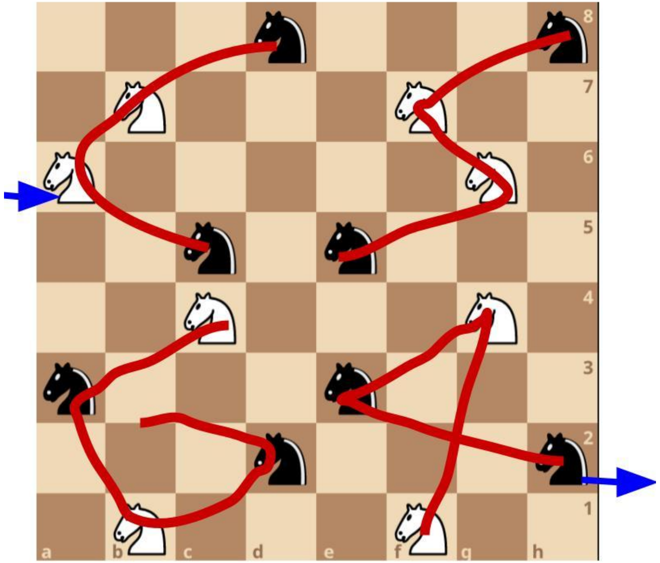
CS64: Computation for Puzzles and Games



Autumn 2022

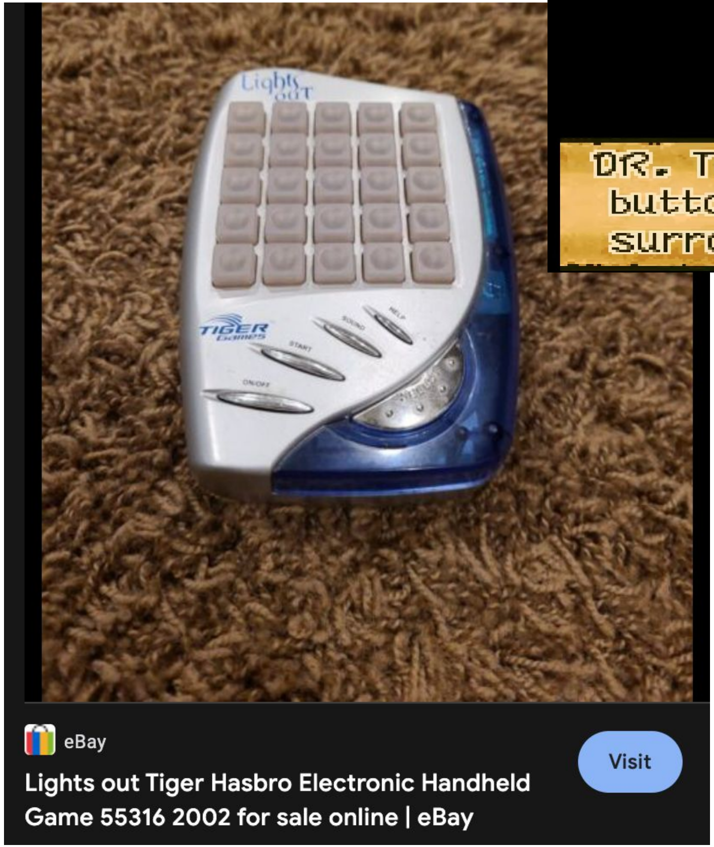
Lecture 5: Lights Out!

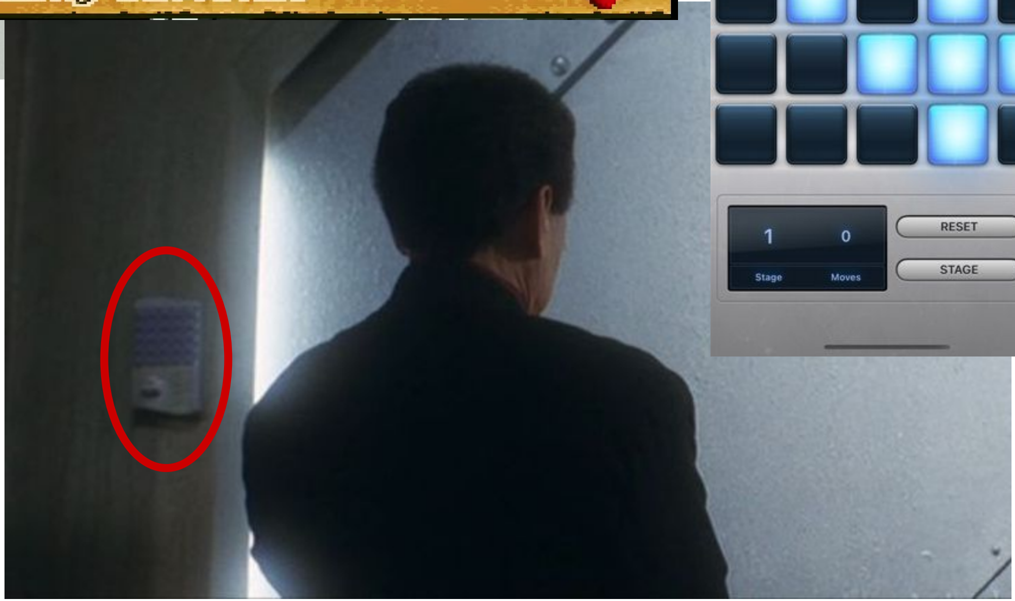
Chess scandal update

Organizers of this week's World Fischer Random Chess Championship have introduced unprecedented new security measures to prevent cheating.

Among the tighter measures at the tournament which starts on Tuesday in Reykjavik, Iceland, is the presence of a medical doctor during the five-day event who will select players and inspect their ears for any transmitters, World Fischer Random organizer Joran Aulin-Jansson told DW.

Lights Out





https://www.jaapsch.net/puzzles/lights.htm#quiet

The rules

- 5x5 grid of buttons, some are initially lit
- Pushing a button toggles the state of that button and its (up to four) orthogonal neighbors
- The goal is – as the name implies – to get all the lights to be off

Exciting live demo!

A heuristic that doesn't work well

- Try to minimize the number of buttons that are on

This configuration with only two lights on...

...requires 13 button presses!

Some useful observations

- The order of the presses does not matter. *(Why not?)*

Some useful observations

- The order of the presses does not matter.
  - Each button's final state is determined entirely by how many total times it and its neighbors were pressed.

Some useful observations

- The order of the presses does not matter.
  - Each button's final state is determined entirely by how many total times it and its neighbors were pressed.

- Because of this, there is no reason to press any individual button more than once.

Strategy 1: Brute force

- Breadth-first search!

- Try each of the 25 possible starting moves.
  - Try each of the 25 possible starting moves from those configurations.
    - etc. etc., repeat until all lights are out

Strategy 1: Brute force

- Breadth-first search!

- Try each of the 25 possible starting moves.
  - Try each of the 25 possible starting moves from those configurations.
    - etc. etc., repeat until all lights are out

- Some optimizations:
  - Keep track of which states we've seen, and don't re-explore those
  - Also keep track of which buttons have been pressed, and don't press a button twice

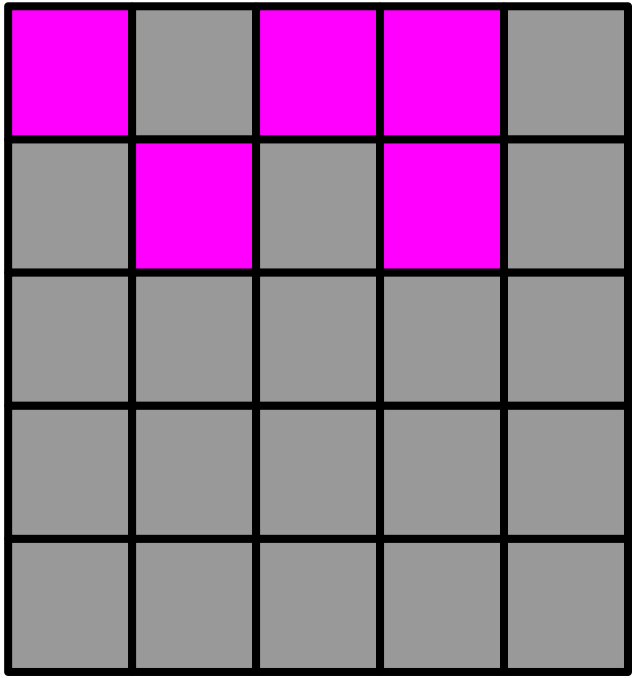
```python
def is_solved(grid):
    for r in grid:
        if True in r:
            return False
    return True

def explore(grid):
    result = []
    for i in range(5):
        for j in range(5):
            new_grid = [list(r) for r in grid]
            new_grid[i][j] = not new_grid[i][j]
            for h, v in ((-1, 0), (1, 0), (0, -1), (0, 1)):
                if (0 <= i+h < 5 and 0 <= j+v < 5):
                    new_grid[i+h][j+v] = not new_grid[i+h][j+v]
            result.append((tuple([tuple(r) for r in new_grid]), i, j))
    return result
```

```python
def bfsolve(grid):
    seen = set()
    seen.add(grid)
    current_band = [(grid, [])]
    steps = 1
    while current_band:
        print("Trying {} steps away...".format(steps))
        new_band = []
        for grid, sofar in current_band:
            for new_grid, i, j in explore(grid):
                if (i, j) in sofar:
                    continue # don't push the same button more than once
                new_sofar = sofar + [(i, j)]
                if is_solved(new_grid):
                    return new_sofar
                if new_grid not in seen:
                    new_band.append((new_grid, new_sofar))
                    seen.add(new_grid)
        current_band = new_band
        steps += 1
    return 'No solution.'

grid = tuple([tuple([x == '1' for x in input()]) for _ in range(5)])
moves = bfsolve(grid)
new_grid = [['0' for c in range(5)] for r in range(5)]
for i, j in moves:
    new_grid[i][j] = '1'
print('\n'.join([''.join(r) for r in new_grid]))
```

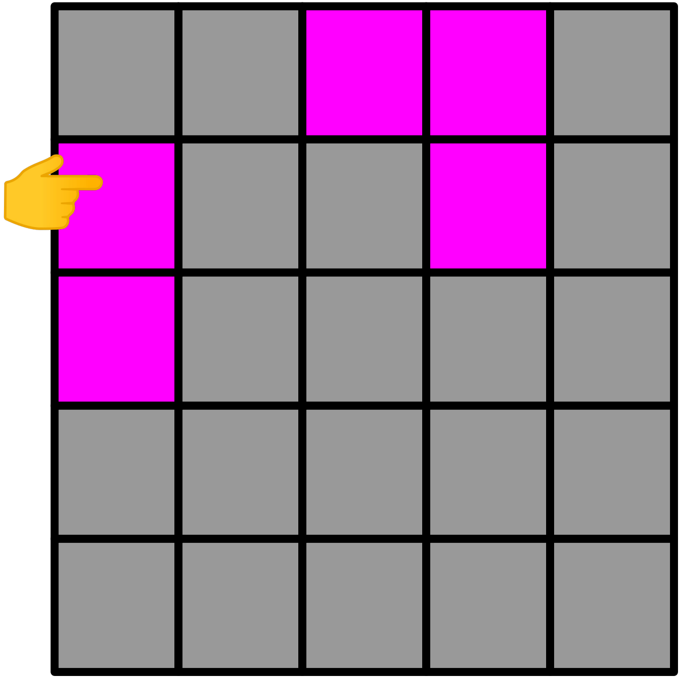
Can we do better?



What about the following strategy:

go through the second row pushing all the buttons below any lights that are on in the first row...

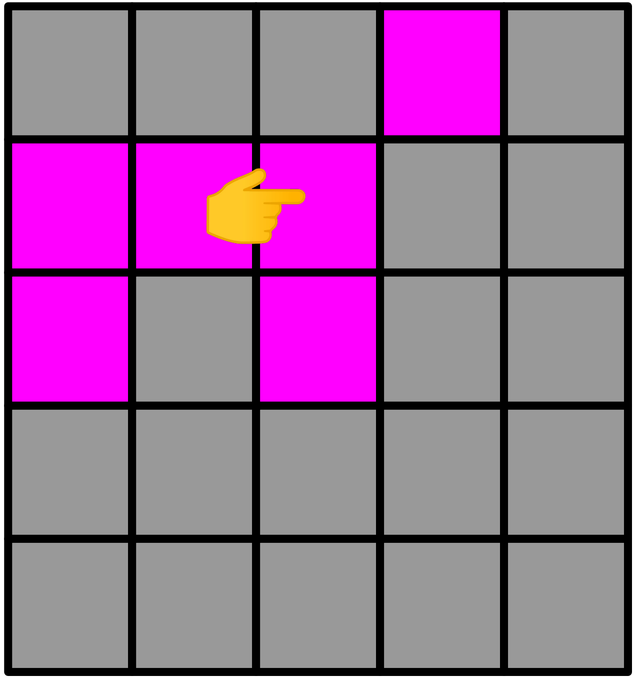
Can we do better?



What about the following strategy:

go through the second row pushing all the buttons below any lights that are on in the first row...

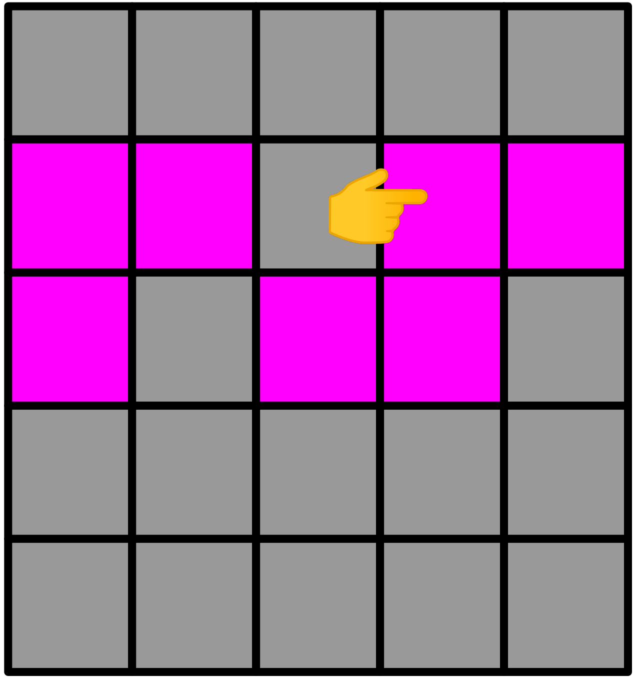
Can we do better?



What about the following strategy:

go through the second row pushing all the buttons below any lights that are on in the first row...

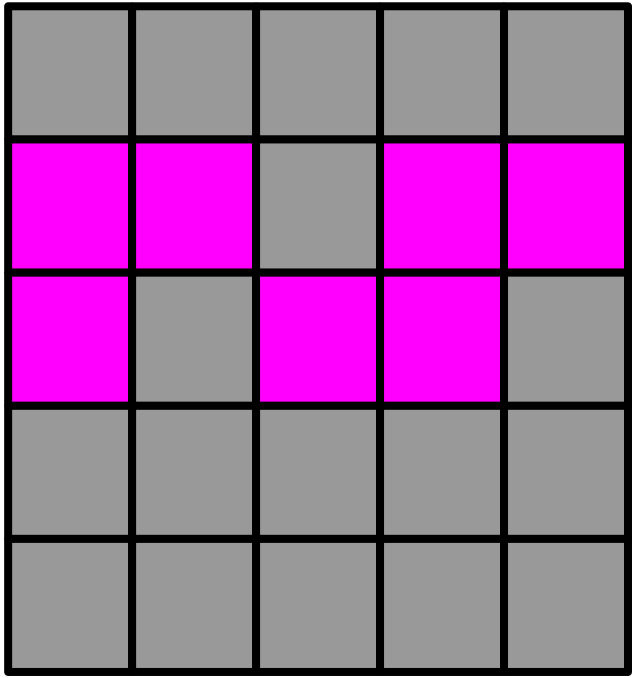
Can we do better?



What about the following strategy:

go through the second row pushing all the buttons below any lights that are on in the first row...

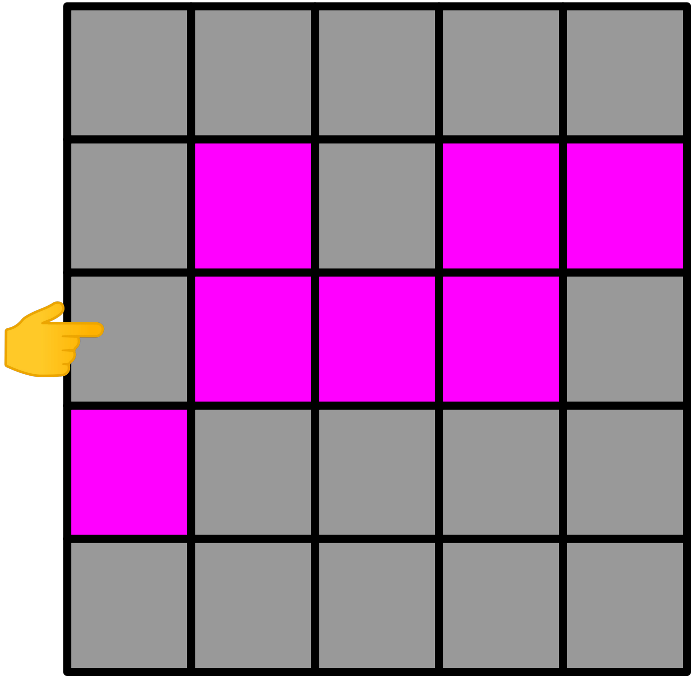
Can we do better?



What about the following strategy:

...then repeat for the third row...

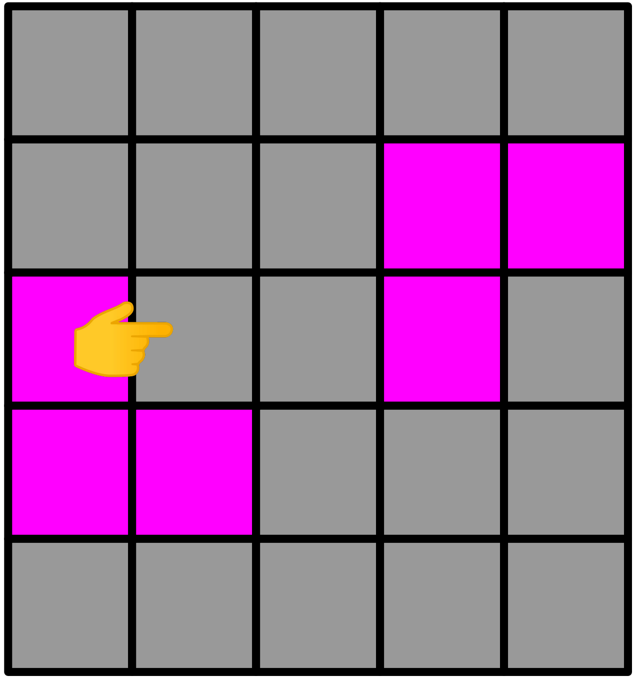
Can we do better?



What about the following strategy:

...then repeat for the third row...

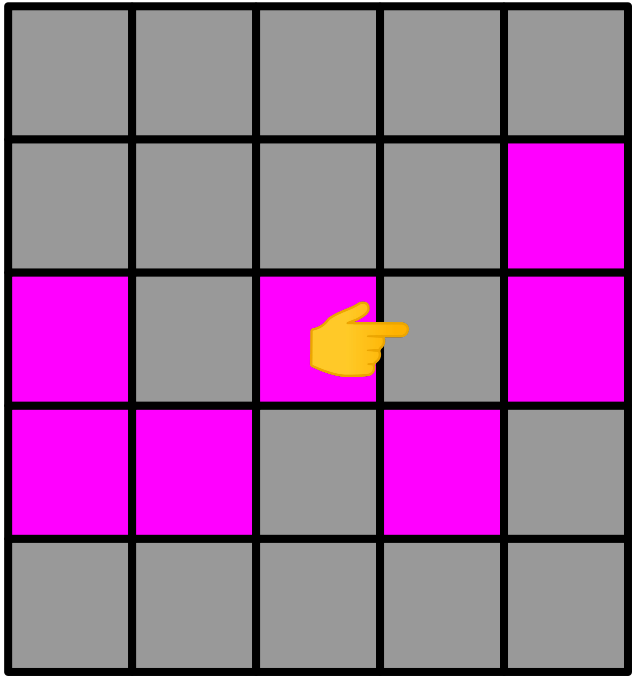
Can we do better?



What about the following strategy:

...then repeat for the third row...

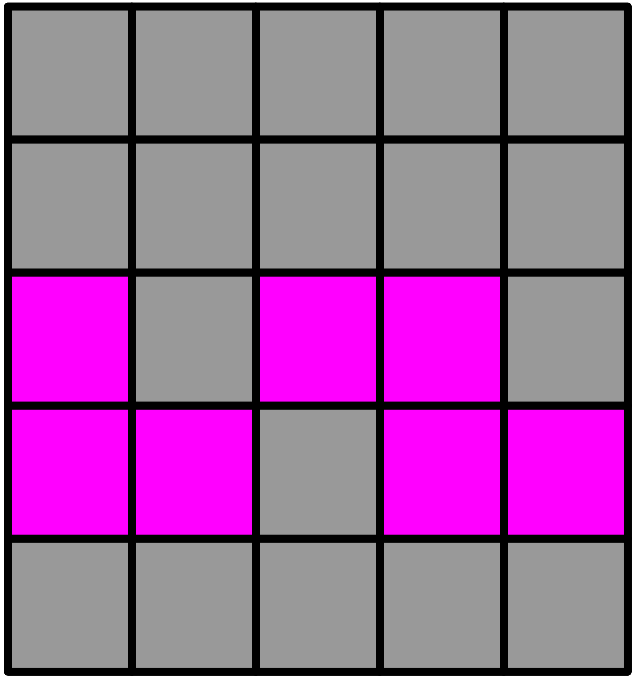
Can we do better?



What about the following strategy:

...then repeat for the third row...

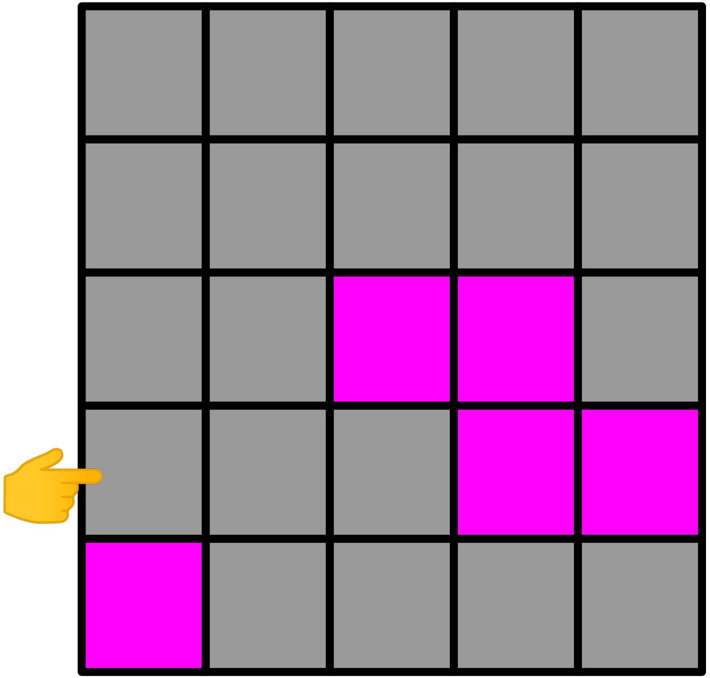
Can we do better?



What about the following strategy:

...and so on

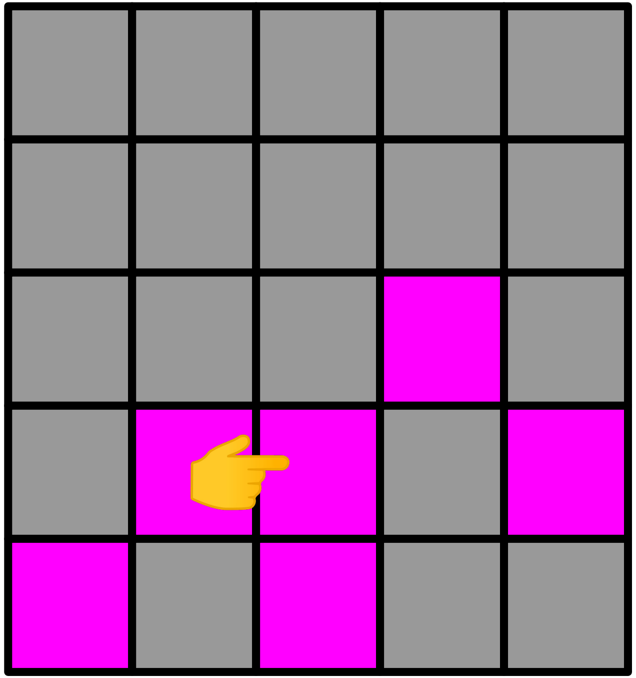
Can we do better?



What about the following strategy:

...and so on

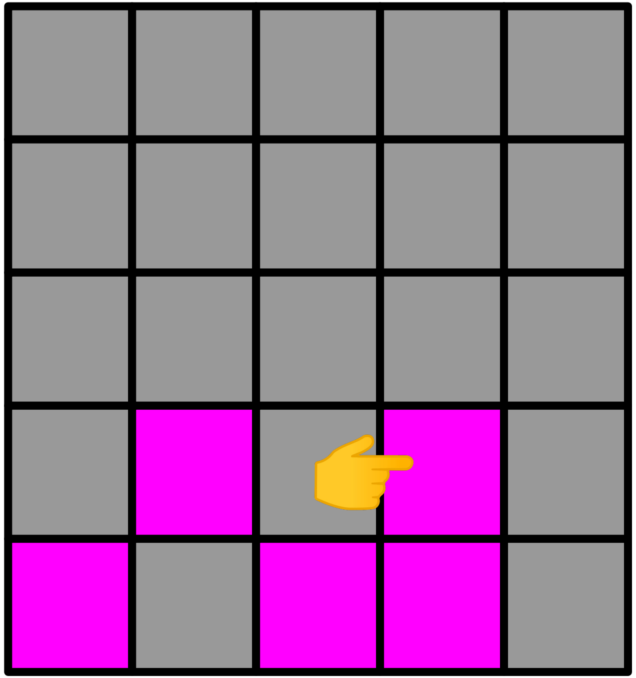
Can we do better?



What about the following strategy:

...and so on

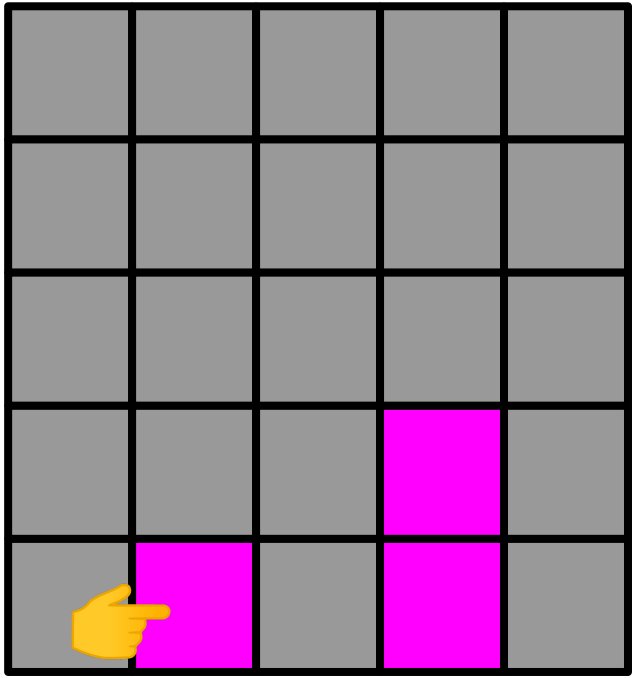
Can we do better?



What about the following strategy:

...and so on

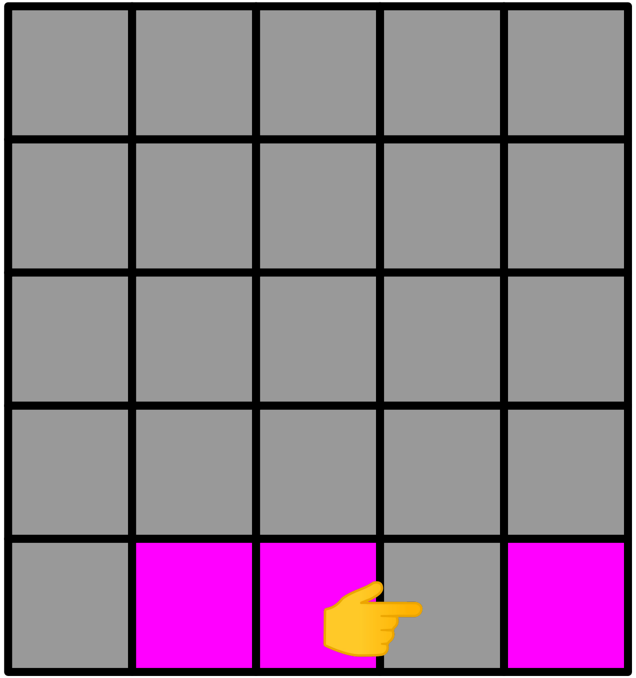
Can we do better?



What about the following strategy:

...and so on

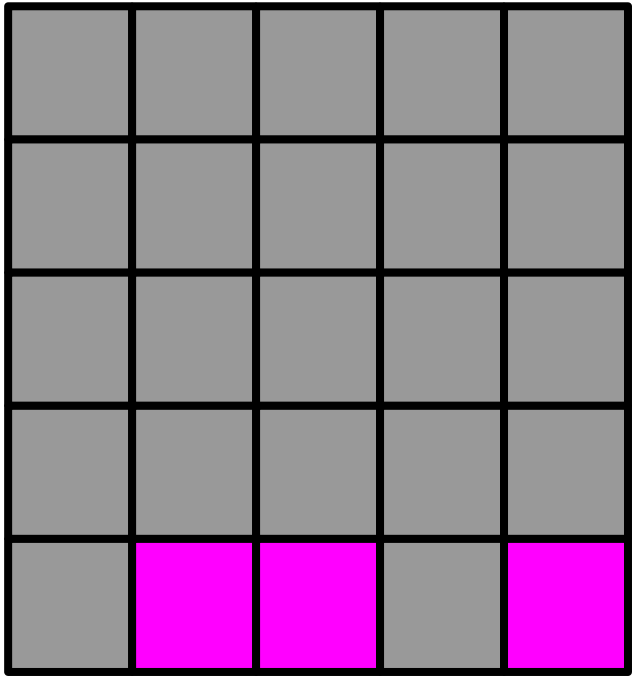
Can we do better?



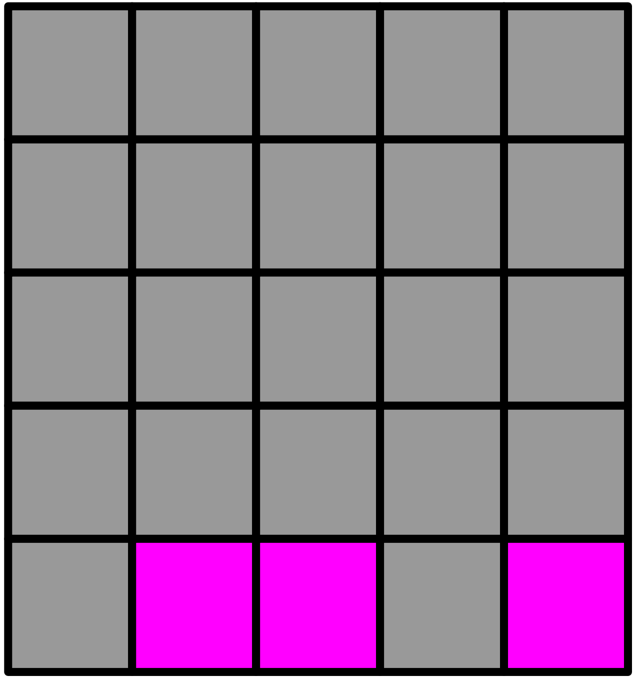
What about the following strategy:

...and so on

Can we do better?

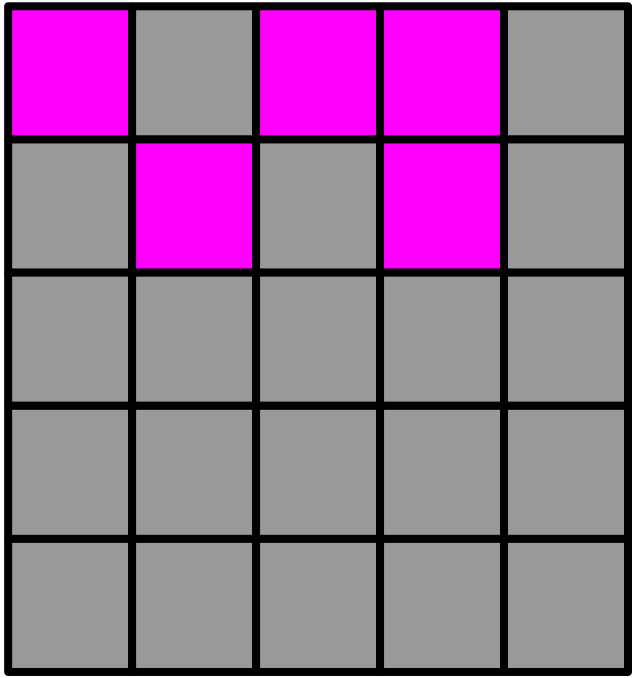


Now what



- Galaxy brain: Turn it over and do the same thing again?
    - Unfortunately, in this case, this just puts us back in the exact same situation...

Let's back up

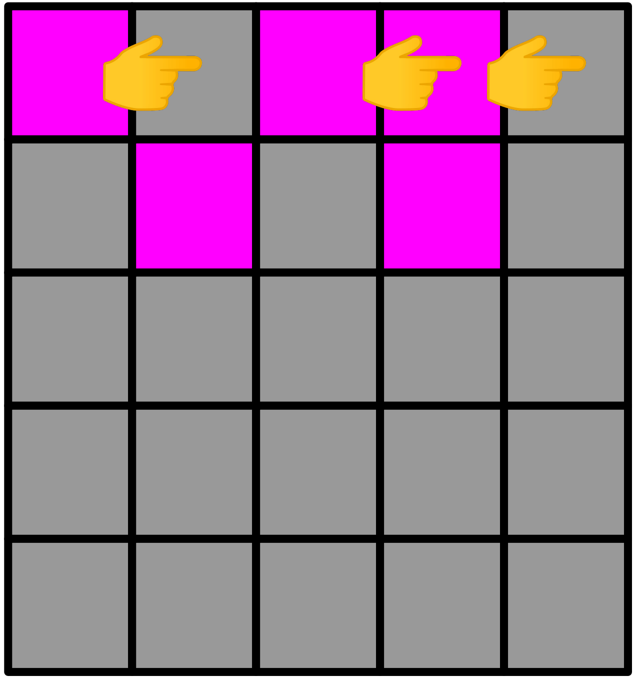


The idea here was on the right track. Once we choose our button presses in the first row, the rest of the solve process is totally determined.

There are $2^5 = 32$ ways to choose what to do in the first row.

So… try all of them and see if any of them work!

Let's back up



The idea here was on the right track. Once we choose our button presses in the first row, the rest of the solve process is totally determined.

There are $2^5 = 32$ ways to choose what to do in the first row.

So... try all of them and see if any of them work!

```python
import itertools

# All possible binary strings of length 5
POSSIBLE_PATTERNS = [''.join(x) for x in itertools.product('01', repeat=5)]

def flip(c):
    return '1' if c == '0' else '0'

def apply_pattern_to_row(pattern, row):
    new_row = row[:]
    for c in range(5):
        if pattern[c] == '1':
            for cc in range(c-1, c+2):
                if 0 <= cc < 5:
                    new_row = new_row[0:cc] + flip(new_row[cc]) + new_row[cc+1:]
    return new_row

grid = [input() for _ in range(5)] + ['00000'] # add dummy extra row for convenience
best_solution = None
best_count = 25
for p in POSSIBLE_PATTERNS:
    curr_pattern = p
    solution = []
    curr_row = grid[0]
    for i in range(5):
        next_pattern = apply_pattern_to_row(curr_pattern, curr_row)
        next_row = ''
        for j in range(5):
            next_row += flip(grid[i+1][j]) if curr_pattern[j] == '1' else grid[i+1][j]
        curr_row = next_row
        solution.append(curr_pattern)
        curr_pattern = next_pattern
    if next_pattern == '00000': # we don't care about next_row now since it's off the board
        press_count = sum([r.count('1') for r in solution])
        if press_count < best_count:
            best_count = press_count
            best_solution = solution
print('No solution' if not best_solution else'\n' + '\n'.join(best_solution))
```

If you use Python and you like puzzles, the itertools library is indispensable

When is it solvable?

Say you want to make a board and hand it to your younger sibling...

Are there unsolvable puzzles? If so, how many?

Everything is linear algebra

- The grid is a 5x5 matrix of 1s and 0s. We are working over the finite field F_2 (basically 1 + 1 = 0)

- Pushing a button is like adding another matrix. E.g., here's the matrix corresponding to pushing the middle button:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Write each button press operation as a column vector like this.

There are 25 such vectors, one for each button.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Write each button press operation as a column vector like this.

There are 25 such vectors, one for each button.

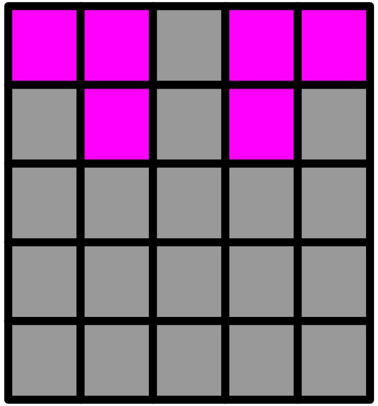
We can stick these together in a 25x25 matrix. Call it *M*.

```
1100010000000000000000000
1110001000000000000000000
0111000100000000000000000
0011100010000000000000000
0001100001000000000000000
1000011000100000000000000
0100011100010000000000000
0010001110001000000000000
0001000111000100000000000
0000100011000010000000000
0000010000110001000000000
0000001000111000100000000
0000000100011100010000000
0000000010001110001000000
0000000001000110000100000
0000000000100001100010000
0000000000010001110001000
0000000000001000111000100
0000000000000100011100010
0000000000000010001100001
0000000000000001000011000
0000000000000000100011100
0000000000000000010001110
0000000000000000001000111
0000000000000000000100011
```

Encode the grid state itself as the column vector g.

Then we want some solution vector s such that $Ms + g = 0$.

(Each entry of s corresponds to "do I use this column vector or not?", i.e., "do I push this button or not?)



$$
\begin{matrix}
1100010000000000000000000 \\
1110001000000000000000000 \\
0111000100000000000000000 \\
0011100010000000000000000 \\
0001100001000000000000000 \\
1000011000100000000000000 \\
0100011100010000000000000 \\
0010001110001000000000000 \\
0001000111000100000000000 \\
0000100011000010000000000 \\
0000010000110001000000000 \\
0000001000111000100000000 \\
0000000100011100010000000 \\
0000000010001110001000000 \\
0000000001000110000100000 \\
0000000000100001100010000 \\
0000000000010001110001000 \\
0000000000001000111000100 \\
0000000000000100011100010 \\
0000000000000010001100001 \\
0000000000000001000011000 \\
0000000000000000100011100 \\
0000000000000000010001110 \\
0000000000000000001000111 \\
0000000000000000000100011
\end{matrix}
\cdot
\begin{matrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}
+
\begin{matrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}
=
\begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}
$$

We're working modulo 2, so any vector plus itself is 0. Therefore we can replace

$Ms + g = 0$

with

$Ms = g$

Now, for which initial grid states g is there a solution s?

We're working modulo 2, so any vector plus itself is 0. Therefore we can replace

$Ms + g = 0$

with

$Ms = g$

Now, for which initial grid states g is there a solution s?

Invert M and check $s = M^{-1}g$? Unfortunately, M is not invertible! (This implies that not all of the buttons are really necessary. In fact, M has rank 23, and so it is possible to solve **any** solvable Lights Out puzzle without using two of the buttons at all.)

Handwaving away some more math...

It can be shown (via more linear algebra) that a configuration is solvable if and only if it is orthogonal to both of these vectors:

$$\vec{n}_1 = (0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0)^T$$

$$\vec{n}_2 = (1,0,1,0,1,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,1,0,1,0,1)^T.$$

(Orthogonal here means that the dot product of either vector with the initial state's vector is 0.)

Handwaving away some more math...

It can be shown (via more linear algebra) that a configuration is solvable if and only if it is orthogonal to both of these vectors:

$$\vec{n}_1 = (0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0)^T$$

$$\vec{n}_2 = (1,0,1,0,1,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,1,0,1,0,1)^T.$$

(Orthogonal here means that the dot product of either vector with the initial state's vector is 0.)

Implication: You can get an unsolvable state by taking any solvable state and toggling a single light (not pressing a button, just changing that one light), except in one of these positions. (They are a small X in the middle of the grid)

Handwaving away some more math...

It can be shown (via more linear algebra) that a configuration is solvable if and only if it is orthogonal to both of these vectors:

$$\vec{n}_1 = (0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0)^T$$

$$\vec{n}_2 = (1,0,1,0,1,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,1,0,1,0,1)^T.$$

(Orthogonal here means that the dot product of either vector with the initial state's vector is 0.)

Implication: You can get an unsolvable state by taking any solvable state and toggling a single light (not pressing a button, just changing that one light), except in one of these positions. (They are a small X in the middle of the grid) So for any solvable state, there are about 20 unsolvable ones, so < 5% of states are solvable?

Handwaving away some more math...

It can be shown (via more linear algebra) that a configuration is solvable if and only if it is orthogonal to both of these vectors:

$$\vec{n}_1 = (0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0)^T$$

$$\vec{n}_2 = (1,0,1,0,1,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,1,0,1,0,1)^T.$$

(Orthogonal here means that the dot product of either vector with the initial state's

No! This argument would only work if there were no overlaps, i.e., each unsolvable state were only reachable from one solvable state. But this turns out to be very untrue.

Impli and toggli), excep rid)

So for any solvable state, there are about 20 unsolvable ones, so < 5% of states are solvable?

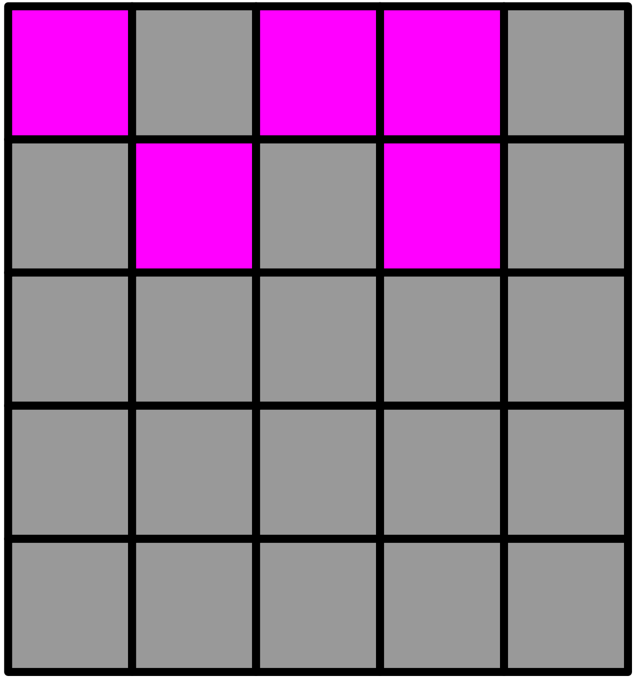
Some final facts

- The actual proportion of solvable initial states is ¼.
    - Instruction manual: "It is possible to create a puzzle so difficult that it may not have a solution!"
    - There are three other "worlds" that you can be stuck in forever!
    - Mean tip: start with a grid with just the top left light on (an unsolvable state), push buttons a bunch of times, then give that puzzle to your younger sibling.
        - What if they start recognizing previously seen bad states and giving up? Try a different one of the three bad worlds
- For any solvable state, there are actually four solutions (recall that two buttons don't matter)

Is this problem tractable?

- We have a pretty fast program for the 5x5 board!
- Recall that any fixed-size game has a constant time solution (however huge the constant!), but we care about how the solving time scales with the size of the game.
- Our method could be extended to arbitrarily sized square (even nonsquare) boards...

Let's back up



The idea here was on the right track. Once we choose our button presses in the first row, the rest of the solve process is totally determined.

There are $2^5 = 32$ ways to choose what to do in the first row.

Oh no! Our algorithm has an exponential component! So this isn't a polynomial-time solution.

In fact, this problem is also NP-complete. Boo!