

# Not All Instances of Hard Problems are Difficult<sup>†</sup>

Will Bryan and Andrew Mertz

March 1, 2024

<sup>†</sup>and they can be a lot of fun

In this talk, we explore some problems from Advent of Code 2023 and the techniques that make these problems simpler than they first appear.

We will also look at some problems just for fun.

# What is AoC?

Advent of Code is an annual series of small programming puzzles for a variety of skill sets and skill levels in any programming language you like.

It runs from December 1<sup>st</sup> to December 25<sup>th</sup> (since 2015).

Here are the current completion statistics for each day. Gold indicates users that have completed both parts of a puzzle, while silver indicates users that have completed only the first half. Each \* or \* star represents up to 7755 users.

25	11188	3070	***
24	12504	5014	***
23	15233	2992	***
22	16484	1001	****
21	14936	10316	****
20	18964	4486	****
19	23210	7295	****
18	25496	4988	*****
17	24701	1128	*****
16	36168	1061	*****
15	41723	4250	*****
14	37274	7582	*****
13	38957	5309	*****
12	31750	14617	*****
11	58390	2438	*****
10	49160	17346	*****
9	77739	1283	*****
8	75250	14948	*****
7	83178	7468	*****
6	104874	1985	*****
5	81386	31578	*****
4	132136	18012	*****
3	132321	20318	*****
2	199469	9535	*****
1	234418	75774	*****

[adventofcode.com/2023/stats](https://adventofcode.com/2023/stats)

# Private Leaderboard

This is the private leaderboard of Will for Advent of Code 2023. You can use a different [Ordering], manage your [Private Leaderboards], use an [API], or switch to another [Event].
















Gold indicates the user got both stars for that day, silver means just the first star, and gray means none.

		1111111111222222	
		1234567890123456789012345	
1)	478	*****	ZeroTau (AoC++)
2)	420	*****	Will (AoC++)
3)	352	*****	Sgr A8
4)	182	*****	one-reader
5)	53	*****	Austin Stortzum
6)	37	*****	zemkat
7)	25	*****	Jack Schmidt
8)	7	*****	Abby Gamboa
9)	2	*****	Guyfee
10)	0	*****	(anonymous user #3280719)

# Global Leaderboard

Below is the Advent of Code 2023 overall leaderboard; these are the 100 users with the highest total score. Getting a star first is worth 100 points, second is 99, and so on down to 1 point at 100th place.

You can change how you appear here on the [\[Settings\]](#) page. You can also view your own [\[Personal Times\]](#) or use a [\[Private Leaderboard\]](#).

1)	3257		xiaowucl
2)	3174		tckmn
3)	2909		5space (AoC++)
4)	2486		nthistle (AoC++) (Sponsor)
5)	2484		jonathanpaulson (AoC++)
6)	2476		Antonio Molina (AoC++) (Sponsor)
7)	2404		dan-simon
8)	2370		bluepichu
9)	2285		leijurv (AoC++)
10)	2241		boboquack
11)	2226		hyper-neutrino
12)	2198		D. Salgado
13)	2092		Ian DeHaan
14)	2046		Noble Mushtak
15)	2034		Anish Singhani (AoC++)

[adventofcode.com/2023/leaderboard](https://adventofcode.com/2023/leaderboard)

# Why do contests?

- Fun
- Learning
- Community
- Profit?

Once you see it...





## Day 1: Sum of Digits

This problem asks us to parse lines of input to find the first and last digits contained within.

Then combine the first digit and the last digit to form a single two-digit number, and sum all such numbers.

The catch is the digits could be spelled out or written as numbers.

## Day 1: Example

two|nine  
eightwo|three  
abc|one2|three|xyz  
xtwo|one3|four  
4|nine|eight|seven2  
zone|eight2|34  
7pqr|st|sixteen

Yields the sum  $29 + 83 + 13 + 24 + 42 + 14 + 76 = 281$ .

## Day 1: Possible Approaches

- Use a regular expression
- Build our own parser
- Use tools like sed or awk
- Use a parser generator like ANTLR

Note the input is small (around 22KB).

While we can find the digits with only one pass over the input. Even if we take multiple passes, we can still solve the problem quickly for input this small.

# Day 1: Solutions

- Python
- Bash pipeline
- Circuit

## Day 16: Light Propagation

This problem asks us to illuminate as many cells of a room as possible by shining a light from any cell on an outer edge.

This is complicated by the fact that there are mirrors and beam splitters in the room.

## Day 16: Rules

If the beam encounters empty space (.), it continues in the same direction.

If the beam encounters a mirror (/ or \), the beam is reflected 90 degrees depending on the angle of the mirror.

If the beam encounters the pointy end of a splitter (| or -), the beam passes as if the splitter were empty space.

If the beam encounters the flat side of a splitter (| or -), the beam is split into two beams going in each of the two directions the splitter's pointy ends are pointing.

Beams do not interact with other beams

## Day 16: Example

.   . . . \ . . . .	>   < < < \ . . . .
. - . \ . . . .	v - . \ ^ . . . .
. . . .   - . . .	. v . . .   - > > >
. . . . . .   .	. v . . . v ^ .   .
. . . . . . . .	. v . . . v ^ . . .
. . . . . . . \	. v . . . v ^ . . \
. . . . / . \ \ . .	. v . . / 2 \ \ . .
. - . - / . .   . .	< - > - / v v   . .
.   . . . . -   . \	.   < < < 2 -   . \
. . / / .   . . . .	. v / / .   . v . .

In this example, 46 cells are *illuminated* when light shines in from the left in the top left cell.

## Day 16: Instance Size

Our problem input is a room with 110 rows and 110 columns.

So there are 12100 cells.

Our instance has 1191 objects in the room.

What are the challenges of this problem?

Let's look at a solution.



## Day 18: Lavaduct Lagoon

Given instructions consisting of direction and distance to dig out the edge of a pit, what will the resulting area of the pit be?

## Day 18: Example

R 6

D 5

L 2

D 2

R 2

D 2

L 5

U 2

L 1

U 2

R 2

U 3

L 2

U 2

#####

#.....#

###...#

..#...#

..#...#

###.###

#...#..

##...##

.#.....#

.#####

#####

#####

#####

..#####

..#####

#####

#####..

#####

.#####

.#####

## Day 18: Bigmode

Finding the area of these small shapes isn't too bad.

Could use a left-to-right scan, flood fill, whatever, and computationally be alright. But what if it was much bigger?

Part 2: Oops, instead of numbers from 2 to 12, we meant to give you a 5 digit hexadecimal number.

Our answer has now gone from a magnitude of  $1e4$  to  $1e14$ .

## Day 18: Thinking with Shapes

Storing and computing is now out of the question.\*

New approach: thinking. What is it that we need the area of?

How can we get the area of that?

## Day 18: Shoelace, Gauss, Surveyor's, etc.

$$A = \frac{1}{2} \sum_{i=1}^n (y_i + y_{i+1})(x_i - x_{i+1})$$

A-----B

| . . . . |

N-M . . . |

. . | . . . |

. . | . . . |

K-L . D-C

| . . . | . .

J I . . E-F

. | . . . . |

. H-----G

$$(y_A + y_B)(x_A - x_B)$$

$$(y_B + y_C)(x_B - x_C)$$

$$(y_C + y_D)(x_C - x_D)$$

$$(y_D + y_E)(x_D - x_E)$$

...

$$(y_N + y_A)(x_N - x_A)$$

## Day 18: Pick's Theorem

Our area calculation is still a smidge short.

If we consider ourselves to be digging out a cube block of dirt, we're standing in the center of the block and therefore not counting all of it when digging.

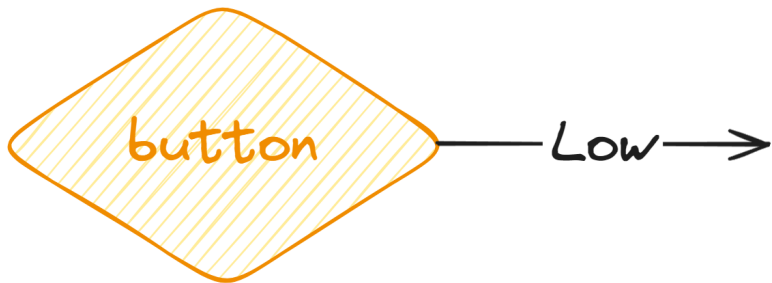
Enter Pick's Theorem:  $A = i + \frac{b}{2} - 1$

## Day 20: Pulse Propagation

You have a collection of modules that can send and receive high or low pulses connected together. There are three main types:

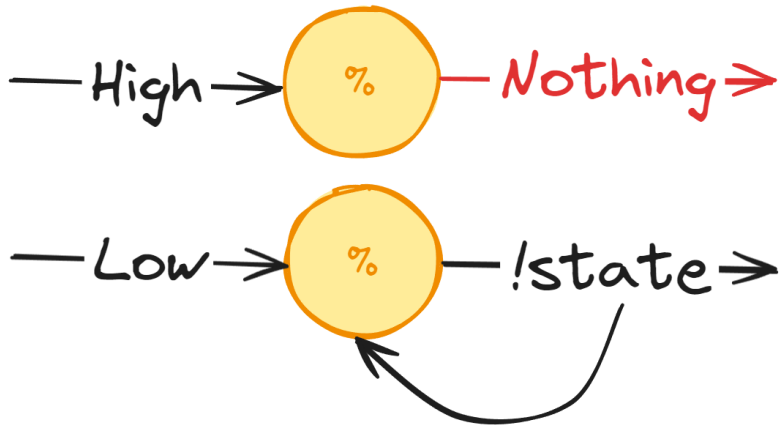
- Flip-flop modules (%), when receiving a low pulse, flip their internal state between sending a high pulse or a low pulse, defaulting to high.
- Conjunction modules (&) remember the most recent pulses from each input, defaulting their memory to low - if all pulses in memory are high, it sends a low pulse, else high.
- There is a button/broadcast module that sends a low pulse to all of its destination modules.

Pulses are processed in the order they are sent.

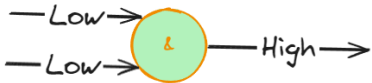
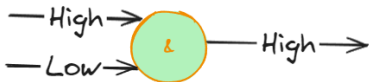
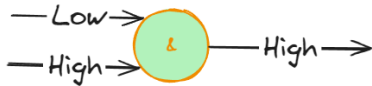
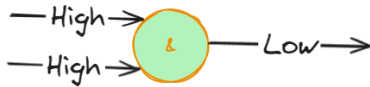




## Day 20: Flip Flop



## Day 20: Conjunction



## Day 20: Example

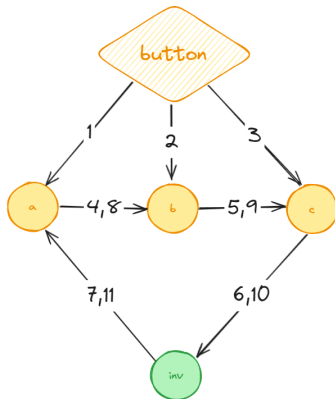
btn  $\rightarrow$  a, b, c

%a  $\rightarrow$  b

%b  $\rightarrow$  c

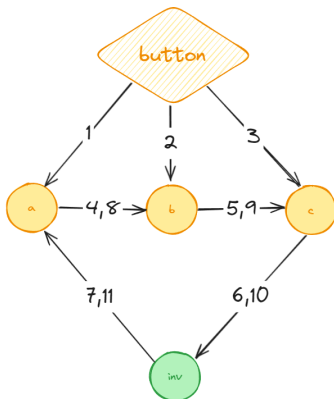
%c  $\rightarrow$  inv

&inv  $\rightarrow$  a



## Day 20: Example

btn -low-> a  
btn -low-> b  
btn -low-> c  
a -high-> b  
b -high-> c  
c -high-> inv  
inv -low-> a  
a -low-> b  
b -low-> c  
c -low-> inv  
inv -high-> a



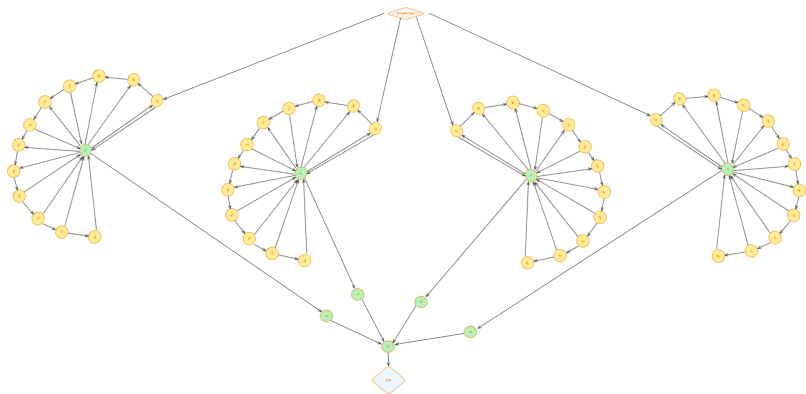
After 1,000 button presses, how many pulses were sent?

Simulation go brrr

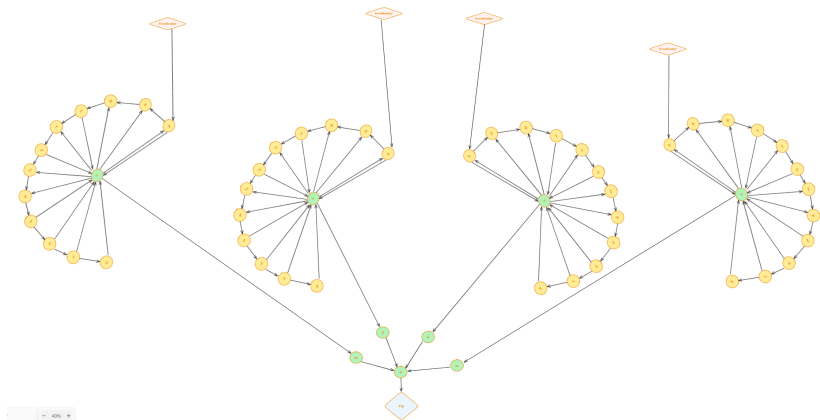
We want to know how many button presses are required to send a single low pulse to module rx.

Simulation go brrr... for a long time. A bit over 270,000 times longer. Clearly, computation is not the way to go.

## Day 20: Graph (Unified)

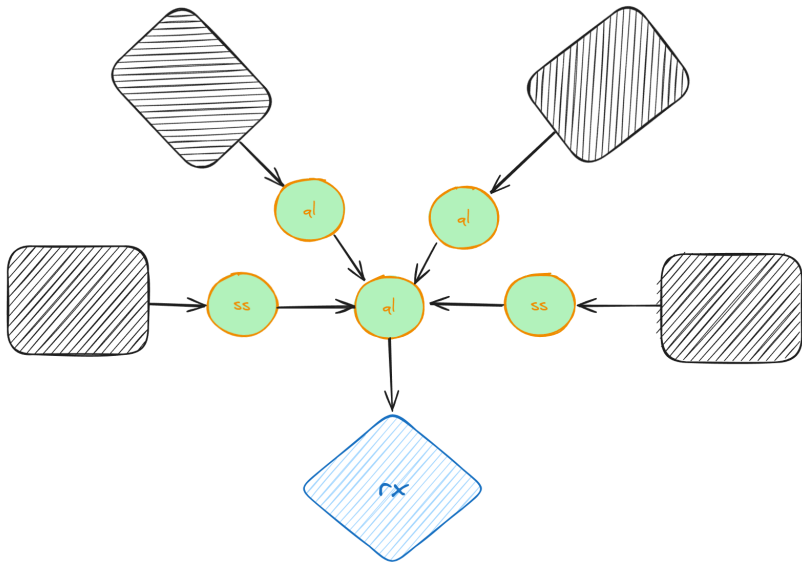


## Day 20: Graph (Separate)

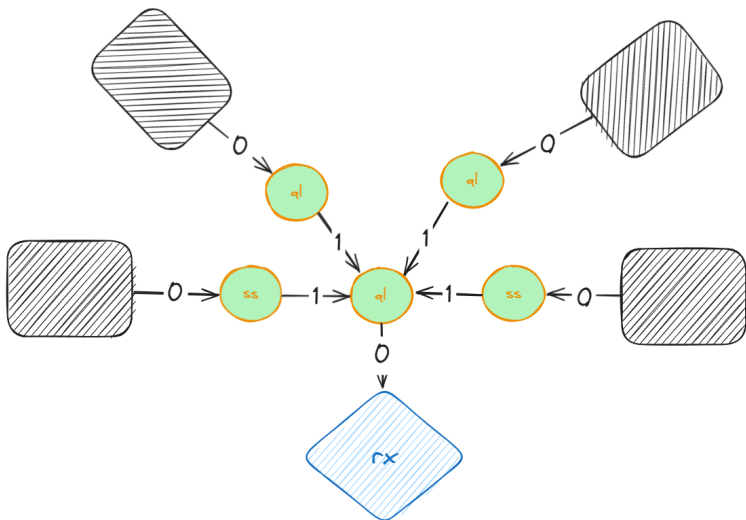




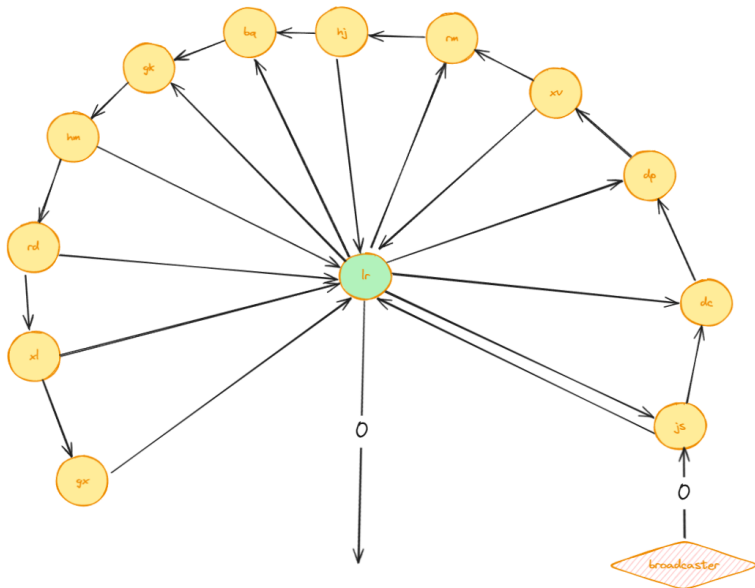
## Day 20: Graph (Black Box)



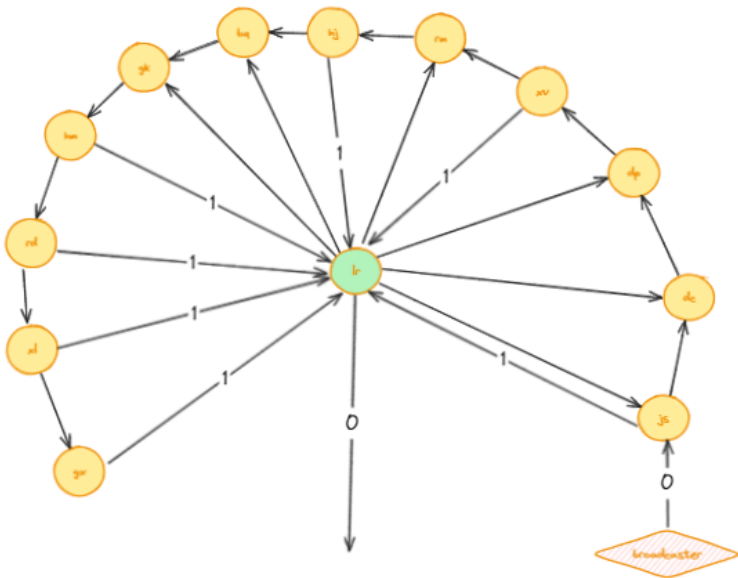
## Day 20: Graph (Black Box w/ Values)



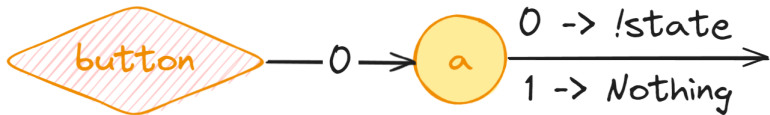
## Day 20: Subgraph



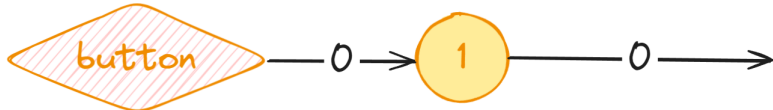
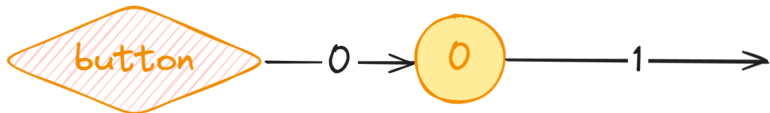
## Day 20: Subgraph (Partially Filled)



## Day 20: Flip Flop Review

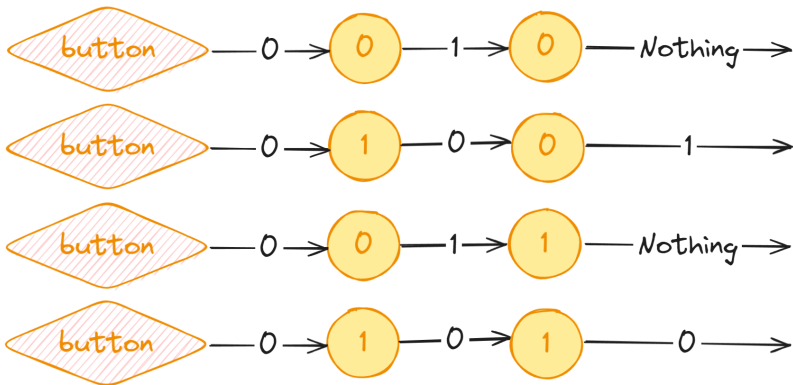


Internal State



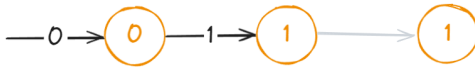
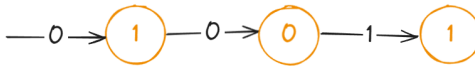
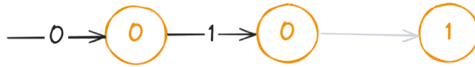
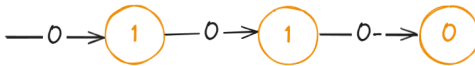
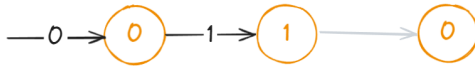
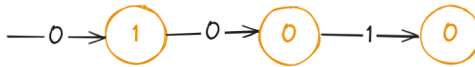
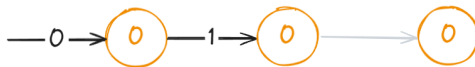
## Day 20: Flip Flop Review 2

Internal State

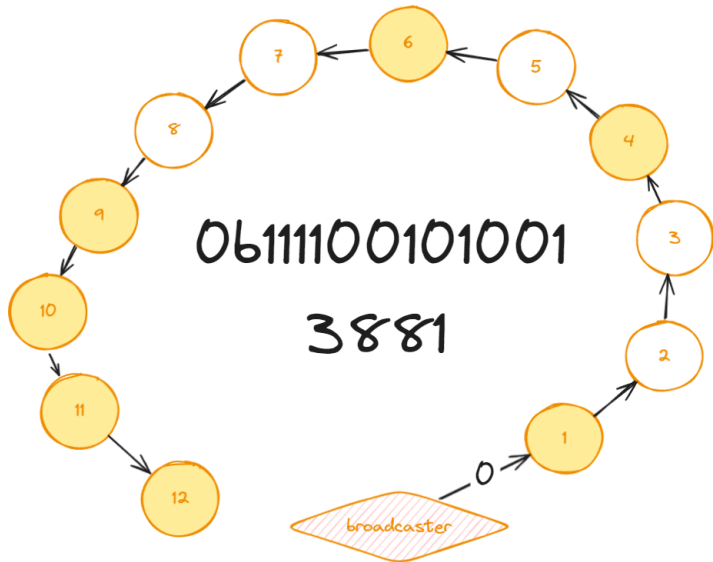


## Day 20: Flip Flop Review 3... bit

Internal State



## Day 20: Binary Counting





## Day 21: Infinite Maze

For the first part of the problem we are given a maze of open and impassable cells, and asked how many locations can be reached in exactly a given number of steps.

Note we are allowed to return locations we have already visited.

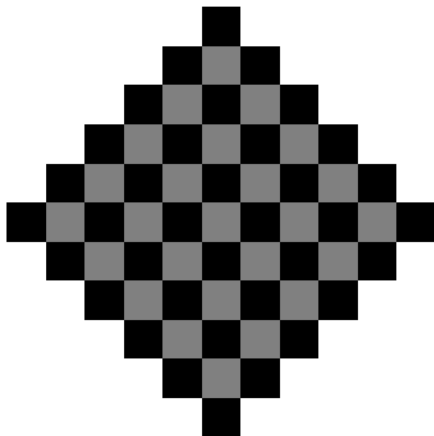
The maze has 131 rows and 131 columns. So this can be brute forced.

## Day 21: Infinite Maze

For the second part of the problem, the maze we were given is tiled in each direction to form an infinite maze.

We are asked to find the number of locations that can be reached in exactly 26,501,365 steps.

This will take a long time to brute force.



## Day 21: Instance

Let's look at our instance of the problem.

Note the structure of our maze has a lot of regularity.

Also, while the full number of steps is too large to brute force, we can solve many smaller instances.

Here is the data in Mathematica.

## Day 24: Never Tell Me The Odds

You're surrounded by hailstones, each with a given position and velocity

PX, PY, PZ @ VX, VY, VZ

Part 1: Looking forward in time, how many of the hailstones' paths will intersect within a test area?

Make it simpler:

- Forward in time:  $t > 0$
- Paths will intersect: hailstones themselves don't have to collide
- Within a test area: specific bounds
- Also, ignore the Z axis.

## Day 24: Never Tell Me The Odds

If we know a starting point and our change, what are we actually working with?

How easy is it to see if lines intersect?

## Day 24: Never Tell Me The Odds, Continued

Although the paths cross, no hailstones will actually collide on their current course. If you throw a rock just right, you can hit every hailstone!

Thanks to magic, the rock can start at any integer position and velocity.

Where and how quickly to throw the rock?

## Day 24: Never Tell Me The Odds, Continued

Now we don't know  $PX$ ,  $PY$ ,  $PZ$ ,  $VX$ ,  $VY$ , or  $VZ$  for our rock

We do know that at some time  $T$ , our rock will collide with a hailstone at position  $X, Y, Z$

No guarantee that  $T_1 = 1$ ,  $T_2 = 2$ , etc., so  $T$  is unknown.



## Day 24: Putting Together the Unknowns

So we have 7 unknowns:  $PX$ ,  $PY$ ,  $PZ$ ,  $VX$ ,  $VY$ ,  $VZ$ , and  $T$ .

What do we know?

At  $T_1$ ,  $X_{rock}, Y_{rock}, Z_{rock} = X_{hail_1}, Y_{hail_1}, Z_{hail_1}$

$$Pos_T = Pos_{Start} + Velocity(T)$$

$$X_{rock} = PX + VX(T)$$

$$X_{hail_1} = PX_{hail_1} + VX_{hail_1}(T)$$

Now we're adding in equations with known variables.

# Questions?

This talk available at [github.com/ZeroTau/AoC2023Talk](https://github.com/ZeroTau/AoC2023Talk)

Thank You!