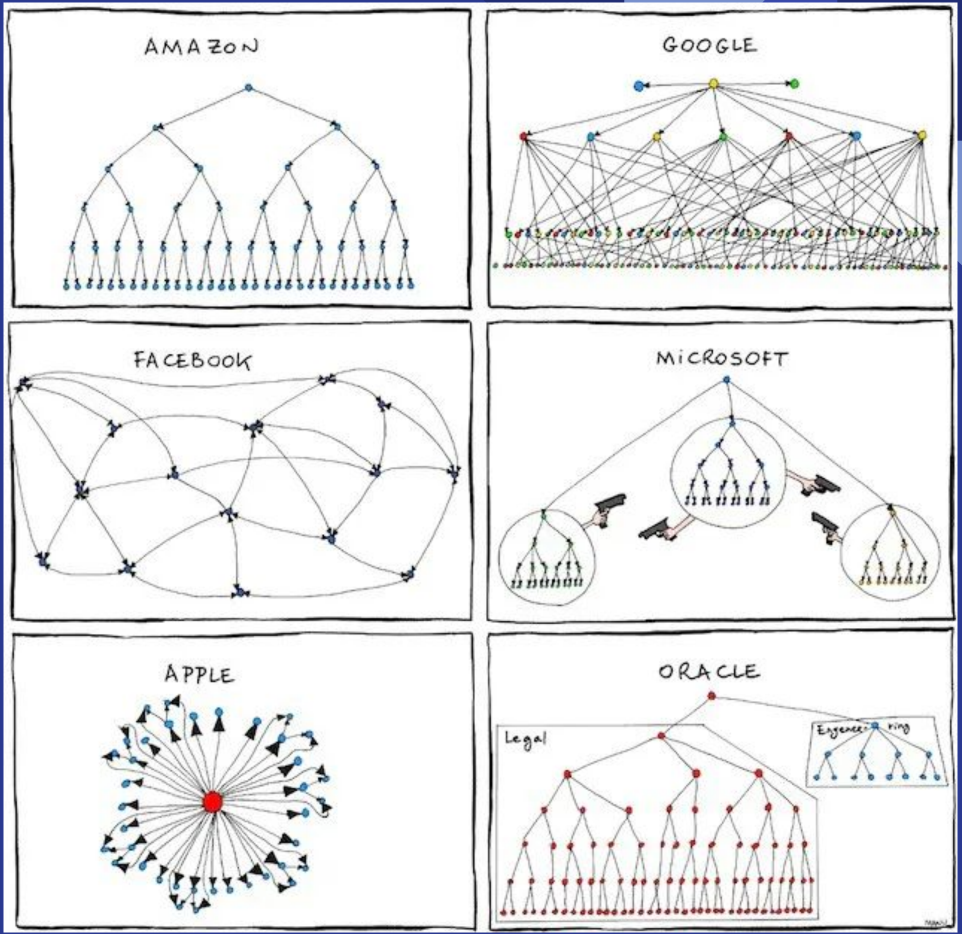
CS 9

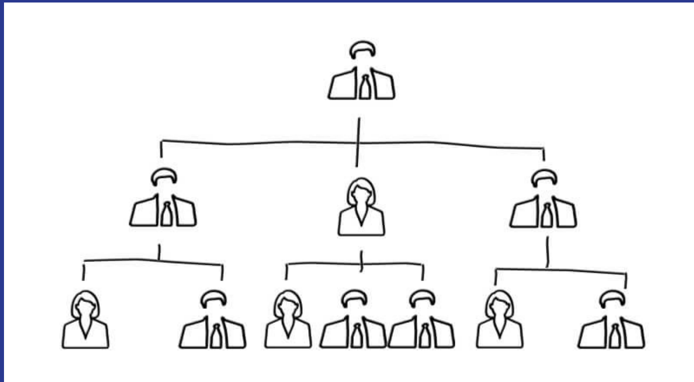
Week 5 Problems

Andrew Benson
Ian Tullis



⭐ Problem 5-1: Longest Reporting Chain

- At the company *Hooli*, the hierarchy of employees is very organized. Aside from the CEO, every employee reports to exactly one other employee. There is no limit to the number of employees that can report to a single employee. There are no reporting cycles, and everyone eventually reports to the CEO.
- You have access to a function getReports(username) that returns a list of usernames of employees that report to the given employee. (If there are none, the list is empty.)
- Given the username of the CEO, determine the length of the longest reporting chain. (Ex: 3 for right example)
    - Extension 1: determine the username of the employee with this reporting chain.
    - Extension 2: Suppose instead of a reporting chain (which goes strictly up in this reporting structure), we care about a chain between any two employees. Determine the length of the longest such chain (Ex: 5 for right example) (Link to Leetcode for Ext 2)



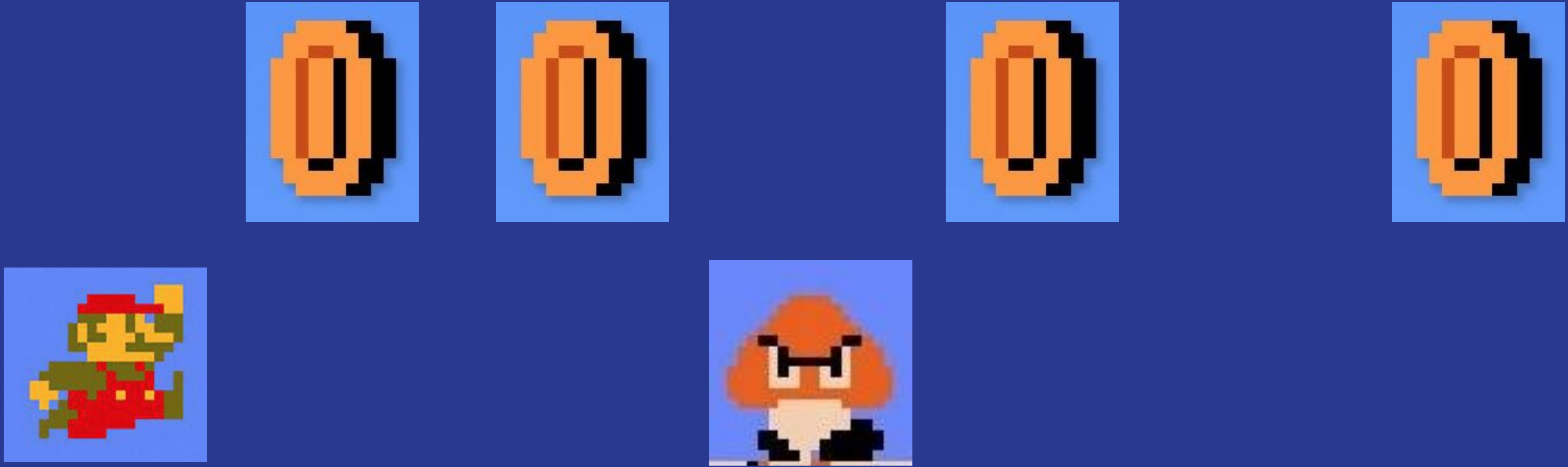
⭐ Problem 5-1: Longest Reporting Chain

- Hints
  - How would you model the structure of the data you're given?
  - Do you see any recursive structure? This doesn't mean you *have* to use recursion, but you could still exploit it.
  - Extension 1: Start with your solution for just the length, and think about how you can augment your return value with the username.
  - Extension 2: The two employees must share a common manager at some point. If you try each employee, one of them must be the "common manager" to the longest chain.

⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

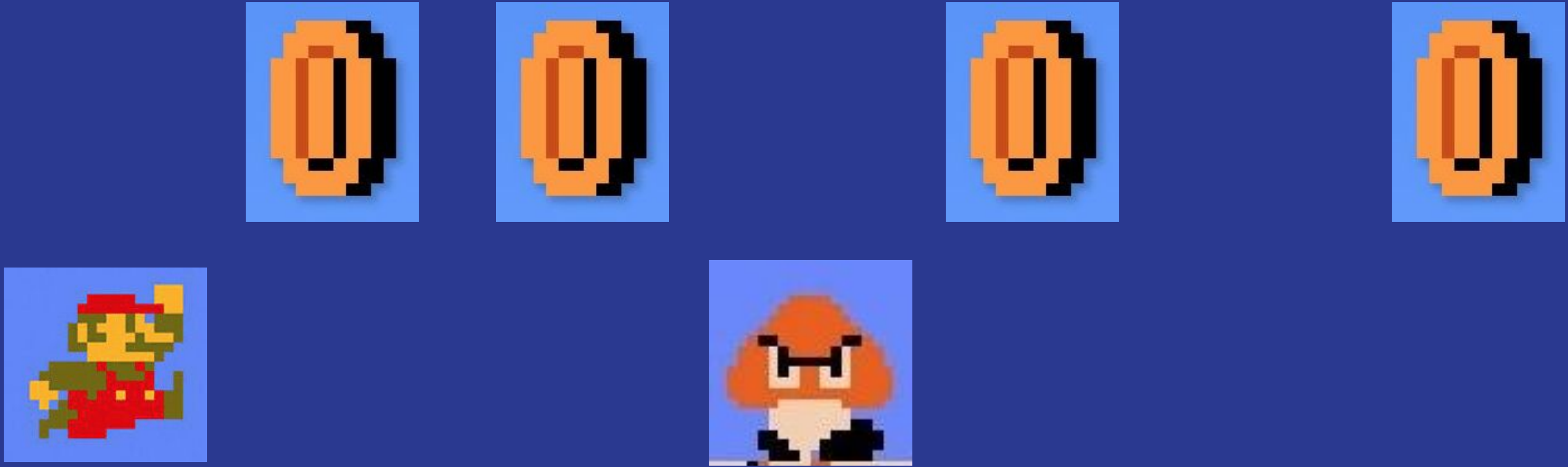
Mario's extremely basic adventure

(probably like 50 bucks on Switch)



⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

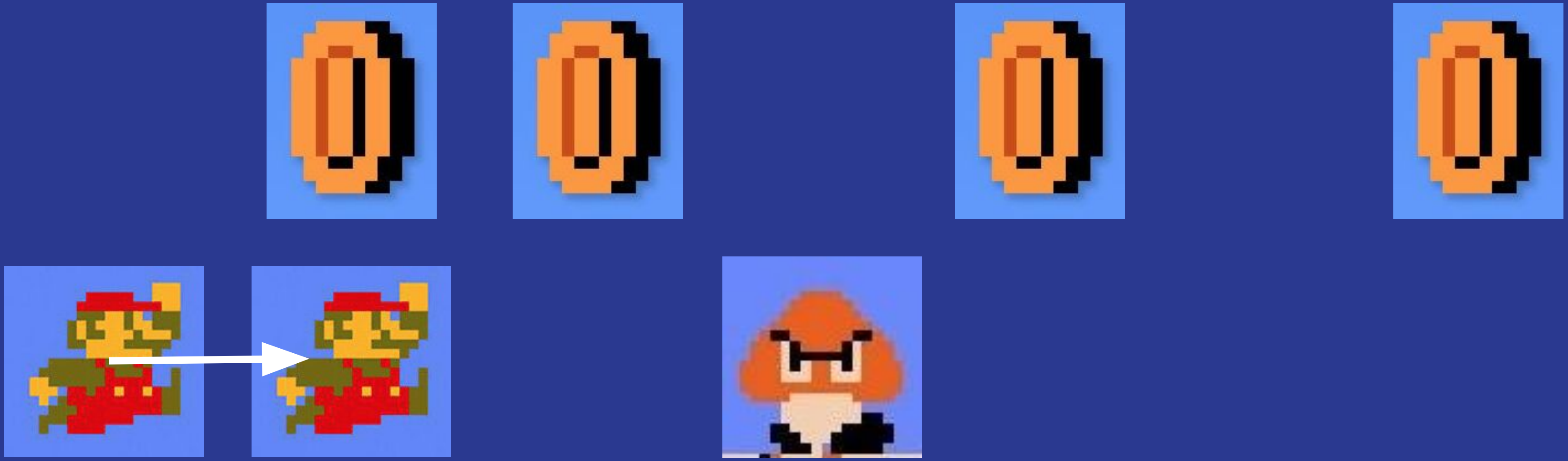
It's only two rows high. Coins only appear in the top row. Enemies only appear in the bottom row. Mario starts in the bottom left.



⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

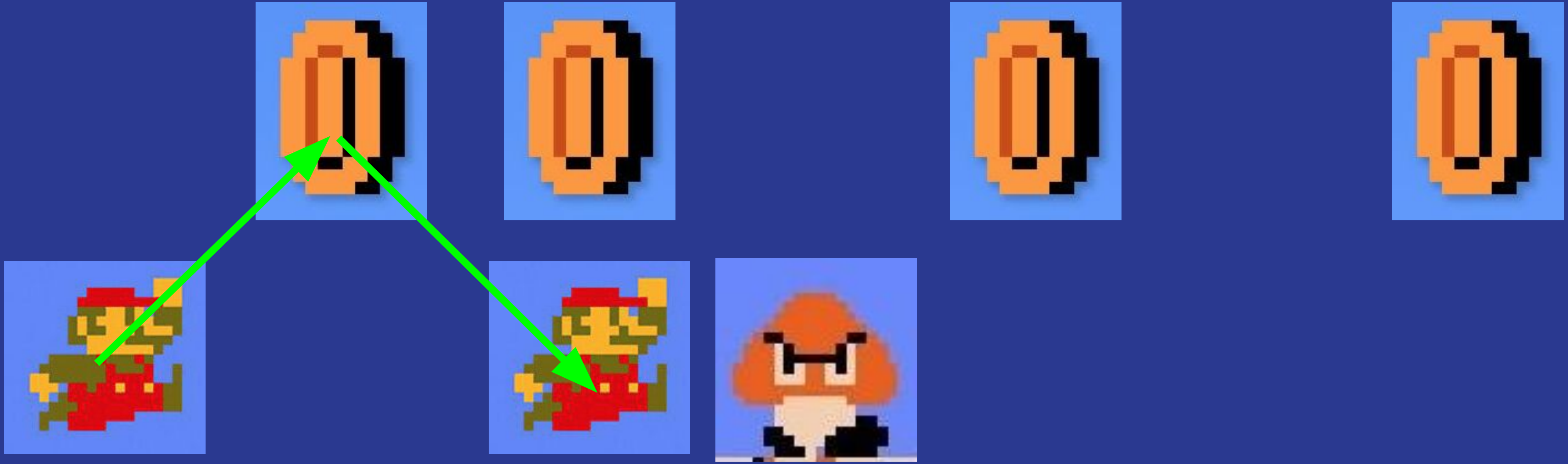
In this game, Mario has two kinds of move

Option 1: Go forward one step



⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

Option 2: Jump



⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

coins are good

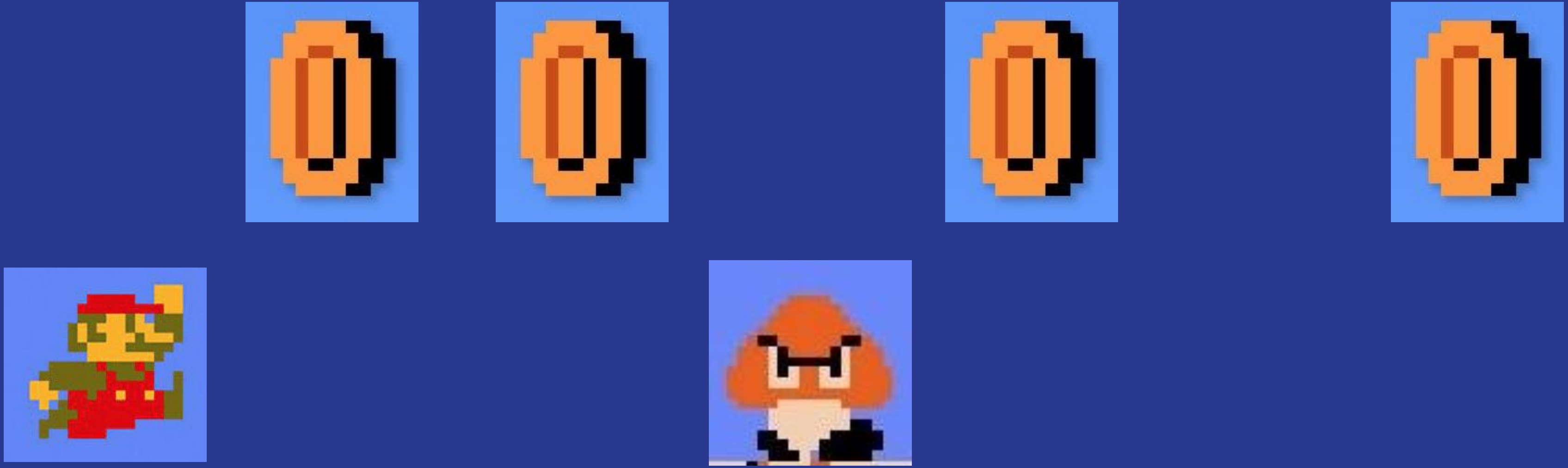
you want as many as possible

because Mario's life is empty

⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

enemies do not move

(they've been doing this for 35+ years, the excitement isn't there anymore)



⭐⭐ Problem 5-2: Not-Very-Super Mario Bros.

not OK to walk into enemies

how did you get hit, it was just standing there

⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

OK to land on enemies

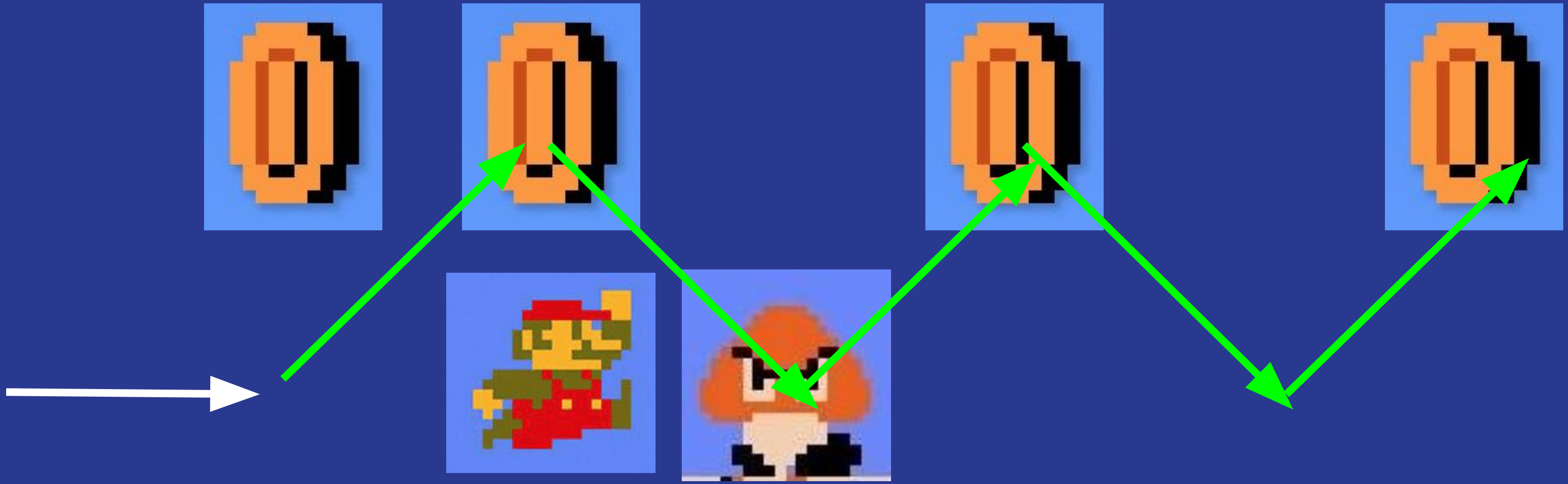
because Mario is an asshole

⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

You're given a list of indices of coins and a list of indices of enemies. Determine the max possible number of coins Mario can get.

⭐⭐ Problem 5-2: Not-Very-Super Mario Bros

Correct answer for this example: 3 coins



⭐⭐⭐ Problem 5-3: Electoral Tie

(This was Ian's actual interview question!)

In US presidential elections, each state is worth a certain number of electoral votes. The candidate who wins a majority of the popular vote in a state wins all of that state's electoral votes (with some exceptions that we will ignore here). There are 538 electoral votes, so there could in theory be a tie.

Suppose that there are only two candidates, and that the election is very close: each state is an independent 50/50 coinflip to go to either candidate. (This independence is obviously unrealistic, but we'll gloss over that.)

Our solution will generalize beyond the USA; say there are N states ($1 <= N <= 100$), and the i-th state is worth V_i votes ($1 <= V_i <= 100$). Determine the probability of a tie.

⭐⭐⭐ Problem 5-3: Electoral Tie

Example: 2 states, each worth 1.

The probability of a tie is ½. Whoever the winner of the first state is, there is a fifty-fifty chance that that same candidate wins the second state.

Example: 3 states, worth 3, 4, and 1.
Call them Balifornia, Boregon, and Bashington.

The probability of a tie is ¼. This happens if Candidate A wins Balifornia and Bashington and Candidate B wins Boregon, or the same happens but with the two candidates swapped.

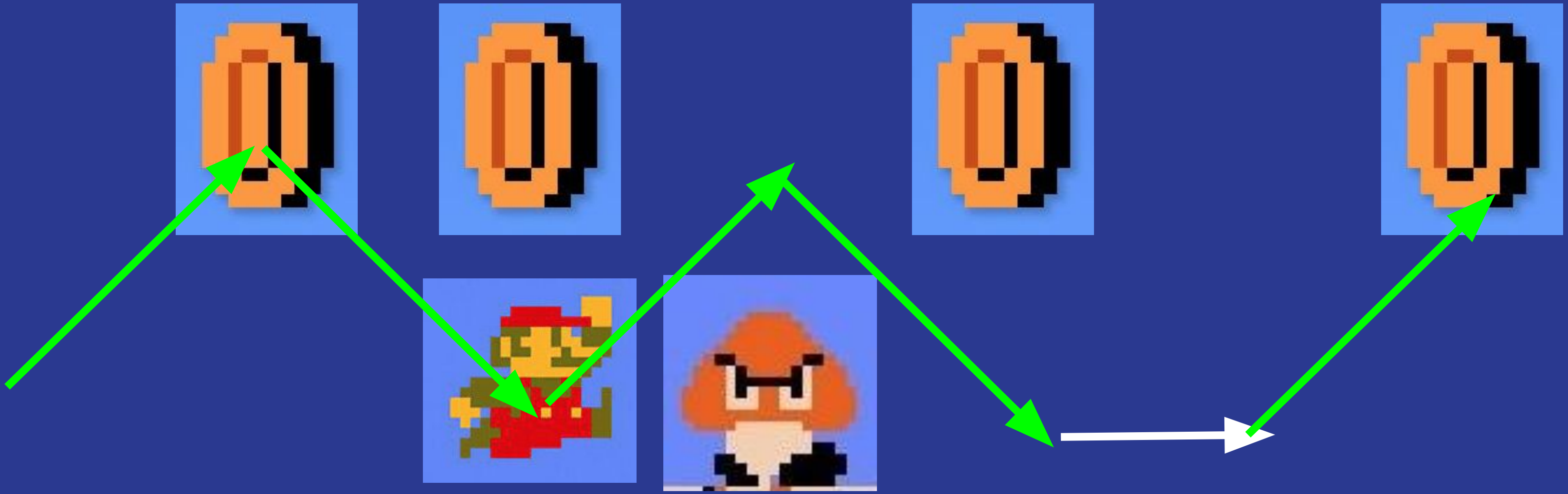
Example: 7 states, worth 1, 2, 4, 8, 16, 32, 64.

The probability of a tie is 0. Whoever wins the 64-vote state wins overall.

Solutions to 5-2 (discussed in-class)

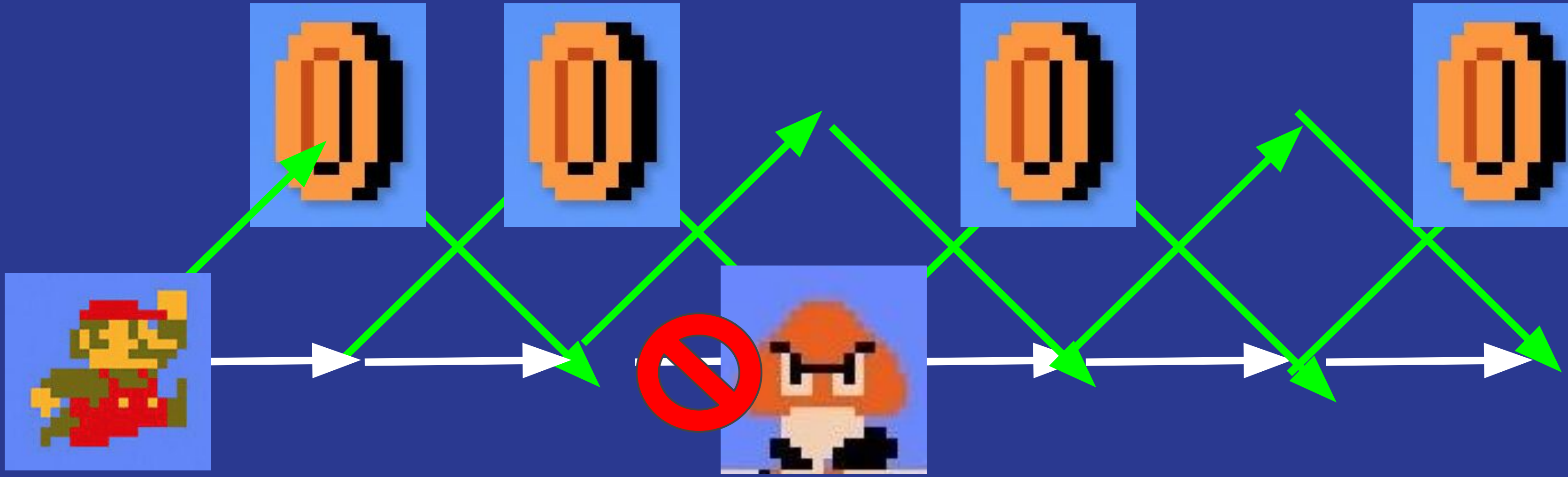
Greedy strategies aren't always optimal

2 coins



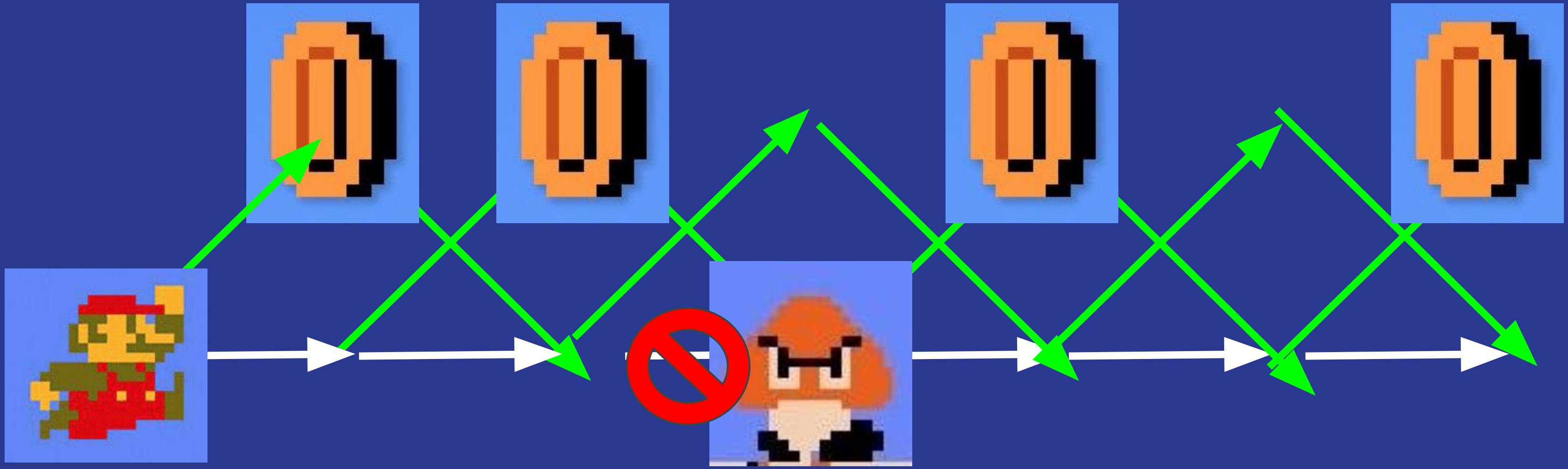
Why not just try every path?

Exponential number…



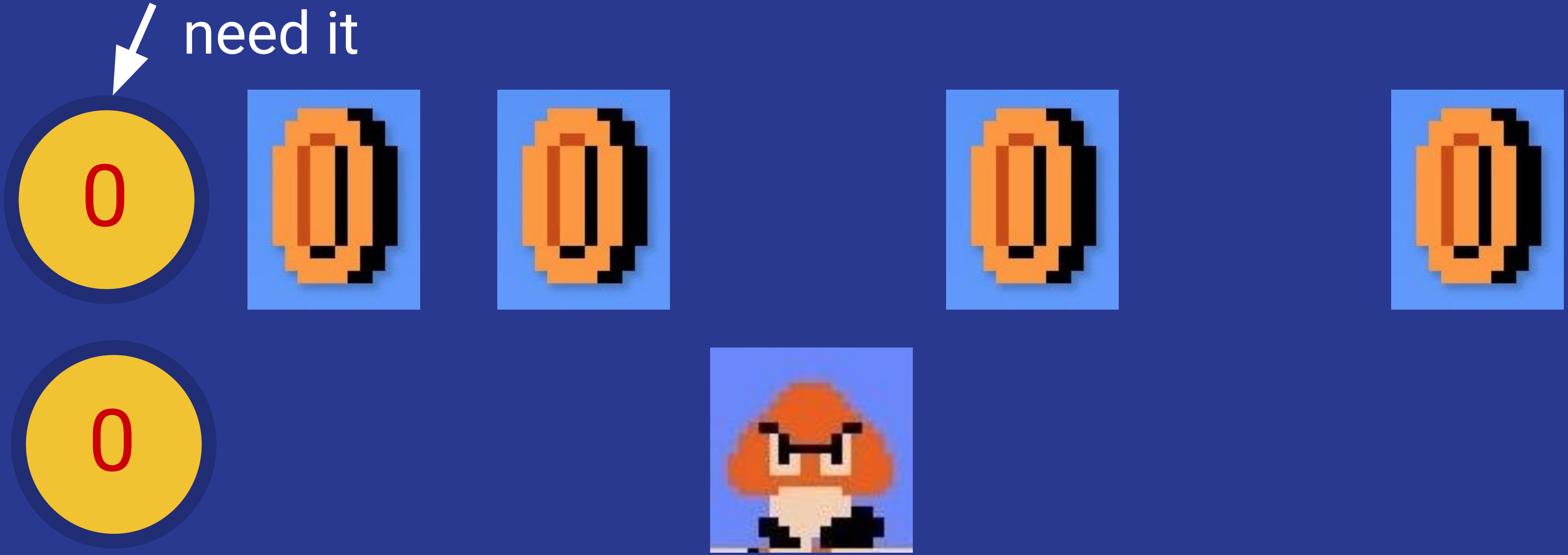
Why not just try every path?

Exponential number… <u>so any solution that explicitly considers them all is exponential</u>

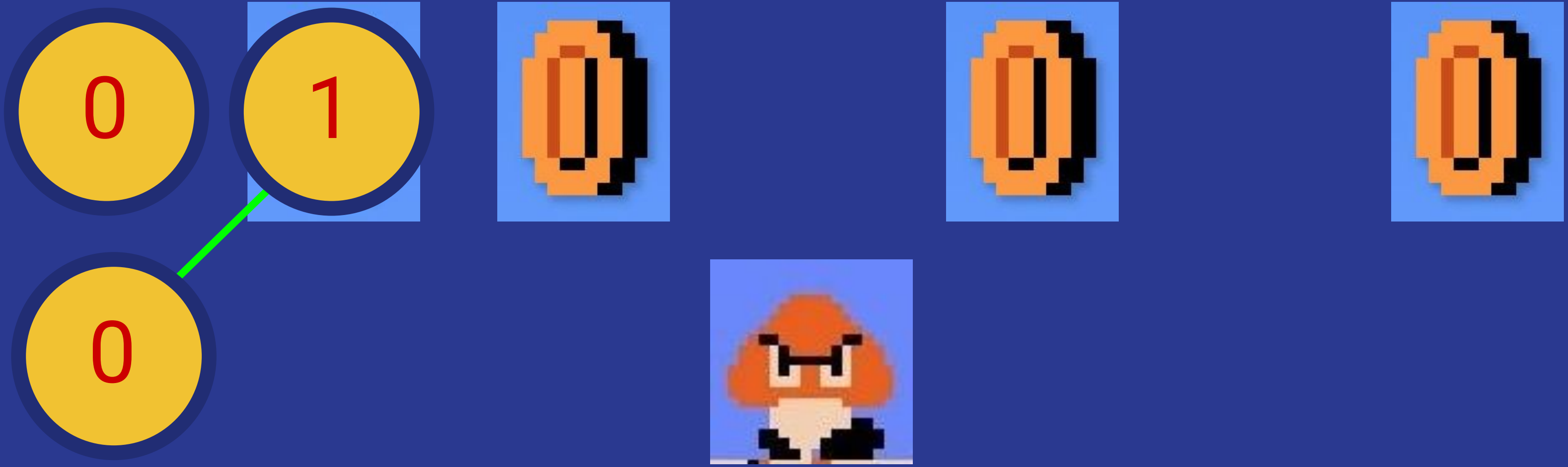


Solving via Dynamic Programming

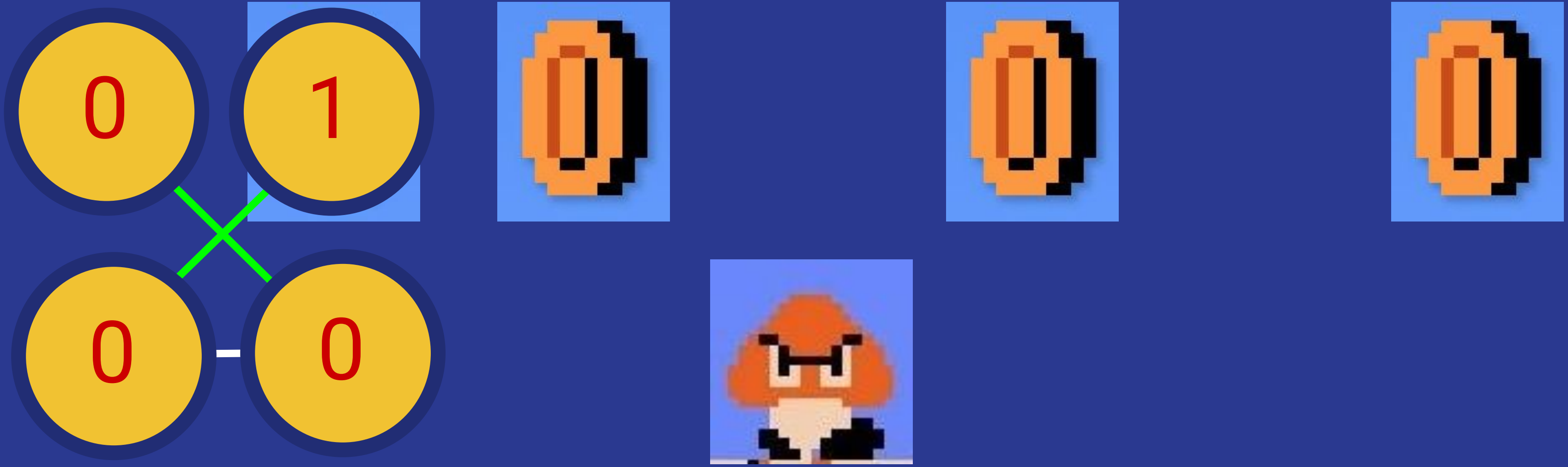
what? we can't get here... but you'll see why we need it



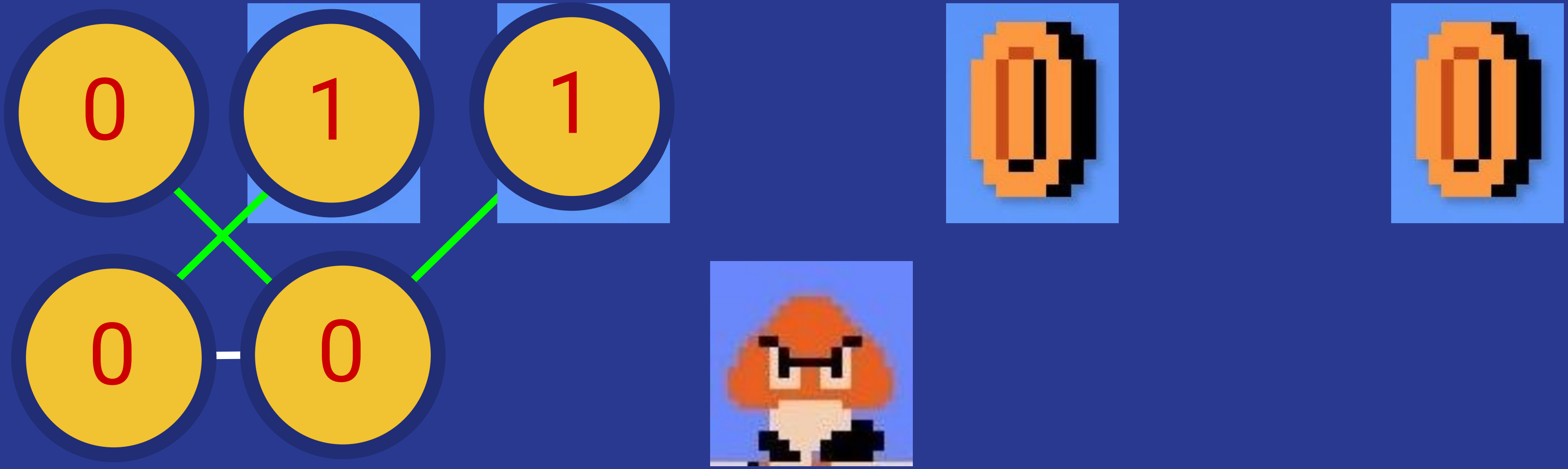
top row cell:
value from downleft,
plus 1 if coin



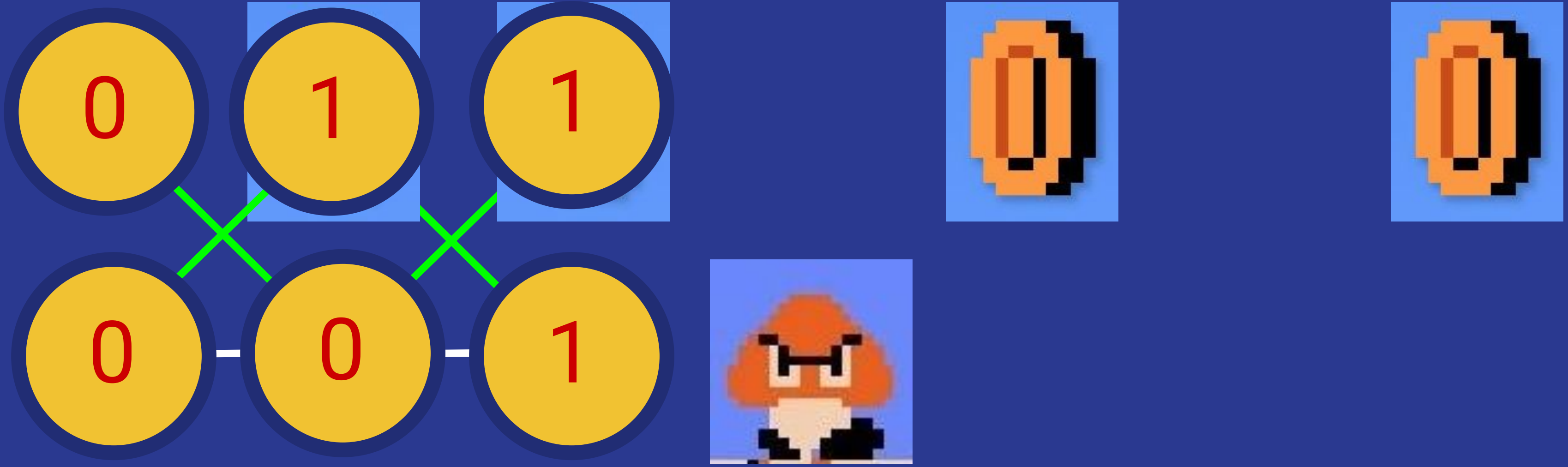
bottom row cell:
max of:
1. value from upleft
2. value from left if
no enemy here



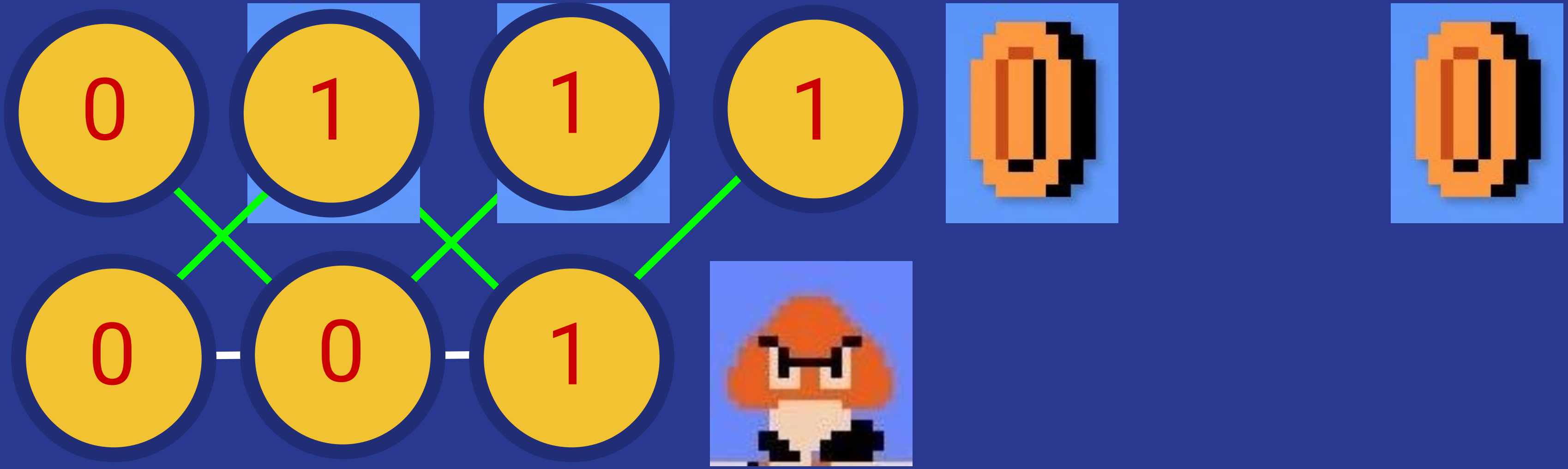
top row cell:
value from downleft,
plus 1 if coin



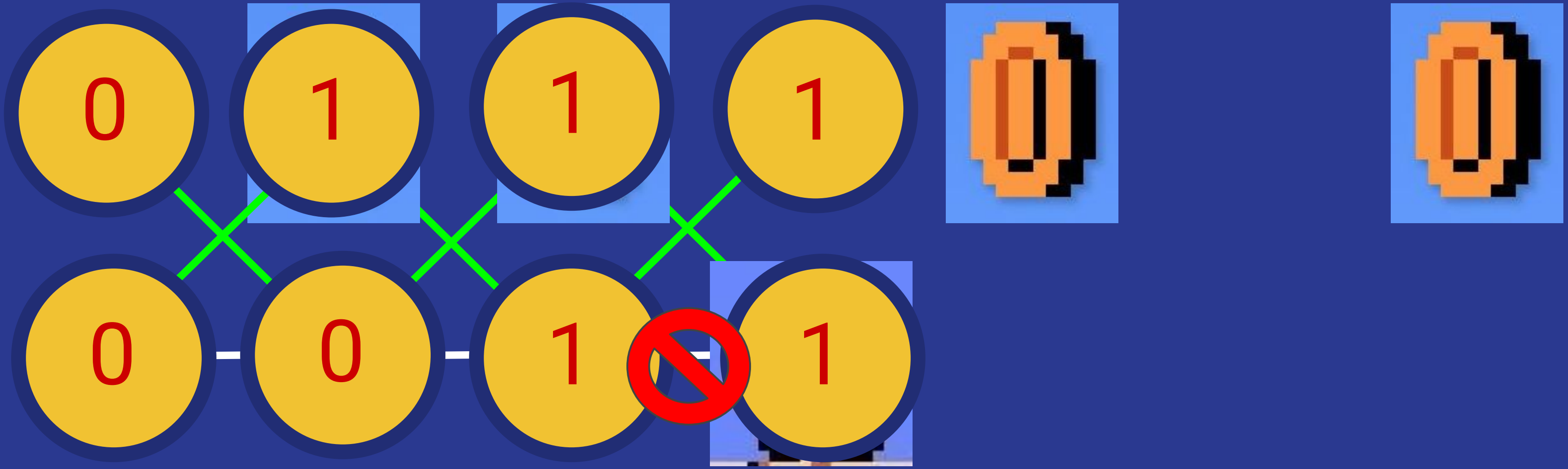
bottom row cell:
max of:
1. value from upleft
2. value from left if
no enemy here



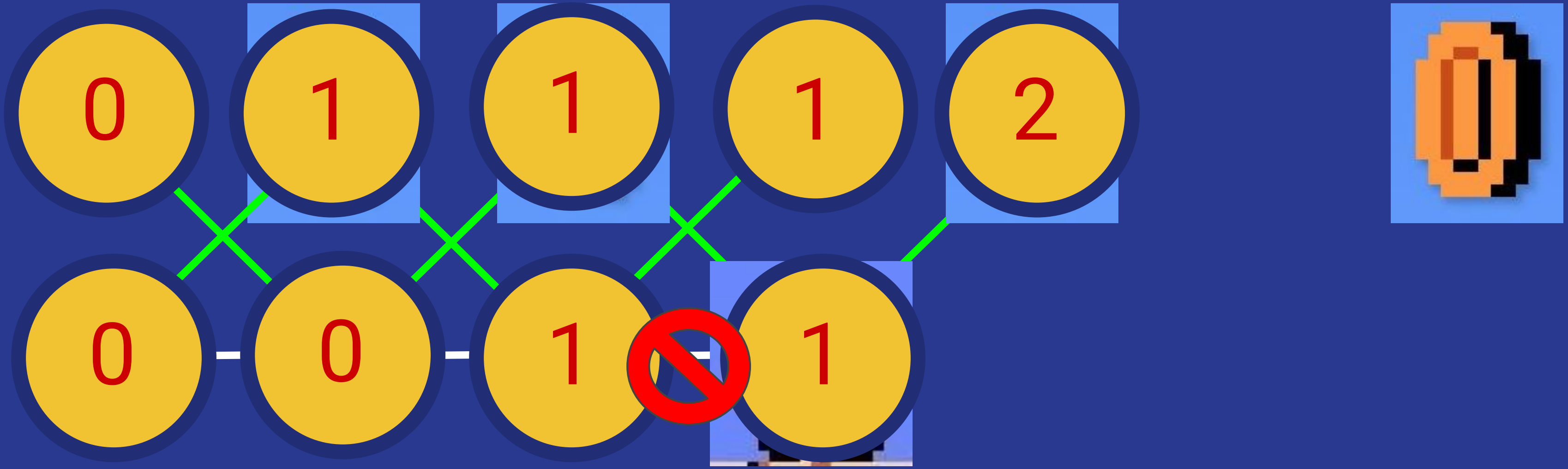
top row cell:
value from downleft,
plus 1 if coin



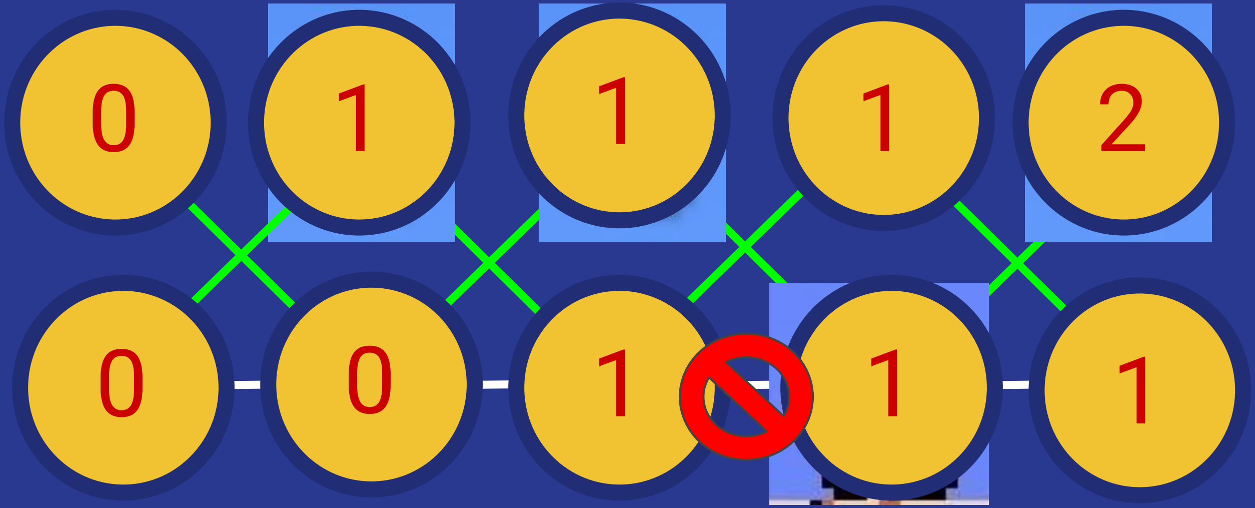
bottom row cell:
max of:
1. value from upleft
2. value from left if
no enemy here



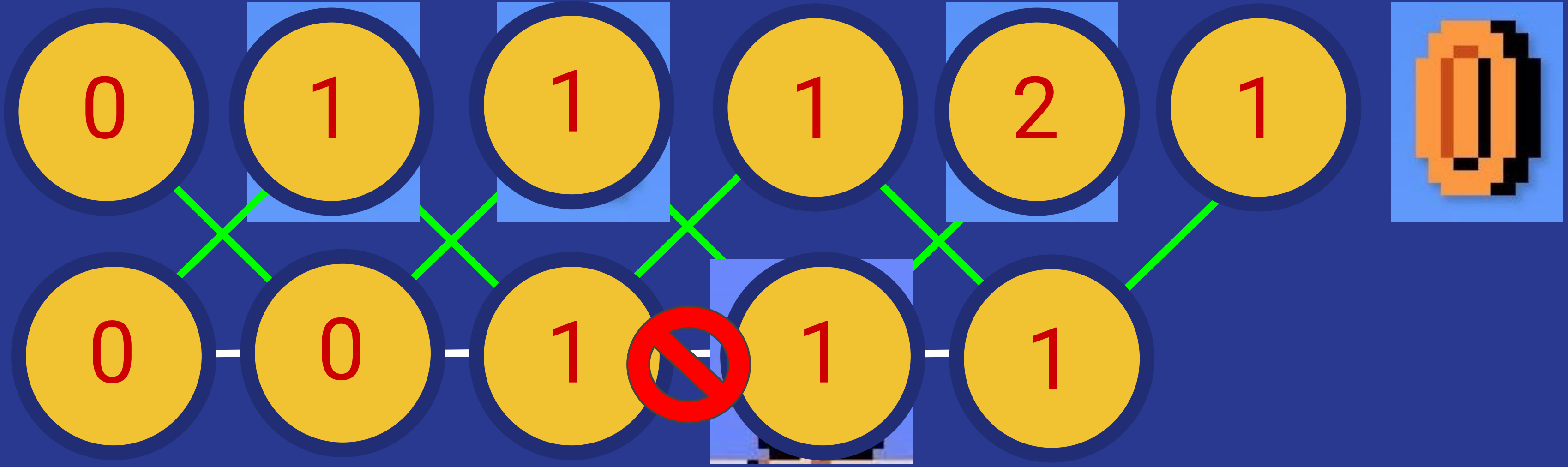
top row cell:
value from downleft,
plus 1 if coin



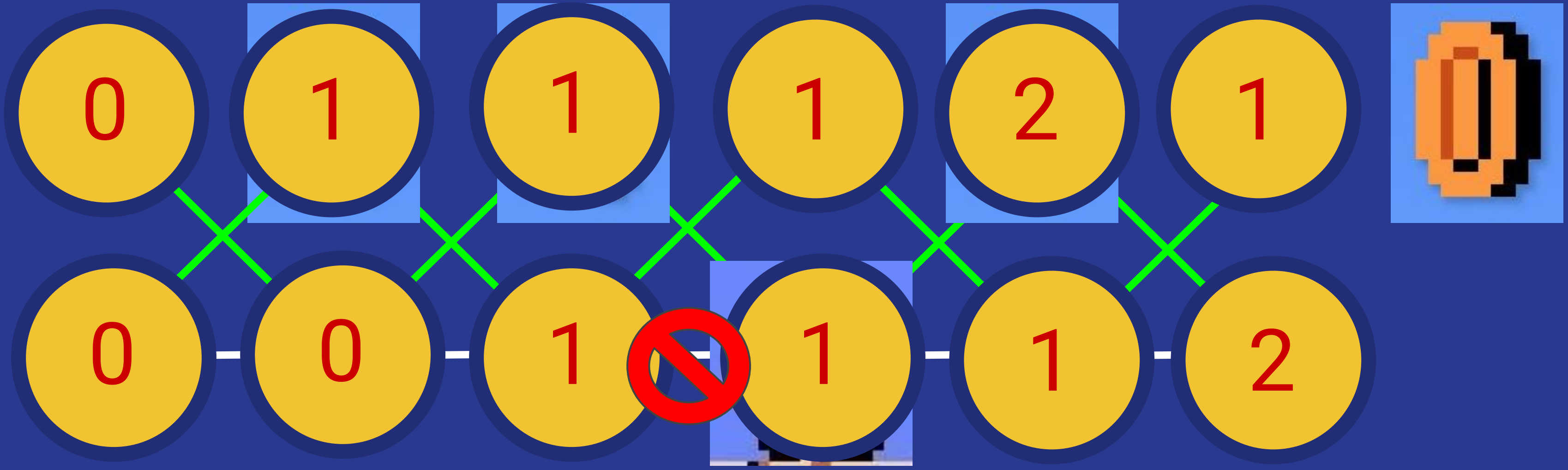
bottom row cell:
max of:
1. value from upleft
2. value from left if no enemy here



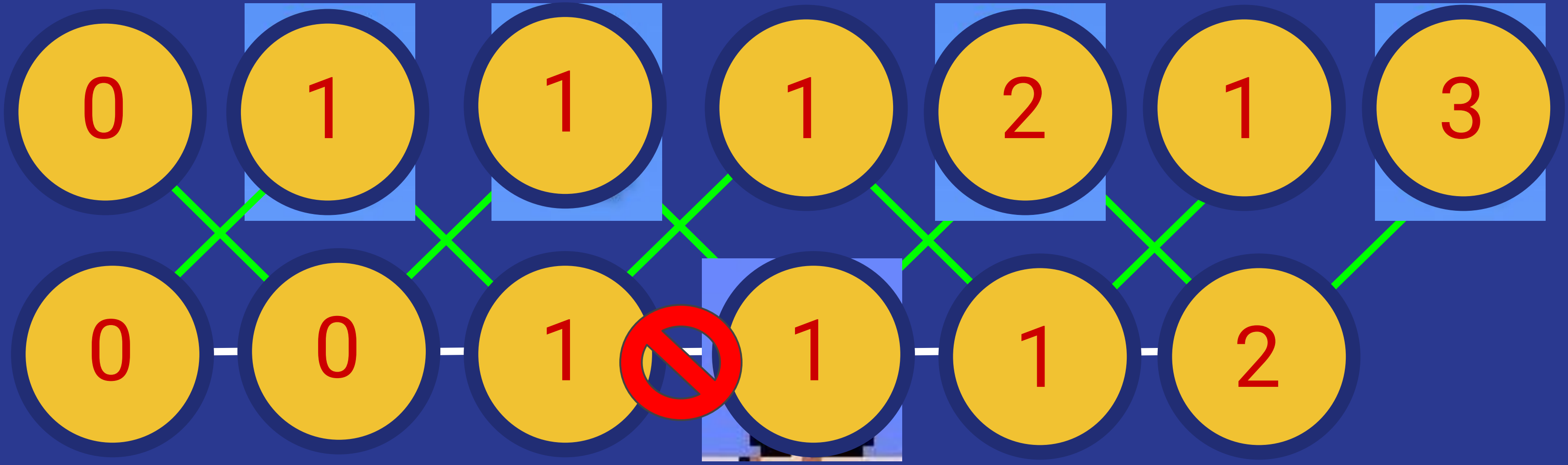
top row cell:
value from downleft,
plus 1 if coin



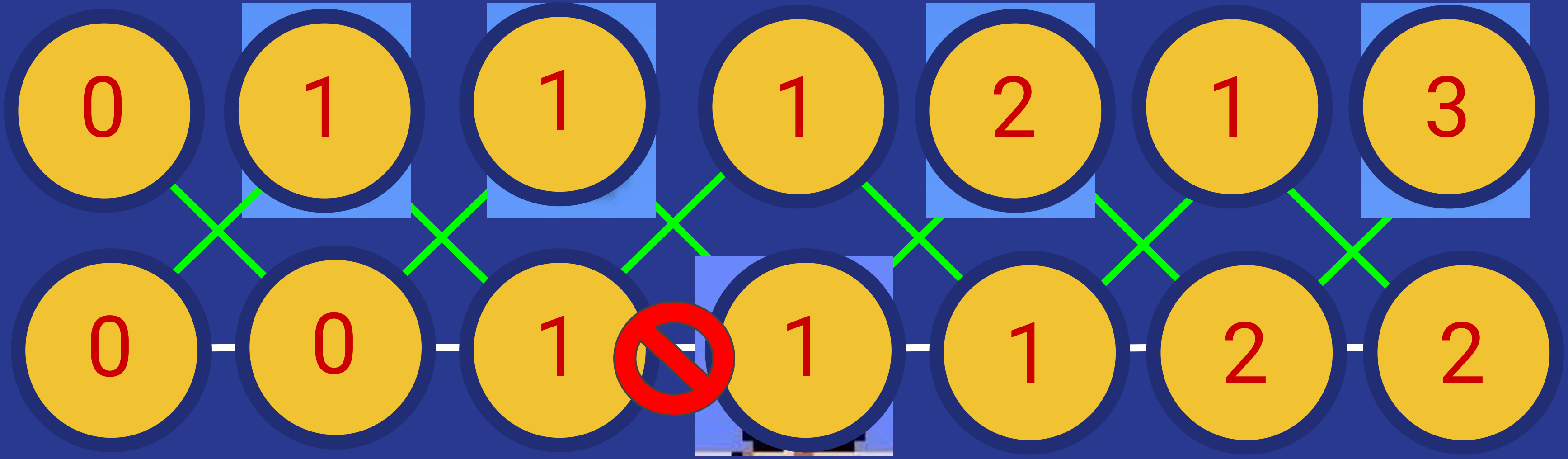
bottom row cell:
max of:
1. value from upleft
2. value from left if no enemy here



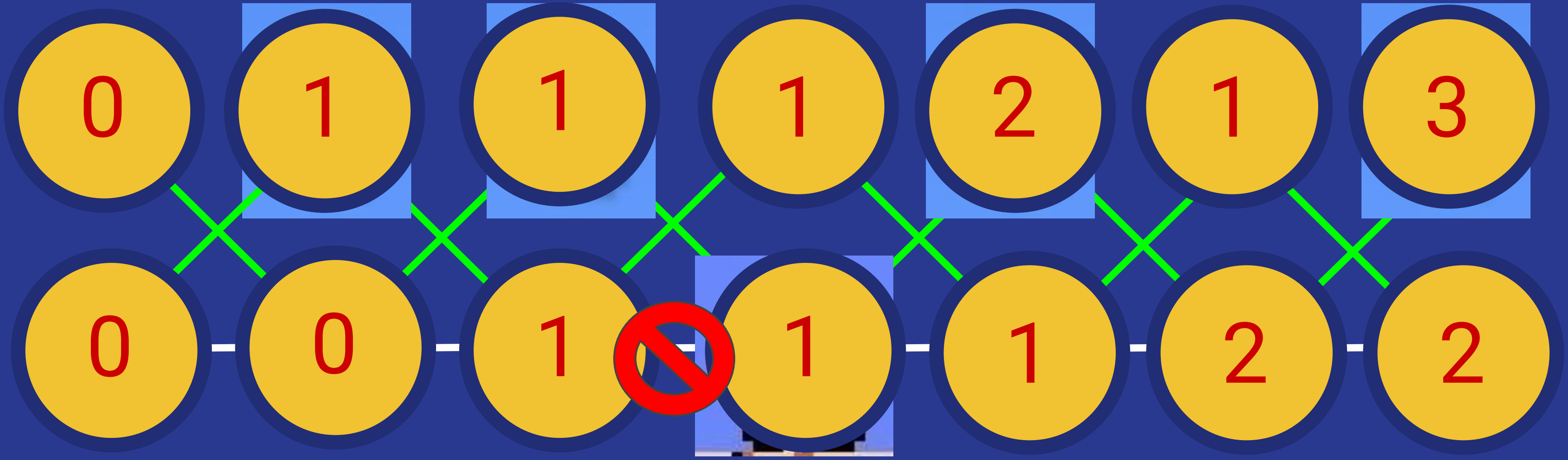
top row cell:
value from downleft,
plus 1 if coin



bottom row cell:
max of:
1. value from upleft
2. value from left if no enemy here



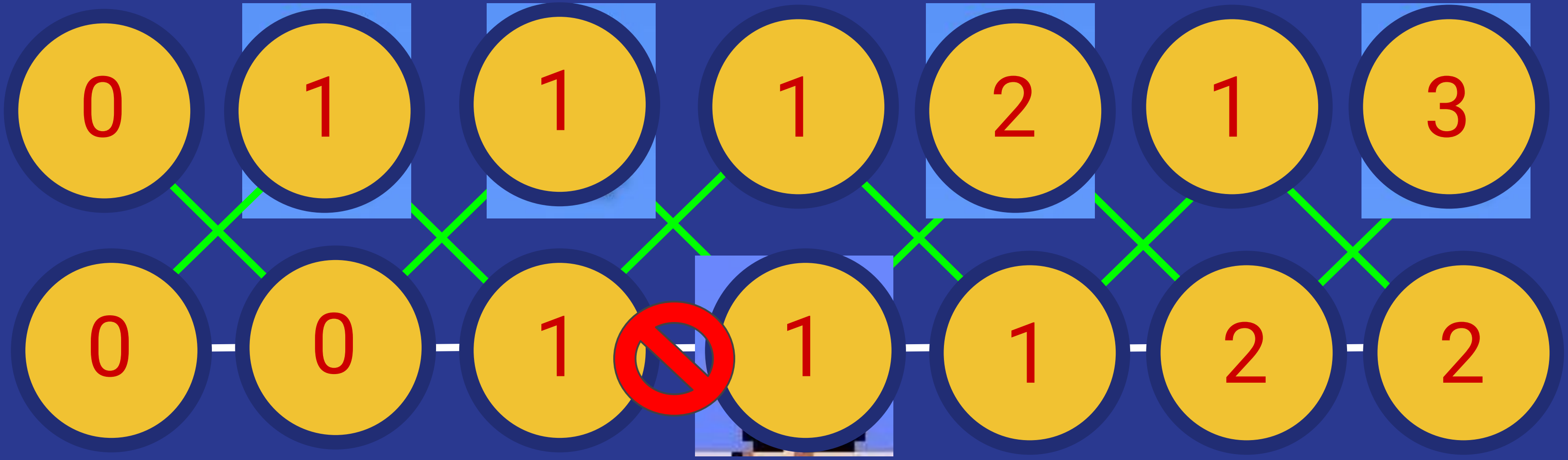
Wait a minute...
Isn't this just the "exponential" slide again?



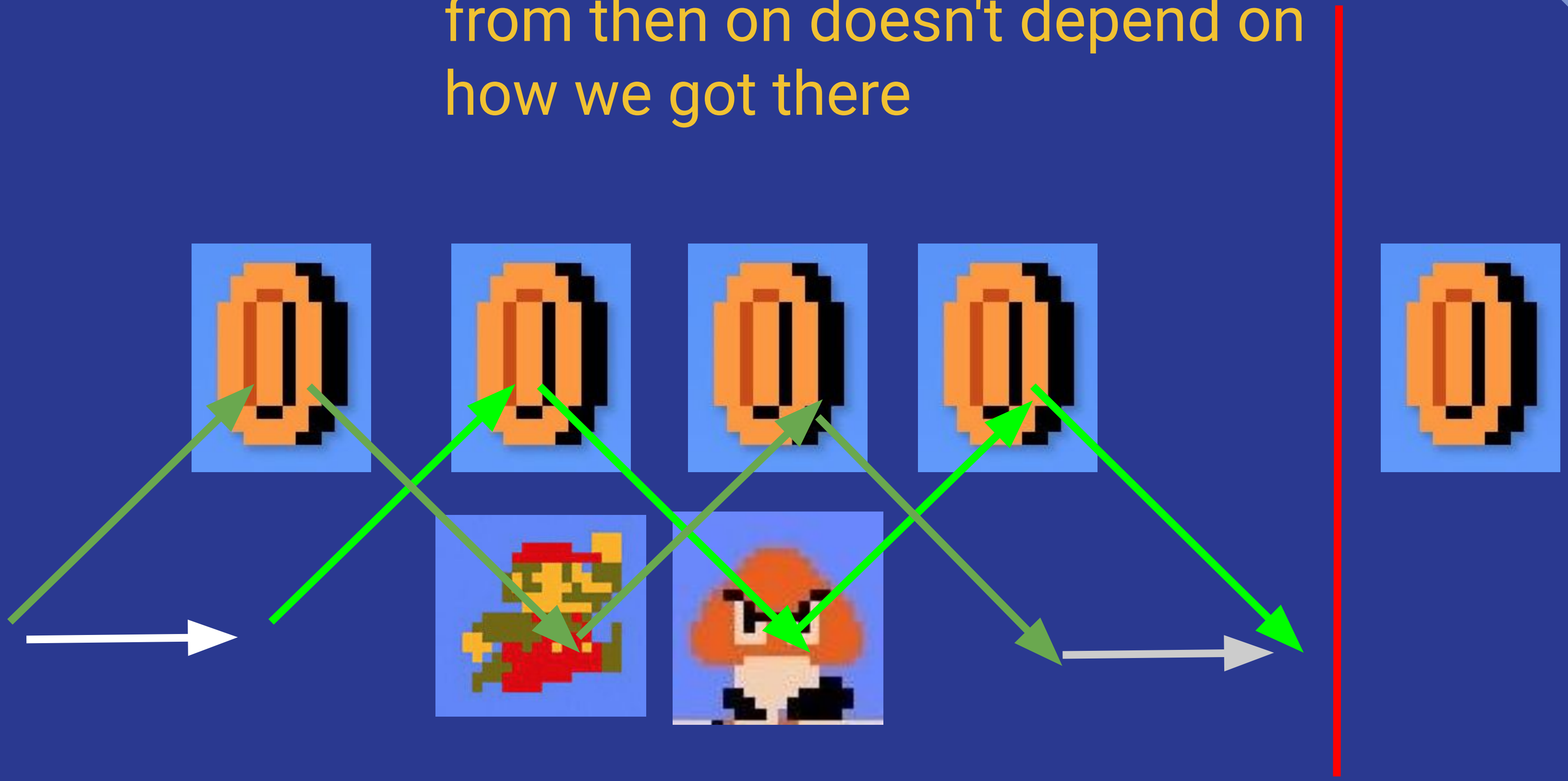
Wait a minute...

Isn't this just the "exponential" slide again?

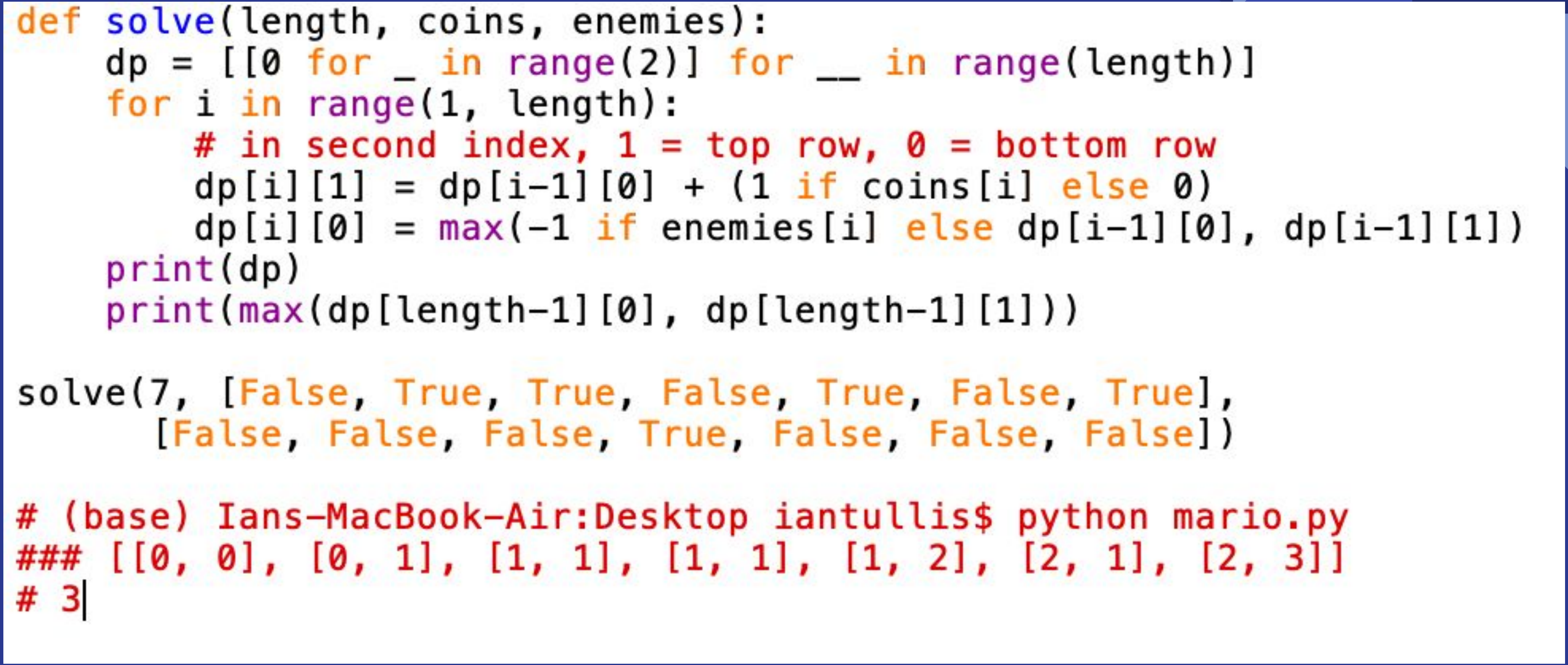
No! We took <u>linear time</u>.



once we get this far, the strategy from then on doesn't depend on how we got there



Code!

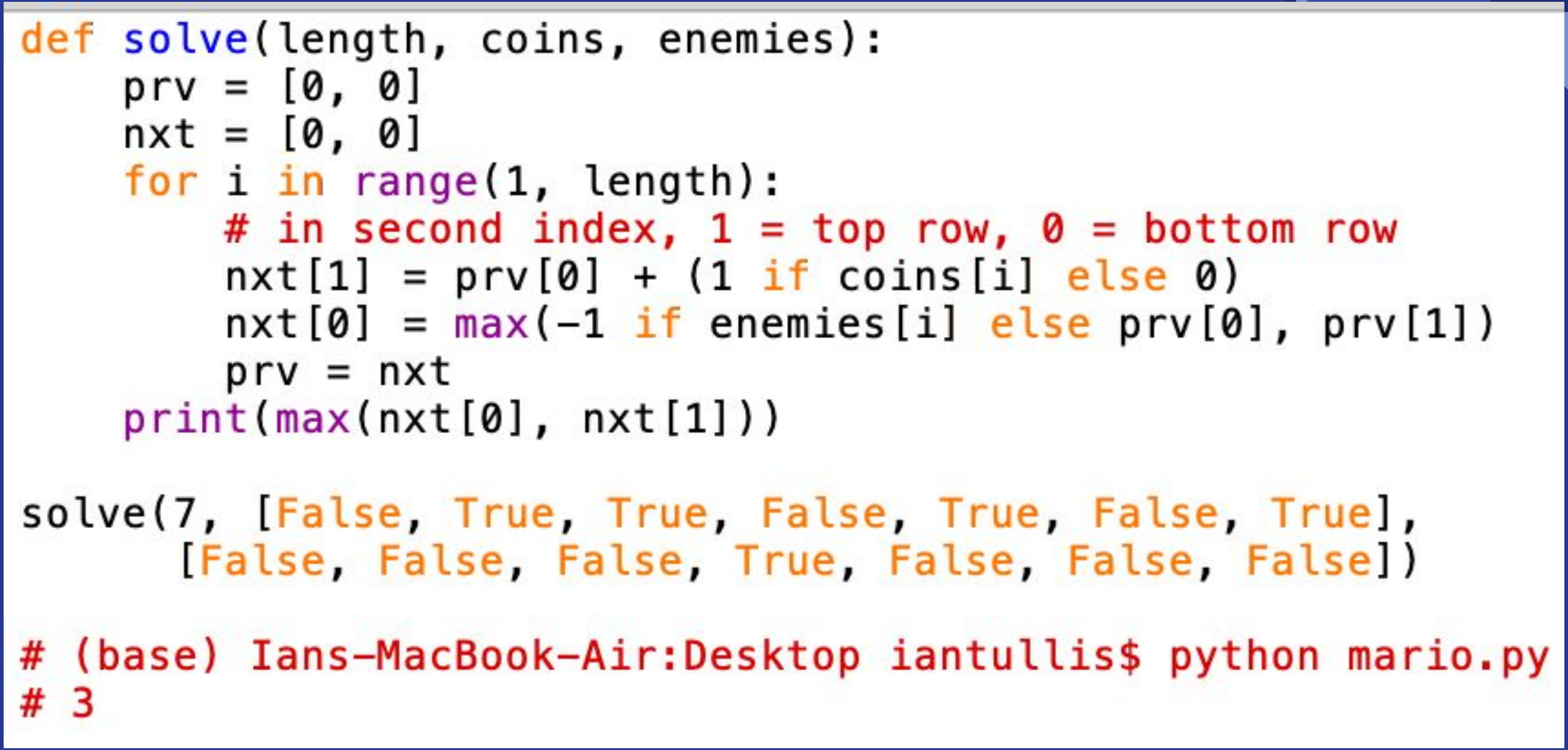


```python
def solve(length, coins, enemies):
    dp = [[0 for _ in range(2)] for __ in range(length)]
    for i in range(1, length):
        # in second index, 1 = top row, 0 = bottom row
        dp[i][1] = dp[i-1][0] + (1 if coins[i] else 0)
        dp[i][0] = max(-1 if enemies[i] else dp[i-1][0], dp[i-1][1])
    print(dp)
    print(max(dp[length-1][0], dp[length-1][1]))

solve(7, [False, True, True, False, True, False, True],
      [False, False, False, True, False, False, False])

# (base) Ians-MacBook-Air:Desktop iantullis$ python mario.py
### [[0, 0], [0, 1], [1, 1], [1, 1], [1, 2], [2, 1], [2, 3]]
# 3
```

More space-efficient code!



```python
def solve(length, coins, enemies):
    prv = [0, 0]
    nxt = [0, 0]
    for i in range(1, length):
        # in second index, 1 = top row, 0 = bottom row
        nxt[1] = prv[0] + (1 if coins[i] else 0)
        nxt[0] = max(-1 if enemies[i] else prv[0], prv[1])
        prv = nxt
    print(max(nxt[0], nxt[1]))

solve(7, [False, True, True, False, True, False, True],
      [False, False, False, True, False, False, False])

# (base) Ians-MacBook-Air:Desktop iantullis$ python mario.py
# 3
```

Don't let Ian forget to say the password

(and the solution to 5-3 is on the next slides!)

Solution to 5-3 (spoiler space)

Electoral Tie: Solution

- We can pick off some special cases:
  - If the total number of votes is odd, then the tie probability is 0.
  - If there is one state that is worth more than the total number of votes from the other states combined, then the tie probability is 0.
- But these only go so far – it'd be better to find a general solution.

- The brute force approach is to explicitly consider each of the 2^n ways the election could go. (Each state is a coinflip, so the results of all n states are essentially given by a binary string of length n.)
  - Unfortunately, this is exponential. Actually a bit worse, $O(n2^n)$, since processing each possibility takes linear time.

- It's natural to try some kind of backtracking approach
  - but this would also end up being exponential…
- We need a way to avoid doing so much redundant work! Notice that, for instance, we consider
  - the first state going to Candidate A, then all the stuff that could happen with the other states
  - the first state going to Candidate B, then all the **same** stuff that could happen with the other states
- When there's redundant work, think dynamic programming/memoization!

- But even if you see this as a DP problem, it's hard to set it up…

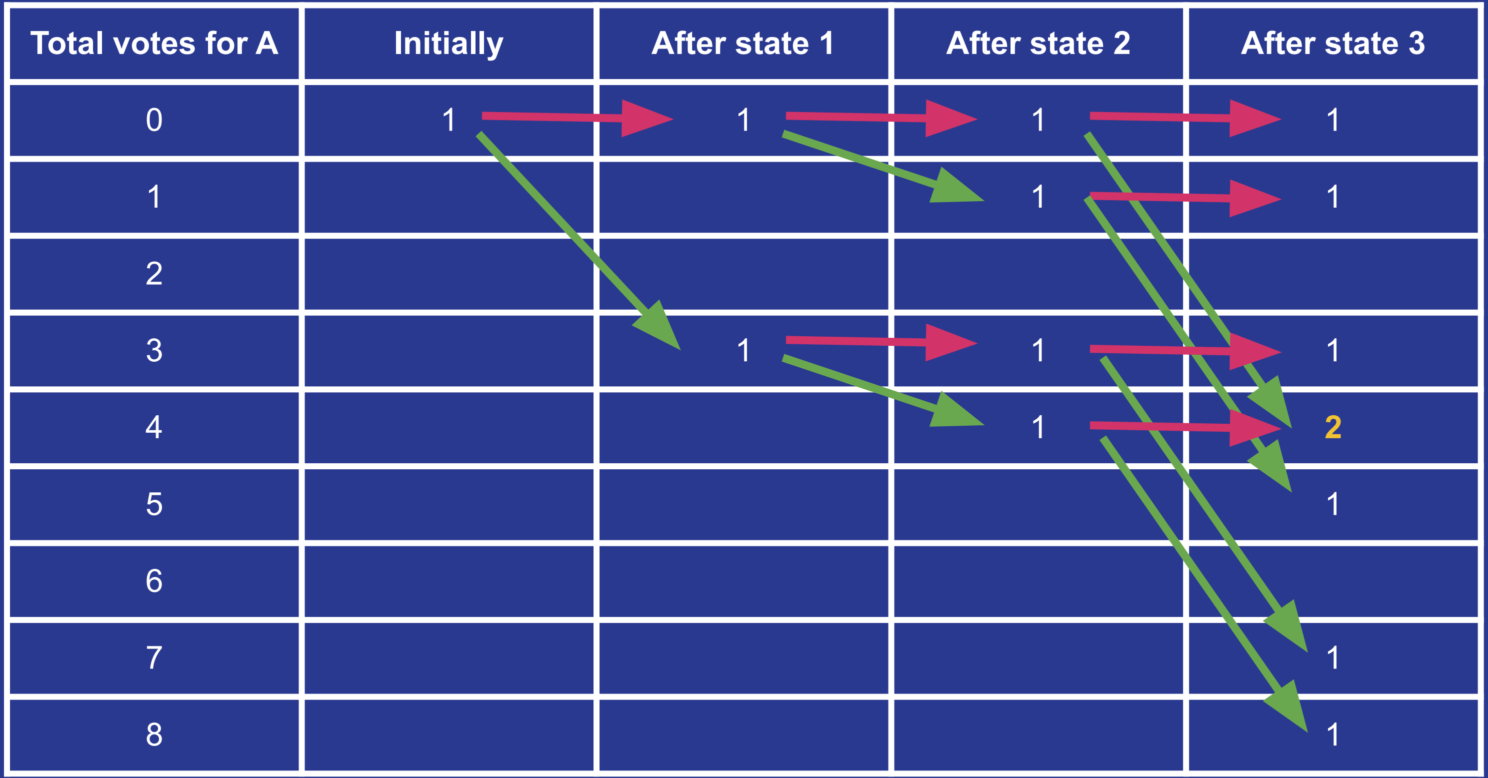
- We'll arbitrarily pretend we're Candidate A.
- We'll go through the states in some arbitrary order
    - as if the results are coming in one state at a time
- We'll maintain a table in which:
    - the columns are the individual states, plus an initial column to represent the situation before any results have come in
    - the rows are the possible totals of votes we (Candidate A) could have so far
    - each cell represents the number of ways we could have that many total votes (row) after results from that state (column) come in

- Example: state values 3, 1, 4

Total votes for A	Initially	After state 1	After state 2	After state 3
0	1	1	1	1
1		don't win this state	1	1
2				
3	win this state	1	1	1
4			1	2
5				1
6				
7				1
8				1

- Example: state values 3, 1, 4

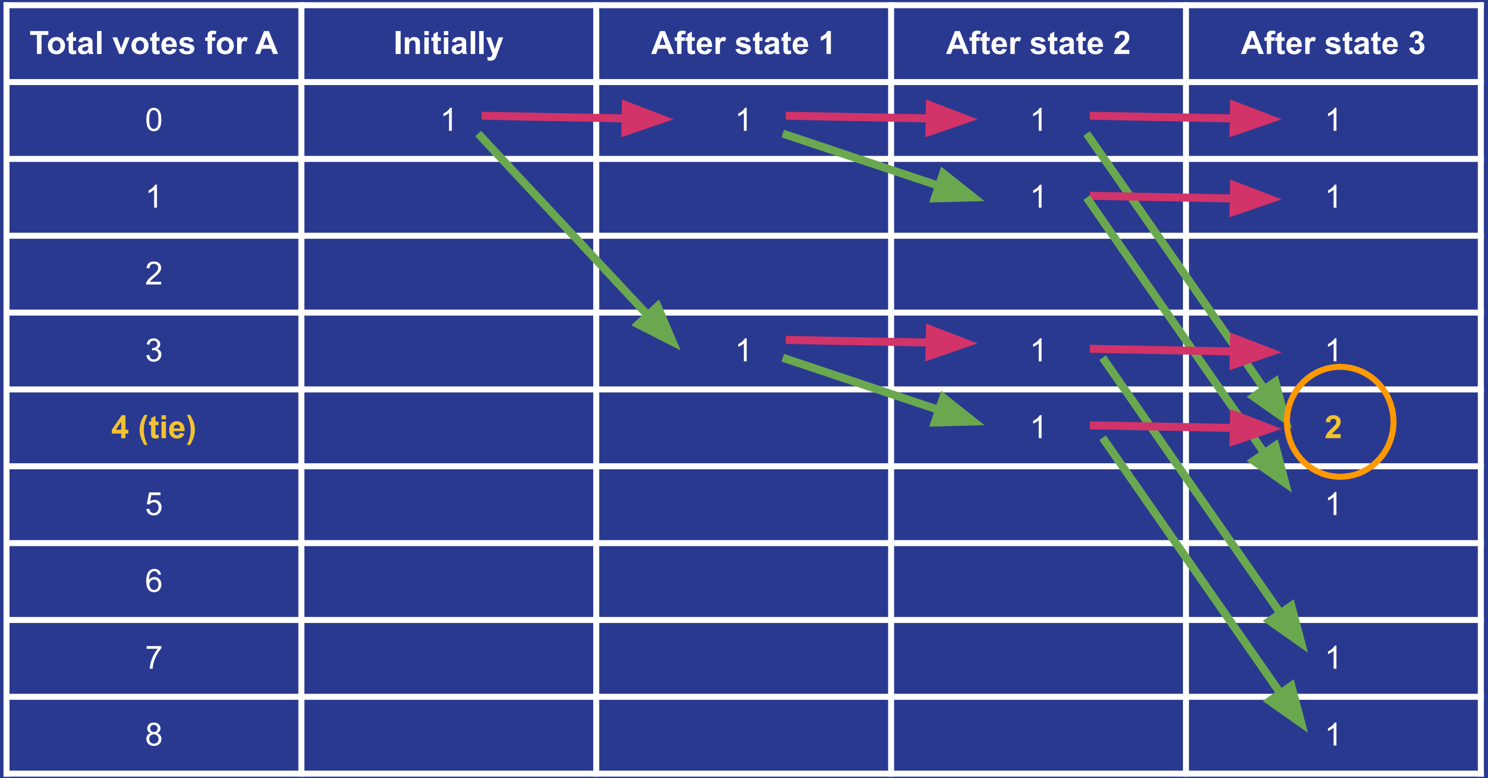
Total votes for A	Initially	After state 1	After state 2	After state 3
0	1	1	1	1
1			1	1
2				
3		1	1	1
4			1	2
5				1
6				
7				1
8				1



Total votes for A	Initially	After state 1	After state 2	After state 3
0	1	1	1	1
1			1	1
2				
3		1	1	1
4			1	**2**
5				1
6				
7				1
8				1

notice that the sum of the i-th column is 2^i, as it should be

answer: 2 / 8



Total votes for A	Initially	After state 1	After state 2	After state 3
0	1	1	1	1
1			1	1
2				
3		1	1	1
4 (tie)			1	2
5				1
6				
7				1
8				1

two ways to get here

- With a larger table, the savings from those "orange" cells (fed by two previous cells) really add up.
- Optimization: We only really need to look at rows up to the tie value. Once that value is exceeded, there can never be a tie on that branch.
- Another optimization: We don't need to store the whole table, since each column depends only on the previous one.
- Overall complexity:
    - $O(N * \mathrm{sum}(V_i))$ time (we have to visit every cell, constant work per cell)
    - $O(\mathrm{sum}(V_i))$ space
- Note that this only gets around the exponential issue because $\mathrm{sum}(V_i)$ is small…