Transition: DS+ Algorithms

Backtracking

Recap:

- Nov. 6: Faculty candidate talk
  ↳ special colloquium
  No class! (go to the talk)

- Dec 7+9: I'm out of town.
  ↳ sub those days, or review
  Exams when I'm back.

or assignments

Rest of term:
- Backtracking → Dyn programming
- Greedy algs
- Graph algorithms

Ch 2: Backtracking:

Maybe you saw in AI?
(Don't worry if not...)

Why we discuss:
It's really recursion (again)!

Also really a form of brute force:
try everything recursively,
& see what works.

N Queens

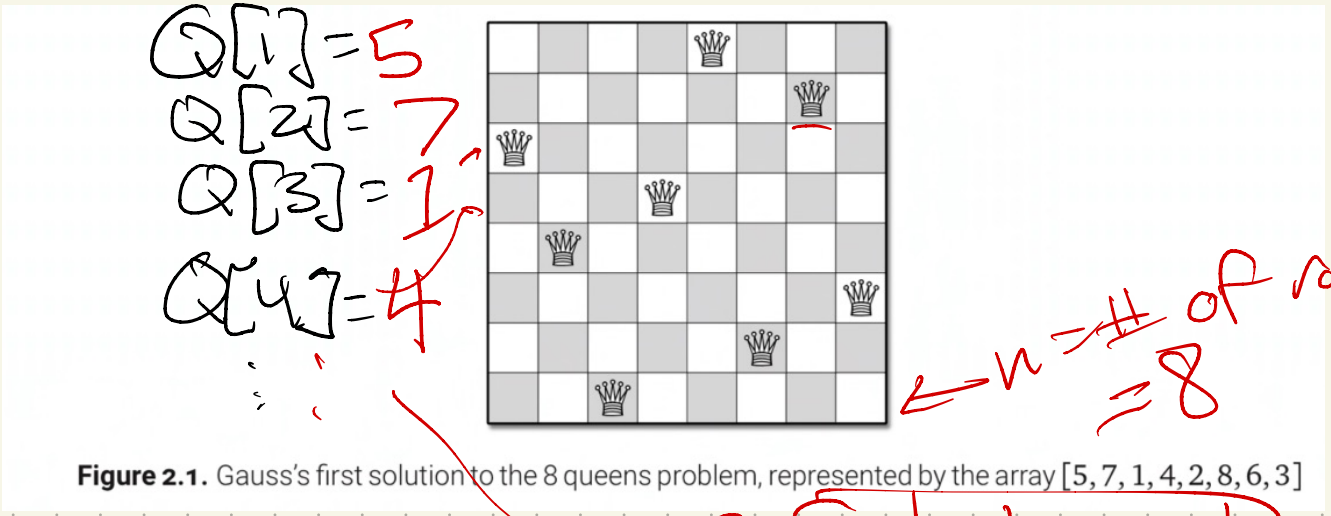


Figure 2.1. Gauss's first solution to the 8 queens problem, represented by the array [5, 7, 1, 4, 2, 8, 6, 3]

Issue: representation!

His choice: one per row, so store index of queen on rows in array, Q[1..n].

Now, how to solve:

Place a queen + keep going.

If you get stuck, "unplace" last queen + back up

The tree (b/c pretty) :

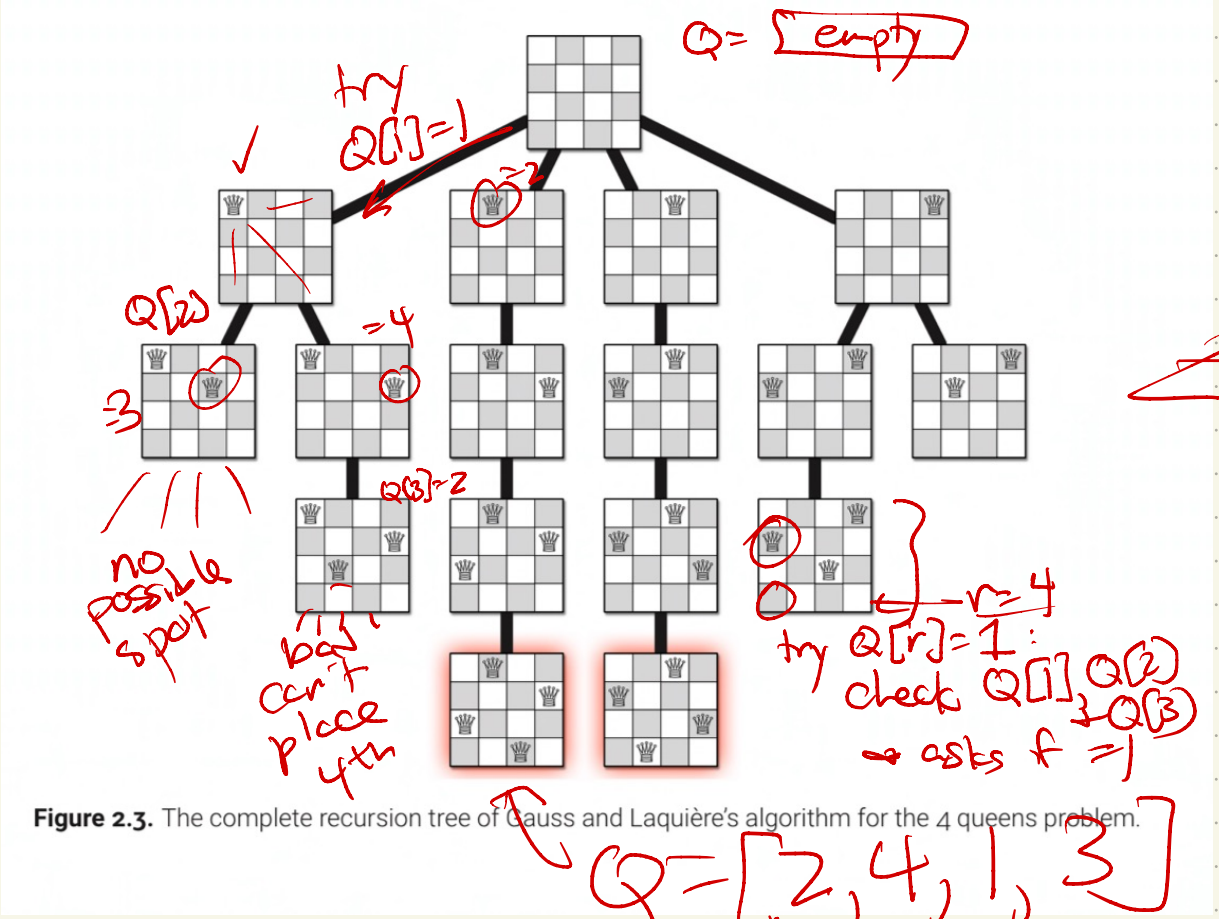


Figure 2.3. The complete recursion tree of Gauss and Laquière's algorithm for the 4 queens problem.

Problem: How to code it?

Result:

```
PLACEQUEENS(Q[1..n], r):
  if r = n + 1
    print Q[1..n]
  else
    for j ← 1 to n
      legal ← TRUE
      for i ← 1 to r − 1
        if (Q[i] = j) or (Q[i] = j + r − i) or (Q[i] = j − r + i)
          legal ← FALSE
      if legal
        Q[r] ← j
        PLACEQUEENS(Q[1..n], r + 1)        ⟨⟨Recursion!⟩⟩
```

Figure 2.2. Gauss and Laquière's backtracking algorithm for the n queens problem.

row I'm trying to fill, so Q[1..r-1] already done

try putting Q[r] = j ie queen in jth spot of row r

Check if queen in upper diagonals

check column j

if legal local, put queen there & move to row r+1

Let's unpack:

Correctness: we try everything!

Runtime:

$Q(n) \leq$ $n^2 + n \cdot Q(n-1)$

All in n rows

Huge! recall: $T(n) = 2T(n-1) + 1$ was already 2^n exponential

```
PLACEQUEENS(Q[1..n], r):
  if r = n + 1
      print Q[1..n]
  else
      for j ← 1 to n
          legal ← TRUE
          for i ← 1 to r − 1
              if (Q[i] = j) or (Q[i] = j + r − i) or (Q[i] = j − r + i)
                  legal ← FALSE
          if legal
              Q[r] ← j
              PLACEQUEENS(Q[1..n], r + 1)        ⟨⟨Recursion!⟩⟩
```

Θ(1)

Figure 2.2. Gauss and Laquière's backtracking algorithm for the n queens problem.

n-1 rows left

Game Trees:

a way to model moves in 2-player games

Assume:

- No randomness so the game is just 2 people taking turns

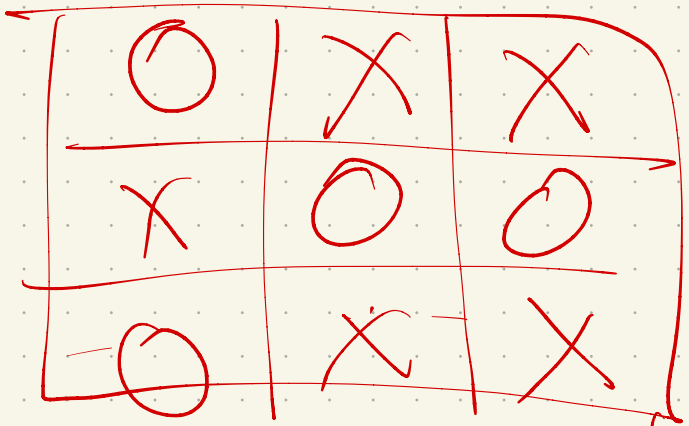
  Ex: Checkers, chess, Nim, Go
  (not settlers of Catan!)

- "Perfect" players:

  Makes rational decisions, & if there is a move to get them to a win state, they do it!

Tic-tac-toe!

player 1: X
player 2: O

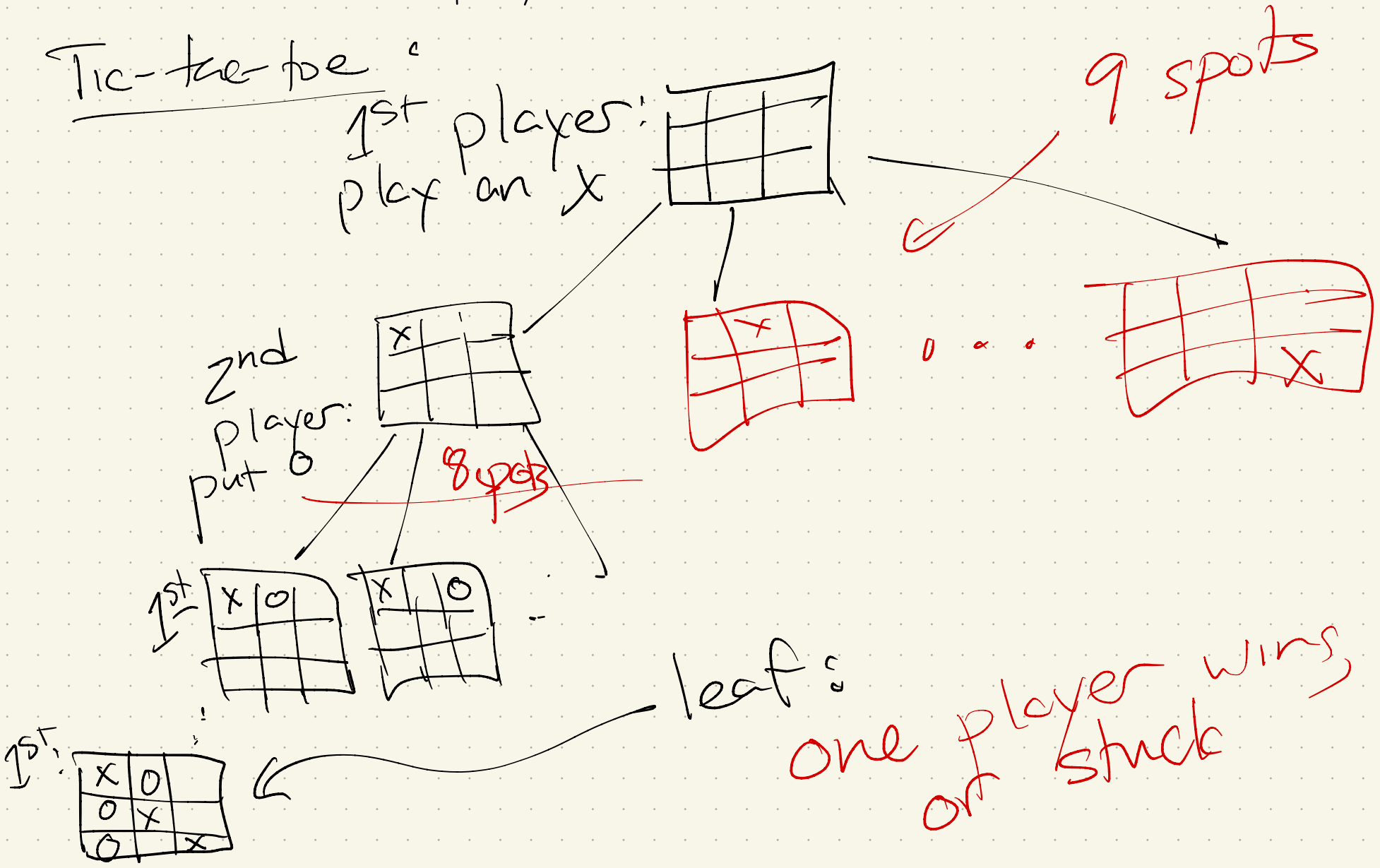


Stuck

Win: 3 in any row or col or diagonal

Idea: Track current state of the game, as play occurs

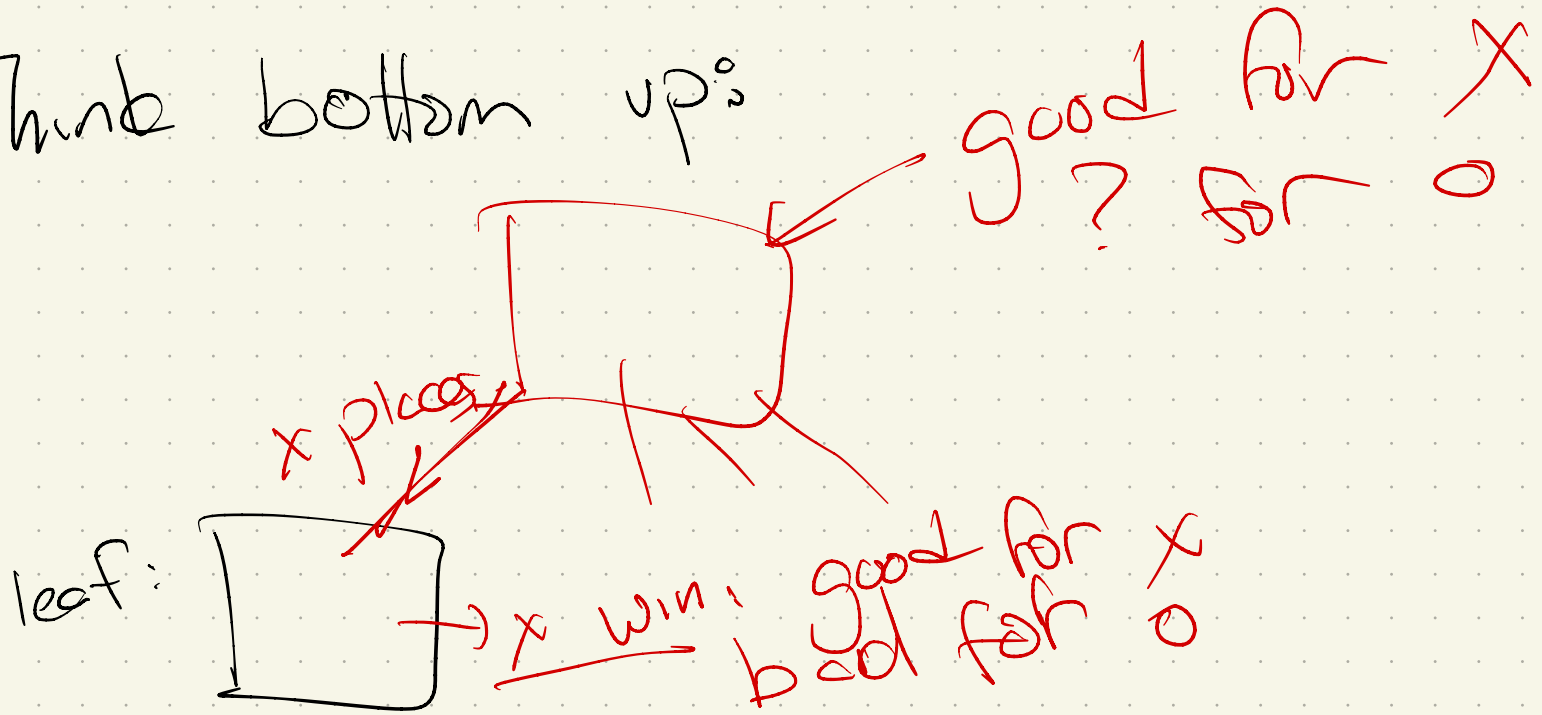
Tic-tac-toe:



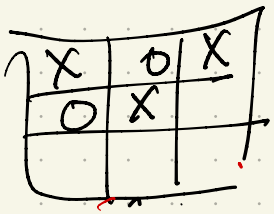
A state is good for player 1 if they either have won, or could move to a bad state for player 2.

and bad if they have lost, or if all possible moves lead to a state that is good for player 2.

Think bottom up:



2's turn

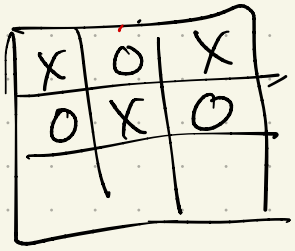


good or bad?

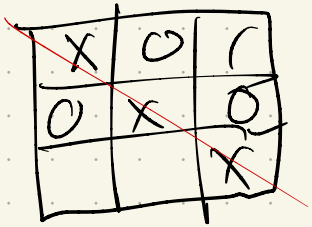
Some other way for O to win

(in rest of tree)

1's turn



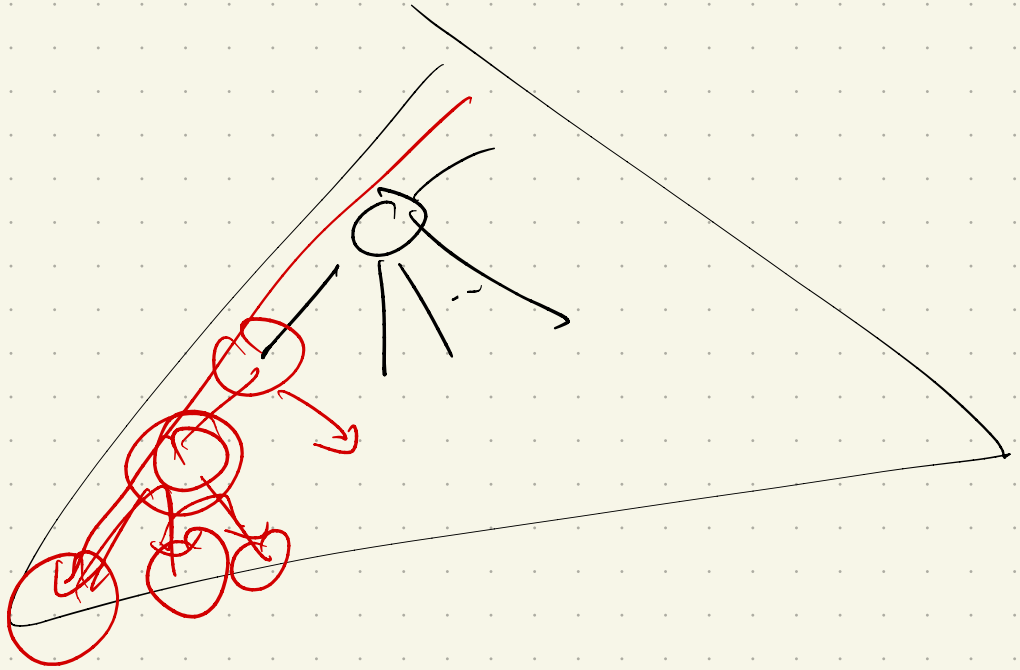
This is good for 1 (He can move somewhere bad for 2)



good for 1
bad for 2

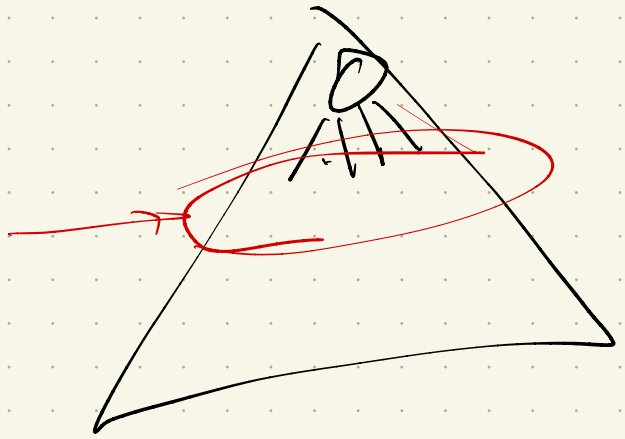
So:

good:
I have a child who other guy thinks is bad:



Bad

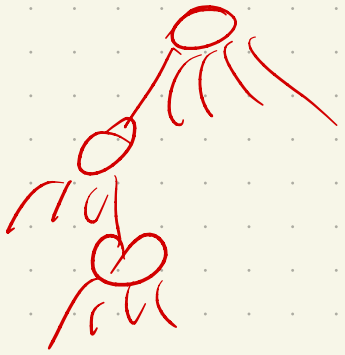
All of these are good for other guy



postorder traversals!
labels leaves + move up

Downsides:

Game trees are HUGE!



Tic-tac-to: over 200,000 leaves

People can still "predict":
we're good at inferring state/strategy
intuitively, with practice

Computers have to search.

Hence - took 60 years to get a decent computer chess player!
Need "heuristics" (aka guesses) to make it work.

Game theory — a bit more complicated.

Here, we assume clear win vs. lose

Game theory suggests more subtle possibilites, as well as simultaneous moves & "randomness"

Example: Odds and Evens

Consider the simple game called **odds and evens**. Suppose that player 1 takes evens and player 2 takes odds. Then, each player simultaneously shows either one finger or two fingers. If the number of fingers matches, then the result is *even*, and player 1 wins the bet ($2). If the number of fingers does not match, then the result is *odd*, and player 2 wins the bet ($2). Eac player has two possible strategies: show one finger or show two fingers. The *payoff matrix* shown below represents the payo to player 1.

Payoff Matrix

Strategy		Player 2: 1	Player 2: 2
Player 1	1	2	-2
	2	-2	2

Even if both know outcomes, result is unclear!

Example: Subset Sum

Given a set X of positive integers and a target value t, is there a subset of X which sums to t?

Ex: $X = \{8, 6, 7, 3, 10, 5, 9\}$ $t = 15$

Yes: $6+9$

$7+3+5$

$10+5$

$7+8$

build a subset whose sum = t

$t = 4$: no

Consider one at a time:

$X = \{8, 6, 7, 5, 3, 1, 9\}$

8 is either in or not

Recursion!

try recursing with 8 in subset
- set $t = t - X[0]$
- try recursing on $X[2..n]$ with t

try recursing without
↳ recurse on $X[2..n]$, with t

Base cases:

if $t = 0$, done!

if $t \neq 0$, and X is empty
↳ fail

Algorithm:

reset to use arrays.

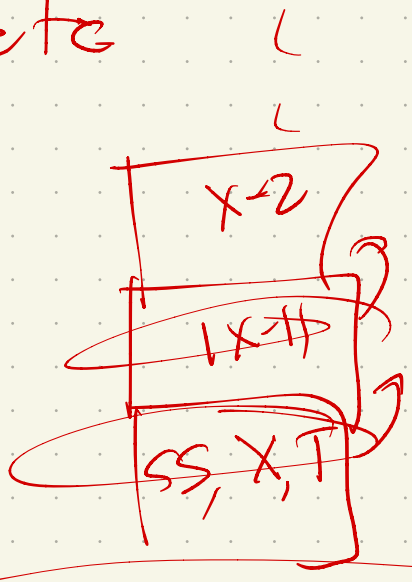
```
⟨⟨Does any subset of X sum to T?⟩⟩
SUBSETSUM(X, T):
  if T = 0
      return TRUE
  else if T < 0 or X = ∅
      return FALSE
  else
      x ← any element of X
      with ← SUBSETSUM(X \ {x}, T − x)      ⟨⟨Recurse!⟩⟩
      wout ← SUBSETSUM(X \ {x}, T)          ⟨⟨Recurse!⟩⟩
      return (with ∨ wout)
```

T/F ? ? T/F

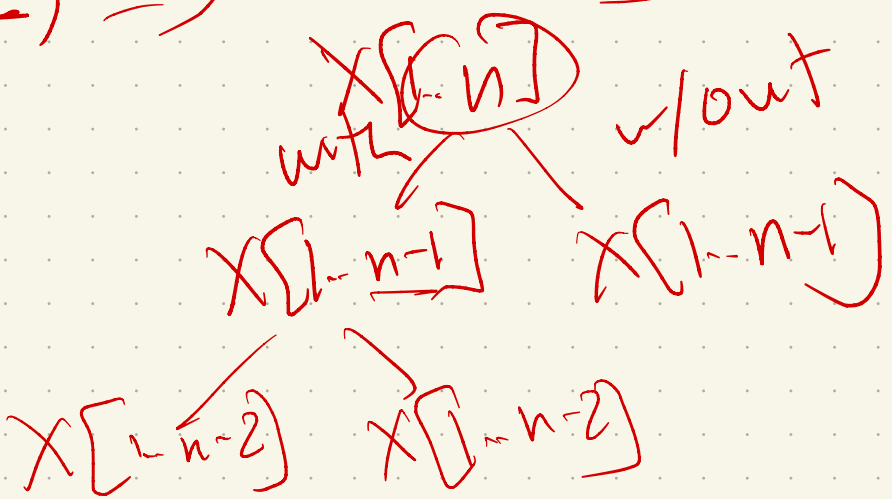
OR!

```
⟨⟨Does any subset of X[1..i] sum to T?⟩⟩
SUBSETSUM(X, i, T):
  if T = 0
      return TRUE
  else if T < 0 or i = 0
      return FALSE
  else
      with ← SUBSETSUM(X, i − 1, T − X[i])   ⟨⟨Recurse!⟩⟩
      wout ← SUBSETSUM(X, i − 1, T)          ⟨⟨Recurse!⟩⟩
      return (with ∨ wout)
```

choose 1st, then 2nd, etc



SS(X, n, T)



Correctness: inductive proof, on size of X

X[1..i]

Base cases:

$i = |X| = 0$ (so $X = \{\}$):

check if T = 0 ✓ correct

Ind Hyp: works for X[1..i-1] or smaller

values of T

size i-1

Ind step: Full array X[1..i]

Consider X[i]: only two options

- try including
- try not including

↳ size of X sets in smaller recursive calls

Runtime:

$$T(n) = 2T(n-1) + O(1)$$

$$r - 2 = 0$$

$$\Rightarrow O(2^n)$$

Next reading:

different backtracking

↳ good: best exponential