04-12-2023

CS051A

INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

21: More adversarial search

Alexandra Papoutsaki

she/her/hers

Lectures

Zilong Ye

he/him/his

Labs

TODAY'S LECTURE IN A NUTSHELL

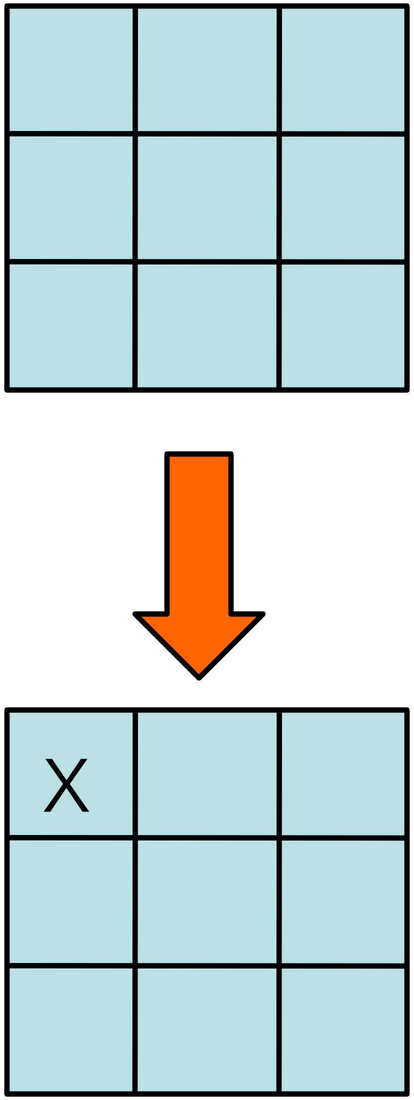
Lecture 22: More adversarial search

- Minimax

Back to tic tac toe

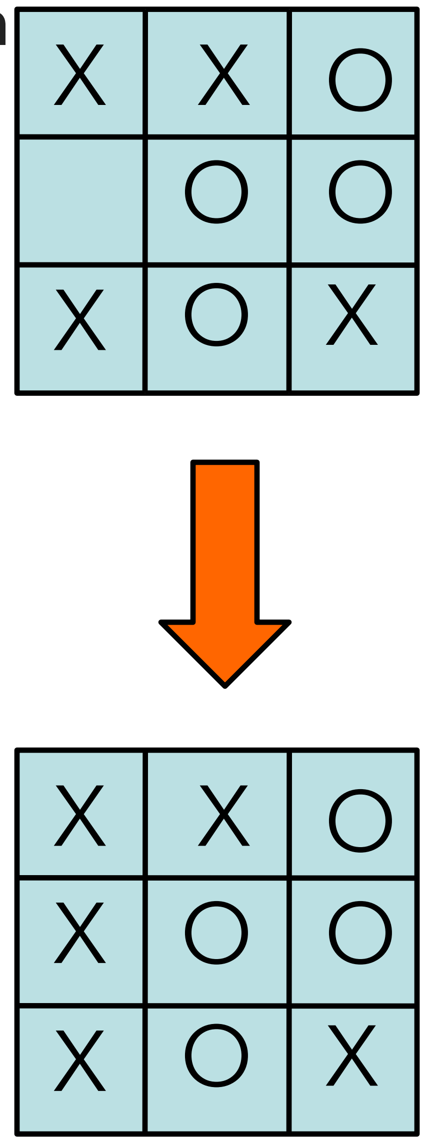
- If we want to write a program to play tic tac toe, what question are we trying to answer?
- Given a state (i.e. board configuration), what move should we make!

Tic tac toe as search

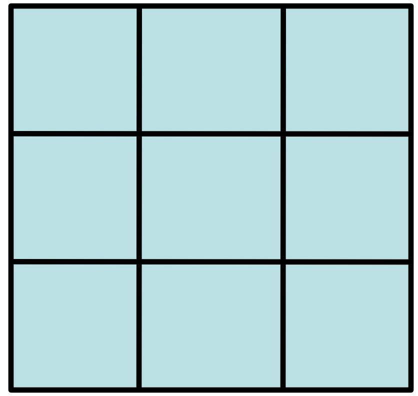


MINIMAX

Tic tac toe as search



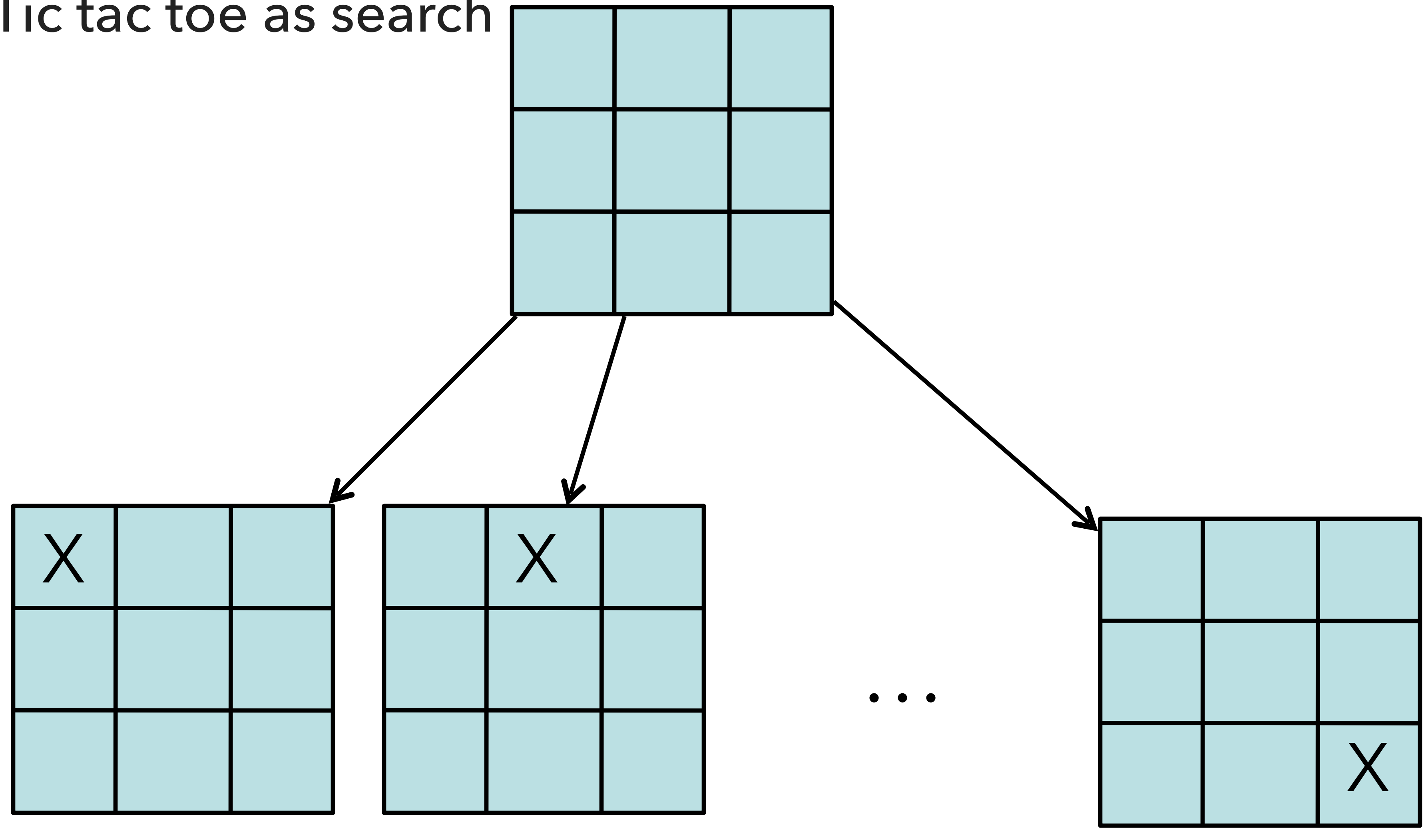
Tic tac toe as search



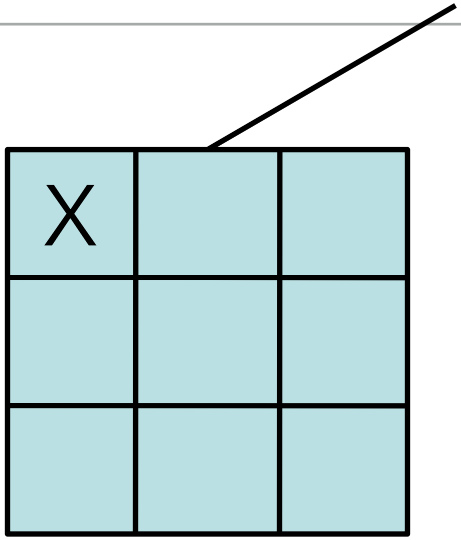
If we want to write a program to play tic tac toe, what question are we trying to answer?

MINIMAX

Tic tac toe as search

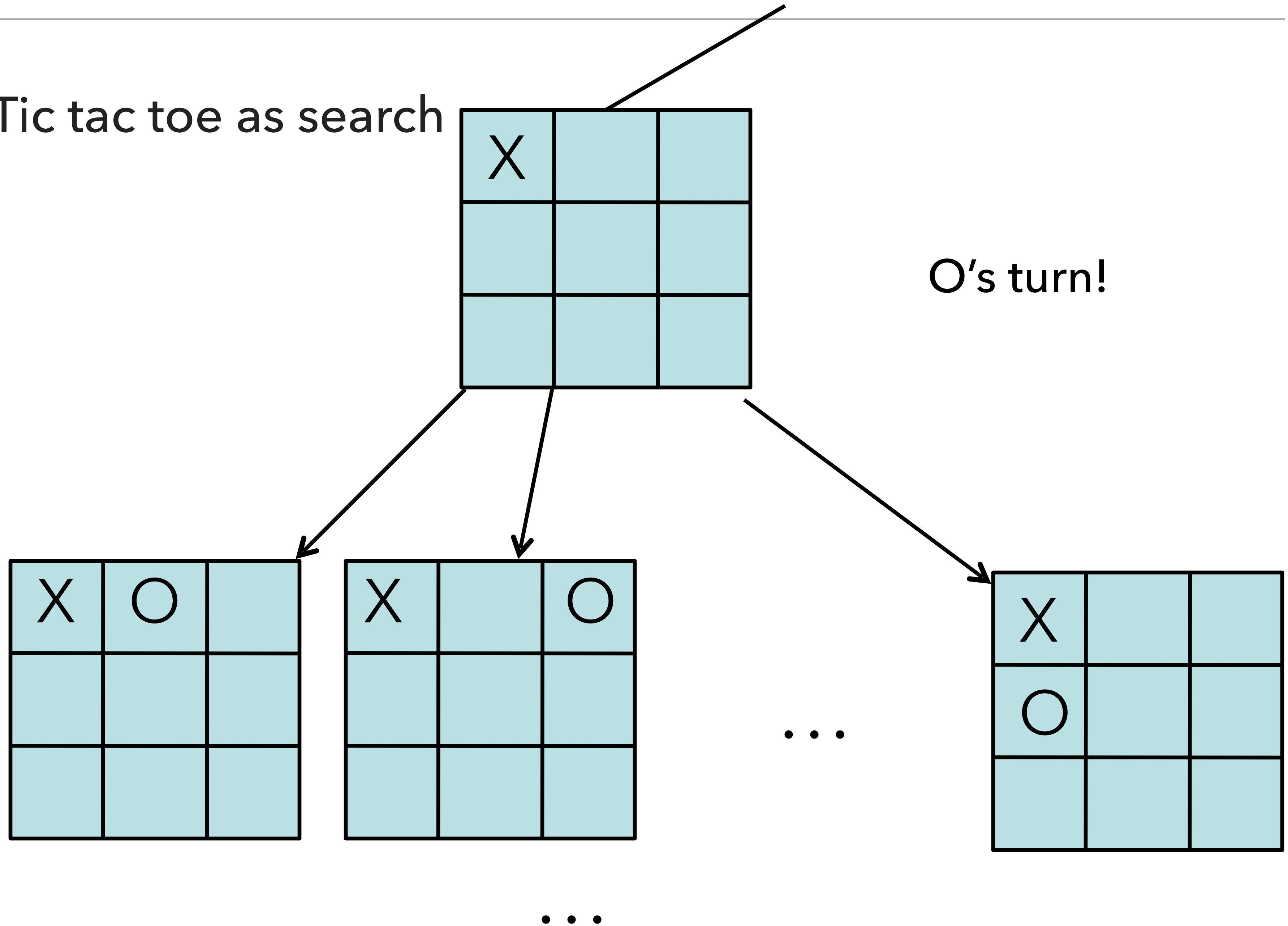


Tic tac toe as search



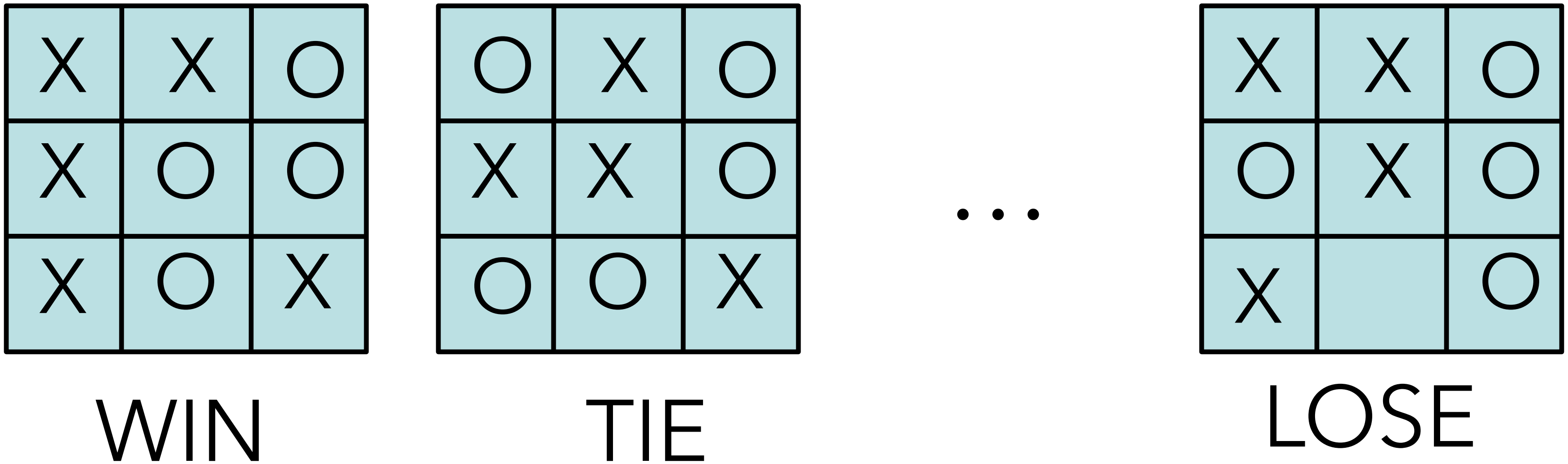
Now what?

Tic tac toe as search



Tic tac toe as search

Eventually, we'll get to a leaf



How does this help us?

Try and make moves that move us towards a win, i.e. where there are leaves with a WIN.

MINIMAX

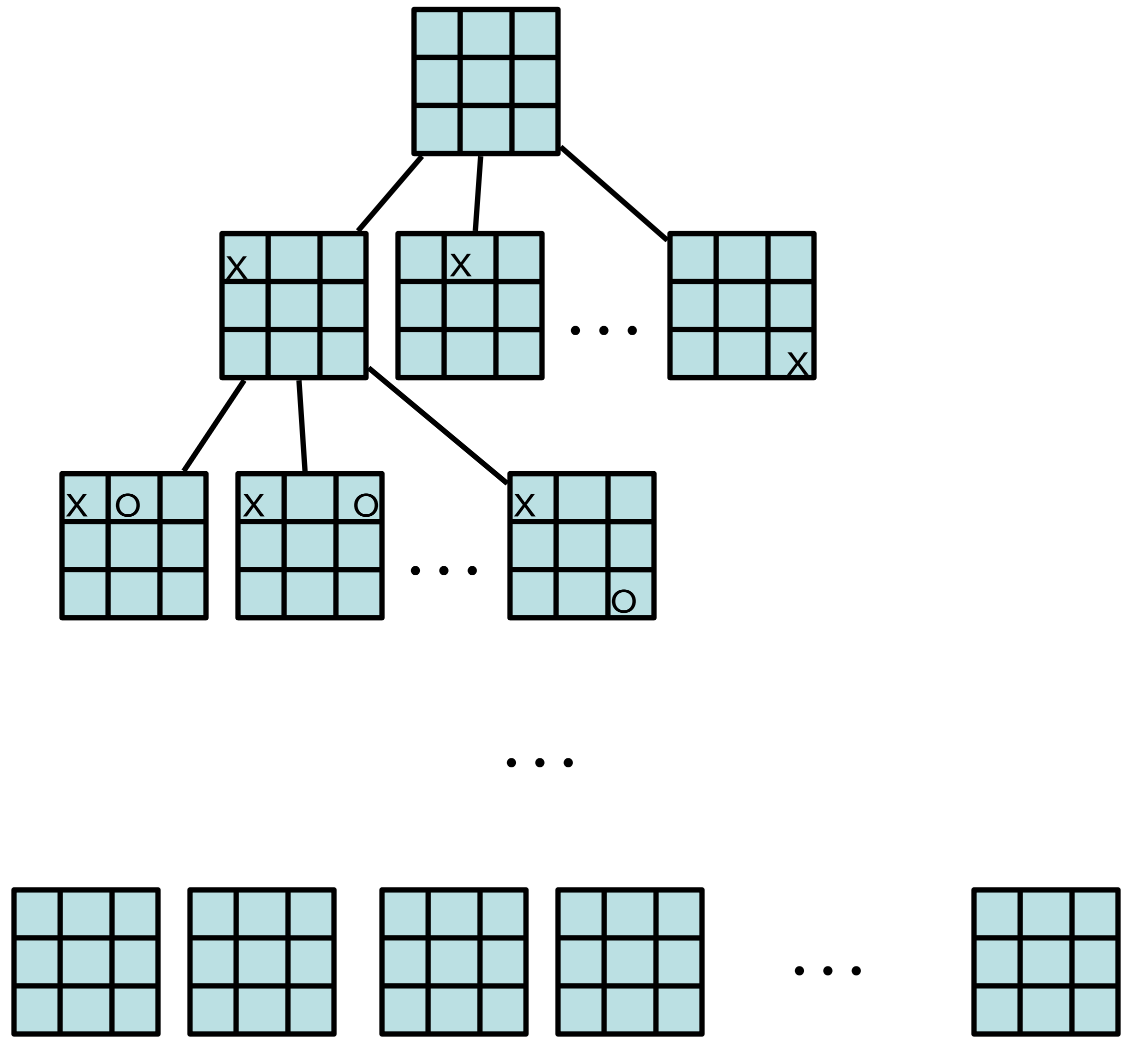
Tic tac toe as search

X’s turn

O’s turn

X’s turn

. . .

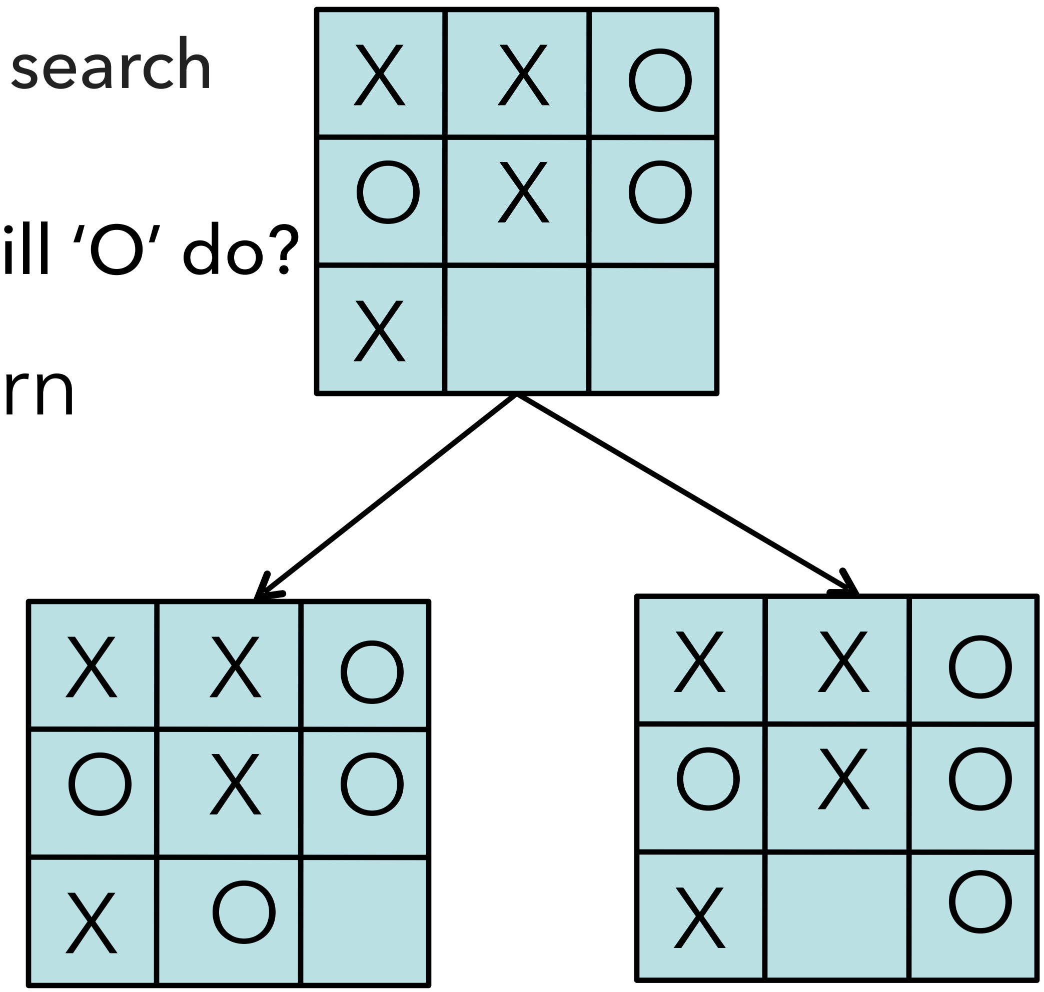


Problem: we don’t know what O will do

Tic tac toe as search

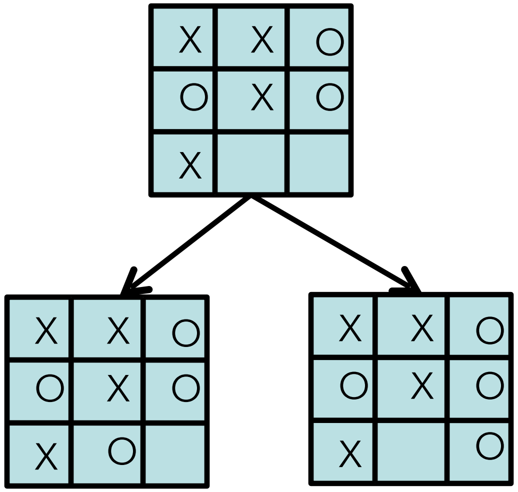
I'm X, what will 'O' do?

O's turn



Minimizing risk

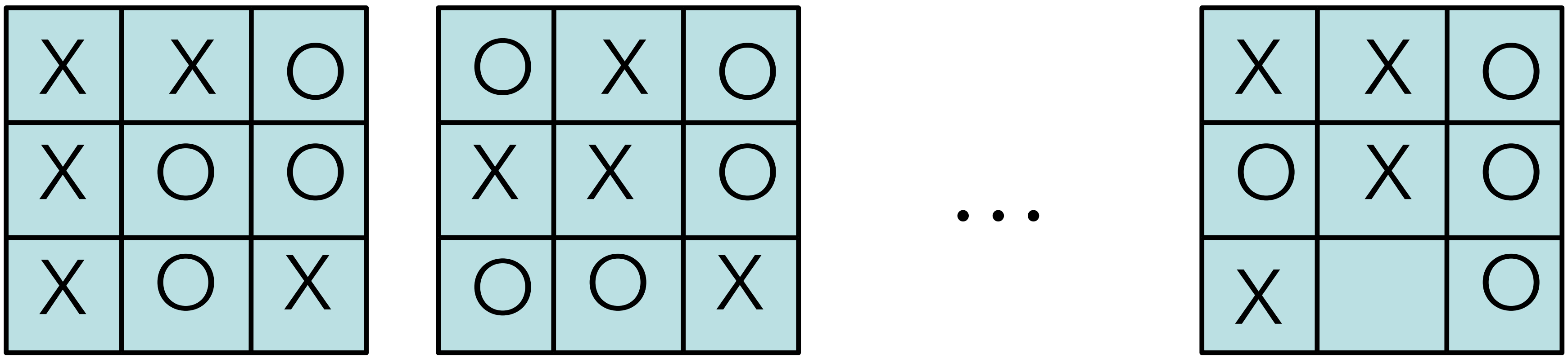
- The computer doesn't know what move O (the opponent) will make.
- It can assume that it will try and make the best move possible.
- Even if O actually makes a different move, we're no worse off. Why?



Optimal strategy

- An optimal strategy is one that is at least as good as any other, no matter what the opponent does.
  - If there's a way to force the win, it will
  - Will only lose if there's no other option

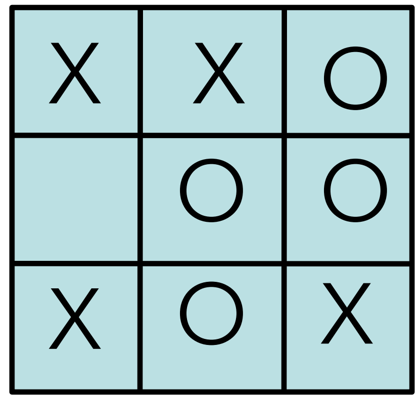
Defining a scoring function



- Idea:
  - define a function that gives us a “score” for how good each state is
  - higher scores mean better

Defining a scoring function

Our (X) turn



What should be the score of this state?

Defining a scoring function

Our (X) turn

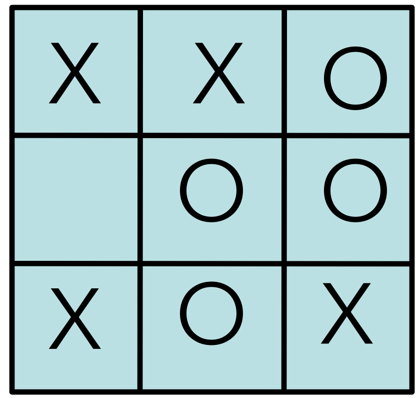
X	X	O
	O	O
X	O	X

What should be the score of this state?

+1: we can get to a win

Defining a scoring function

Opponent’s (O) turn



What should be the score of this state?

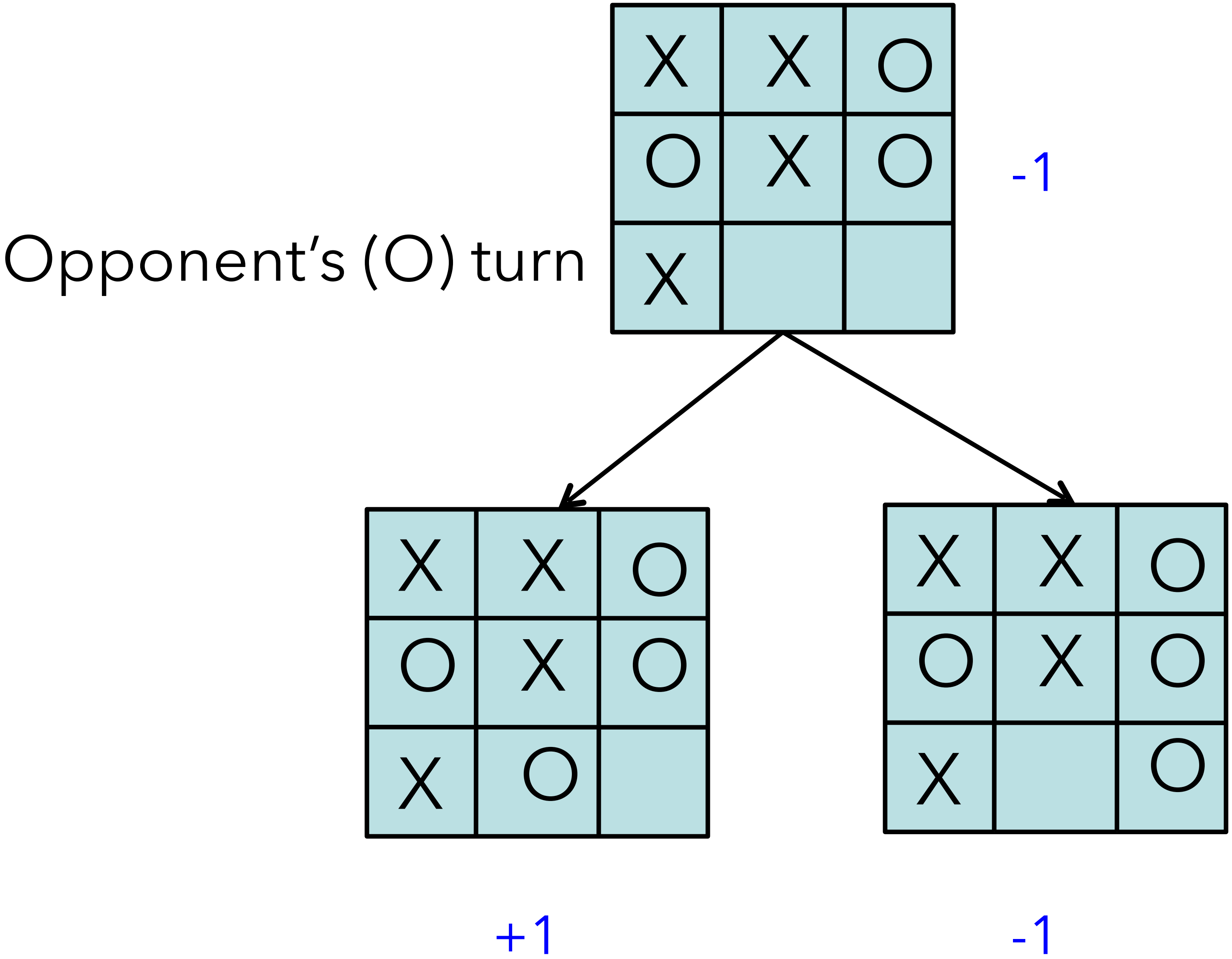
Defining a scoring function

Opponent's (O) turn

X	X	O
	O	O
X	O	X

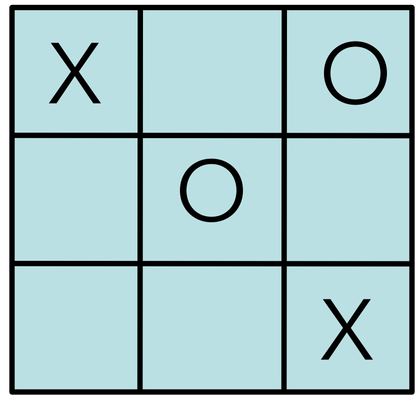
What should be the score of this state?

-1: opponent can get to a win


Opponent's (O) turn
X X O
O X O
X
-1
X X O
O X O
X O
+1
X X O
O X O
X O
-1

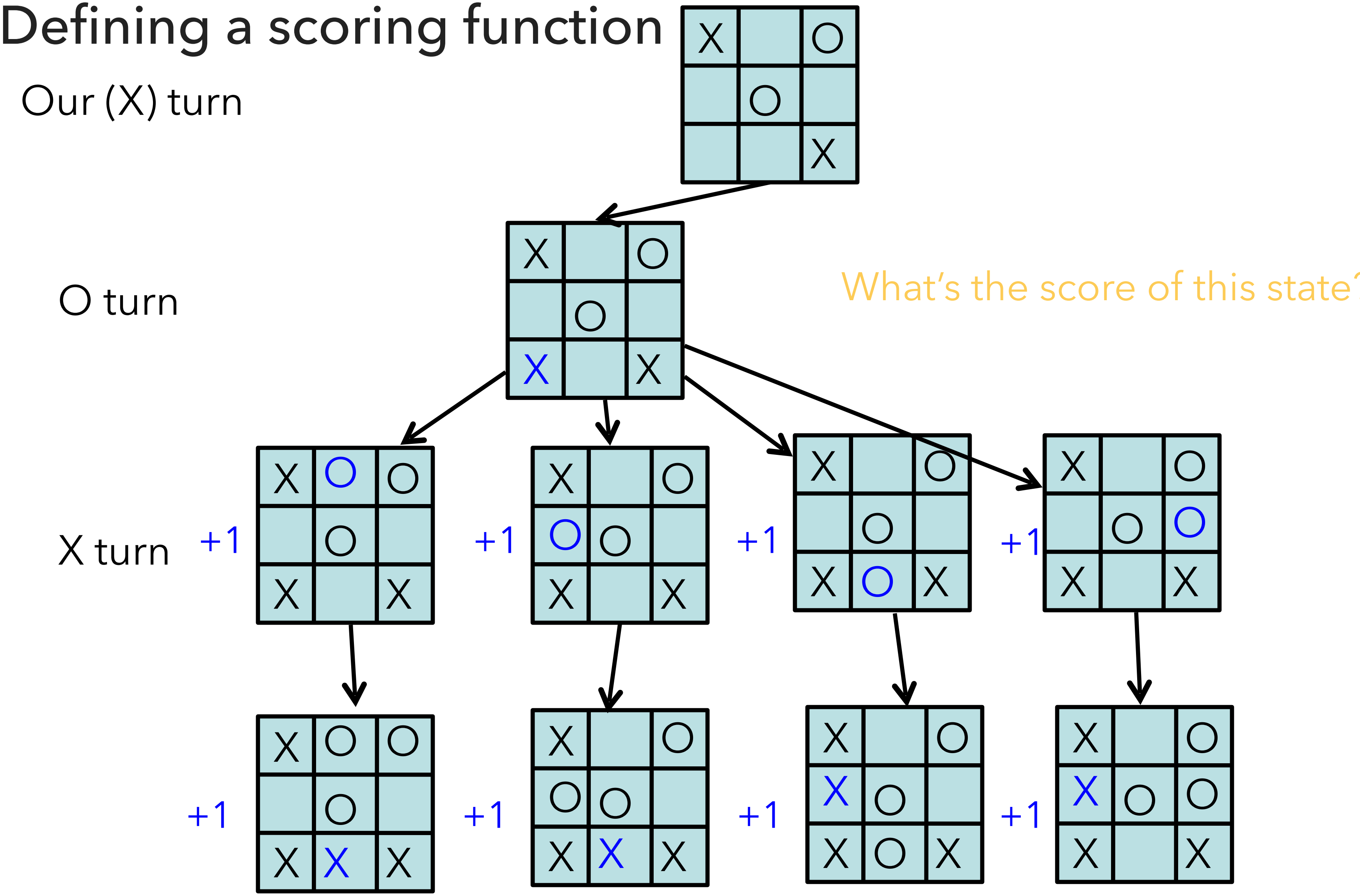
Defining a scoring function

Our (X) turn

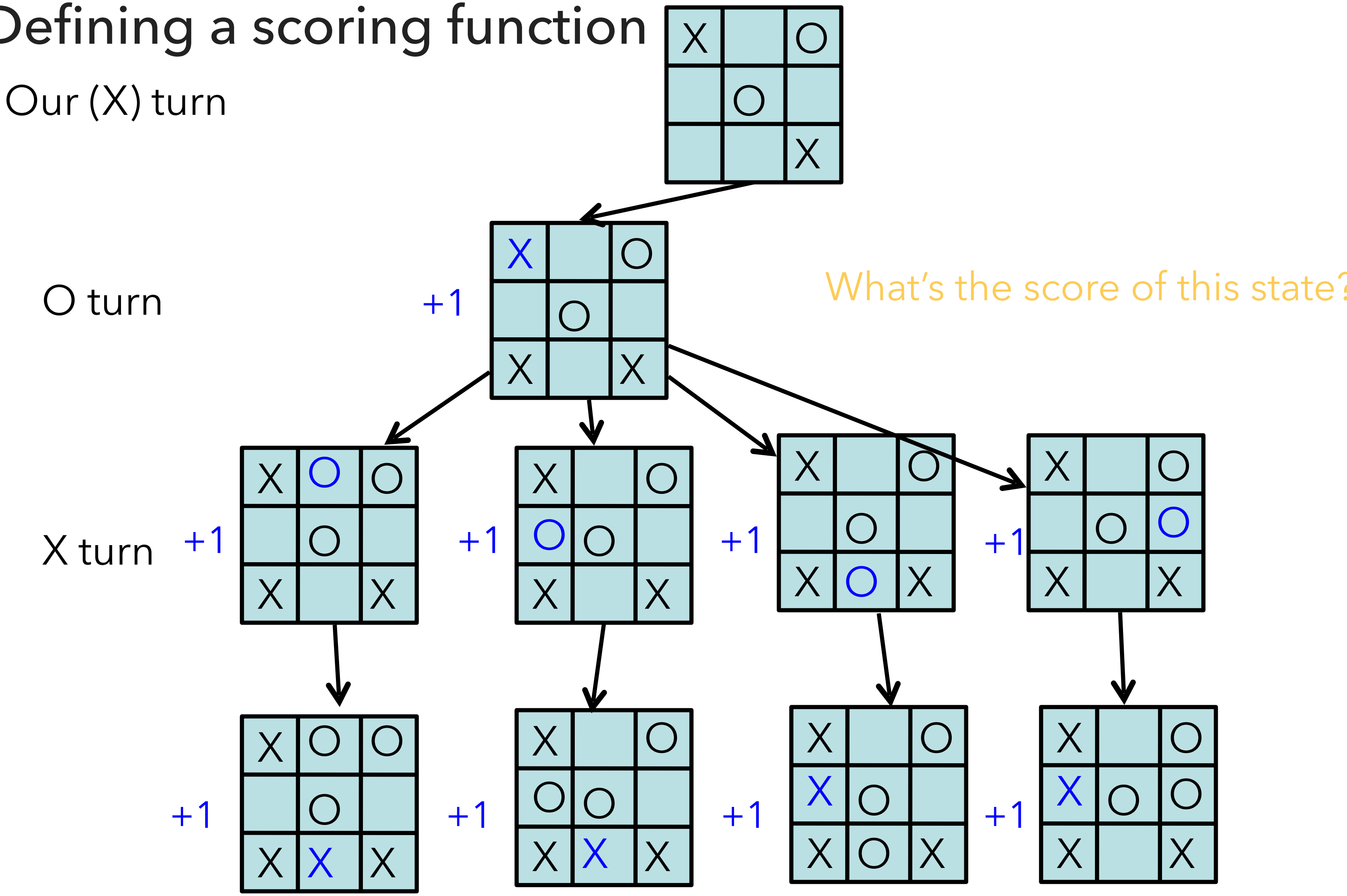


What should be the score of this state?

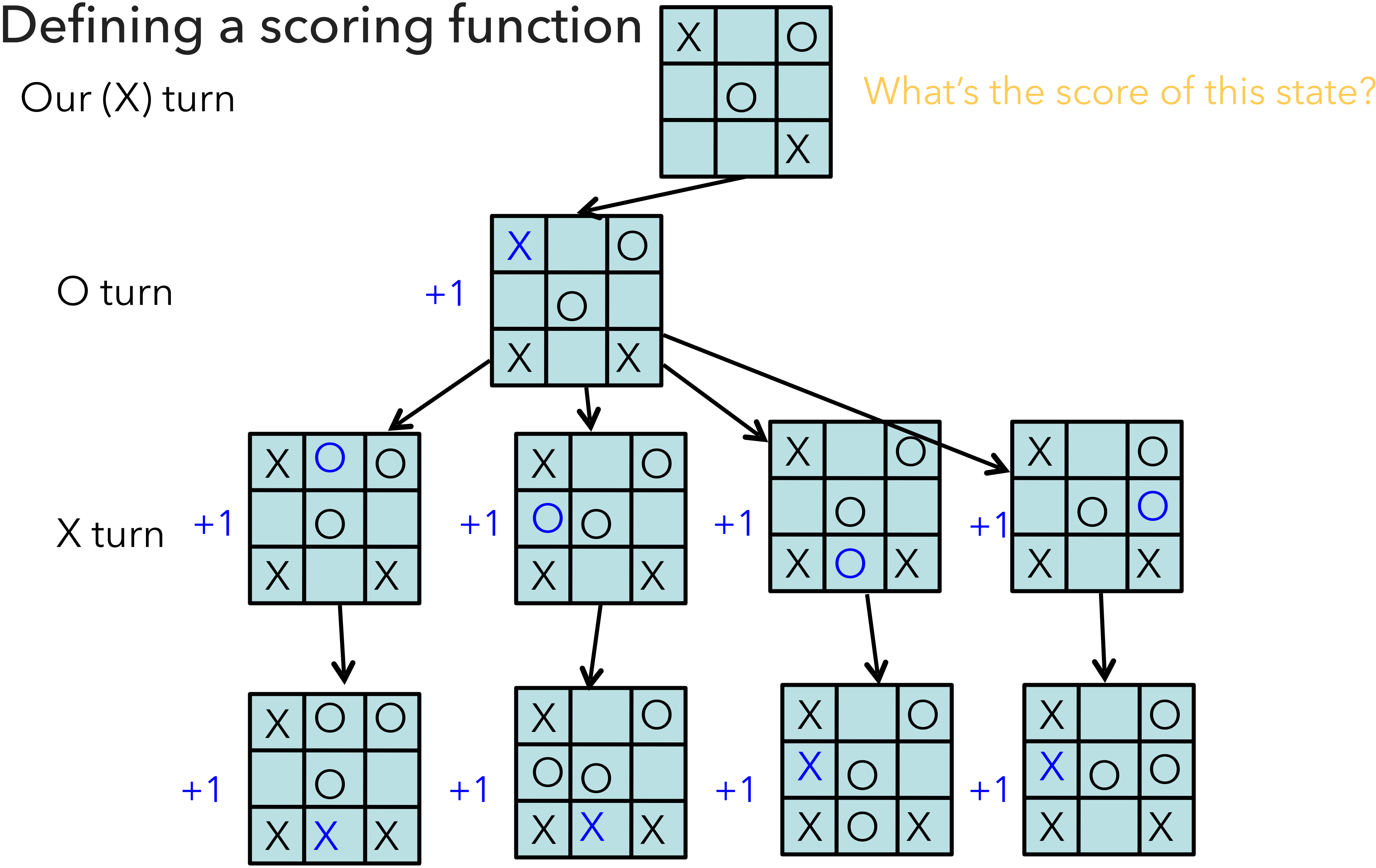
Defining a scoring function



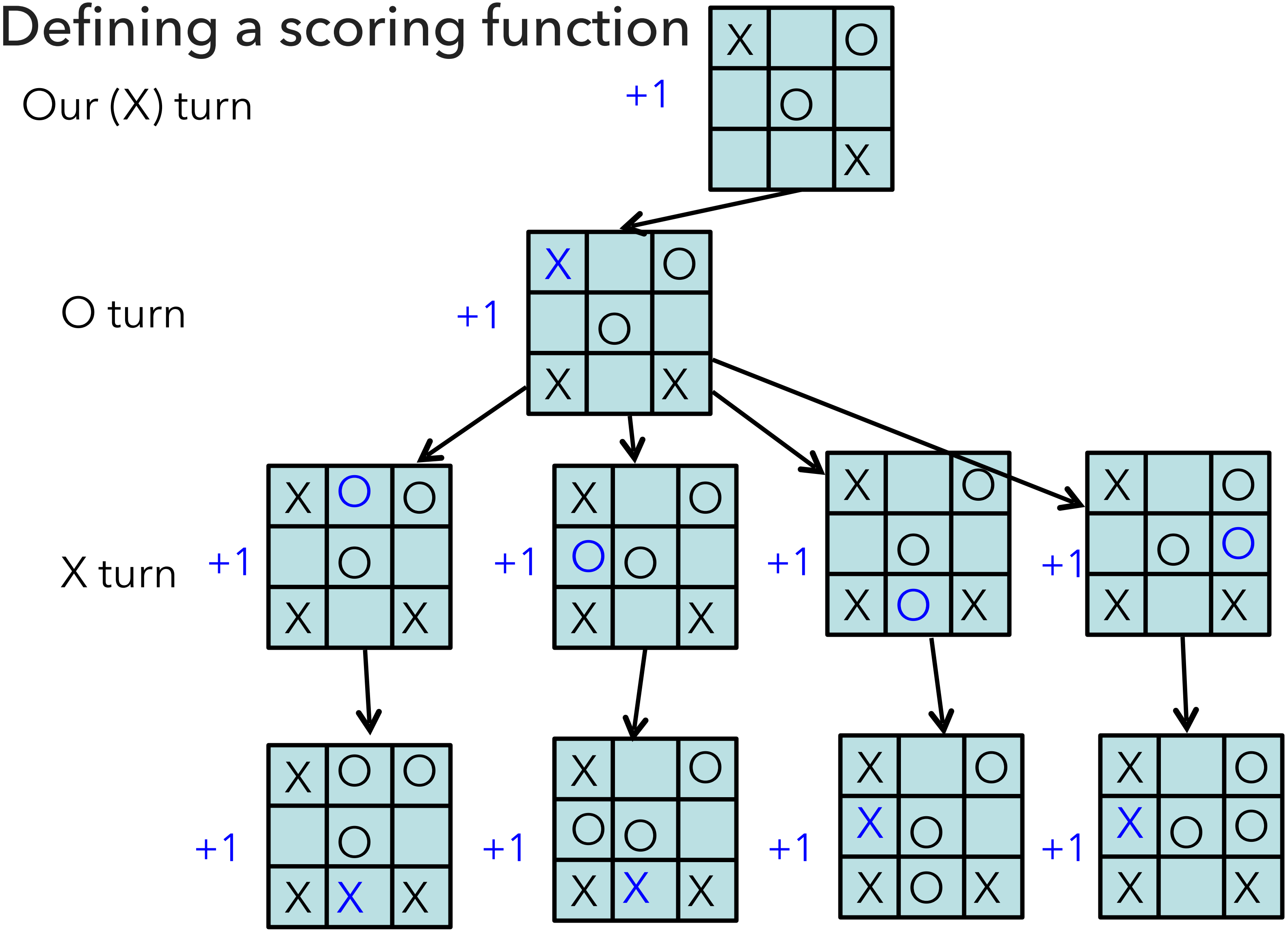
Defining a scoring function



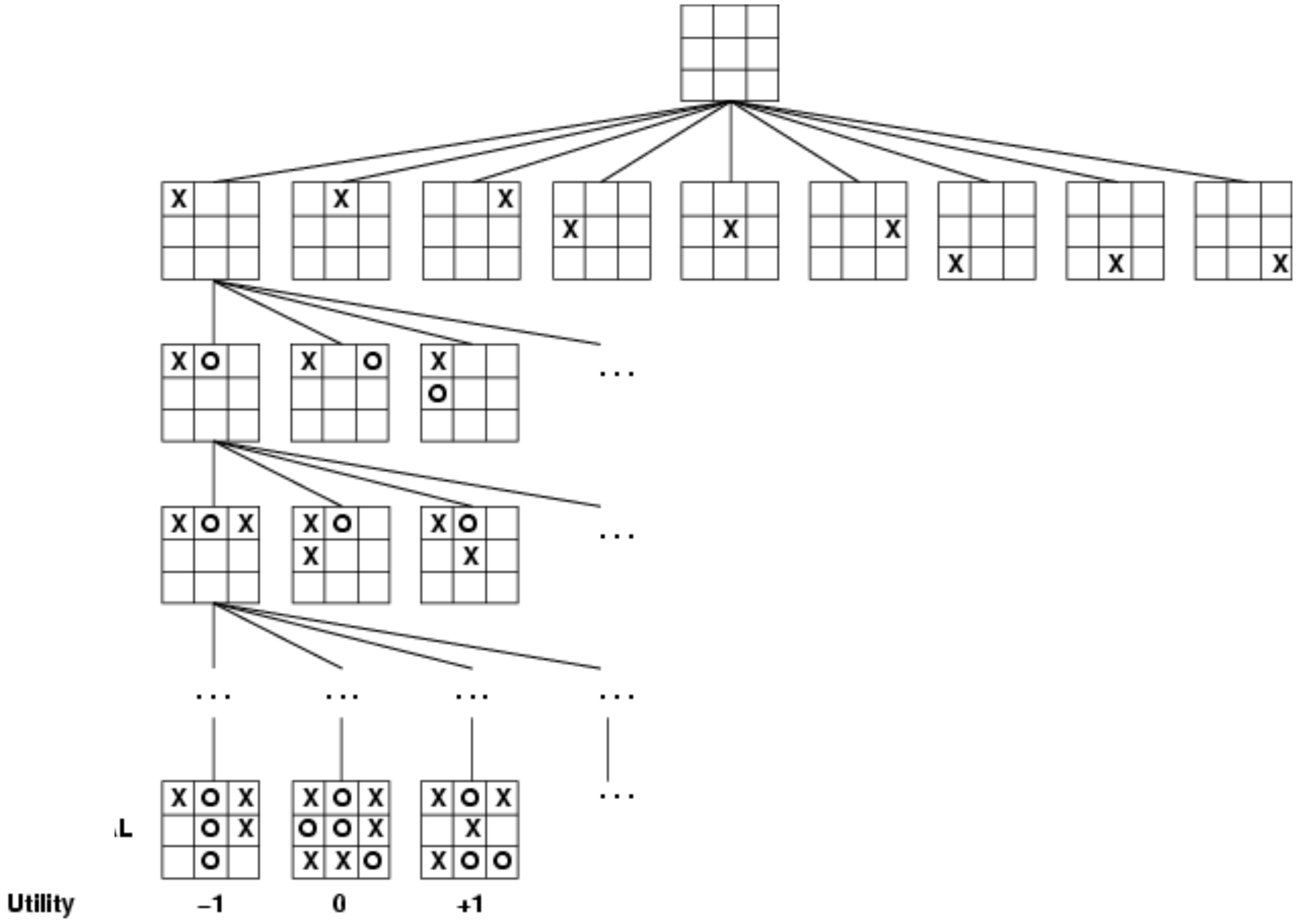
Defining a scoring function



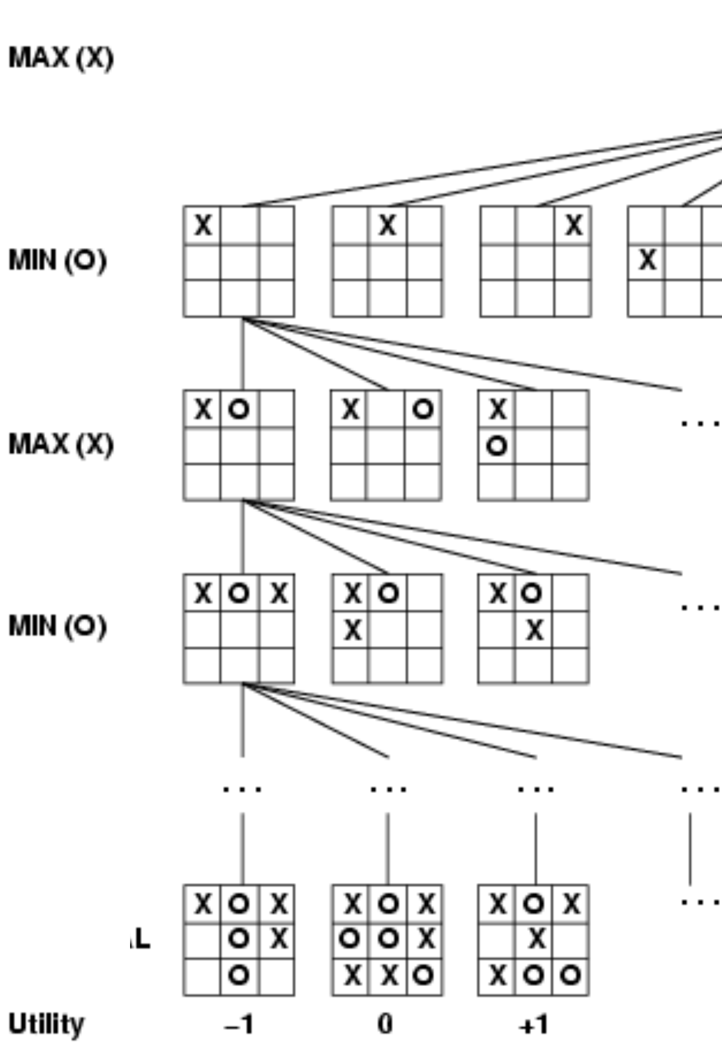
Defining a scoring function



How can X play optimally?



How can X play optimally?



When it's my turn, pick the highest scoring state

When it's the opponent's turn, assume the lowest scoring state (from my perspective)

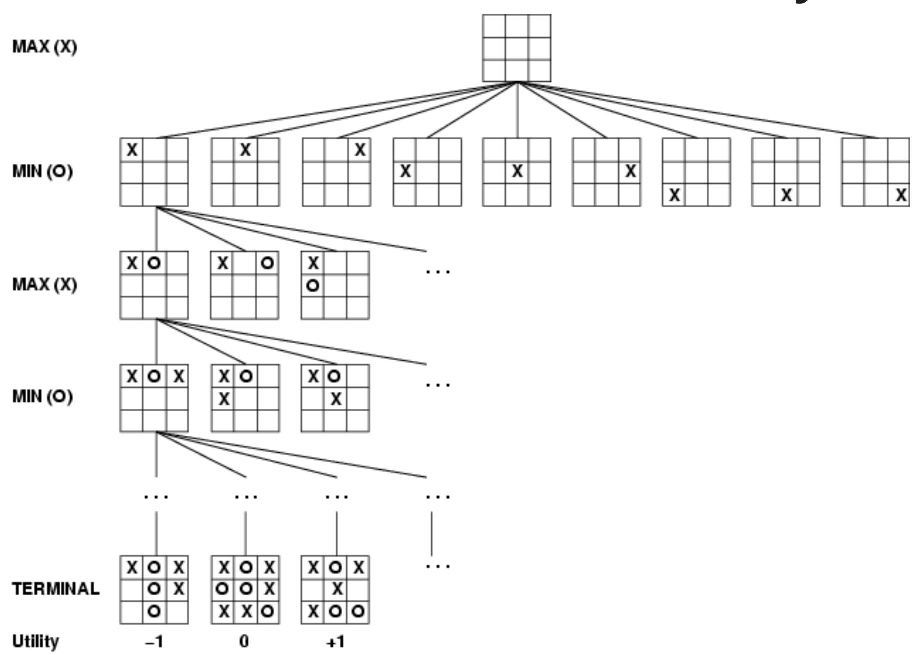
If we can reach the end games, we can percolate these answers all the way back up

How can X play optimally?

Start from the bottom and propagate the score up:

- if X's turn, pick the move that maximizes the utility
- if O's turn, pick the move that minimizes the utility

Is this optimal?



Minimax Algorithm: An Optimal Strategy

```
minimax(state) =
     if state is a terminal state
          score(state)
     else if MY turn
          over all next states, s: return the maximum of minimax(s)
     else if OPPONENTS turn
          over all next states, s: return the minimum  of minimax(s)
```

Uses recursion to compute the “value” of each state

Searches down to the leaves, then the values are “backed up” through the tree as the recursion finishes

Minimax Algorithm: An Optimal Strategy

```
minimax(state) =
     if state is a terminal state
          score(state)
     else if MY turn
          over all next states, s: return the maximum of minimax(s)
     else if OPPONENTS turn
          over all next states, s: return the minimum  of minimax(s)
```

Uses recursion to compute the "value" of each state

Searches down to the leaves, then the values are "backed up" through the tree as the recursion finishes

What type of search is this?

Minimax Algorithm: An Optimal Strategy

```
minimax(state) =
     if state is a terminal state
          score(state)
     else if MY turn
          over all next states, s: return the maximum of minimax(s)
     else if OPPONENTS turn
          over all next states, s: return the minimum  of minimax(s)
```

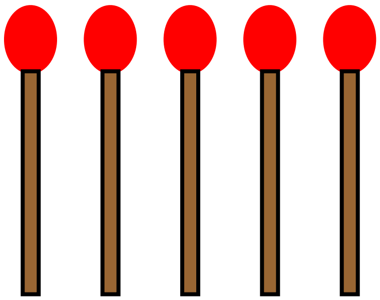
Uses recursion to compute the “value” of each state

Searches down to the leaves, then the values are “backed up” through the tree as the recursion finishes

What type of search is this?

DFS!

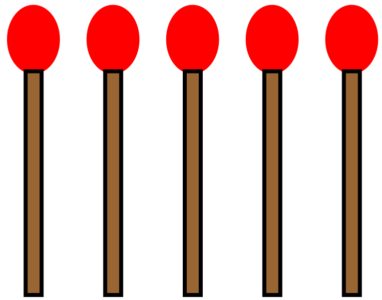
Baby Nim



Take 1 or 2 at each turn
Goal: take the last match

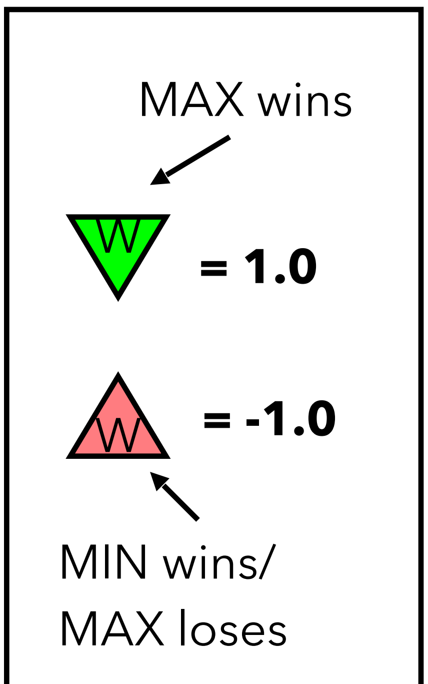
What move should I make?

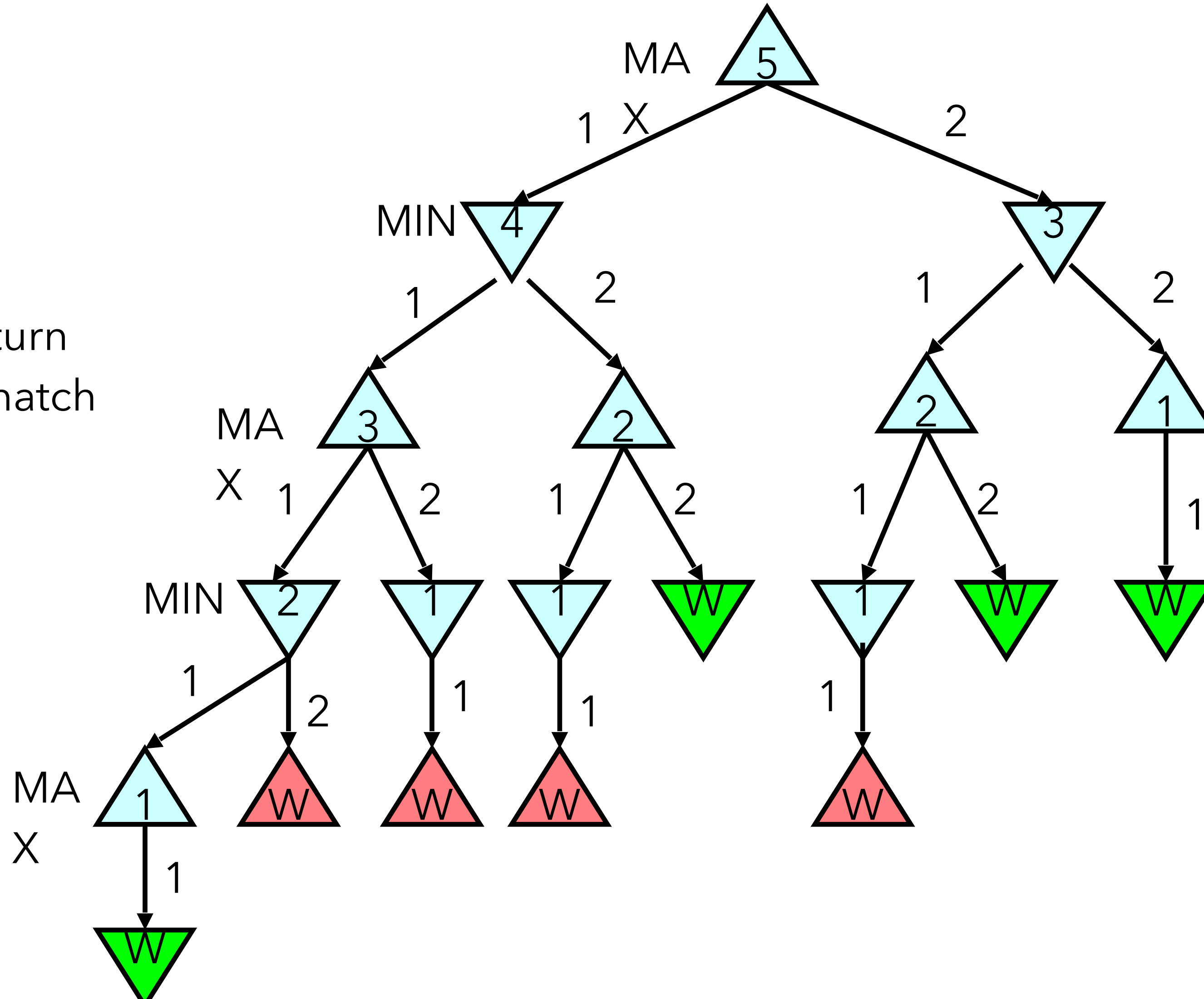
MINIMAX



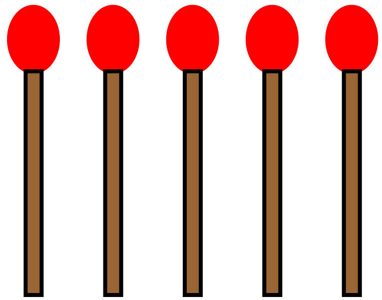
Take 1 or 2 at each turn

Goal: take the last match

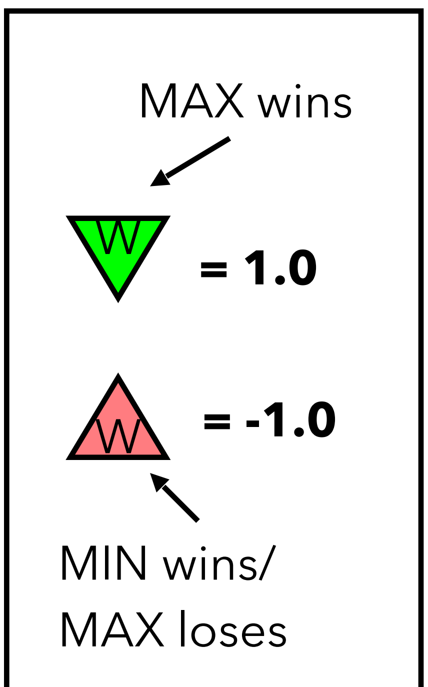


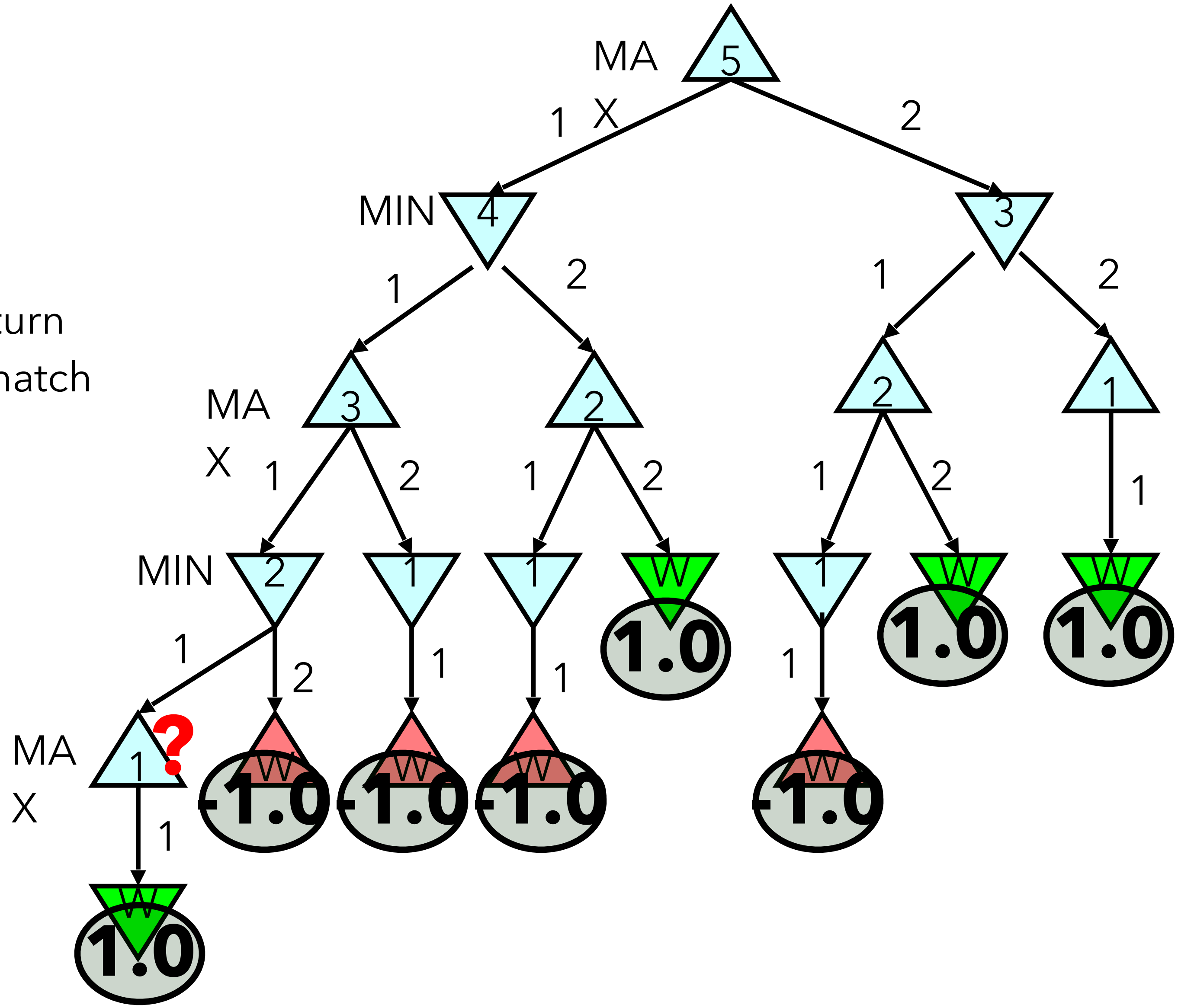


MINIMAX

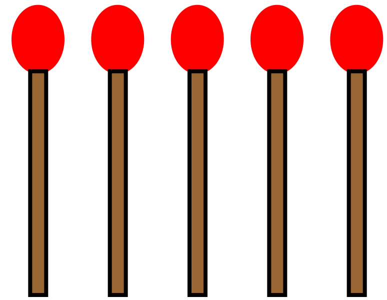


Take 1 or 2 at each turn
Goal: take the last match

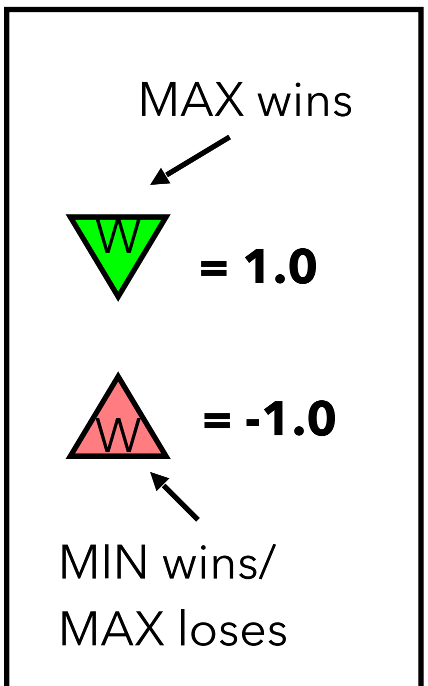


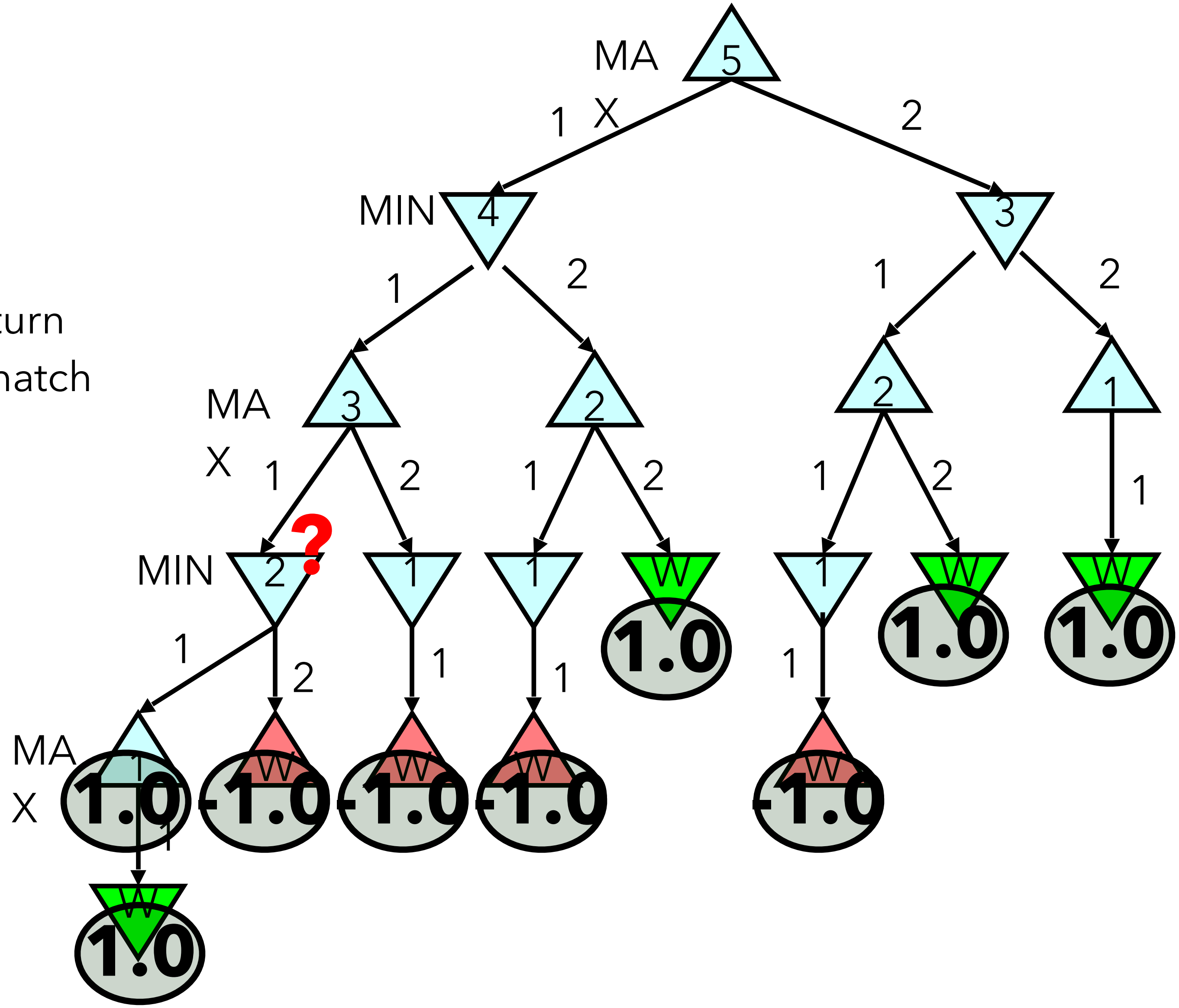


MINIMAX

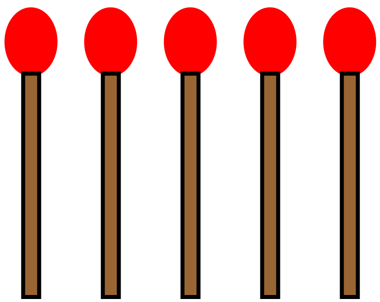


Take 1 or 2 at each turn
Goal: take the last match

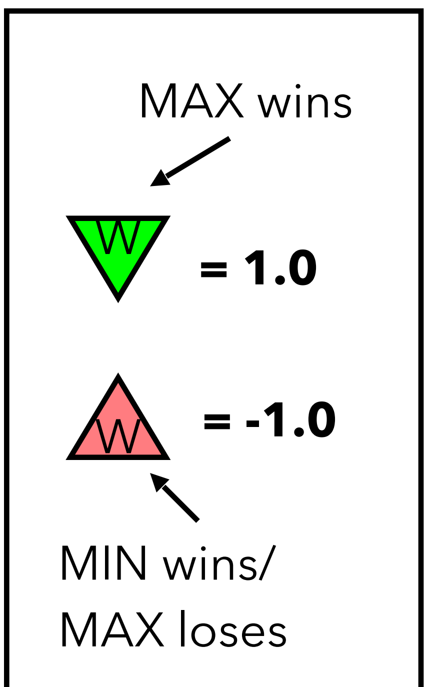


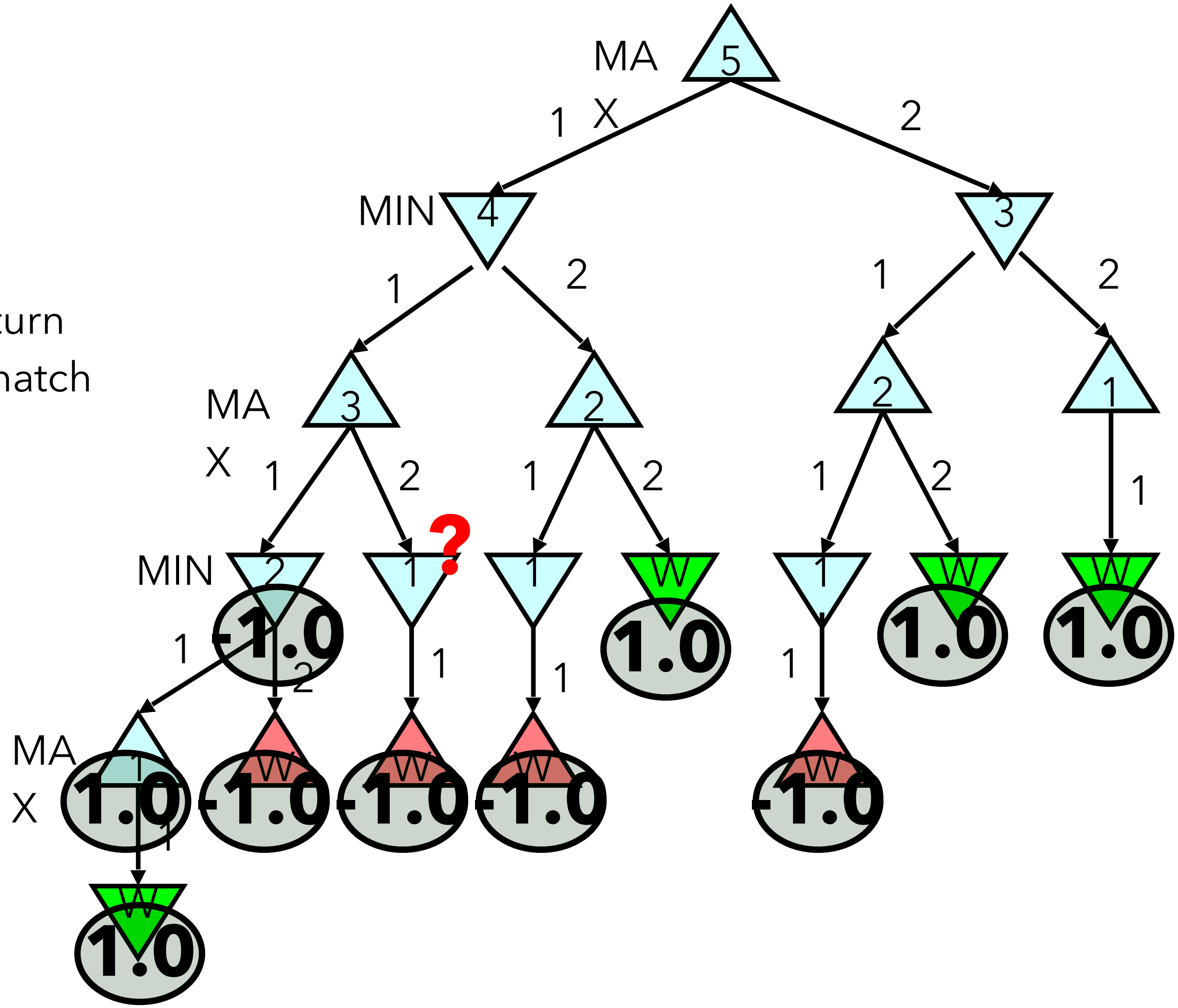


MINIMAX

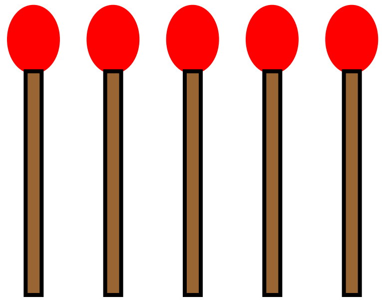


Take 1 or 2 at each turn
Goal: take the last match

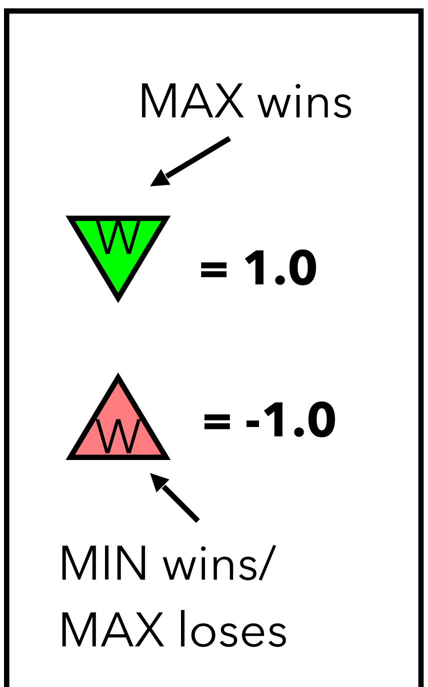


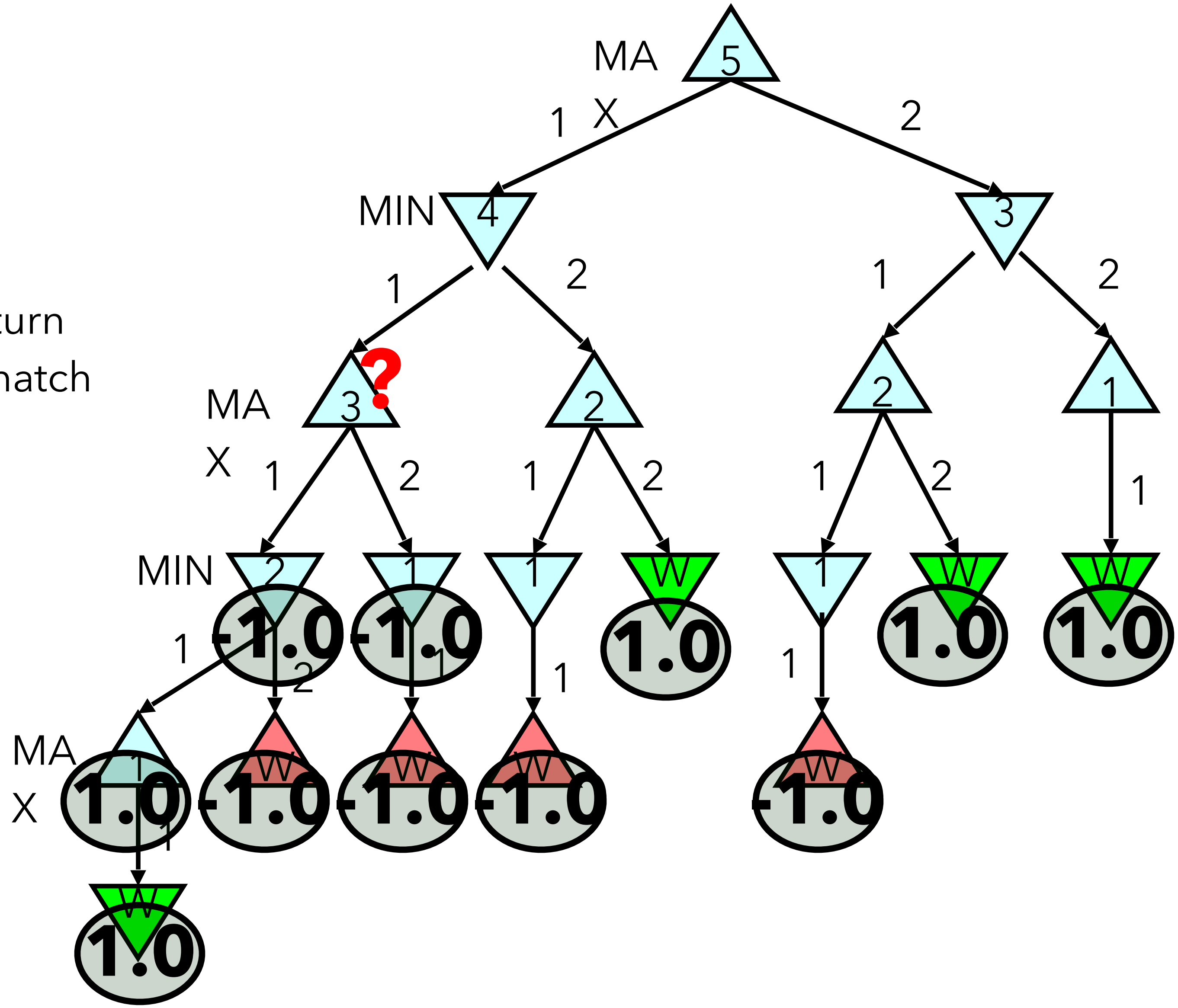


MINIMAX

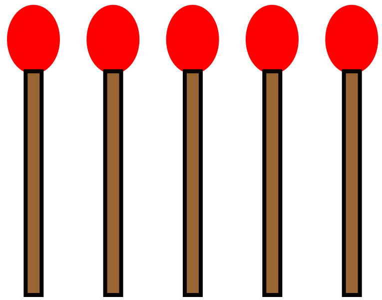


Take 1 or 2 at each turn
Goal: take the last match

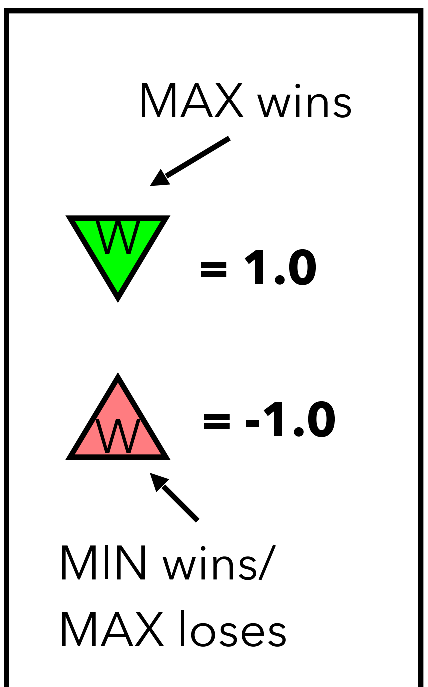


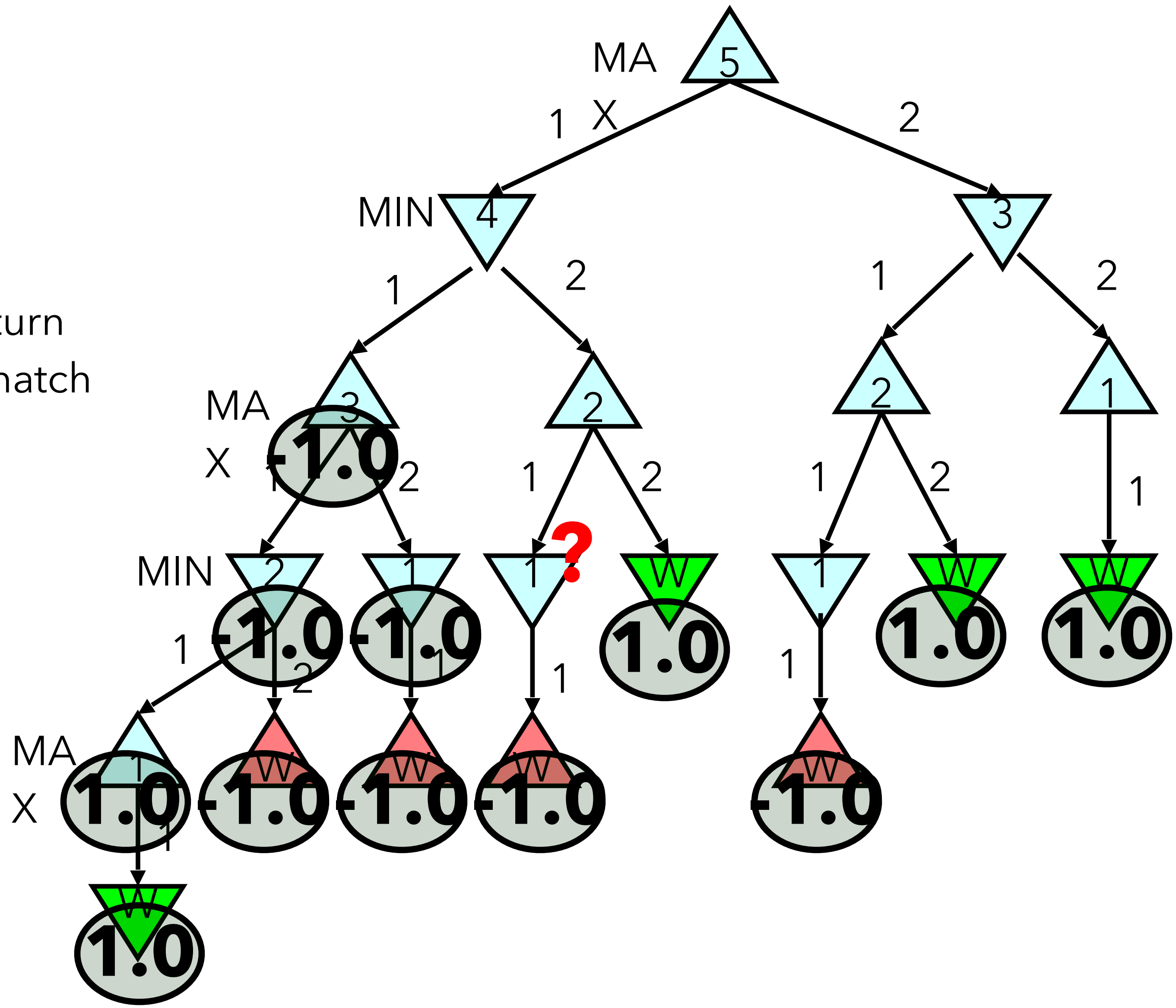


MINIMAX

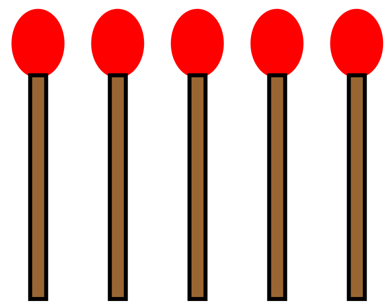


Take 1 or 2 at each turn
Goal: take the last match

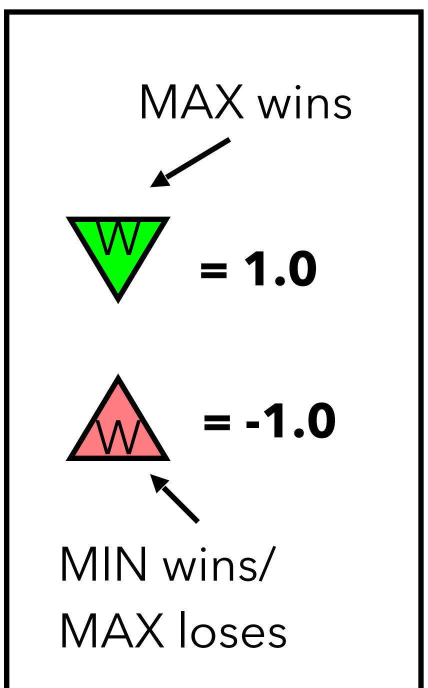


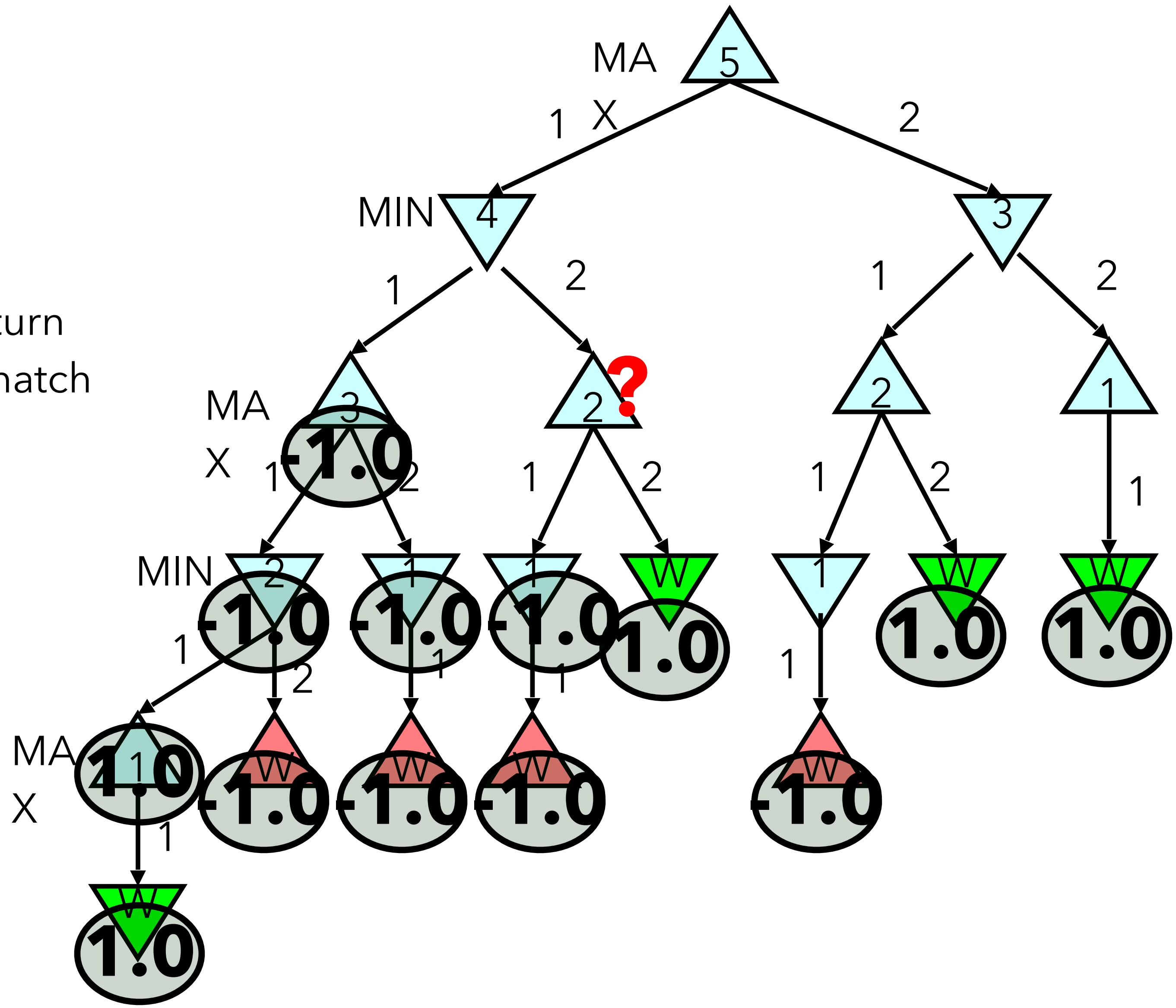


MINIMAX

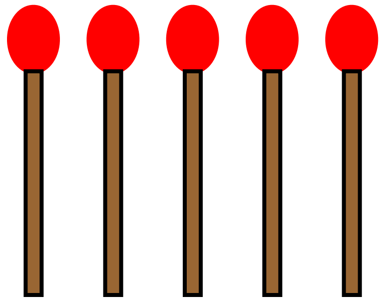


Take 1 or 2 at each turn
Goal: take the last match



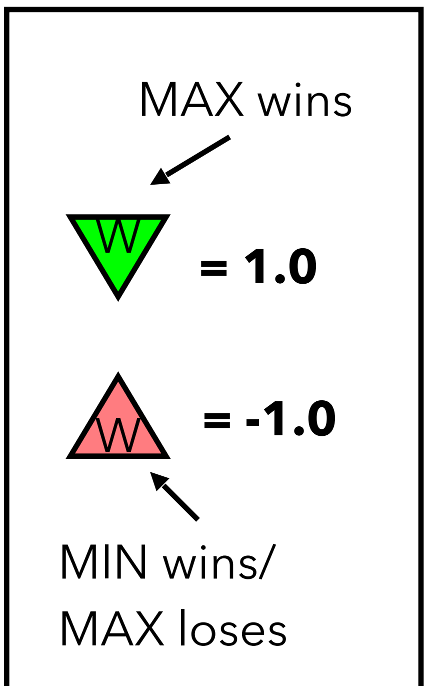


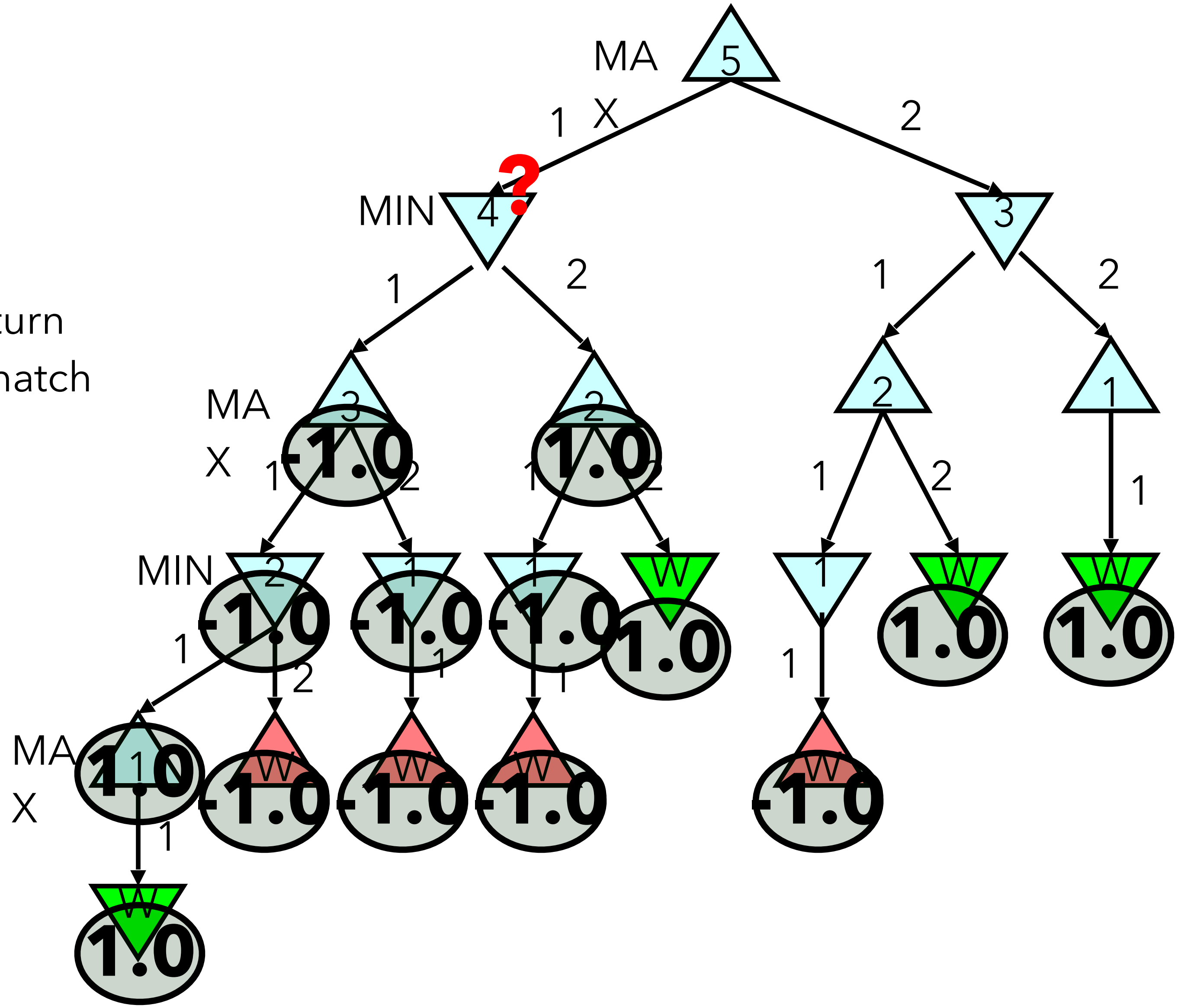
MINIMAX



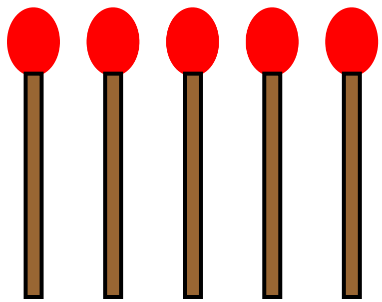
Take 1 or 2 at each turn

Goal: take the last match

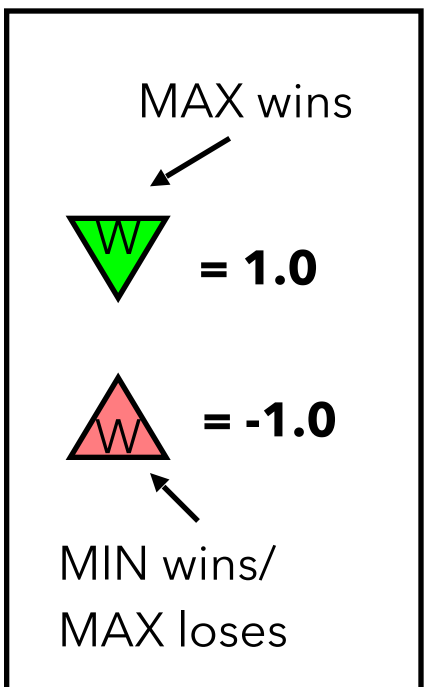


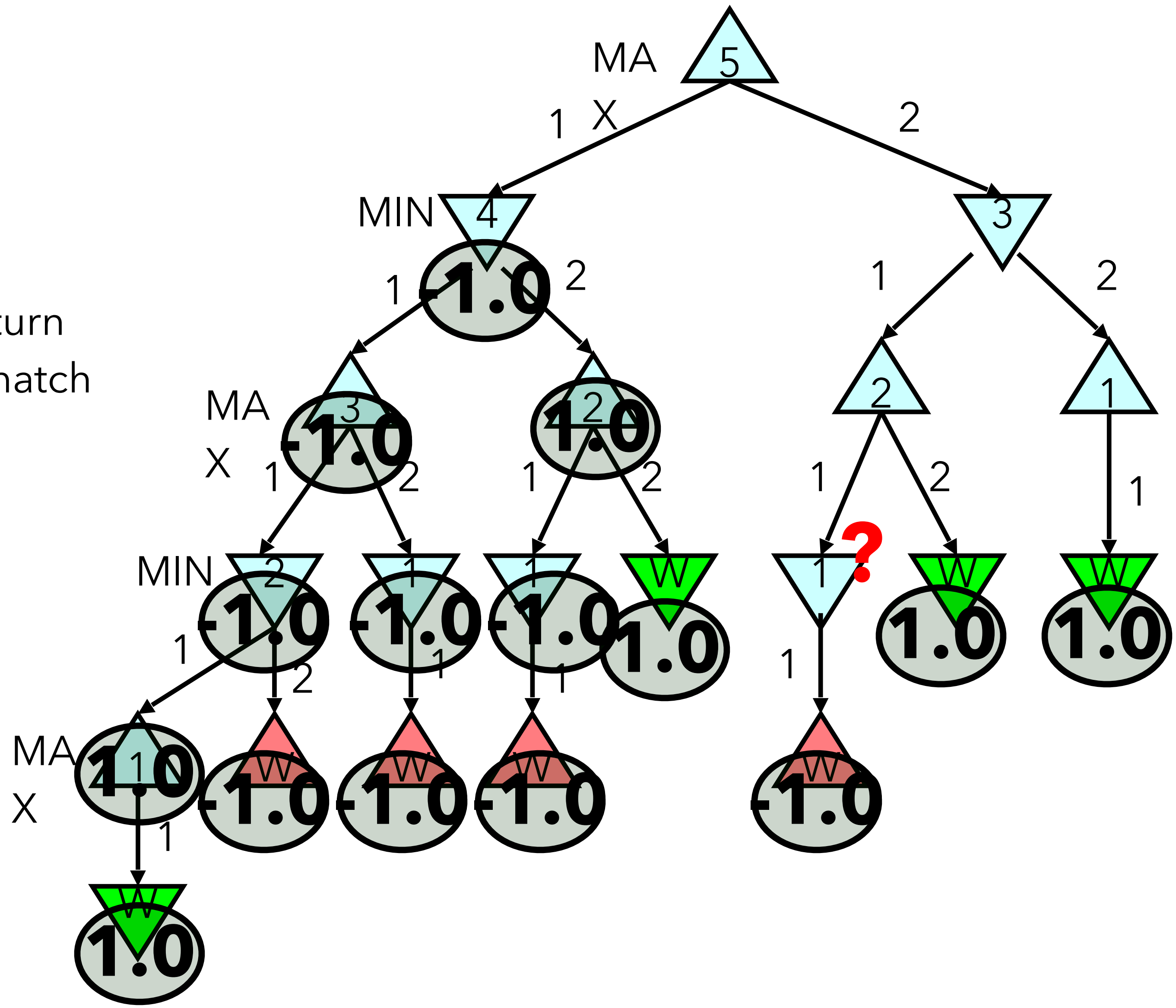


MINIMAX

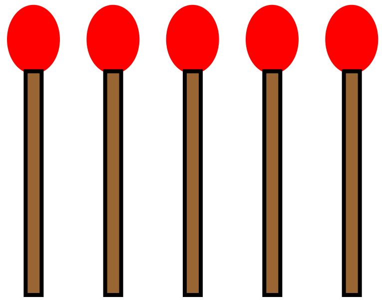


Take 1 or 2 at each turn
Goal: take the last match



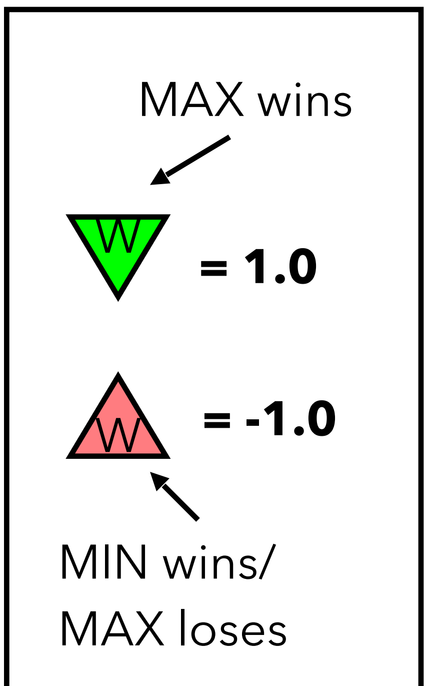


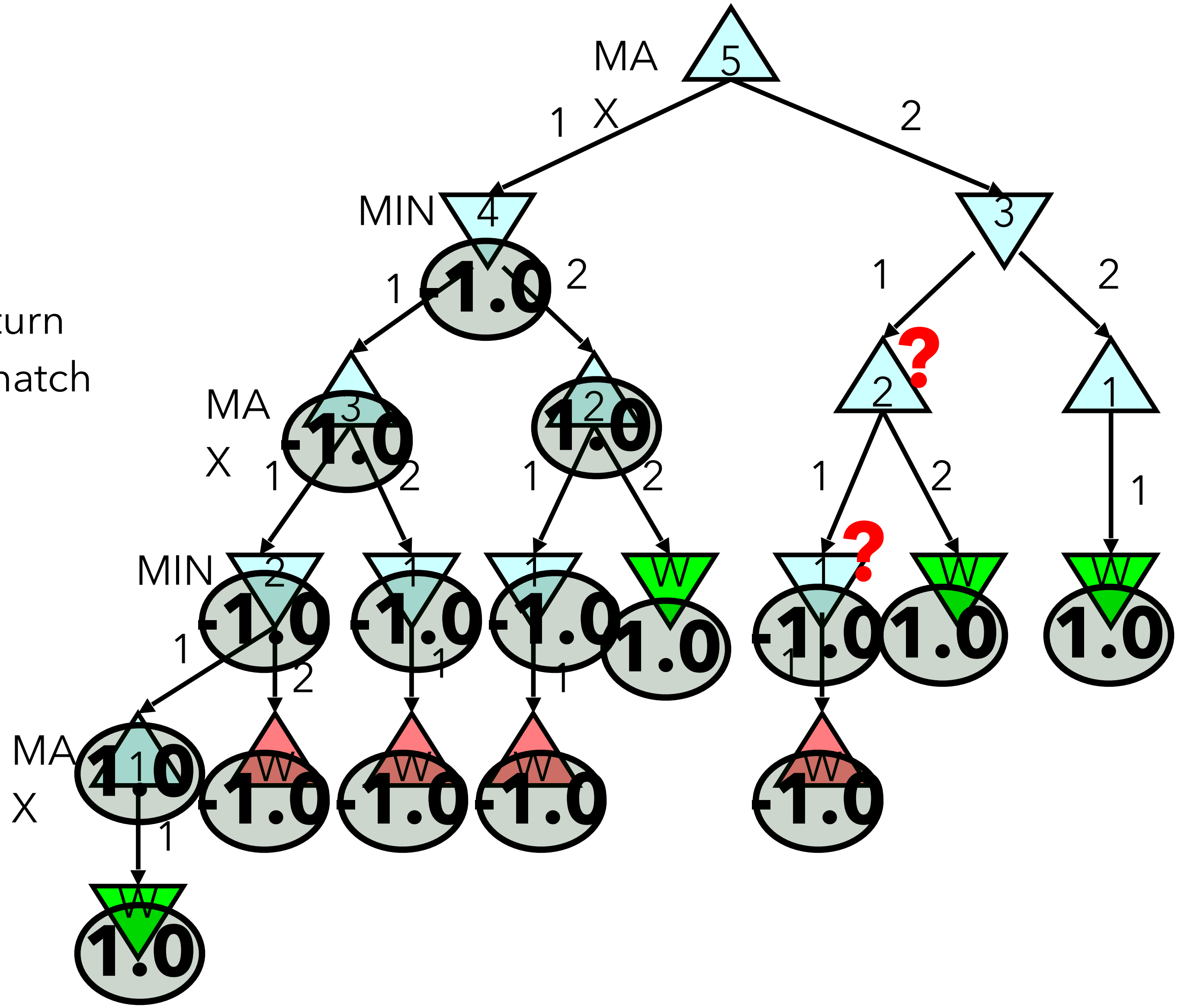
MINIMAX



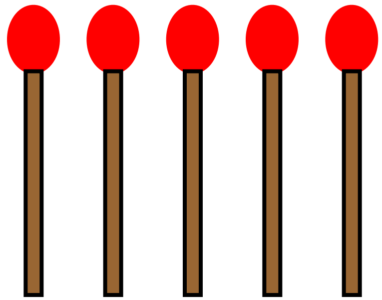
Take 1 or 2 at each turn

Goal: take the last match



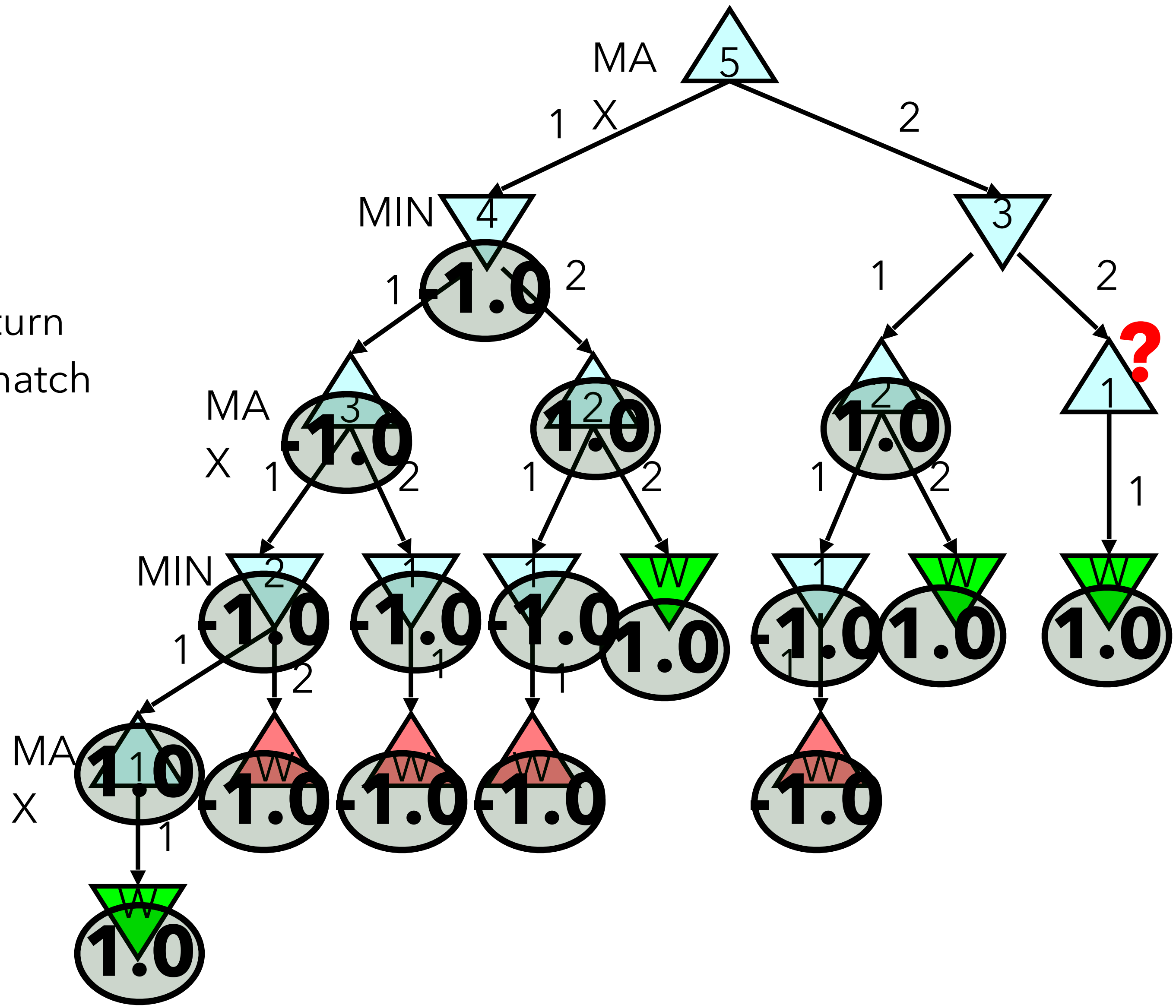


MINIMAX

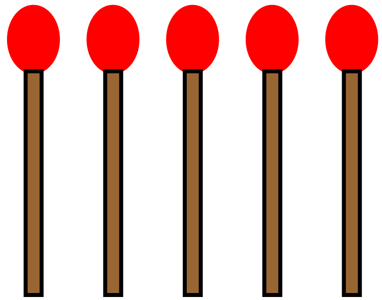


Take 1 or 2 at each turn

Goal: take the last match

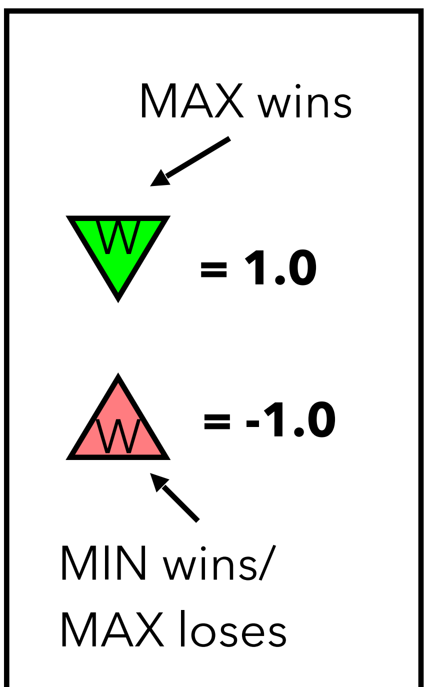


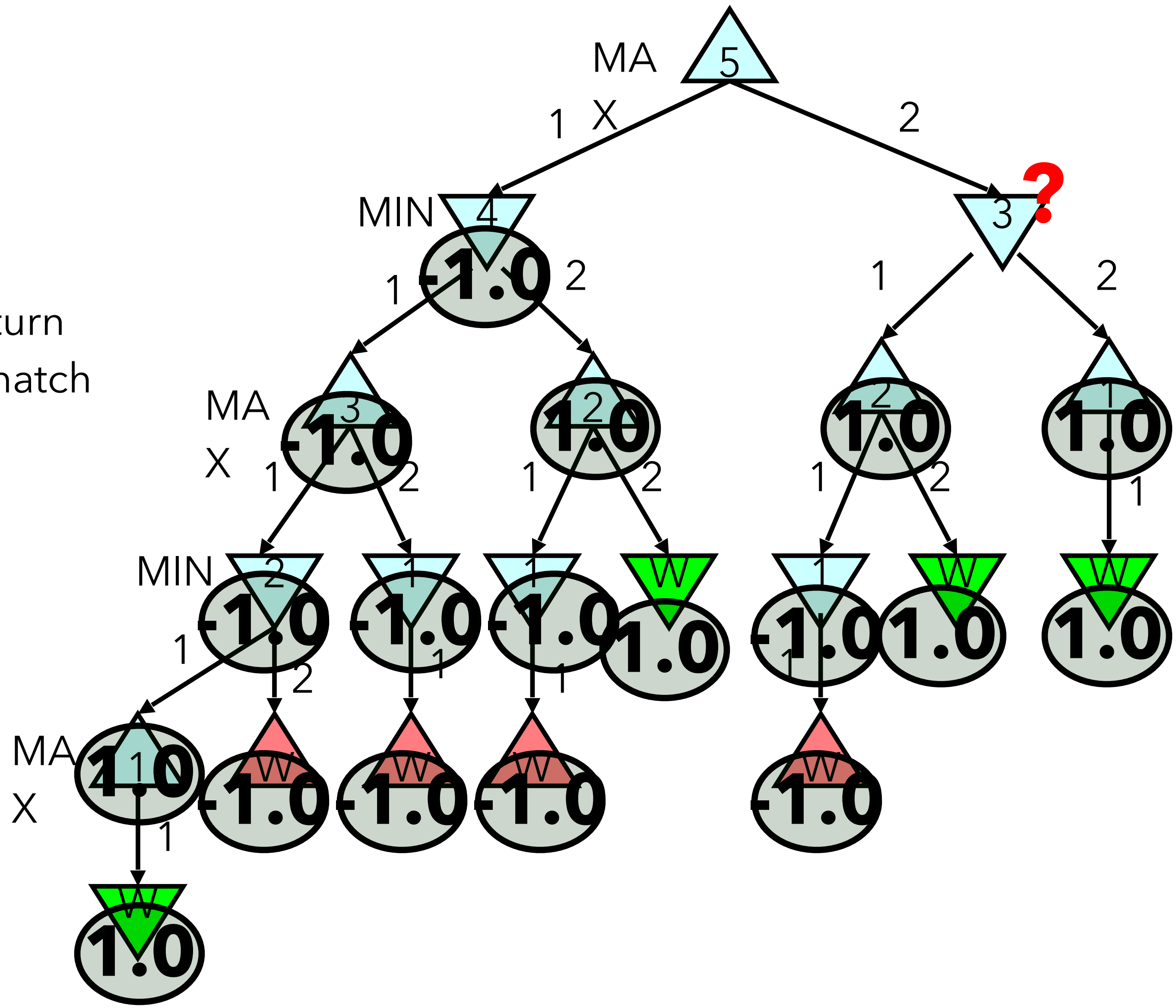
MINIMAX



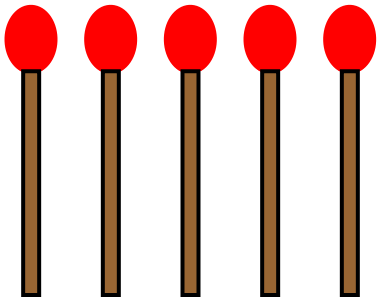
Take 1 or 2 at each turn

Goal: take the last match

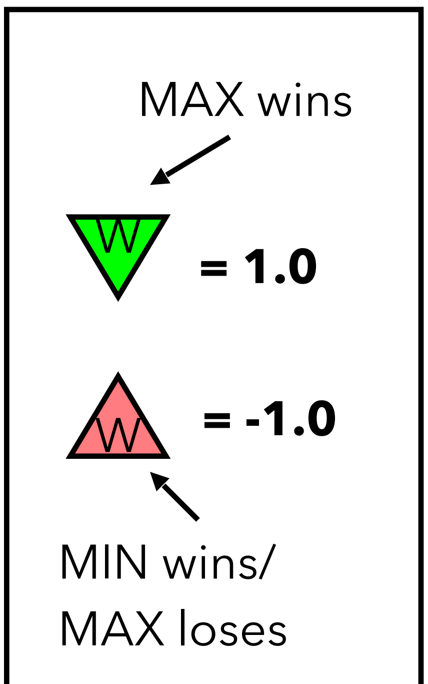




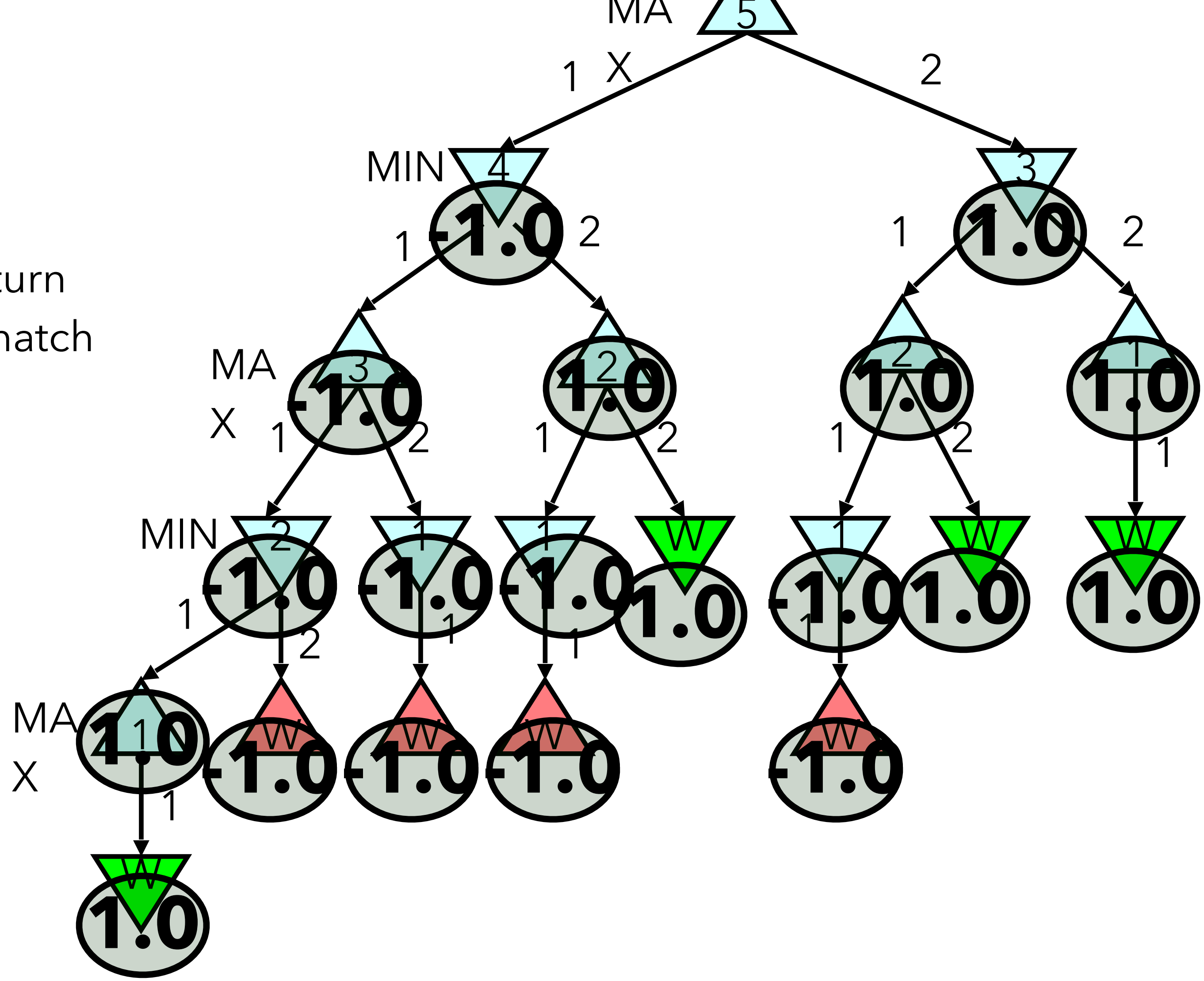
MINIMAX



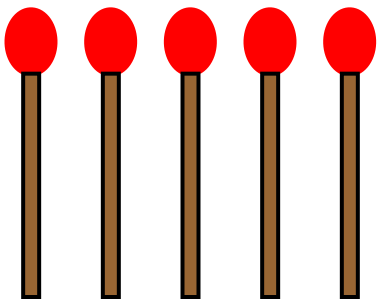
Take 1 or 2 at each turn
Goal: take the last match



Which move?



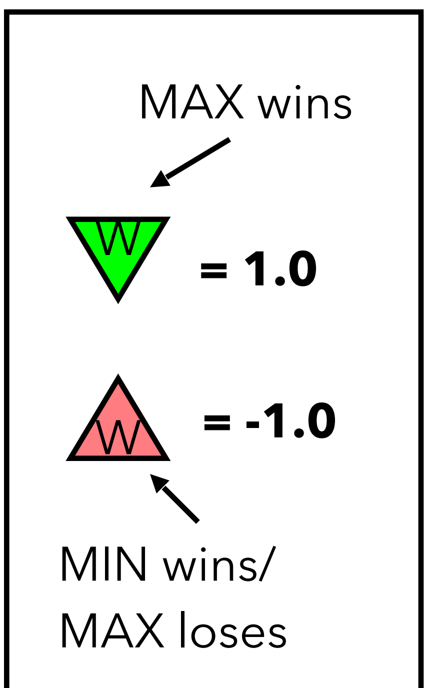
MINIMAX

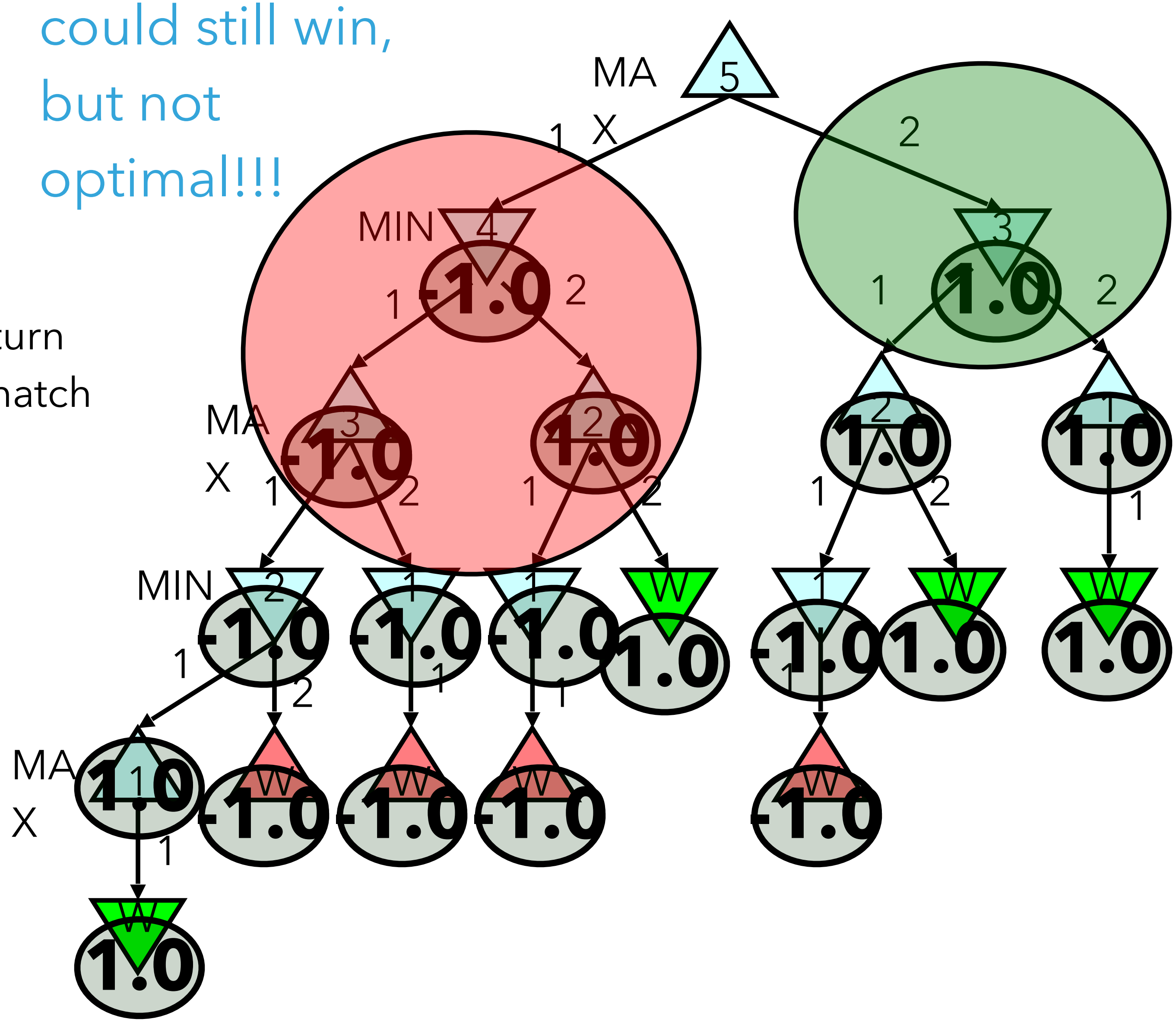


could still win, but not optimal!!!

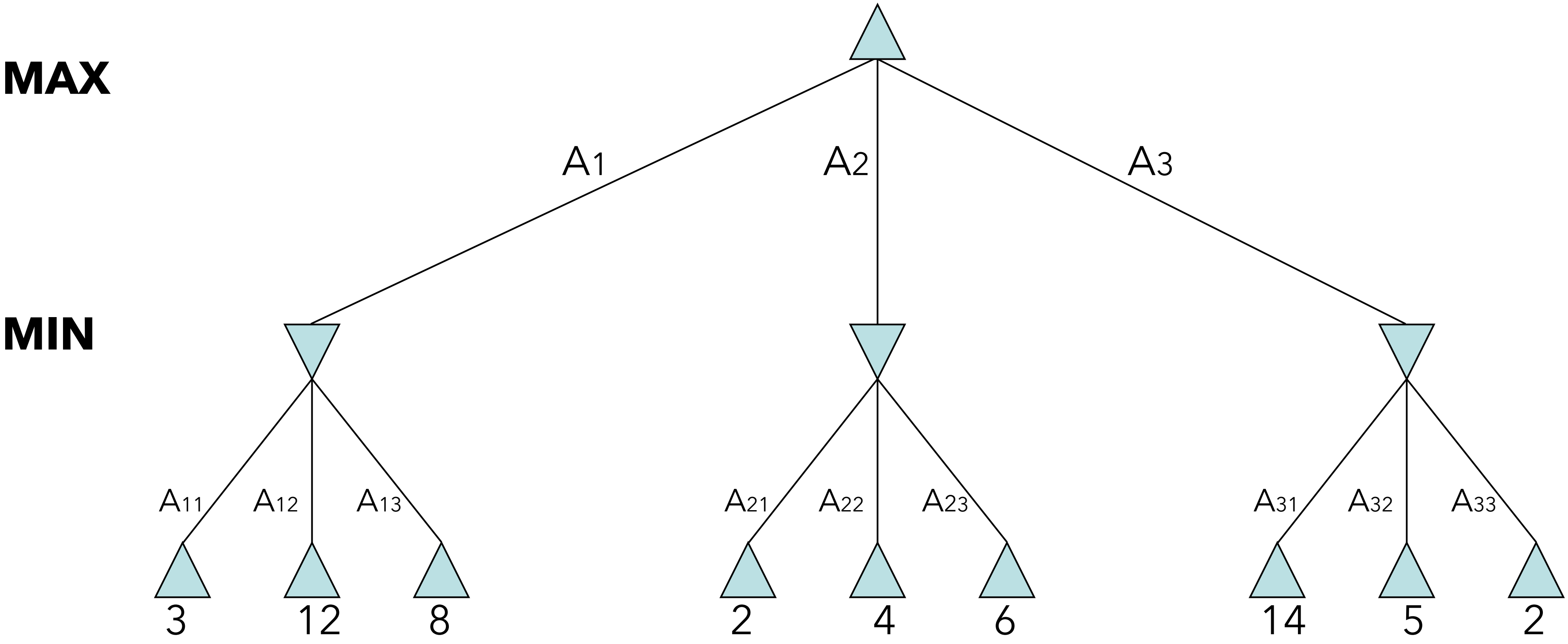
Take 1 or 2 at each turn

Goal: take the last match



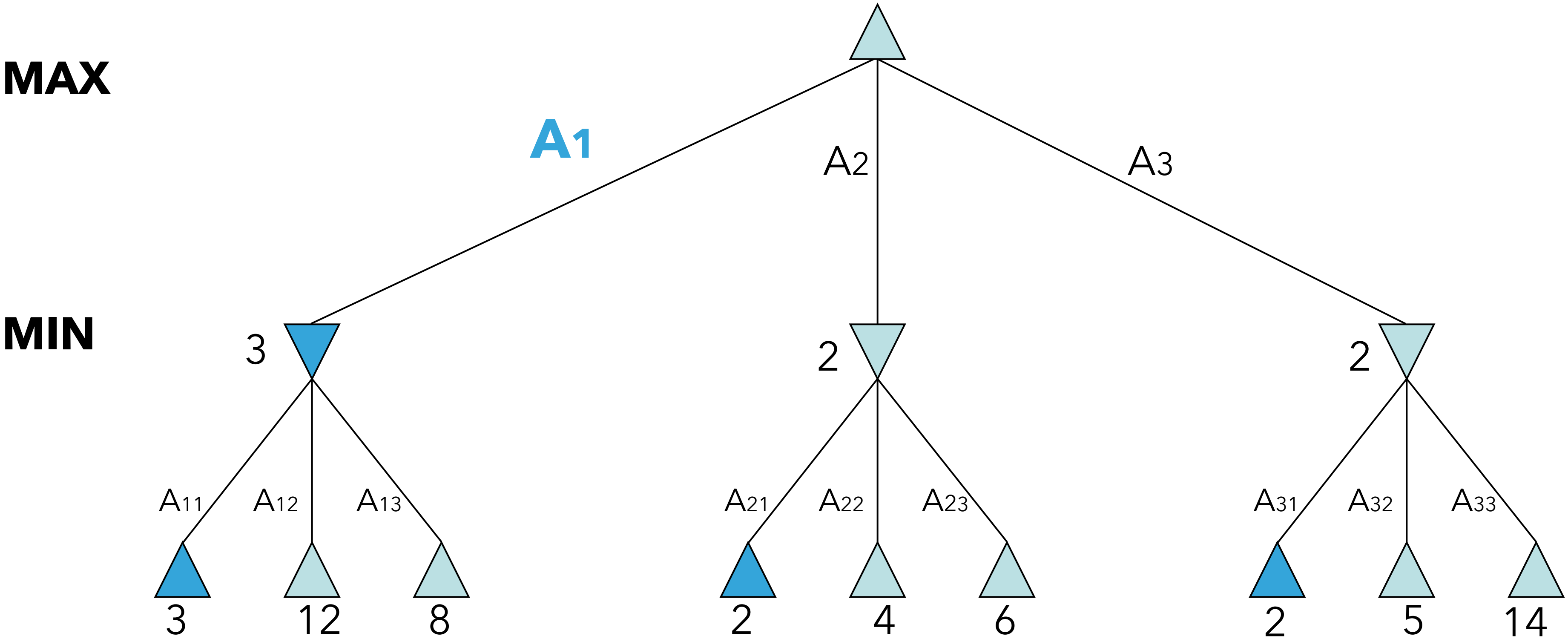


MINIMAX



Which move should be made: A_1, A_2 or A_3?

MINIMAX



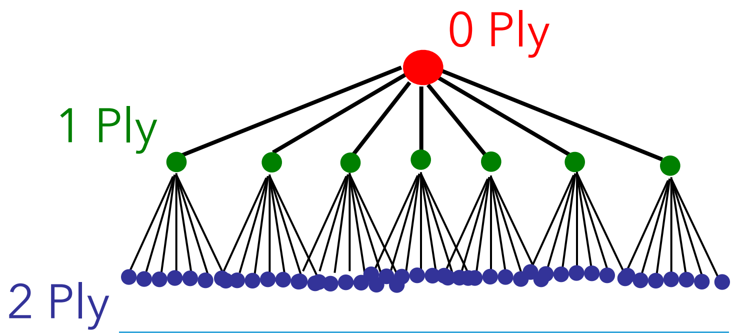
MAX
MIN
A1
A2
A3
3
2
2
A11
A12
A13
A21
A22
A23
A31
A32
A33
3
12
8
2
4
6
2
5
14

Properties of minimax

Minimax is optimal! Are we done?

Game state space

On average, there are ~35 possible moves that a chess player can make from any board configuration…

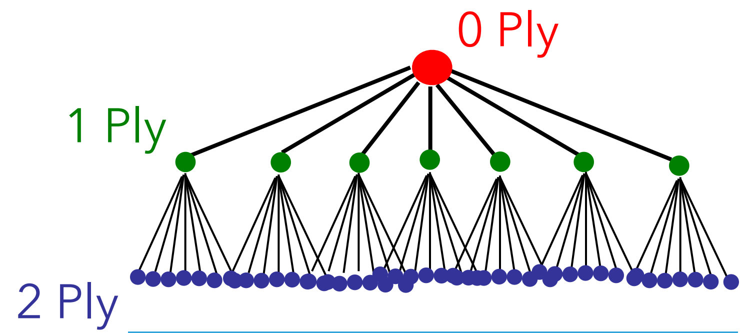


Branching Factor Estimates for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

Game state space

On average, there are ~35 possible moves that a chess player can make from any board configuration…



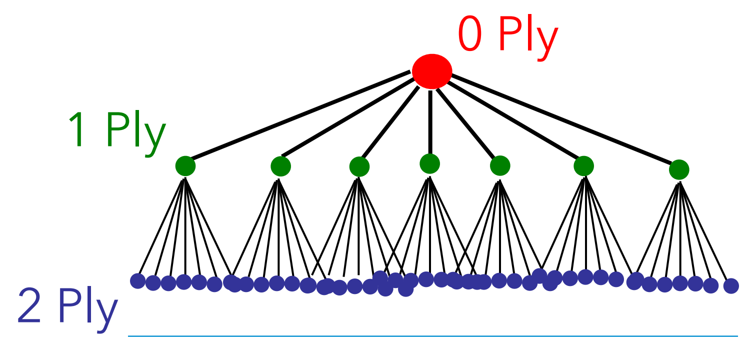
Branching Factor Estimates for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

Boundaries for qualitatively different games…

Game state space

On average, there are ~35 possible moves that a chess player can make from any board configuration…



Branching Factor Estimates for different two-player games

Game	Branching factor
Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

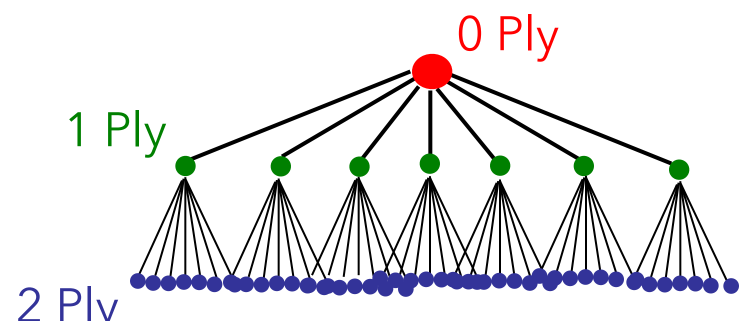
"solved" games

computer-dominated games

human-dominated games

Game state space

On average, there are ~35 possible moves that a chess player can make from any board configuration…



Branching Factor Estimates for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

"solved" games

computer-dominated games

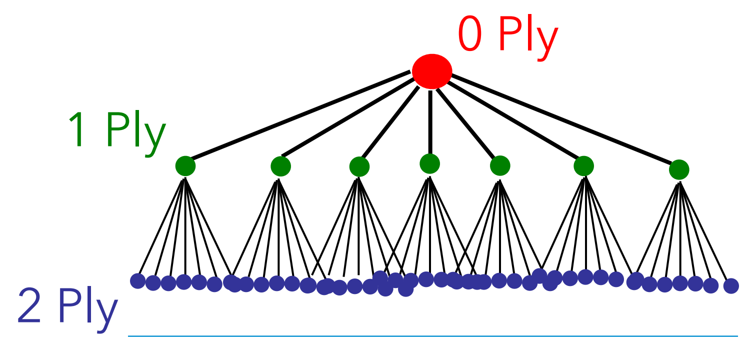
Is this true?

human-dominated games

Game state space

AlphaGo (created by Google), in April 2016 beat one of the best Go players:

http://www.nytimes.com/2016/04/05/science/google-alphago-artificial-intelligence.html



Branching Factor Estimates for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

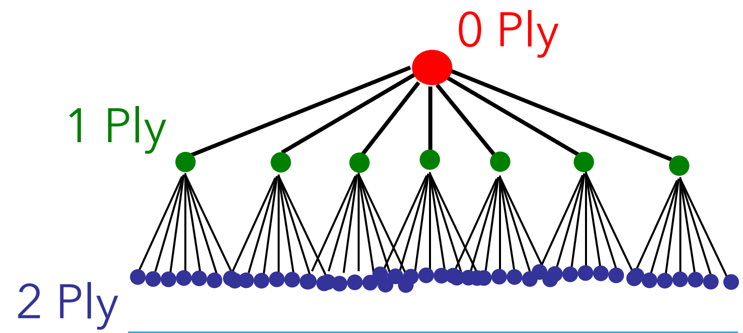
Alpha-Beta pruning

An optimal pruning strategy

- only prunes paths that are suboptimal (i.e. wouldn't be chosen by an optimal playing player).
- returns the ***same*** result as minimax, but faster.

Game state space

- Pruning helps get a bit deeper
- For many games, still can't search the entire tree
- Now what?



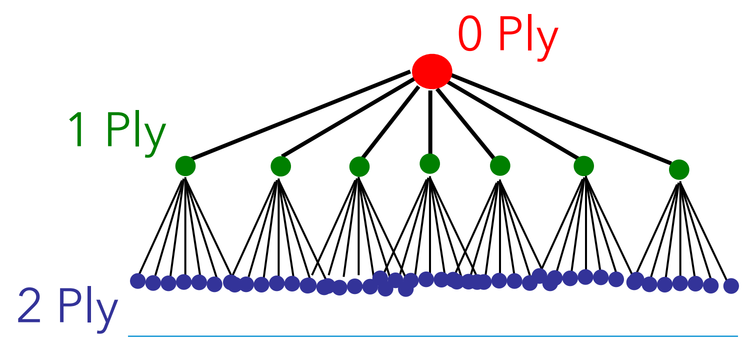
Branching Factor Estimates for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

computer-dominated games

Game state space

- Pruning helps get a bit deeper
- For many games, still can't search the entire tree
- Go as deep as you can:
  - estimate the score/quality of the state (called an evaluation function)
  - use that instead of the real score



Branching Factor Estimates for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

Tic Tac Toe evaluation functions

O	X	X
	O	

Ideas?

MINIMAX

Tic Tac Toe

Assume MAX is using “X”

EVAL(*state*) =

if state is win for MAX:

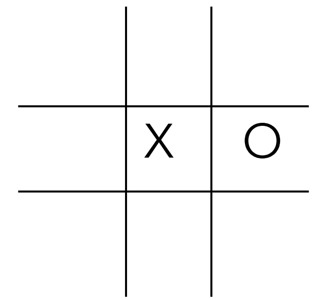
$+\infty$

if state is win for MIN:

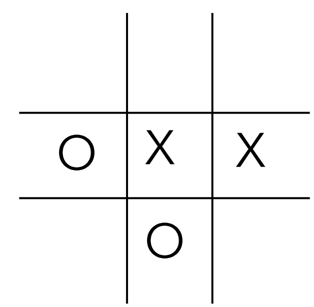
$-\infty$

else:

(number of rows, columns and diagonals available to MAX) -
(number of rows, columns and diagonals available to MIN)

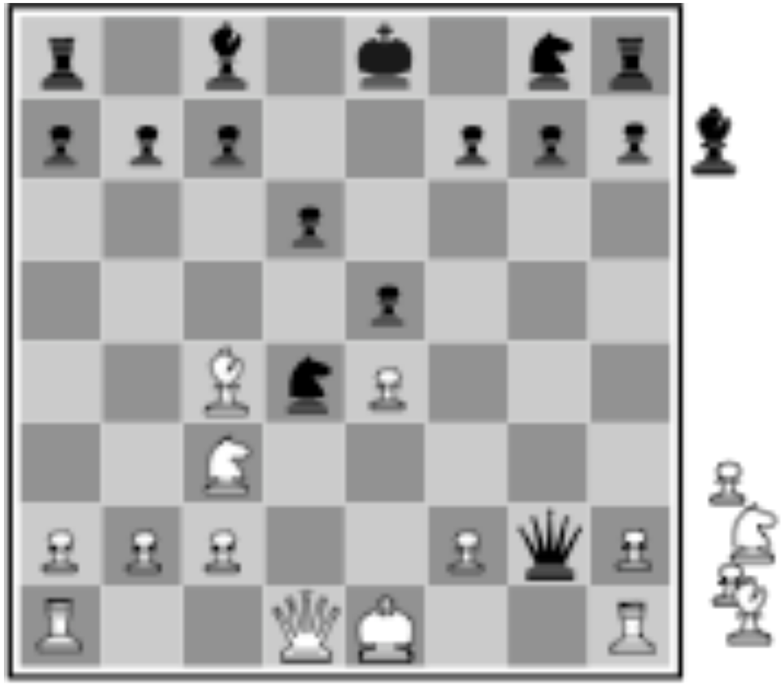


= 6 - 4 = 2



= 4 - 3 = 1

Chess evaluation functions

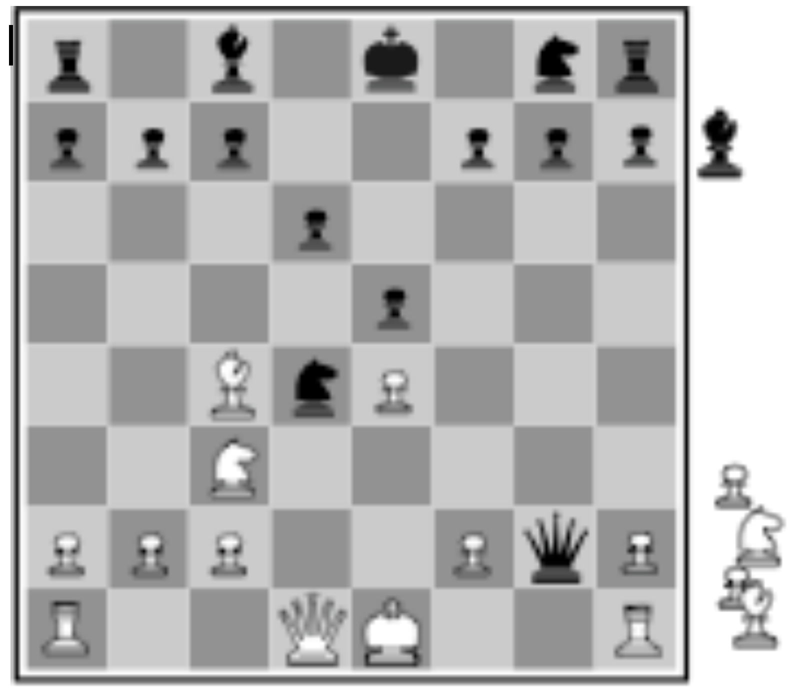


Ideas?

MINIMAX

Assume each piece has the following val

- pawn = 1;
- knight = 3;
- bishop = 3;
- rook = 5;
- queen = 9;



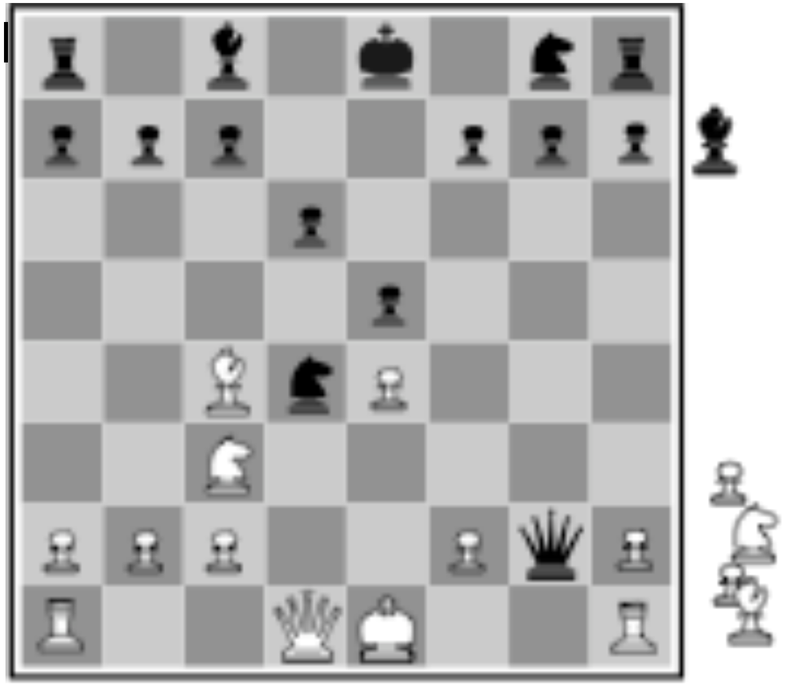
EVAL(*state*) =
sum of the value of white pieces –
sum of the value of black pieces

= 31 - 36 = -5

MINIMAX

Assume each piece has the following val

pawn	= 1;
knight	= 3;
bishop	= 3;
rook	= 5;
queen	= 9;

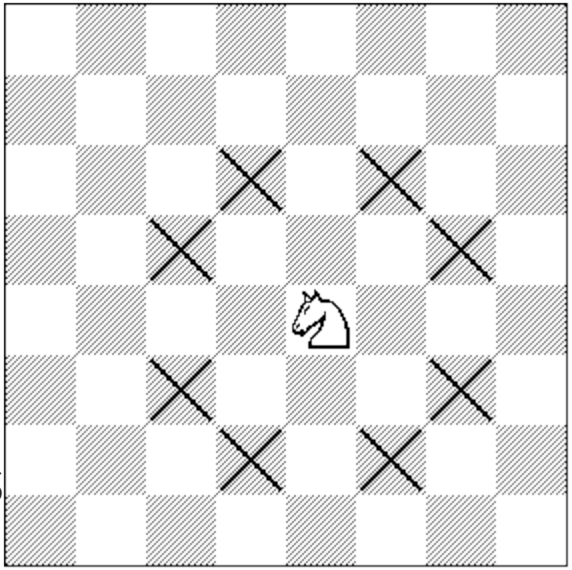


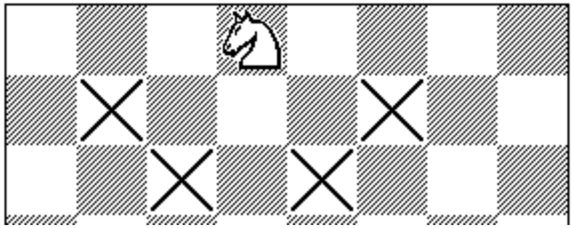
EVAL(*state*) =
sum of the value of white pieces –
sum of the value of black pieces

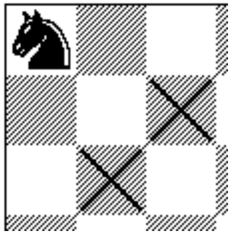
Any problems with this?

Chess EVAL

- Ignores actual positions!
- Actual heuristic functions are often a weighted combination of features







Chess EVAL

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + w_3 f_3(s) + \ldots$$

- f_1: number of pawns
- f_2: number of attacked knights
- f_3: 1 if king has knighted, 0 otherwise

A feature can be any numerical information about the board

- as general as the number of pawns
- to specific board configurations

Deep Blue: 8000 features!

history/end-game tables

History

- keep track of the quality of moves from previous games
- use these instead of search

end-game tables

- do a reverse search of certain game configurations, for example all board configurations with king, rook and king
- tells you what to do in ***any*** configuration meeting this criterion
- if you ever see one of these during search, you lookup exactly what to do

end-game tables

- Devastatingly good
- Allows much deeper branching
  - for example, if the end-game table encodes a 20-move finish and we can search up to 14
  - can search up to depth 34
- Stiller (1991) explored all end-games with 6 pieces
  - one case check-mate required 223 moves!
  - https://www.nytimes.com/1991/10/30/us/computer-is-pushed-to-edge-to-solve-old-chess-problem.html
- Traditional rules of chess require a capture or pawn move within 50 or it’s a stalemate

Opening moves

- At the very beginning, we're the farthest possible from any goal state
- People are good with opening moves
- Tons of books, etc. on opening moves
- Most chess programs use a database of opening moves rather than search

Nim

- K piles of coins
- On your turn you must take one or more coins from one pile
- Player that takes the last coin wins
- Example: https://www.goobix.com/games/nim/

Resources

- practice_midterm_2.py

Homework

- Assignment 10 (cont'd)