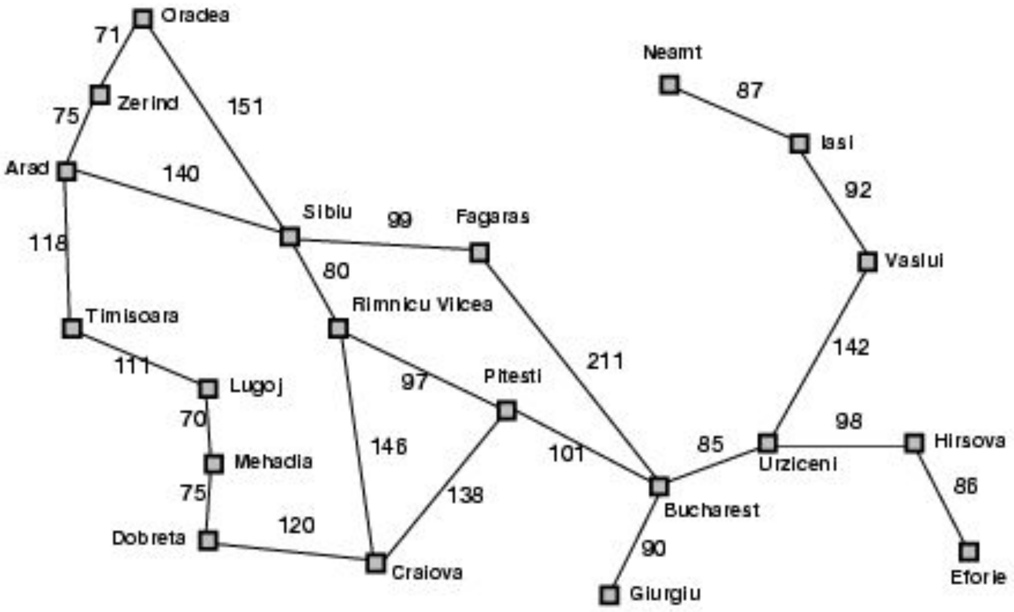
Adversarial Search

willie

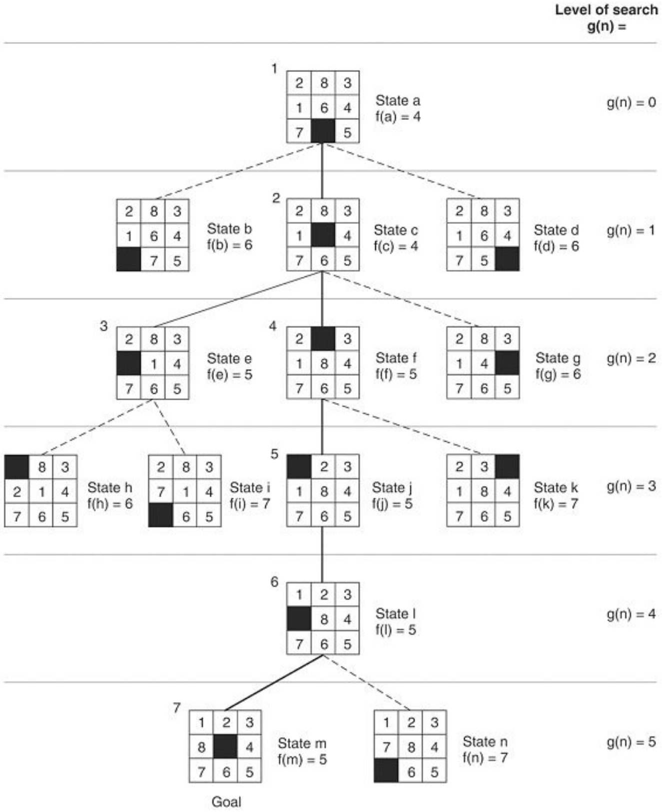
Solving problems with search



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Informed search



For this one example:

A*	7 nodes
BFS	46 nodes
DFS	31 nodes

h2 dominates h1

Averaging over 100 instances of 8-puzzle, at d=12, nodes examined:

A*

h1	227
h2	73

IDS 3,644,404

**EXAM QUESTION?!?!

Algebra Problems

Solve an algebra problem such as $x^2+3x = 0$

- Initial state
- State space
- Successor function
- Goal
- Heuristics

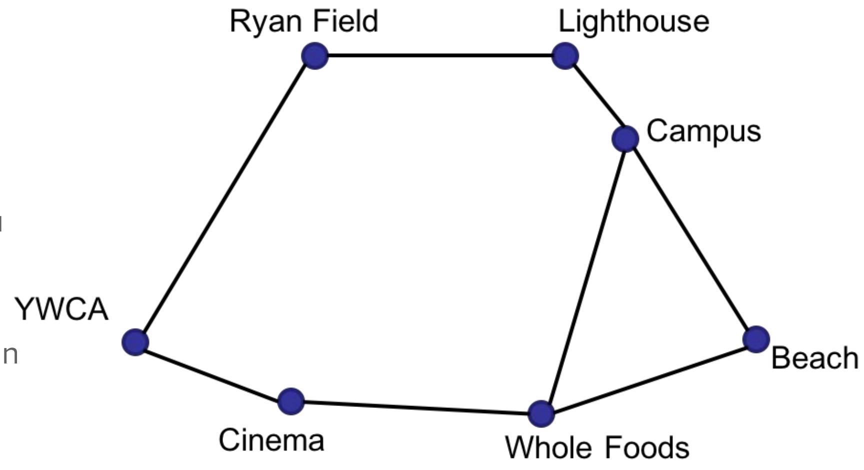
BigByte

Campus —> Cinema?
Evaluation function takes in heuristic
g(n) = how long it takes you to take the roads

You work at a food delivery start-up called BigByte. The company consists of two people, you and Willie, both alumni of Nerdwestern University located in Heavanston. Of the two, you take care of the software engineering and Willie handles the rest.

BigByte's target client restaurants and ordering customers are all based in Heavanston, which has a handful of landmarks and a heavily fluctuating traffic pattern. To ensure that your delivery fleet works efficiently, Willie asked you to come up with a route planning system.

All major map services are expensive and your start-up begins its services in a week, so you'll have to come up with something yourself and do it fast. Fortunately, Restaurants and Heavanston residents all live close to landmarks, and you only need to tell delivery drivers how to best reach one landmark from another. You have a printed map of all Heavanston roads and data from a traffic survey maintained by University Archive. For you, that means you only need to write one more piece of software before the launch: an implementation of the A* algorithm that finds the best way to travel between any pair of landmarks in town.

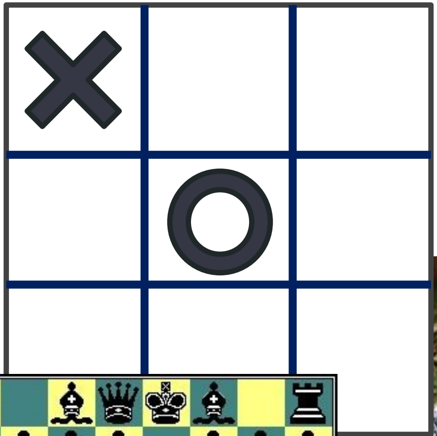


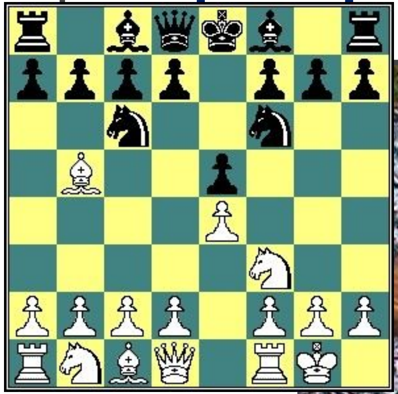
Moving on

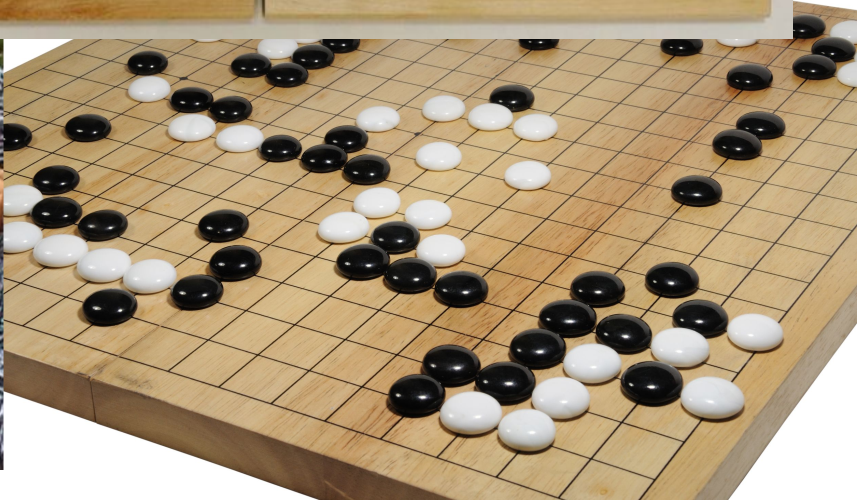
“Within ten years a digital computer will be the world's chess champion”

Simon & Newell, 1958

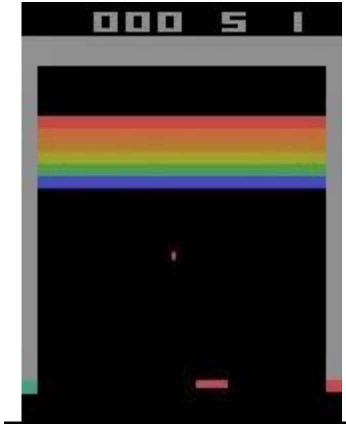
Opponent doesn’t want you to win

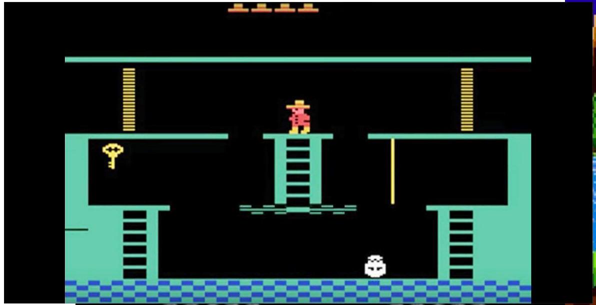






Video games too





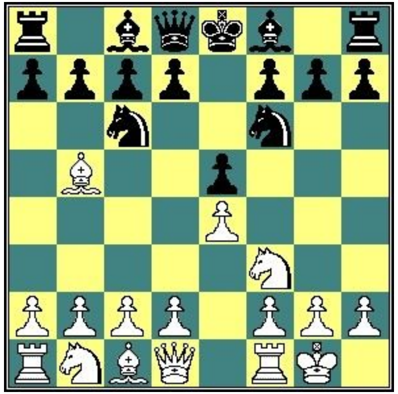
But let’s just focus on simple, turn-taking board games

Strategic thinking $\stackrel{?}{=}$ intelligence

Two-player games have been a key focus of AI as long as computers have been around…
Humans and computers have different relative strengths in these games:

humans

good at evaluating the strength of a board for a player



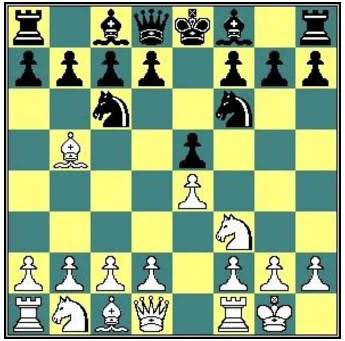
computers

good at looking ahead in the game to find winning combinations of moves

Human Intelligence vs Computer Intelligence:
Evaluation function
General evaluation in humans vs range of options in computers

How humans play games...

An experiment (by deGroot) was performed in which chess positions were shown to novice and expert players...



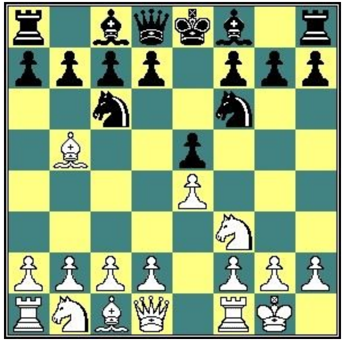
- experts could reconstruct these perfectly

- novice players did far worse...

Humans study/look for patterns (but not a sequence of events)

How humans play games...

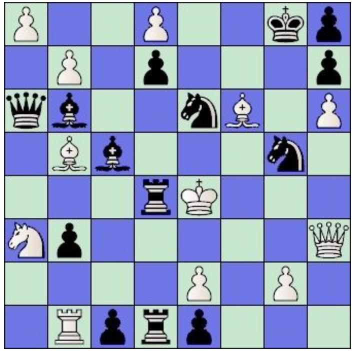
An experiment (by deGroot) was performed in which chess positions were shown to novice and expert players...



- experts could reconstruct these perfectly

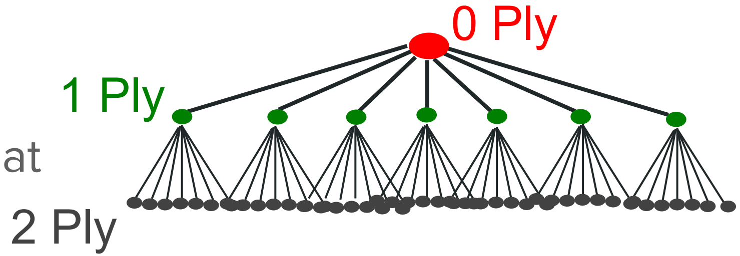
- novice players did far worse...

Random chess positions (not legal ones) were then shown to the two groups



- experts and novices did just as badly at reconstructing them!

Games' Branching Factors



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

Hydra - successor to Deep Blue

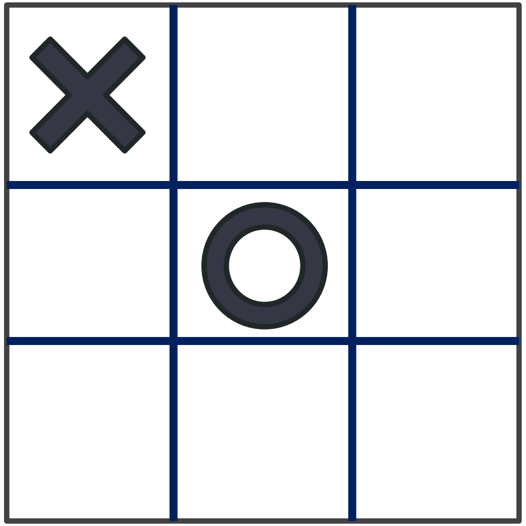
Up to **18 ply**

Branching Factor Estimates
for different two-player games

Tic-tac-toe	4
Connect Four	4
Checkers	2.8
Othello	10
Chess	35
Go	250

ex. 100 ply = 250^100

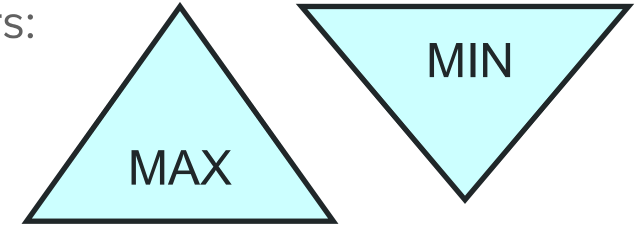
How computers play games



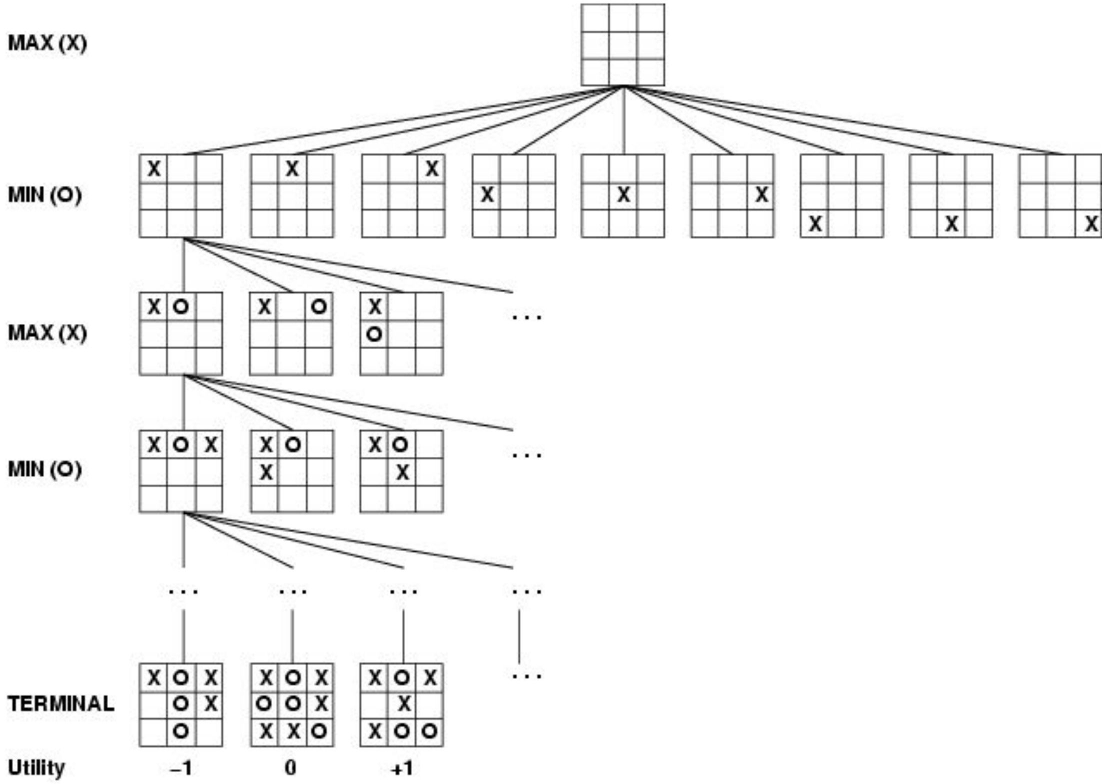
Define the problem:

- State: state of the board, whose turn
- Successor function: given a player's move, returns a new state
- Goal test: Is game over? Did one player win? Is there a draw?
- **Utility function: Payoff (+1 winning, -1 losing, 0 draw)**

2 players:



How computers play games...

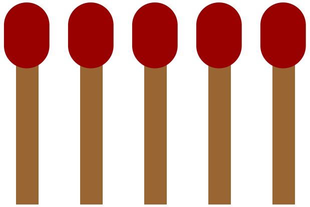


Optimal Strategy

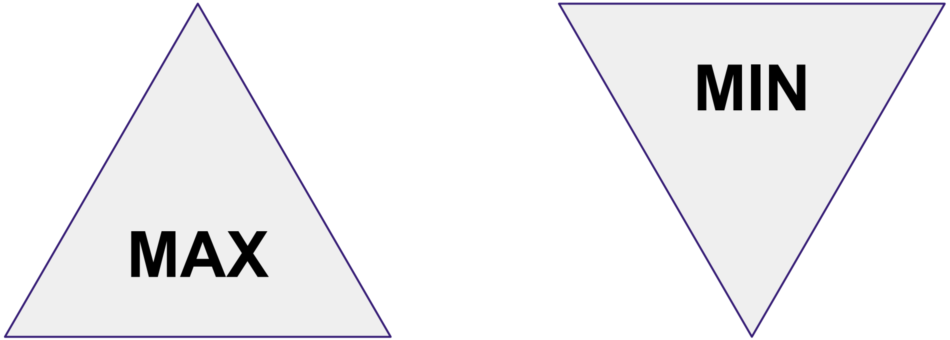
An **Optimal Strategy** is one that is as least as good as any other, no matter what the opponent does

- If there's a way to force the win, it will
- Will only lose if there's no other option

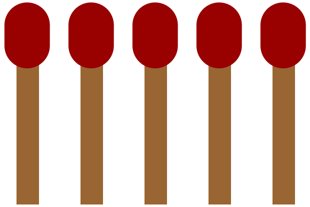
Baby Nim



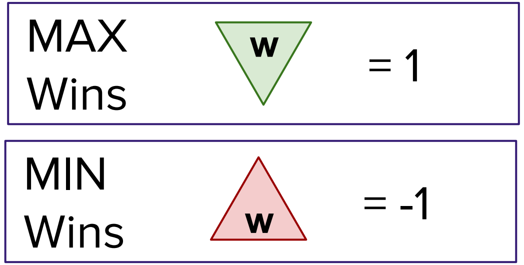
Take 1 or 2 at each turn
Goal: take the last match

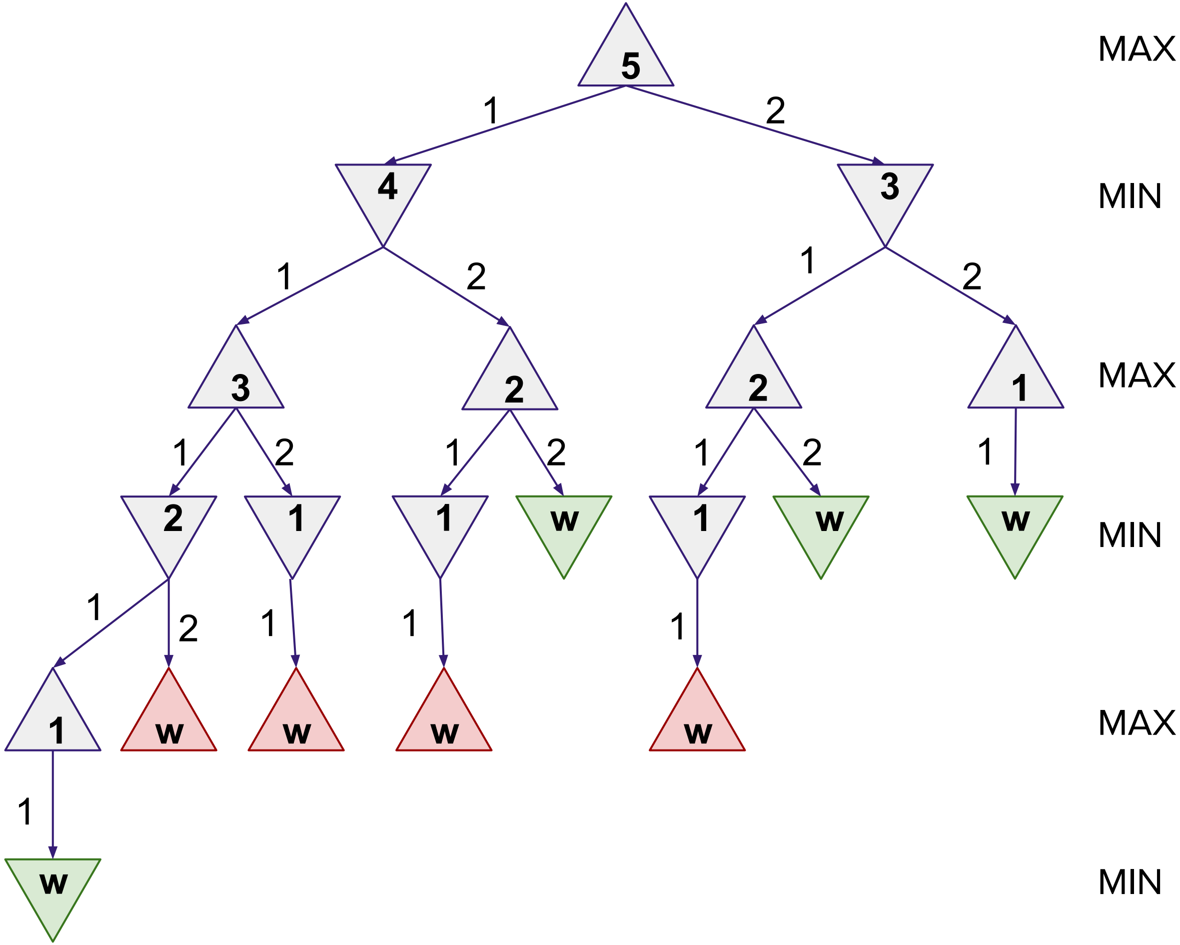


Baby Nim

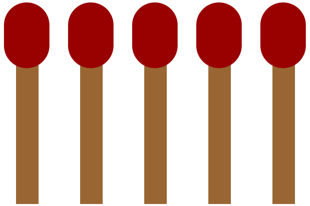


Take 1 or 2 at each turn
Goal: take the last match

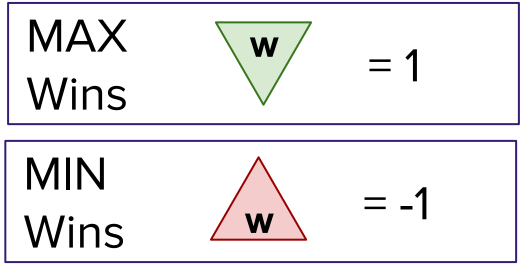


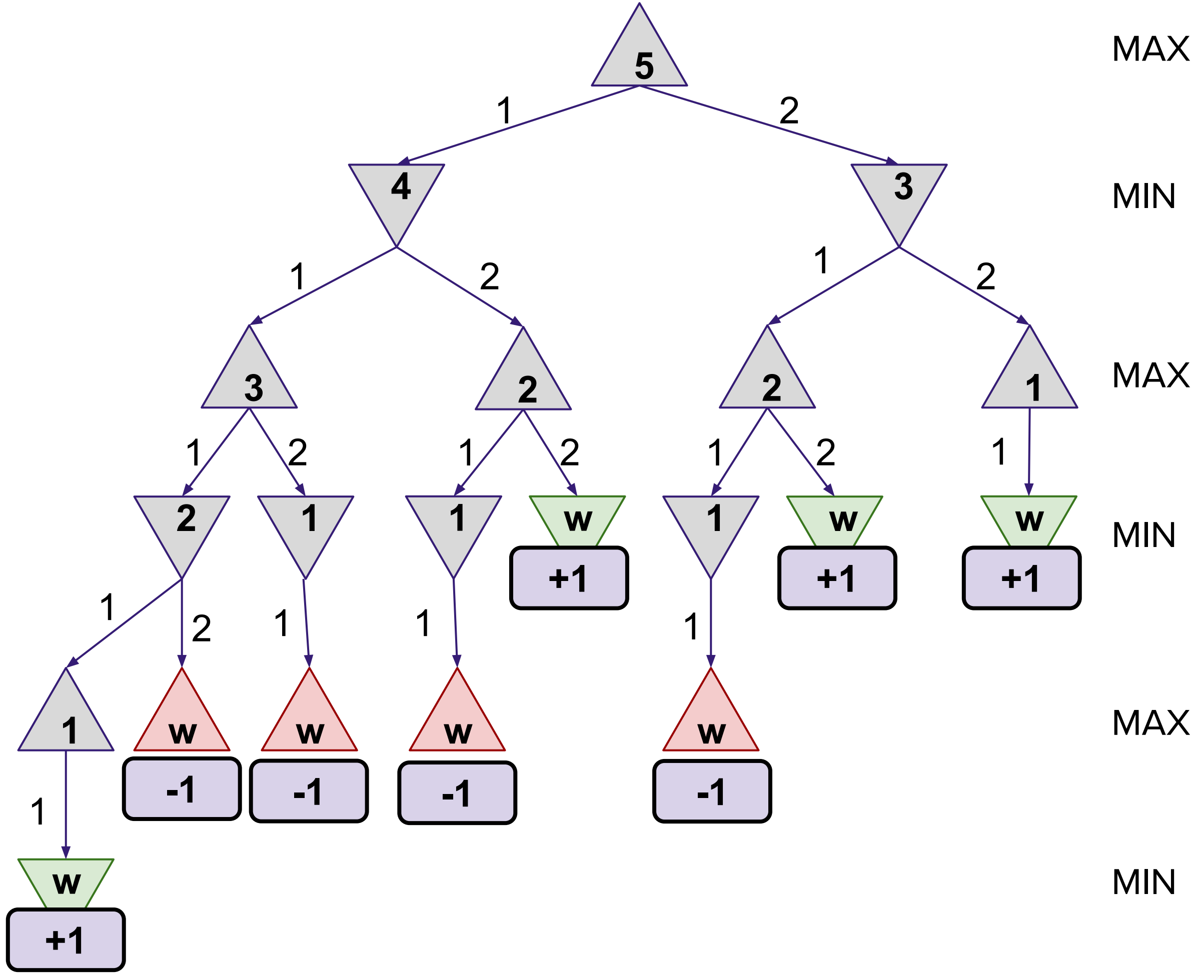


Baby Nim

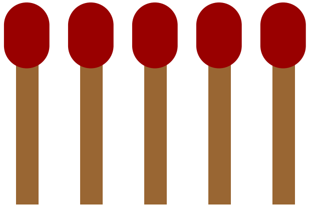


Take 1 or 2 at each turn
Goal: take the last match

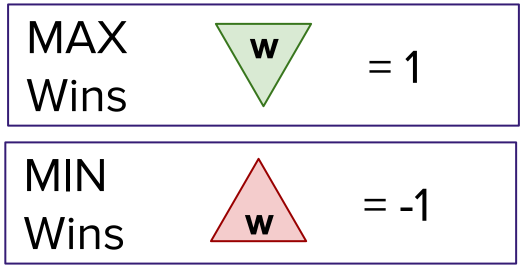


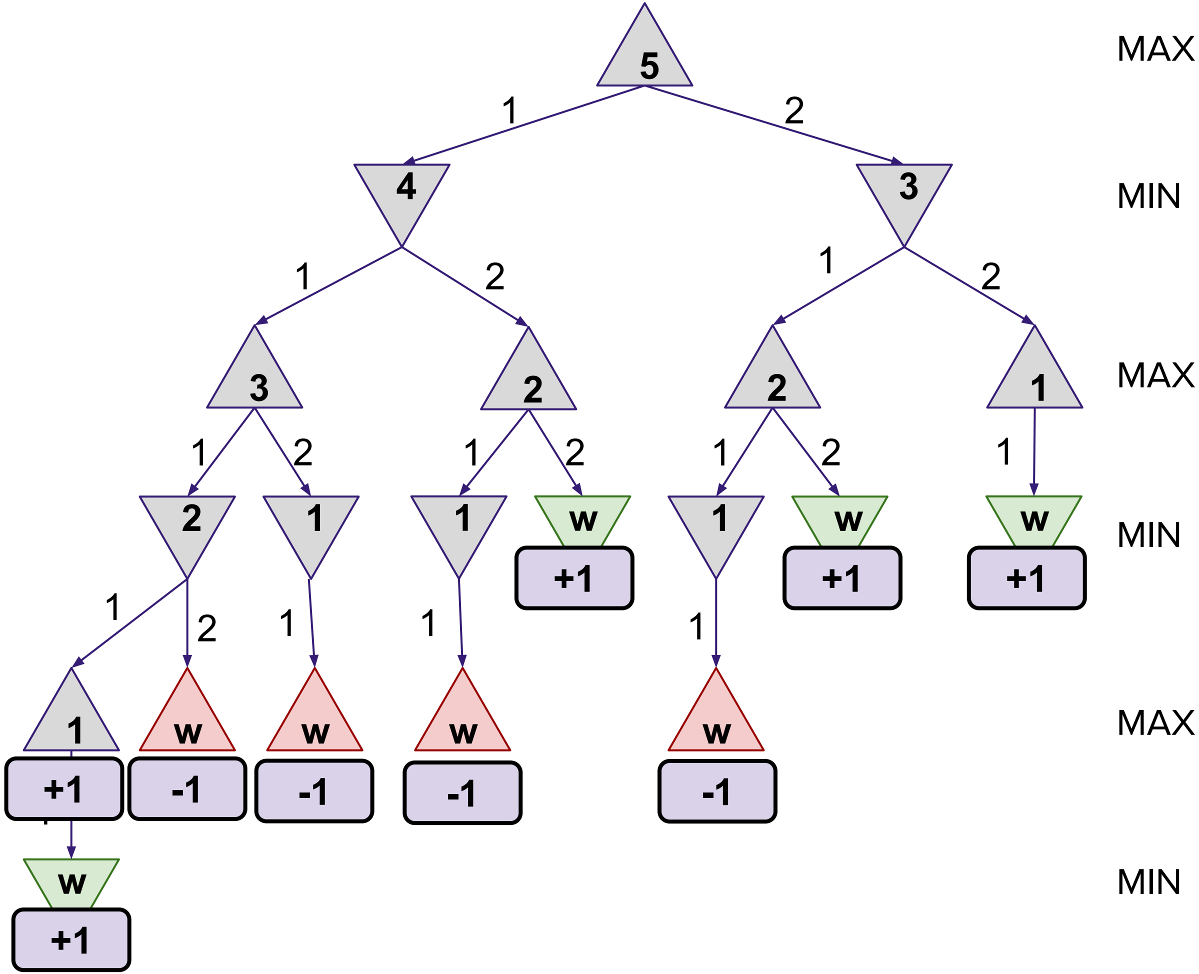


Baby Nim

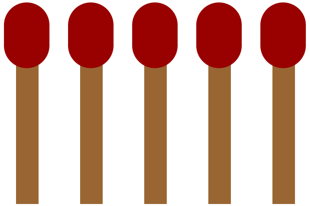


Take 1 or 2 at each turn
Goal: take the last match

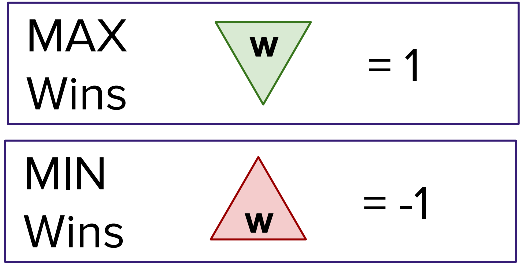


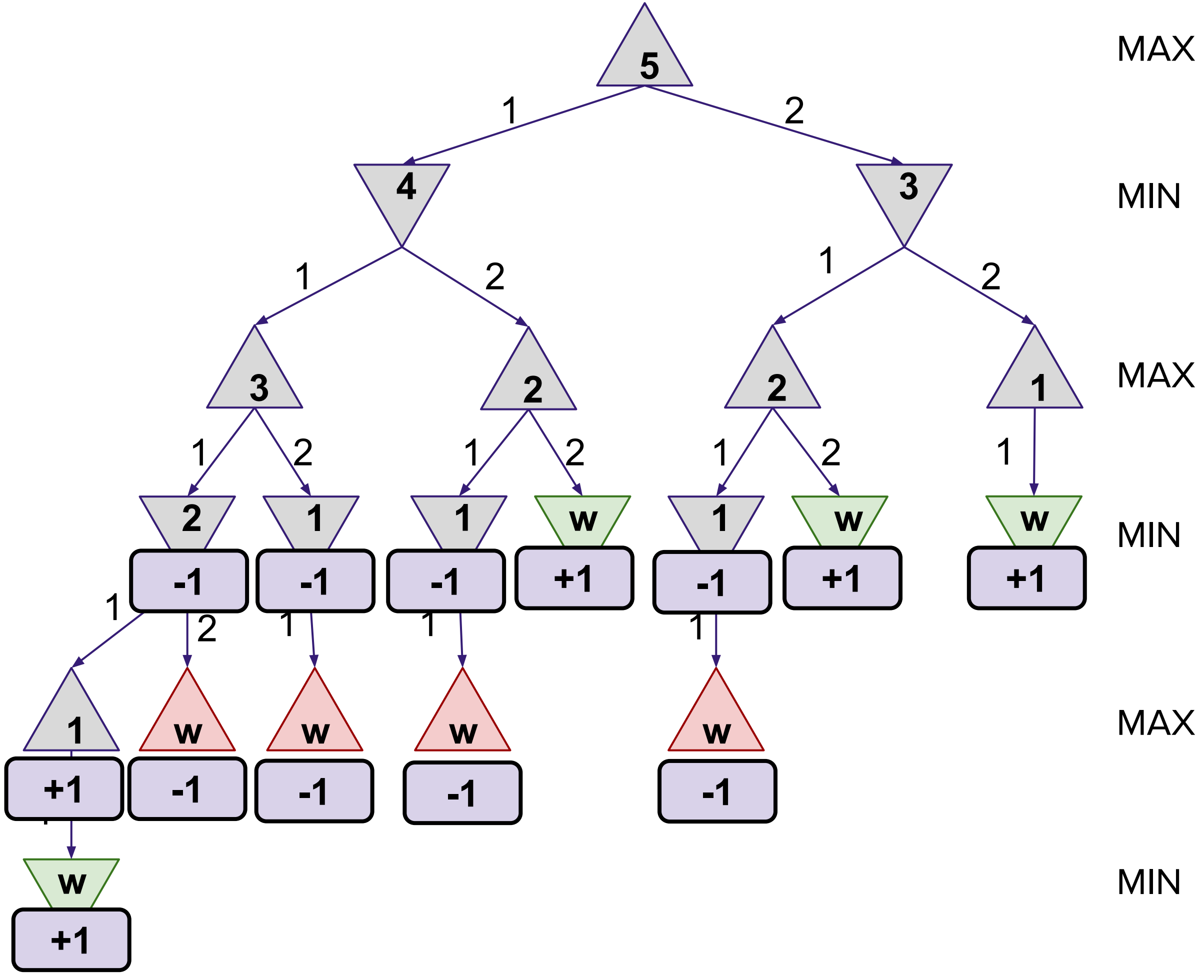


Baby Nim

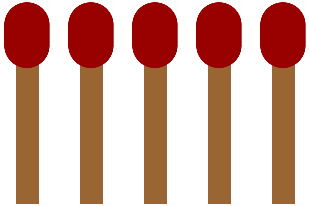


Take 1 or 2 at each turn
Goal: take the last match

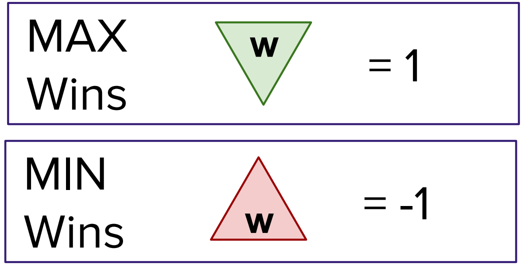


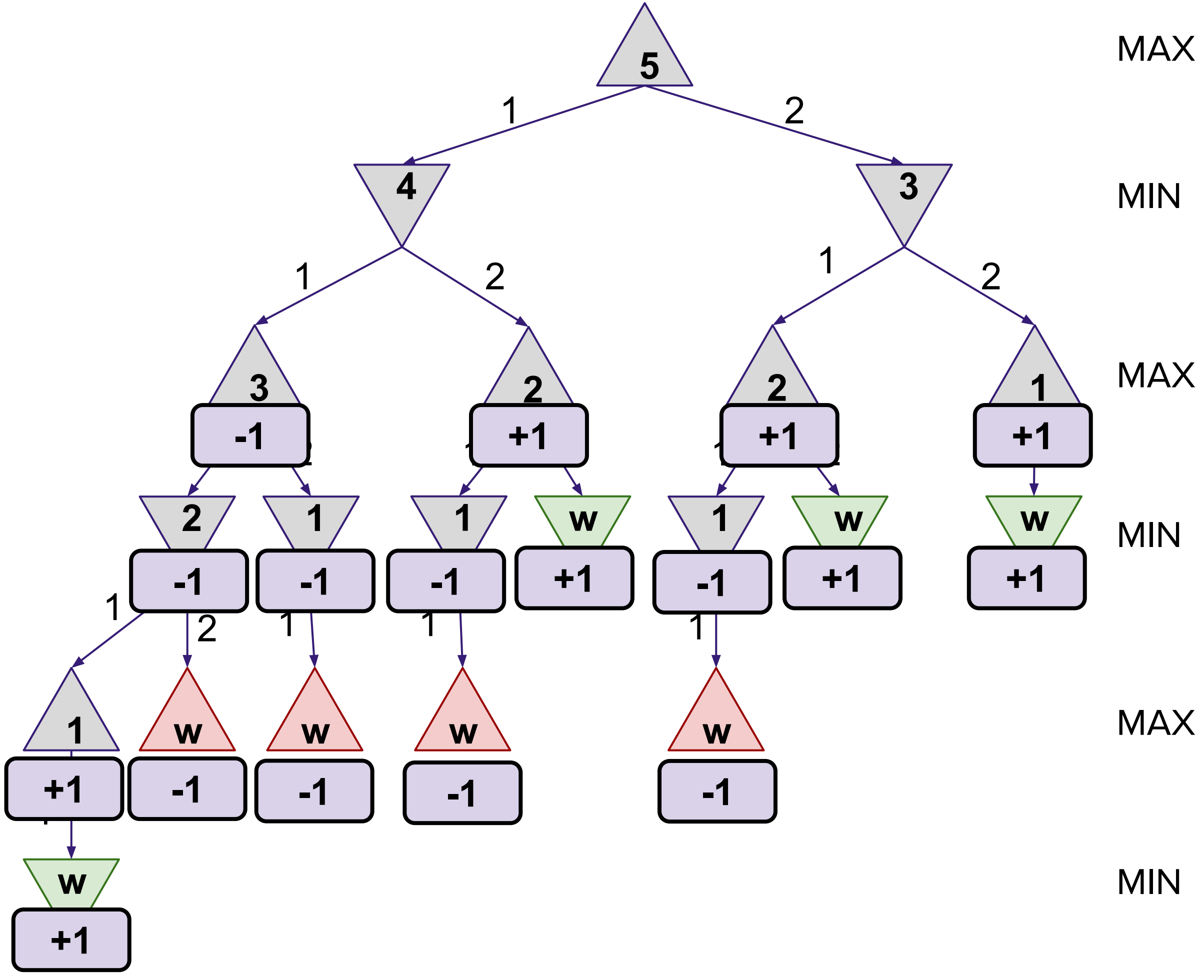


Baby Nim

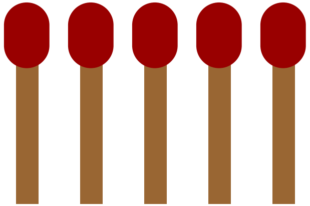


Take 1 or 2 at each turn
Goal: take the last match

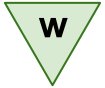




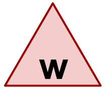
Baby Nim



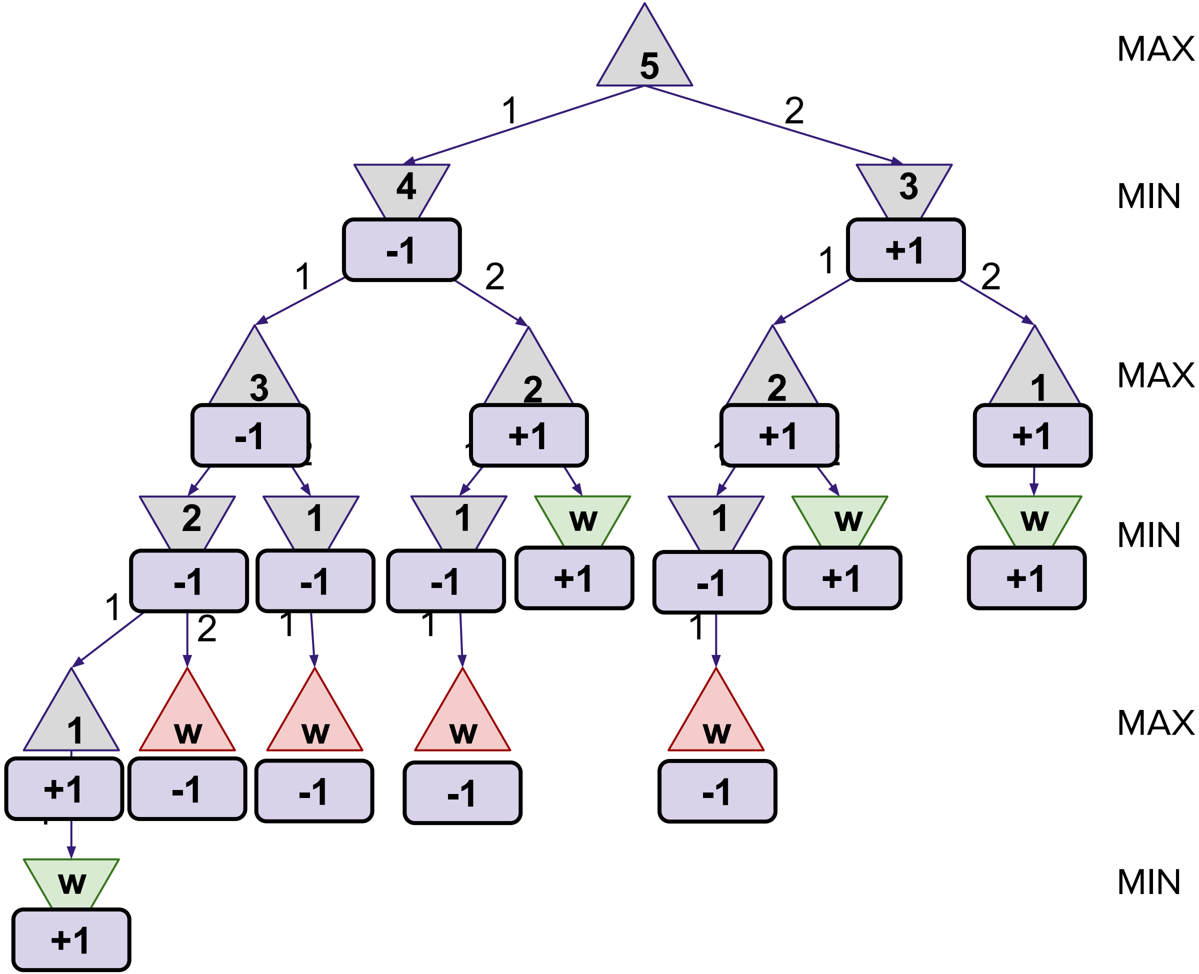
Take 1 or 2 at each turn
Goal: take the last match

MAX Wins 

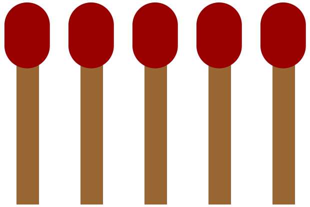
= 1

MIN Wins 

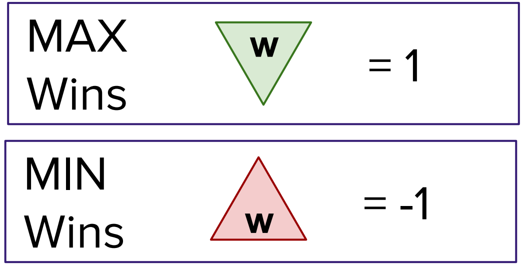
= -1

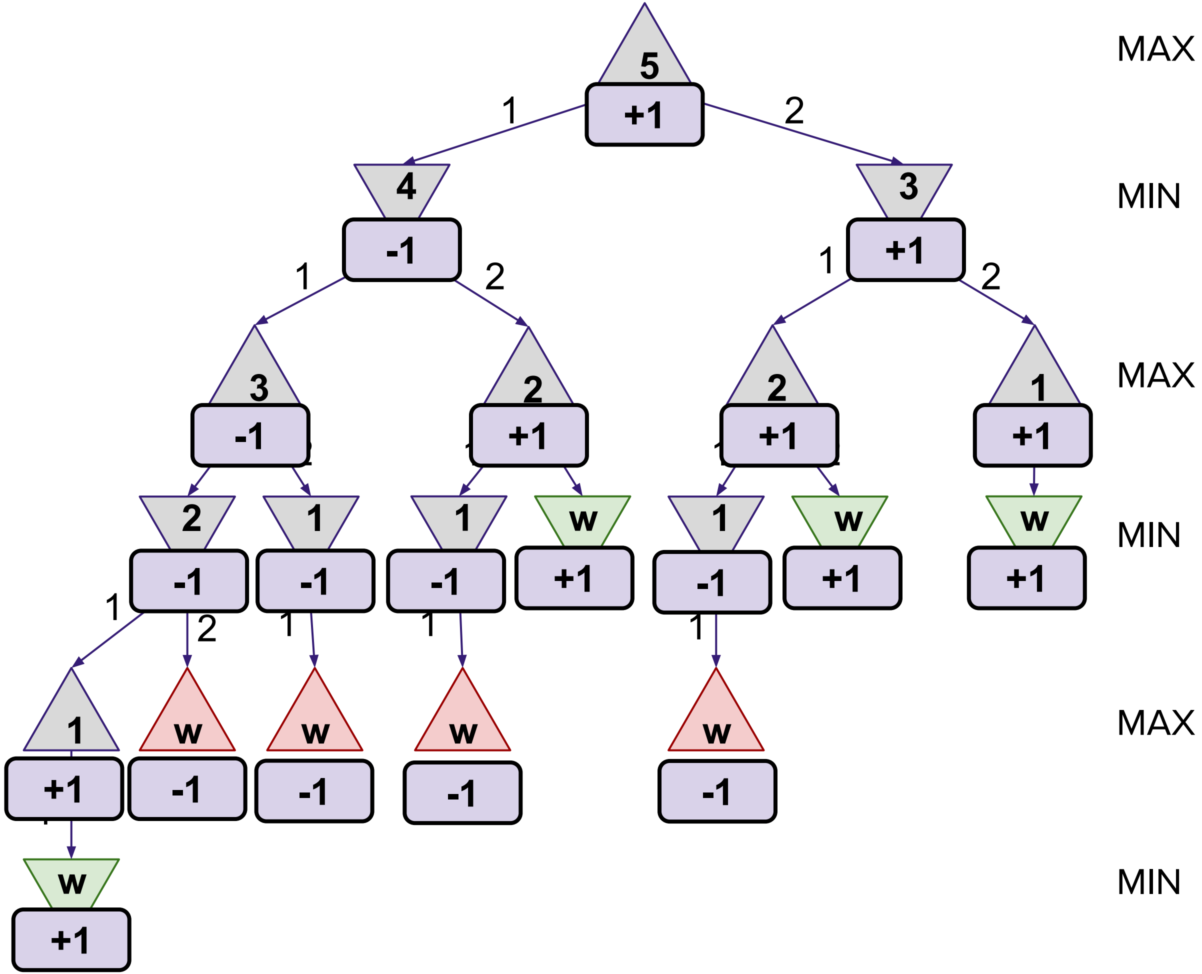


Baby Nim

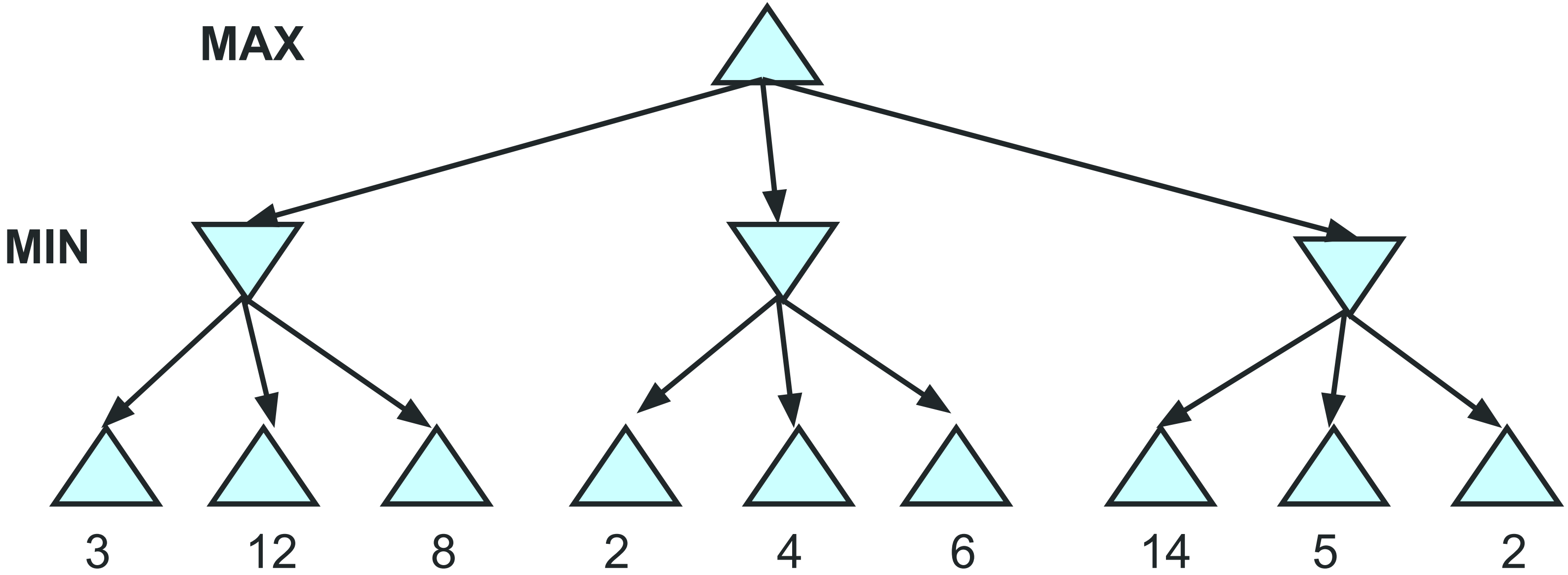


Take 1 or 2 at each turn
Goal: take the last match





MINIMAX example 2



Minimax: An Optimal Strategy

function MINIMAX-DECISION(*state*) **returns** *an action*

$v \leftarrow$ MAX-VALUE(*state*)
return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*) **returns** *a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow -\infty$
for a, s in SUCCESSORS(*state*) **do**
$v \leftarrow$ MAX(v, MIN-VALUE(s))
return v

---

function MIN-VALUE(*state*) **returns** *a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow \infty$
for a, s in SUCCESSORS(*state*) **do**
$v \leftarrow$ MIN(v, MAX-VALUE(s))
return v

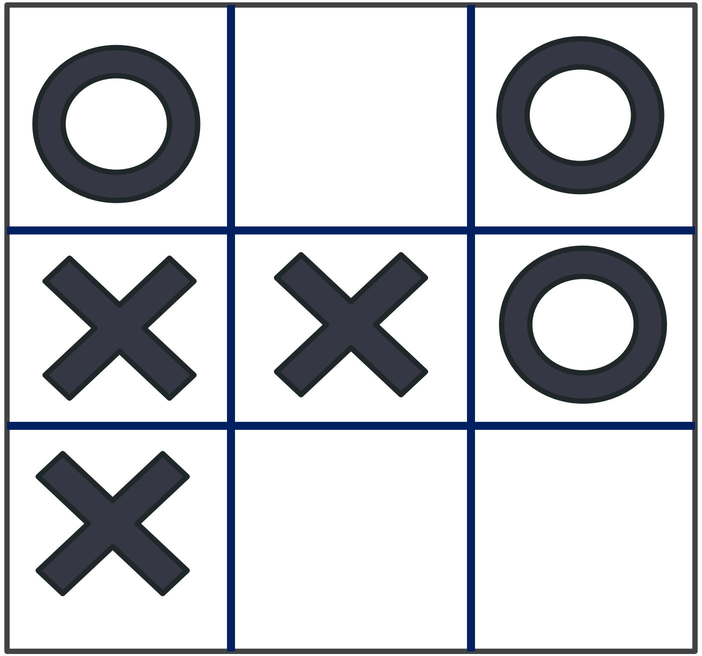
Minimax Algorithm: An Optimal Strategy

Choose the best move based on the resulting states' MINIMAX-VALUE...

```
MINIMAX-VALUE(n) =
    if n is a terminal state
        then Utility(n)
    else if MAX's turn
        the MAXIMUM MINIMAX-VALUE
        of all possible successors to n
    else if MIN's turn
        the MINIMUM MINIMAX-VALUE
        of all possible successors to n
```

Try it out

All positions of X will result in a -1



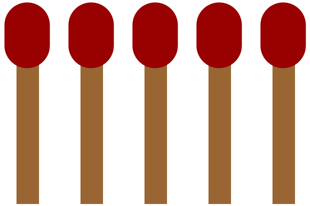
It is currently X’s turn

Use MINIMAX to determine what move X should make?

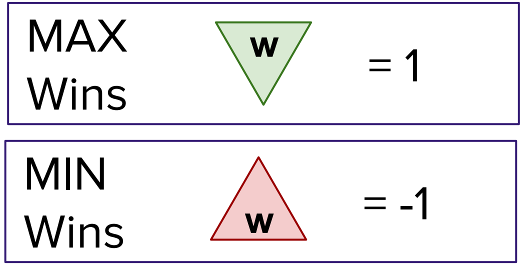
Who will win the game?

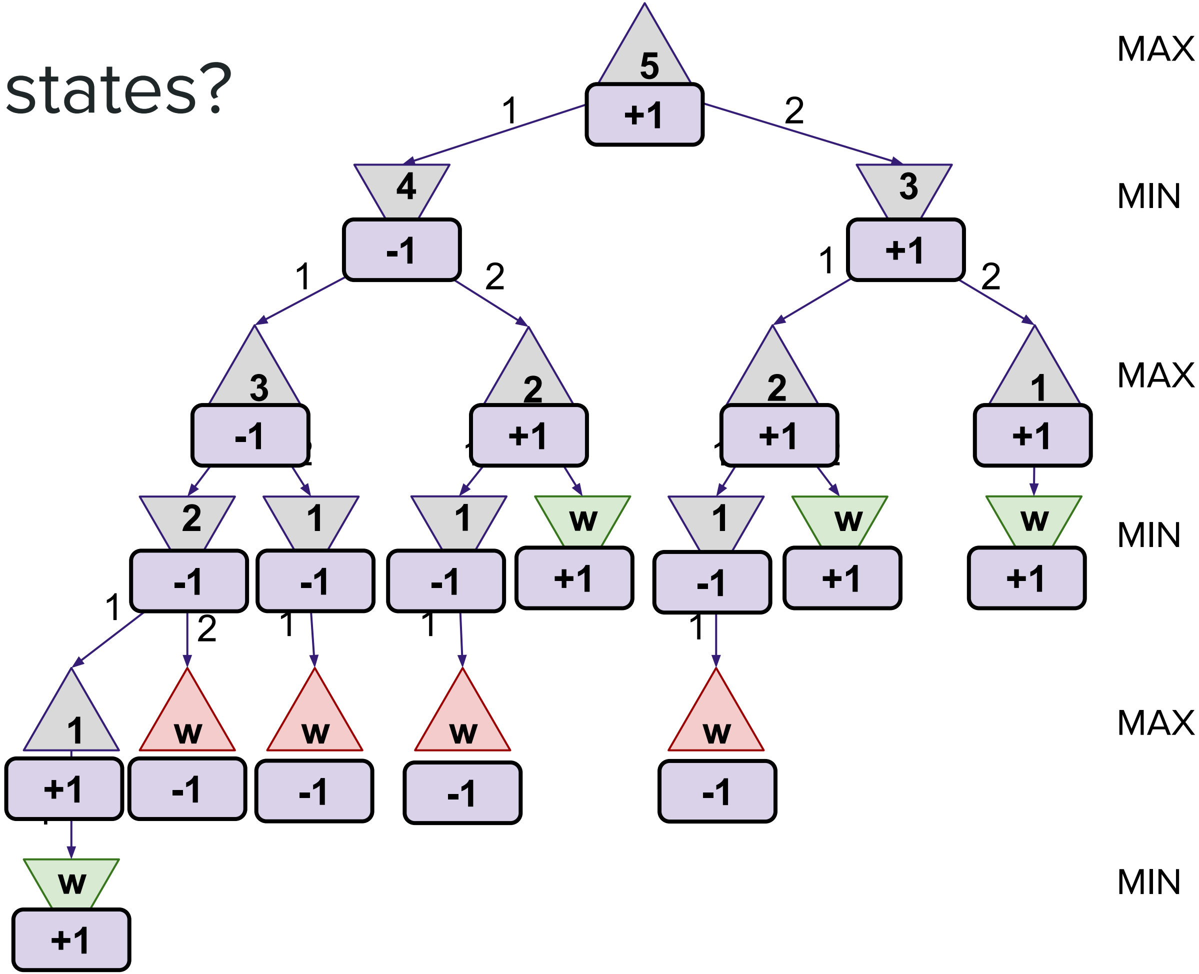
https://upload.wikimedia.org/wikipedia/commons/e/e1/Plminmax.gif

How many game states?



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





Properties of minimax

For chess, b ≈ 35, d ≈100 (100 ply) for "reasonable" games

- exact solution completely infeasible

Is minimax reasonable for

Mancala?	Tic Tac Toe?	Connect Four?
B? 6	B? 4	B? 4
D? ?	D? 9	D? 20

Resource limits

Suppose we have 100 secs, and can explore 10^4 nodes/sec

➔ can explore 10^6 nodes per move

Standard approach (Shannon, 1950):

- **evaluation function**

  estimated desirability of position

- **cutoff test:**

  e.g., depth limit

Cutting off search

Change:

- if TERMINAL-TEST(state) then return UTILITY(state)

into

- if CUTOFF-TEST(state,depth) then return EVAL(state)

- Introduces a fixed-depth limit
  - Is selected so that the amount of time will not exceed what the rules of the game allow
- When cuttoff occurs, the evaluation is performed

Heuristic EVAL

Idea: produce an estimate of the expected utility of the game from a given position.

Performance depends on quality of EVAL.

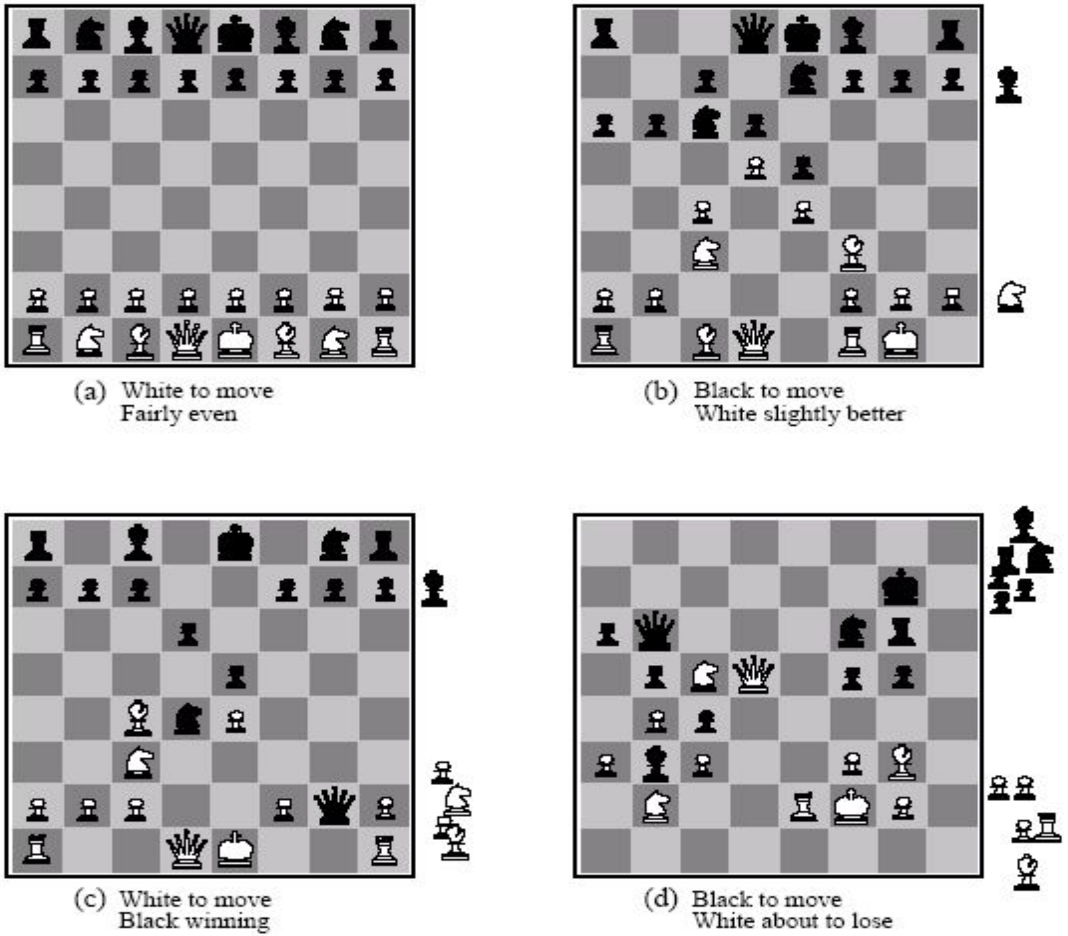
Requirements:

- EVAL should order terminal-nodes in the same way as UTILITY.
- Computation may not take too long.
- For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.

Simple Mancala Heuristic: Goodness of board = # stones in my Mancala minus the number of stones in my opponents.

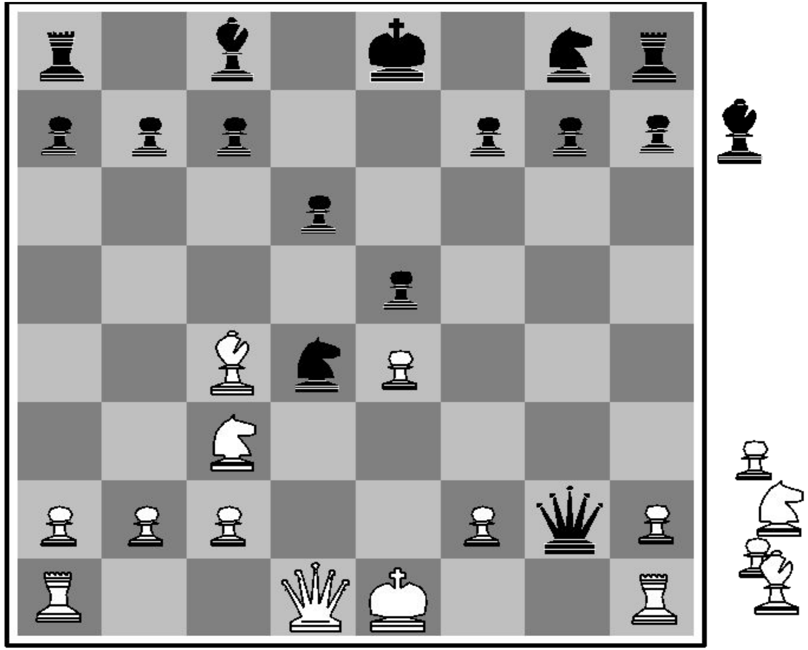
Heuristic EVAL example

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

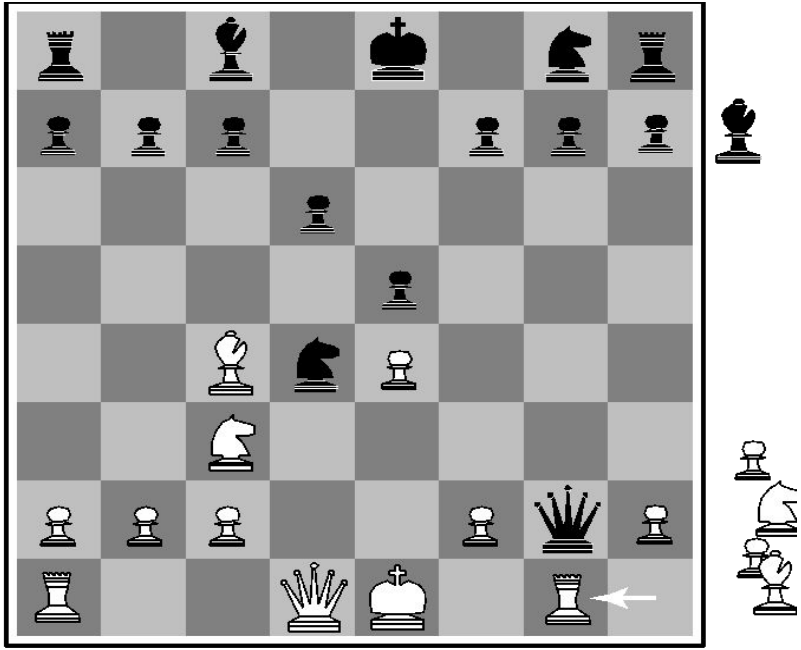


Heuristic difficulties

Simple heuristic - weighing the pieces by material value

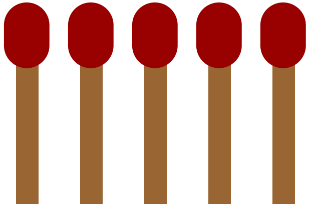


(a) White to move

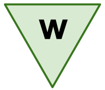


(b) White to move

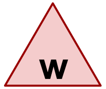
Baby Nim



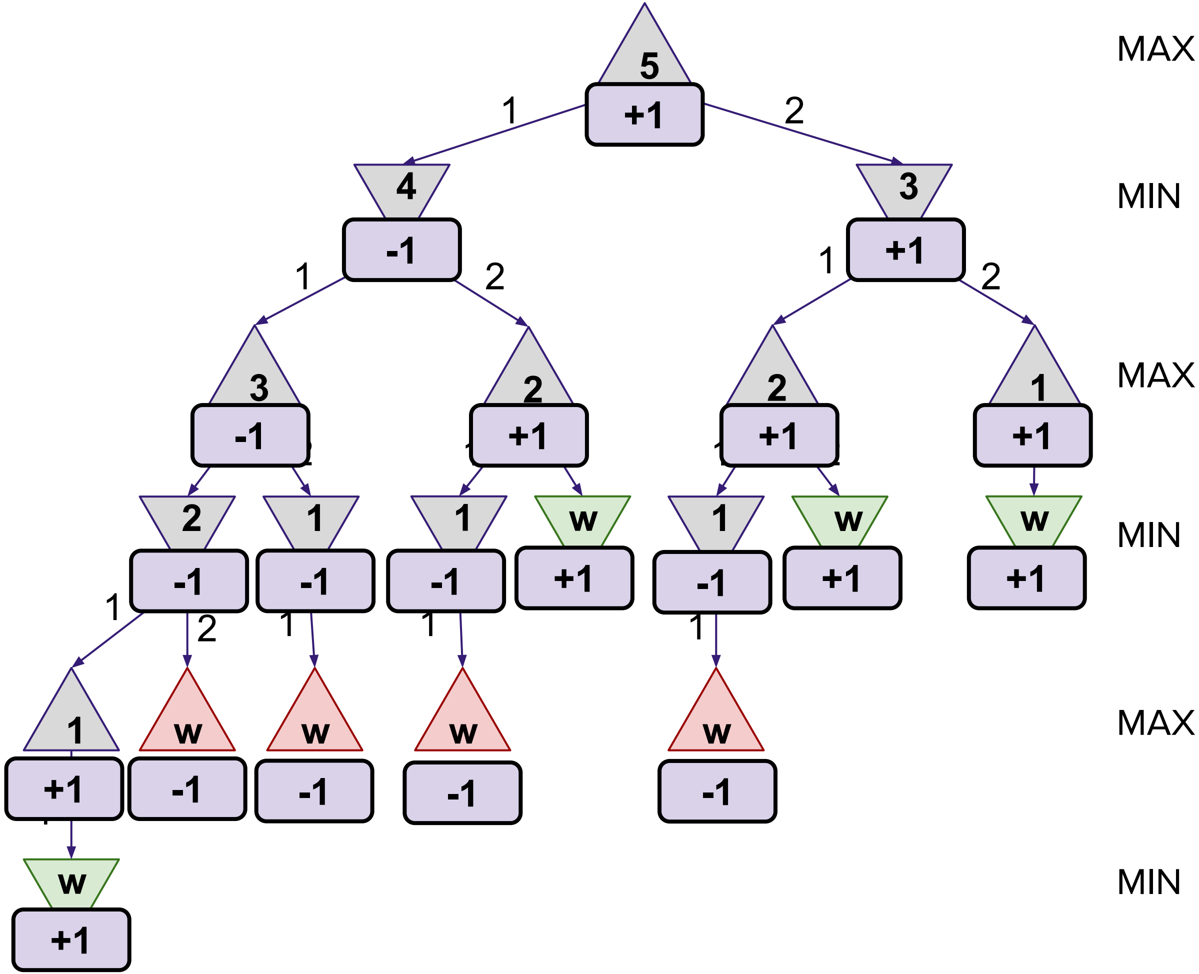
Take 1 or 2 at each turn
Goal: take the last match

MAX Wins 

= 1

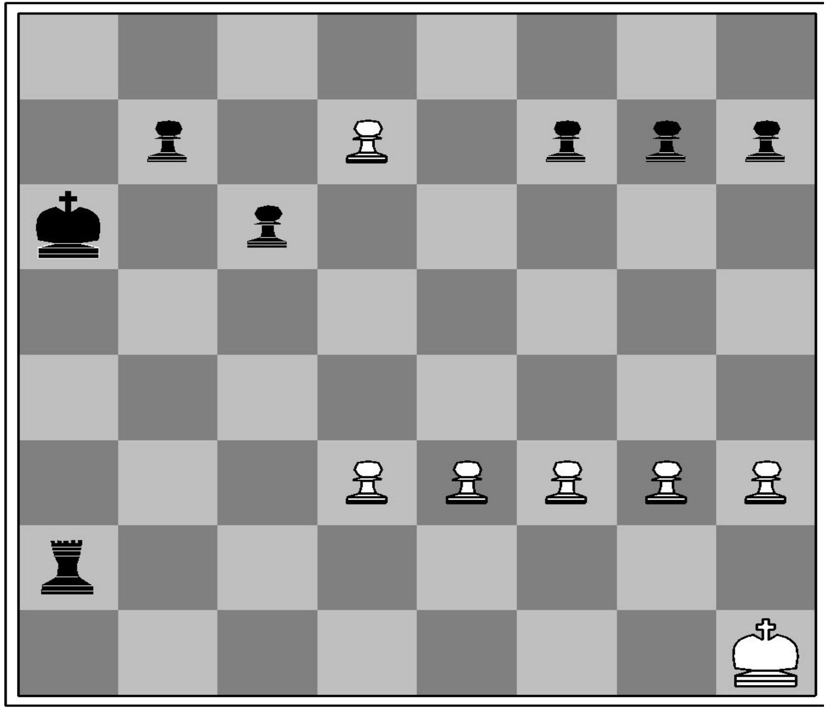
MIN Wins 

= -1



Weakest link : Not knowing what’s “on the horizon”

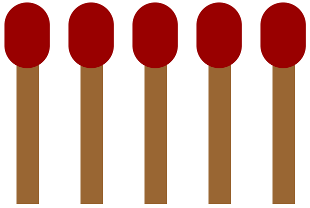
Horizon effect



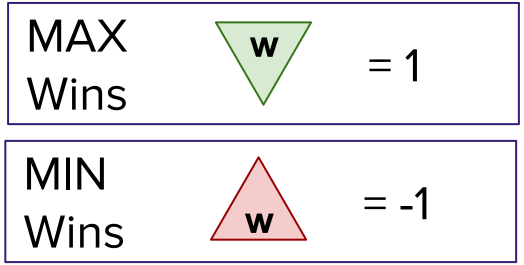
Black to move

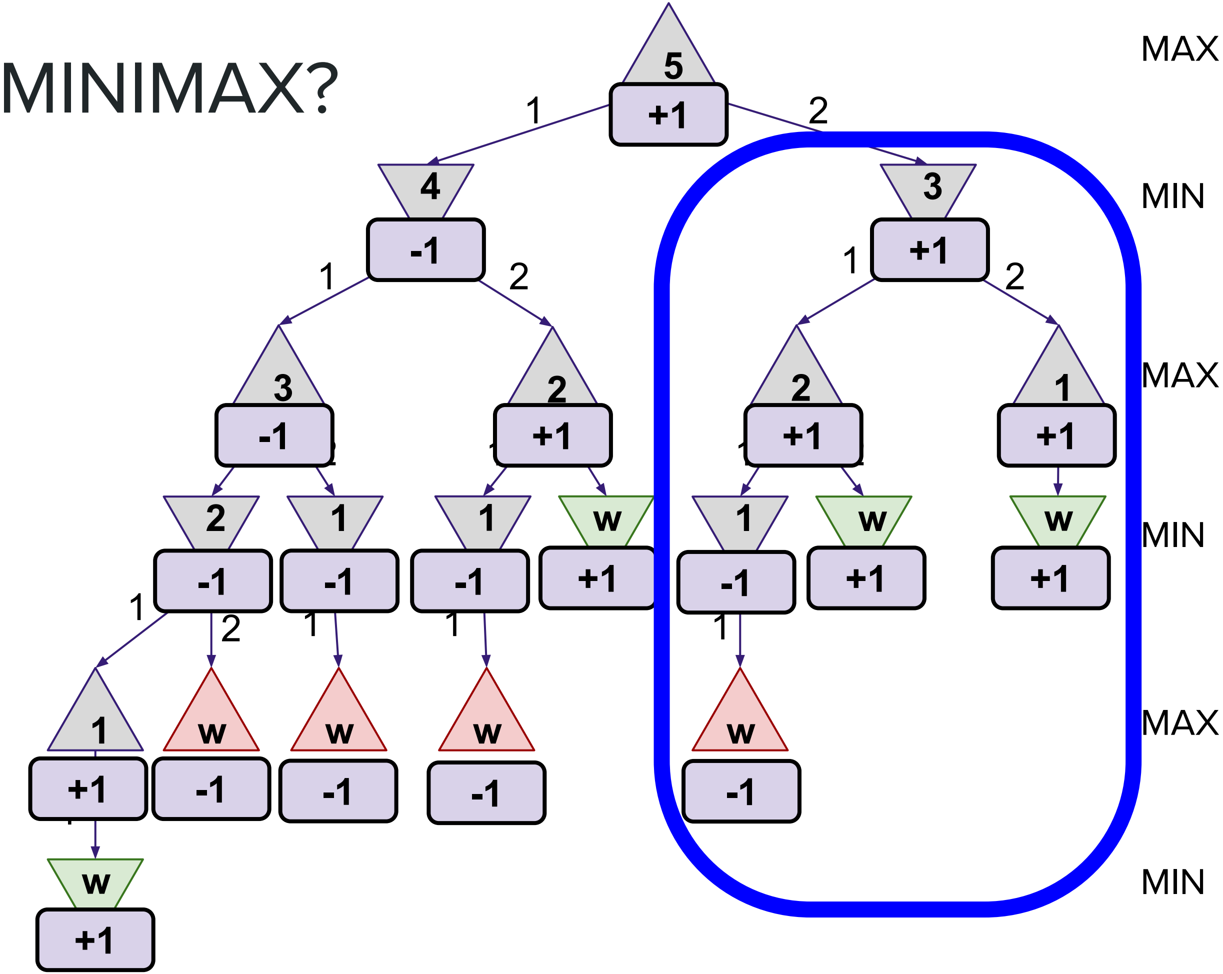
Fixed depth search thinks it can avoid the queening move

Challenges with MINIMAX?



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





Alpha-Beta Pruning

Pruning: eliminate parts of the tree from consideration

Alpha-Beta pruning: prunes away branches that can't possibly influence the final decision

Consider a node n

- If a player has a better choice m (at a parent or further up), then n will never be reached
- So, once we know enough about n by looking at some successors, then we can prune it.