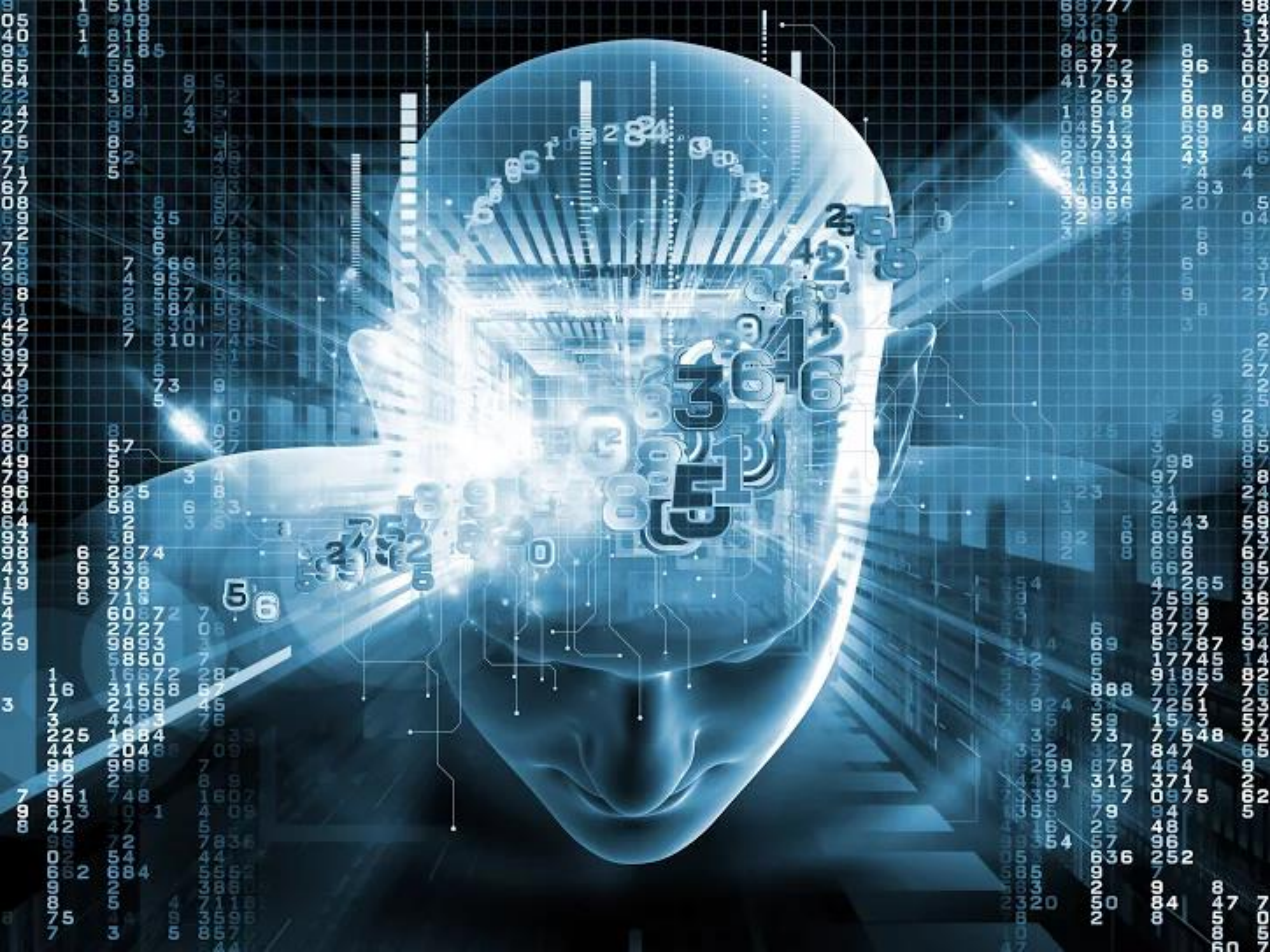


Adversaries

What happens when you are confronted with a world in which there is an agent trying to defeat you?

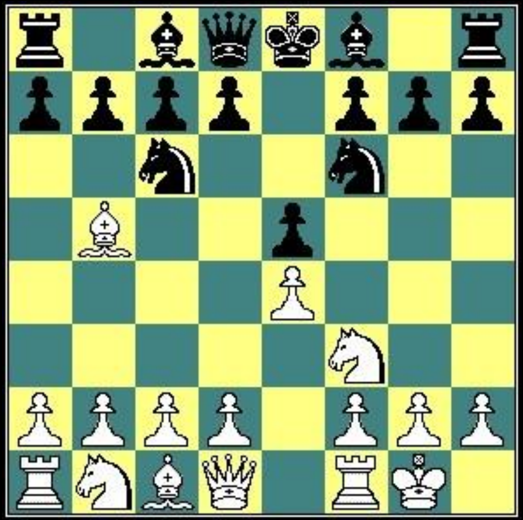
Adversaries

You are trying to maximize your benefits while someone is trying to maximize theirs.

If the situation is zero-sum, then your reasoning has to incorporate their actions as well as your own.

humans

good at evaluating the strength of a board for a player



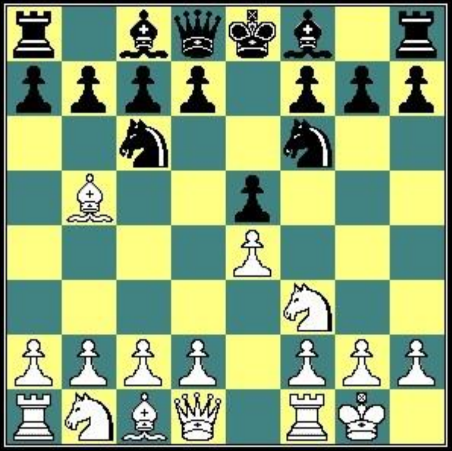
computers

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

How humans play games…

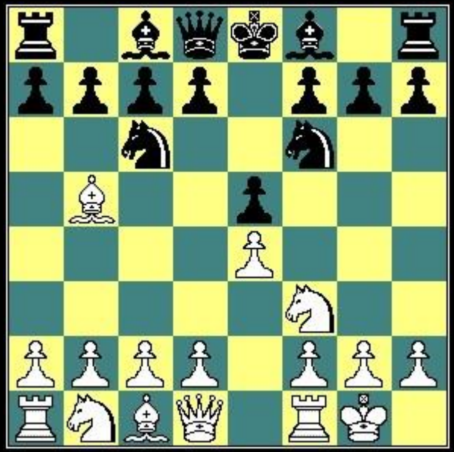
An experiment (by deGroot) was performed in which chess positions were shown to novice and expert players…

- experts could reconstruct these perfectly
- novice players did far worse…



How humans play games…

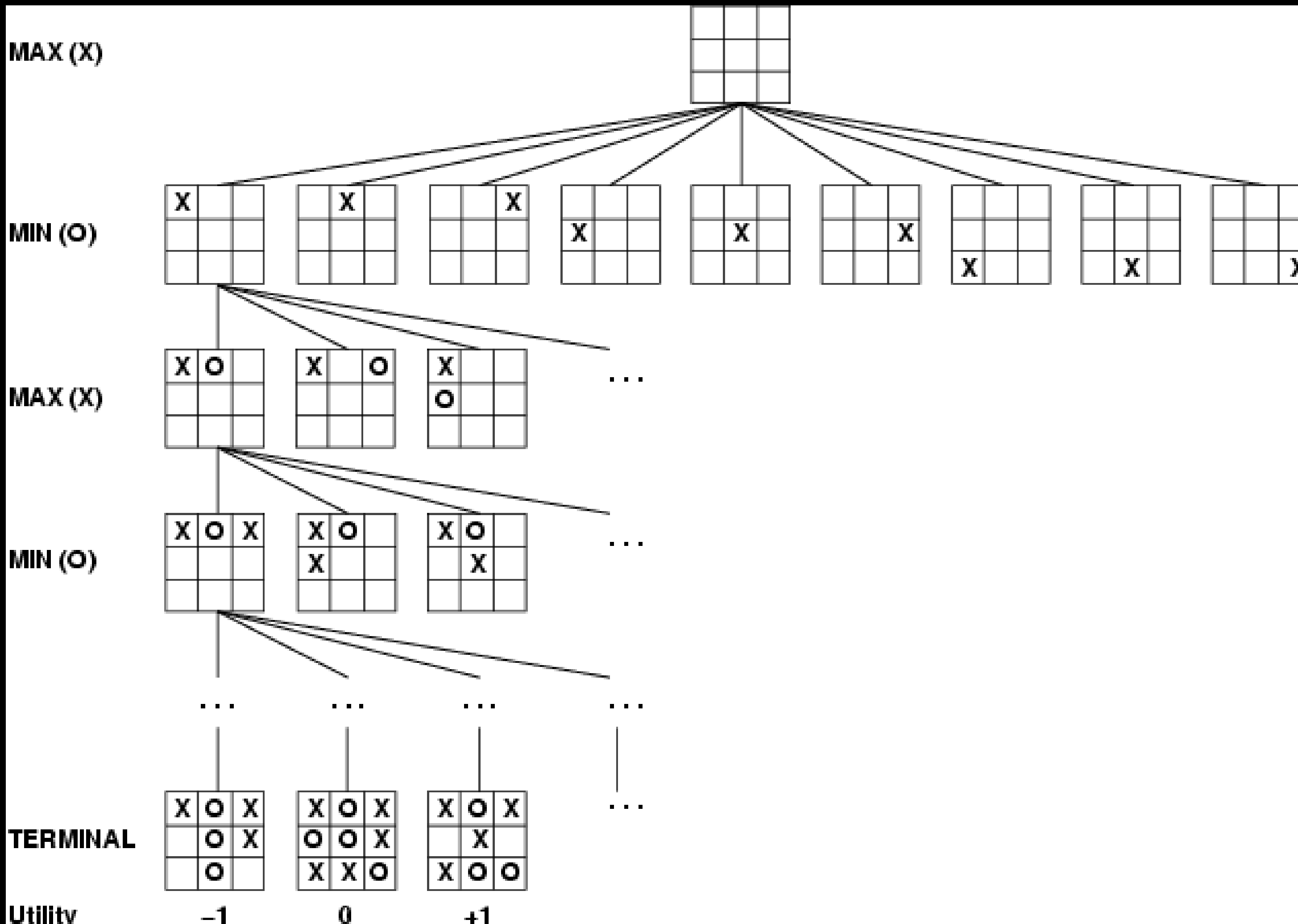
An experiment (by deGroot) was performed in which chess positions were shown to novice and expert players…



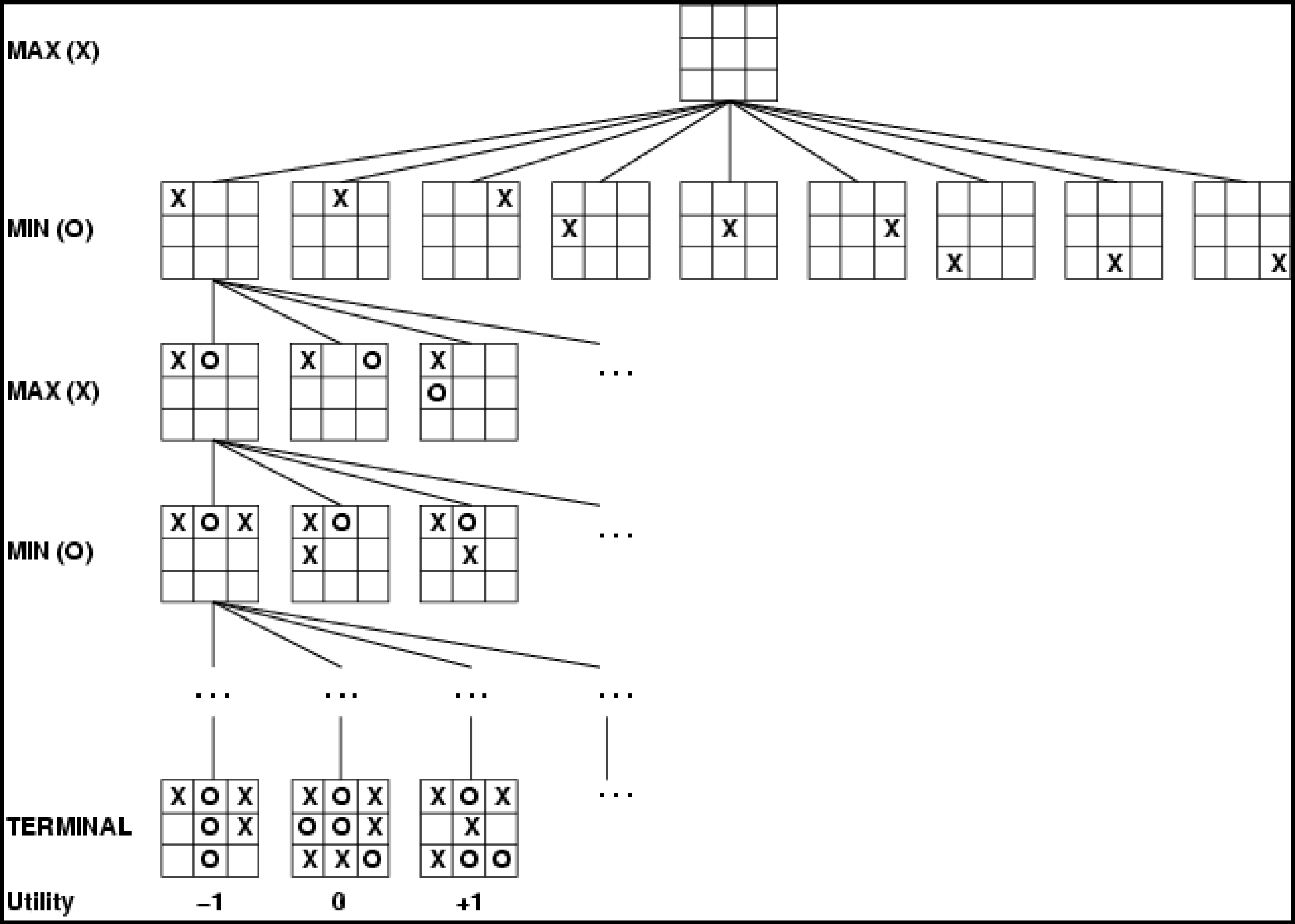
- experts could reconstruct these perfectly
- novice players did far worse…

<u>Random</u> chess positions (not legal ones) were then shown to the two groups

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


MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
–1
0
+1

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence of actions
- Non-observable → sensorless (conformant) problem
  - Agent may have no idea where it is; solution is a sequence
- Nondeterministic and/or partially observable → contingency problem
  - percepts provide new information about current state
  - often interleave search, execution
- Unknown state space → exploration problem

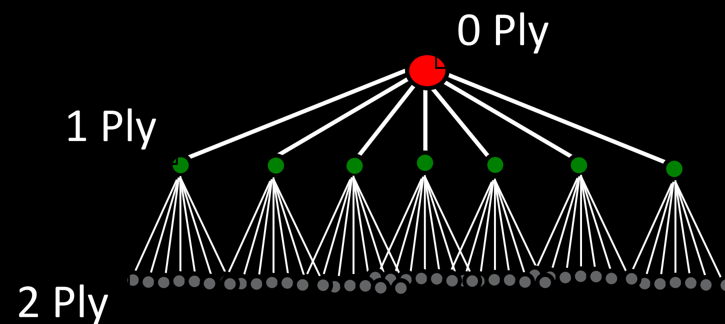

MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
–1
0
+1

Games' Branching Factors

• On average, there are fewer than 40 possible moves that a chess player can make from any board configuration…

18 Ply!!

Hydra at home in the United Arab Emirates…



Branching Factor Estimates
for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	40
Go	300

- 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

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*

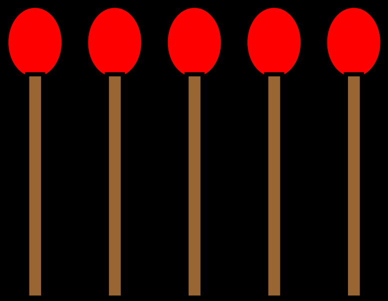
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow \infty$
for a, s in SUCCESSORS(*state*) **do**
  $v \leftarrow$ MIN(v, MAX-VALUE(s))
return v

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

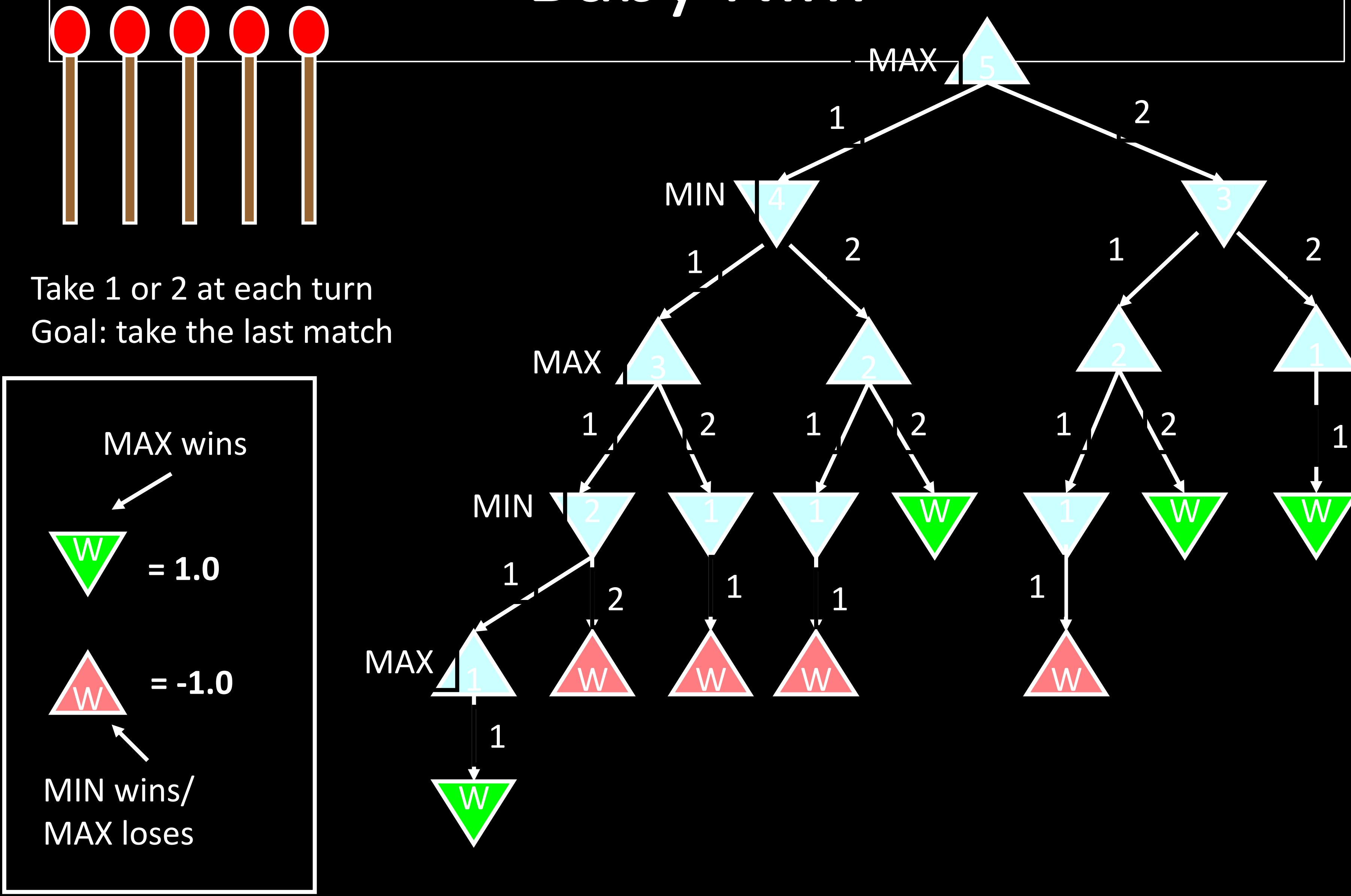
Baby Nim



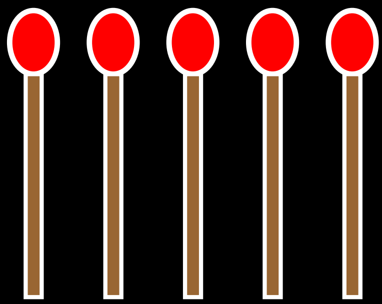
Take 1 or 2 at each turn
Goal: take the last match

Baby Nim

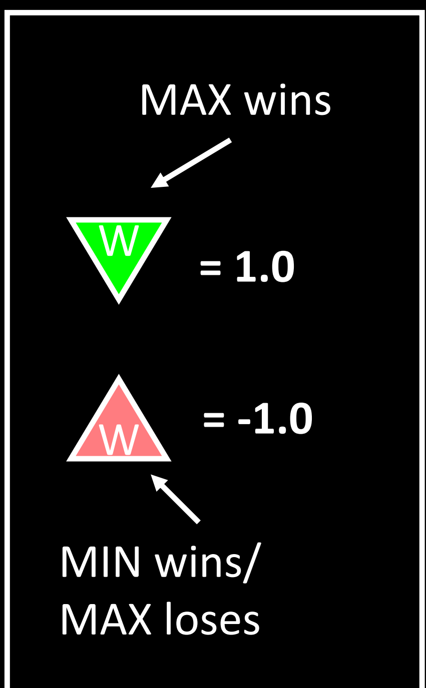
Take 1 or 2 at each turn
Goal: take the last match

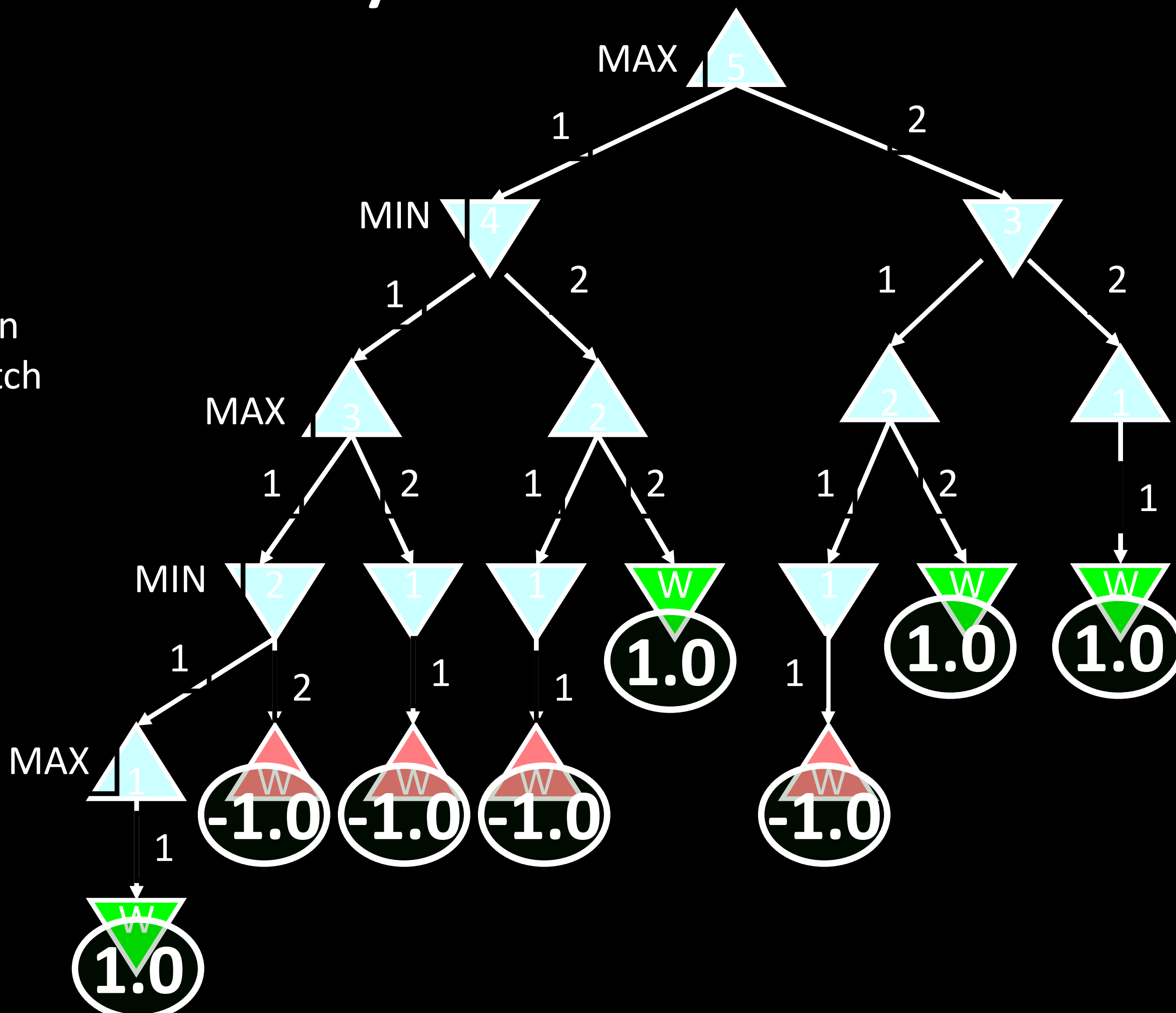


Baby Nim

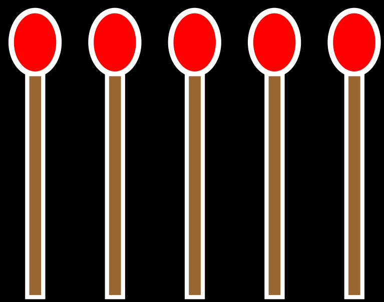


Take 1 or 2 at each turn
Goal: take the last match

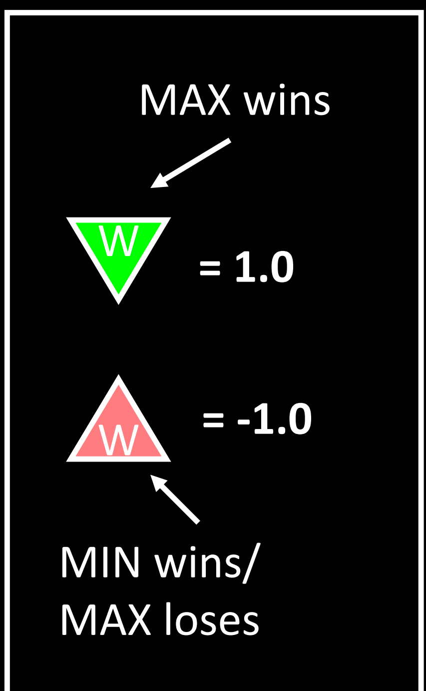


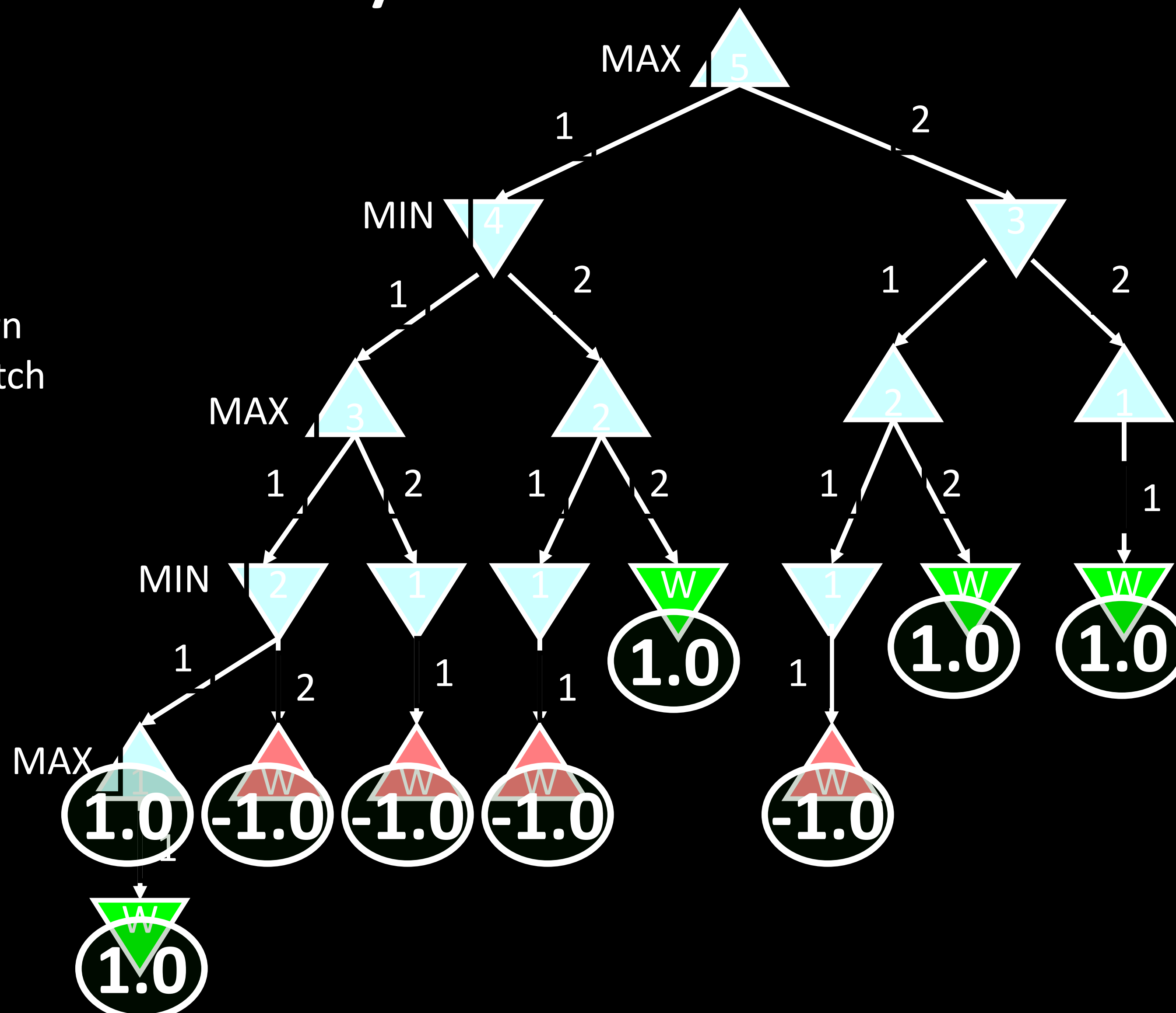


Baby Nim

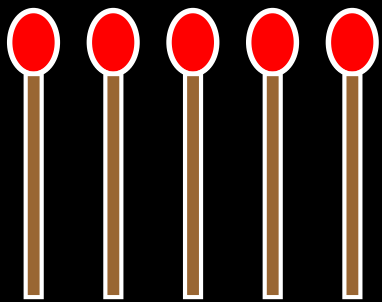


Take 1 or 2 at each turn
Goal: take the last match

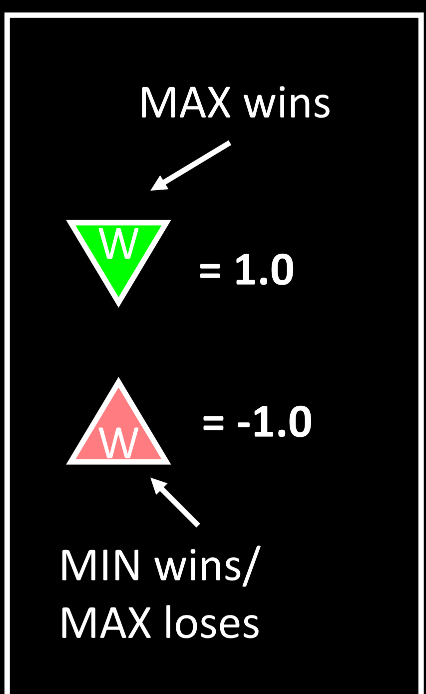


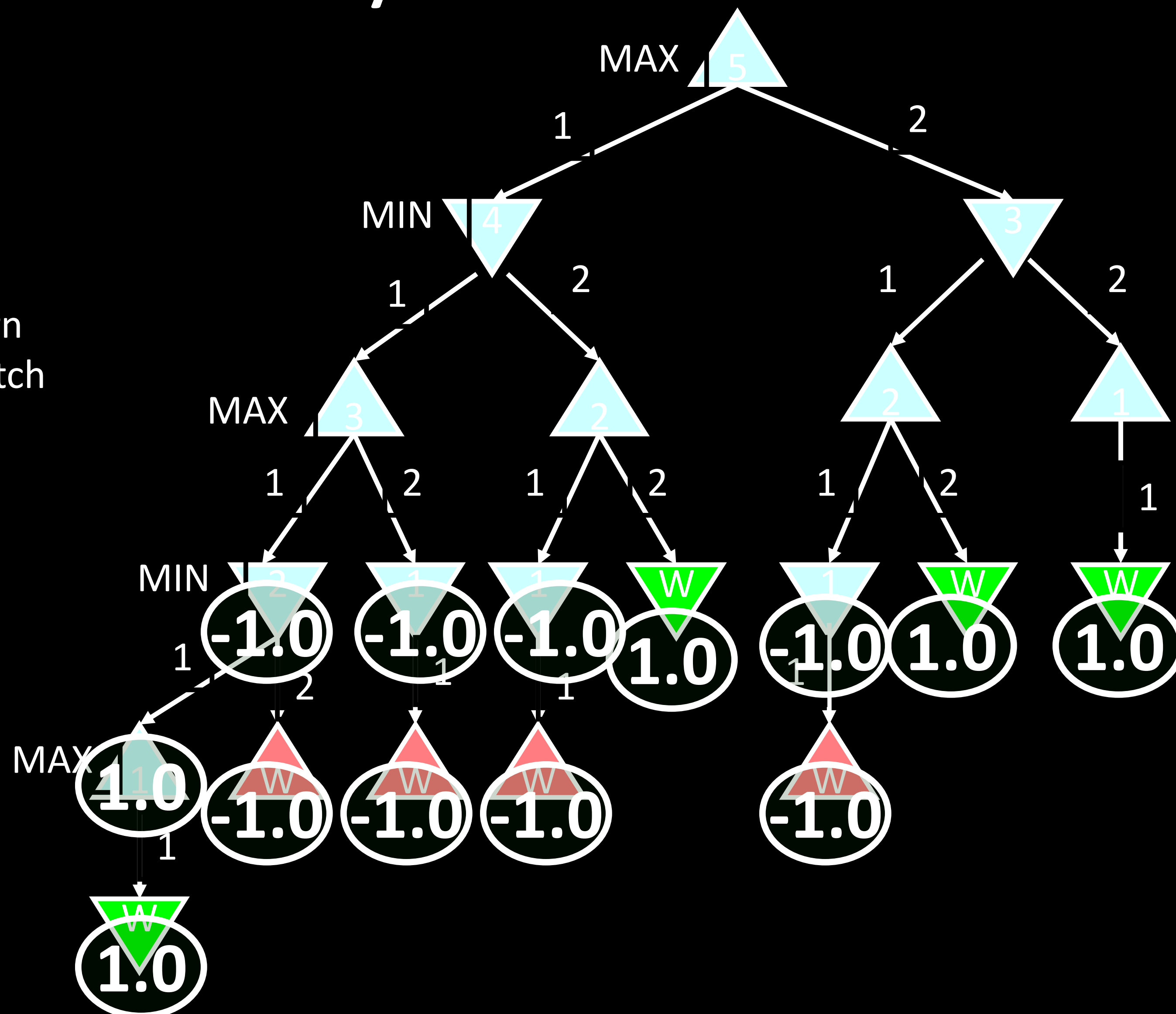


Baby Nim

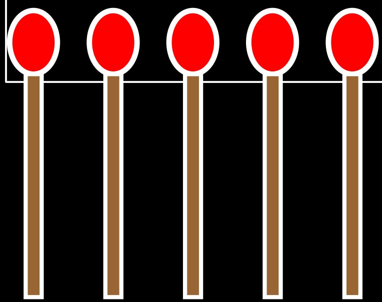


Take 1 or 2 at each turn
Goal: take the last match

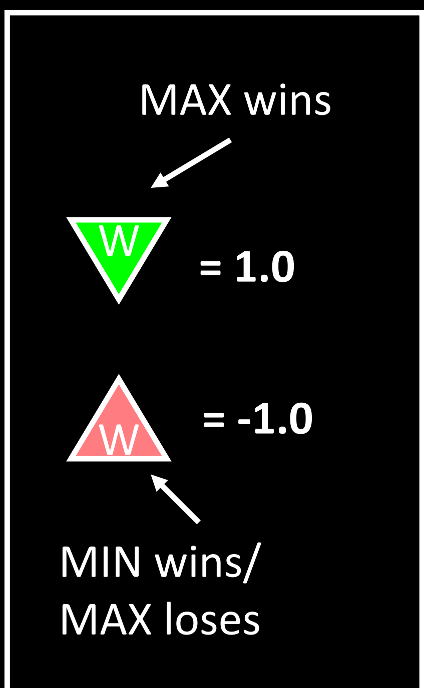


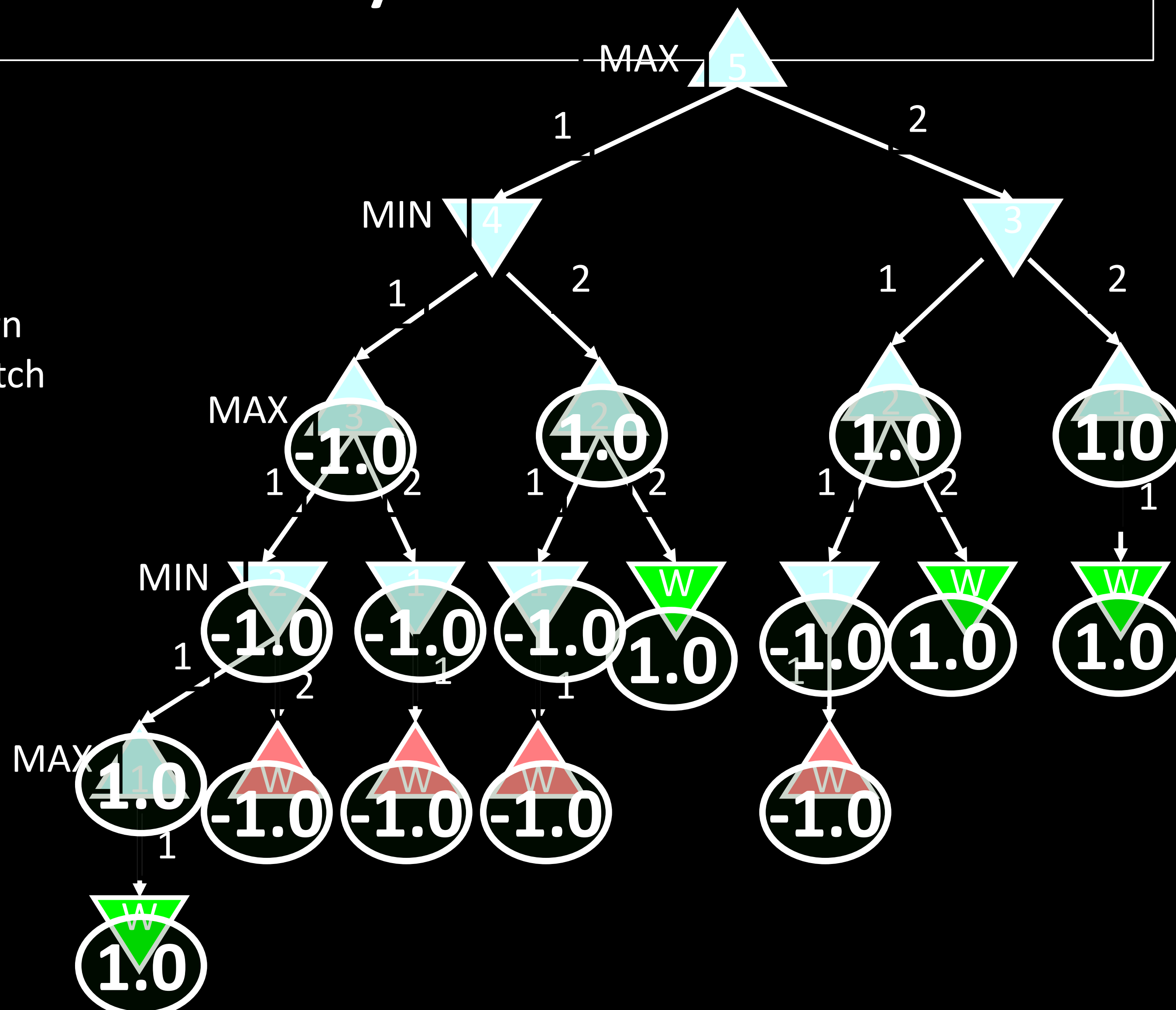


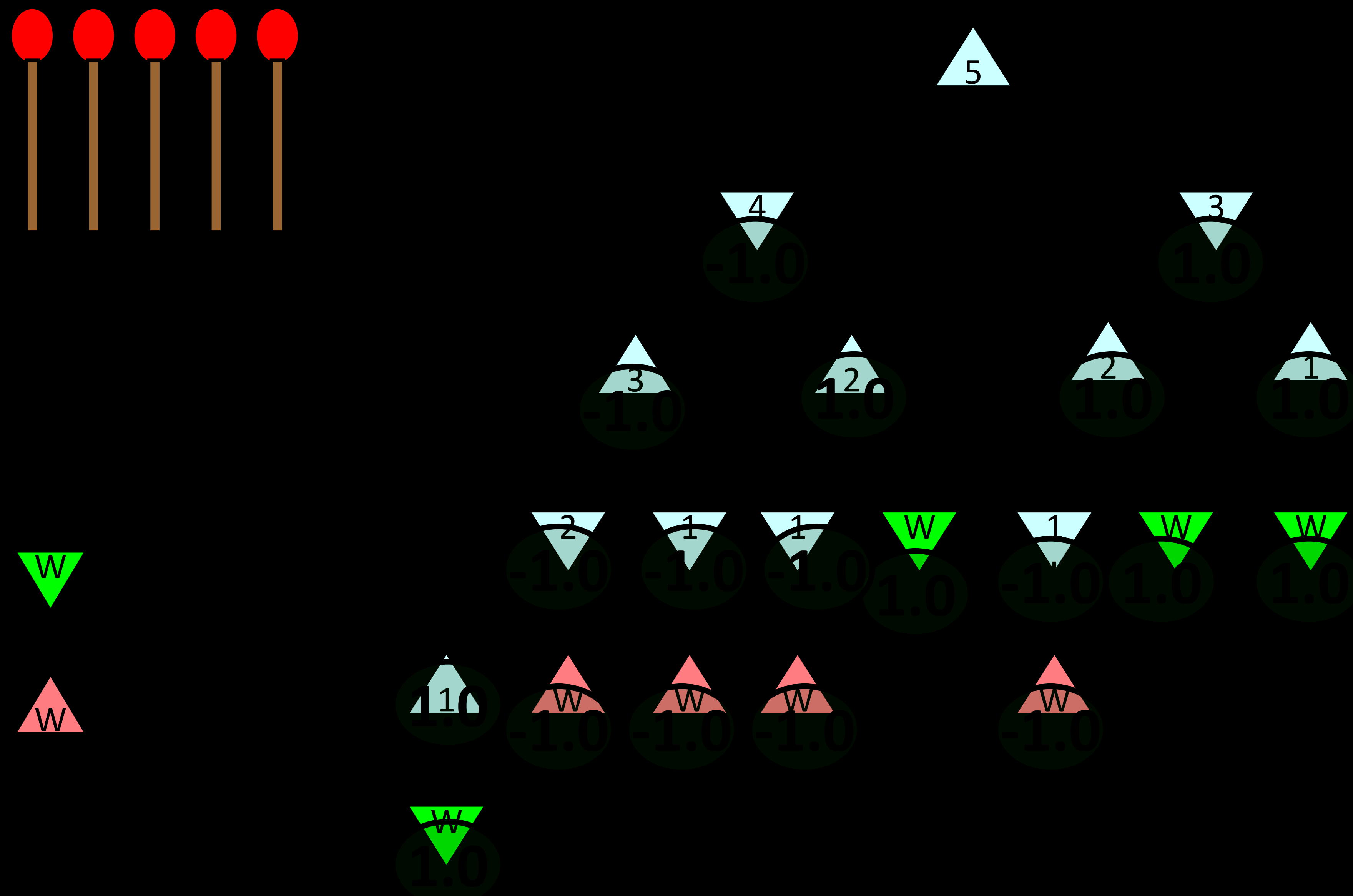
Baby Nim



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

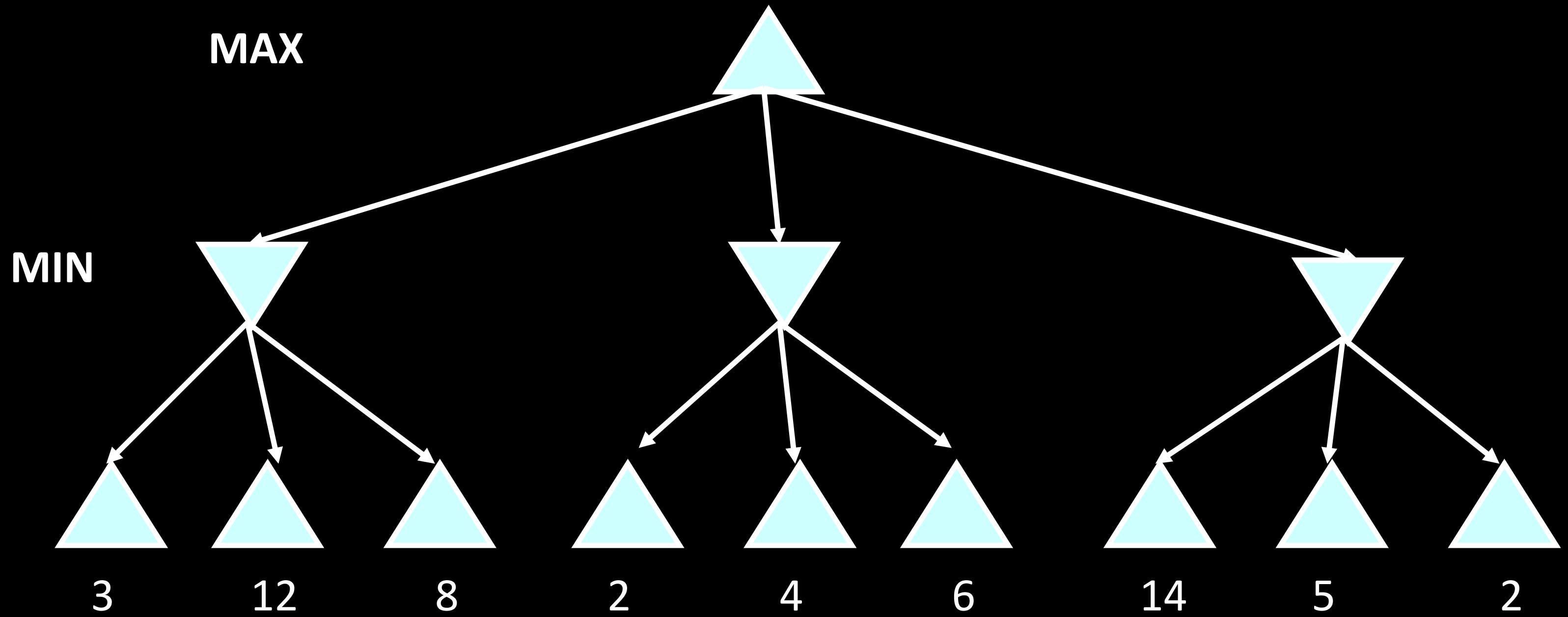






5
4
3
-1.0
1.0
3
2
2
1
2
1
1
W
1
W
W
W
W
W
W
W
W
W

MINIMAX example 2

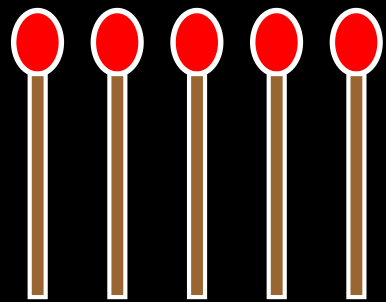


Properties of minimax

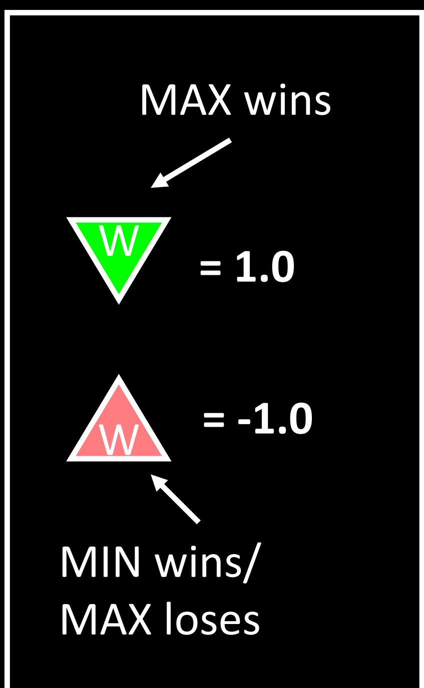
- For chess, b ≈ 35, d ≈100 for "reasonable" games
  → exact solution completely infeasible
-

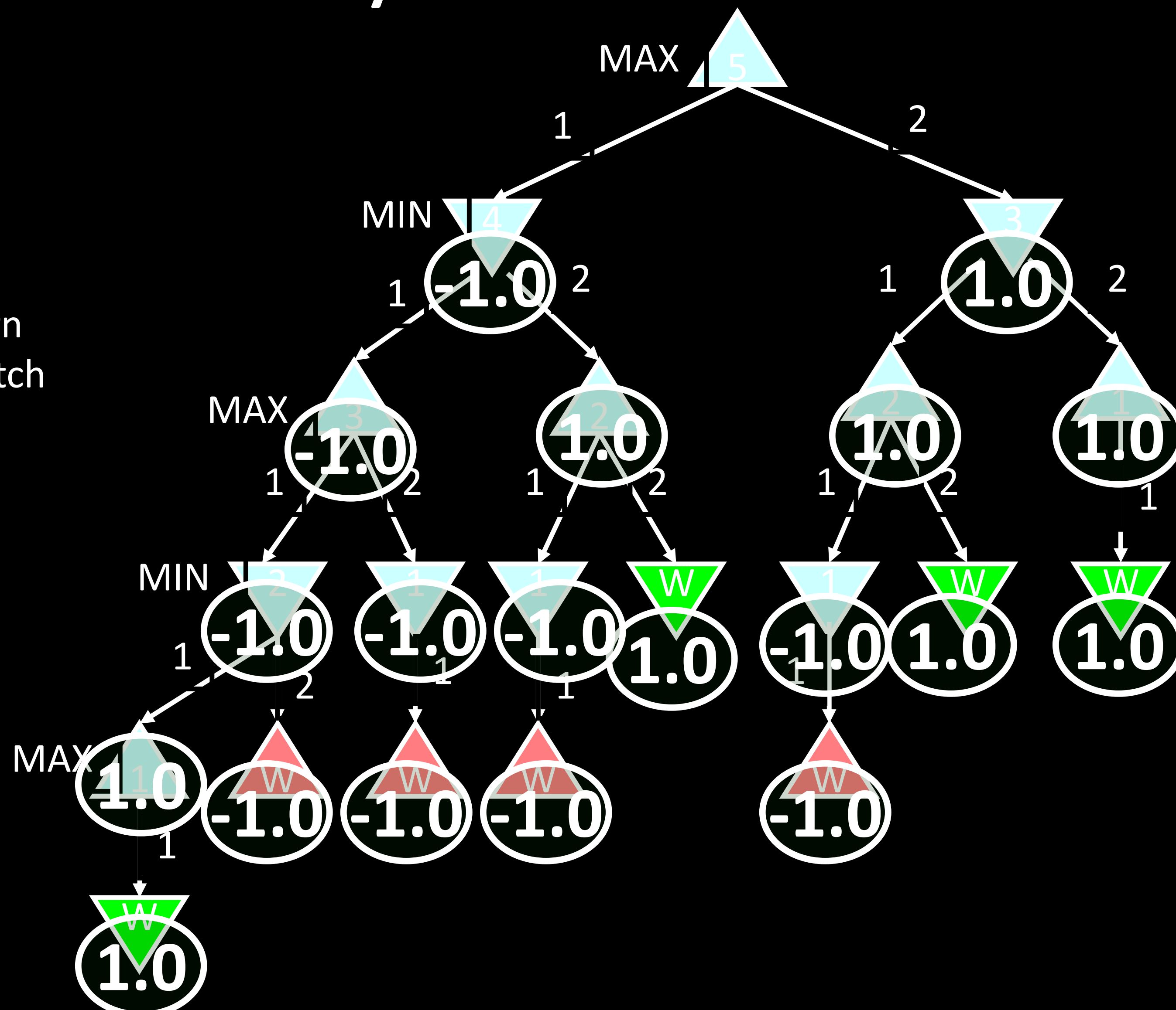
- Is minimax reasonable for
  - Mancala?
    - B?
    - D?
  - Tic Tac Toe?
    - B?
    - D?

Baby Nim



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

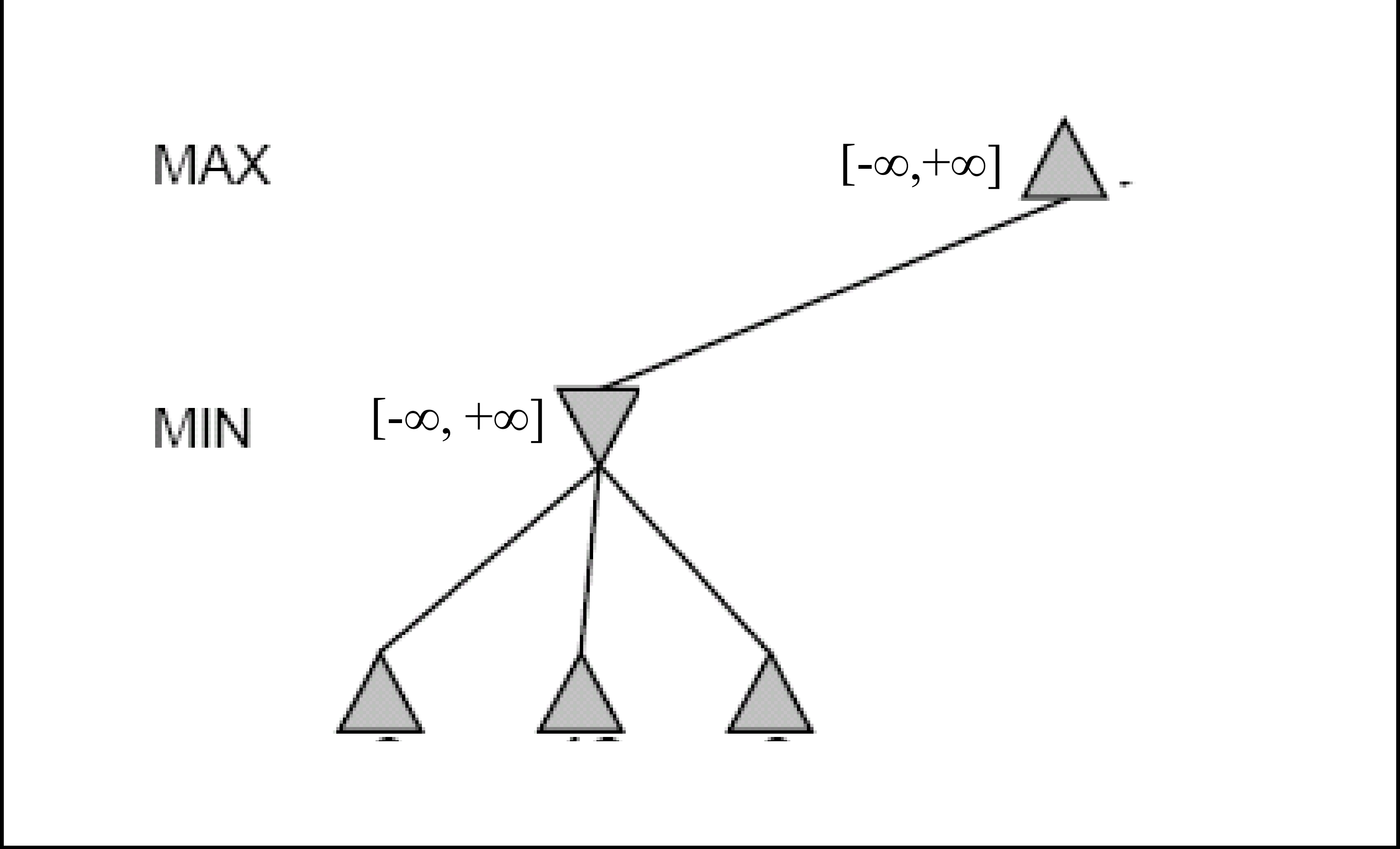
Consider a node *n*

If a player has a better choice *m* (at a parent or further up), then *n* will never be reached

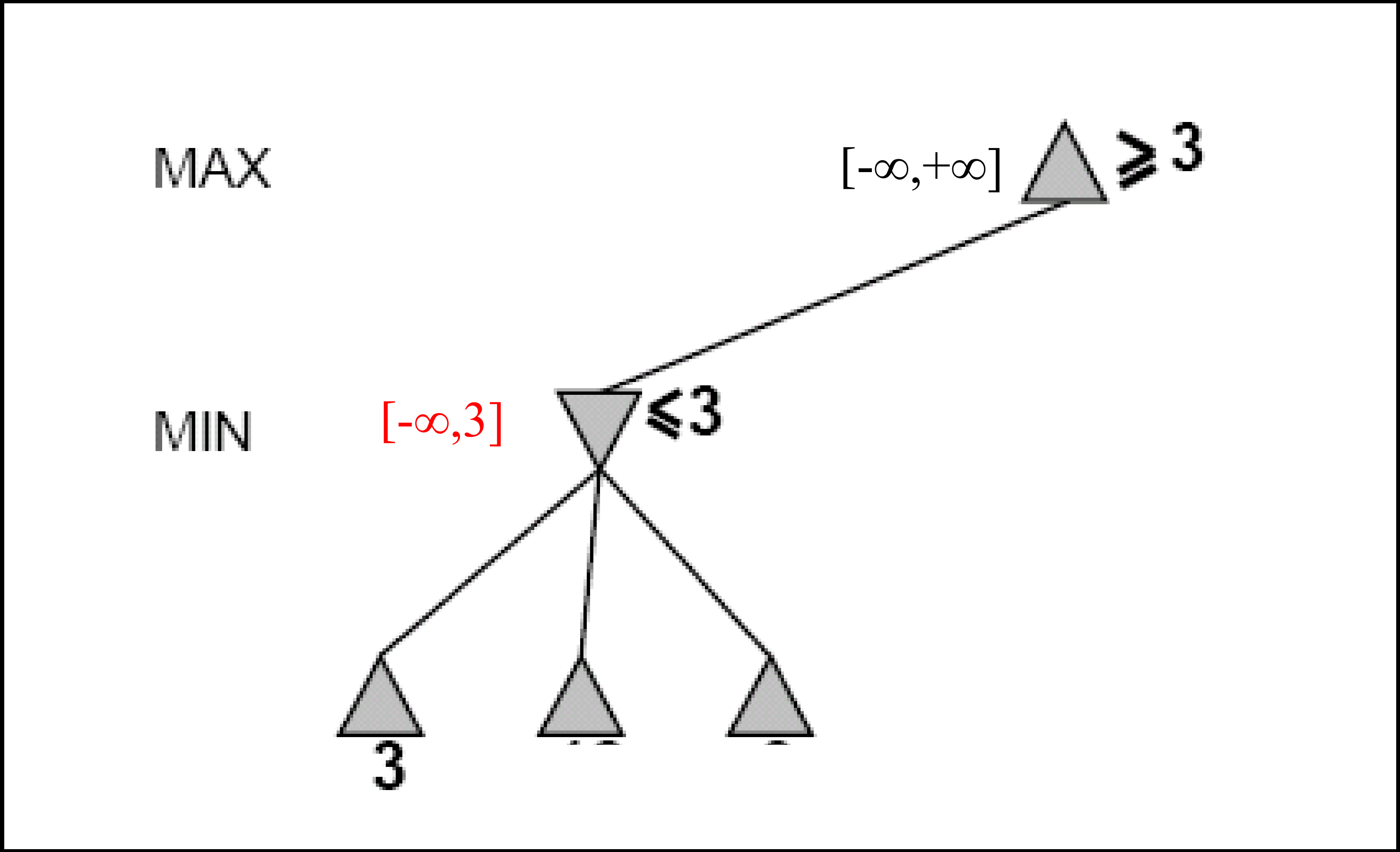
So, once we know enough about *n* by looking at some successors, then we can prune it.

Alpha-Beta Example

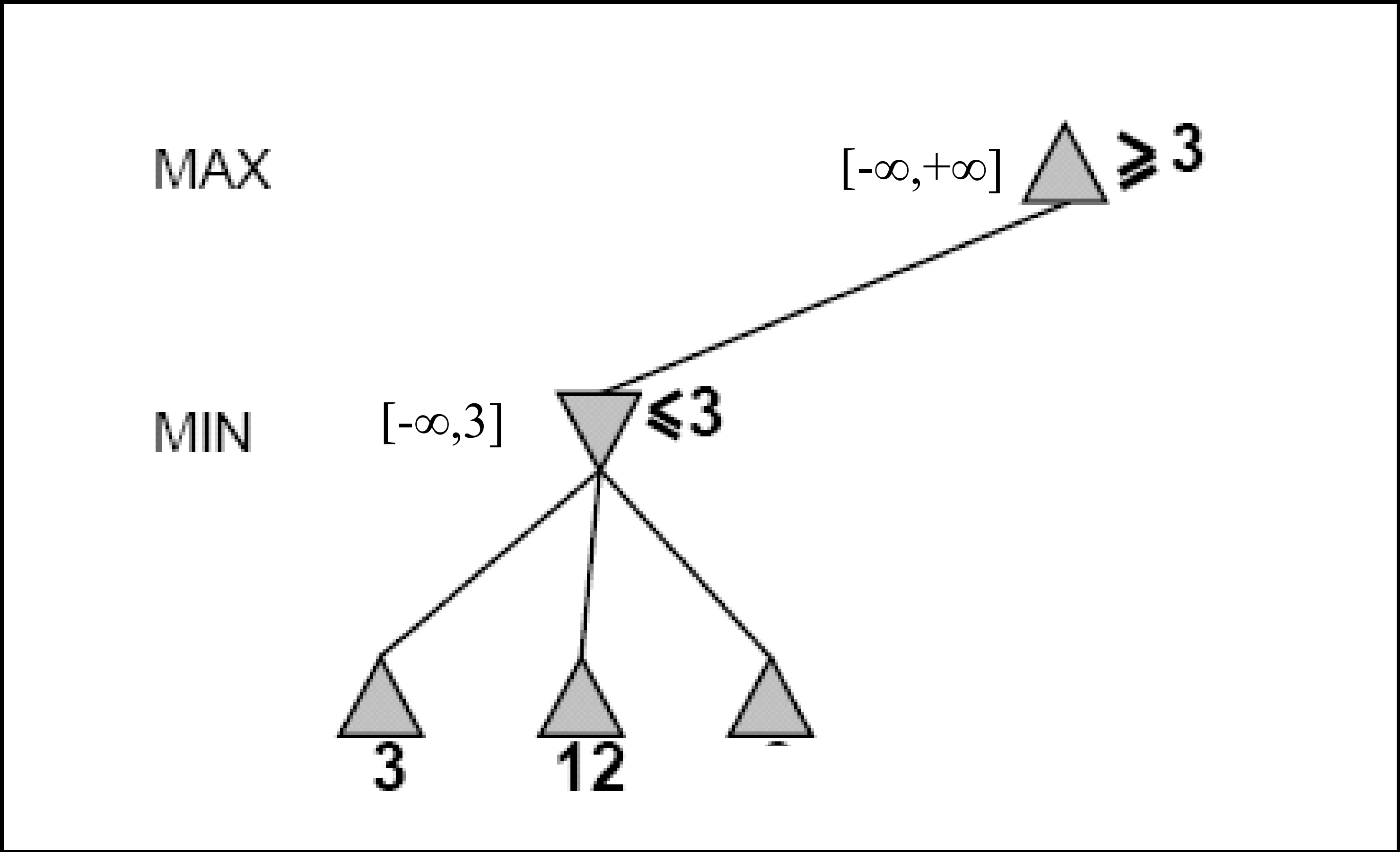
Do DF-search until first leaf

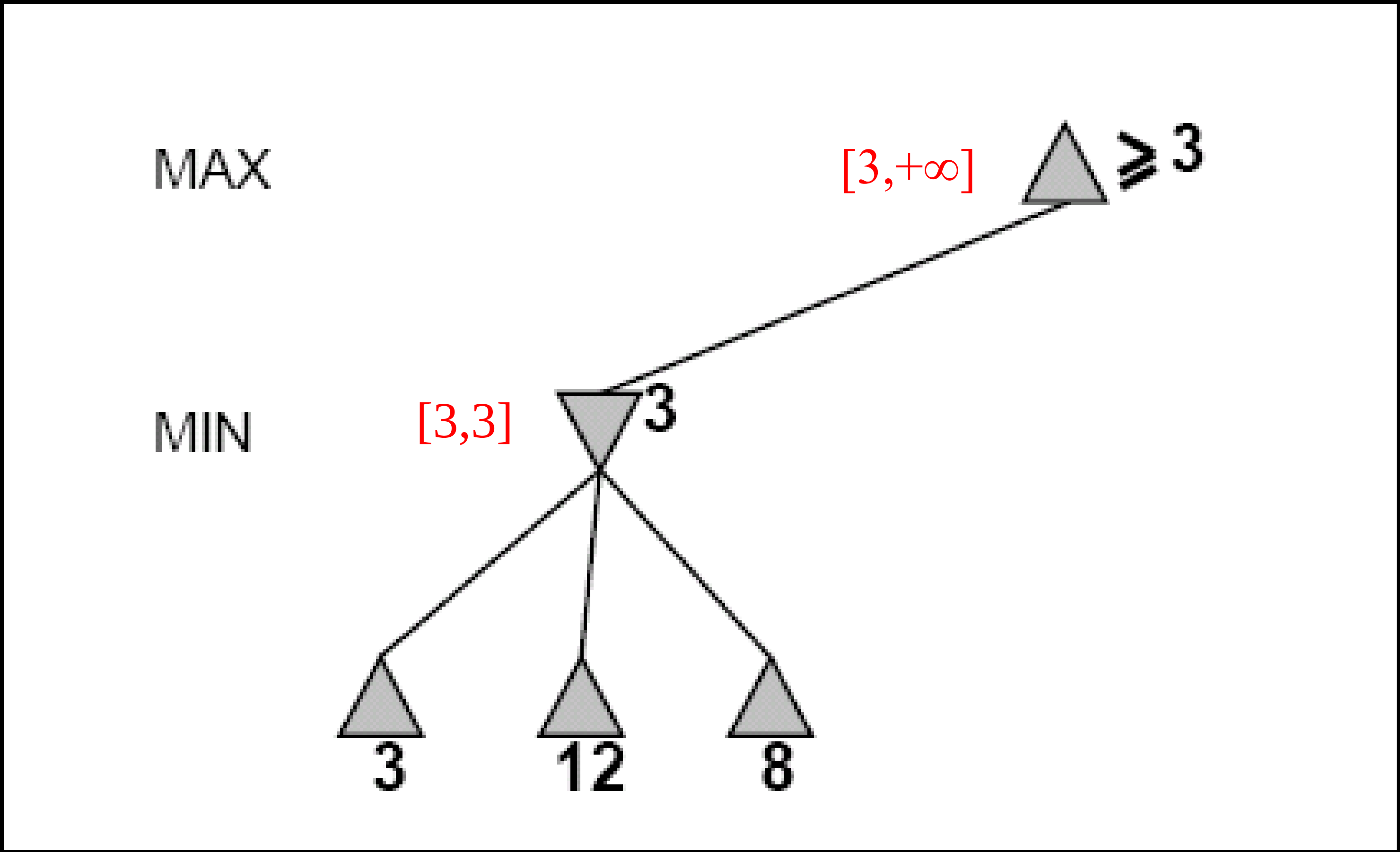


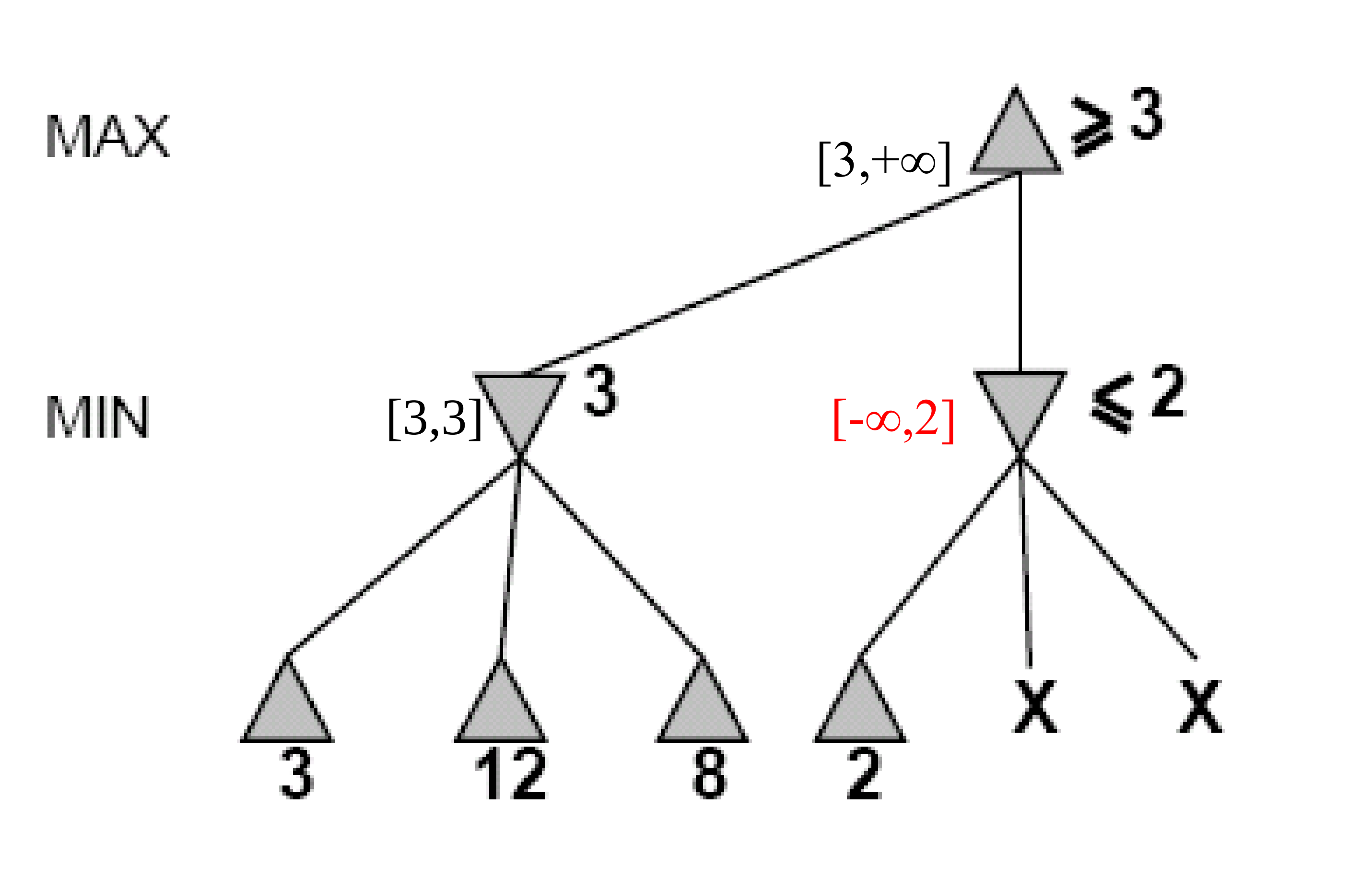
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)

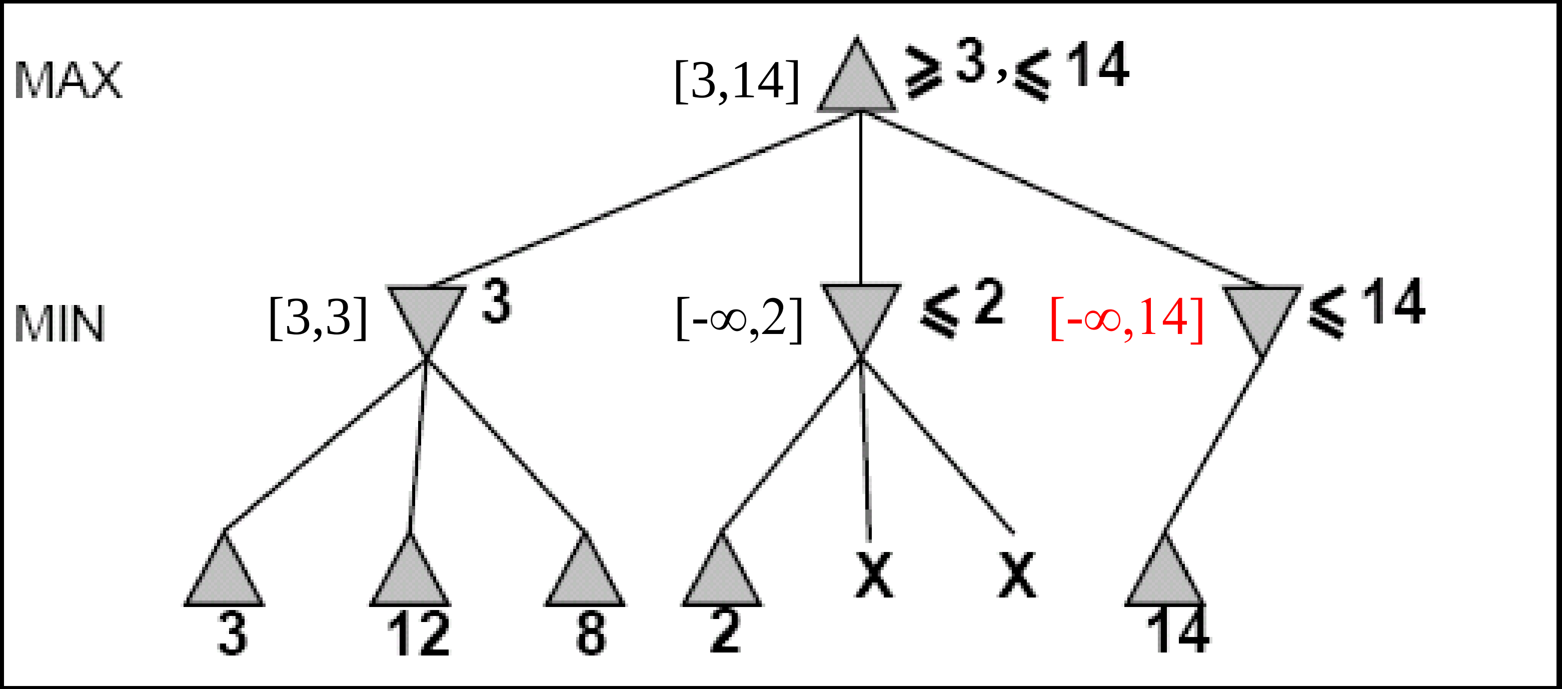



MAX
MIN
[3,+∞]
≥ 3
[3,3]
3
3
12
8

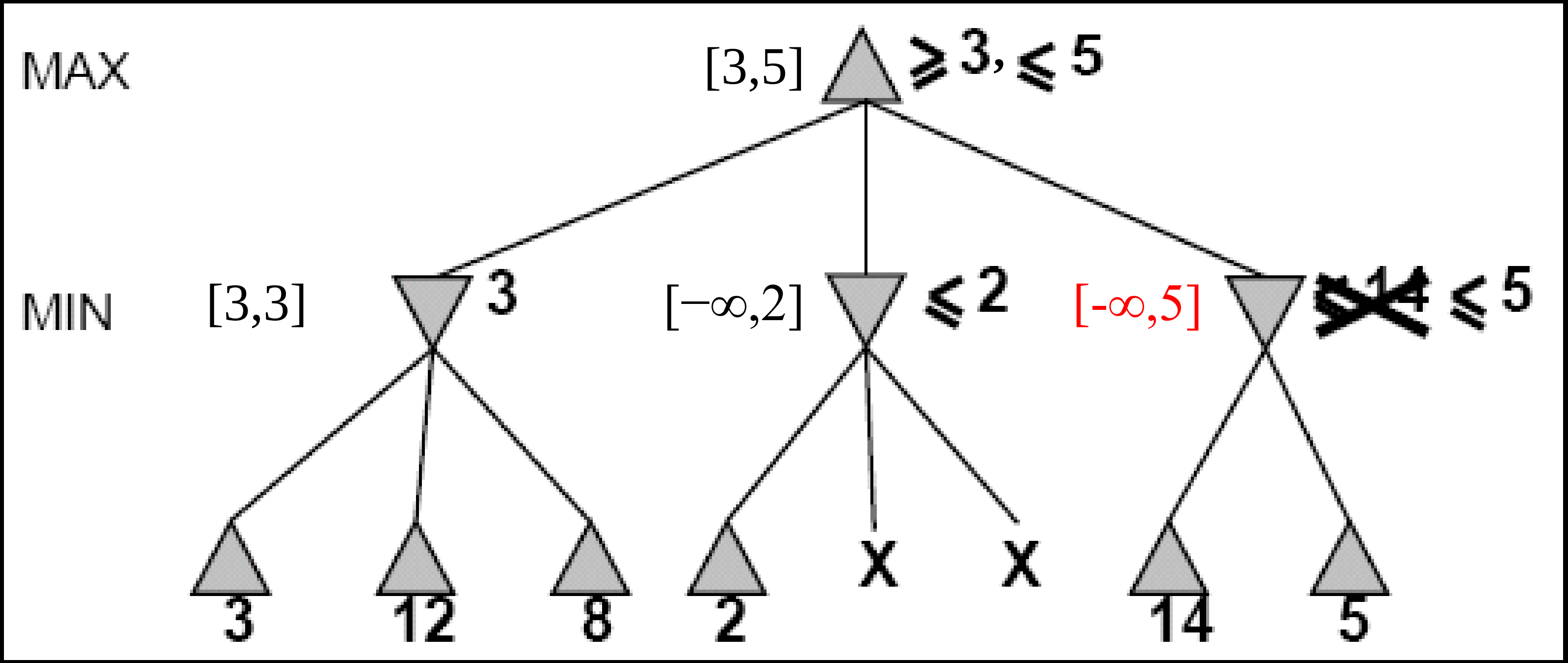


MAX
[3,+∞]
≥ 3
MIN
[3,3]
3
[-∞,2]
≤ 2
3
12
8
2
X
X

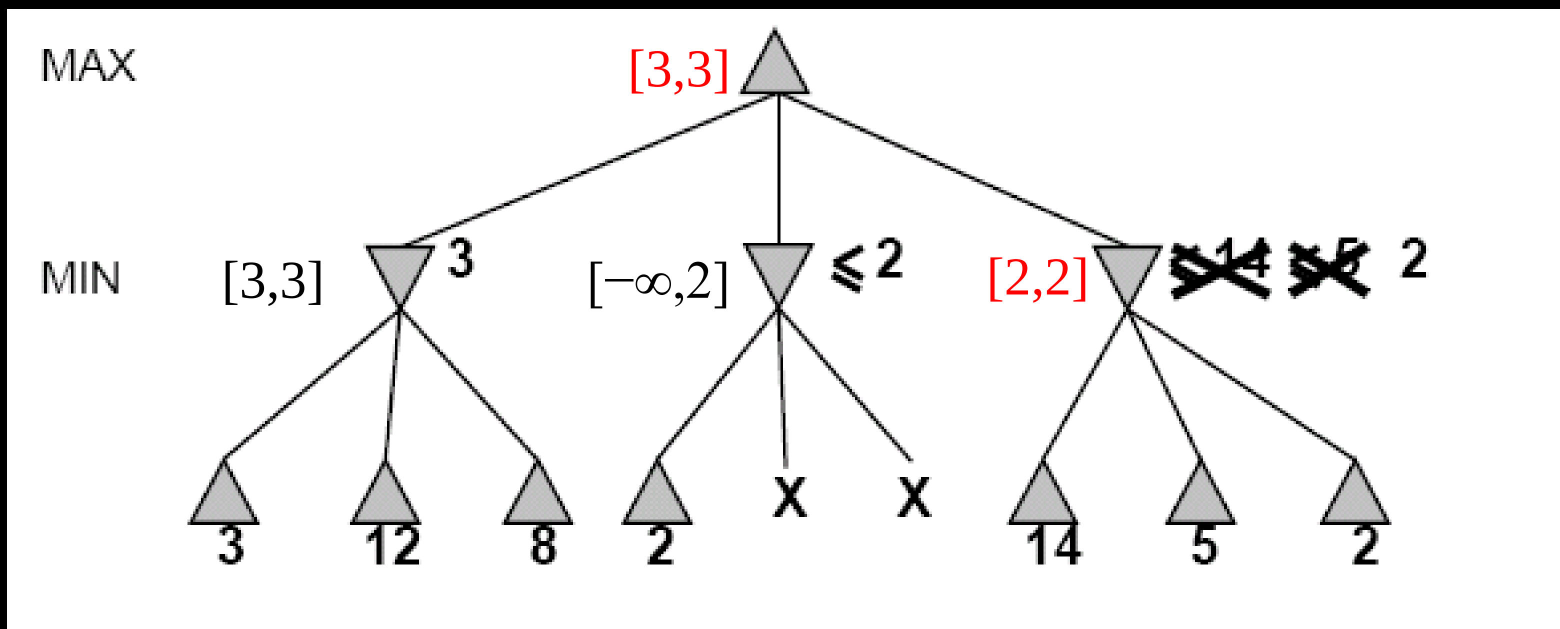
Alpha-Beta Example (continued)



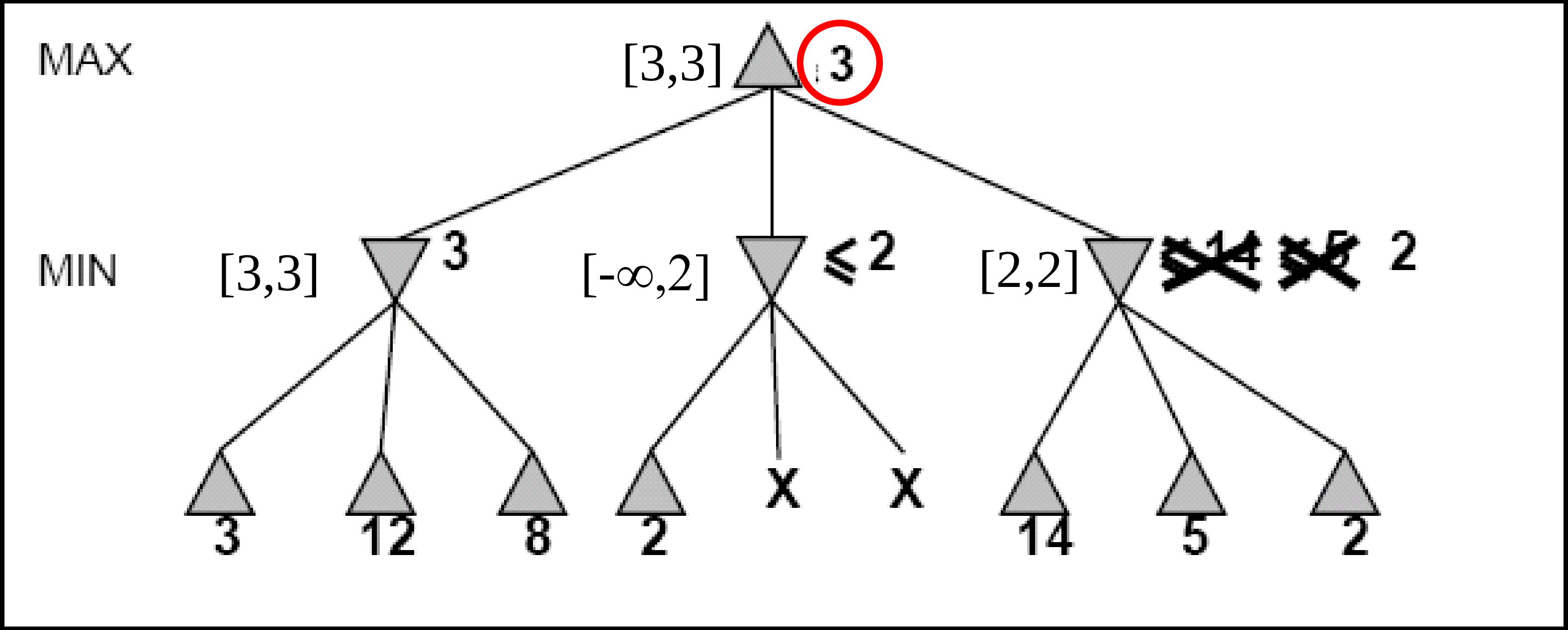
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)



Alpha-Beta Example (continued)



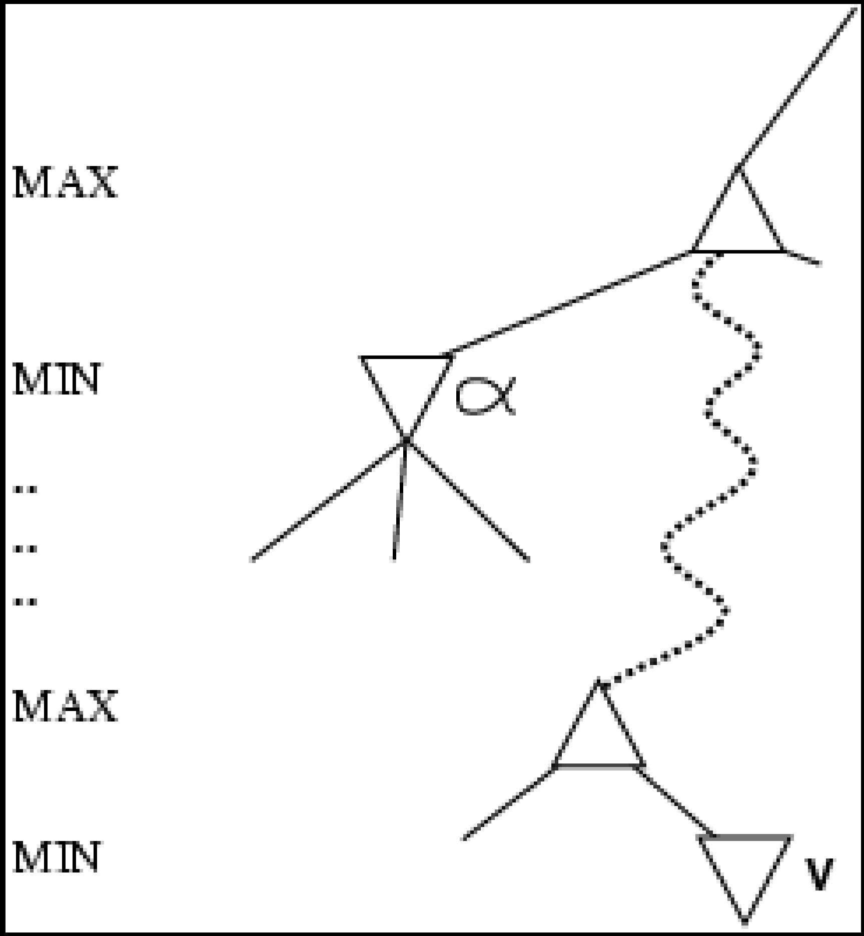
Properties of α-β

- Pruning does not affect final result
-
- However, effectiveness of pruning affected by…?

- What impact can it have on running time?

Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If *v* is worse than α, *max* will avoid it
  → prune that branch
- Define β similarly for *min*



```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game

   v ← MAX-VALUE(state, −∞, +∞)
   return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state

   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s, α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state

   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← +∞
   for a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s, α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Problems with AB Pruning?

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

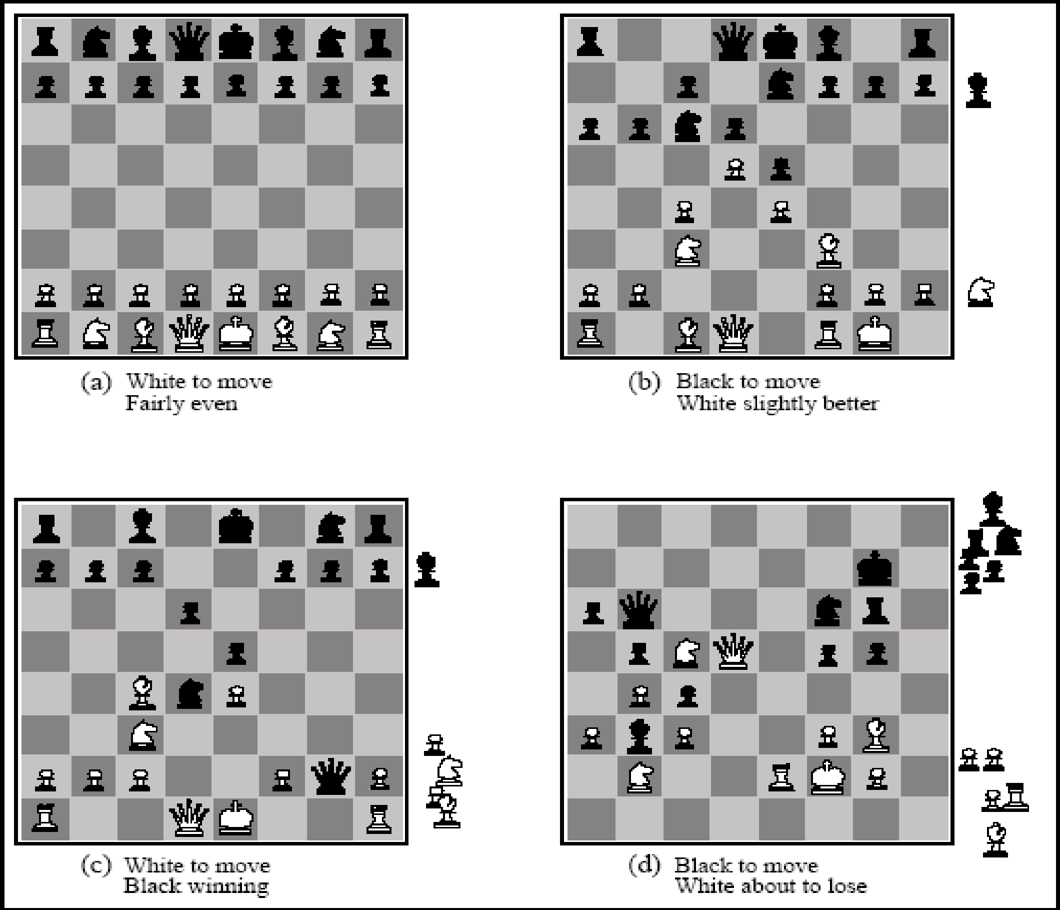
- Change:
  - if TERMINAL-TEST(state) then return UTILITY(state)
- into
  - if CUTOFF-TEST(state,depth) then return EVAL(state)

- 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.

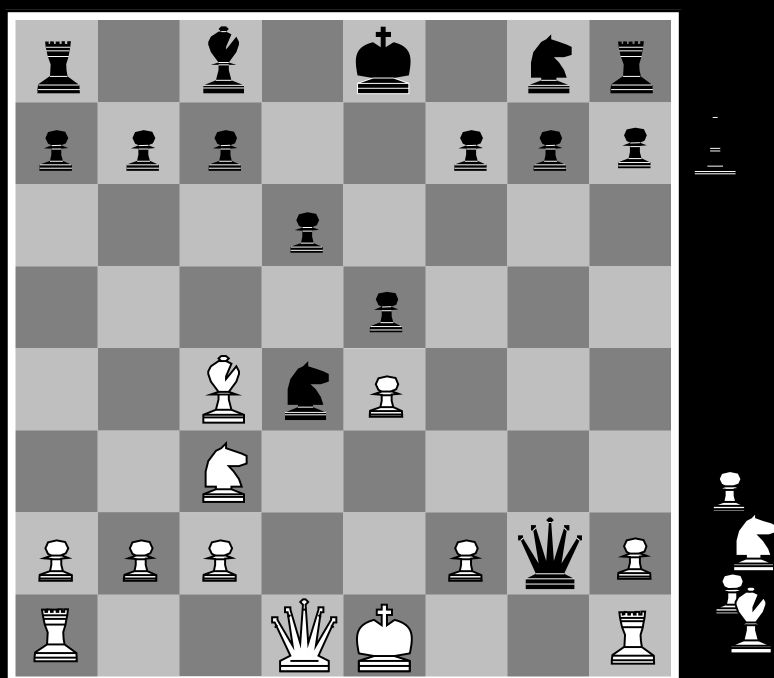
- Idea: produce an estimate of the expected utility of the game from a given position.
- Performance depends on quality of EVAL.
- Requirements:
  - EVAL should order terminal-nodes in the same way as UTILITY.
  - Computation may not take too long.
  - For non-terminal states the EVAL should be strongly 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.

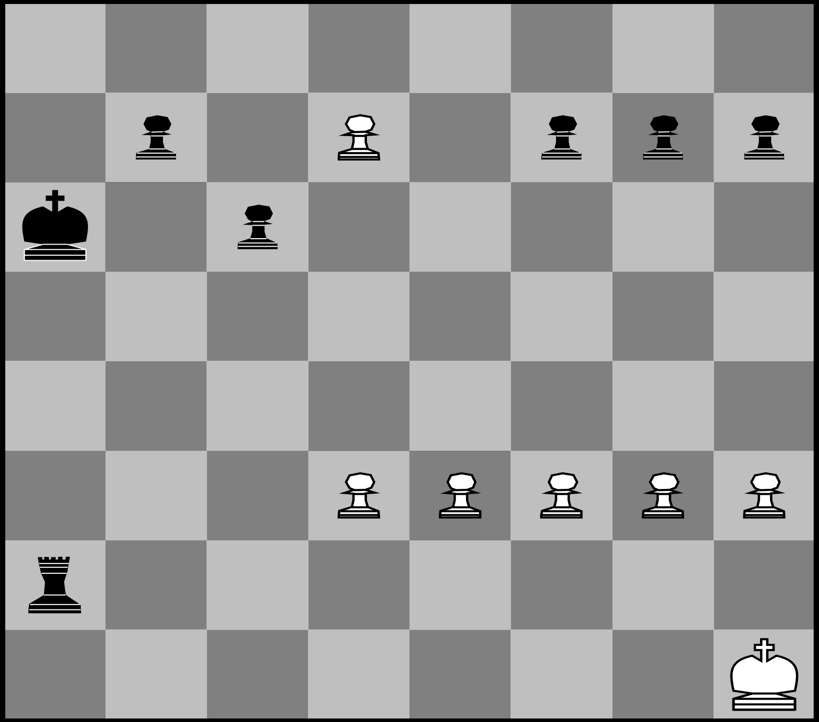
Heuristic EVAL example

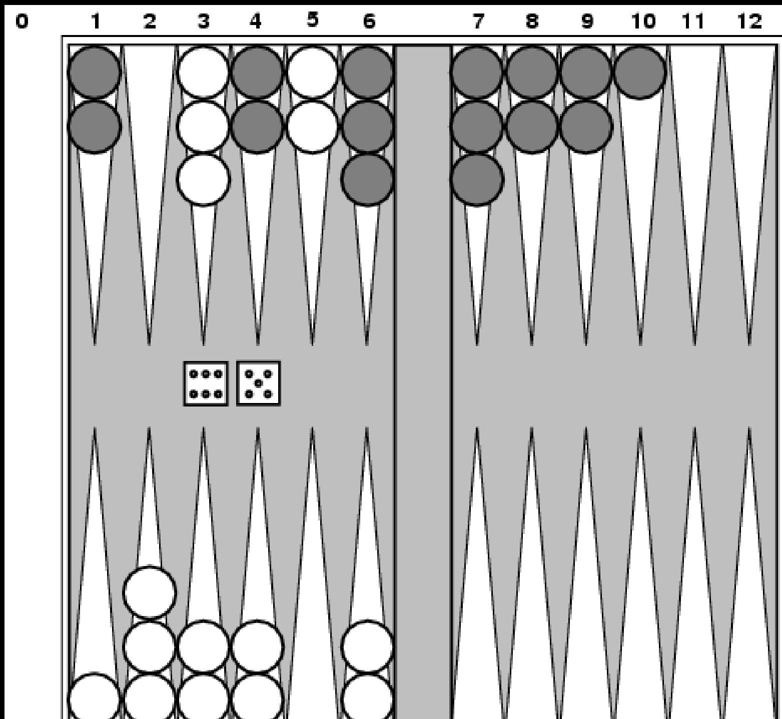


$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

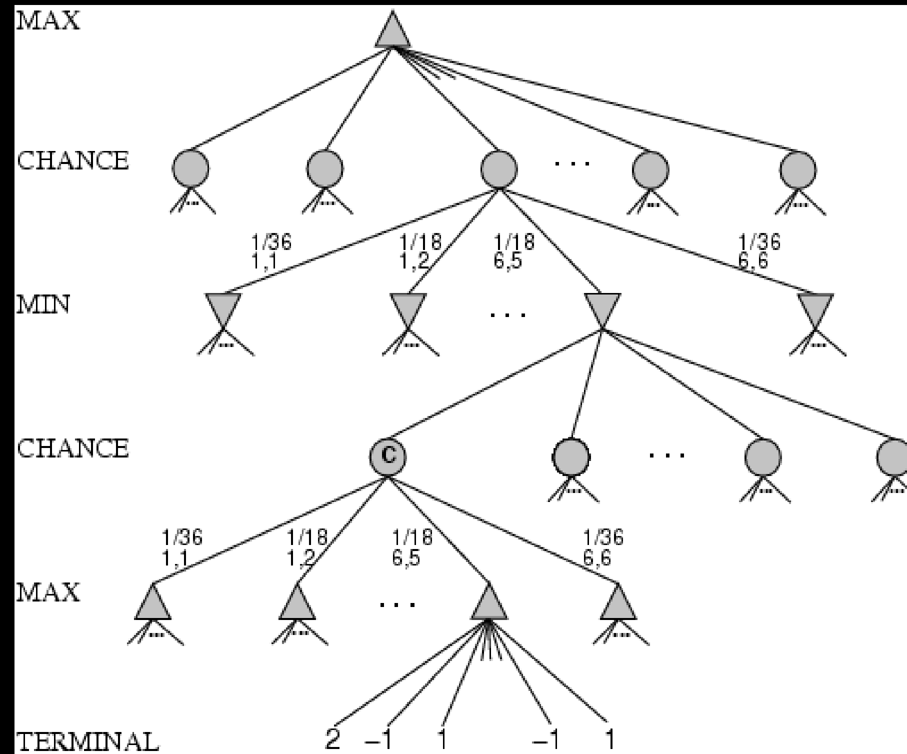


Fixed depth search thinks it can avoid the queening move

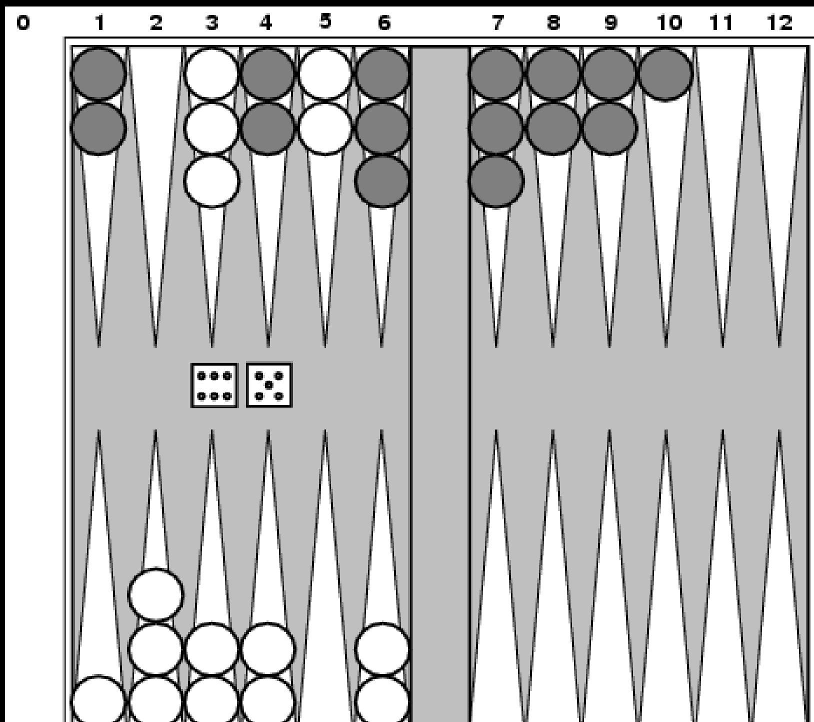


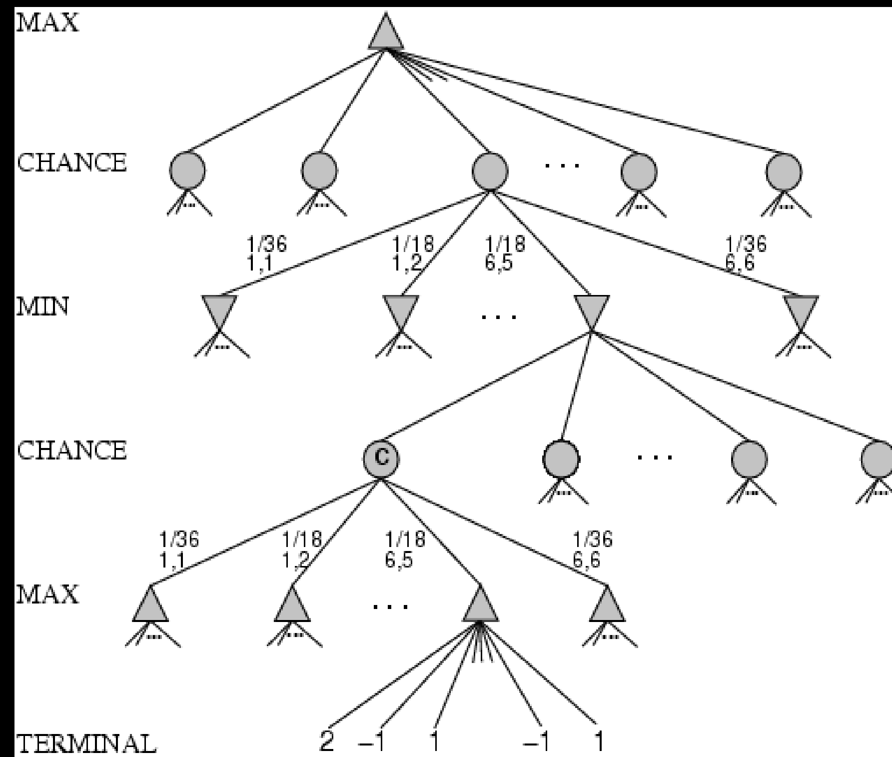


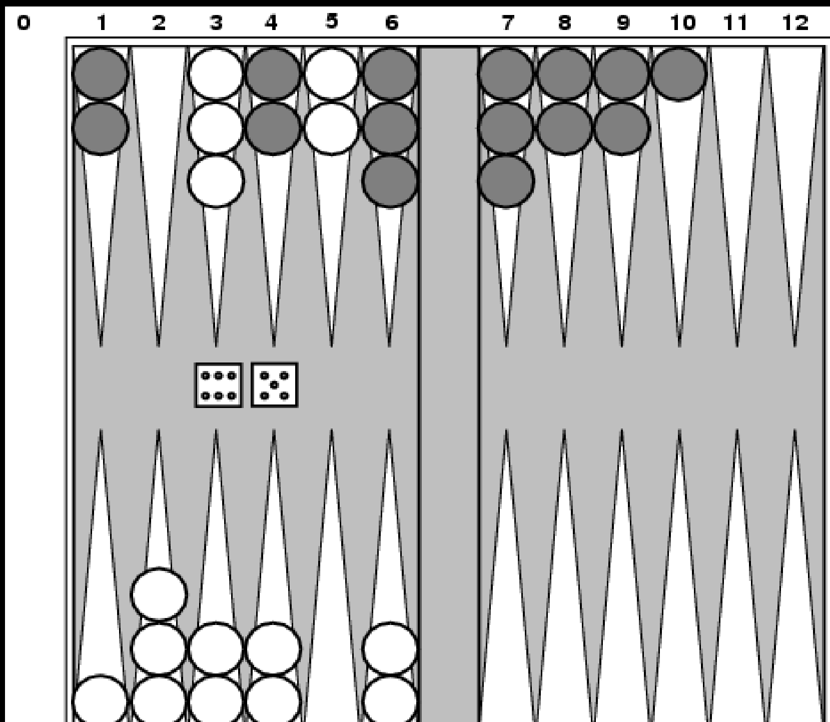
0
1
2
3
4
5
6
7
8
9
10
11
12

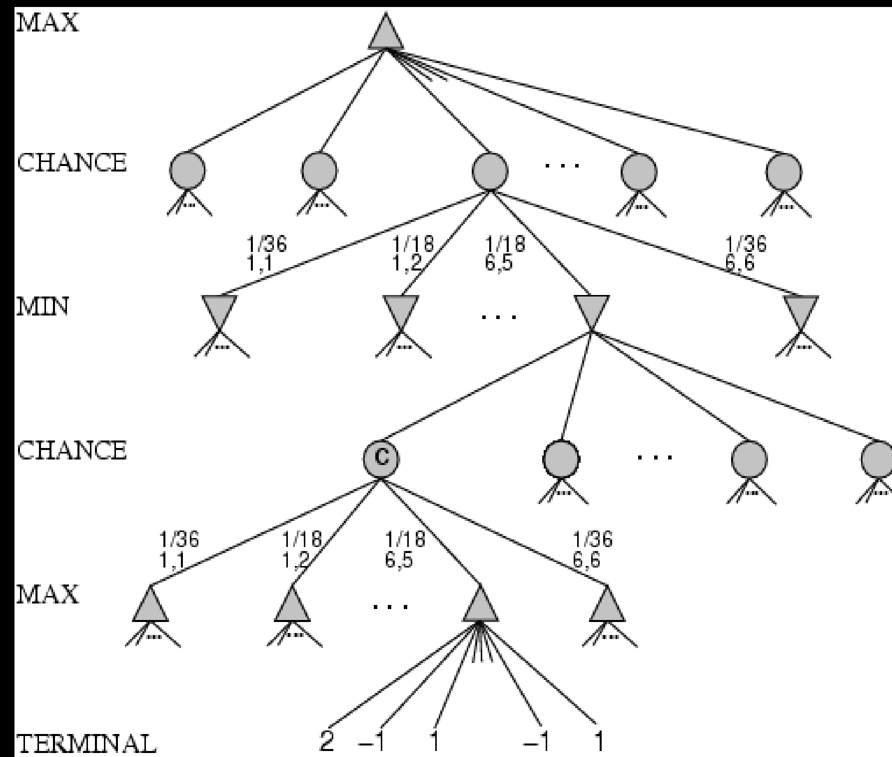


MAX
CHANCE
1/36 1,1
1/18 1,2
1/18 6,5
1/36 6,6
MIN
CHANCE
C
1/36 1,1
1/18 1,2
1/18 6,5
1/36 6,6
MAX
TERMINAL
2
–1
1
–1
1


0
1
2
3
4
5
6
7
8
9
10
11
12


MAX
CHANCE
MIN
CHANCE
MAX
TERMINAL
1/36
1,1
1/18
1,2
1/18
6,5
1/36
6,6
C
2
–1
1
–1
1


0
1
2
3
4
5
6
7
8
9
10
11
12


MAX
CHANCE
MIN
CHANCE
MAX
TERMINAL
1/36
1,1
1/18
1,2
1/18
6,5
1/36
6,6
C
2
–1
1
–1
1

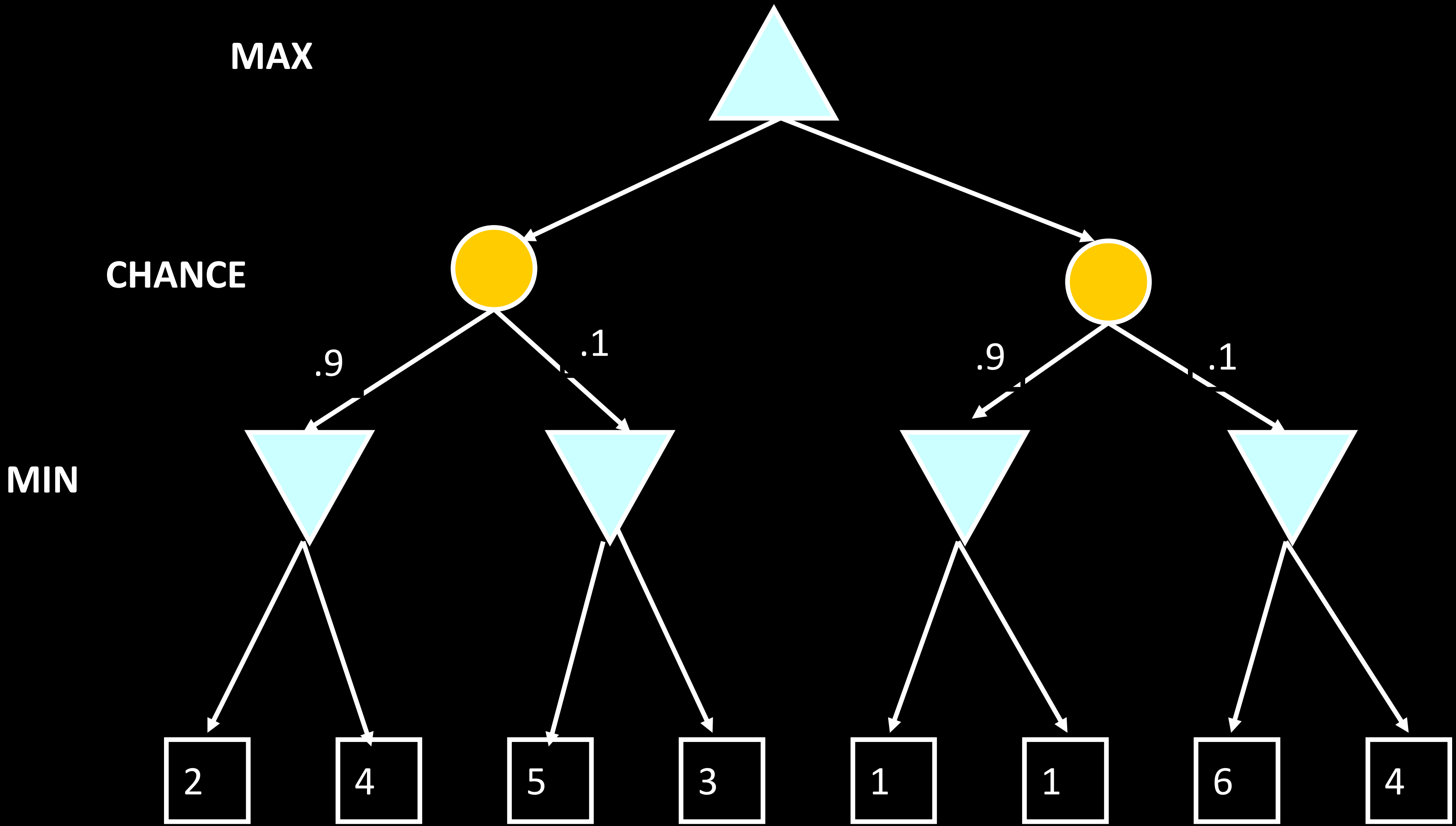
Expecti minimax value

EXPECTI-MINIMAX-VALUE(n)=

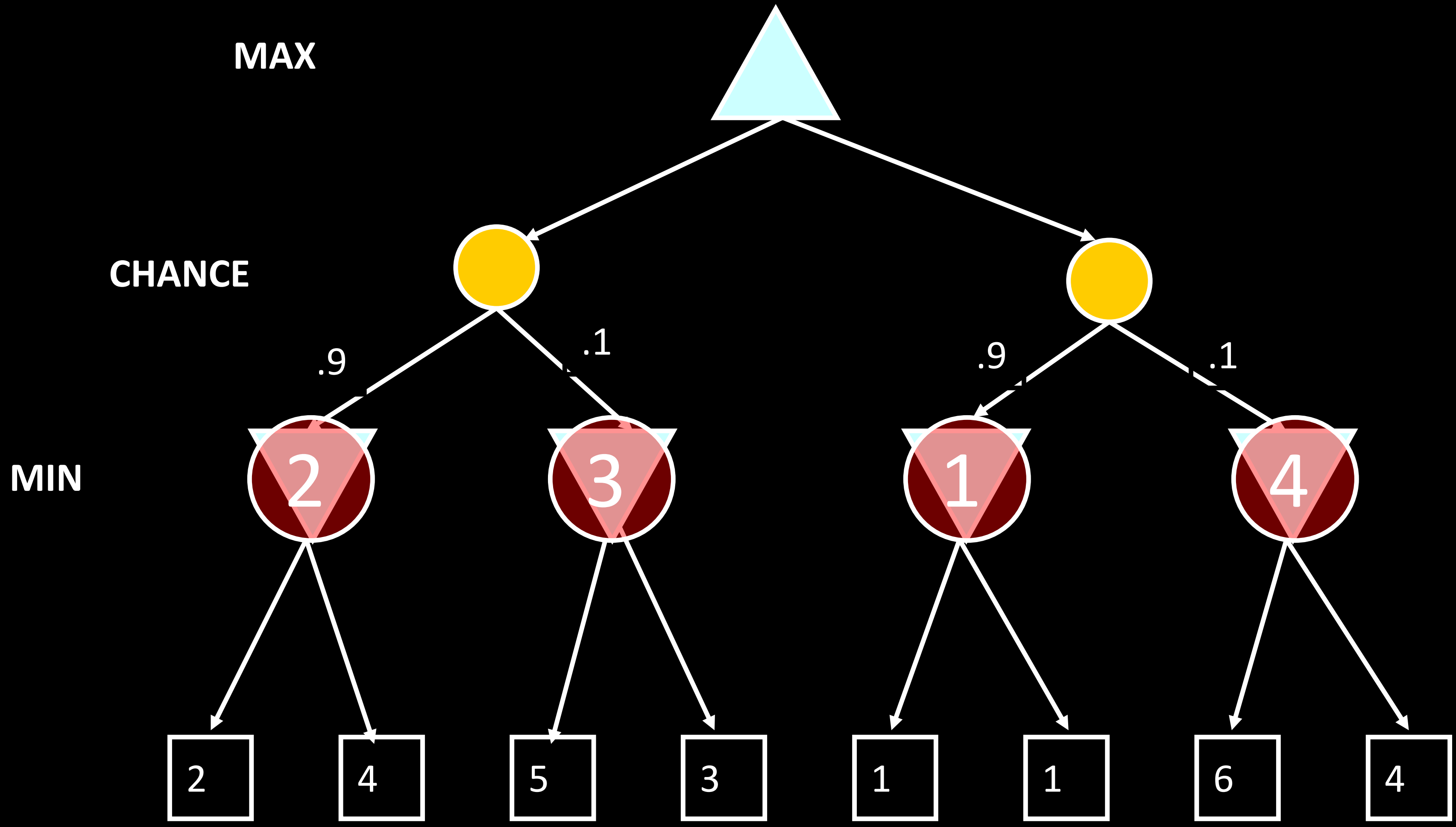
- UTILITY(n) If n is a terminal
- $\max_{s \in successors(n)}$ MINIMAX-VALUE(s) If n is a max node
- $\min_{s \in successors(n)}$ MINIMAX-VALUE(s) If n is a min node
- $\sum_{s \in successors(n)} P(s)$. EXPECTIMINIMAX(s) If n is a chance node

These equations can be backed-up recursively all the way to the root of the game tree.

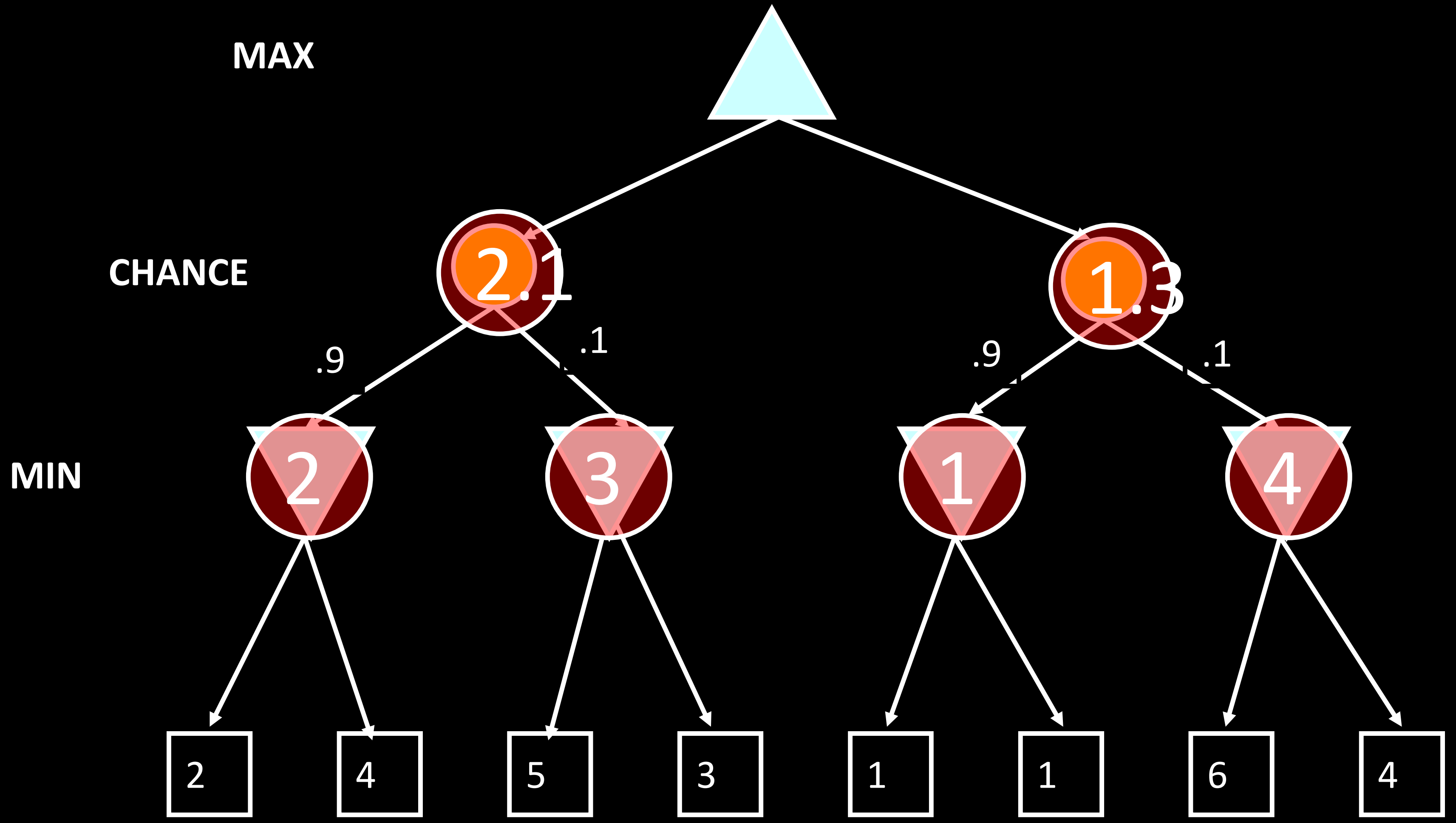
EXPECTEDMINIMAX example



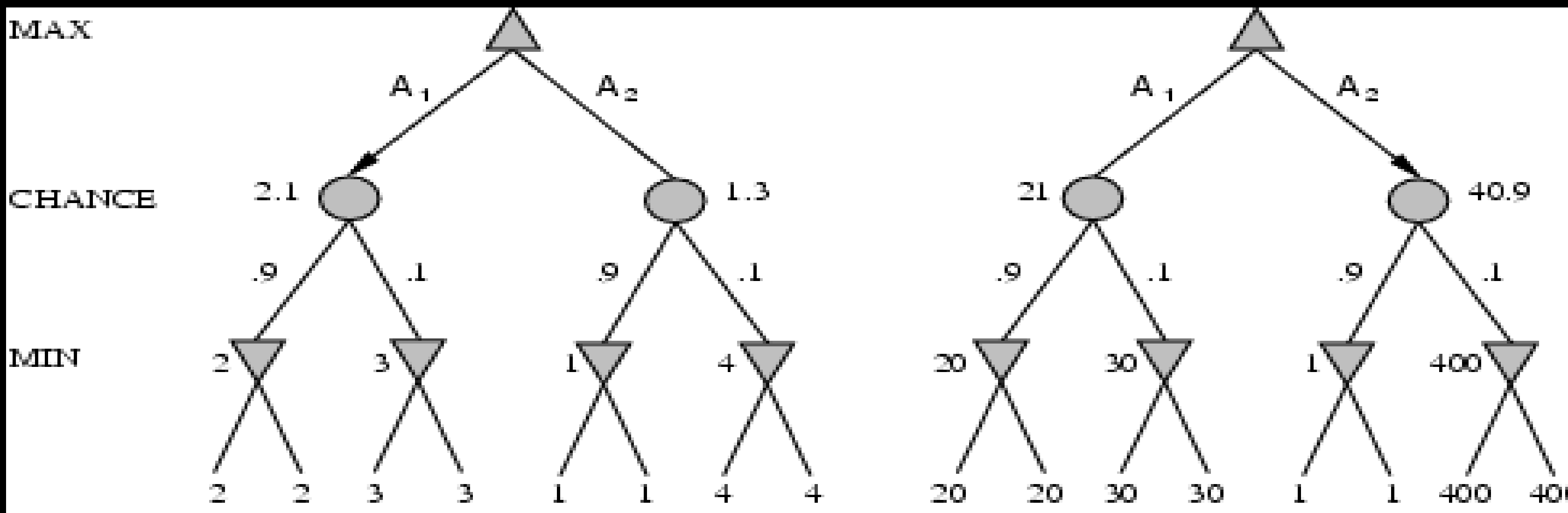
EXPECTIMINIMAX example

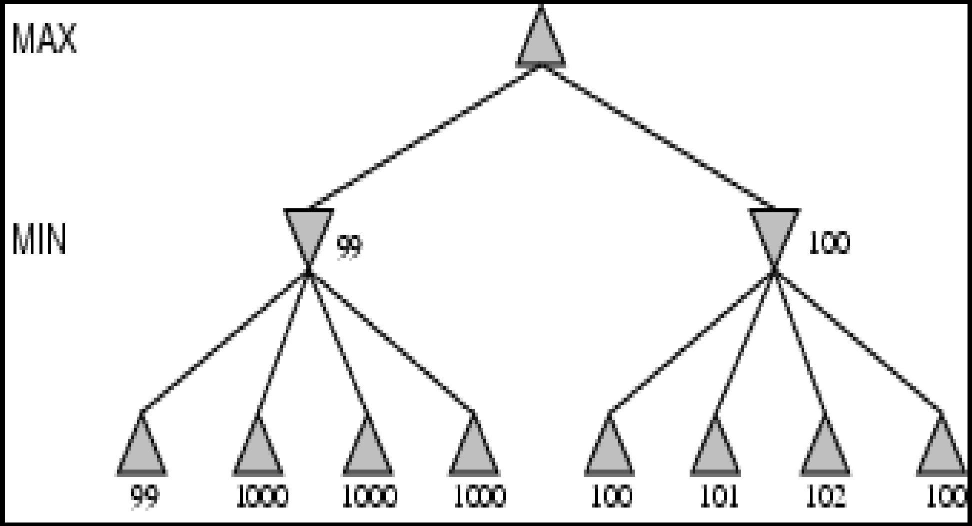


EXPECTIMINIMAX example



Position evaluation with chance nodes





- What will minimax do here?
- Is that OK?
- What might you do instead?