

# Chapter 5: Adversarial Search

---

These slides are adapted from AI Berkeley course and Russell and Norvig textbook.

# Adversarial Search (Games)

---

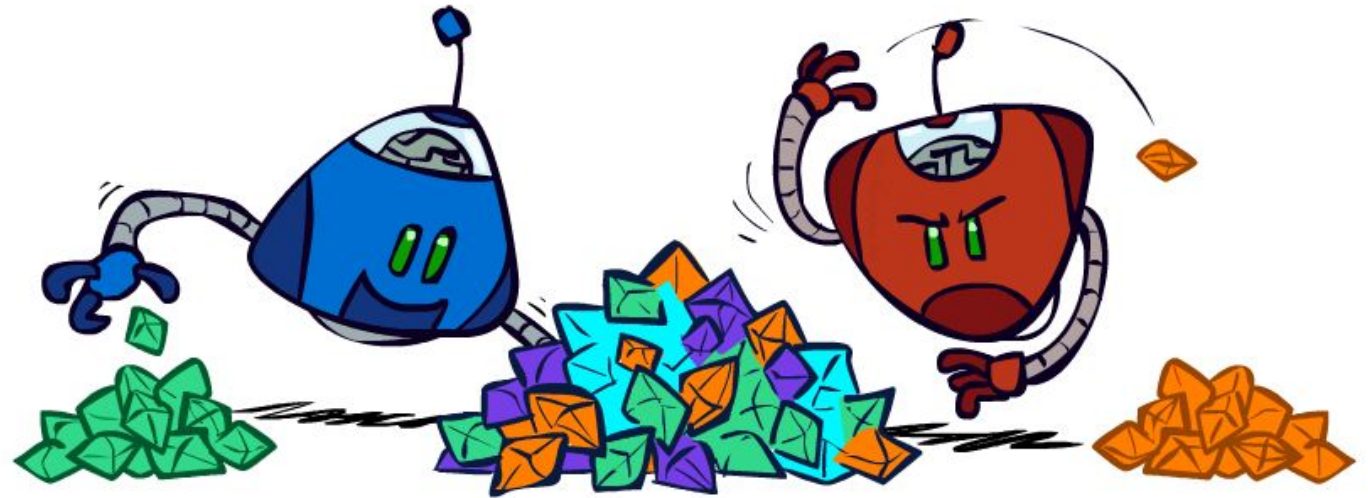
- Agents are in **competitive** environments (the agents' goals are in conflict).
- Types of Games
  - Deterministic vs. stochastic
  - One, two, or more players
  - Zero sum
    - The utility values at the end of the game are always equal and opposite like chess.
  - Perfect information (deterministic and fully-observable)

# Game Formulation

---

- A game can be formally defined as a kind of search problem with the following elements:
  - Initial State  $S_0$  which specifies how the game is set up at the start.
  - Players:  $P=\{1...N\}$ : Defines which player has the move in a state.
  - Actions:  $A$ : Returns the set of legal moves in a state.
  - Transition Function:  $S \times A \rightarrow S$
  - Terminal Test:  $S \rightarrow \{\text{true if the game is over, otherwise false}\}$ 
    - States where the game has ended are called **terminal states**.
  - **Utilities**:  $S \times P \rightarrow R$ 
    - A utility function defines the **final numeric value for a game that ends in a terminal state  $s$  for a player  $p$** .
    - For example, in chess, the outcome is a **win, loss, or draw**, with values **+1, 0, or 1/2**.
- The game solution for a player is a **strategy**:  $S \rightarrow A$  to define its move for a given state.
- An **optimal** strategy **leads to outcomes at least as good as any other strategy when one is playing an infallible opponent**.

# Zero-Sum Games



## Zero-Sum Games

- Agents have **opposite utilities** (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes
- **Adversarial**, pure competition

## General Games

- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible

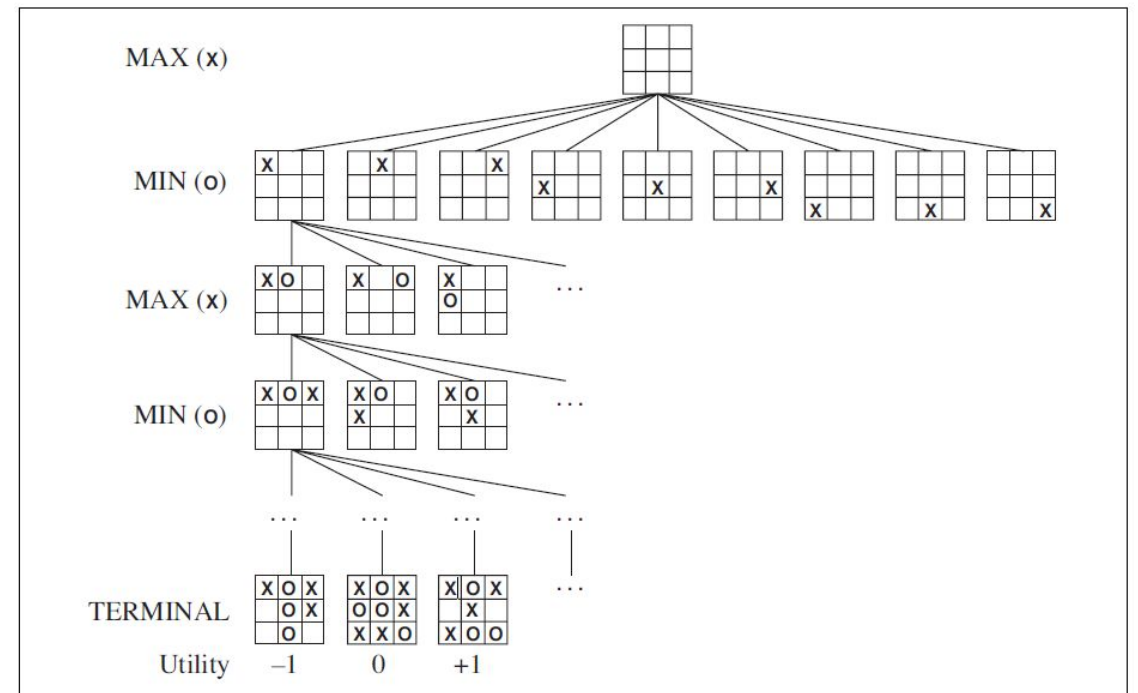
# Deterministic Zero-sum Game Setup

---

- We first consider games with two players (MAX and MIN)
- MAX moves first, and then they take turns moving until the game is over.
- At the end of the game, points are awarded to the winning player and penalties are given to the loser.

# Partial Game Tree Example

- Tic-tac-toe's game tree has  $9! = 362,880$  terminal nodes.
- For chess there are over  $10^{40}$  nodes!!
- How a player could choose an optimal move?!
- Examine **enough** nodes to allow him to determine what move to make as we will see 😊



solved game m3naha en ana 3rft eh hya el strategy elly lw est5dmtha, most7el el odamy yeksbnny, lakn lw howa byl3b perfect, fa 3l a2al hat3adel m3ah.

el minmax by-assume en el two players are playing optimal.

fa hwa el max hyo7 yshof fl current state eh, w yshof el opponent hyfkr ezay, fa y2ol ana lw khadt el action ai, el opponent hyfkr bl tre2a de, w ye3ml 1,2,3,4,5 w yekhtar el min bta3ha, fa ana akny bl3b shatarang, w a7sb 3adad n2lat mo3yna, w 3la asasha, ashuf hwa a a7sn 7aga 3nd 3dowy a5asro beha, w a7sn 7aga 3ndy, w al3bha.

# Minimax values

Given a game tree, the optimal strategy can be determined from the **minimax value**

of each node, which we write as **MINIMAX(n)**.

- The minimax value of a node is the utility (for MAX) of being in the corresponding state, *assuming that both players play optimally* from there to the end of the game.
- The minimax value for a state  $s$  is calculated as:

$\text{MINIMAX}(s) =$

assume we are max.

1st iter -> iterate over all possible actions, then recursively, compute the minimax, assuming that min is going to paly, fa hayb2a 3ndk kaza option ngrbhom kolohom, w nakhud akbur value mwgoda.  
w da 34an ana ka max, b7awl amaximize 3la ad ma a2dr el value elly ana l2tha.

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

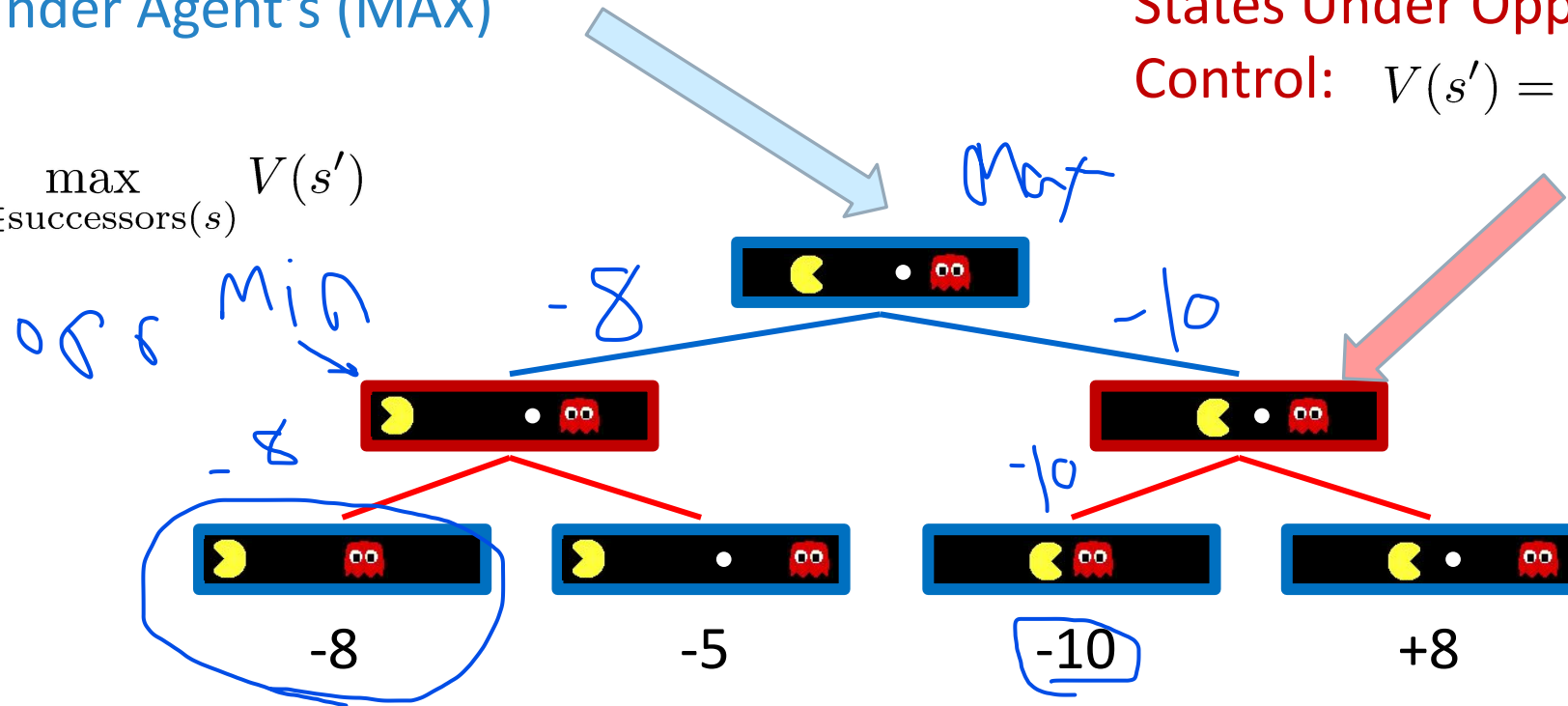
# Minimax Values- Example Pacman Game

States Under Agent's (MAX)  
Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's (MIN)

Control:  $V(s') = \min_{s \in \text{successors}(s')} V(s)$



assume packman is max.  
so in the first iteration, it will check all its possible cases, and maximize on it, recursively calling minimax on the cases, but reversing the turn.

so in the second iteration we will look at our selves as min, and try to minimize, and this makes sense, because we put ourselves in our opponents shoes, and check what is his optimal choice that he should take now.

Terminal States:

$$V(s) = \text{known}$$



# The Minimax Algorithm

---

**function** MINIMAX-DECISION(*state*) *returns an action*  
    **return**  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(\text{state}, a))$

---

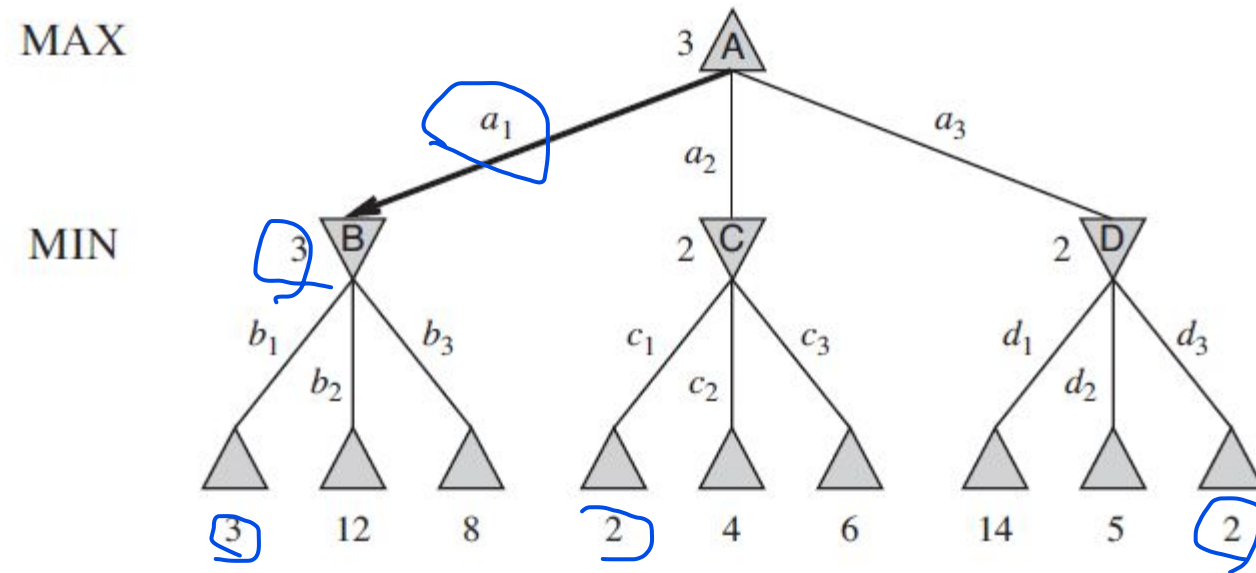
**function** MAX-VALUE(*state*) *returns a utility value*  
    **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)  
     $v \leftarrow -\infty$   
    **for each** *a* **in** ACTIONS(*state*) **do**  
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$   
    **return** *v*

---

**function** MIN-VALUE(*state*) *returns a utility value*  
    **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)  
     $v \leftarrow \infty$   
    **for each** *a* **in** ACTIONS(*state*) **do**  
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$   
    **return** *v*

# Example

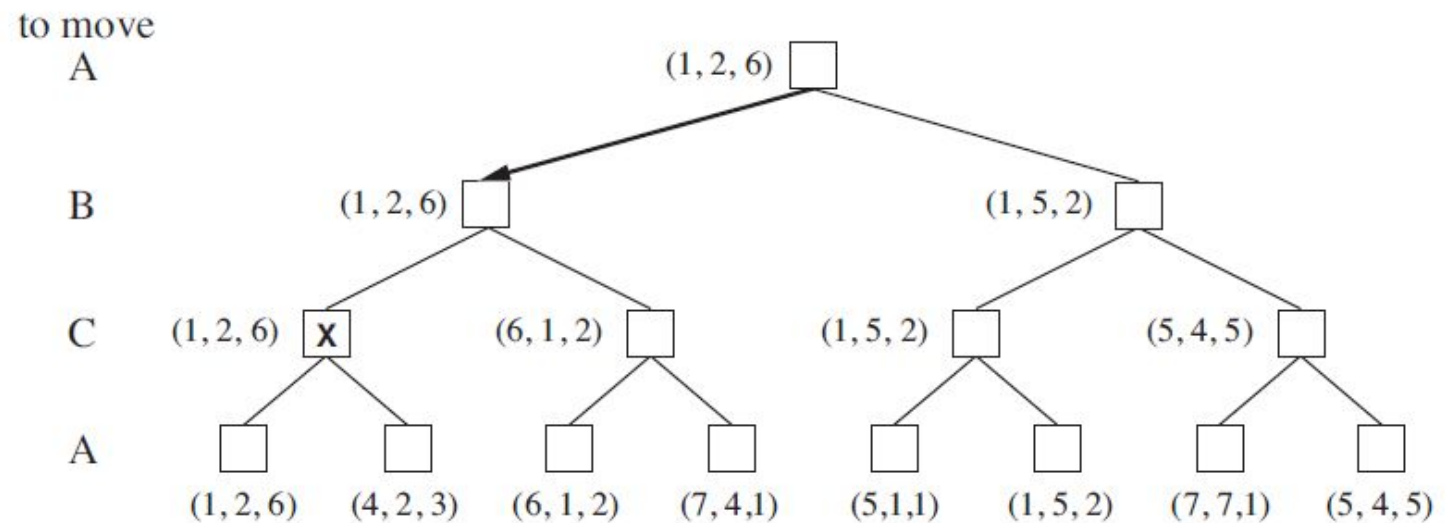
---



this is the optimization of the coming algorithms.

# Minimax for Multi-player Games

- Terminals have utility tuples.
- Node values are utility tuples.
- Each player maximizes its own component.
- Can give rise to cooperation and competition dynamically.



**Figure 5.4** The first three plies of a game tree with three players ( $A$ ,  $B$ ,  $C$ ). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

hena kol whed byfkr fe nfso bs, w m7dsh mohtm bl tany.  
lw ana khadt krar 7asen el akhren, helw mfesh mushkela, tb lw darhom, ana mkansh asdy, bs msh muhem brdu.

# Minimax Properties

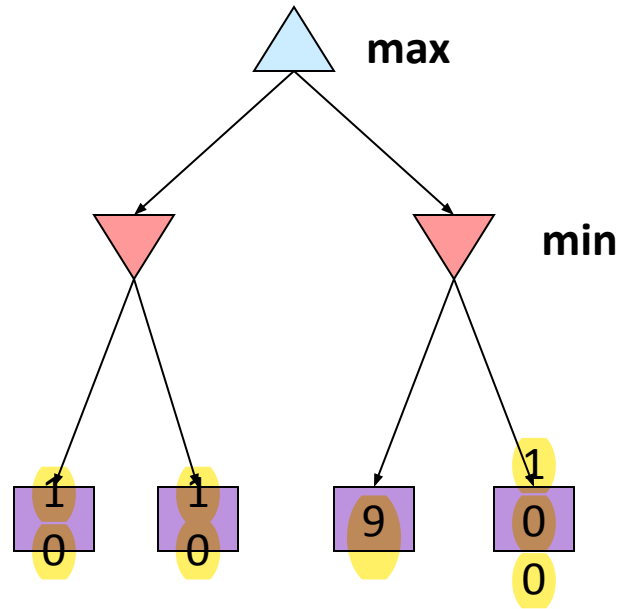
---

- Complete: Yes, if the tree is finite
- Optimal: Yes, but against an optimal opponent.
- Time Complexity:  $O(b^m)$  □ For chess,  $b=35$ ,  $m=100$ , infeasible time!
- Space Complexity:  $O(bm)$  (depth-first exploration)

$b$  is the branching factor (legal moves at each point),  $m$  is the maximum depth of the tree.

# Minimax Properties (cont.)

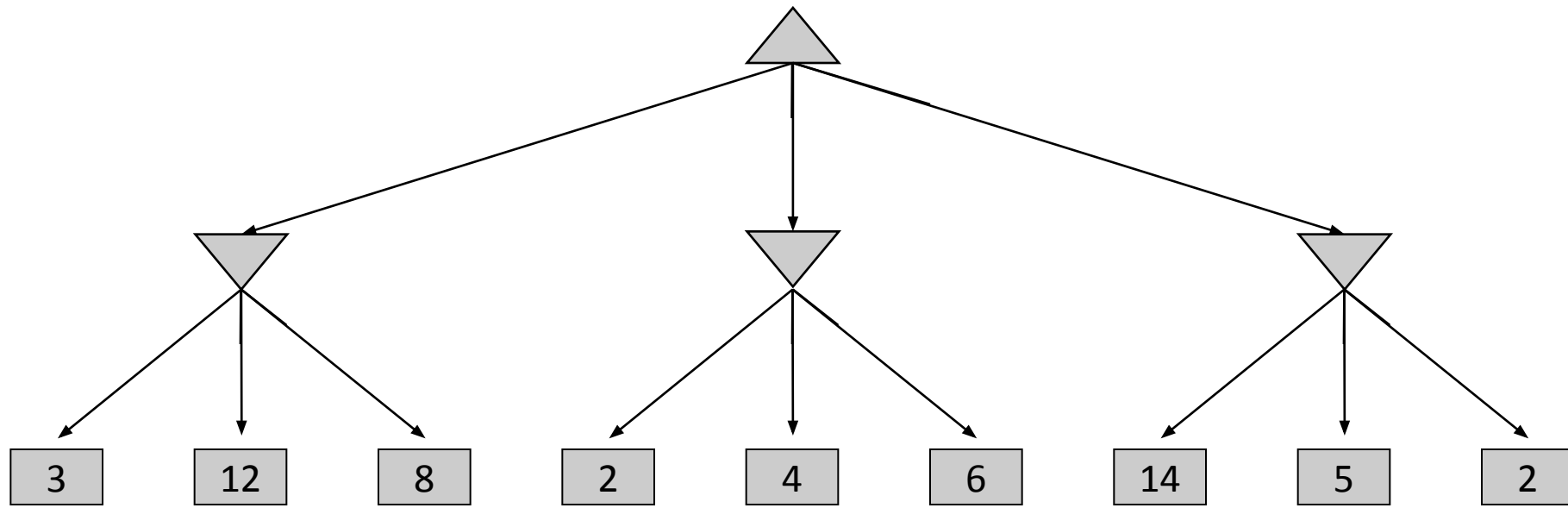
---



What if min plays non-optimally?

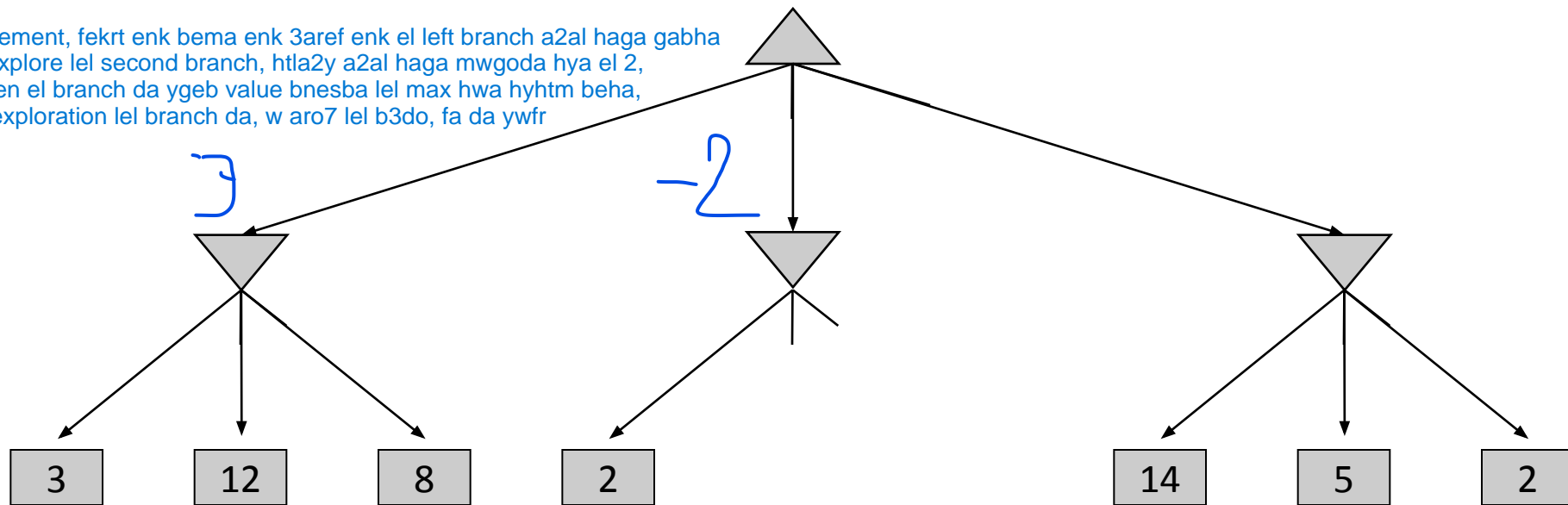
# Minimax Example

---



# Pruning?

hena feh 7etet enhancement, fekr enk bema enk 3aref enk el left branch a2al haga gabha kant 3, w wnta bt3ml explore lel second branch, htla2y a2al haga mwgoda hya el 2, fa keda keda most7el en el branch da ygeb value bnesba lel max hwa hyhtm beha, fa hena momken abtl exploration lel branch da, w aro7 lel b3do, fa da ywfr 3lya shwyt w2t.



# Alpha-Beta Pruning

---

- Alpha-beta pruning computes the correct minimax decision but prunes away branches that cannot possibly influence the final decision.

bygeb nfs el results bta3t el minimax, lakn asr3, fa lazmn yetl3 nfs el results bta3t el minimax, 34an lw mtl3hash, te3rf enk 3aml 7aga ghlt.

.



# Alpha-Beta Pruning

## General configuration

- We're computing the MIN-VALUE at some node  $n$
- We're looping over  $n$ 's children
- $n$ 's estimate of the childrens' min is dropping
- Who cares about  $n$ 's value? MAX
- Let  $a$  be the best value that MAX can get at any choice point along the current path from the root
- If  $n$  becomes worse than  $a$ , MAX will avoid it, so we can stop considering  $n$ 's other children (it's already bad enough that it won't be played)

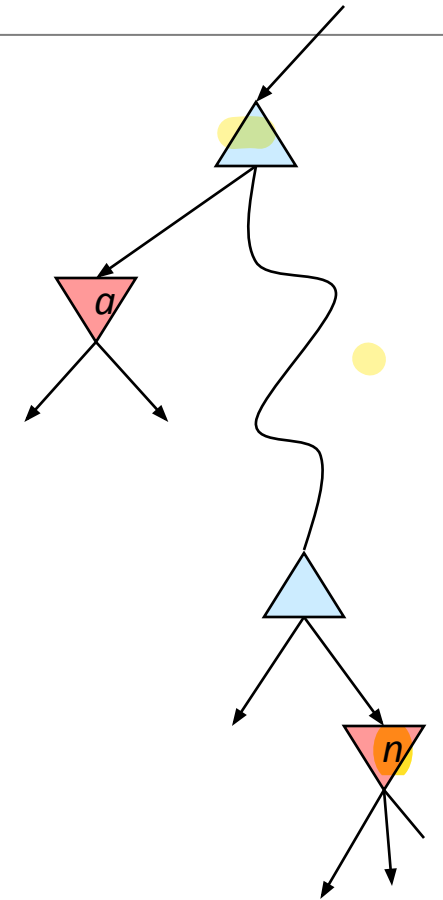
MAX

MIN

⋮

MAX

MIN



# Alpha-Beta Pruning Algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action  
   $v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$   
  return the action in  $\text{ACTIONS}(\text{state})$  with value  $v$ 
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value  
  if  $\text{TERMINAL-TEST}(\text{state})$  then return  $\text{UTILITY}(\text{state})$   
   $v \leftarrow -\infty$   
  for each  $a$  in  $\text{ACTIONS}(\text{state})$  do  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$   
    if  $v \geq \beta$  then return  $v$   
     $\alpha \leftarrow \text{MAX}(\alpha, v)$   
  return  $v$ 
```

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value  
  if  $\text{TERMINAL-TEST}(\text{state})$  then return  $\text{UTILITY}(\text{state})$   
   $v \leftarrow +\infty$   
  for each  $a$  in  $\text{ACTIONS}(\text{state})$  do  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$   
    if  $v < \alpha$  then return  $v$   
     $\beta \leftarrow \text{MIN}(\beta, v)$   
  return  $v$ 
```

# Alpha-Beta Pruning

---

- $\alpha$  = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
- $\beta$  = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

# Alpha-Beta Pruning Properties

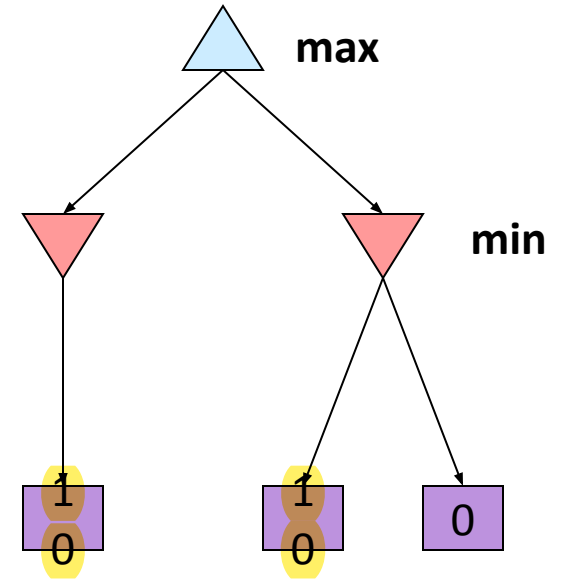
- The alpha-beta pruning has **no effect** on minimax value computed for the root!

- • Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value

- Good child ordering improves effectiveness of pruning

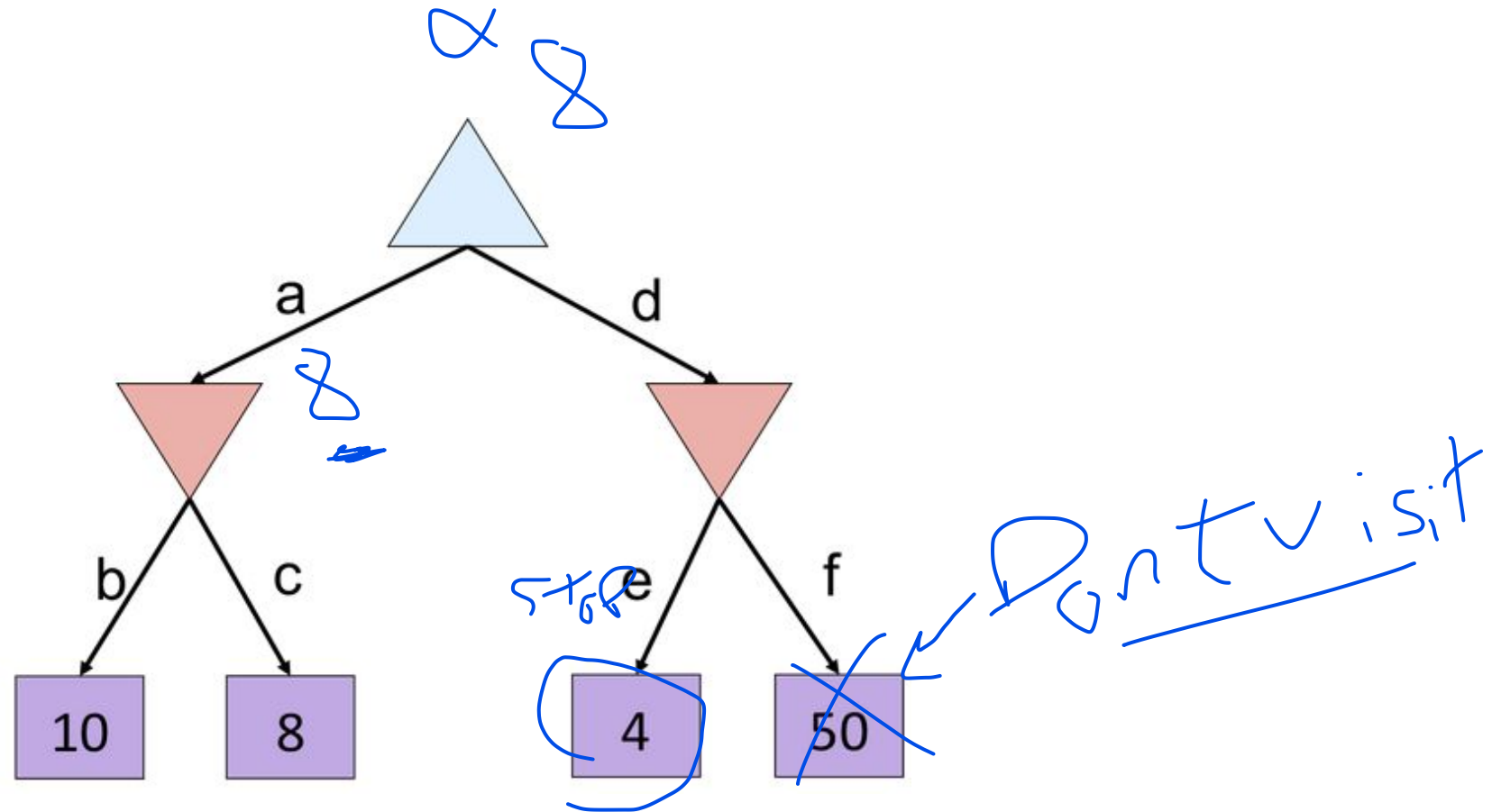
- With “perfect ordering”:
  - Time complexity drops to  $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless...

This is a simple example of **meta**reasoning (computing about what to compute)

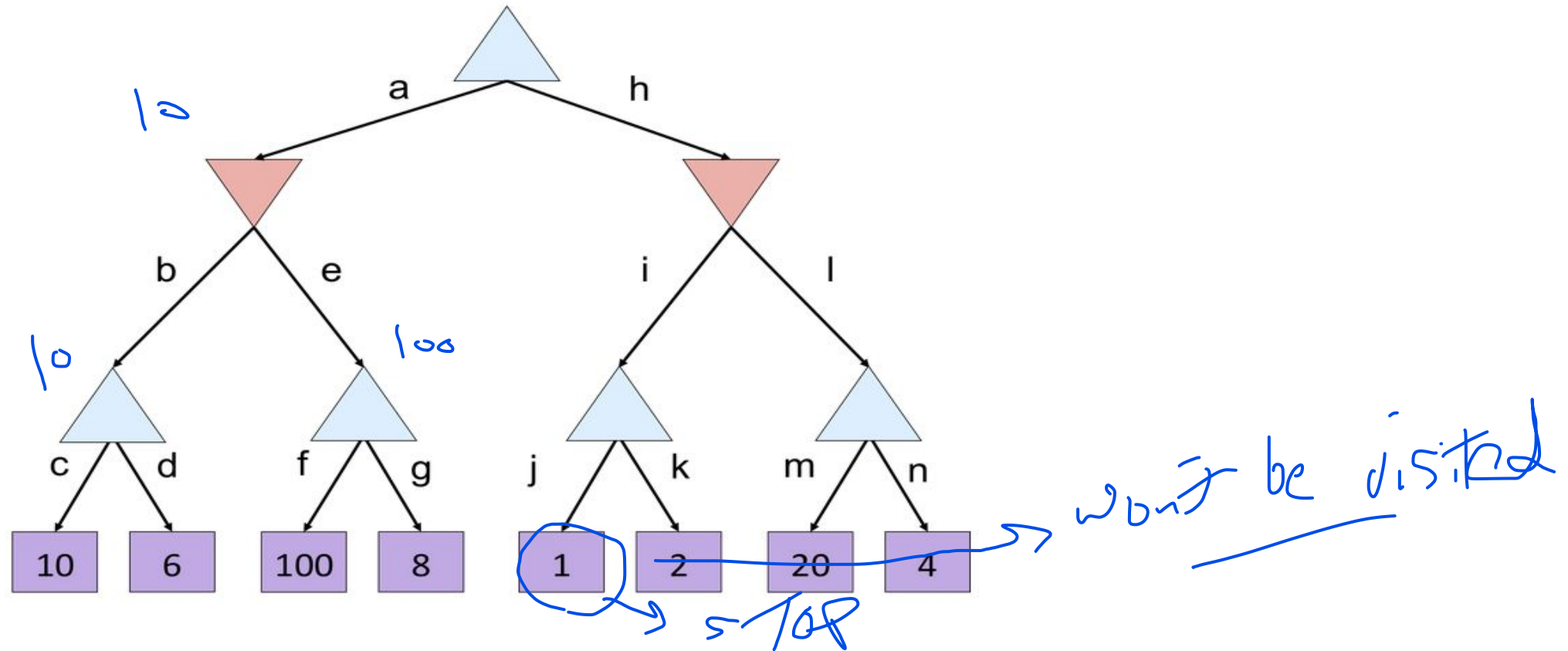


# Alpha-Beta Example 1

---



# Alpha-Beta Example 2



# Evaluation Functions

---

- Apply evaluation function to **approximately estimate** the utility at **nonterminal** nodes turning them into terminal leaves, and **cut-off the search depth**.
- Alter minimax or **alpha–beta** in two ways:
  1. Replace the **utility function** by a **heuristic evaluation function EVAL**, which estimates the position's utility.
  2. Replace the terminal test by a **cutoff test** that decides when to apply EVAL.
- The heuristic minimax for state  $s$  and maximum depth  $d$ :

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s, d) \\ \max_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

# Evaluation Functions Design

---

1. The evaluation function should order the *terminal* states in the same way as the true utility function: states that are wins must evaluate better than draws, which in turn must be better than losses.
  2. The computation must not take too long!
  3. For nonterminal states, the evaluation function should be strongly correlated with the actual chances of winning.
- The evaluation function is typically a weighted linear sum of features:

e.g. For chess,  $f_1(s) = (\text{num white queens} - \text{num black queens})$ , etc.


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

The weights of the evaluation function can be estimated by the machine learning techniques of Chapter 18.



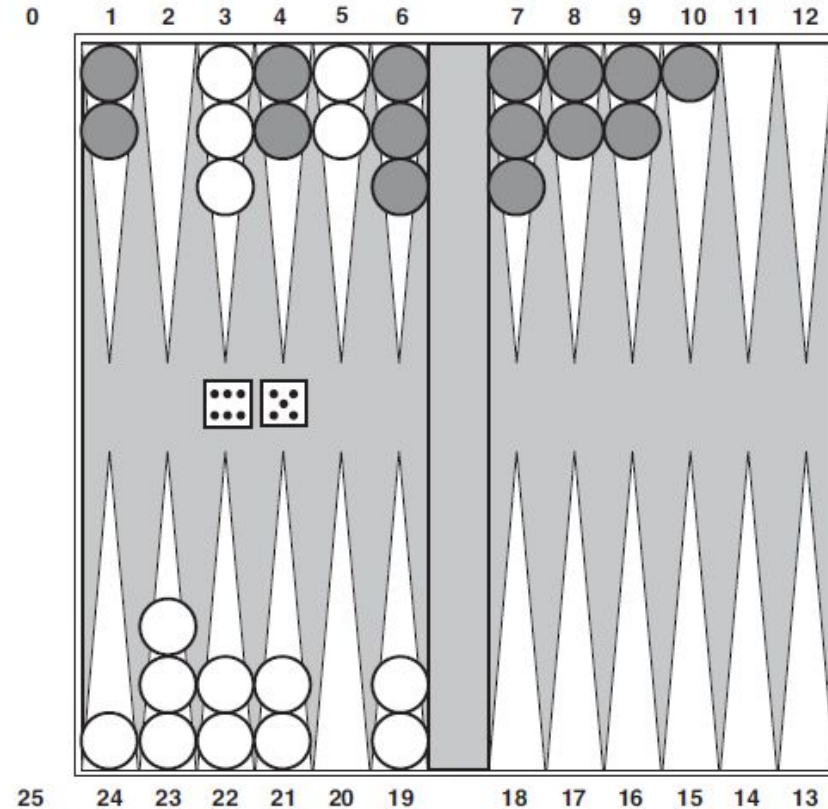
# Stochastic Games

---

- Many games has uncertainty through a random element, such as rolling dice.
  - A game tree in stochastic games like backgammon must include **chance nodes** in addition to MAX and MIN nodes.
- 
- Sources of uncertainty:
    - Explicit randomness: rolling dice
    - Unpredictable opponents: the ghosts in a pacman game respond randomly
    - Actions can fail: when moving a robot, wheels might slip

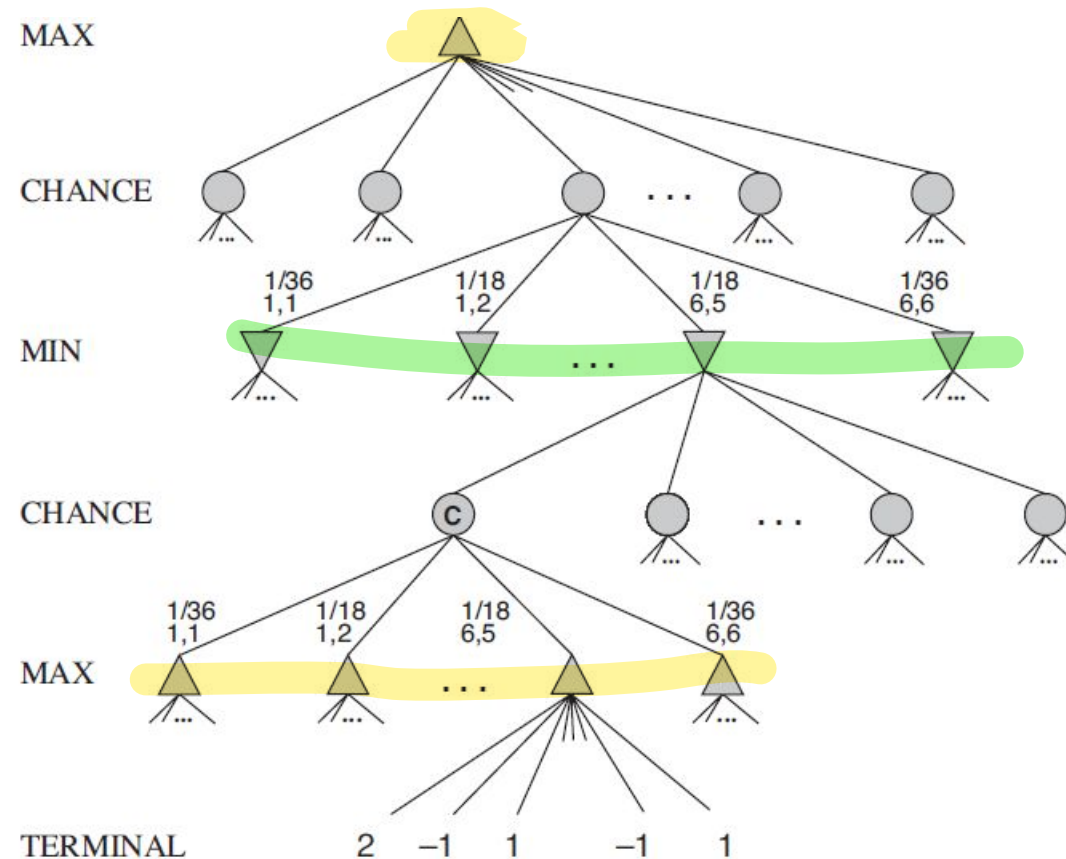
# Example: Backgammon Game

- Although White knows what his or her own legal moves are, White does not know what Black **is going to roll** and thus does not know what **Black's legal moves will be.**
- That means White **cannot construct a standard game tree** of the sort we saw in chess and tic-tac-toe.
- A game tree in backgammon must include **chance nodes** in addition to MAX and MIN nodes.



# Example: Backgammon Game Tree

- The branches leading from each chance node denote the possible dice rolls; each branch is labeled with the roll and its probability.
- There are 36 ways to roll two dice, each equally likely; but because a 6–5 is the same as a 5–6,
- there are only 21 distinct rolls.
- The six doubles (1–1 through 6–6) each have a probability of  $1/36$ , so we say  $P(1-1) = 1/36$ . The other 15 distinct rolls each have a  $1/18$  probability



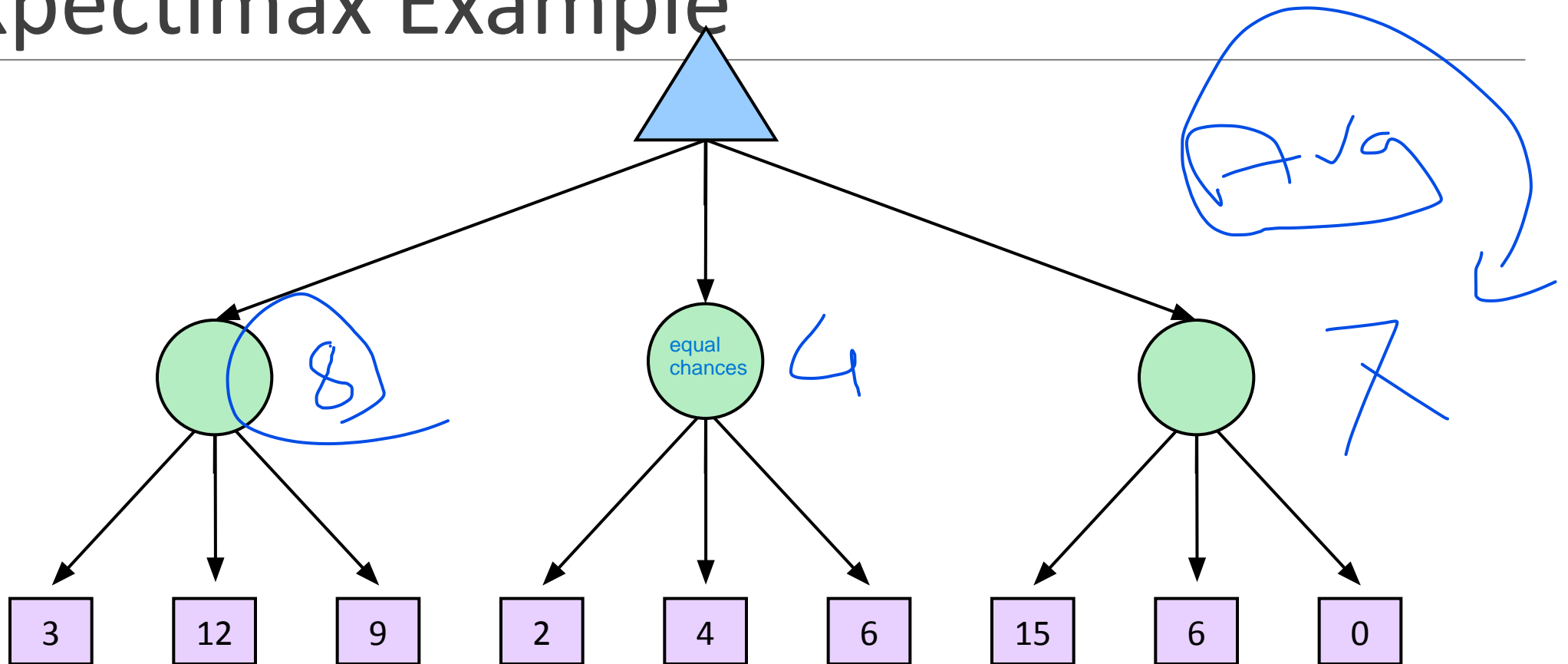
# Expectiminimax Algorithm

- For chance nodes, expectiminimax computes the expected value, which is the sum of the value over all outcomes, weighted by the probability of each chance action.
- The terminal nodes and MAX and MIN nodes work exactly the same way as the minimax algorithm.

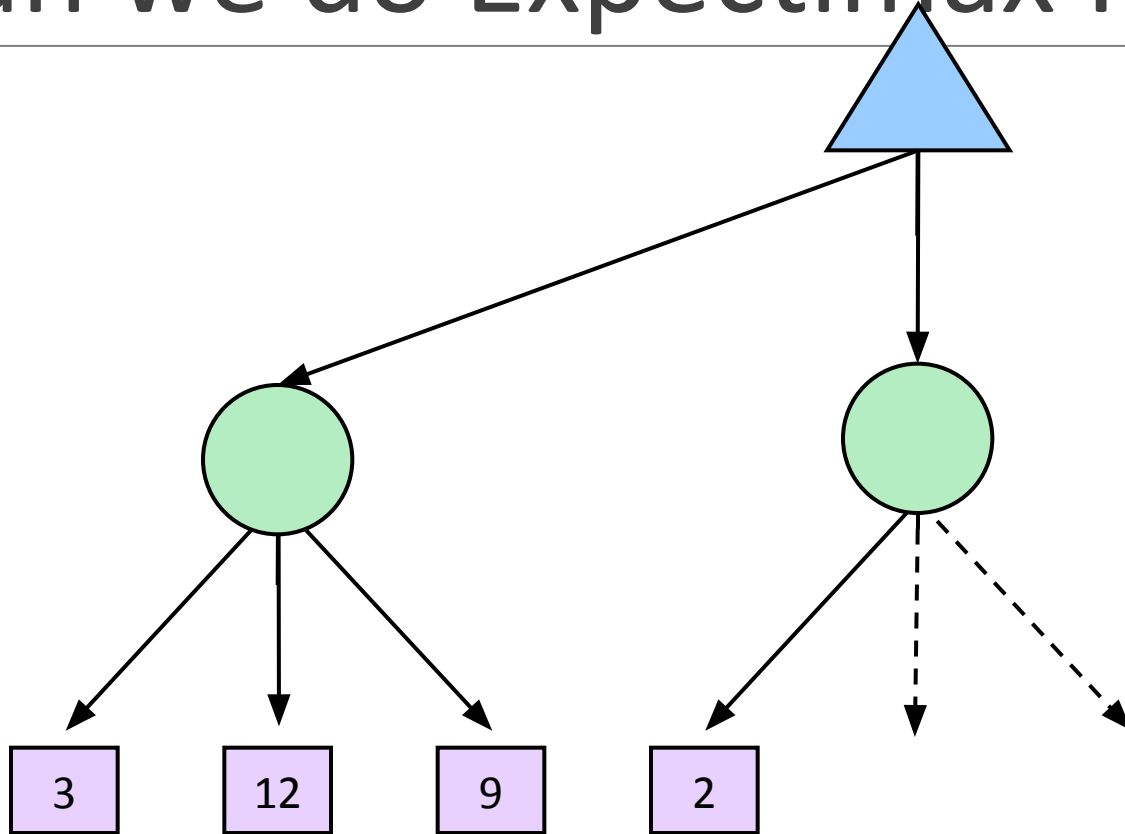
$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\ \sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE} \end{cases}$$

where  $r$  represents a possible dice roll (or other chance event) and  $\text{RESULT}(s, r)$  is the same state as  $s$ , with the additional fact that the result of the dice roll is  $r$ .

# Expectimax Example

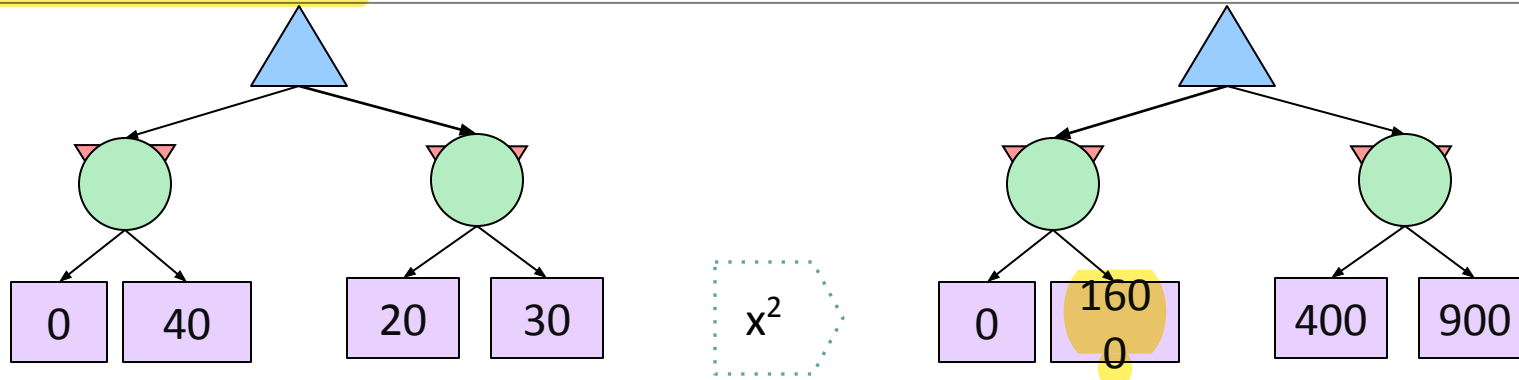


# Can we do Expectimax Pruning? Why?



- It might seem impossible because the value of chance node is the *average* of its children's values, and in order to compute the average of a set of numbers, we must look at all the numbers.
- But if we put bounds on the possible values of the utility function, then we can arrive at bounds for the average without looking at every number.
- For example, say that all utility values are between  $-2$  and  $+2$ ; then the value of leaf nodes is bounded, and in turn we can place an upper bound on the value of a chance node without looking at all its children.

# Utility-Scale

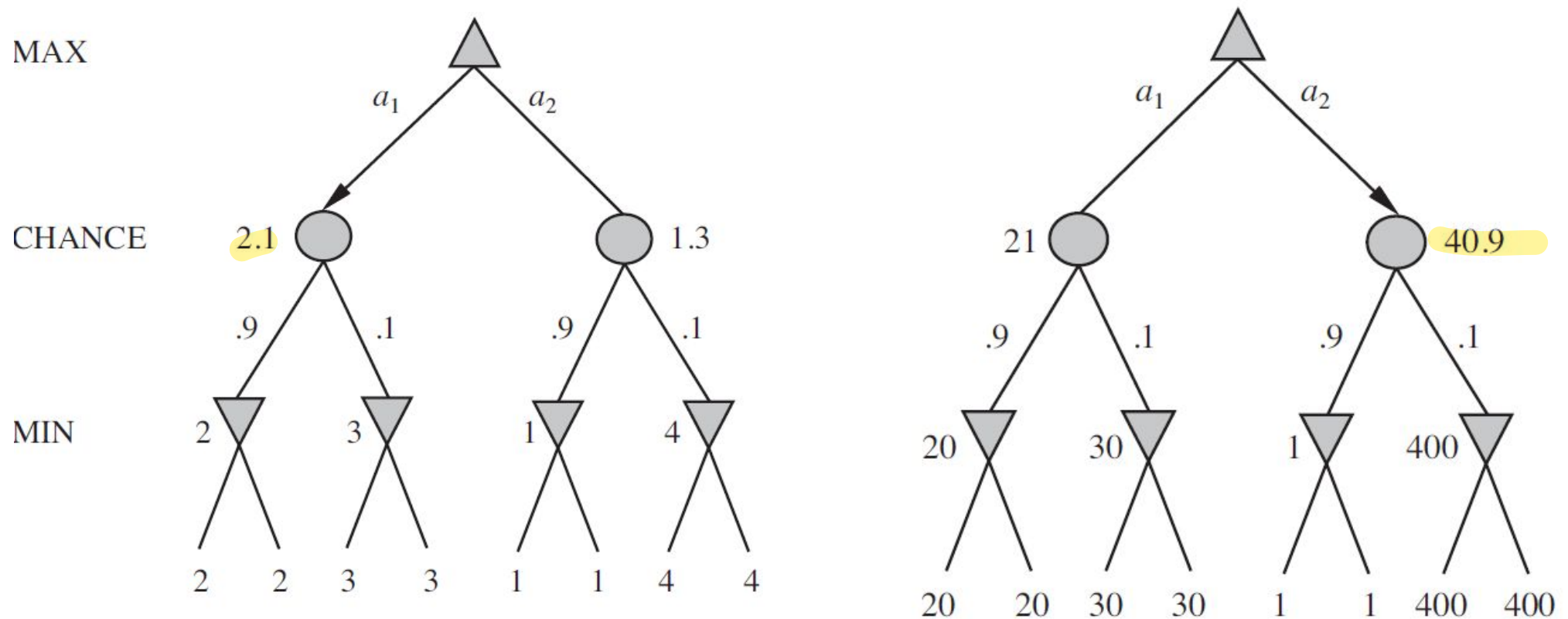


For worst-case minimax reasoning, terminal function scale doesn't matter.

- We just want better states to have higher evaluations (get the ordering right).
- We call this **insensitivity to monotonic transformations**.

For average-case expectimax reasoning, we need *magnitudes* to be meaningful.

# Expectimax Example



- An order-preserving transformation on leaf values changes the best move!



# Partially Observable Games

---

- For example, card games where the opponent's cards are unknown.
- Consider all possible deals of the invisible cards; solve each one as if it were a fully observable game; and then choose the move that has the best outcome averaged over all the deals.
- Suppose that each deal  $s$  occurs with probability  $P(s)$ ; then the move to play would be:

$$\operatorname{argmax}_a \sum_s P(s) \operatorname{MINIMAX}(\operatorname{RESULT}(s, a)) .$$

# Partially Observable Games- Card games

- In most card games, the number of possible deals is rather large.
- For example, in bridge 10, 400, 600 possible deals!!
- Apply Monte Carlo approximation by taking a random sample of N deals, where the probability of deal s appearing in the sample is proportional to P(s):

$$\operatorname{argmax}_a \frac{1}{N} \sum_{i=1}^N \operatorname{MINIMAX}(\operatorname{RESULT}(s_i, a)) .$$

- As N increases, the sum over the random sample tends to the exact value.
- P(s) does not appear explicitly in the summation, because the samples are already drawn according to P(s).

Monte Carlo Tree search ✓

# Summary

---

- Adversarial Search
- Minimax Algorithm
- Alpha beta pruning
- Stochastic games
- Expectimax algorithm
- Evaluation functions and cutting-off the search
- Partially observable games

