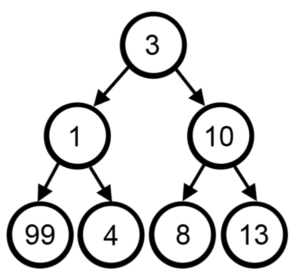
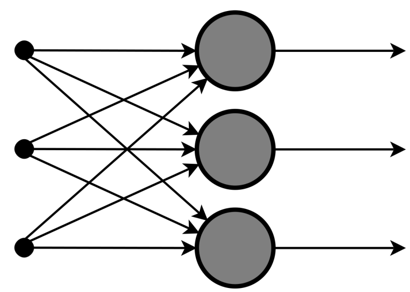
CSE 40171:
Artificial Intelligence





Adversarial Search: Alpha-Beta Pruning;
Imperfect Decisions

Homework #3 is due **tonight**
at 11:59PM

What limitations does minimax have?

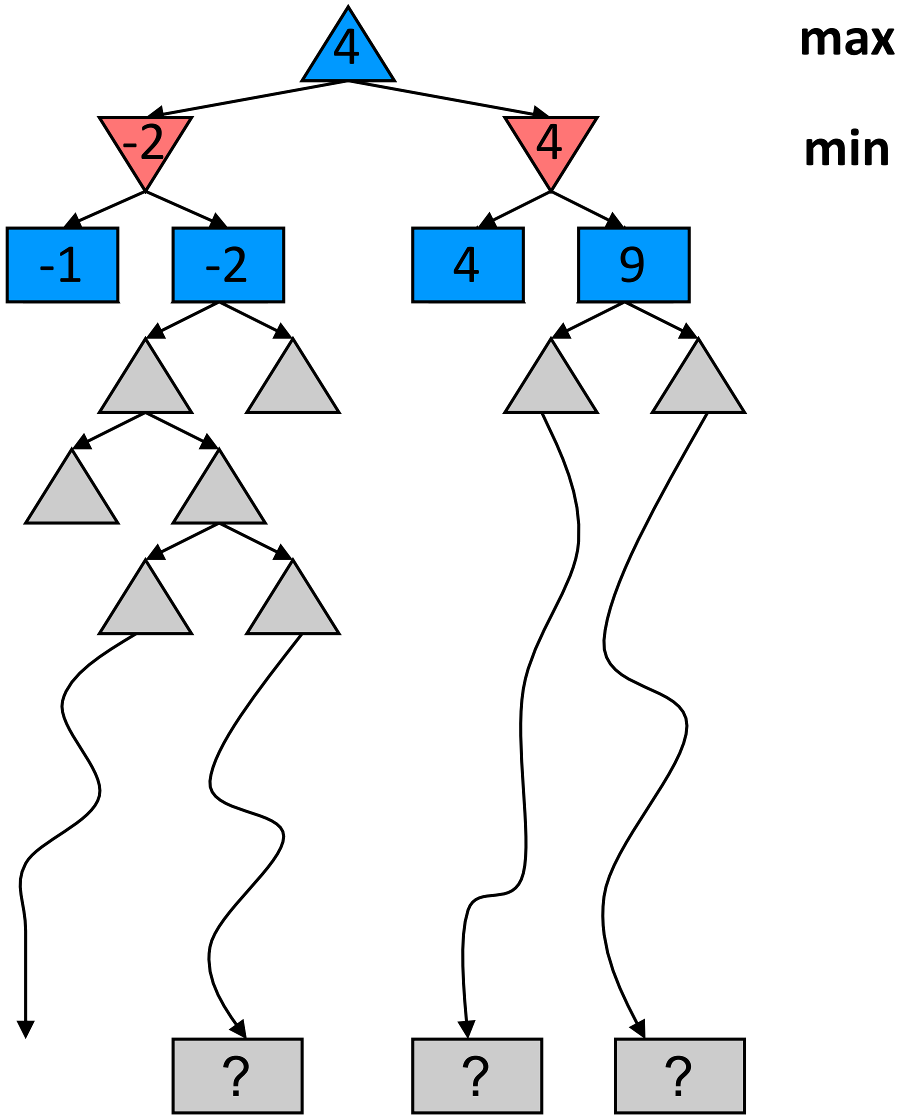


Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Resource Limits

Problem: In realistic games, cannot search to leaves!

Solution: Depth-limited search

- Instead, search only to a limited depth in the tree
- Replace terminal utilities with an evaluation function for non-terminal positions

Resource Limits

Example:

- Suppose we have 100 seconds and can explore 10K nodes / sec
- This means we can check 1M nodes per move
- **α-β** pruning reaches about depth 8 – decent chess program

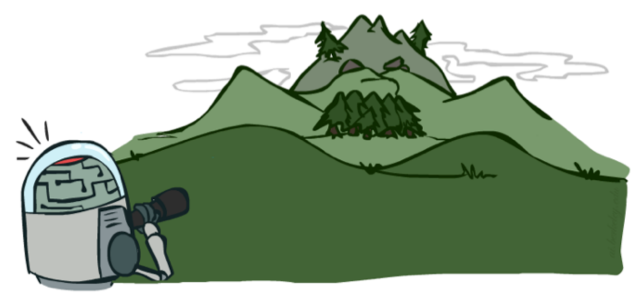
Guarantee of optimal play is gone

Depth Matters

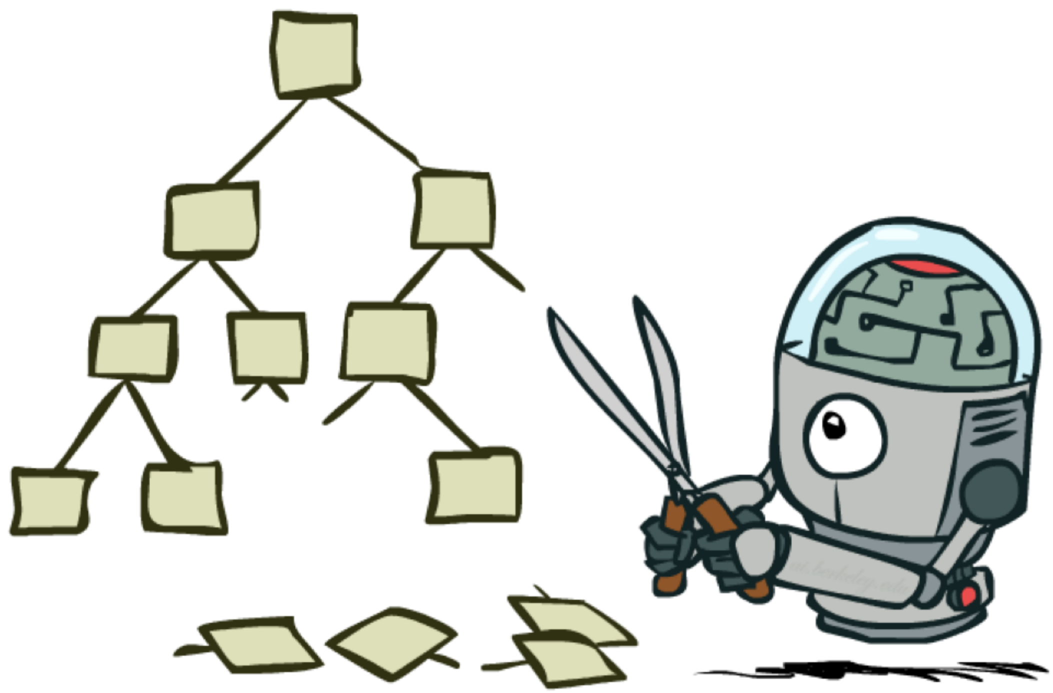
Evaluation functions are always imperfect

The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters

An important example of the tradeoff between complexity of features and complexity of computation

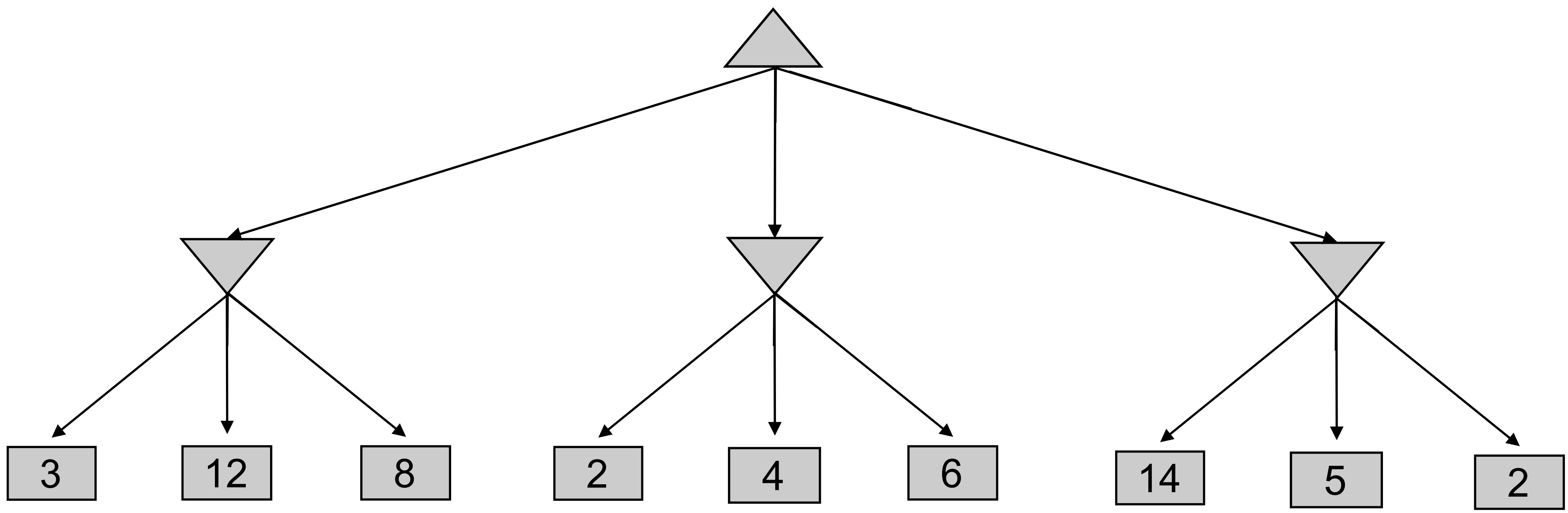


Game Tree Pruning



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Minimax Example



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Motivating Example

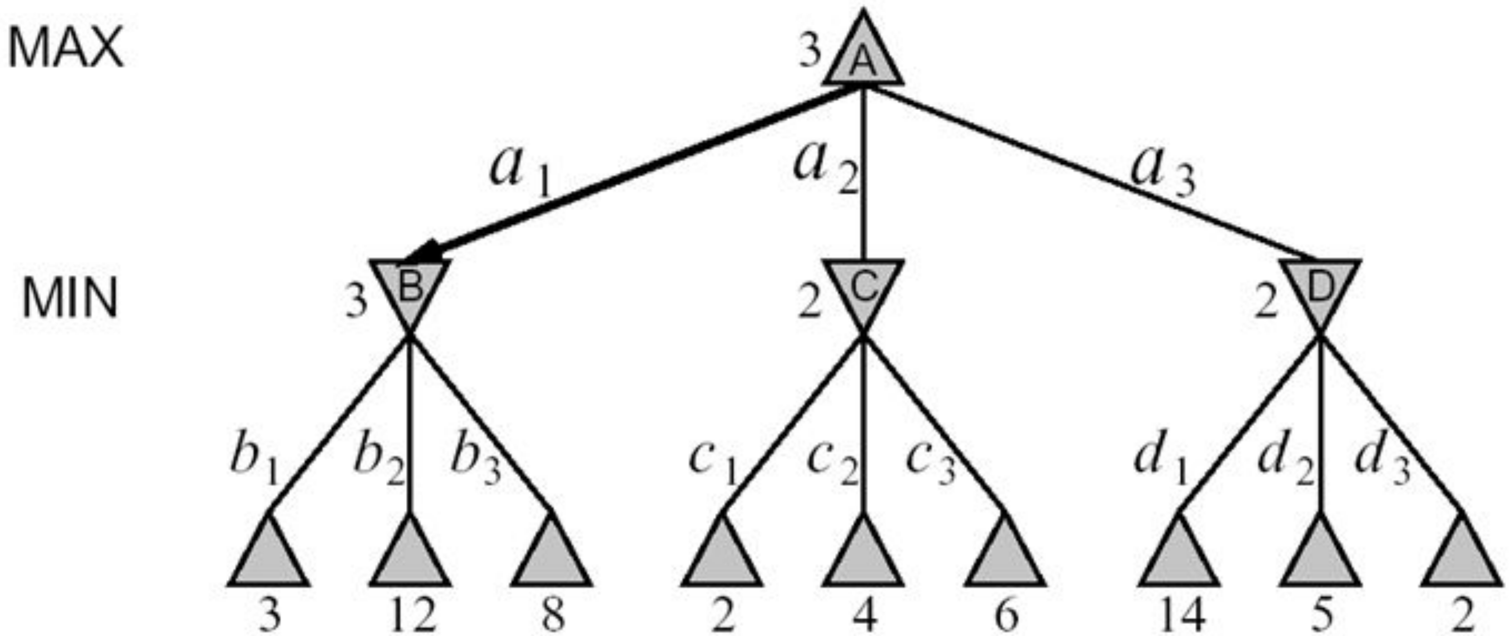
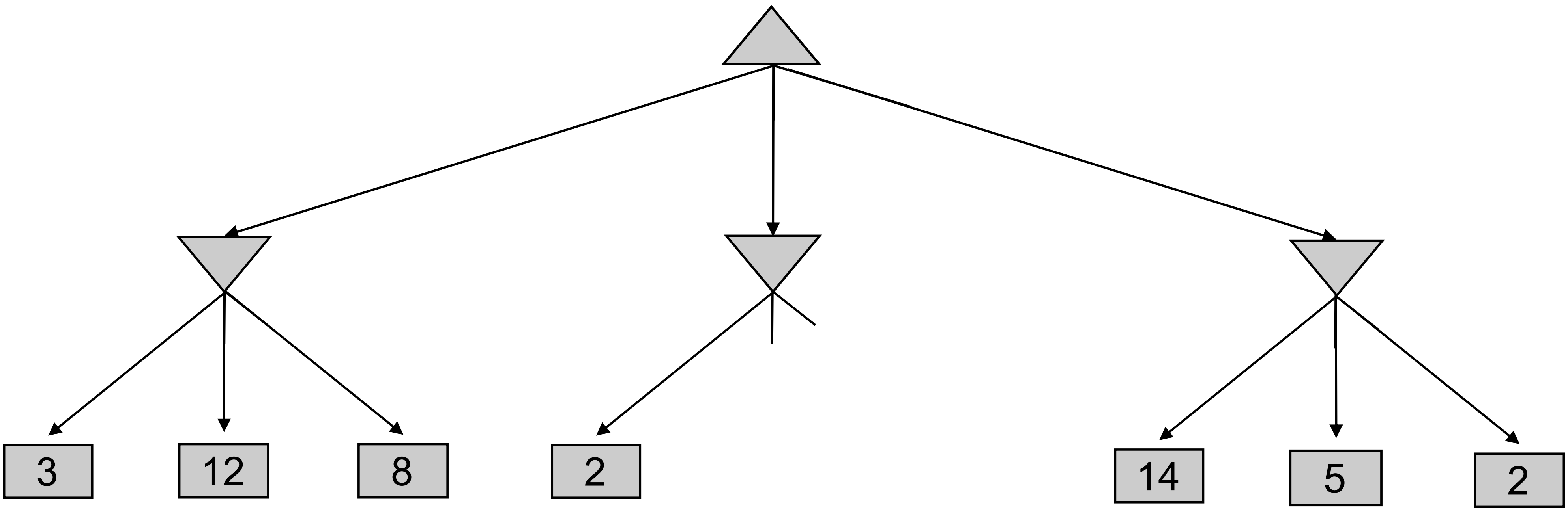


Image credit: Russell and Norvig

Minimax Pruning



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Alpha-Beta Pruning

When applied to a standard minimax tree, it returns the same move as minimax would

But always prunes away branches that cannot possibly influence the final decision

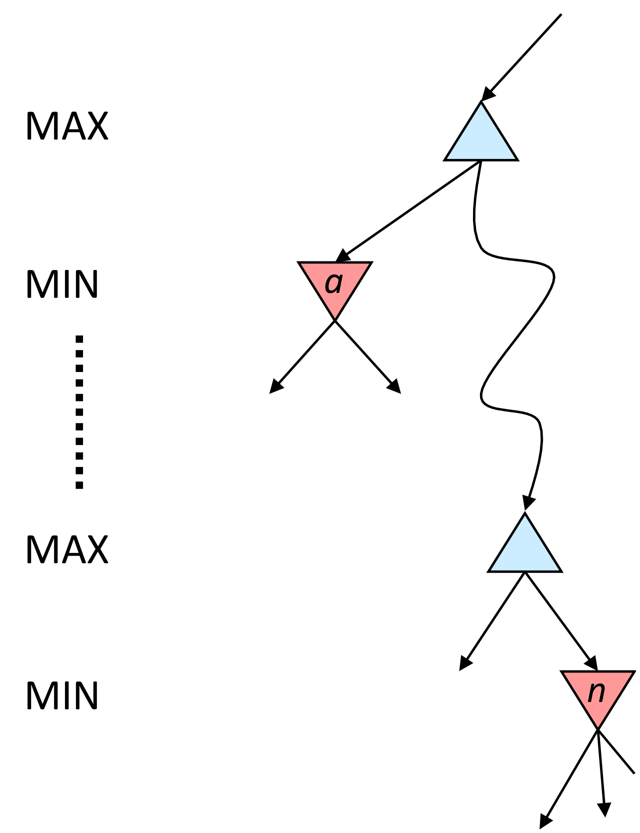
What are alpha and beta?

α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

General Configuration (MIN Version)

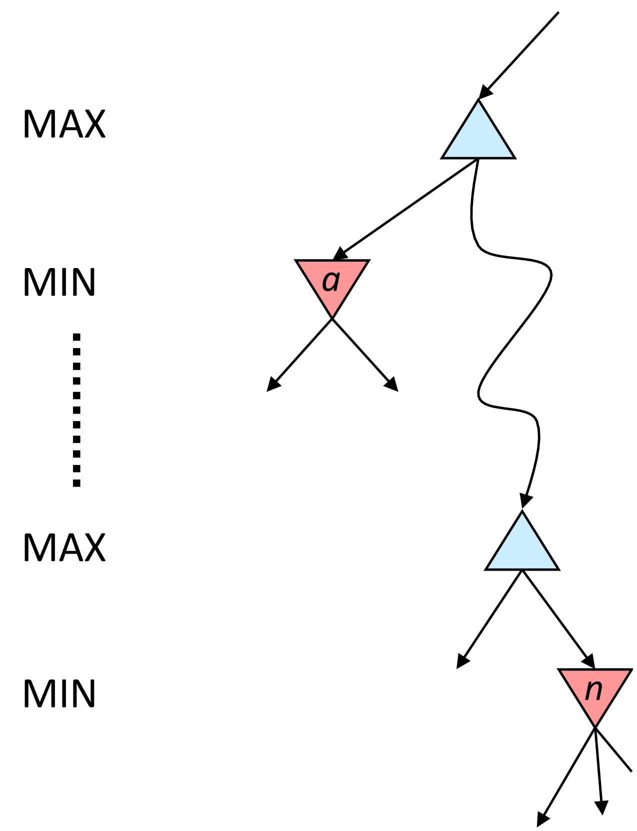
- We're computing the MIN-VALUE at some node n
- We're looping over n's children
- n's estimate of the childrens' min is dropping
- Who cares about n's value? MAX
- Let a be the best value that MAX can get at any choice point along the current path from the root
- If n becomes worse than a, MAX will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



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

General Configuration (MAX Version)

The MAX version is simply symmetric



Alpha-Beta Implementation

α: MAX's best option on path to root

β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = max(β, v)
    return v
```

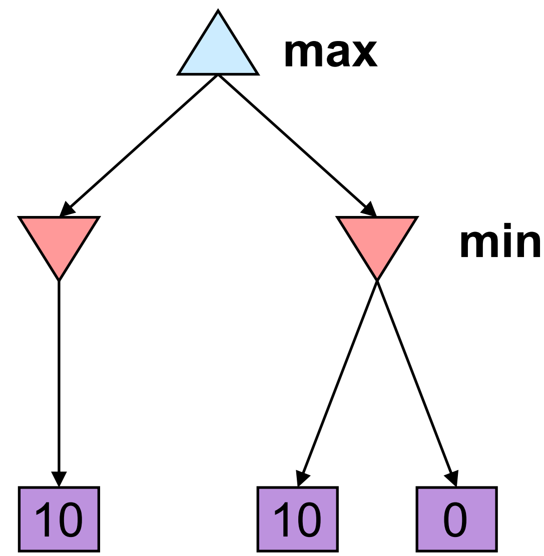
Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Alpha-Beta Pruning Properties

This pruning has no effect on minimax value computed for the root!

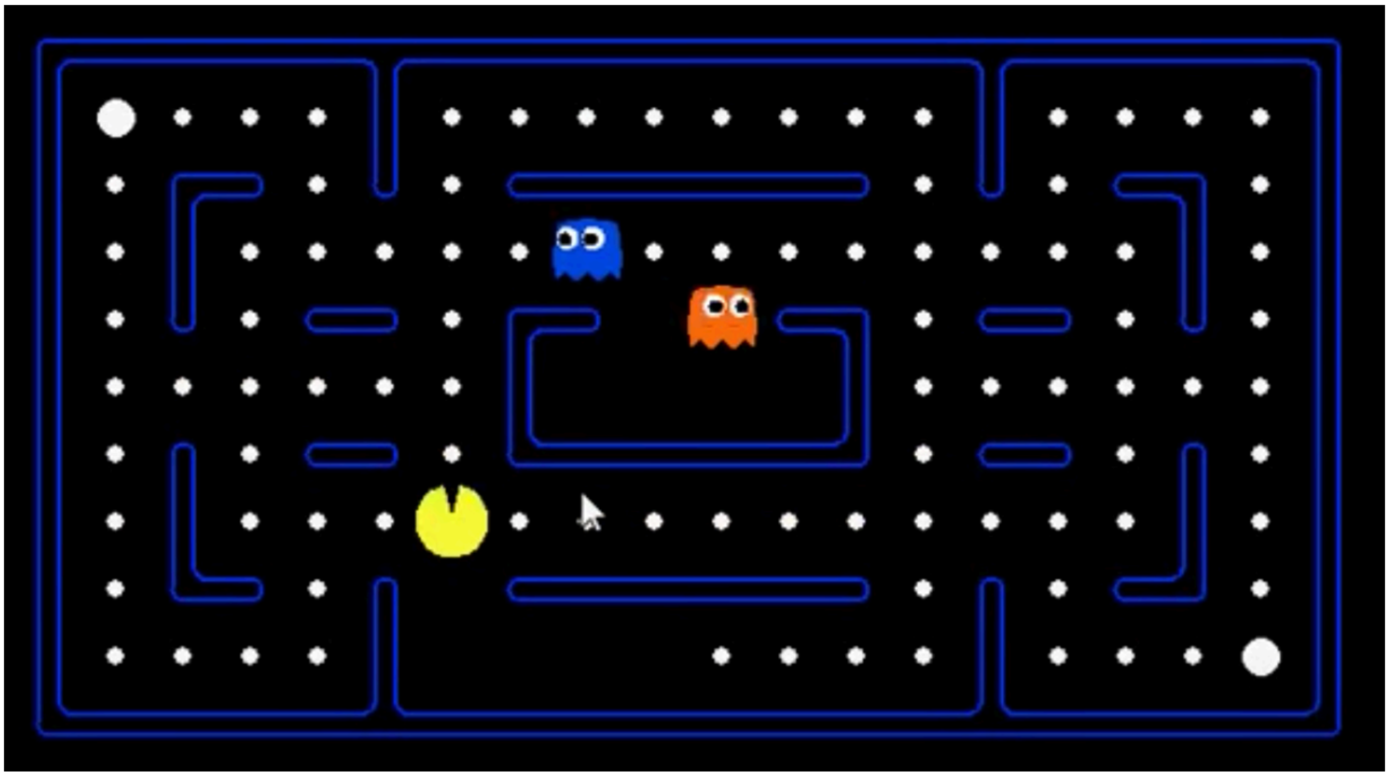
Values of intermediate nodes might be wrong

- Important: children of the root may have the wrong value
- So the most naive version won't let you do action selection



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Demo: Minimax + Alpha-Beta Pruning



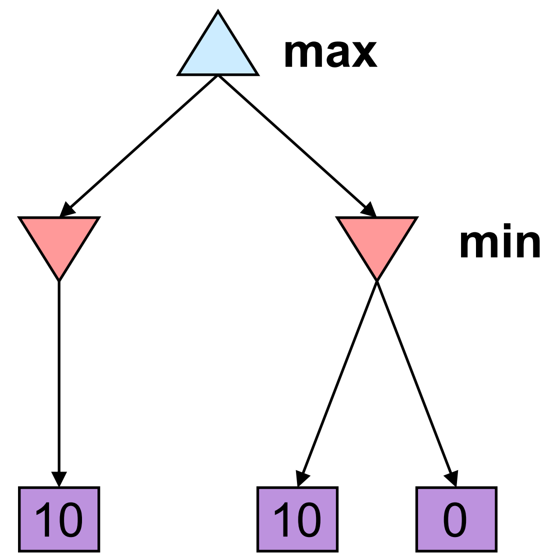
https://www.youtube.com/watch?v=_bEQJKXZ1-U

Alpha-Beta Pruning Properties

Good child ordering improves effectiveness of pruning

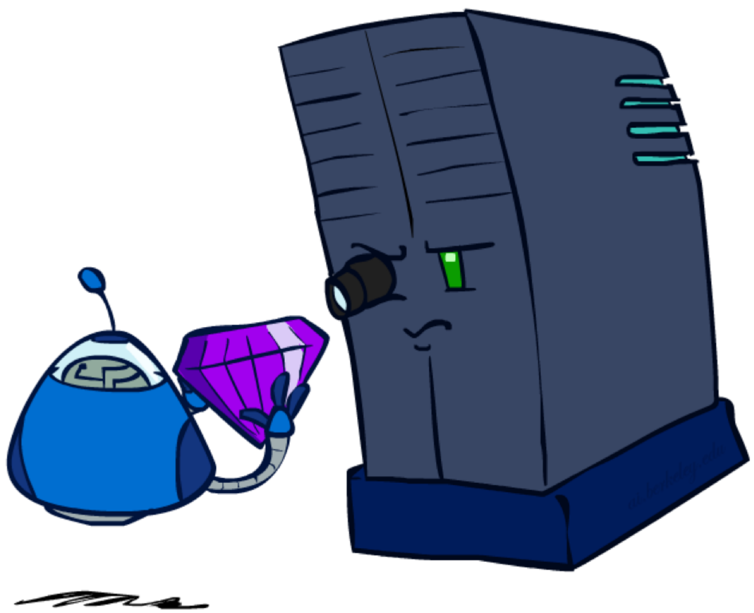
With "perfect ordering":

- 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 metareasoning (computing about what to compute)

Evaluation Functions



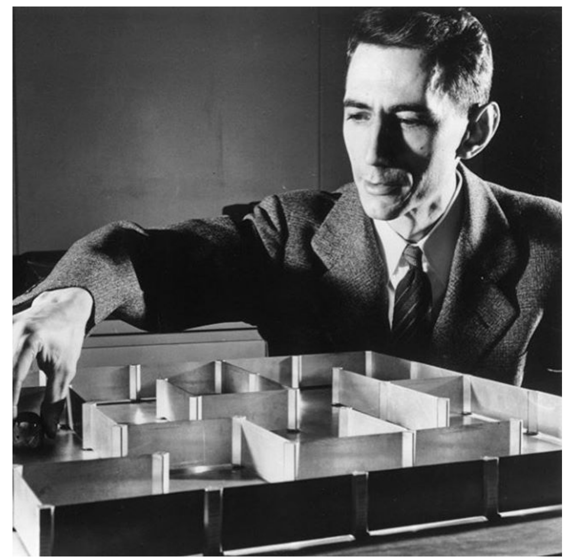
Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

It turns out that alpha-beta pruning isn't so good…

It must search all the way to terminal states for at least a portion of the search space

This is usually not practical, because we need to play the game in a reasonable amount of time

Shannon's suggestion: cutoff earlier via a heuristic **evaluation function**



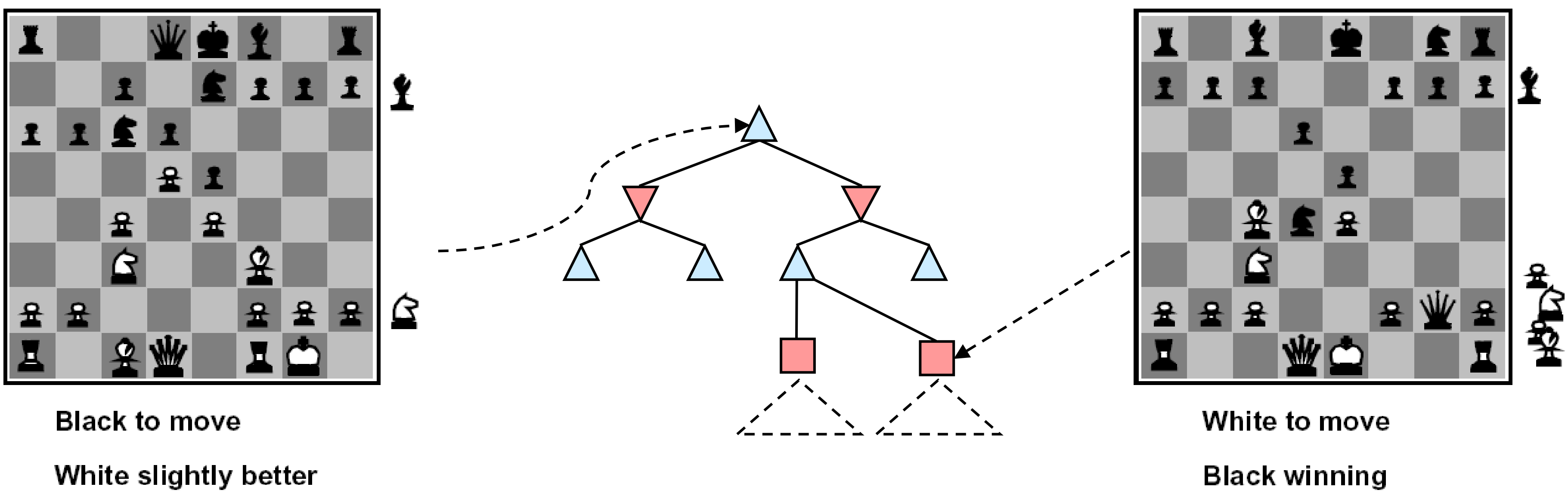
Cutoff Test

H-MINIMAX(s, d) =

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

Evaluation Functions

Evaluation functions score non-terminals in depth-limited search



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Evaluation Functions

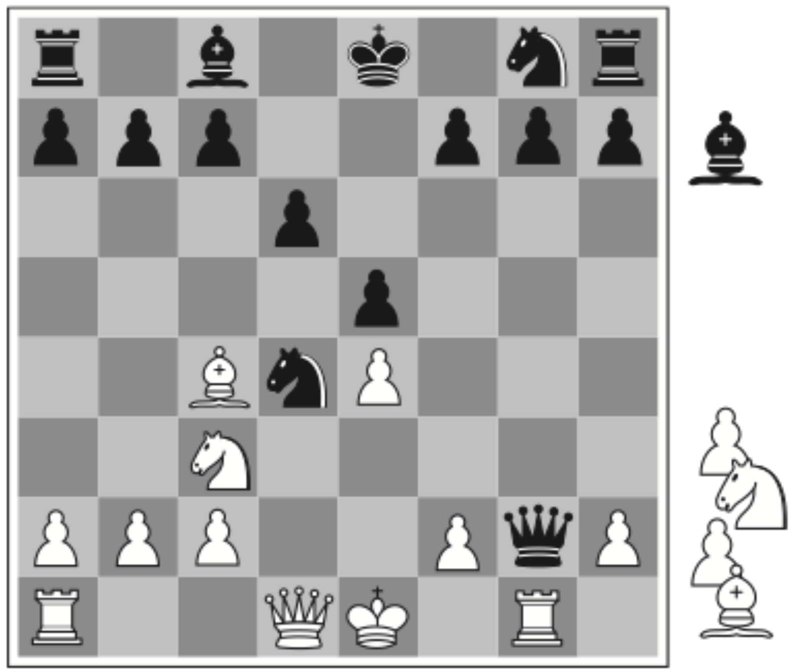
Ideal function: returns the actual minimax value of the position

In practice: typically weighted linear sum of features:

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

e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Be wary of simple approaches

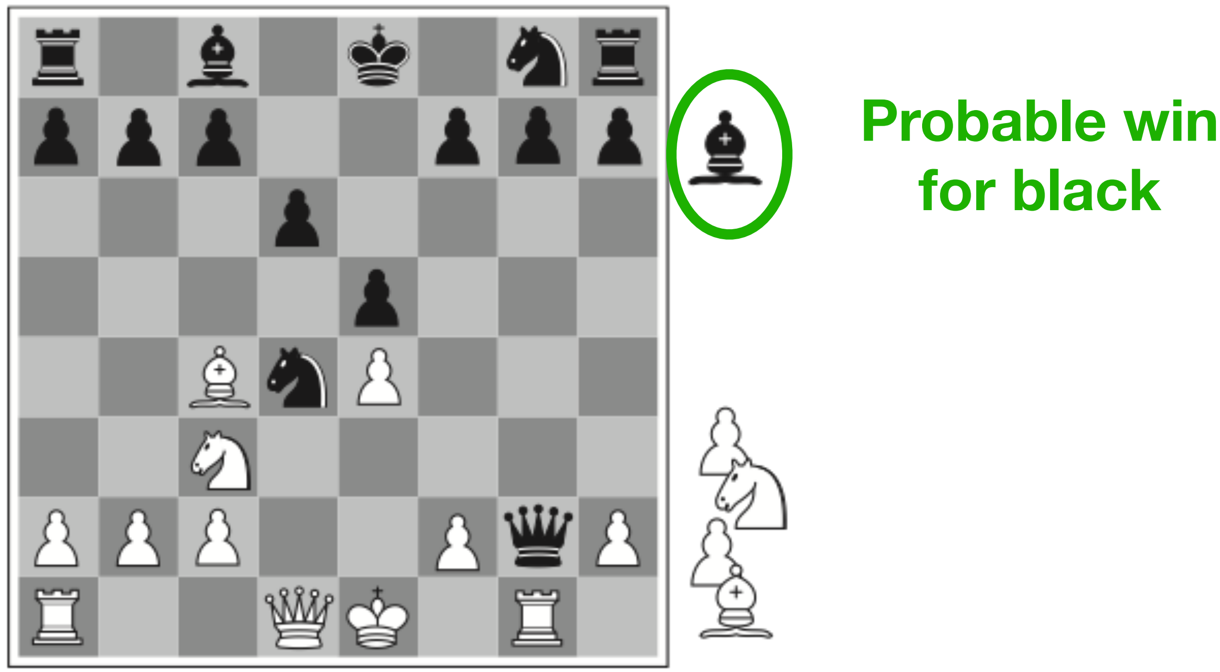


(b) White to move

Heuristic: Material Advantage

Image credit: Russell and Norvig

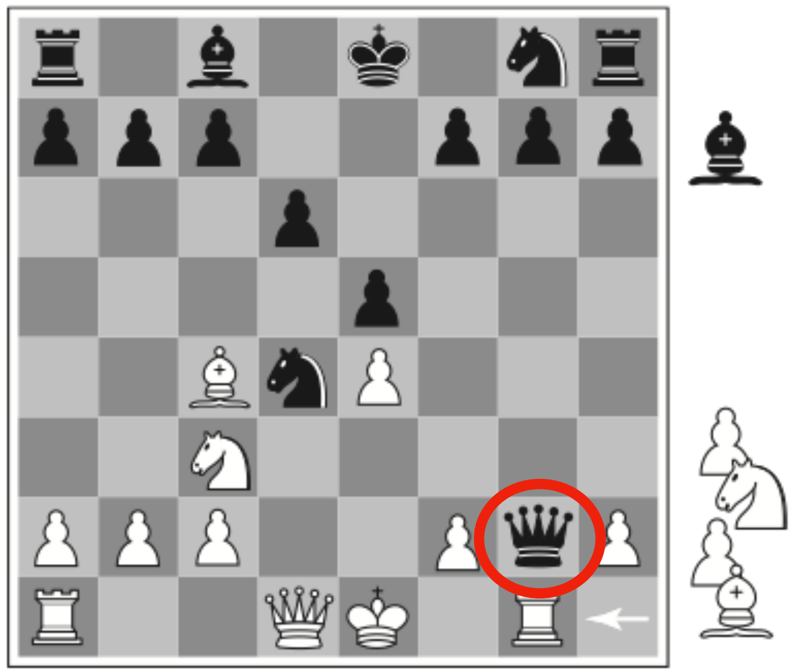
Be wary of simple approaches



(b) White to move

Image credit: Russell and Norvig

Be wary of simple approaches



(b) White to move

Image credit: Russell and Norvig