Adversarial Search: Alpha-Beta Pruning; Imperfect Decisions
Homework #3 is due **tonight** at 11:59PM
What limitations does minimax have?
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
- $\alpha$-$\beta$ 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
Minimax Example
Motivating Example
Minimax Pruning
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?

\( \alpha = \) 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.
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)
General Configuration (MAX Version)

The MAX version is simply symmetric

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
**Alpha-Beta Implementation**

\[ \alpha: \text{MAX’s best option on path to root} \]

\[ \beta: \text{MIN’s best option on path to root} \]

```python
def max_value(state, alpha, beta):
    initialize \( v = -\infty \)
    for each successor of state:
        \( v = \max(v, \text{value}(\text{successor}, \alpha, \beta)) \)
        if \( v \geq \beta \) return \( v \)
    \( \alpha = \max(\alpha, v) \)
    return \( v \)

def min_value(state, alpha, beta):
    initialize \( v = +\infty \)
    for each successor of state:
        \( v = \min(v, \text{value}(\text{successor}, \alpha, \beta)) \)
        if \( v \leq \alpha \) return \( v \)
    \( \beta = \max(\beta, v) \)
    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
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
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

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

Evaluation functions score non-terminals in depth-limited search

Black to move
White slightly better

White to move
Black winning
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. $f_1(s) = \text{(num white queens – num black queens)}$, etc.
Be wary of simple approaches

Heuristic: Material Advantage
Be wary of simple approaches

(b) White to move

Probable win for black
Be wary of simple approaches

(b) White to move

Image credit: Russell and Norvig