# CS 188: Artificial Intelligence
## Spring 2010

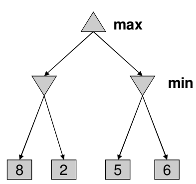
Lecture 7: Minimax and Alpha-Beta Search

2/9/2010

Pieter Abbeel – UC Berkeley

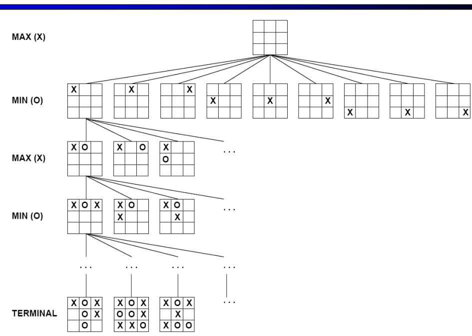
Many slides adapted from Dan Klein

## Simple two-player game example






## Tic-tac-toe Game Tree





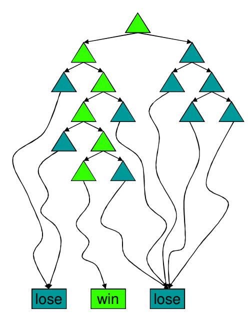

## Deterministic Games

- Many possible formalizations, one is:
  - States: S (start at $s_0$)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: SxA → S
  - Terminal Test: S → {t,f}
  - Terminal Utilities: SxP → R
- Solution for a player is a policy: S → A



## Deterministic Single-Player?

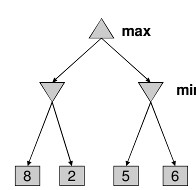
- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- … it's just search!
- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don't have path sums as before (utilities at end)
- After search, can pick move that leads to best node





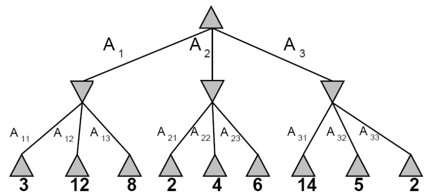

## Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result
- **Minimax search**
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves*
  - Choose move to position with highest minimax value = best achievable utility against best play




** Slightly different from the book definition*

## Minimax Example




7

## Minimax Search

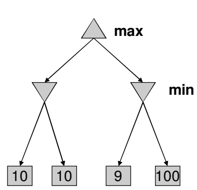
```
function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
   return v
```

8

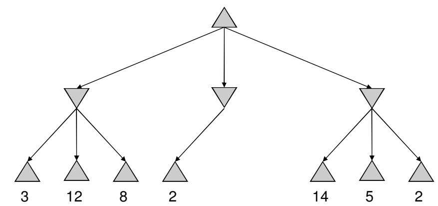
## Minimax Properties

- Optimal against a perfect player. Otherwise?
- Time complexity?
  - O($b^m$)
- Space complexity?
  - O(bm)
- For chess, b ≈ 35, m ≈ 100
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?




10

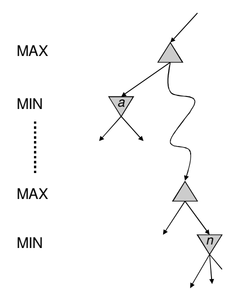
## Pruning




13

## Alpha-Beta Pruning

- General configuration
  - We're computing the MIN-VALUE at *n*
  - We're looping over *n*'s children
  - *n*'s value estimate is dropping
  - *a* is the best value that MAX can get at any choice point along the current path
  - If *n* becomes worse than *a*, MAX will avoid it, so can stop considering *n*'s other children
  - Define *b* similarly for MIN

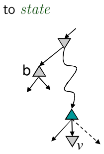



15

## Alpha-Beta Pseudocode

```
function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
   return 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
```



# Alpha-Beta Pruning Properties

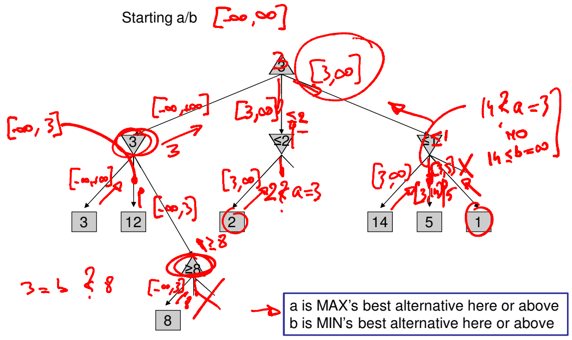
- This pruning has no effect on final result at the root
- Values of intermediate nodes might be wrong!
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of metareasoning (computing about what to compute)

17

21

# Alpha-Beta Pruning Example

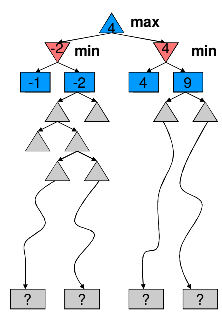
Starting a/b




a is MAX's best alternative here or above
b is MIN's best alternative here or above

# Resource Limits

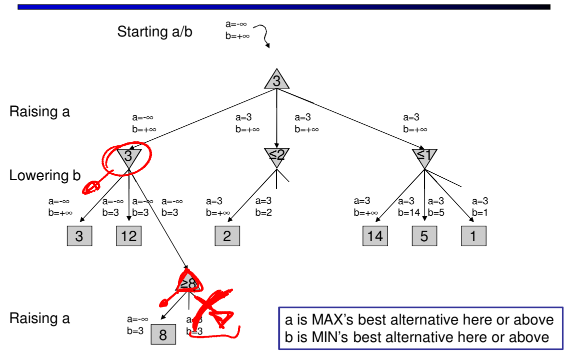
- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program




23

# Alpha-Beta Pruning Example

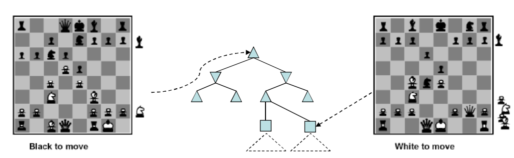
Starting a/b




a is MAX's best alternative here or above
b is MIN's best alternative here or above

# Evaluation Functions

- Function which scores non-terminals



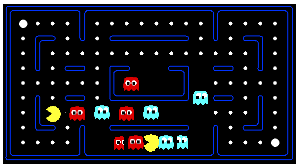

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

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

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

24

## Evaluation for Pacman



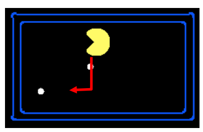
[DEMO: thrashing, smart ghosts]

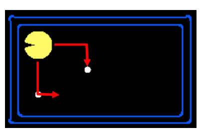
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$



## Why Pacman Can Starve

- He knows his score will go up by eating the dot now
- He knows his score will go up just as much by eating the dot later on
- There are no point-scoring opportunities after eating the dot
- Therefore, waiting seems just as good as eating



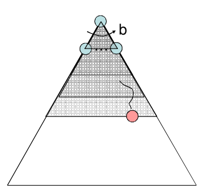


## Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.

   ….and so on.




Why do we want to do this for multiplayer games?



## Non-Zero-Sum Games

- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (or back up) nodes from children