




Adversarial Search

CIS*3700 (Winter 2007)




Why Game Search?

- Good domains to measure success or failure: still valid.
- No need for large amount of knowledge; some kind of search will be enough: not quite true.
 - For small games like Tic-Tac-Toe, it's possible.
 - For medium games like Checkers, there are about 10^{40} positions to explore.
 - For large games like Chess, there are about 35^{100} positions to search.




What's Special?

- "Unpredictable" opponent: need to specify a move for every possible opponent's reply
- Time limits: it's unlikely to find goal or be perfect → must approximate




Two-Player Games

- Two players alternate in making moves.
- At each turn, the rules define what moves are legal and what are their effects; no element of chance.
- Each player has complete information about his opponent's position, including the choices open to him and the moves he has made.
- A game begins from a specified state and ends in a win, a loss, or possibly a draw.

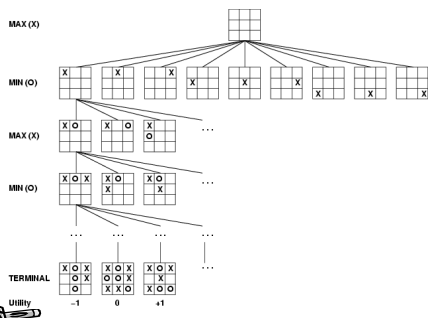


Game Trees


- A game tree represents all possible plays of a game.
- Root node: initial state where the first player has his turn.
- Successor nodes: those that can be reached by both players.
- Terminal nodes: those corresponding to a win, a loss, or a draw.

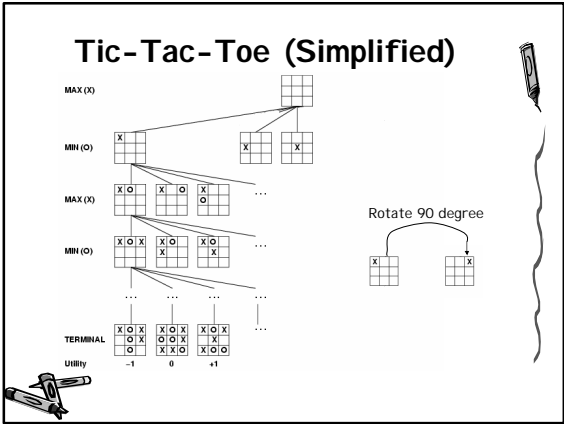


Tic-Tac-Toe



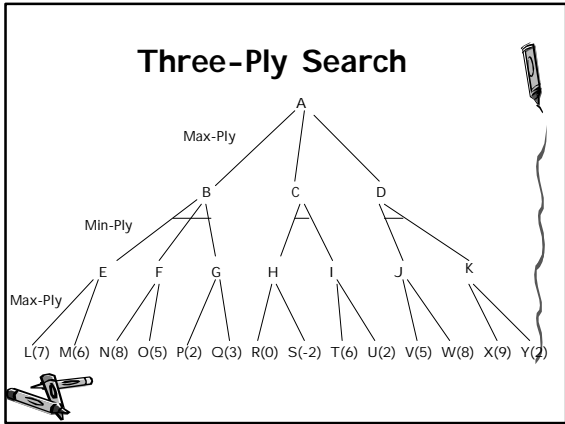
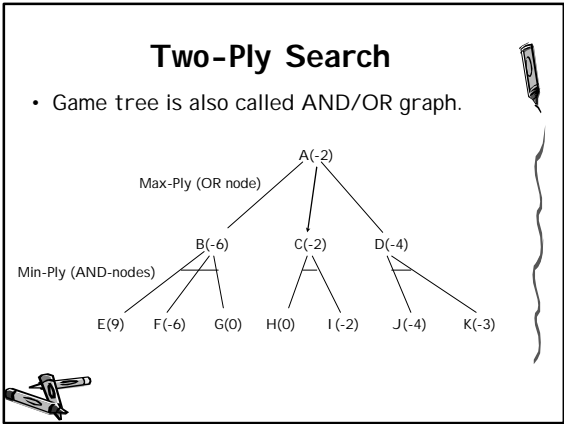
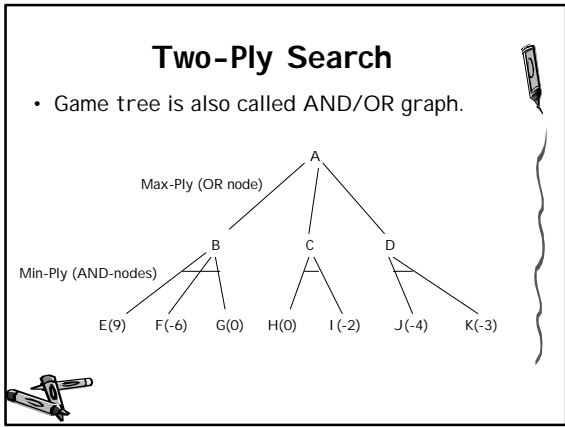
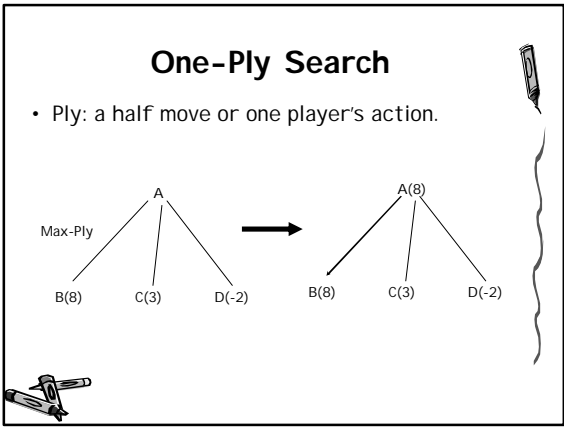
The diagram illustrates a game tree for Tic-Tac-Toe. It starts with a root node labeled MAX (X) containing an empty 3x3 grid. This node branches into several MIN (O) nodes, each with a different grid state. These MIN nodes further branch into MAX (X) nodes, and so on, eventually leading to terminal nodes. The terminal nodes are labeled with utility values: -1 (representing a loss for X), 0 (representing a draw), and +1 (representing a win for X). Ellipses (...) indicate that the tree continues beyond the shown nodes.

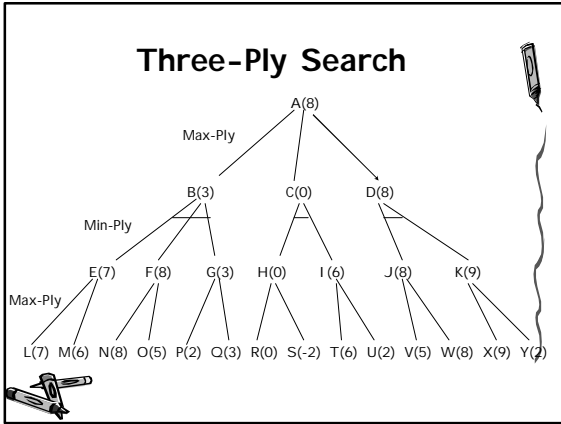




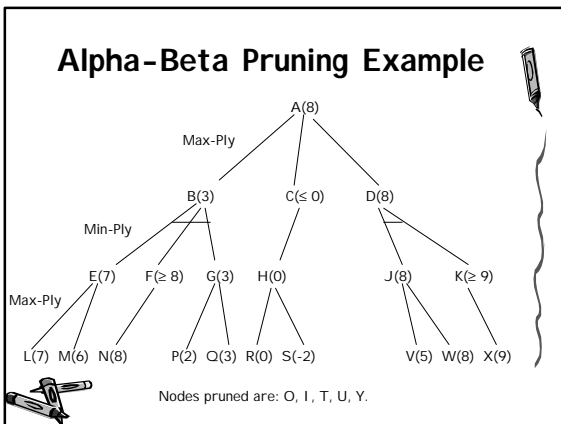
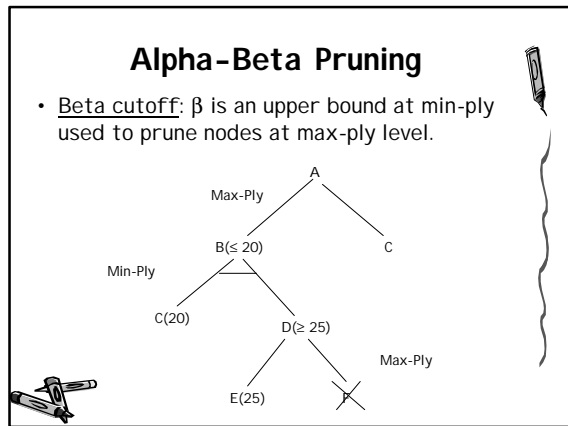
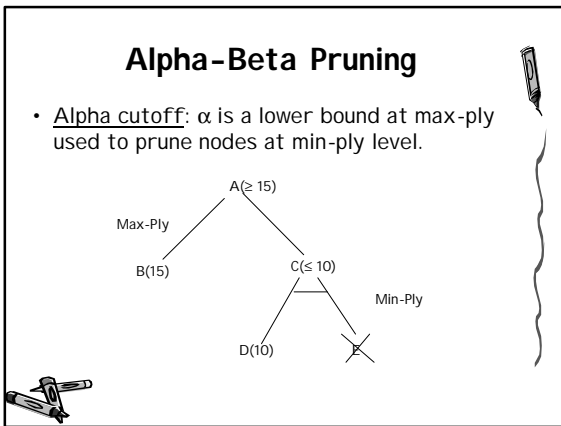
MiniMax Search

- **Static Evaluation Function:** estimate the value of a position without looking at any of the position's successors. Similar to $h(n)$ in state-space search.
- **Backup Evaluation:** select the min value at the minimizing play and the max value at the maximizing ply.
- Search is performed in a depth-first, depth-limited fashion.





- ### Properties of MiniMax Search
- Complete? Yes (if tree is finite)
 - Optimal? Yes (against an optimal opponent)
 - Time complexity? $O(b^m)$
 - Space complexity? $O(bm)$ (depth-first exploration)
 - For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games \rightarrow exact solution completely infeasible



Alpha-Beta Pruning Algorithm

```

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

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
inputs: state, current state in game
 $\alpha$ , the value of the best alternative for MAX along the path to state
 $\beta$ , the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) then return UTILITY(state)
 $v \leftarrow -\infty$ 
for  $a, s$  in SUCCESSORS(state) do
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
if  $v \geq \beta$  then return  $v$ 
 $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
return  $v$ 
    
```

Alpha-Beta Pruning Algorithm

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
inputs: state, current state in game
        $\alpha$ , the value of the best alternative for MAX along the path to state
        $\beta$ , the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) then return UTILITY(state)
 $v \leftarrow +\infty$ 
for  $a, s$  in SUCCESSORS(state) do
   $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
  if  $v \leq \alpha$  then return  $v$ 
   $\beta \leftarrow \text{MIN}(\beta, v)$ 
return  $v$ 
```

Evaluation Functions

- For many games, we typically use a linear weighted sum of features
$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$
- For Chess, e.g., $w_1 = 9$ with
 $f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.}$

How Many Ply Needed?

- How many ply for a chess game?
 - 4-ply \approx human novice
 - 8-ply \approx typical PC, human master
 - 12-ply \approx Deep Blue, Kasparov
- Has one side won?
- How many ply have we already explored?
- How promising is this path?
- How much time is left?

Other Two-Player Games

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Other Two-Player Games

- Othello: human champions refuse to compete against computers, who are too good.
- Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.