Adversarial Search
CS472/CS473 — Fall 2005

Slide CS472 – Adversarial Search 1

Game Playing
An AI Favorite
- structured task
- clear definition of success and failure
- does not require large amounts of knowledge (at first glance)
- focus on games of perfect information

Slide CS472 – Adversarial Search 2

Game Playing
Initial State is the initial board/position
Successor Function defines the set of legal moves from any position
Terminal Test determines when the game is over
Utility Function gives a numeric outcome for the game

Slide CS472 – Adversarial Search 3

Game Playing as Search

Slide CS472 – Adversarial Search 4

Partial Search Tree for Tic-Tac-Toe

Slide CS472 – Adversarial Search 5

Simplified Minimax Algorithm
1. Expand the entire tree below the root.
2. Evaluate the terminal nodes as wins for the minimizer or maximizer (i.e. utility).
3. Select an unlabeled node, \( n \), all of whose children have been assigned values. If there is no such node, we’re done — return the value assigned to the root.
4. If \( n \) is a minimizer move, assign it a value that is the minimum of the values of its children. If \( n \) is a maximizer move, assign it a value that is the maximum of the values of its children. Return to Step 3.

Slide CS472 – Adversarial Search 6
**Another Example**

```
MAX
A1  A2  A3
A11 A12 A13

MIN
A2  A3
A21 A22 A23

A31 A32 A33
```

---

**Minimax**

```
function MINIMAX-DECISION(game) returns an operator
for each op in OPERATORS[game] do
    VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
end
return the op with the highest VALUE[op]
```

```
function MINIMAX-VALUE(state, game) returns a utility value
if TERMINAL-TEST[game](state) then
    return UTILITY[game](state)
else if MAX is to move in state then
    return the highest MINIMAX-VALUE of SUCCESSORS(state)
else
    return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

---

**Improving Minimax — \(\alpha - \beta\) pruning**

**Idea:** Avoid generating the whole search tree

**Approach:** Analyze which subtrees have no influence on the solution

---

**\(\alpha - \beta\) Search**

\(\alpha = \) lower bound on Max’s outcome; initially set to \(-\infty\)
\(\beta = \) upper bound on Min’s outcome; initially set to \(+\infty\)

We’ll call \(\alpha - \beta\) procedure recursively with a narrowing range between \(\alpha\) and \(\beta\).

Maximizing levels may reset \(\alpha\) to a higher value; Minimizing levels may reset \(\beta\) to a lower value.