In game playing, choosing an action must take the opponent into account.

A search problem for a game is defined by:
- $s_{\text{current}}$: Current state/position of the game.
- $\text{PLAYER}(s)$: Whose turn is it in a given state?
- $\text{EXPAND}(s)$: A set of states from possible moves.
- $\text{GAME-OVER}(s)$: Is the game over in a given state?
- $\text{WHO-WON}(s, p)$: If $s$ is the end of the game, who won?

The state space of many games is too large to search. An alternative is to search as deeply as possible, estimate the values of the fringe states, and combine the values into an overall evaluation.

**Search in Game Playing**
- **max.** The player whose turn it is to move.
- **min.** The other player.
- **Ply.** A synonym for “depth.”
- **Evaluation function.** Estimates max’s utility.
Minimax

**Max-Value Procedure**

Should maximize evaluation function assuming that min minimizes it. Pseudocode assumes it is max's turn to move, and that turns are interleaved.

```
function Max-Value(s, bound)
    if Game-Over(s) or bound = 0
        then return Evaluation(s)
    max ← −∞
    for each s_successor in Expand(s)
        do eval ← Min-Value(s_successor, bound − 1)
        if eval > max then max ← eval
    return max
```

CS 3793 Artificial Intelligence

**Min-Value Procedure**

Assume min minimizes max's evaluation function.

```
function Min-Value(s, bound)
    if Game-Over(s) or bound = 0
        then return Evaluation(s)
    min ← ∞
    for each s_successor in Expand(s)
        do eval ← Max-Value(s_successor, bound − 1)
        if eval < min then min ← eval
    return min
```

CS 3793 Artificial Intelligence

Example

```
Minimax
evaluation = +10 if I have 3 in a row
−10 if my opponent has 3 in a row
+1 for each potential 3 in a row for me
−1 for each potential 3 in a row for my opponent
```

MAX

```
min(−2,−3,−2,−3) = −3
```

MIN

```
max(−3,−4) = −3
min(−4,−3,−4,−3) = −4
```

Alpha-Beta Pruning

**Idea of Alpha-Beta Pruning**

- Alpha-beta pruning avoids search that won't change the minimax evaluation.
- Example: If max has a move with value 3, stop searching other moves known to be ≤ 3.
- General Principle: Consider a state s.
  - α = largest max in ancestors of s.
  - β = smallest min in ancestors of s.
  - If α ≥ β, processing s cannot change eval.
- Proof: Let v = s's minimax value.
  - α ≥ β implies α ≥ v or v ≥ β.
  - α ≥ v implies v can't change ancestor with α.
  - v ≥ β implies v can't change ancestor with β.
  - This implies v cannot propagate to the top.
α-β MAX-VALUE Procedure

Parameters:
α is a known max value in an ancestor.
β is a known min value in an ancestor.

function MAX-VALUE(s, bound, α, β)
if Game-Over(s) or bound = 0
    then return Evaluation(s)
for each s-successor in Expand(s)
do eval ← MIN-VALUE(s-successor, bound−1, α, β)
    if eval > α then α ← eval
    if α ≥ β then return α
return α

CS 3793 Artificial Intelligence
Game Playing – 8

α-β MIN-VALUE Procedure

Parameters:
α is a known max value in an ancestor.
β is a known min value in an ancestor.

function MIN-VALUE(state, bound, α, β)
if TERMINAL(state) or bound = 0
    then return Evaluation(state)
for each s-successor in Expand(s)
do eval ← MAX-VALUE(s-successor, bound−1, α, β)
    if eval < β then β ← eval
    if β ≤ α then return β
return β

CS 3793 Artificial Intelligence
Game Playing – 9

Example

Minimax with Alpha–Beta Pruning

MAX

MIN

MAX

MIN

Issues

Performance of Minimax and Alpha-Beta
□ b = branching factor
   d = depth of search
□ Minimax visits every state from level 0 to d.
   \[ \sum_{i=0}^{d} b^i = \frac{b^{d+1} - 1}{b - 1} \in O(b^d) \]
□ Alpha-Beta visits as few as Ω(b^{d/2}) states.
   Depends on a good ordering from Expand.
   Actual programs approach the minimum bound.
□ Alpha-beta pruning allows programs to look ahead nearly twice as many moves as minimax.

CS 3793 Artificial Intelligence
Game Playing – 11
Other Issues

- Horizon problem
- Quiescence
- Data bases of openings and end games
- Games of chance