⑤ Adversarial Search

★ Types of Games

⇒ Axes

- → Deterministic or Stochastic?
- → One, two or many players?
- → Zero Sum?
- → Perfect information ?
  (Can you see the state)

⇒ Want algorithms for calculating a Strategy (policy) which recommends a move from each state.

★ Deterministic Games

⇒ Many possible formalization, one is:

- → States S (start at S_0)
- → Players P={1..N} (usually take turns)
- → Actions: A (may depend on player / state)
- → Transition Function: $S \times A \rightarrow S$
- → Terminal Test: $S \rightarrow \{t, f\}$
- → Terminal Utilities: $S \times P \rightarrow R$

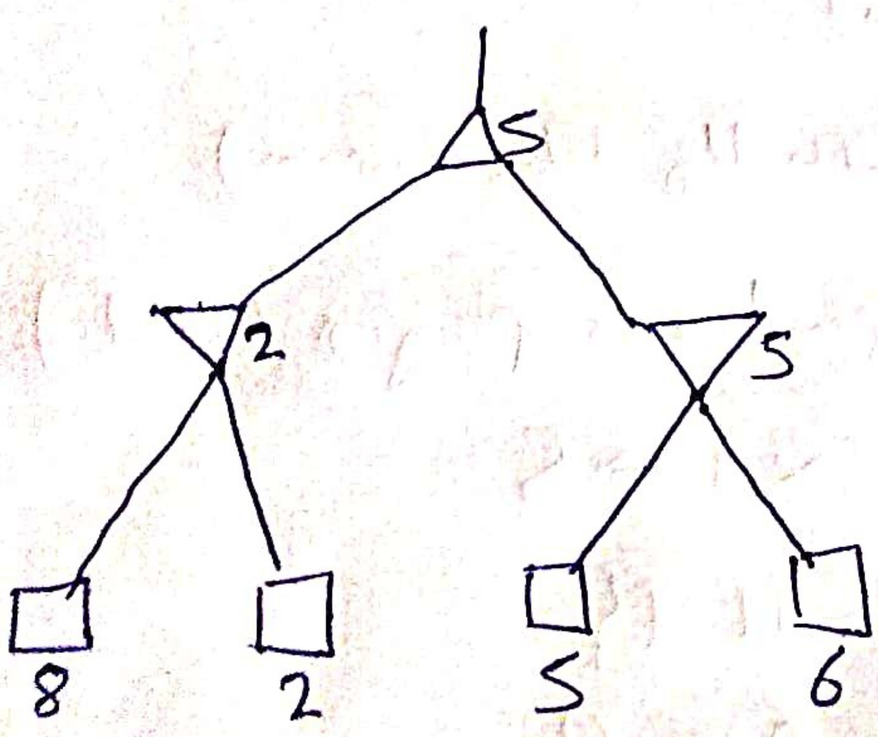
⇒ Solution for a player is a Policy:

$(S \rightarrow A)$

Zero-Sum Game	General Game
⇒ Agents have opposite utilities. ⇒ Single value and one maximizes and the other minimizes ⇒ Adversarial, pure competition	⇒ Agents have independent utilities

★ Adversarial Search (Minmax)

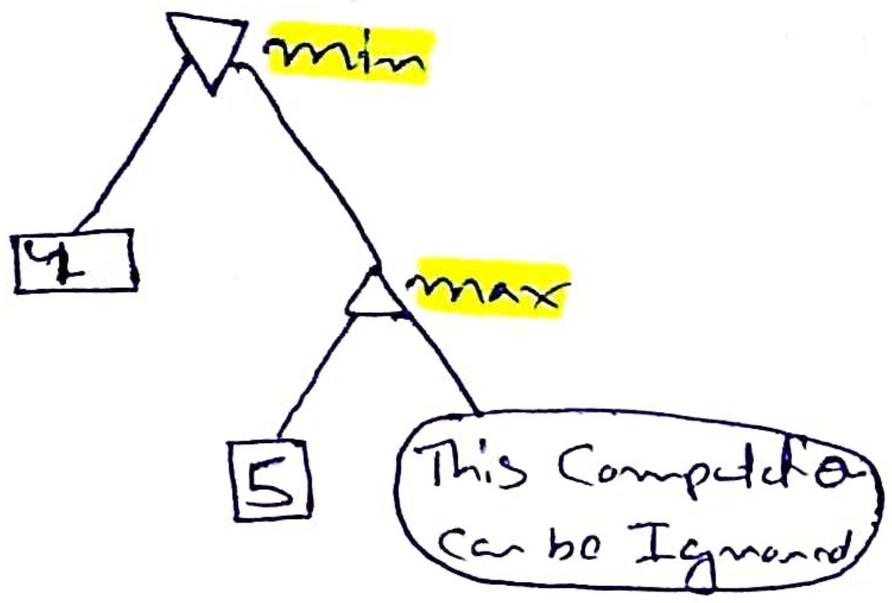
→ A state-space search tree
→ Players alternate turns
→ Compute each node's minmax value:
{The best achievable utility against a rational (optimal) adversary}

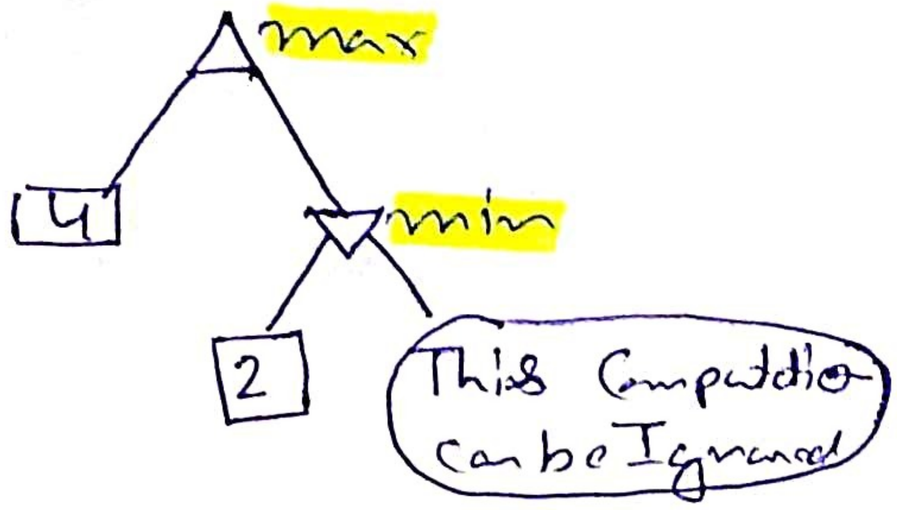



```
def max-value(state):
    initialize V = -∞
    for each successor of state:
        V = max(V, min-value(successor))
    return V
```

```
def min-value(state
    initialize V = +∞
    for each successor of state:
        V = min(V, max-value(successor
    return V
```

* Alpha-Beta Pruning





α : MAX's best option on path to root

β : MIN's best option on p

✓ def max-value(state, α, β):

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, α, β):

Initialize $v = +\infty$

for each successor of state:

$v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$

if $v \leq \alpha$ return v

$\beta = \min(\beta, v)$

return v

★ Alpha-Beta pruning Properties

⇒ This pruning has no effect on minimax value computed for the root!

⇒ Values of intermediate nodes might be wrong.

⇒ Good child ordering improves effectiveness of Pruning.

★ Resource Limits

Problem: In realistic game, cannot search to leaves!

Solution: Depth-limited Search

- → Search only to a limited depth in the tree.
- → Replace terminal utilities with a evaluation function for non-terminal positions.

⇒ Guarantee of optimal play is gone

⇒ Use iterative deepening for an anytime algorithm.

★ 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.

★ Synergies between Evaluation Function and Alpha-Beta

⇒ Alpha-Beta: Amount of pruning depends on expansion ordering.

↳ Evaluation function can provide guidance to expand most promising nodes first