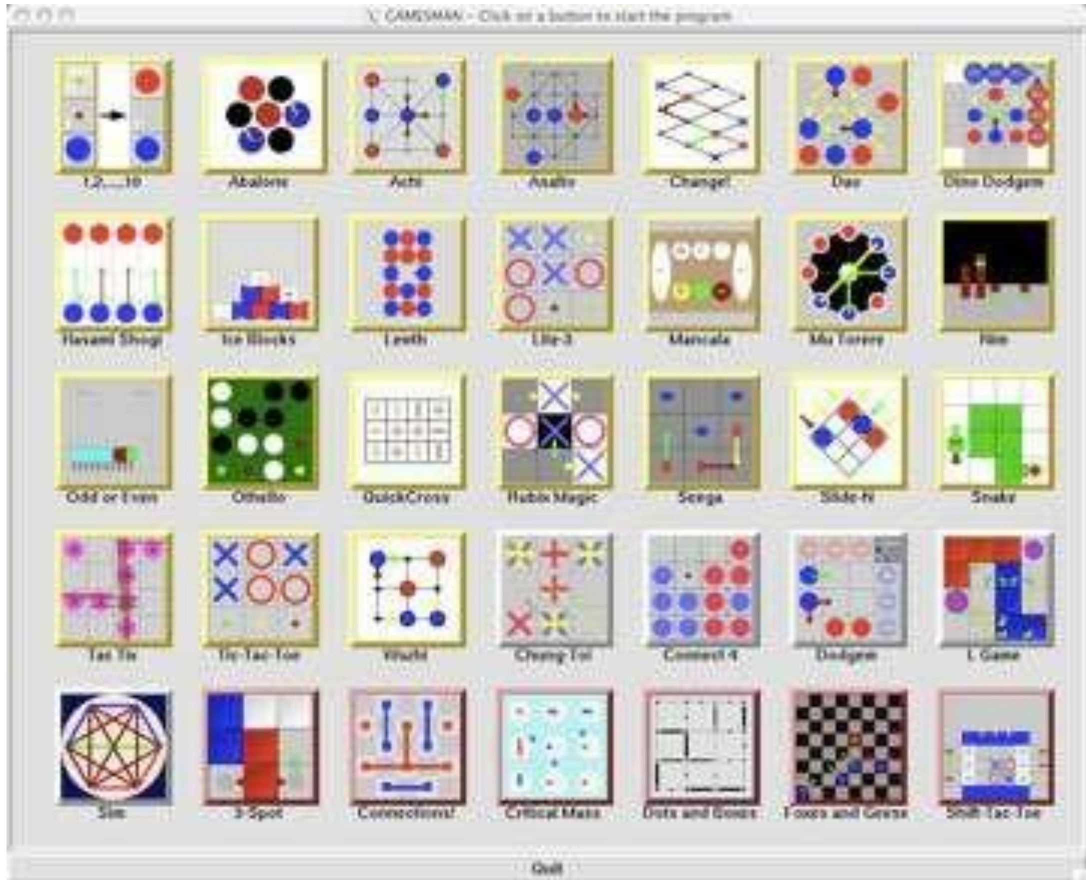
GAME Theory – The Science of Strategies

Adversarial Search

- Multi-agent environment:
  - any given agent needs to consider the actions of other agents and how they affect its own welfare
  - introduce possible contingencies into the agent's problem-solving process
  - cooperative vs. competitive
- Adversarial search problems: agents have conflicting goals -- games

Games vs. Search Problems

- "Unpredictable" opponent
  - specifying a move for every possible opponent reply
- Time limits
  - unlikely to find goal, must approximate



AI and Games

- In AI, “games” have special format:
  - deterministic, turn-taking, 2-player, zero-sum games of perfect information
  - Zero-sum describes a situation in which a participant’s gain or loss is exactly balanced by the losses or gains of the other participant(s)
  - Or, the total payoff to all players is the same for every instance of the game (constant sum)

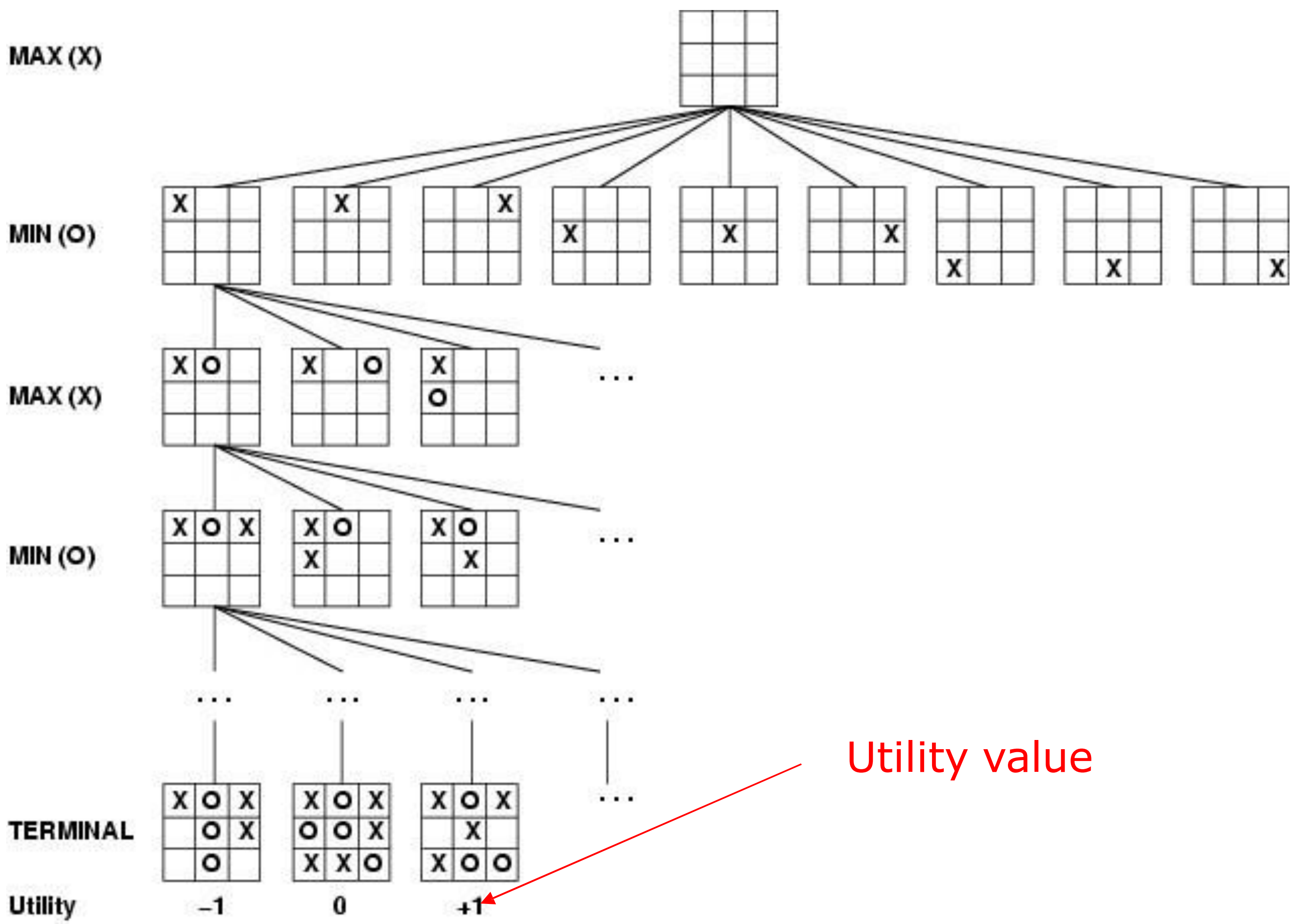
Go! 围棋

Game Problem Formulation

- A game with 2 players (MAX and MIN, MAX moves first, turn-taking) can be defined as a search problem with:
  - initial state: board position
  - player: player to move
  - successor function: a list of legal (move, state) pairs
  - goal test: whether the game is over – terminal states
  - utility function: gives a numeric value for the terminal states (win, loss, draw)

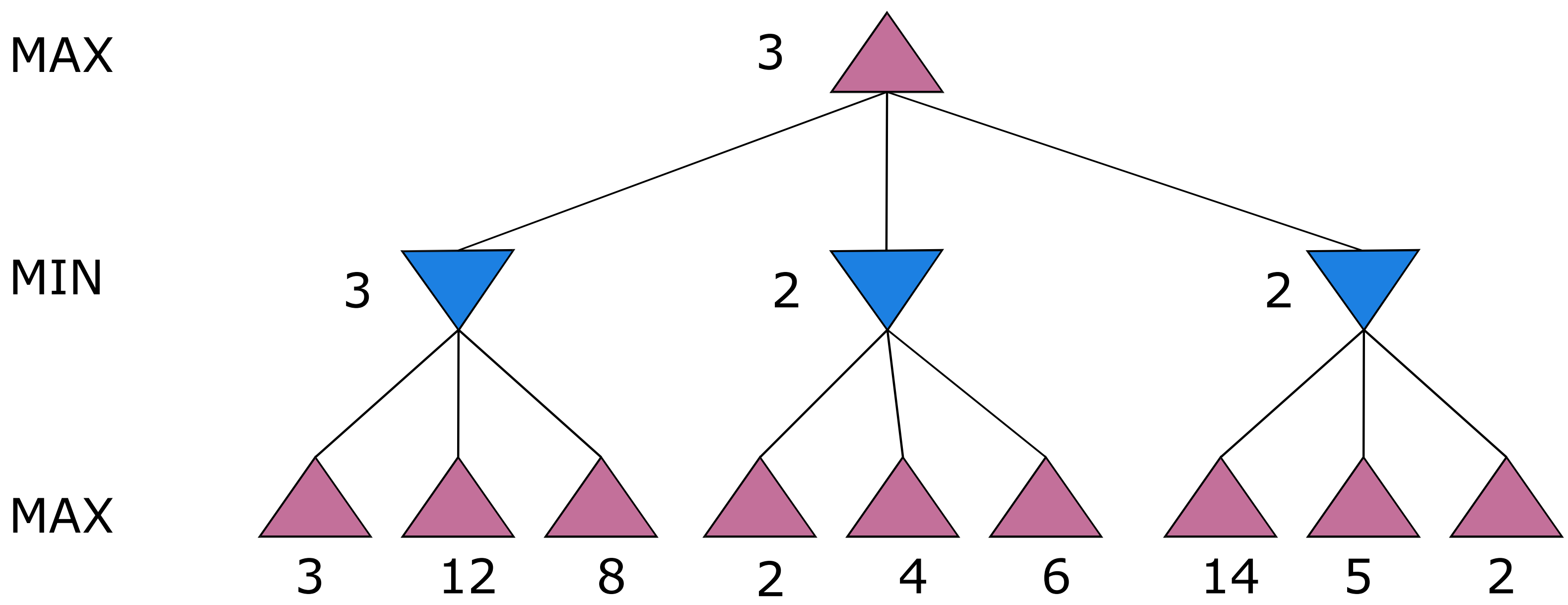
- Game tree = initial state + legal moves

Game Tree (2-player, deterministic)



Optimal Strategies

- MAX must find a contingent strategy, specifying MAX's move in:
  - the initial state
  - the states resulting from every possible response by MIN
- E.g., 2-ply game (the tree is one move deep, consisting of two half-moves, each of which is called a ply):



Minimax Value

- Perfect play for deterministic game, assume both players play optimally
- Idea: choose move to position with highest minimax value = best achievable payoff against best play

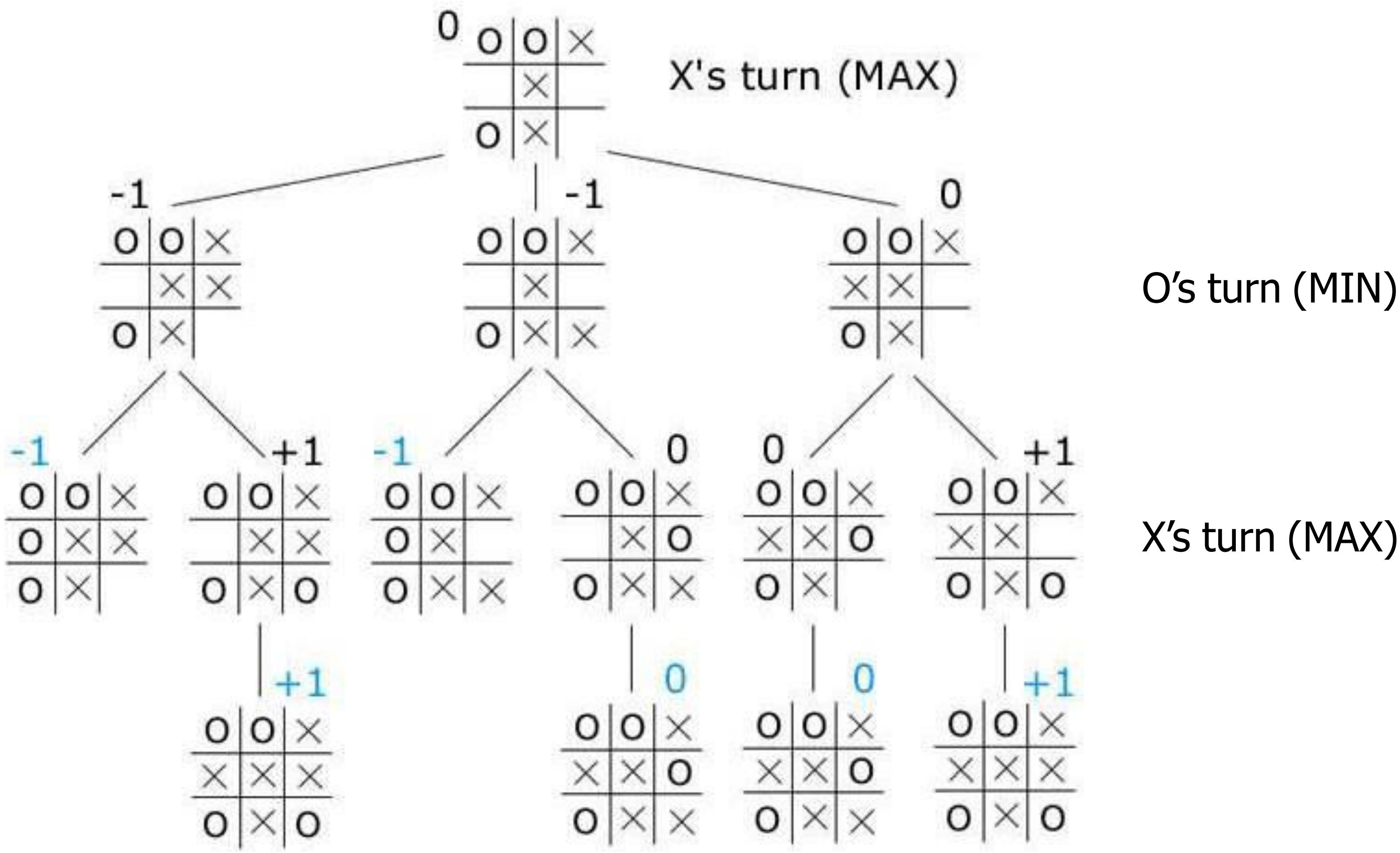
$$MINIMAX-VALUE(n) =$$

$$Utility(n) \quad \text{if n is a terminal state}$$

$$\max_{s \in Successors(n)} MINIMAX(s) \quad \text{if n is a MAX node}$$

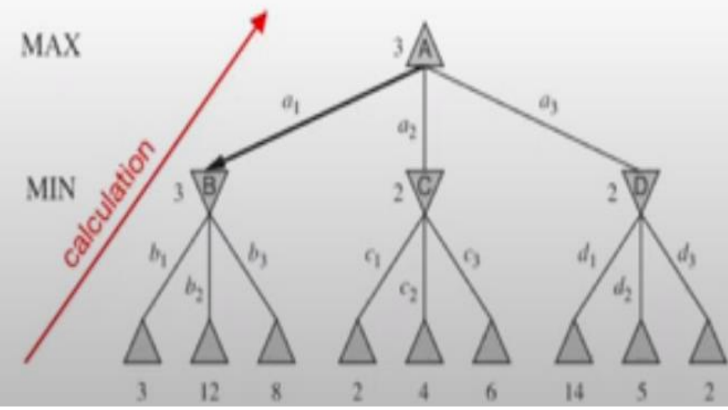
$$\min_{s \in Successors(n)} MINIMAX(s) \quad \text{if n is a MIN node}$$

A Partial Game Tree for Tic-Tac-Toe



How to calculate MinMax Value ?

$$
\text{MINIMAX}(s) =
\begin{cases}
\text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\
\max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MAX} \\
\min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MIN}
\end{cases}
$$



Minimax with Tic-Tac-Toe

- Work on board.

- Walk through a real tic-tac-toe program.

Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

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
```

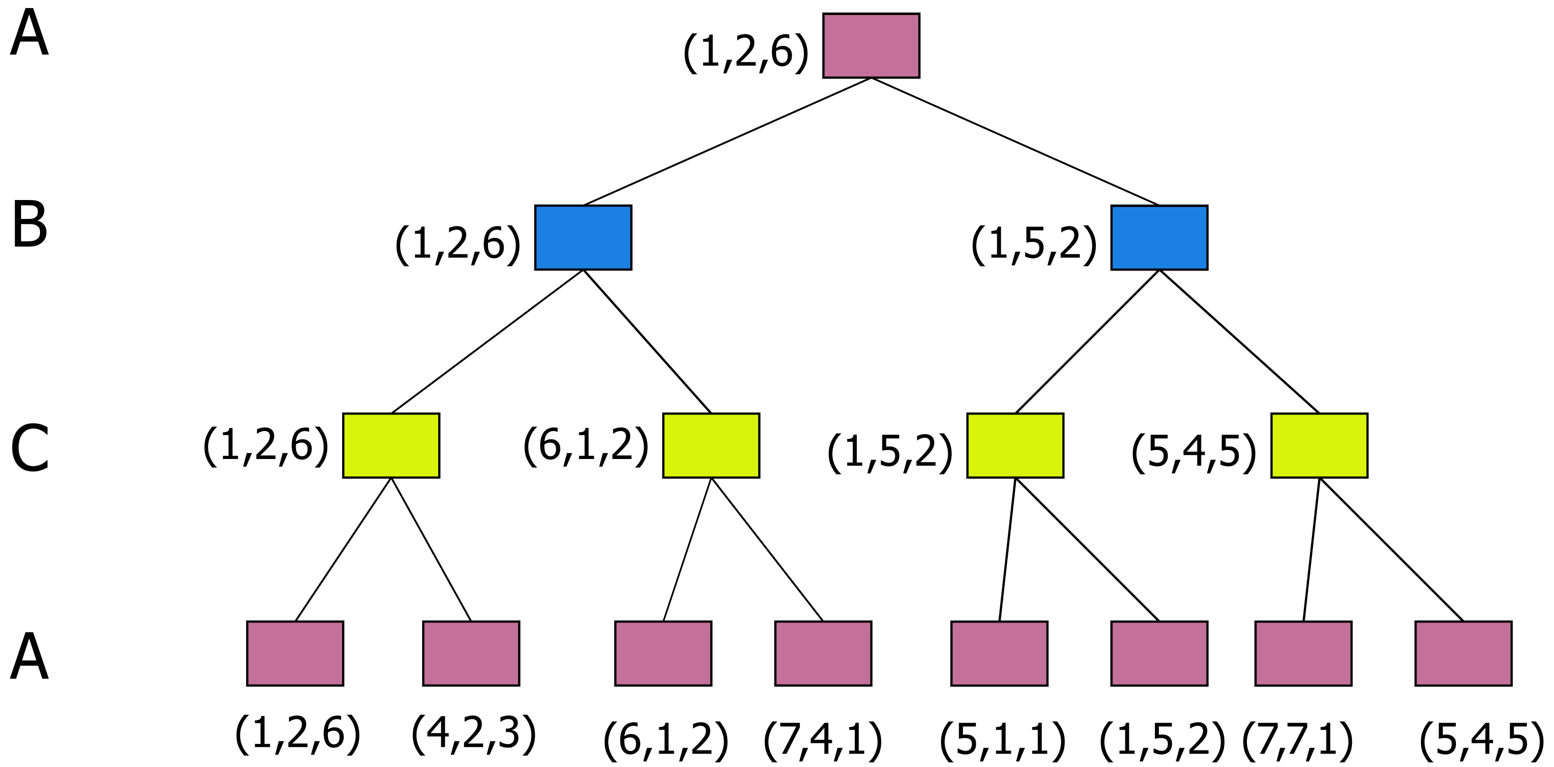
Analysis of Minimax

- Optimal play for MAX assumes that MIN also plays optimally, what if MIN does not play optimally?
- A complete depth-first search?
  - Yes
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - $O(bm)$ (depth-first exploration)

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  ☐ exact solution completely infeasible

Optimal Decisions for Multiplayer Games

- Extend minimax idea to multiplayer



Interesting Thoughts

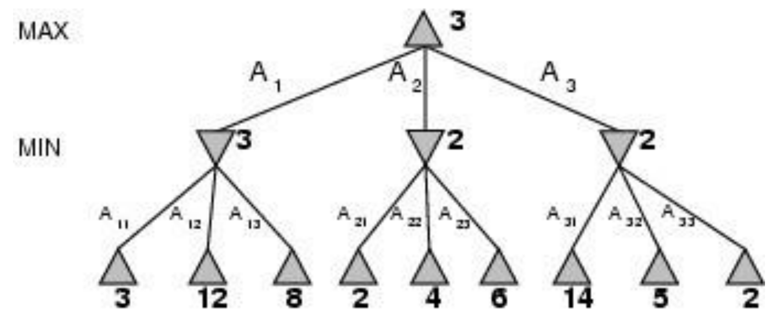
- Multiplayer games usually involve alliances, which can be a natural consequence of optimal strategies

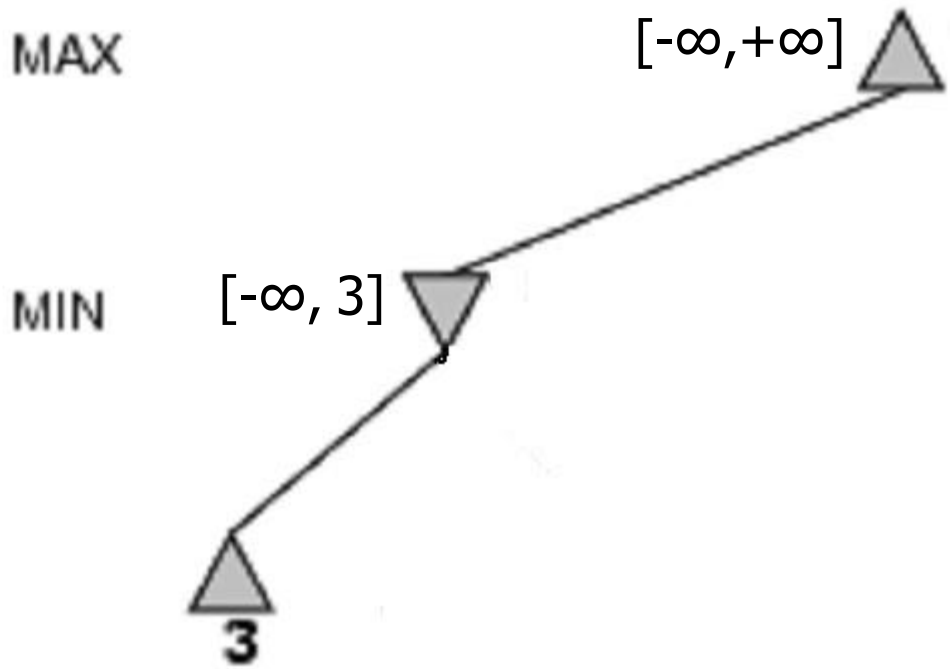
- If the game is non zero-sum, collaboration can also occur
  - For example, a terminal state with utilities <Va = 1000, Vb = 1000>
  - The optimal strategy is for both players to do everything possible to reach this state

α-β Pruning

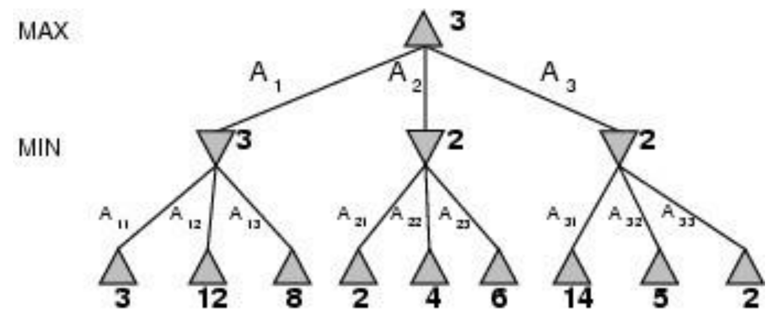
- The number of game states with minimax search is exponential in the # of moves
- Is it possible to compute the correct minimax decision without looking at every node in the game tree?
- Need to prune away branches that cannot possibly influence the final decision

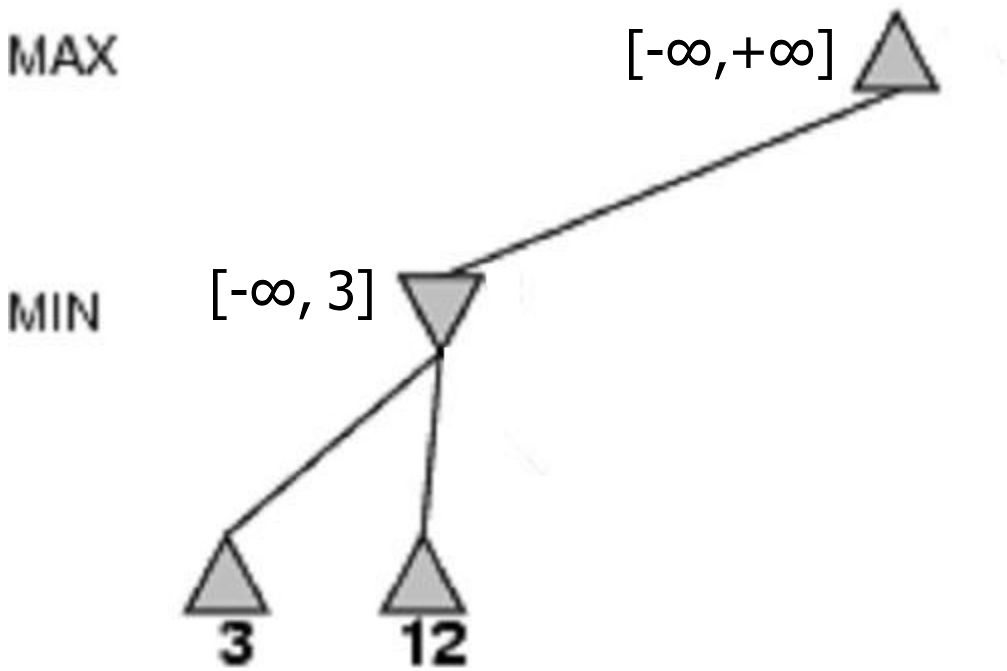
α-β Pruning Example



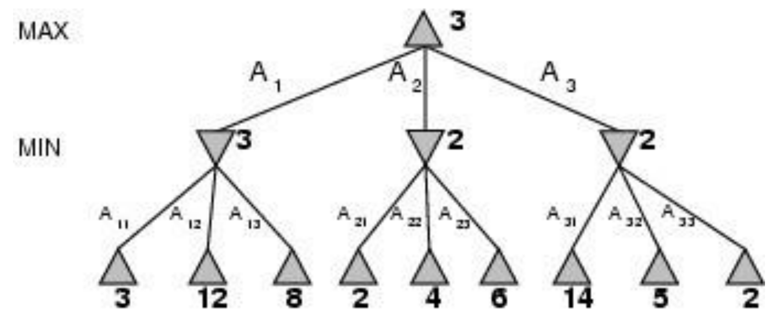


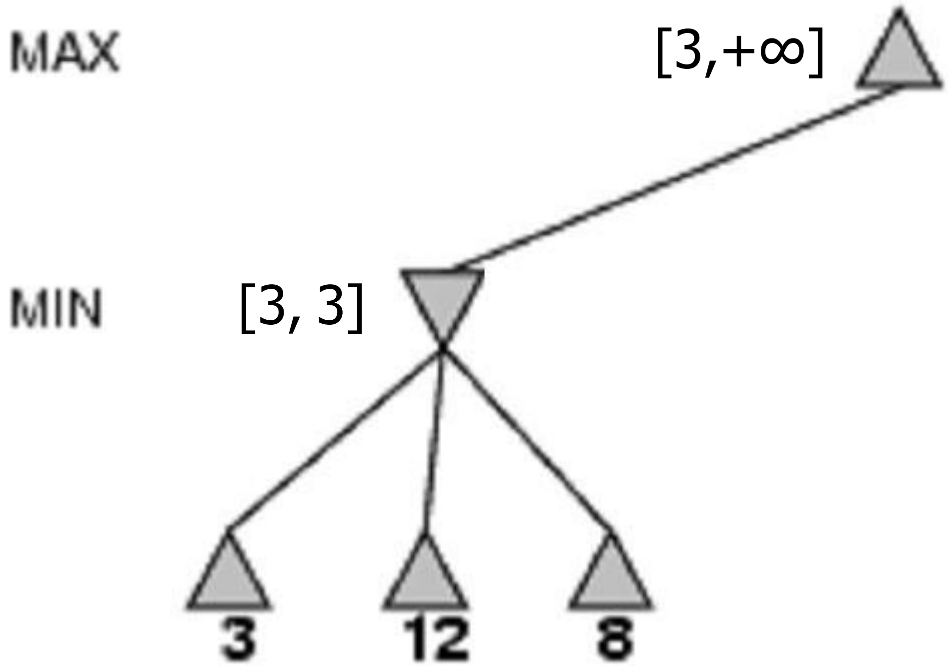
α-β Pruning Example



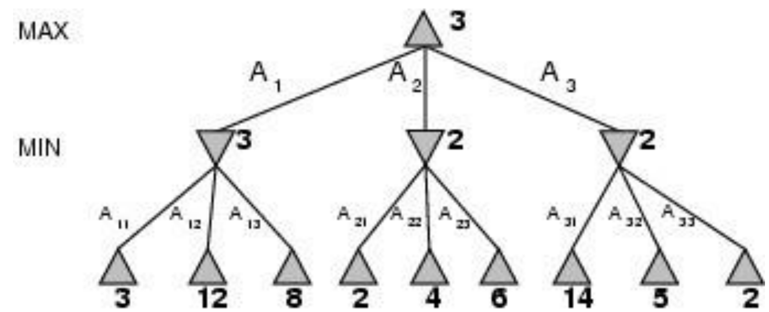


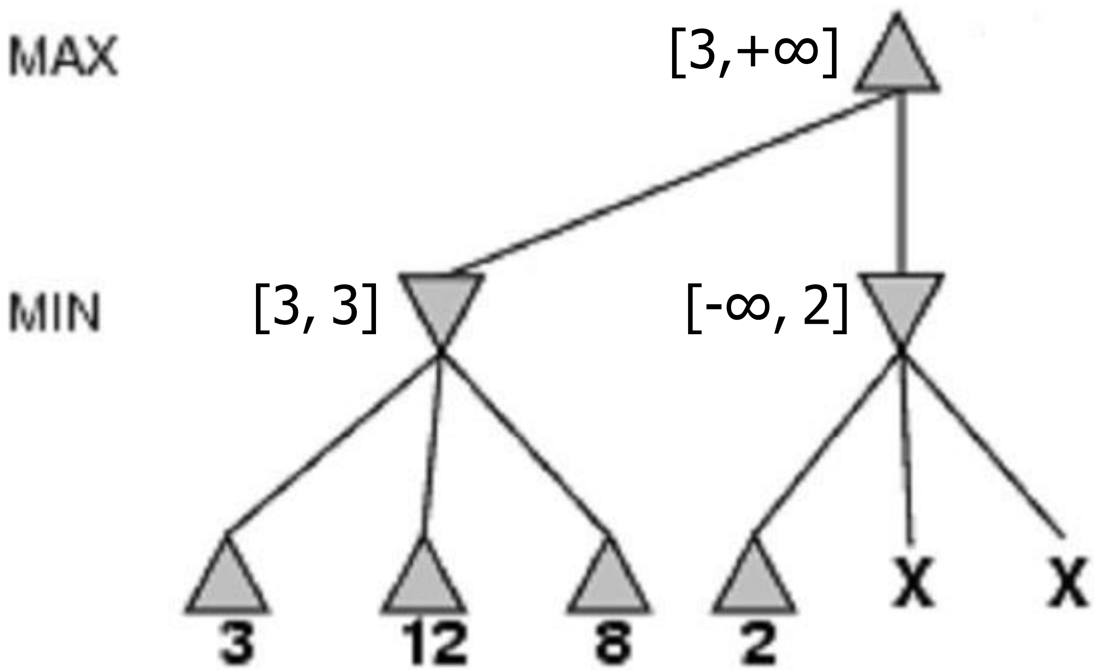
α-β Pruning Example



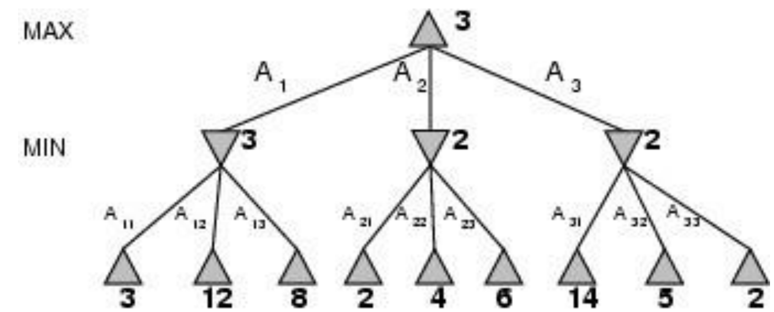


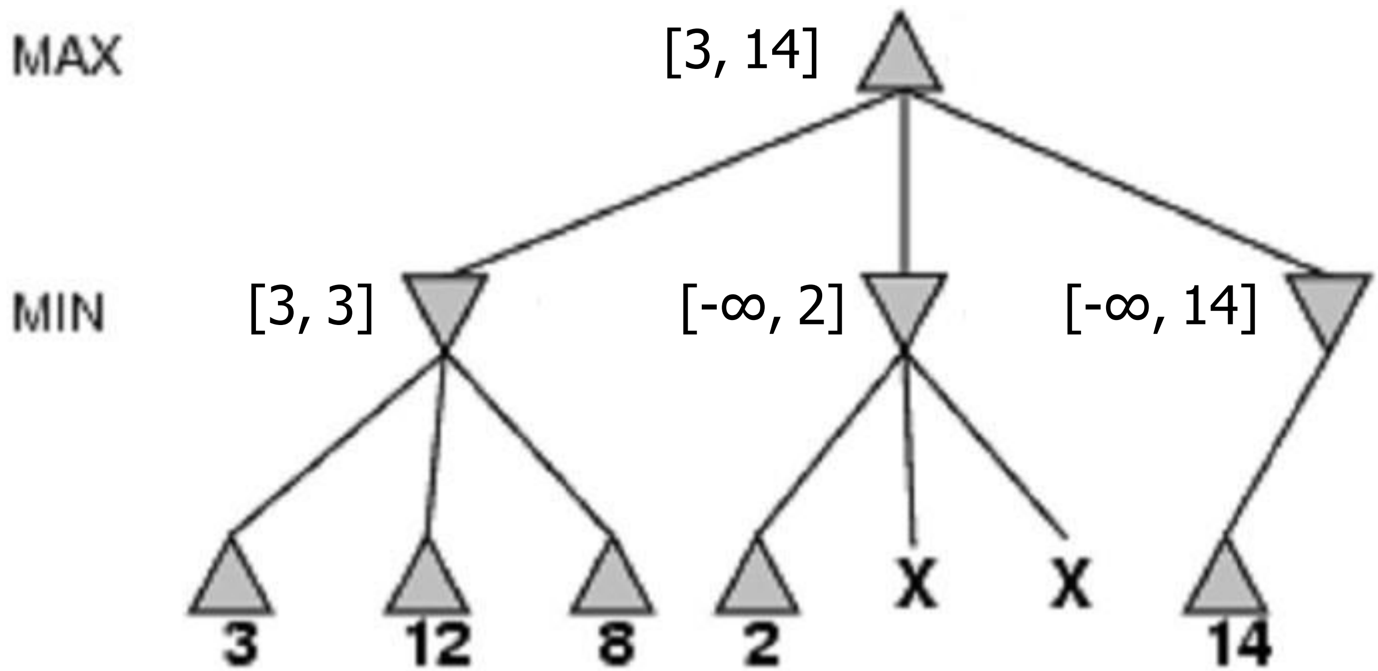
α-β Pruning Example



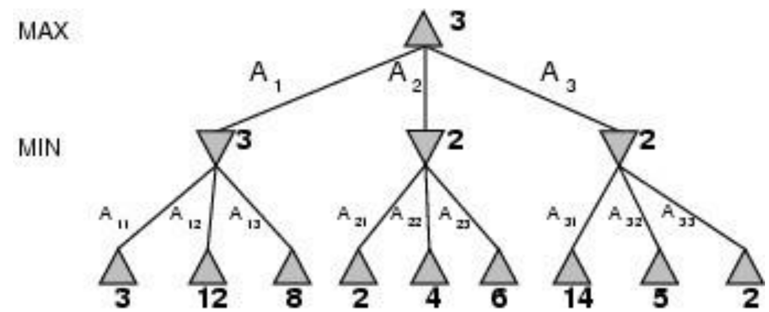


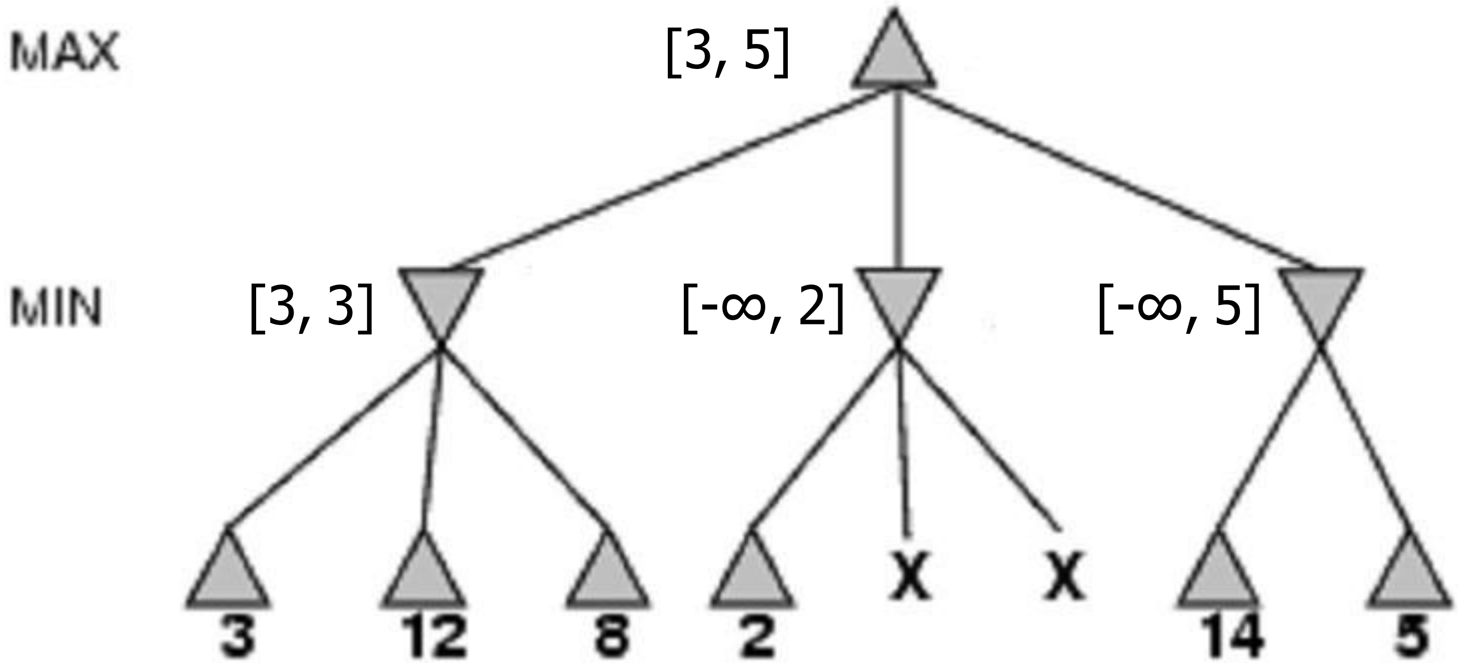
α-β Pruning Example



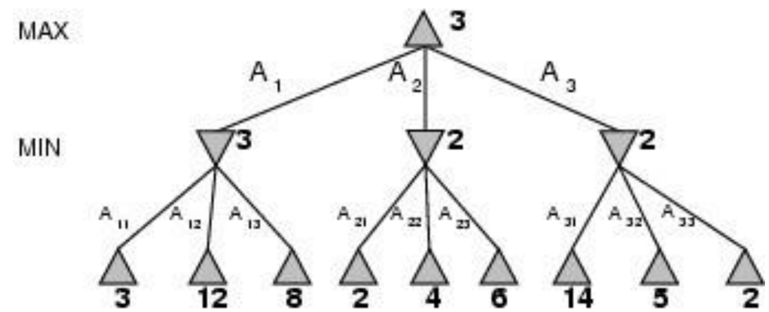


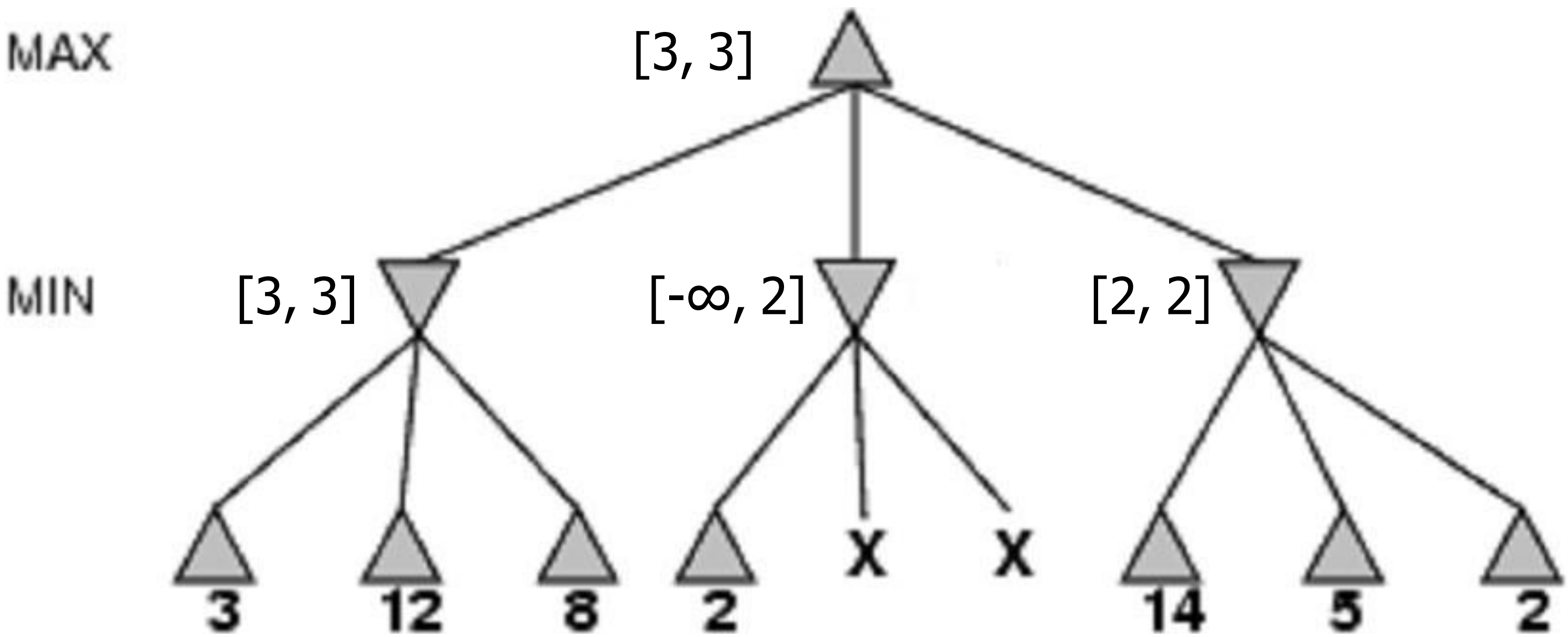
α-β Pruning Example





α-β Pruning Example



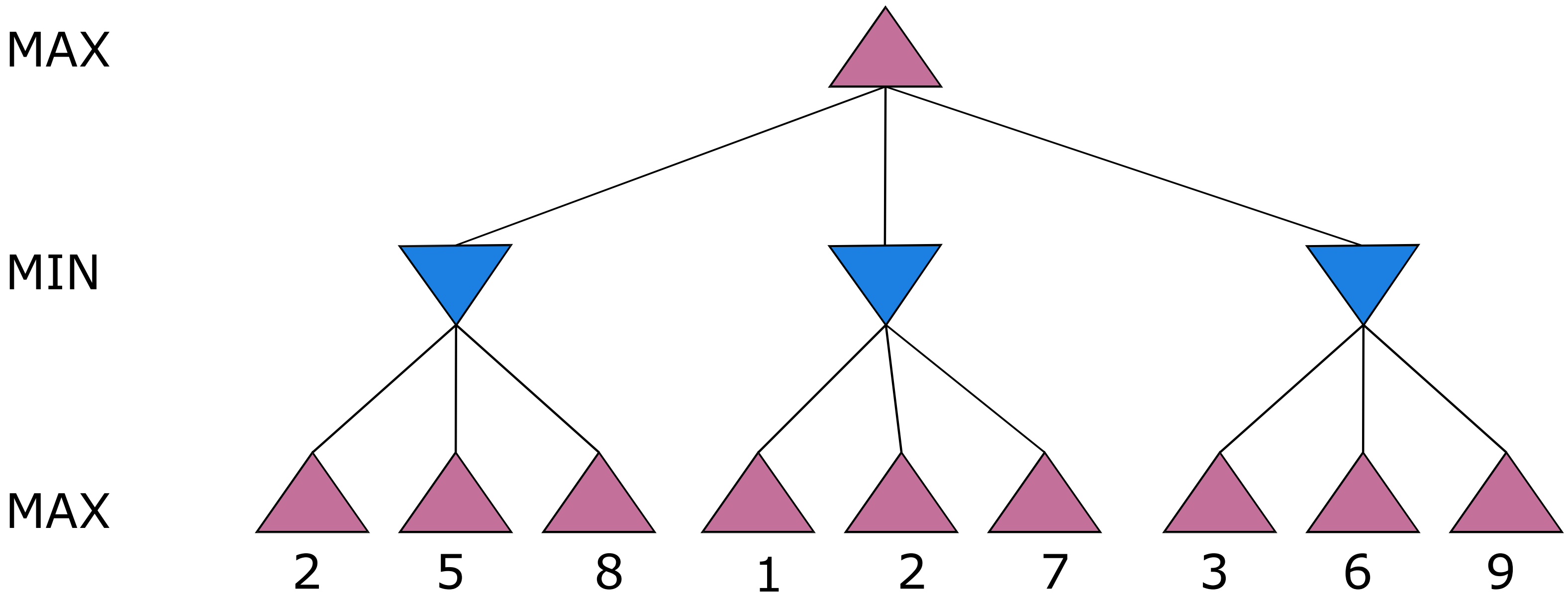


A Simple Formula

$$
\begin{aligned}
&MINIMAX-VALUE(root)\\
&= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2))\\
&= \max(3, \min(2,x,y), 2)\\
&= \max(3, z, 2) \qquad \text{where } z \le 2\\
&= 3
\end{aligned}
$$

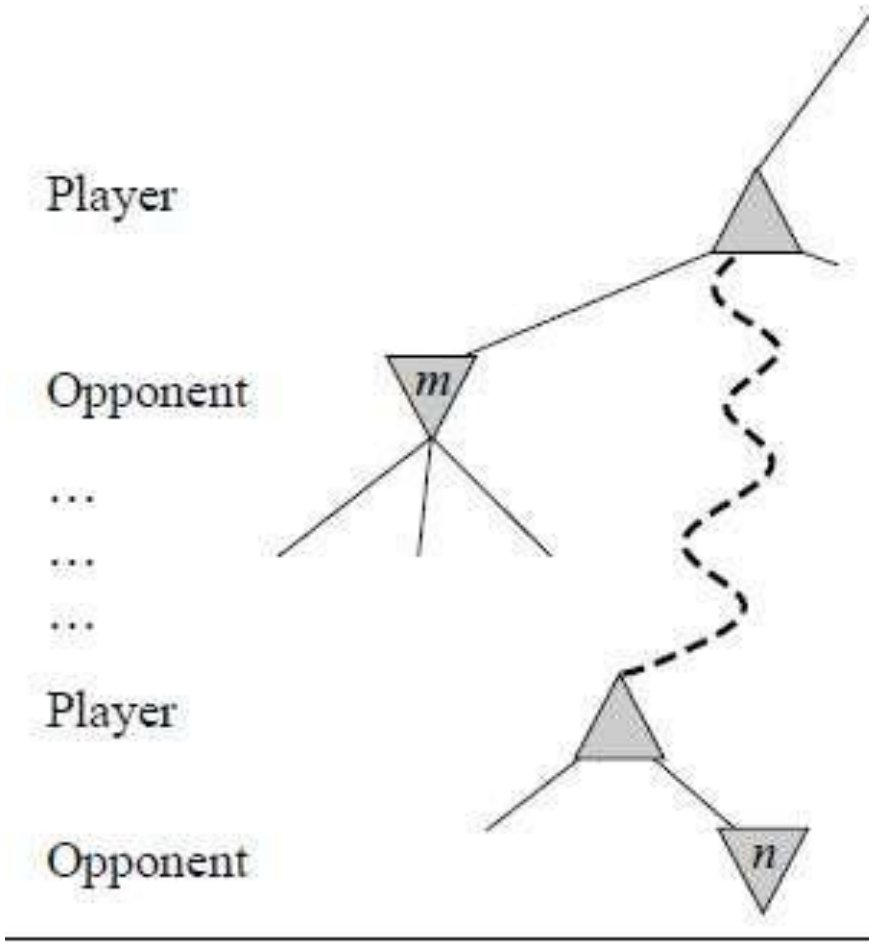
The value of the root and hence the minimax decision are independent of the values of pruned leaves x and y

α-β Pruning Exercise



Basic Idea of α-β

- Consider a node n such that Player has a choice of moving to
- If Player has a better choice m either at the parent of n or at any choice point further up, then n will never be reached in actual play
- α-β pruning gets it name from the two parameters that describe bounds on the backed-up values



Definitions of α and β

- α = the value of the best (highest-value) choice we have found so far at any choice point along the path for MAX

- β = the value of the best (lowest-value) choice we have found so far at any choice point along the path for MIN

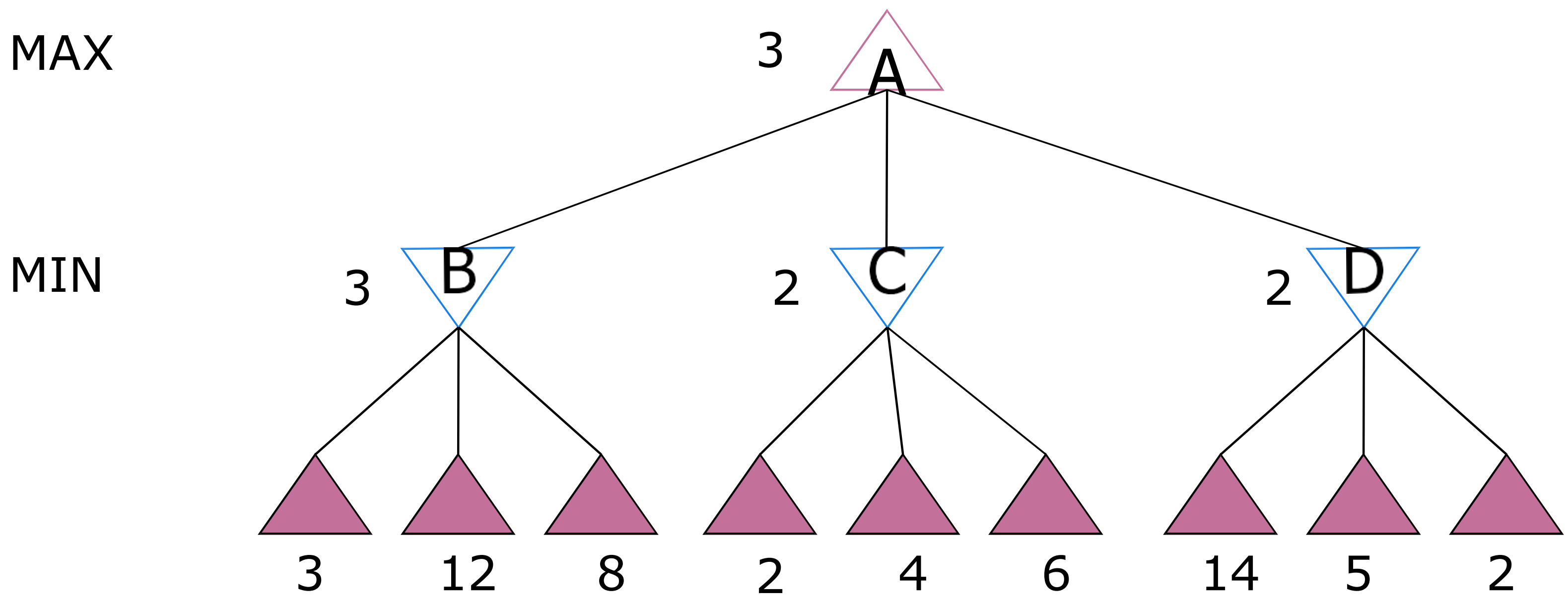
- α-β search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is worse than the current α or β for MAX or MIN respectively

```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in SUCCESSORS(state) with value v
```

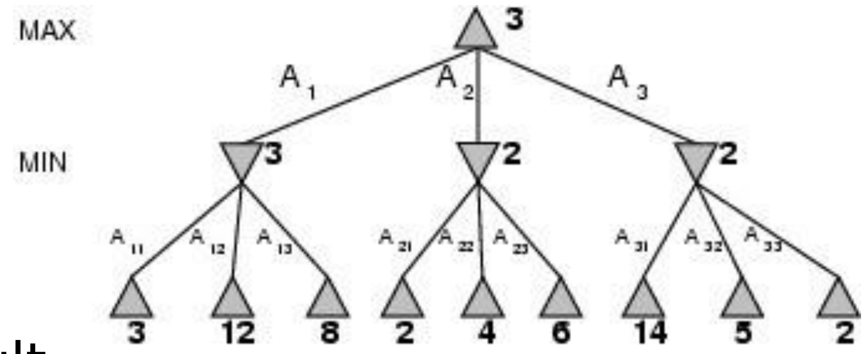
```
function MAX-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state
   if TERMINAL-TEST(state) then return UTILITY(state)
   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
```

```
function MIN-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 ← MIN(v, MAX-VALUE(s, α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Now, Trace The Behavior



Analysis of α-β



- Pruning does not affect final result

- The effectiveness of alpha-beta pruning is highly dependent on the order of successors
- It might be worthwhile to try to examine first the successors that are likely to be best

- With "perfect ordering," time complexity = $O(b^{m/2})$
  - effective branching factor becomes $\sqrt{b}$
  - For chess, 6 instead of 35
  - it can look ahead roughly twice as far as minimax in the same amount of time
  - Ordering in chess: captures, threats, forward moves, and then backward moves

Random Ordering?

- If successors are examined in random order rather than best-first, the complexity will be roughly $O(b^{3m/4})$

- Adding dynamic move-ordering schemes, such as trying first the moves that were found to be best last time, brings us close to the theoretical limit

- The best moves are often called killer moves (killer move heuristic)

Dealing with Repeated States

- In games, repeated states occur frequently because of transpositions --
  - different permutations of the move sequence end up in the same position
  - e.g., [a1, b1, a2, b2] vs. [a1, b2, a2, b1]
- It's worthwhile to store the evaluation of this position in a hash table the first time it is encountered
  - similar to the "explored set" in graph-search
- Tradeoff:
  - Transposition table can be too big
  - Which to keep and which to discard

Imperfect, Real-Time Decisions

- Minimax generates the entire game search space
- Alpha-beta prunes large part of it, but still needs to search all the way to terminal states
- However, moves must be made in reasonable amount of time

- Standard approach: turning non-terminal nodes into terminal leaves
  - cutoff test: replaces terminal test, e.g., depth limit
  - heuristic evaluation function = estimated desirability or utility of position