ADVERSARIAL SEARCH AND GAMES

Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

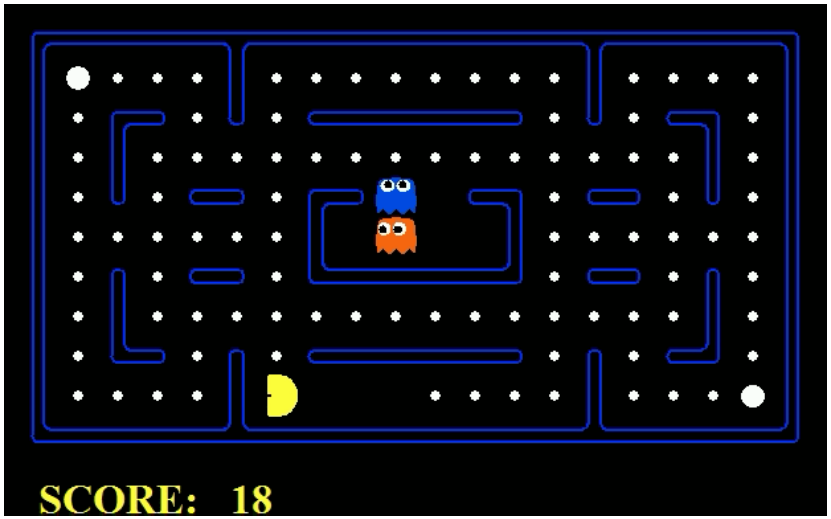
Outline

- Two-player zero-sum games
- Optimal decisions in games
- Heuristic alpha-beta tree search
- Stochastic games

Two-player zero-sum games

Game theory and AI games

- Game theory views any multiagent environment as a game.
  - The impact of each agent on the others is **significant**, regardless of whether the agents are cooperative or competitive.
- Each agent needs to consider the actions of other agents and how they affect its own welfare.



Two-player zero-sum games

- The games most commonly studied within AI are

Perfect information

Fully observable

Zero-sum

What is good for one player is just as bad for the other.

No "win-win" outcome

Deterministic

Two-player

Turn-taking

- The terms are slightly different from those in search.
  - Action → **Move** and State → **Position**

Two-player zero-sum games

- Two players are MAX and MIN. MAX moves first.
- The players take turns moving until the game is over.
- At the end of the game, points are awarded to the winning player and penalties are given to the loser.

- The two players are rational, trying to maximize their utilities.

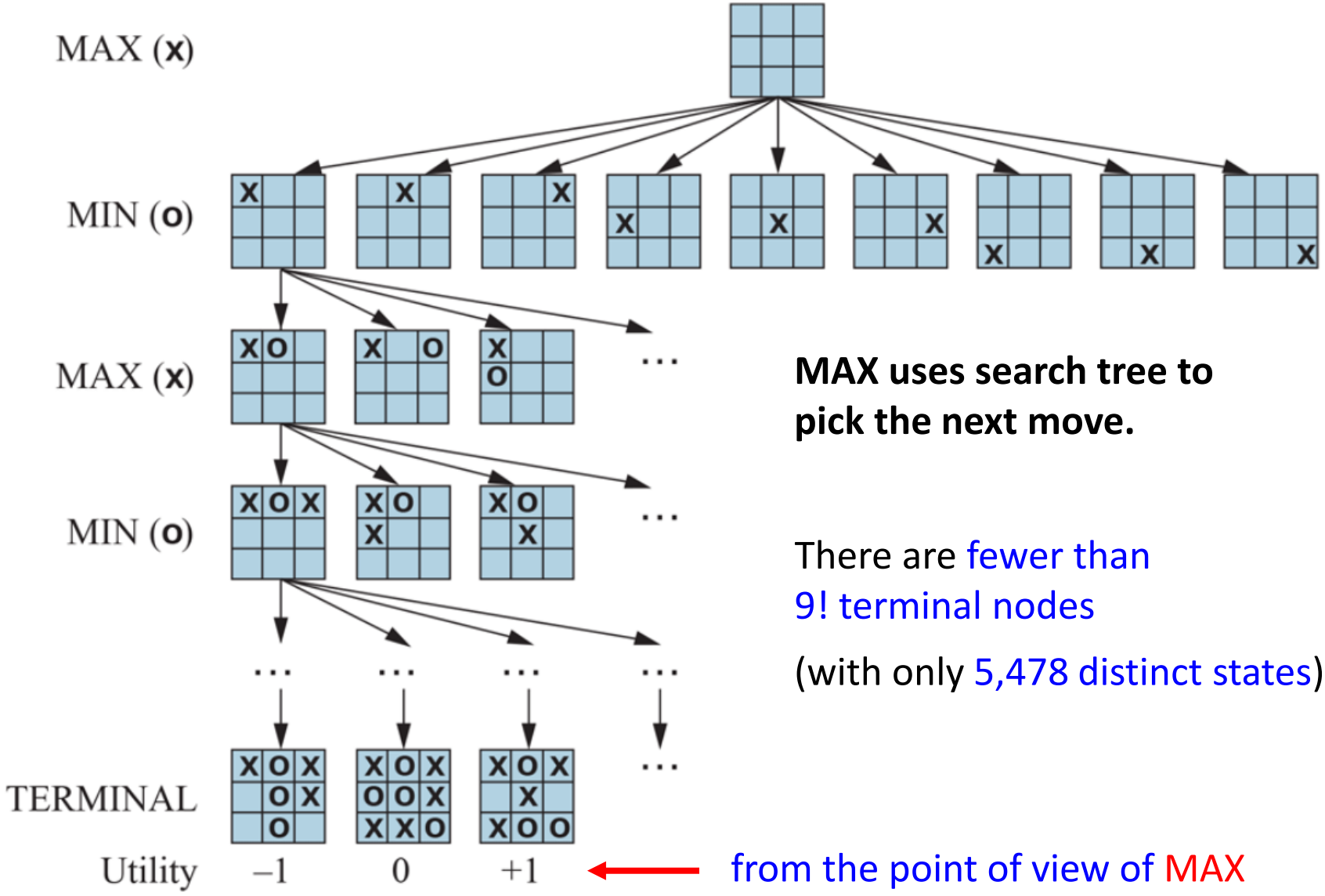
Game formulation

- $\boldsymbol{S_0}$: The initial state, which specifies how the game is set up at the start.
- $TO\text{-}MOVE(s)$: The player whose turn it is to move in state
- $\boldsymbol{ACTIONS(s)}$: The set of legal moves in state .
- $\boldsymbol{RESULT(s,a)}$: The transition model, which defines the state resulting from acting in state s.
- $IS\text{-}TERMINAL(s)$: A terminal test, which is true when the game is over and false otherwise.
  - States where the game has ended are called terminal states.
- $UTILITY(s,p)$: A utility function defines the final numeric value to player when the game ends in terminal state
  - E.g., chess: win (**+1**), lose (**-1**) and draw (**0**), backgammon: [0, 192]

State space graph and Game tree

- The initial state, $ACTIONS$ function, and $RESULT$ function defines the state space graph.
- The complete game tree is a search tree that follows every sequence of moves all the way to a terminal state.
  - It may be infinite if the state space itself is unbounded or if the rules of the game allow for infinitely repeating positions

A game tree for Tic-tac-toe



MAX uses search tree to pick the next move.

There are fewer than 9! terminal nodes

(with only 5,478 distinct states)

from the point of view of MAX

Examples of game: Checkers

- Complexity
  - ~ 10^{18} nodes, which may require 100k years with 106 positions/sec
- **Chinook** (1989-2007)
  - The first computer program that won the world champion title in a competition against humans
  - 1990: won 2 games in competition with world champion Tinsley (final score: 2-4, 33 draws). 1994: 6 draws
- Chinook's search
  - Ran on regular PCs, played perfectly by using alpha-beta search combining with a database of 39 trillion endgame positions

Examples of game: Chess

- Complexity
  - b ≈ 35, d ≈ 100, 10^{154} nodes (!!)
  - Completely impractical to search this
- **Deep Blue** (May 11, 1997)
  - Kasparov lost a 6-game match against IBM's Deep Blue (1 win Kasp – 2 wins DB) and 3 ties.
- In the future, focus will be to allow computers to **LEARN** to play chess rather than being **TOLD** how it should play

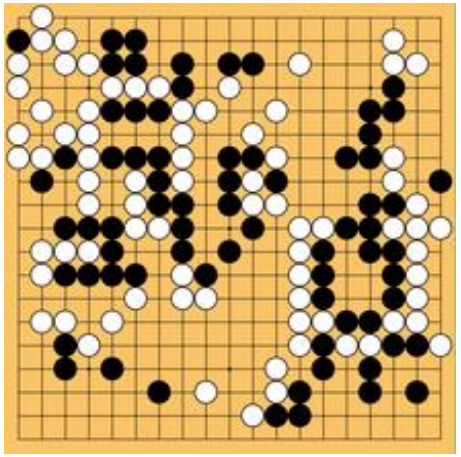
Deep Blue

- Ran on a parallel computer with 30 IBM RS/6000 processors doing alpha–beta search
- Searched up to 30 billion positions/move, average depth 14 (be able to reach to 40 plies)
- Evaluation function: 8000 features
  - highly specific patterns of pieces (~4000 positions)
  - 700,000 grandmaster games in database
- Working at 200 million positions/sec, even Deep Blue would require $\mathbf{10^{100}}$ years to evaluate all possible games.
  - (The universe is only 10^{10} years old.)
- Now: algorithmic improvements have allowed programs running on standard PCs to win World Computer Chess Championships.
  - Pruning heuristics reduce the effective branching factor to less than 3

GO

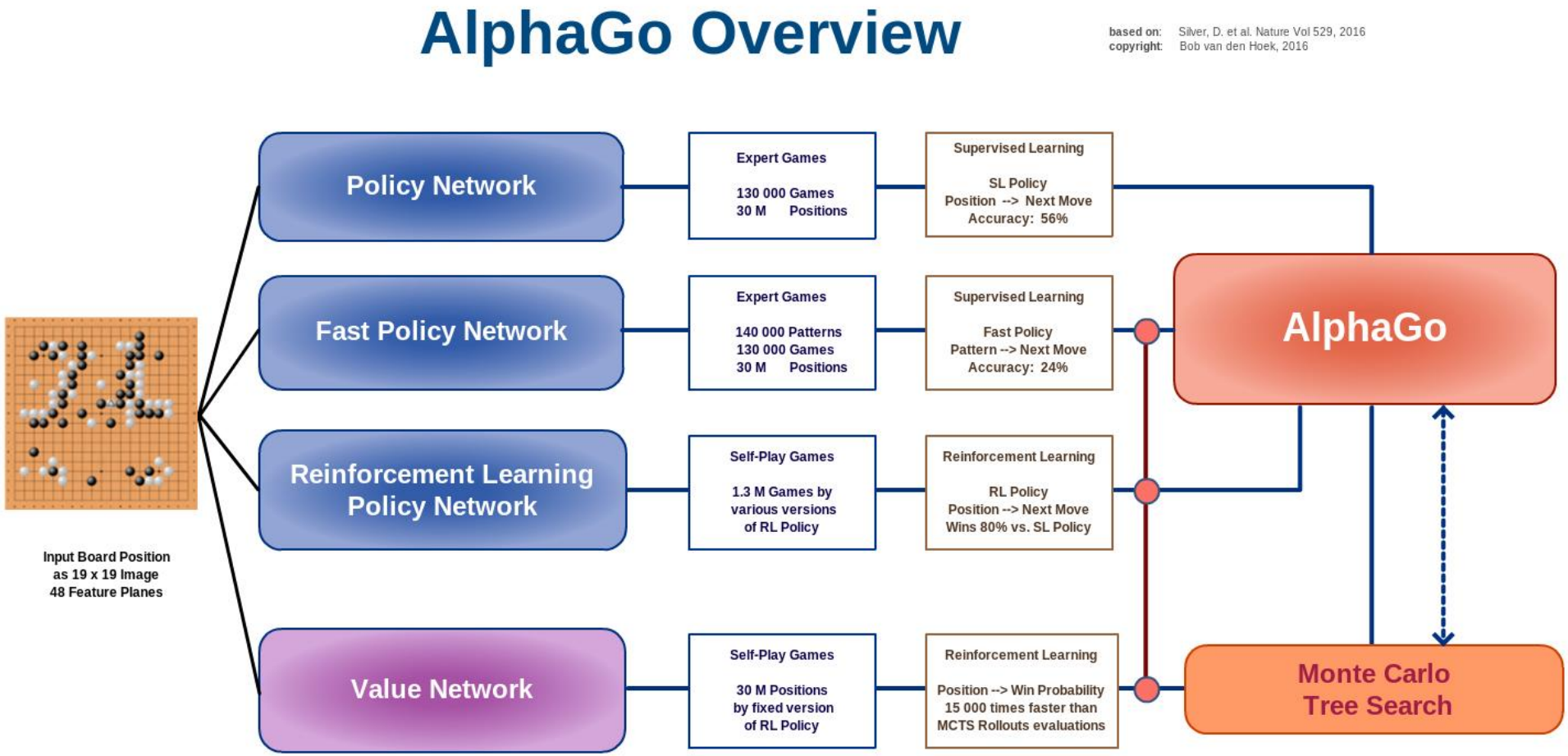
1 million trillion trillion trillion trillion more configurations than chess!

- Complexity
  - Board of 19x19, b ≈ 361, average depth ≈ 200
  - 10^{174} possible board configuration.
  - The control of territory is unpredictable until the endgame



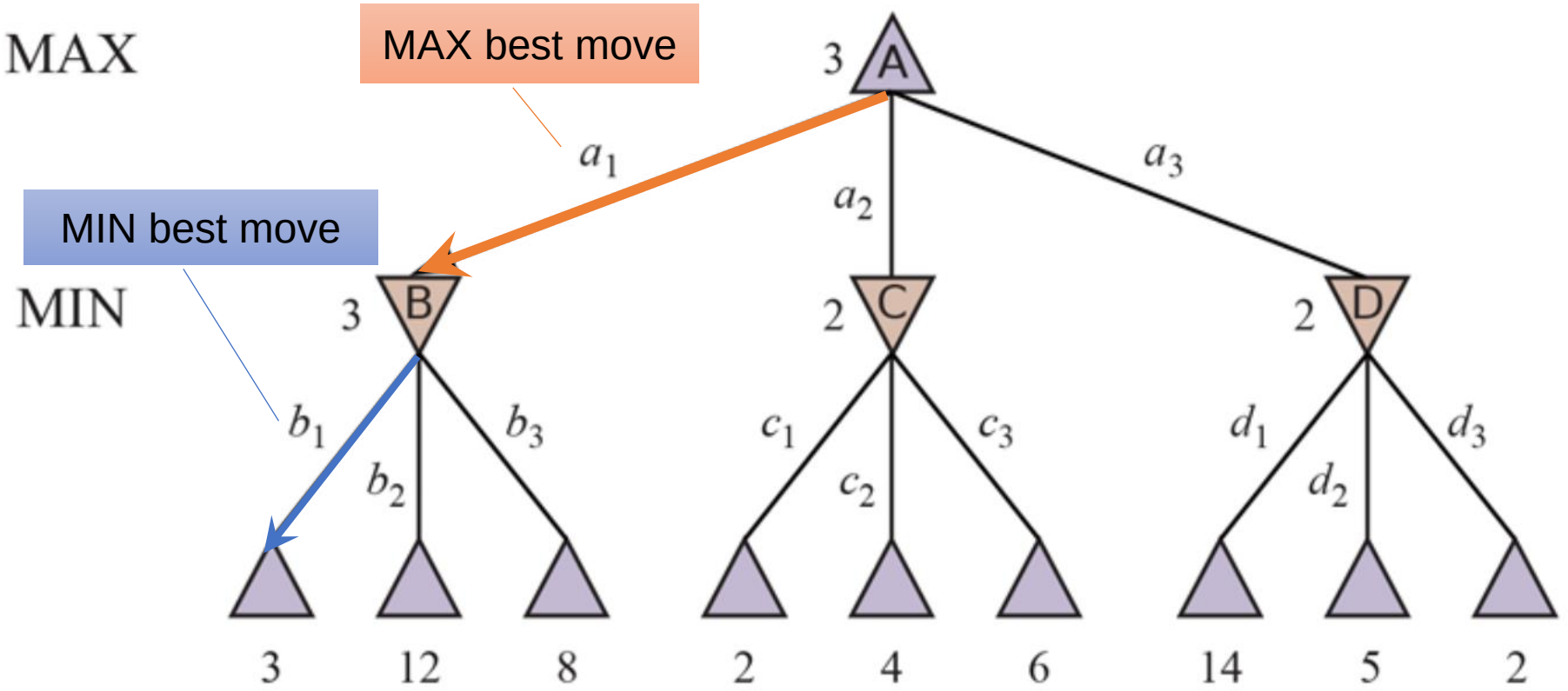
- **AlphaGo** (2016) by Google
  - Beat 9-dan professional Lee Sedol (4-1)
  - Machine learning + Monte Carlo search guided by a “value network” and a “policy network” (implemented using *deep neural network* technology)
  - Learn from human + Learn by itself (self-play games)

An overview of AlphaGo



Optimal decisions in games

A two-ply game tree



MAX's best move at the root is a_1, leading to the state with the highest minimax value. MIN's best reply is b_1, heading to the state with the lowest minimax value.

Δ nodes: MAX's turn to move, ∇ nodes: MIN's turn to move; terminal nodes show the utility values for MAX and other nodes are labeled with their minimax values.

The minimax algorithm

- Assume that both agents play optimally from any state s to the end of the game.
- The minimax value of state s is the utility of being in s.
  - The minimax value of a terminal state is just its utility.
- MAX prefers to move to a state of maximum value, and MIN prefers a state of minimum value.

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s, \textsc{max}) & \text{if } \textsc{Is-Terminal}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{To-Move}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{To-Move}(s) = \textsc{min} \end{cases}$$

For MAX

The minimax algorithm

- Recursively proceed all the way down to the leave nodes and backs up the minimax values as the recursion unwinds.

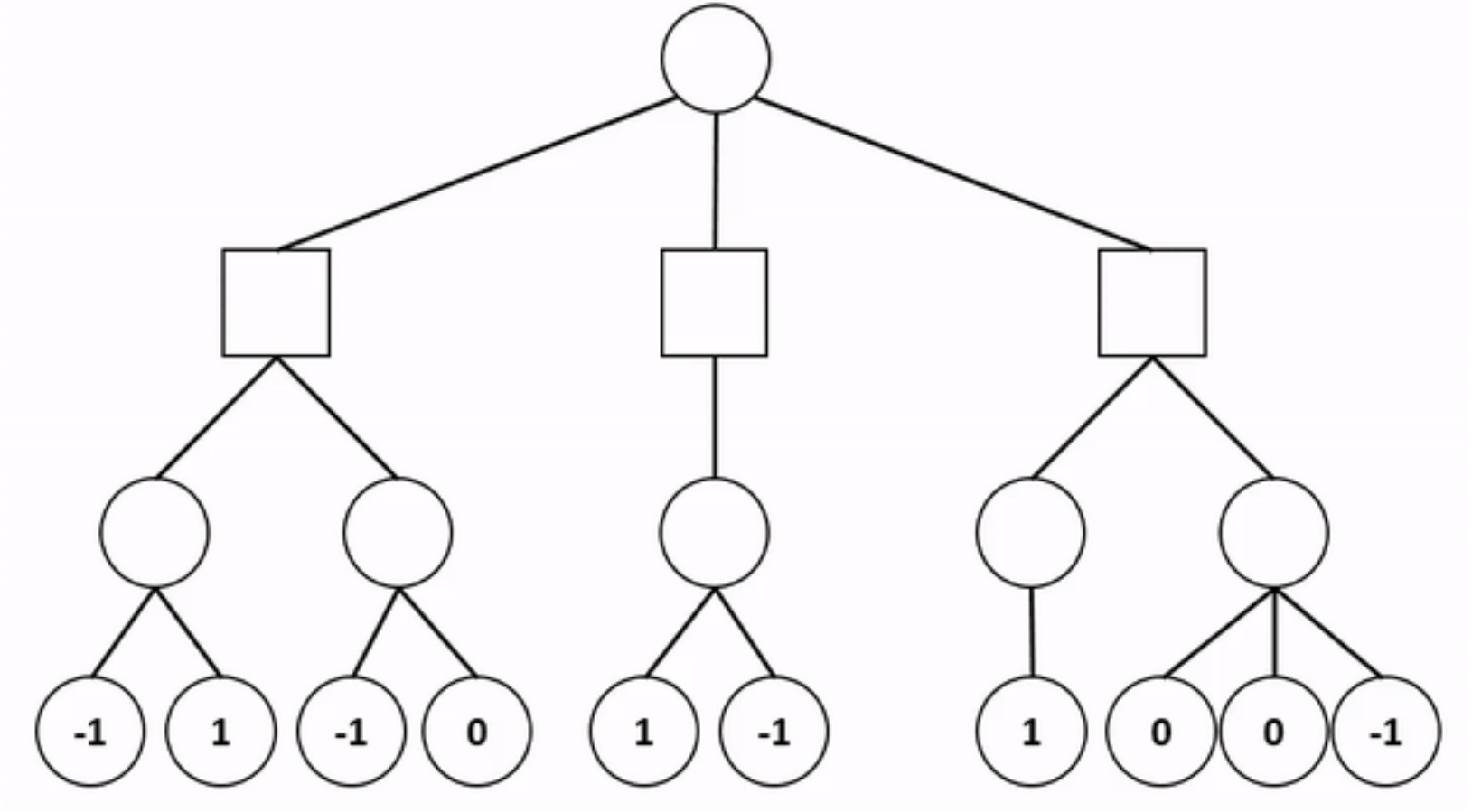


Image credit: Medium

```
function MINIMAX-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move
```

```
function MAX-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
        if v2 > v then
            v, move ← v2, a
    return v, move
```

```
function MIN-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
        if v2 < v then
            v, move ← v2, a
    return v, move
```

What if MIN does not play optimally?

- MAX will do at least as well as against an optimal player.
- However, that does not mean that it is always best to play the optimal move when facing a suboptimal opponent.
- Consider the following situation.

TIE

An optimal play by both sides will lead to a draw.

10 possible response moves by MIN that all seem reasonable, but 9 of them are a loss for MIN and one is a loss for MAX.

WIN?

MAX makes a risky move

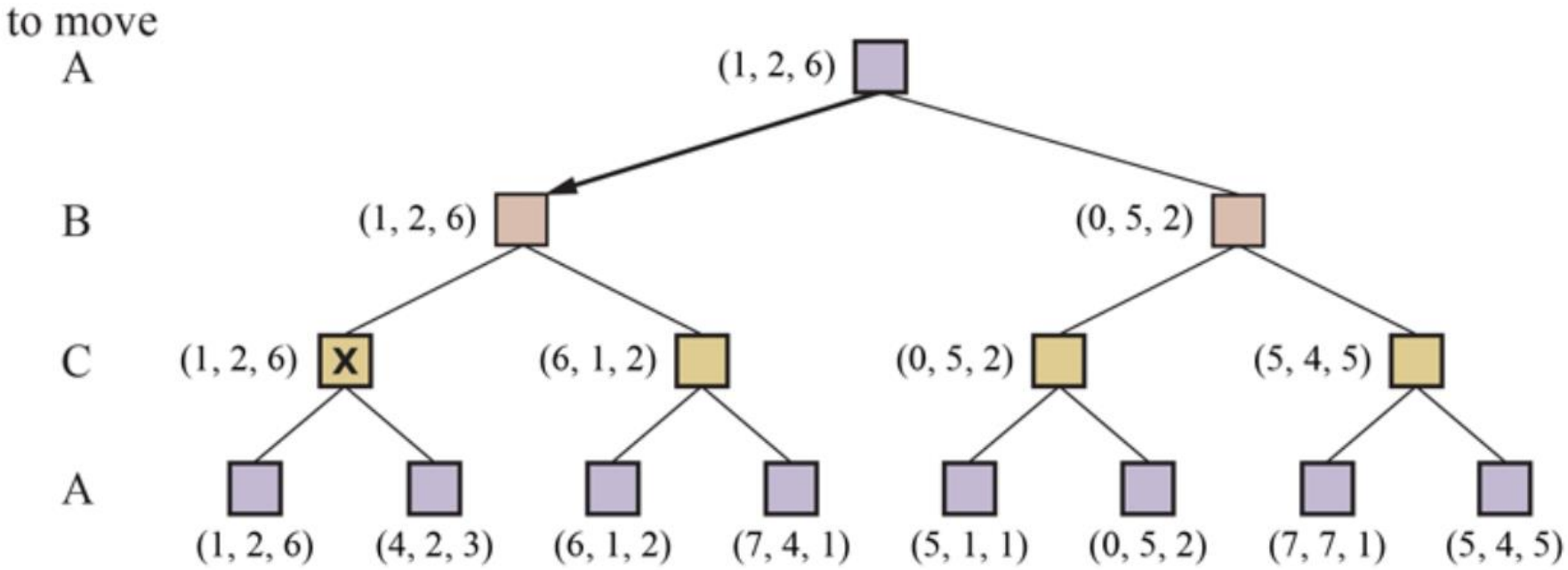
MIN cannot discover the optimal move

An evaluation of Minimax algorithm

- A complete depth-first exploration of the game tree
- Completeness: Yes (if tree is finite)
- Optimality : Yes (against an optimal opponent)
- Time complexity: $O(b^m) \rightarrow$ infeasible for practical game
  - m: the maximum depth of the tree, b: the legal moves at each point
- Space complexity: $O(bm)$ (depth-first exploration)

Optimality in multiplayer games

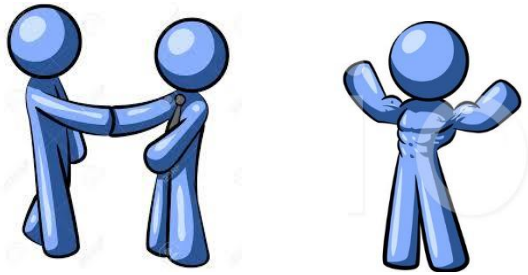
- The $UTILITY$ function is improved to return a vector of utilities.
  - For terminal states, this vector gives the utility of the state from each player's viewpoint.



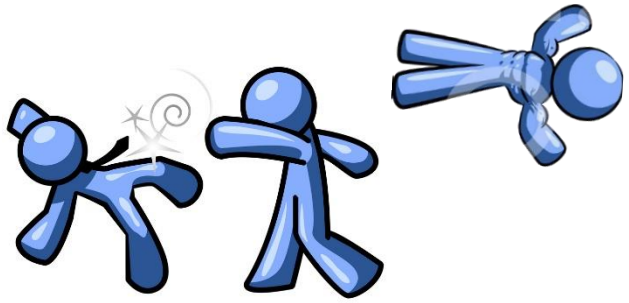
A three-ply game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

Optimality in multiplayer games

- Multiplayer games usually involve alliances, which are made and broken as the game proceeds.



A and B are weak while C is strong.
A forms an alliance with B.

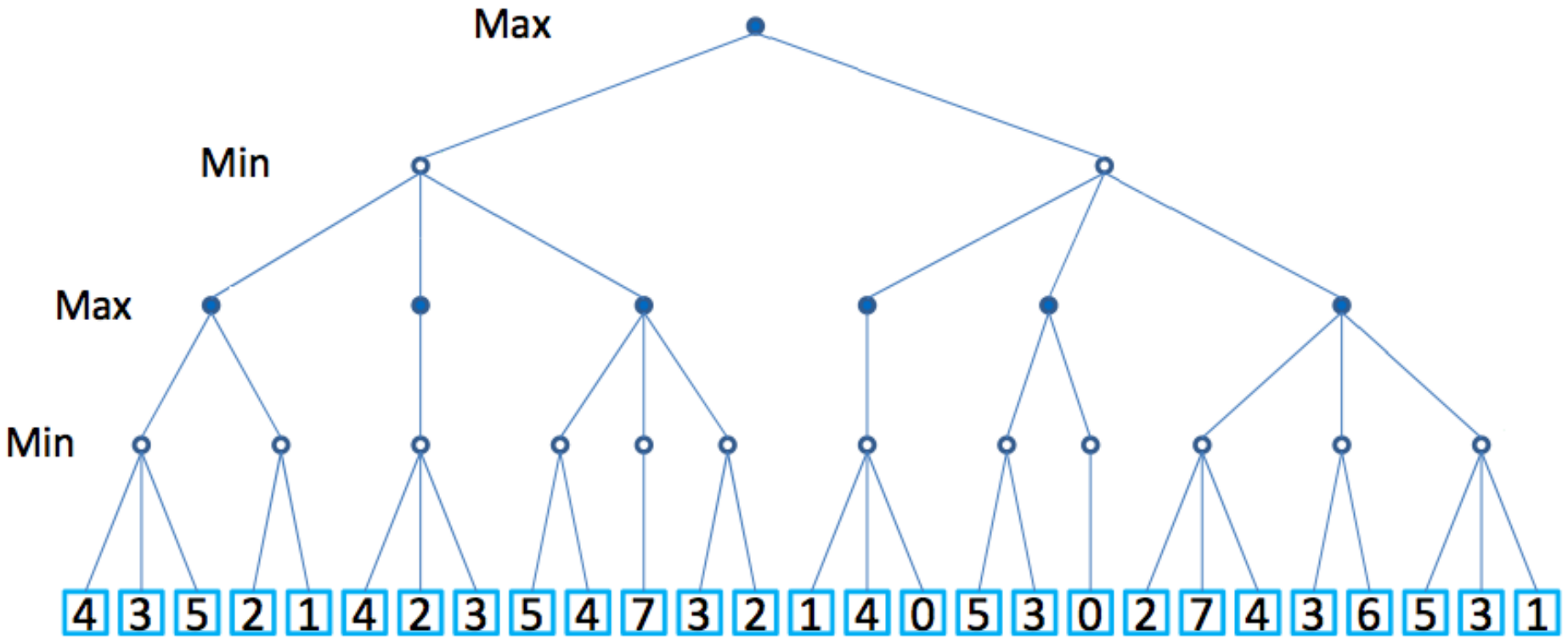


C becomes weak.
A or B could violate the agreement

- If the game is not zero-sum, collaboration can also occur with just two players.

Quiz 01: Minimax algorithm

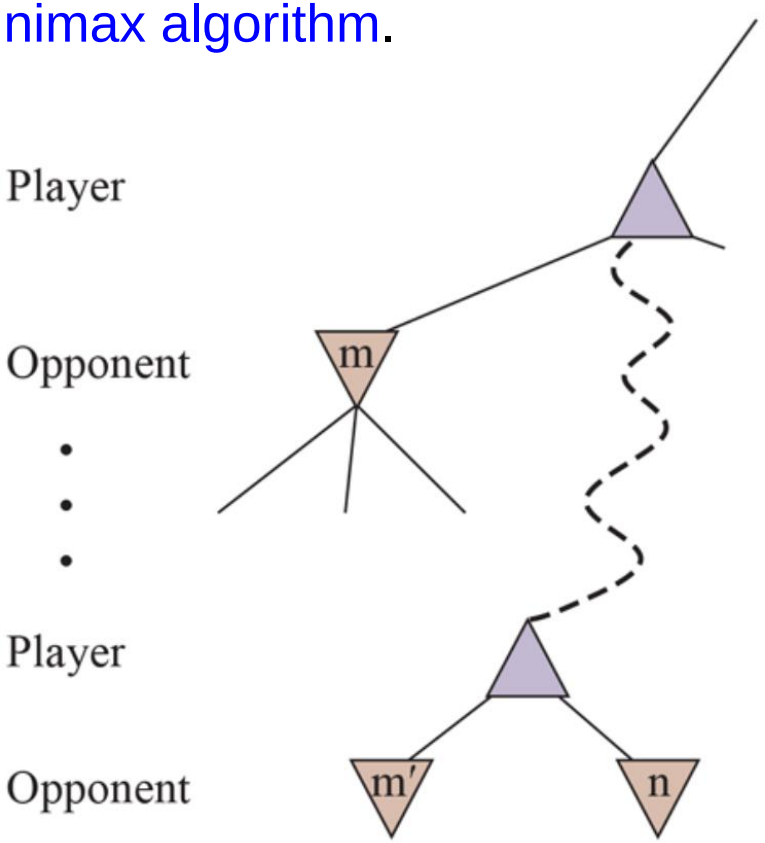
- Calculate the utility value for the remaining nodes
- Which node should MAX and MIN choose?



Alpha-beta pruning

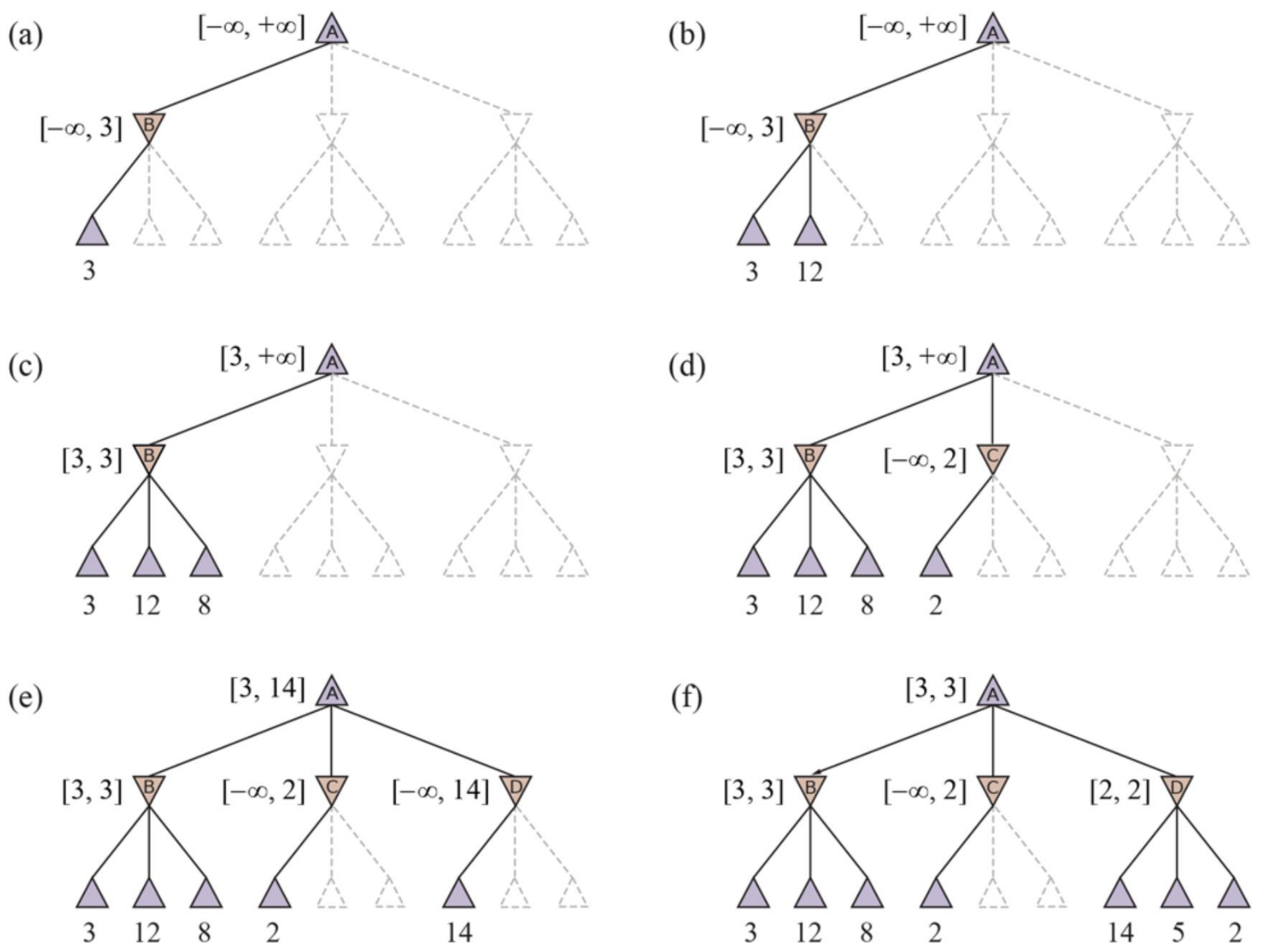
- Alpha-beta pruning aims to cut back any branches of the game tree that cannot possibly influence the final decision.
  - Its **worst case** is as good as the minimax algorithm.

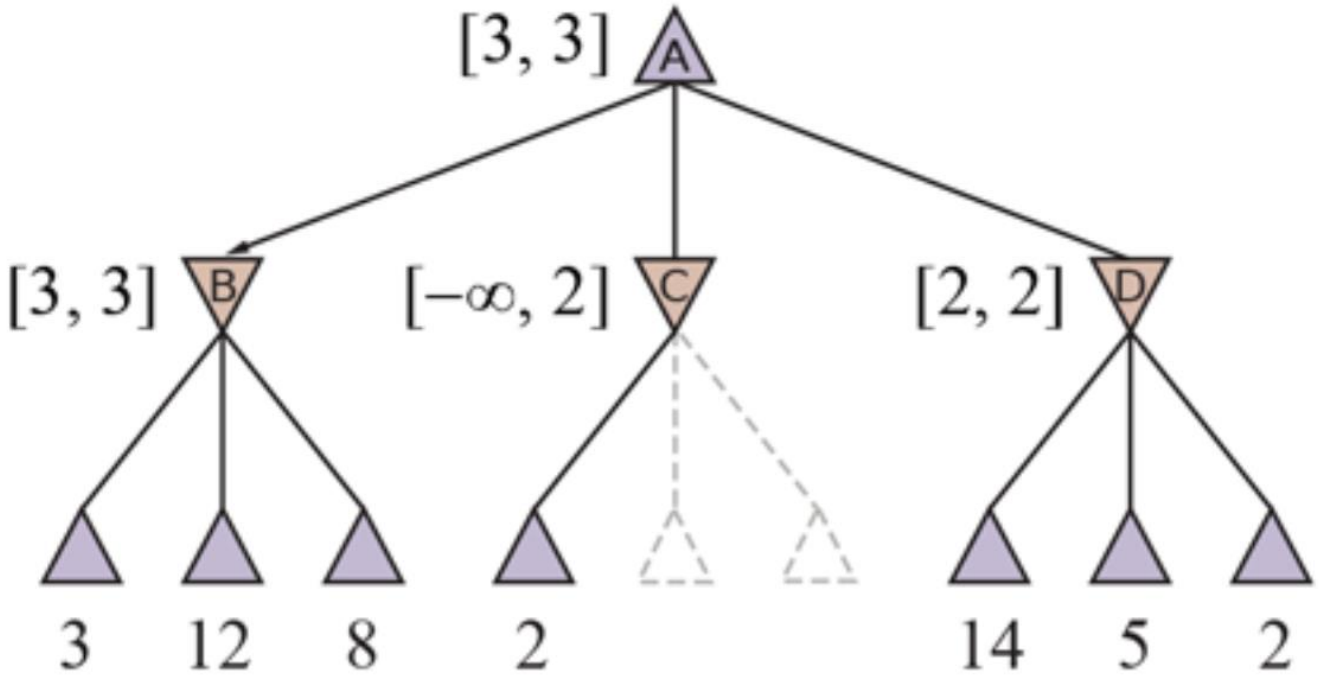
The player plans to move to a node n. If he has a better choice either at the same level, e.g., node m', or at any point higher up in the tree, e.g., node m, then he will never move n.



Alpha-beta pruning

- The two parameters, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, describes the bounds on the backed-up values that appear anywhere along the path.
  - $\boldsymbol{\alpha}$ = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
  - **β** = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.
- The algorithm updates these values as it goes along.
- Pruning at a current node happens when its value is worse than the current α or β value for MAX or MIN, respectively.


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2



- Let the two unevaluated successors of node C have values x and y.
- Then the value of the root node is given by

$$MINIMAX(root) = \max(\min(3,12,8), \min(2,x,y), \min(14,5,2))$$
$$= \max(3, \min(2,x,y), 2)$$
$$= \max(3,z,2) \qquad where\ z = \min(2,x,y) \leq 2$$
$$= 3$$

```
function ALPHA-BETA-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state, −∞, +∞)
    return move
```

```
function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
        if v2 > v then
            v, move ← v2, a
            α ← MAX(α, v)
        if v ≥ β then return v, move
    return v, move
```

```
function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
        if v2 < v then
            v, move ← v2, a
            β ← MIN(β, v)
    if v ≤ α then return v, move
        return v, move
    return v, move
```

Good move ordering

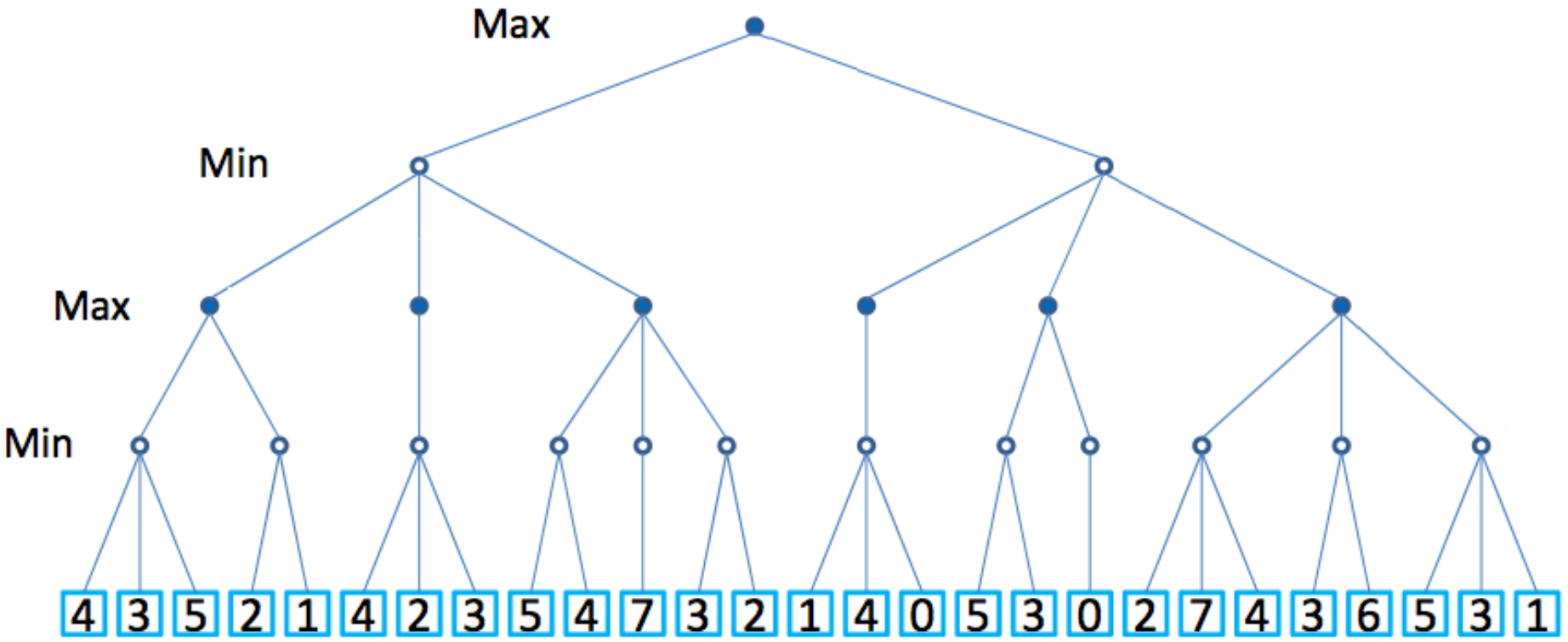
- It might be worthwhile to first examine the successors that are likely to be best.
  - E.g., the successors of node D in the previous example.
- Alpha–beta with perfect move ordering can solve a tree roughly twice as deep as minimax in the same duration.
  - Perfect move ordering: $O(b^{m/2}) \rightarrow$ effective branching factor $\sqrt{b}$
  - Random move ordering: $O(b^{3m/4})$ for moderate b
- Obviously, we cannot achieve perfect move ordering.

Good move ordering

- Dynamic move-ordering schemes bring us quite close to the theoretical limit.
  - E.g., trying first the moves that were found to be best **in the past**
- Killer move heuristic: run IDS search with 1 ply deep and record the best path, from which search 1 ply deeper.
- Transposition table avoids re-evaluation a state by caching the heuristic value of states.
  - Transpositions are different permutations of the move sequence that end up in the same position.

Quiz 02: Alpha-beta pruning

- Calculate the utility value for the remaining nodes.
- Which nodes should be pruned?



Imperfect real-time decisions

- *Evaluation functions*
- *Cutting off search*
- *Forward pruning*
- *Search versus Lookup*

Heuristic minimax

- Both minimax and alpha-beta pruning search all the way to terminal states.
  - This depth is usually **impractical** because moves must be made in a reasonable amount of time (~ minutes).
- Cut off the search earlier with some depth limit
- Use an evaluation function
  - An estimation for the desirability of position (win, lose, tie?)

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s,\text{MAX}) & \text{if IS-CUTOFF}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,\ a), d+1) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,\ a), d+1) & \text{if TO-MOVE}(s) = \text{MIN}. \end{cases}$$

Evaluation functions

- These evaluation function should order the terminal states in the same way as the true utility function does
  - States that are wins must evaluate better than draws, which in turn must be better than losses.
- The computation must not take too long!
- For nonterminal states, their orders should be strongly correlated with the actual chances of winning.

Evaluation functions

- For chess, typically linear weighted sum of features

$$\boldsymbol{Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)}$$

  - where f_i could be the numbers of each kind of piece on the board, and w_i could be the values of the pieces
  - E.g., $Eval(s) = 9q + 5r + 3b + 3n + p$
- Implicit strong assumption: the contribution of each feature is independent of the values of the other features.
  - E.g., assign the value 3 to a bishop ignores the fact that bishops are more powerful in the endgame $\rightarrow$ **Nonlinear combination**

Cutting off search

- *Minimax Cutoff* is identical to *Minimax Value* except
  1. $Is - Terminal?$ is replaced by $Is - Cutoff?$
  2. $Utility$ is replaced by $Eval$

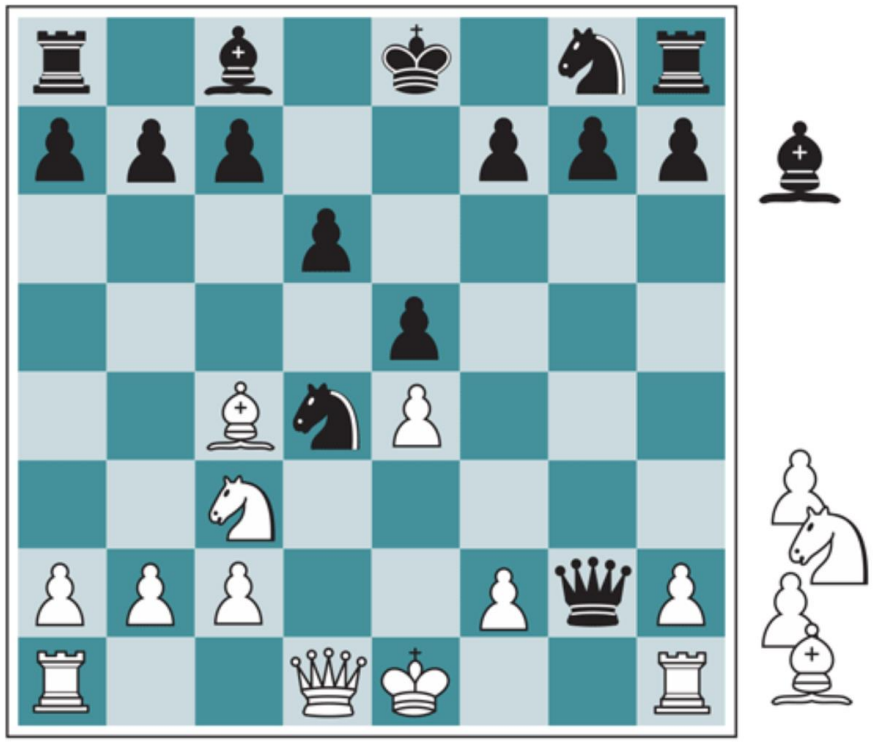
If IS-CUTOFF(state, depth) **then return** EVAL(state)

- Does it work in practice?
  - $b^m = 10^6, b = 35 \rightarrow m = 4$
  - 4-ply lookahead is a hopeless chess player!
  - 4-ply ≈ human novice, 8-ply ≈ typical PC, human master, 12-ply ≈ Deep Blue, Kasparov

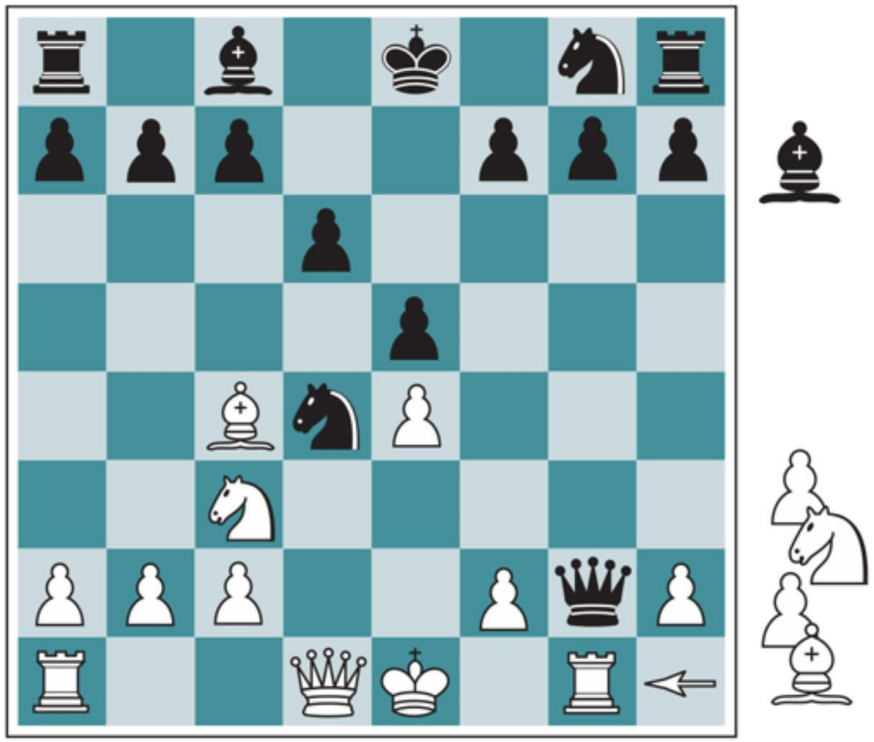
A more sophisticated cutoff test

- Quiescent positions are those unlikely to exhibit wild swings in value in the near future.
  - E.g., in chess, positions in which favorable captures can be made are not quiescent for an evaluation function counting material only
- **Quiescence search:** expand nonquiescent positions until quiescent positions are reached.

Quiescent positions: An example



(a) White to move

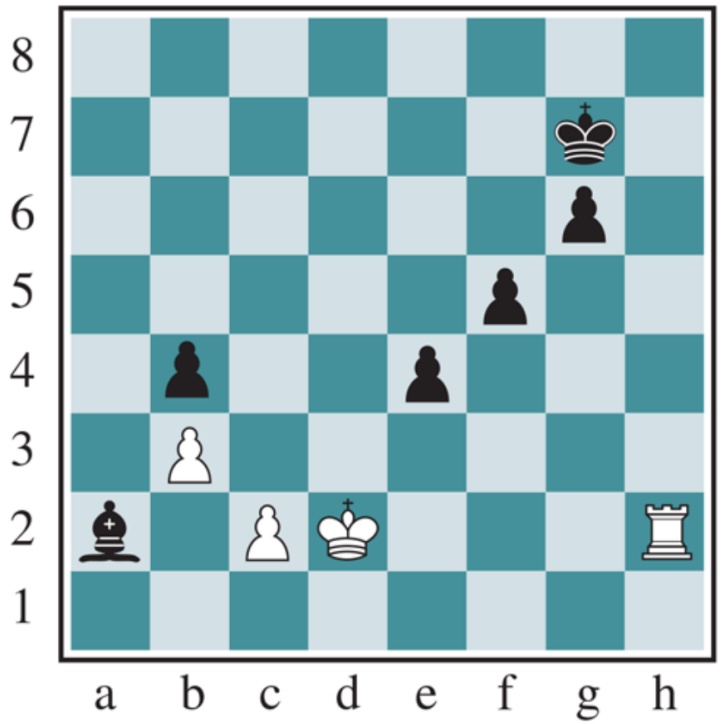


(b) White to move

Two chess positions that differ only in the position of the rook at lower right.
In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

A more sophisticated cutoff test

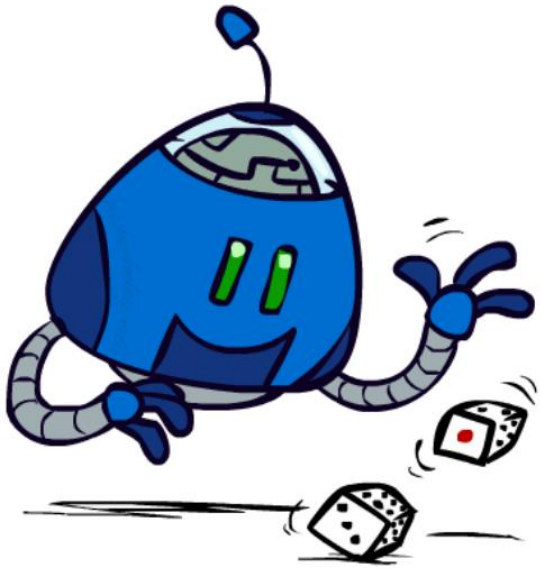
- Horizon effect: The program is facing an evitable serious loss and temporarily avoid it by delaying tactics.



With Black to move, the black bishop is surely doomed. But Black can forestall that event by checking the white king with its pawns, forcing the king to capture the pawns.

A more sophisticated cutoff test

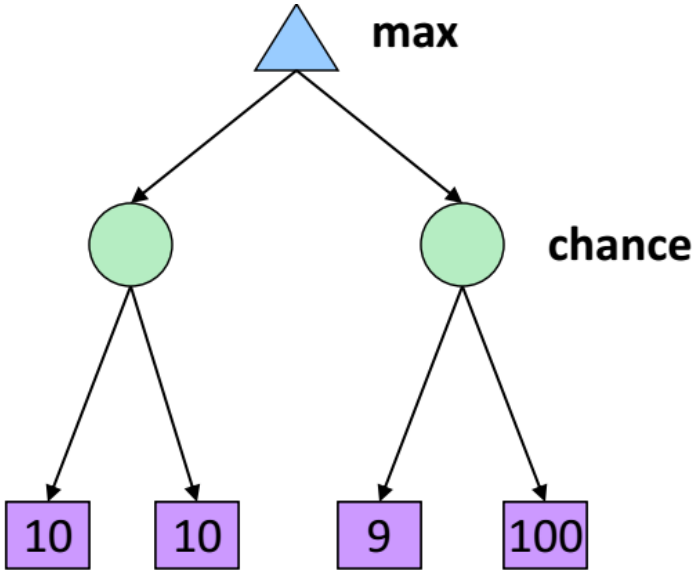
- Singular extension: a move that is “clearly better” than all other moves in a given position.
  - The algorithm allows for further consideration on a legal singular extension $\rightarrow$ deeper search tree, yet only a few singular extensions.
- Beam search
  - Forward pruning, consider only a “beam” of the n best moves only
  - Most humans consider only a few moves from each position
  - PROBCUT, or probabilistic cut, algorithm (Buro, 1995)
- Search vs. Lookup
  - Use table lookup rather than search for the opening and ending

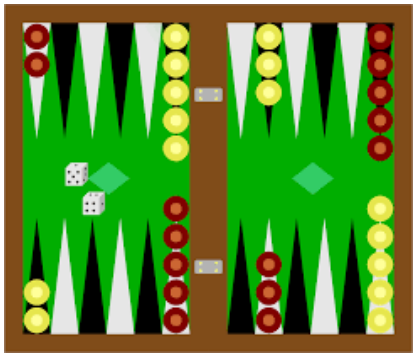


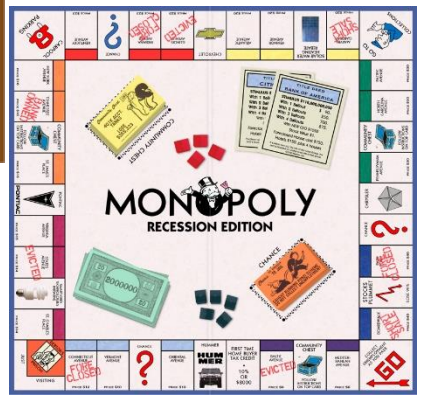
Stochastic games

Stochastic behaviors

- Uncertain outcomes controlled by chance, not an adversary!
- *Why wouldn't we know what the result of an action will be?*
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when a robot is moving, wheels might slip

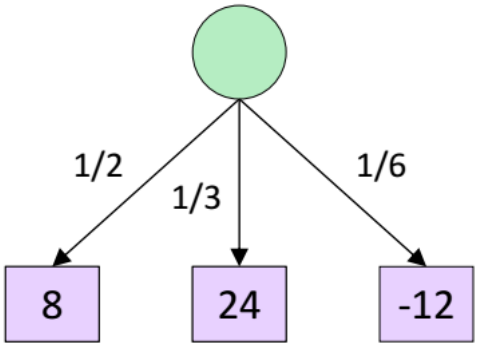






Expectimax search

- Values reflect the average-case (expectimax) outcomes, not worst-case (minimax) outcomes



v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

- **Expectimax search:** compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes, but the outcome is uncertain
  - Calculate expected utilities, i.e., take weighted average of children
- For minimax, terminal function scale doesn't matter
  - Monotonic transformations: better states to have higher evaluations
- For expectimax, we need magnitudes to be meaningful

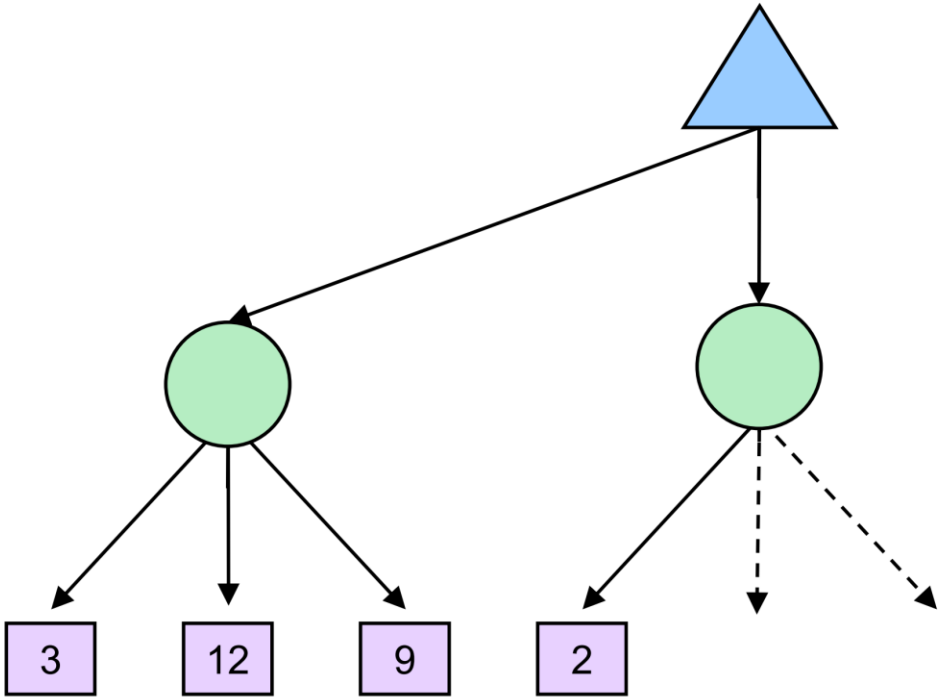
Expectimax search: Pseudo code

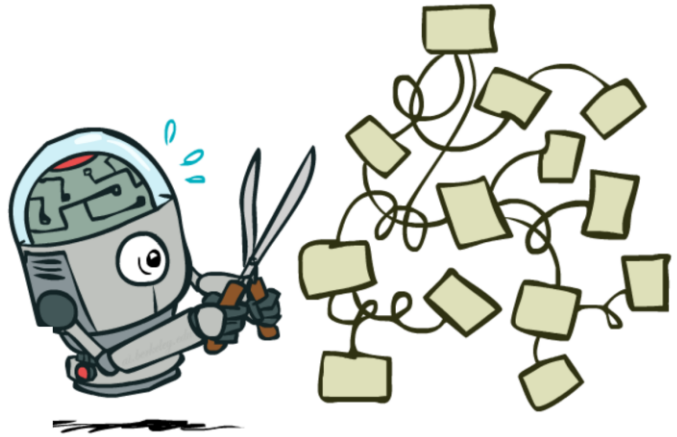
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

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

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

Expectimax pruning

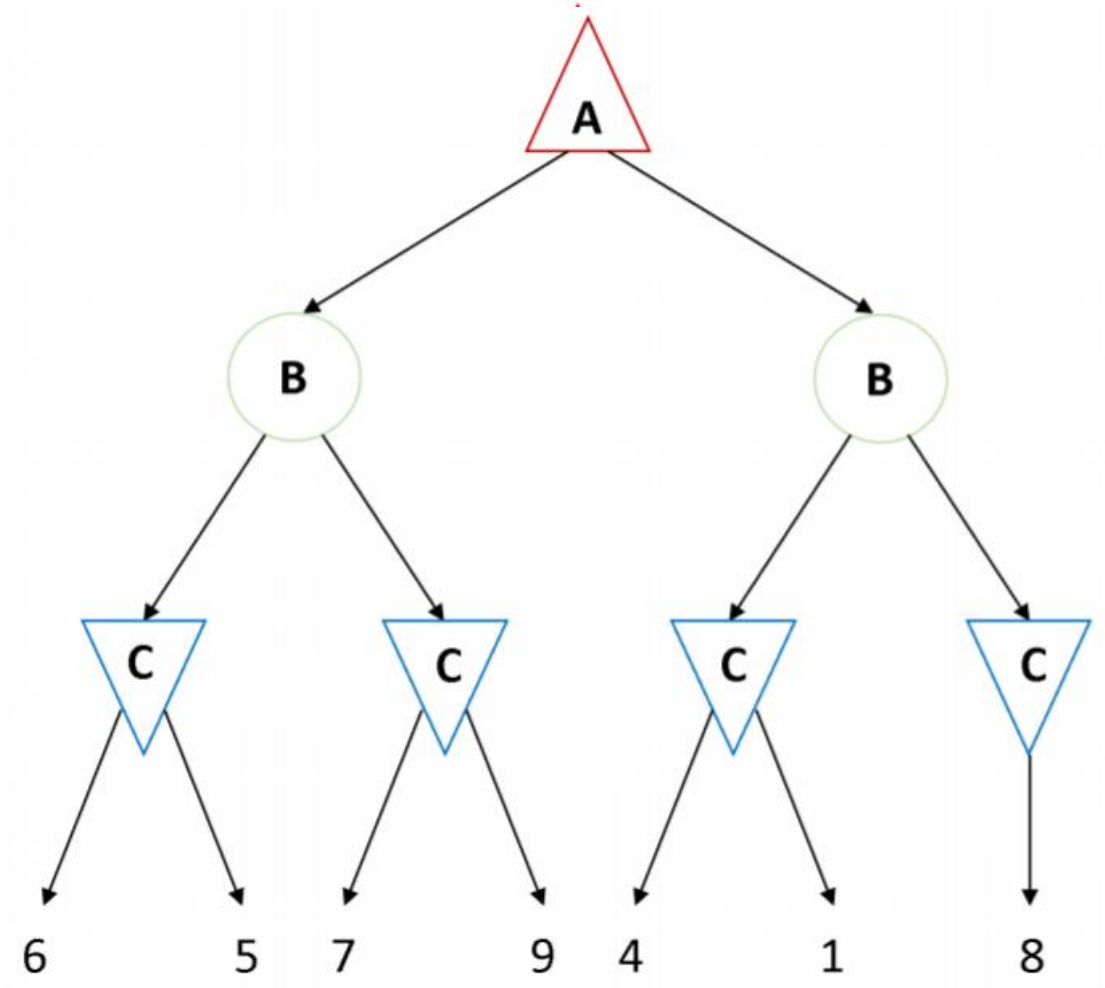




Is it possible to perform pruning in expectimax search?

Expectimax pruning

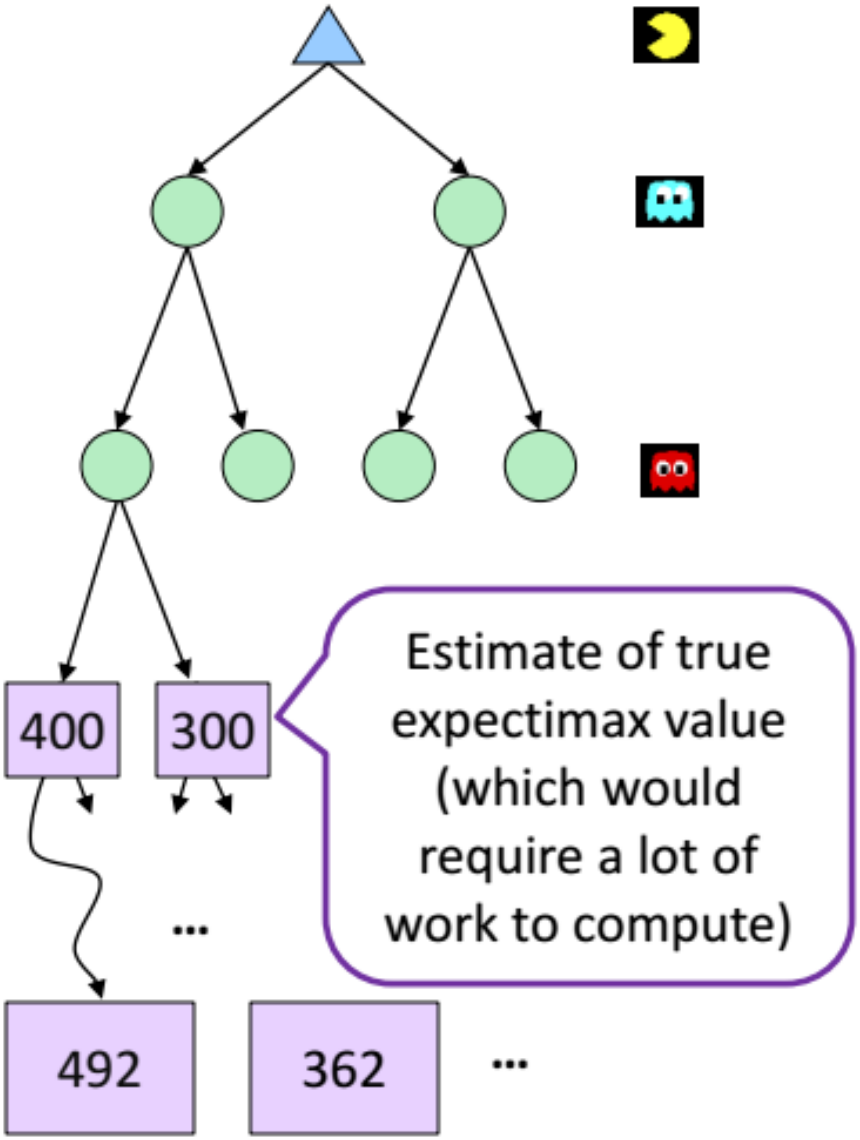
- Pruning can only be possible with knowledge of a fix range.



How to prune this tree?

- Each child has an equal probability of being chosen
- The values can only be in the range 0-9 (inclusive).

Depth-limited expectimax




...the end.