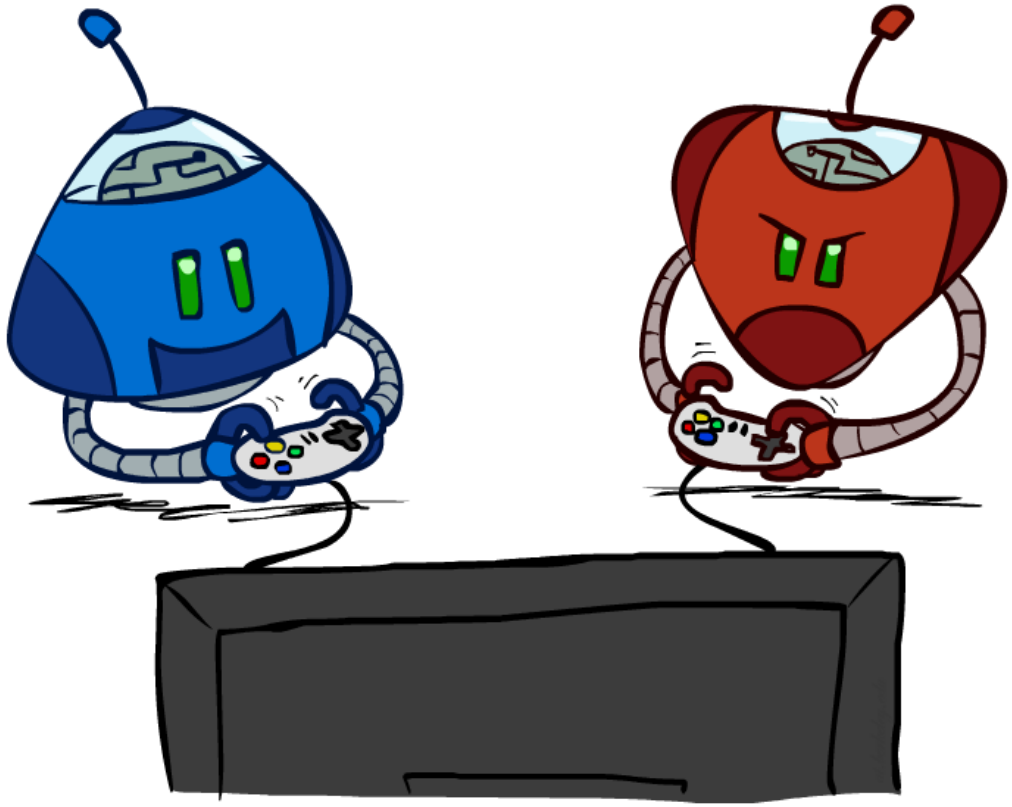
Artificial Intelligence

Adversarial Search



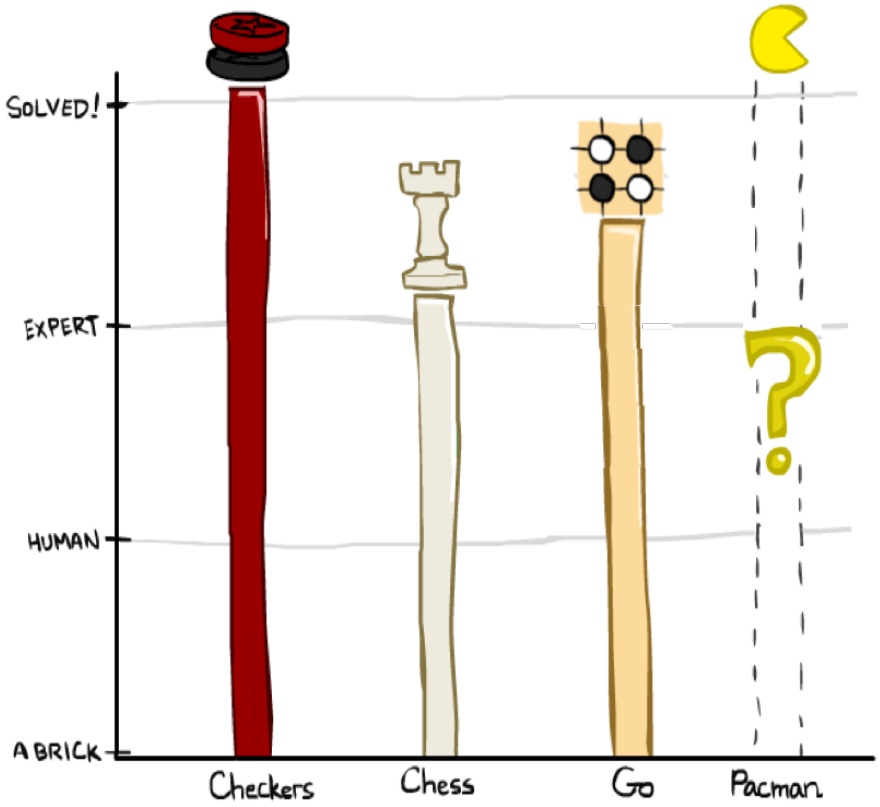
Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

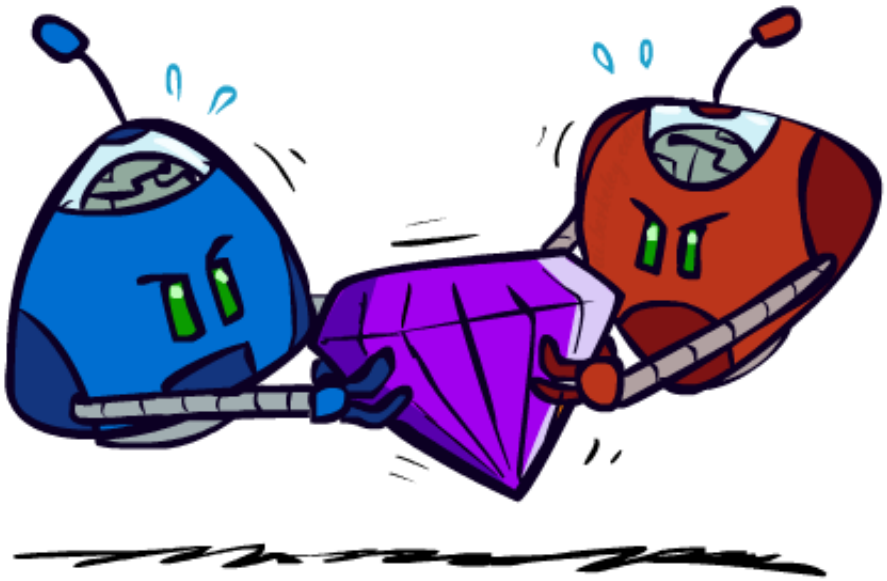
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go:** 2016: AlphaGo defeats 18-time human world champion Lee Sedol in a five-game match. AlphaGo won all but the fourth game. All games were won by resignation. The match has been compared with the historic chess match between Deep Blue and Garry Kasparov in 1997. In go, b > 300!
  Later, AlphaZero was developed that learned from many millions of simulated games, instead of using pattern knowledge bases from games played by humans. Nobody can beat AlphaZero ever since. When they tested AlphaZero against AlphaGo, the result was 100:0.

- **Pacman**

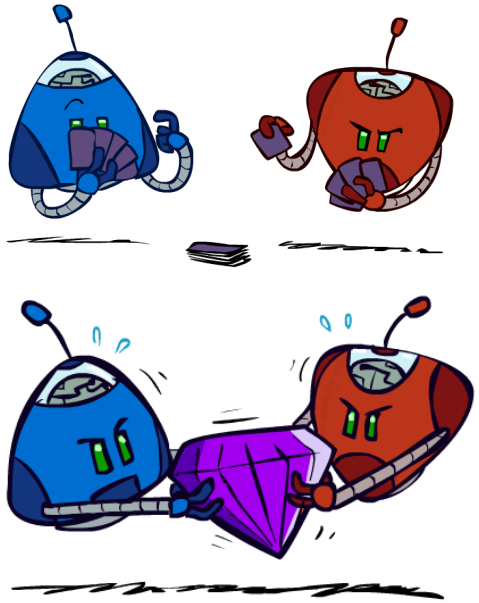


Adversarial Games



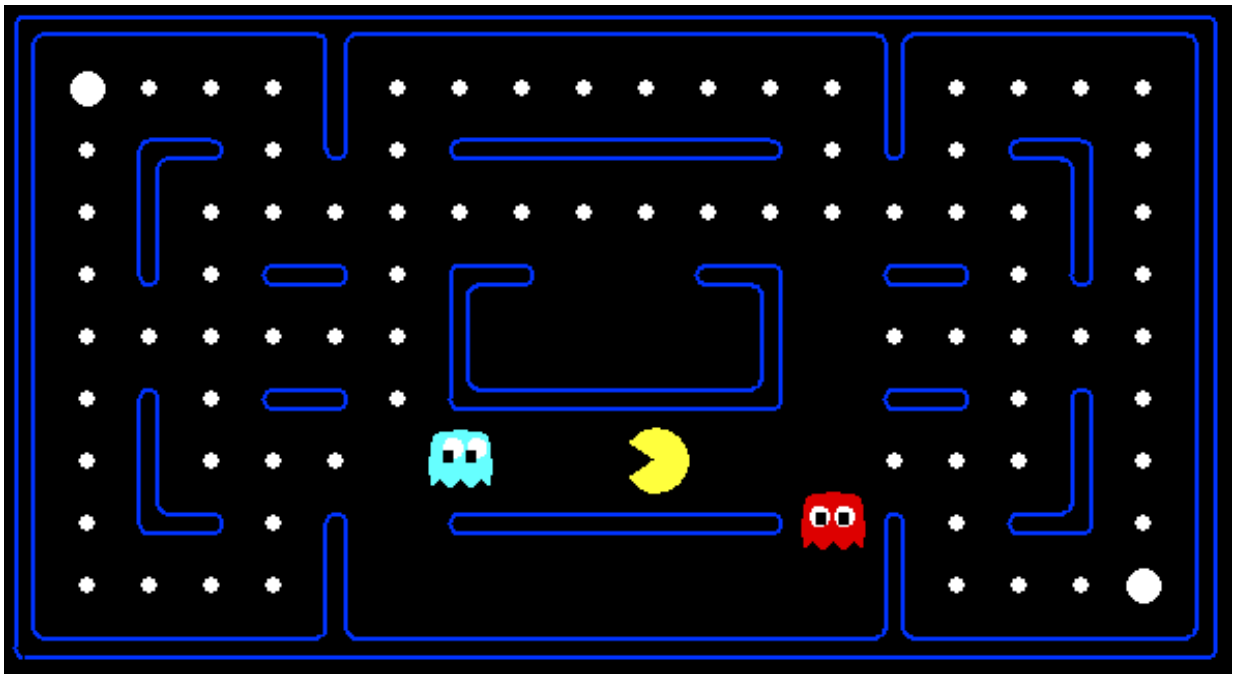
Types of Games

- Many different kinds of games!
- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Collaboration vs. competitiveness
  - Adversarial: where goals are adversarial
    - each player makes his best move, which for the opponent is the least favorable
  - Zero sum?
  - Perfect information (can you see the state)?



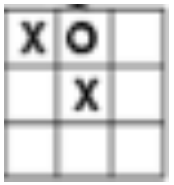
- We would like to have algorithms for calculating a strategy (policy) which recommends a move from each state

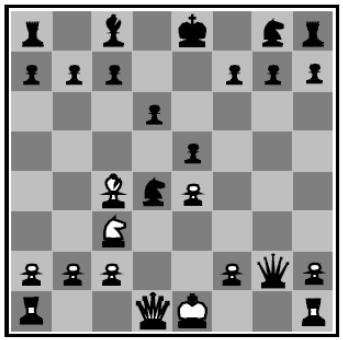
An example of a multi-agent, adversarial, stochastic game



Deterministic Games

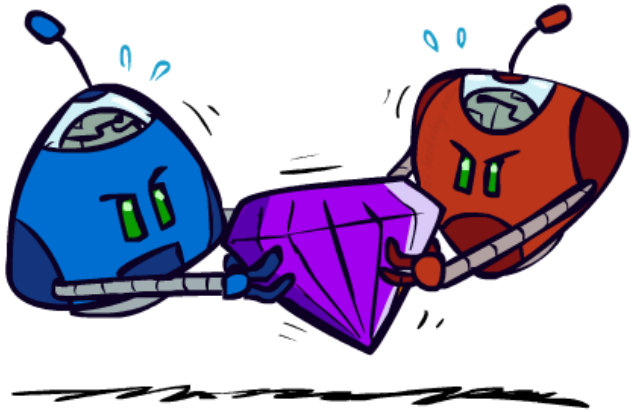
- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1, ..., N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: $S \times A \rightarrow S$
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $S \times P \rightarrow R$

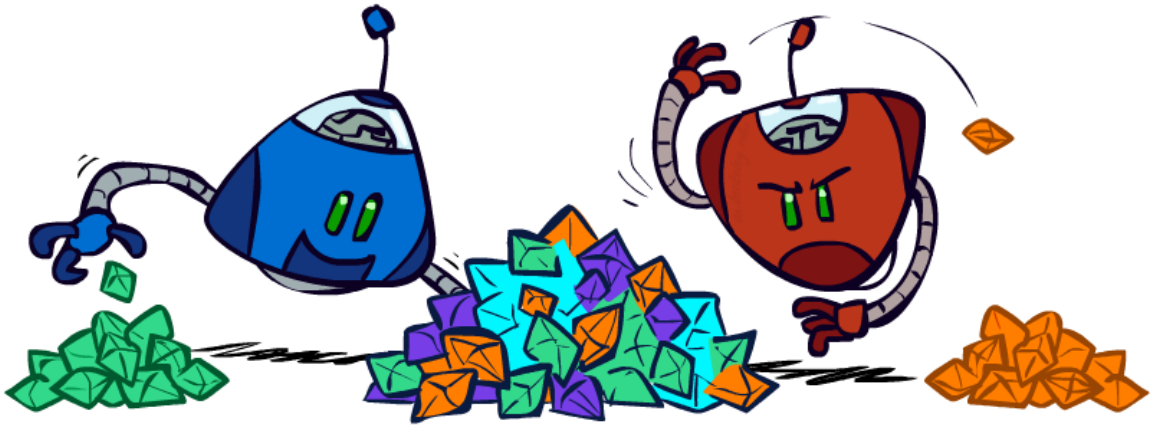




- Solution for a player is a policy: $S \rightarrow A$

Zero-Sum Games

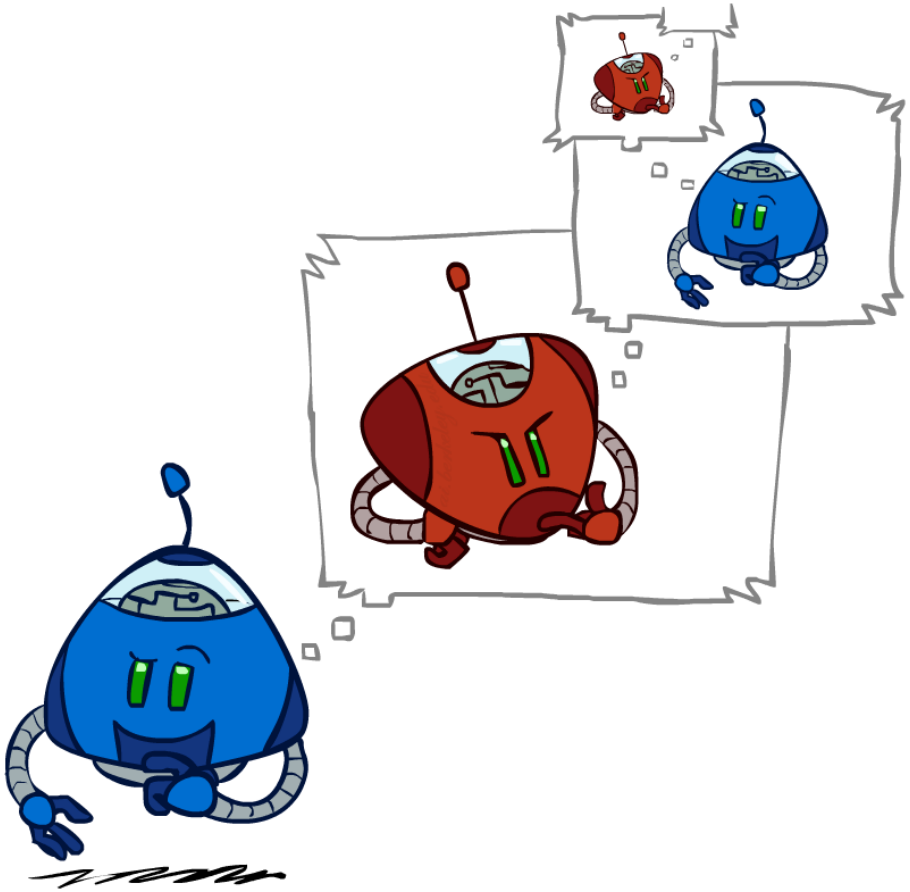




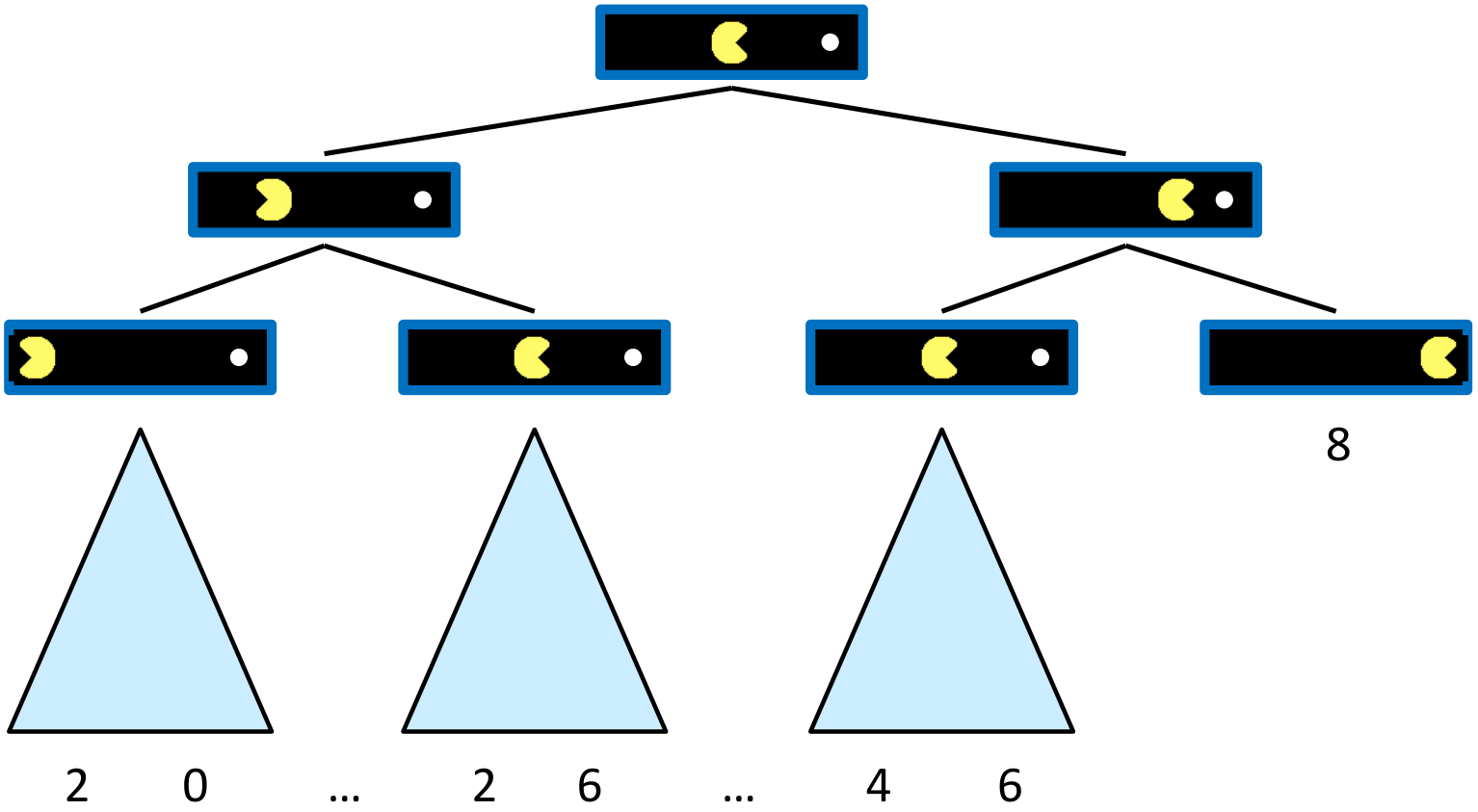
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
  - As much as one wins, the opponent looses exactly the same utility value

- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

Adversarial Search

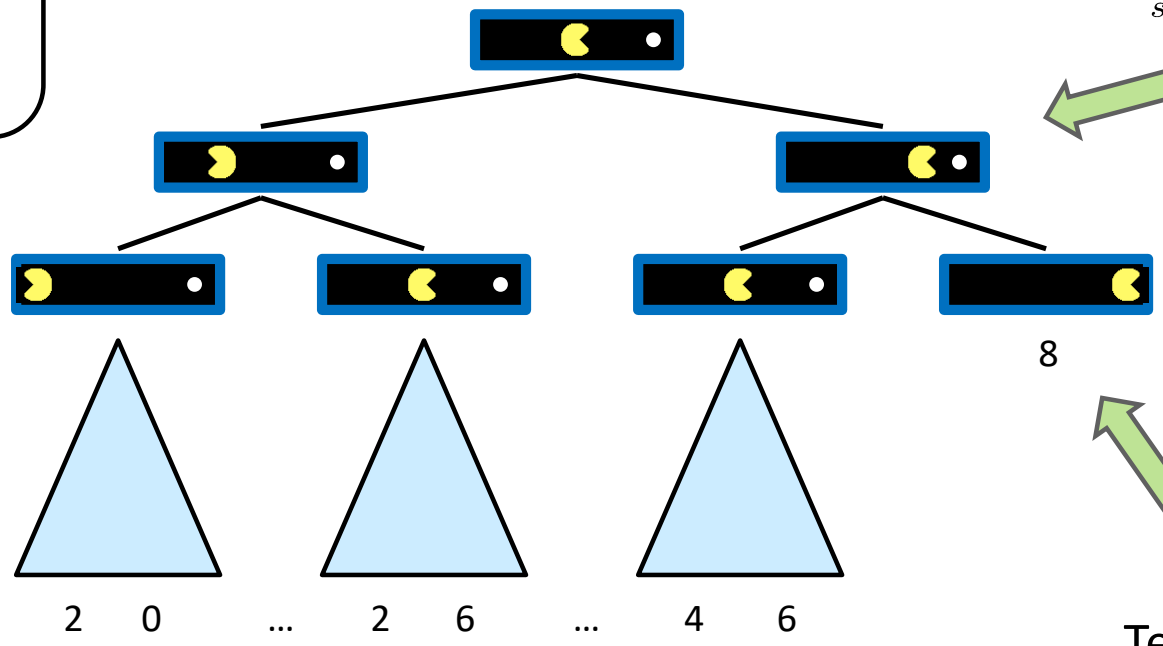


Single-Agent Trees



Value of a State

Value of a state:
The best achievable outcome (utility) from that state



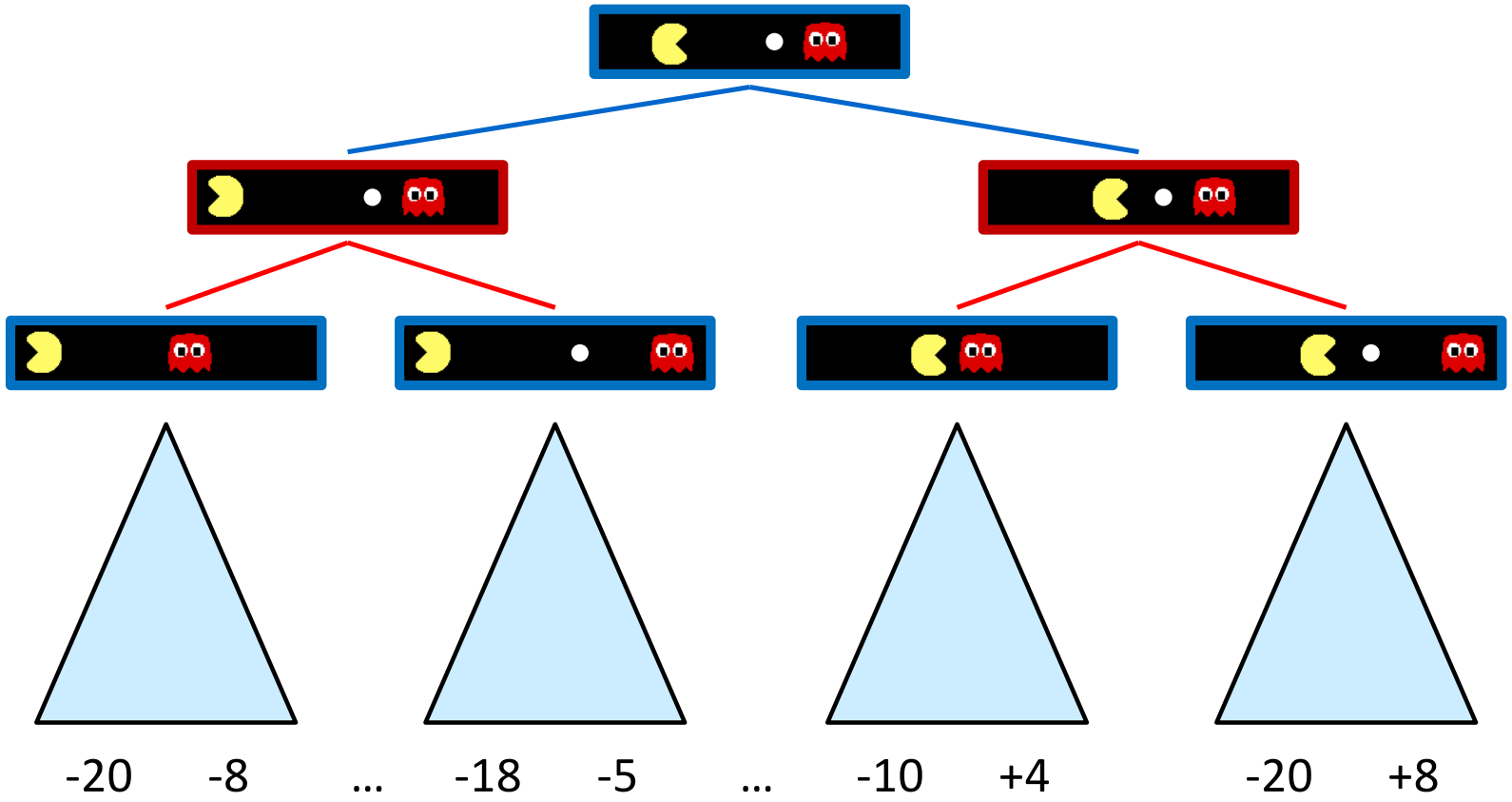
Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

Terminal States:

$$V(s) = \text{known}$$

Adversarial Game Trees



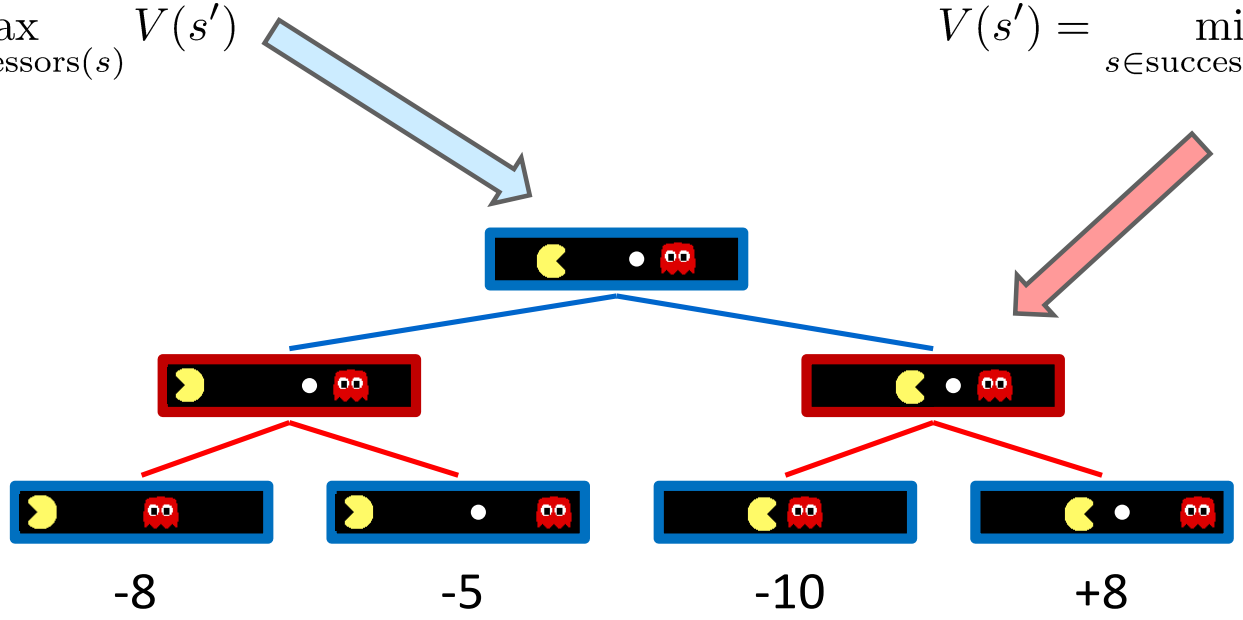
Minimax Values

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

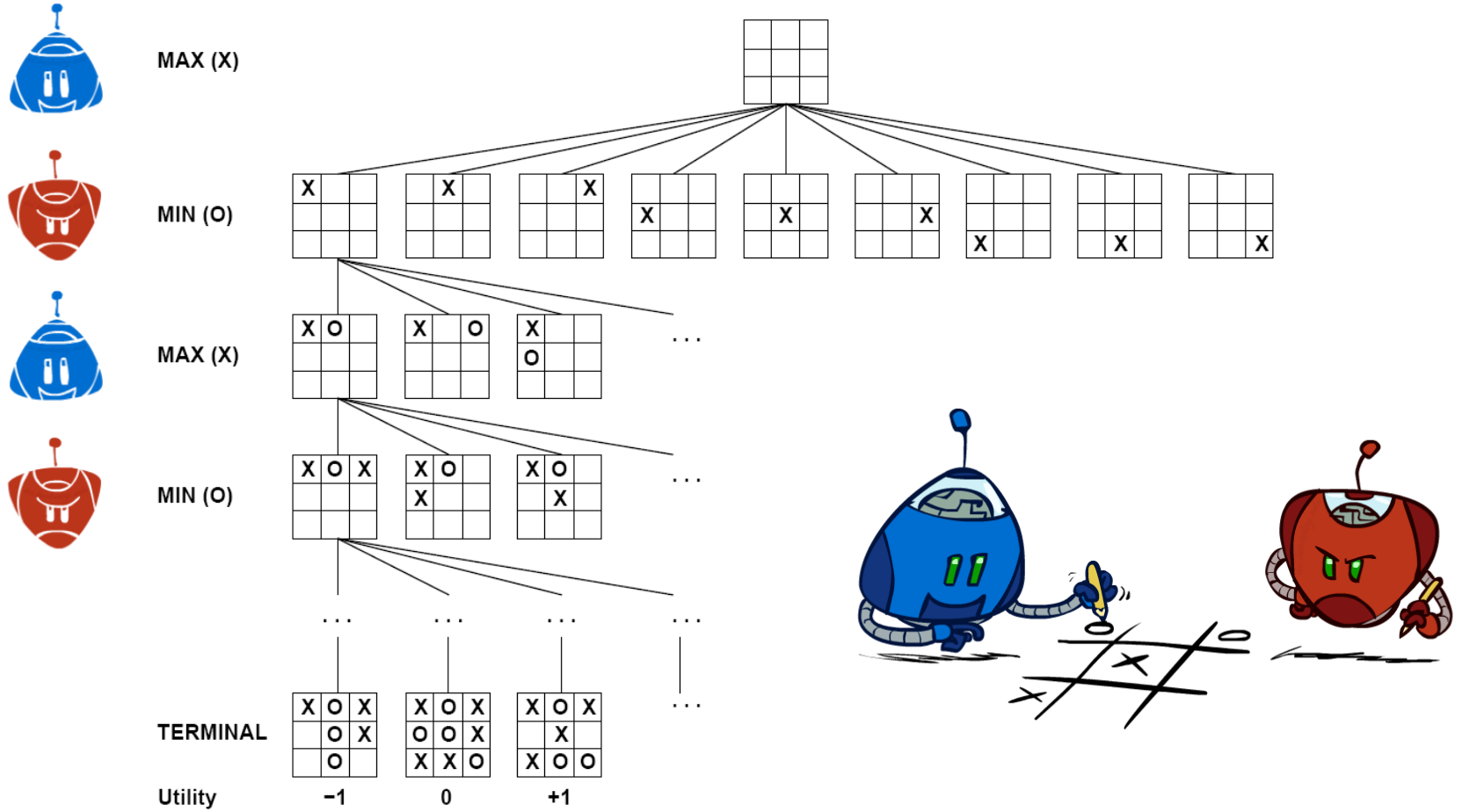
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

$$V(s) = \text{known}$$

Tic-Tac-Toe Game Tree

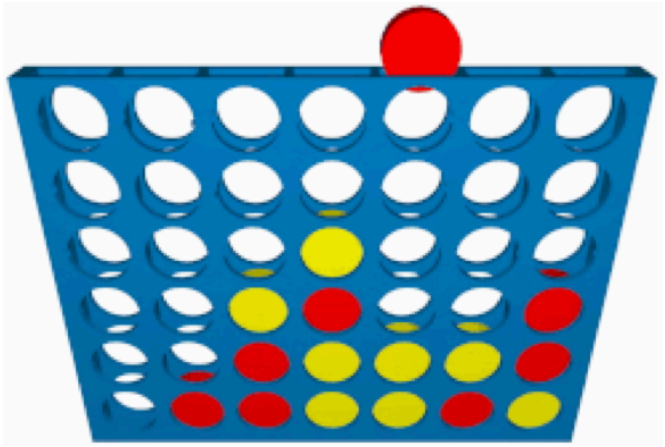


Evaluation criteria – Utility Function

- X-0:
  - Do we prevent opponent from placing three equal values horizontally, vertically or diagonally
  - Is it possible to place three equal own values horizontally, vertically or diagonally
  - What's the mobility (own, opponent's)
- Reversi:
  - Capturing angle
  - How many opponent disks will change color
  - What's the mobility (own, opponent's)
  - Will the opponent be blocked (unable to make a move)
  - ...

Utility Function for the game “4 wins”

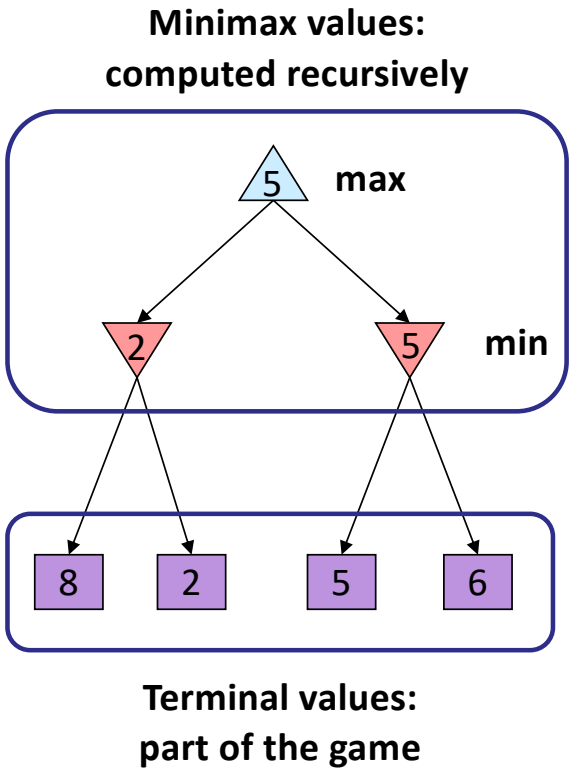
- Sequence of 2 disks with the same color:
  - Horizontally or vertically: 10 / -10 points
  - Diagonally: 20 / -20 points
- Sequence of 3 disks with the same color:
  - Horizontally or vertically: 100 / -100 points
  - Diagonally: 200 / -200 points
- Sequence of 4 own disks: ∞
- Sequence of 4 opponent disks: -∞



1. Function is based **only on own** disks
2. Function takes into account opponent disks also

Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary



What does this mean for the opponent?

- Everything that is max for the first player is min for the second player and vice versa.
- If the game has multiple players, then opponent moves are minimized in two or more partial moves

Minimax algorithm

- For two player games
- Assumes that opponent plays diligently and as good as we do

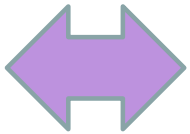
For $s \in$ Next_moves(n)

Minimax(n)=

$$\begin{cases} \text{Utility(n), if n is terminal node (state)} \\ \text{Max Minimax(s), if n is MAX node} \\ \text{Min Minimax(s), if n is MIN node} \end{cases}$$

Minimax Implementation

```
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
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

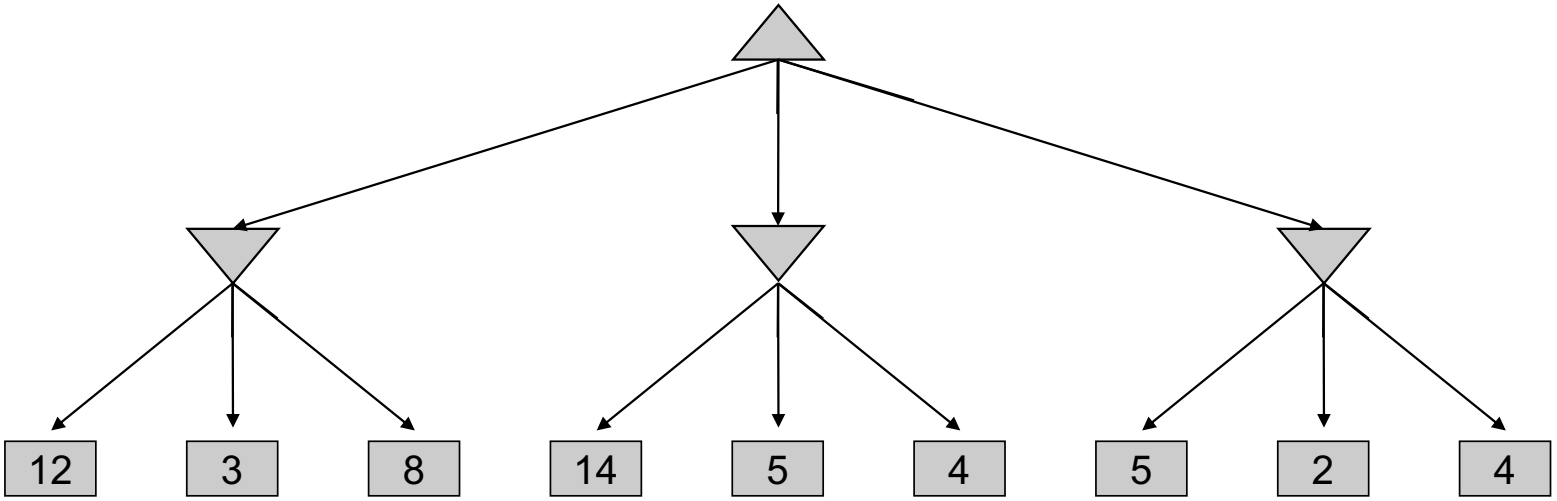
Minimax Implementation (Dispatch)

```
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 MIN: return min-value(state)
```

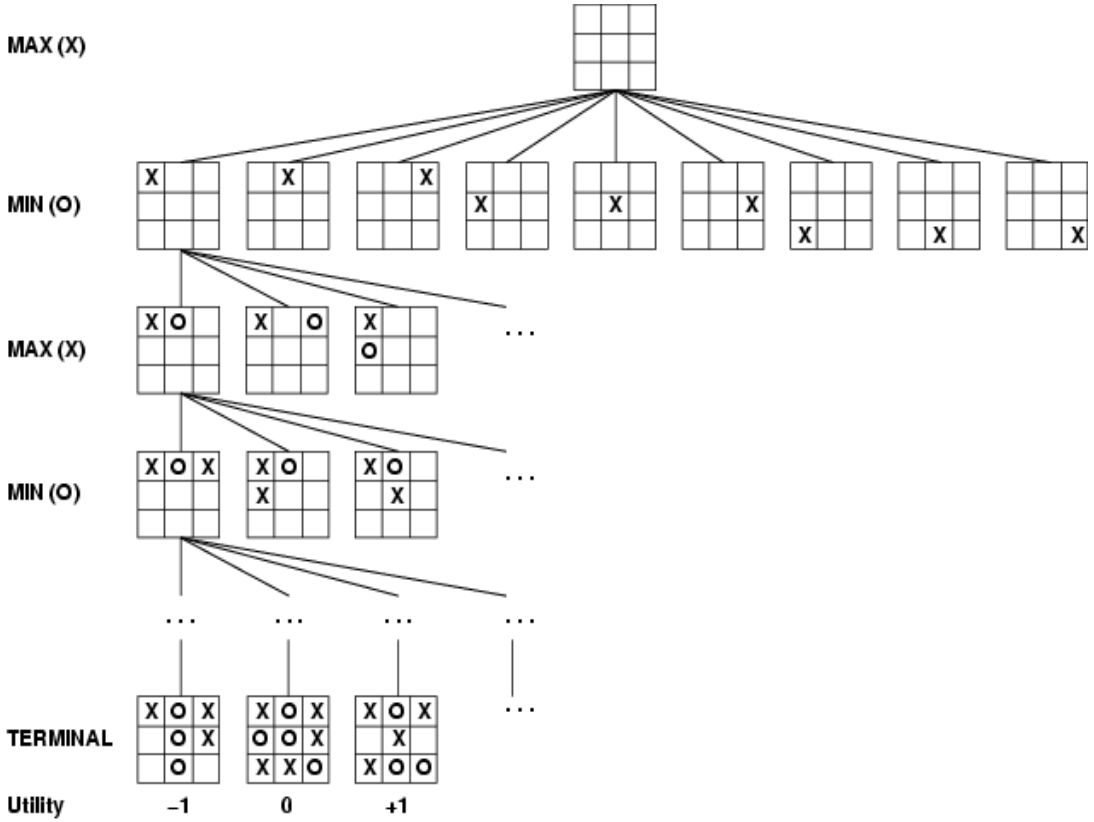
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

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

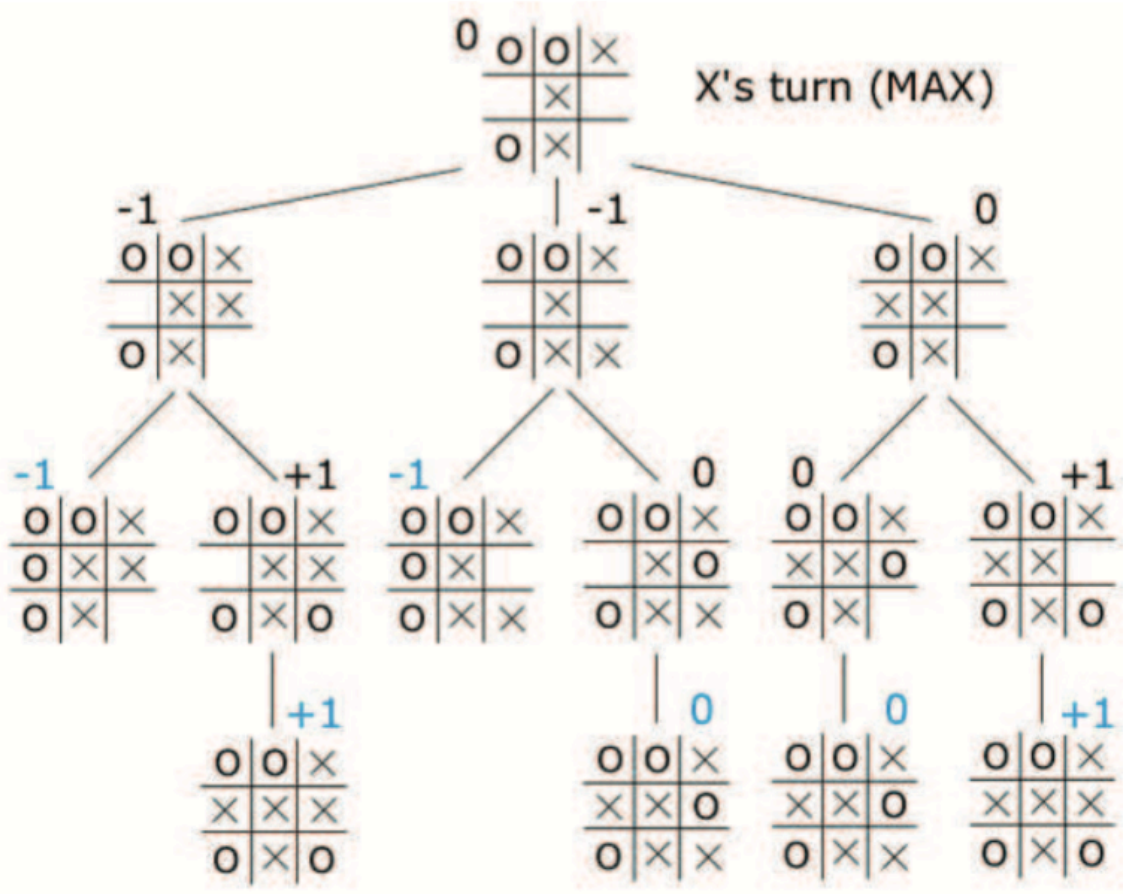
Minimax Example



How this applies to X-0

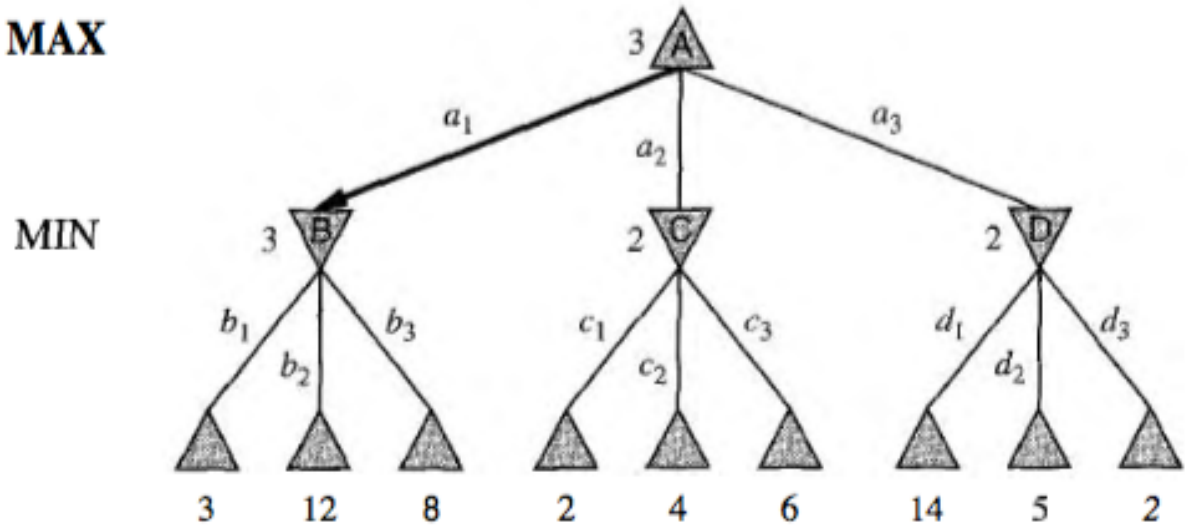


Another Min-max example for tic-tac-toe



2-ply game

*This definition of **optimal play for MAX** assumes that MIN also plays optimally it maximizes the worst-case outcome for MAX*



An example for a 3-player game

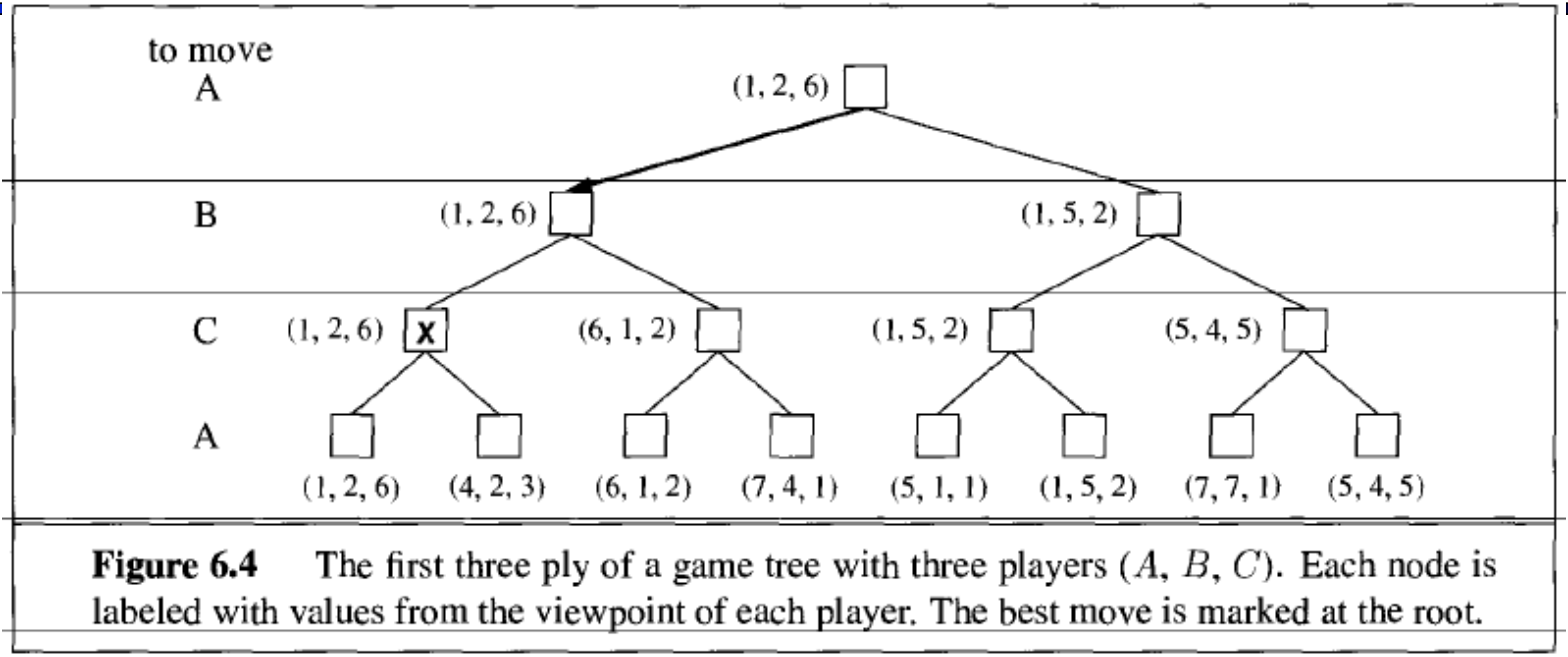


Figure 6.4 The first three ply of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

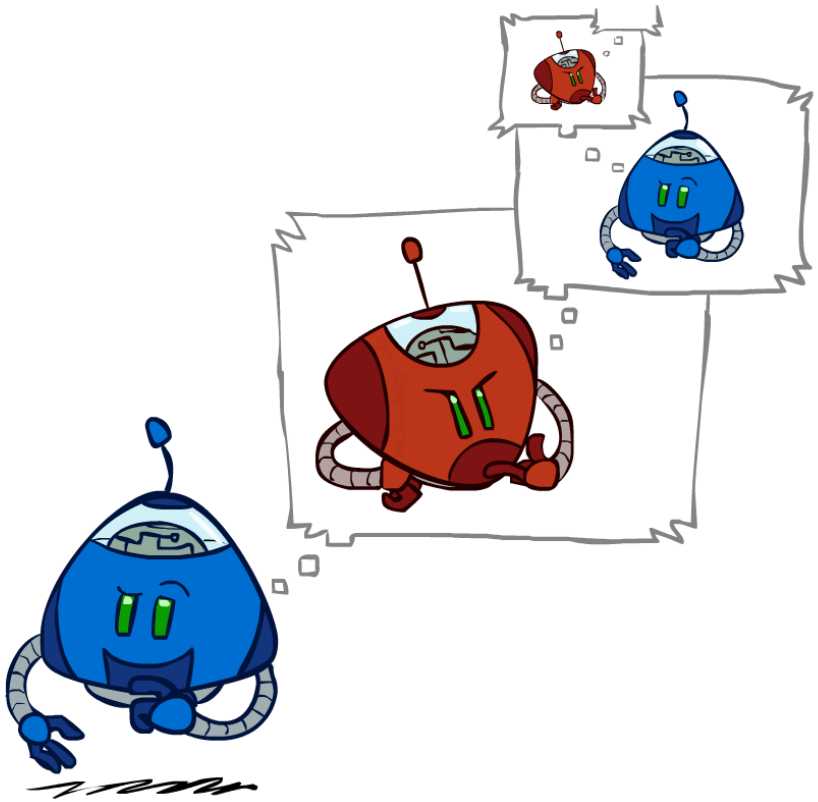
- The utility function returns an array
- Every player chooses his own maximum
- How do we model alliances?

Note: Three player game, Russell and Norvig, chapter 6, p. 6

Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: O(bm)

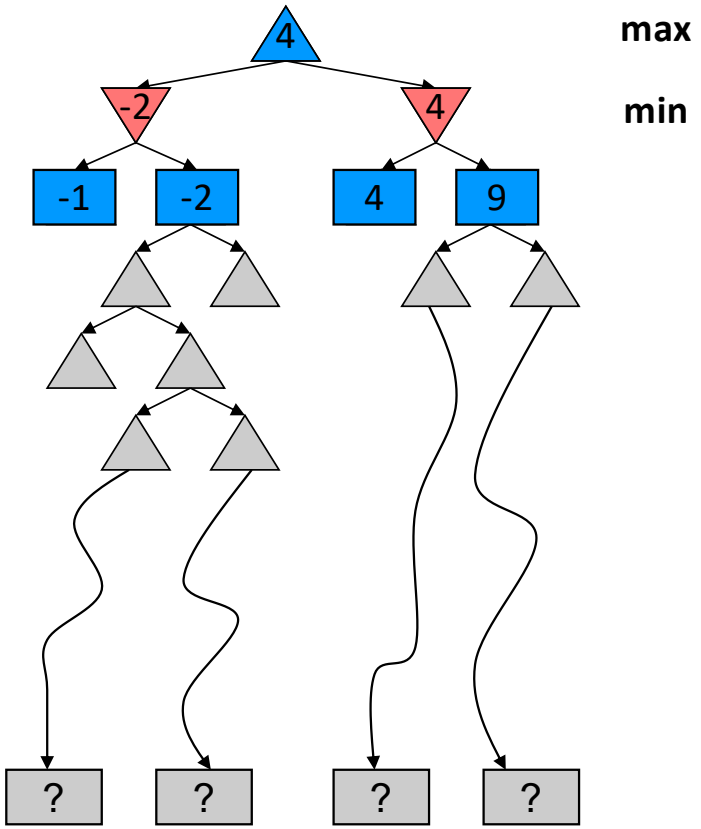
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Resource Limits

Resource Limits

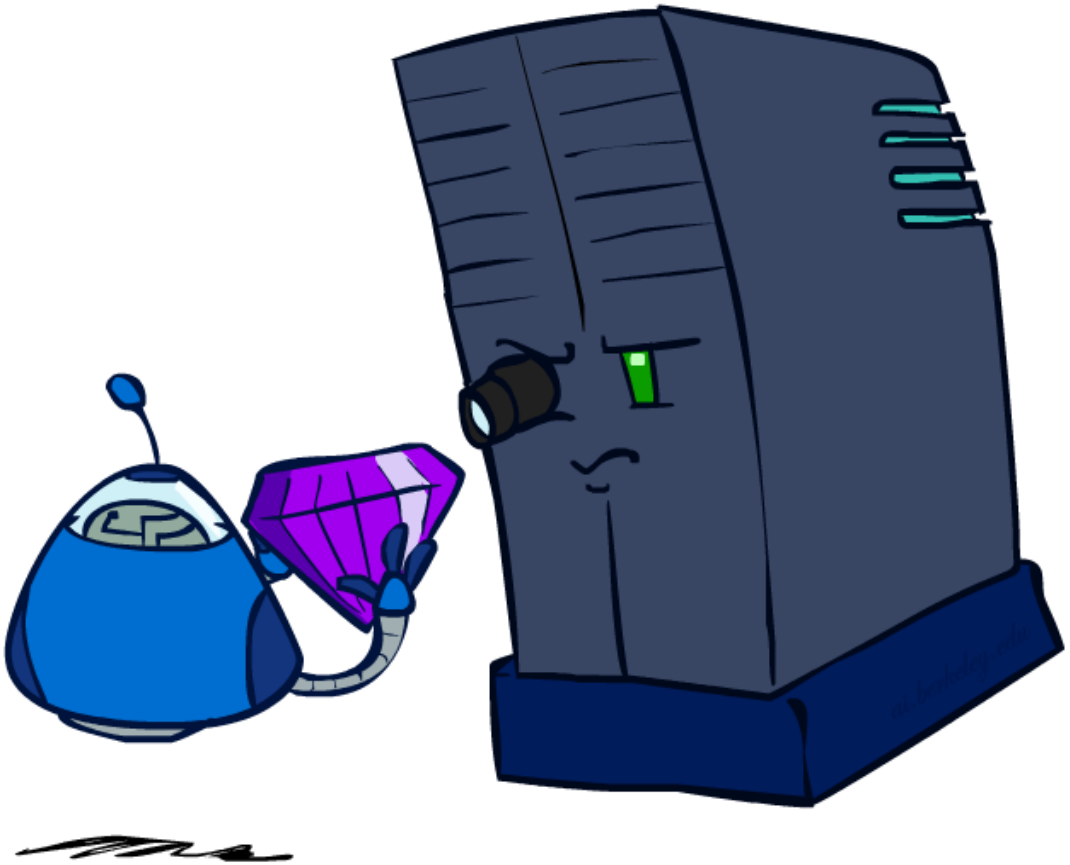
- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



Depth Matters

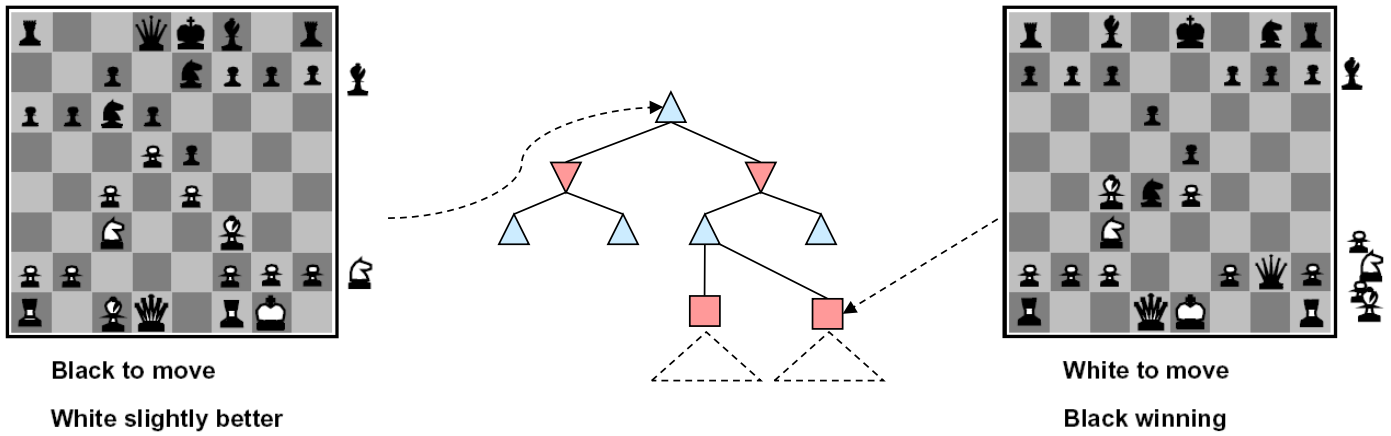
- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation

Evaluation Functions



Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search



- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

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

- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Dynamic function for move evaluation

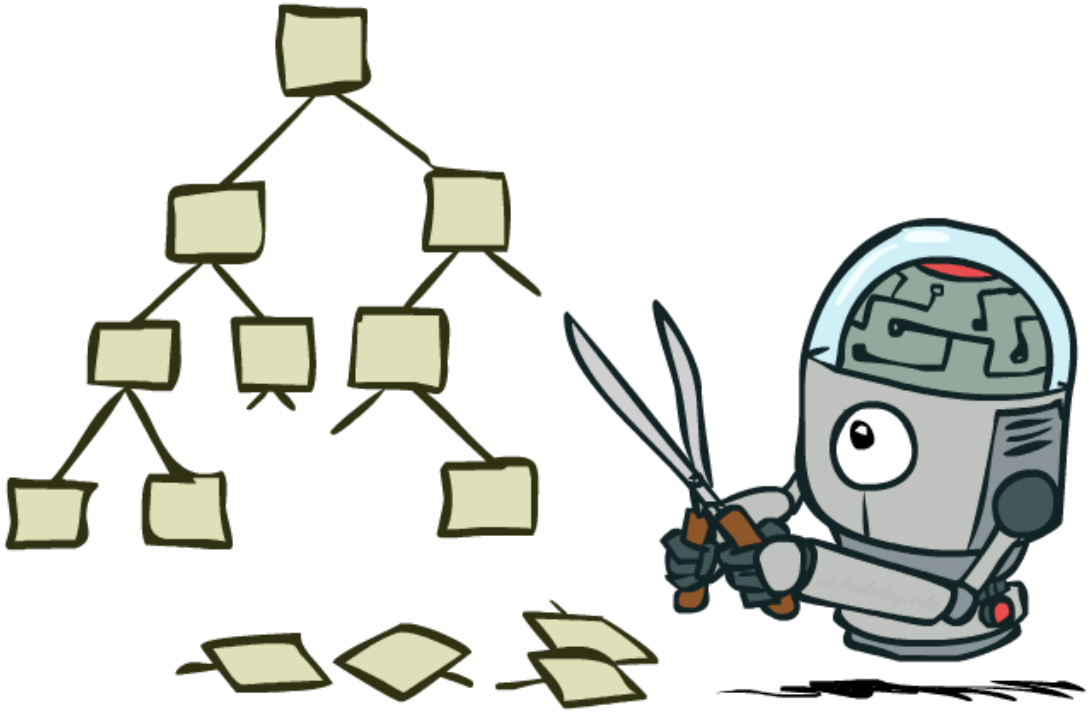
- If the game is always played with the same strategy, then it holds:

  $f = Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

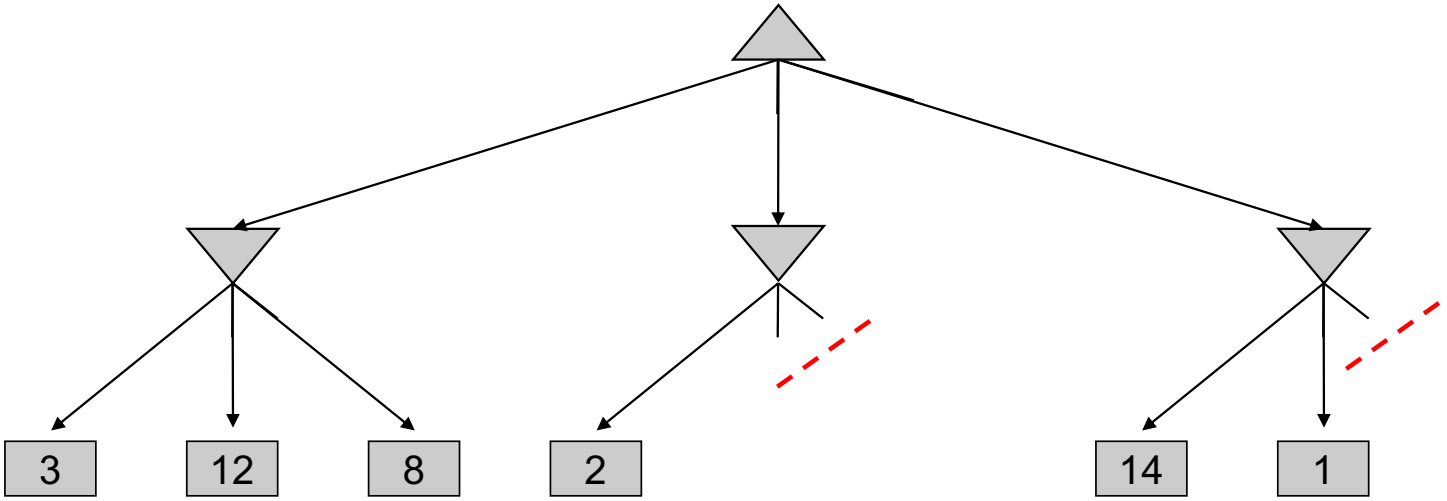
- If during the game play we estimate that some of the criteria are played more often than others, then we can introduce change in the weight array

- Changing weights are a result of **learning**.
- How to learn?:
  - Genetic algorithms
  - Neural networks
  - Learning automata
  - …

Game Tree Pruning

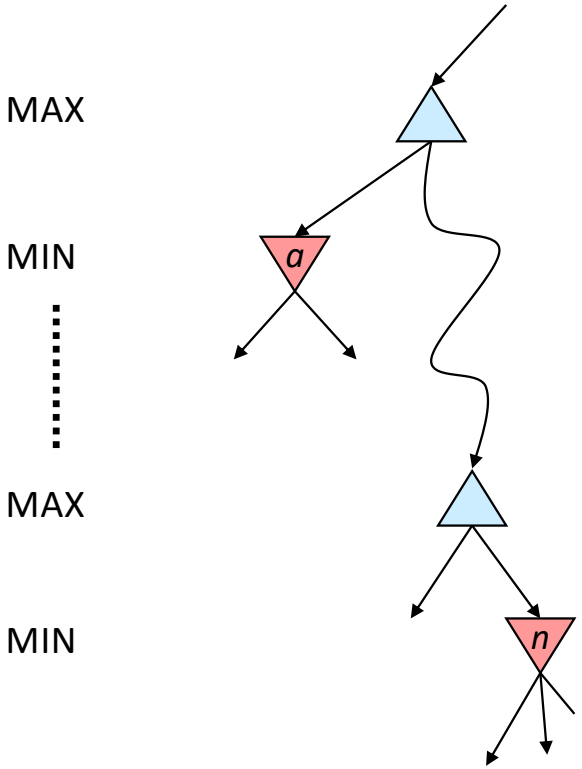


Minimax with alpha-beta Pruning



Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let *α* be the best value that MAX can get at any choice point along the current path from the root
  - If *n* becomes worse than *α*, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)



- MAX version is symmetric

Alpha-Beta Implementation

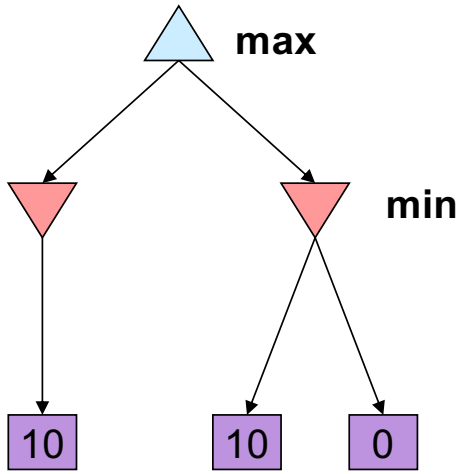
α: MAX's best option on path to root
β: MIN's best option on path to root

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

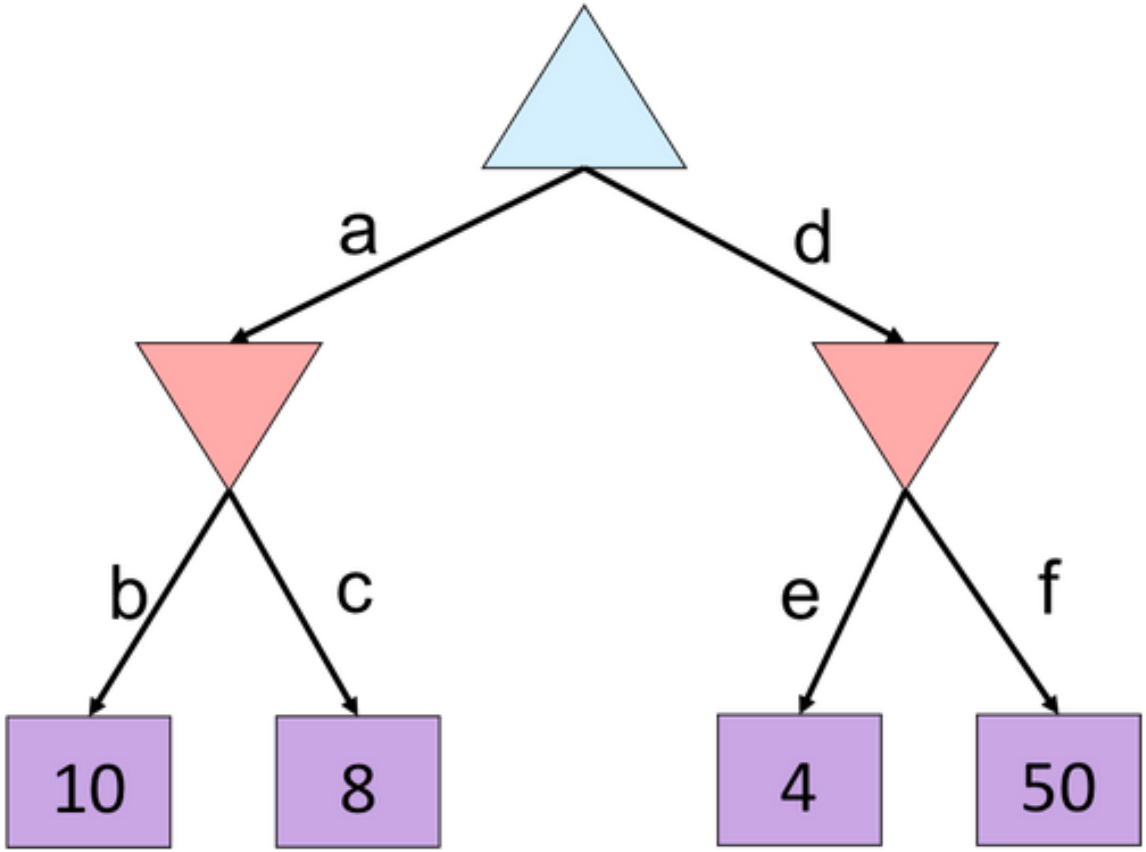
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Pruning Properties

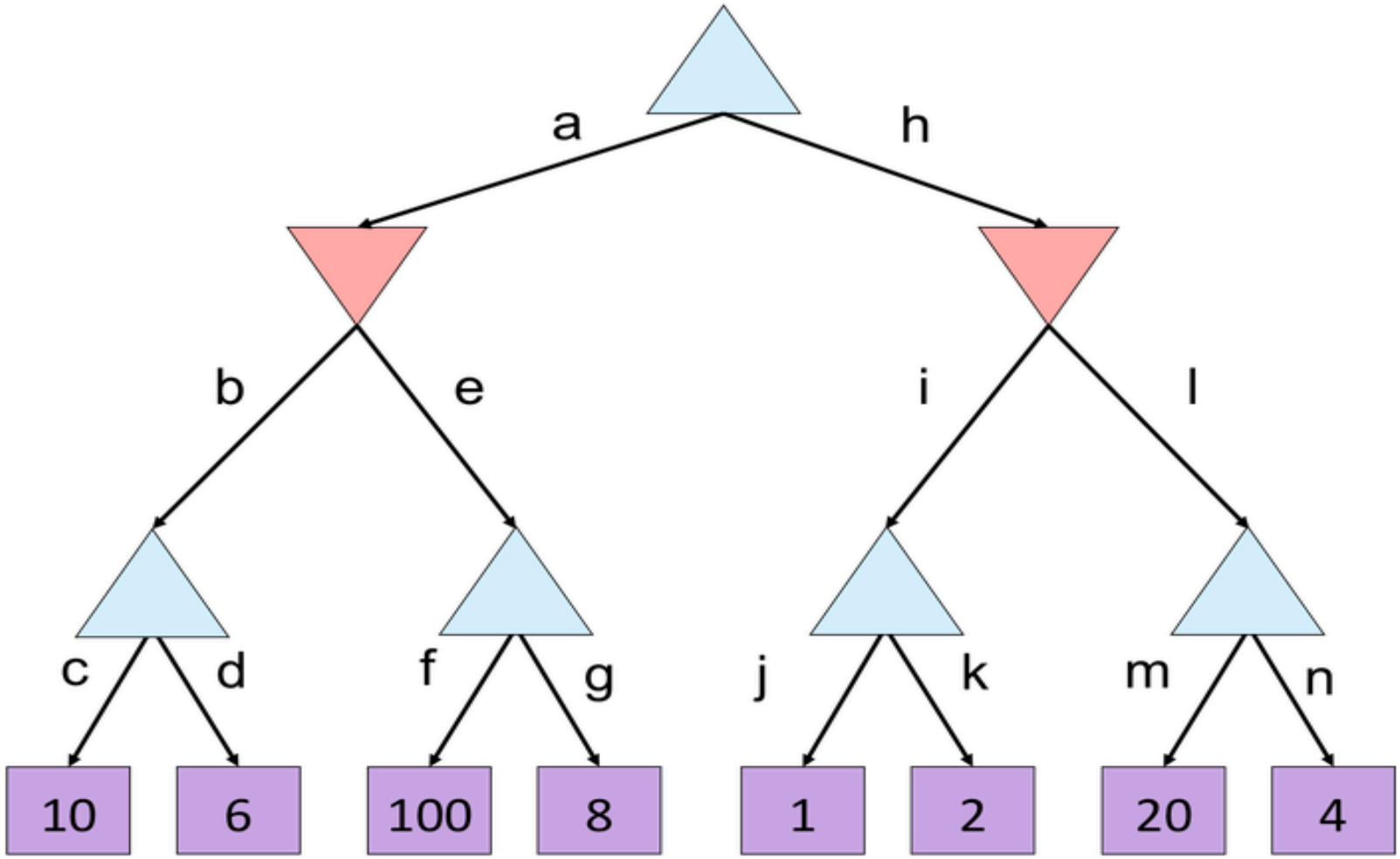
- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless …



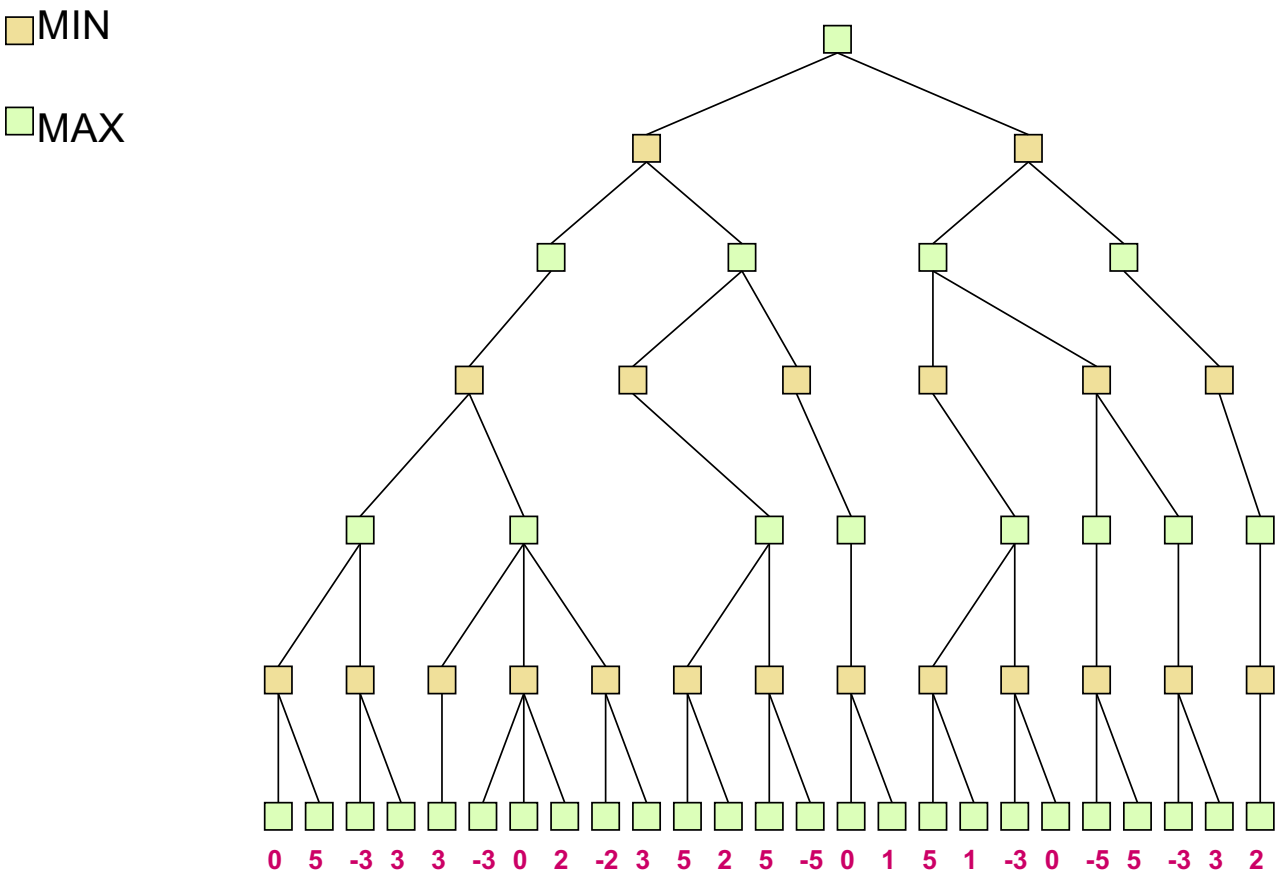
Alpha-Beta Quiz



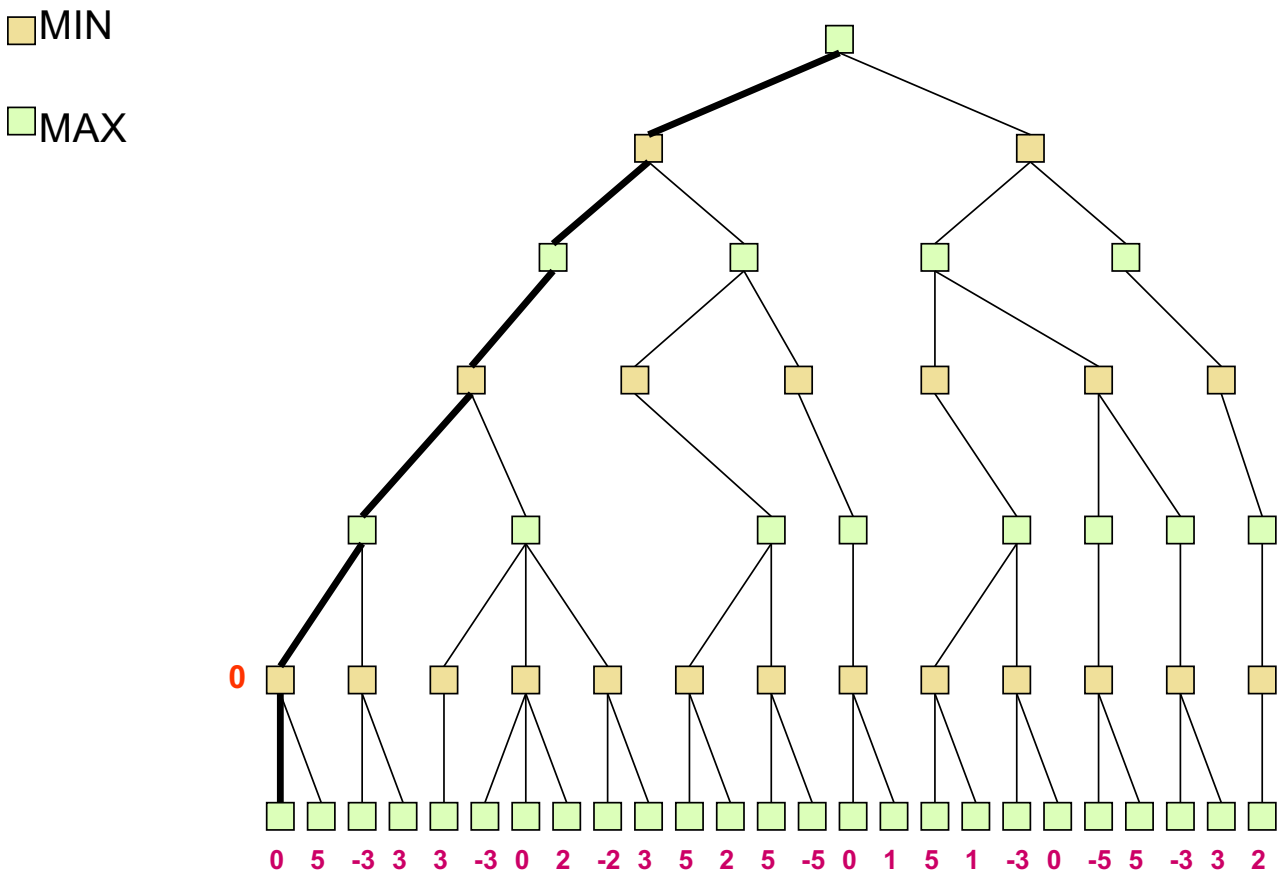
Alpha-Beta Quiz 2



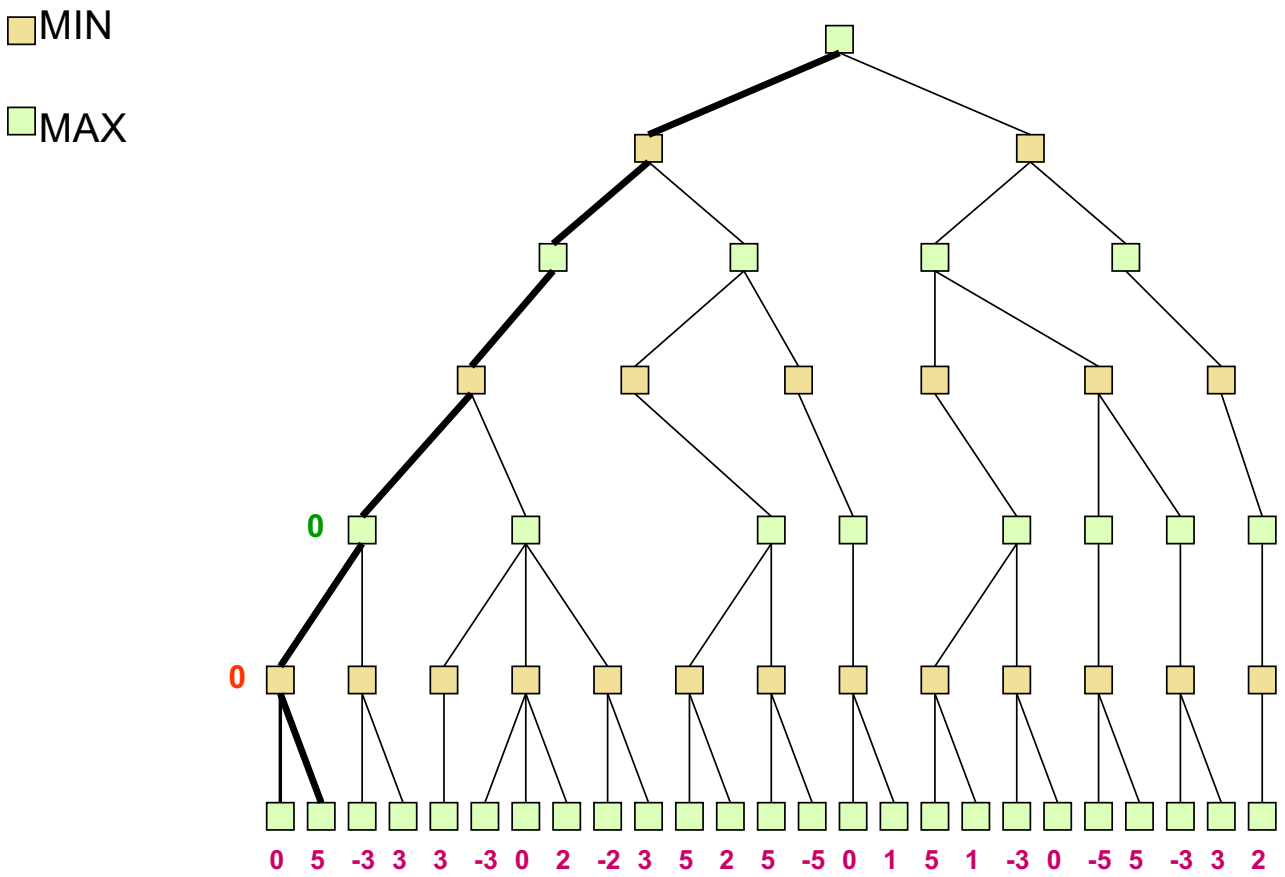
Example: α – β pruning



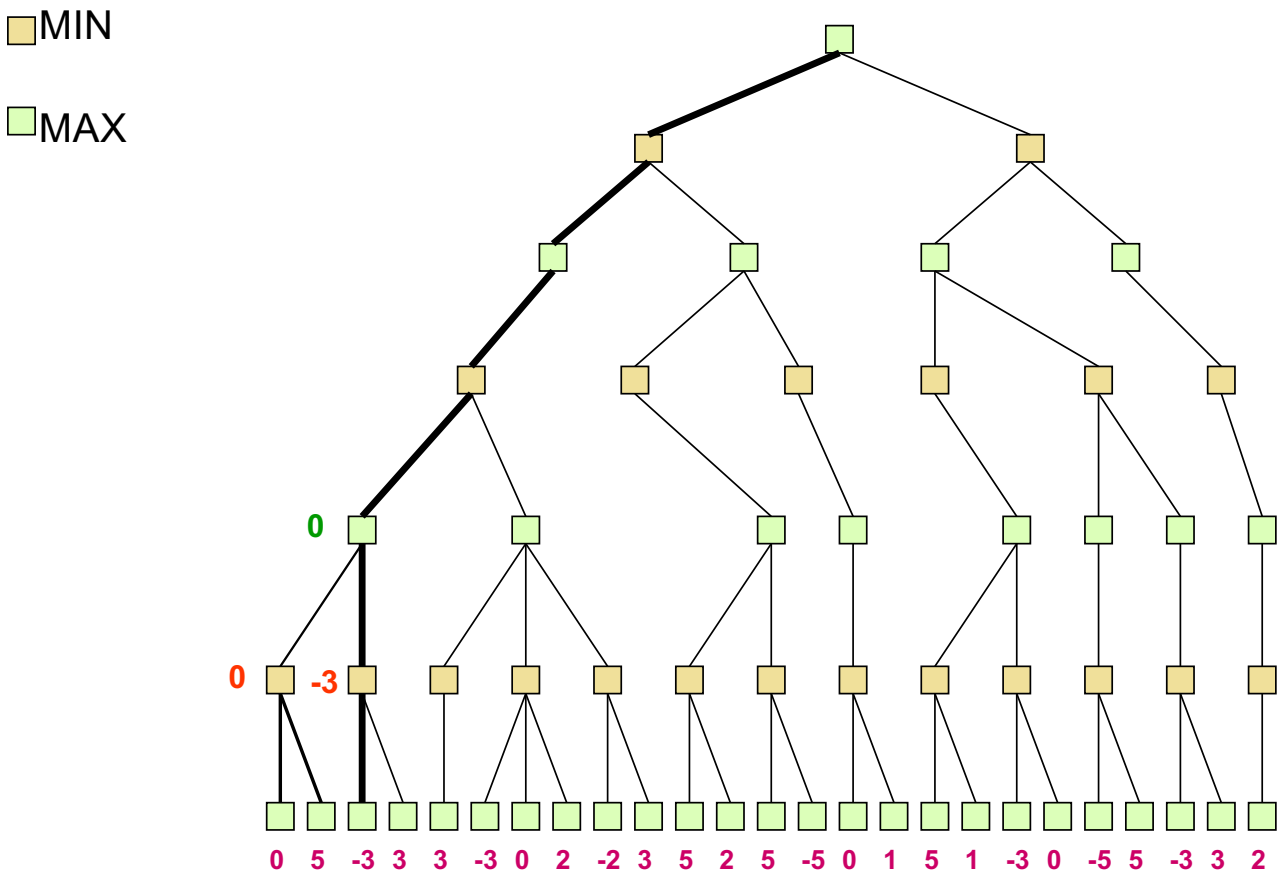
Example: α – β pruning



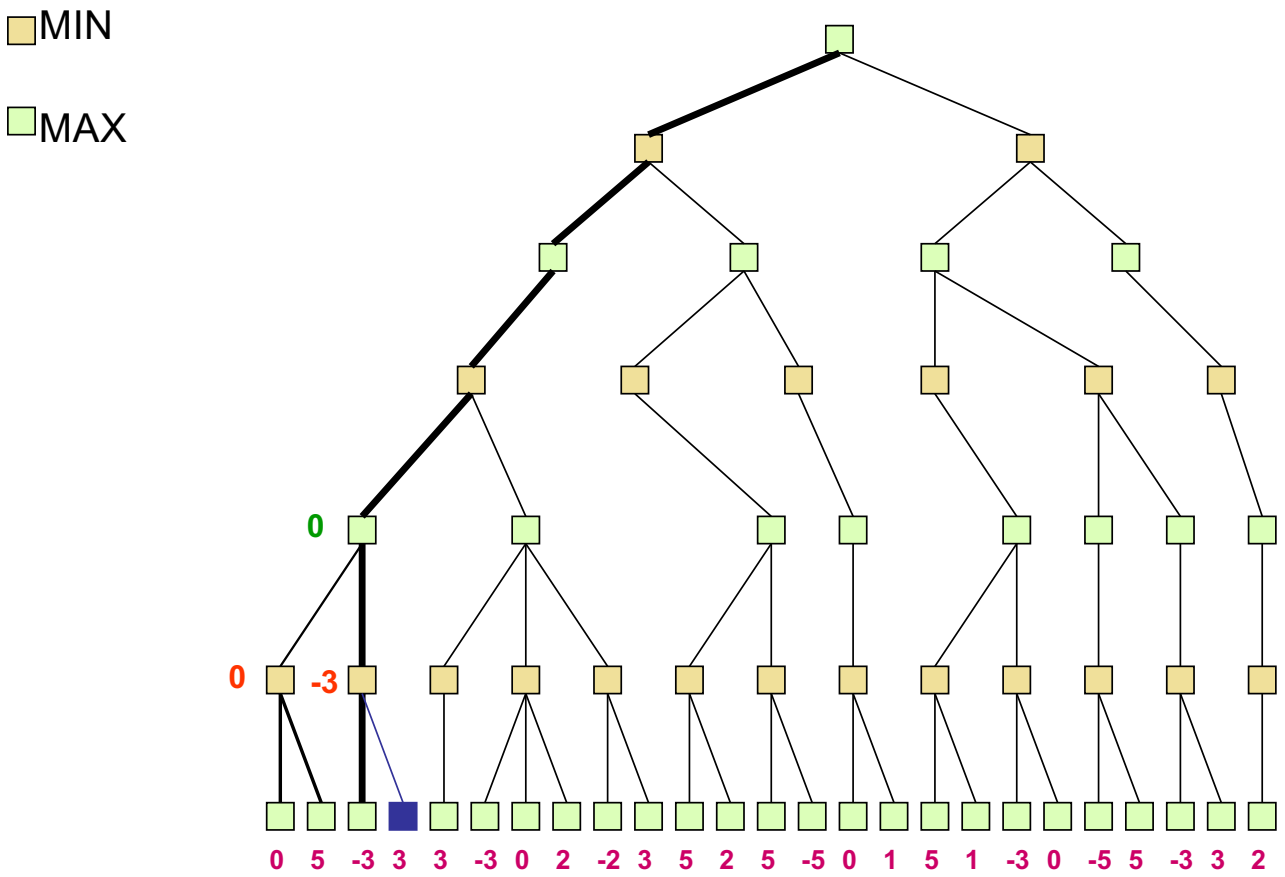
Example: α – β pruning



Example: α – β pruning

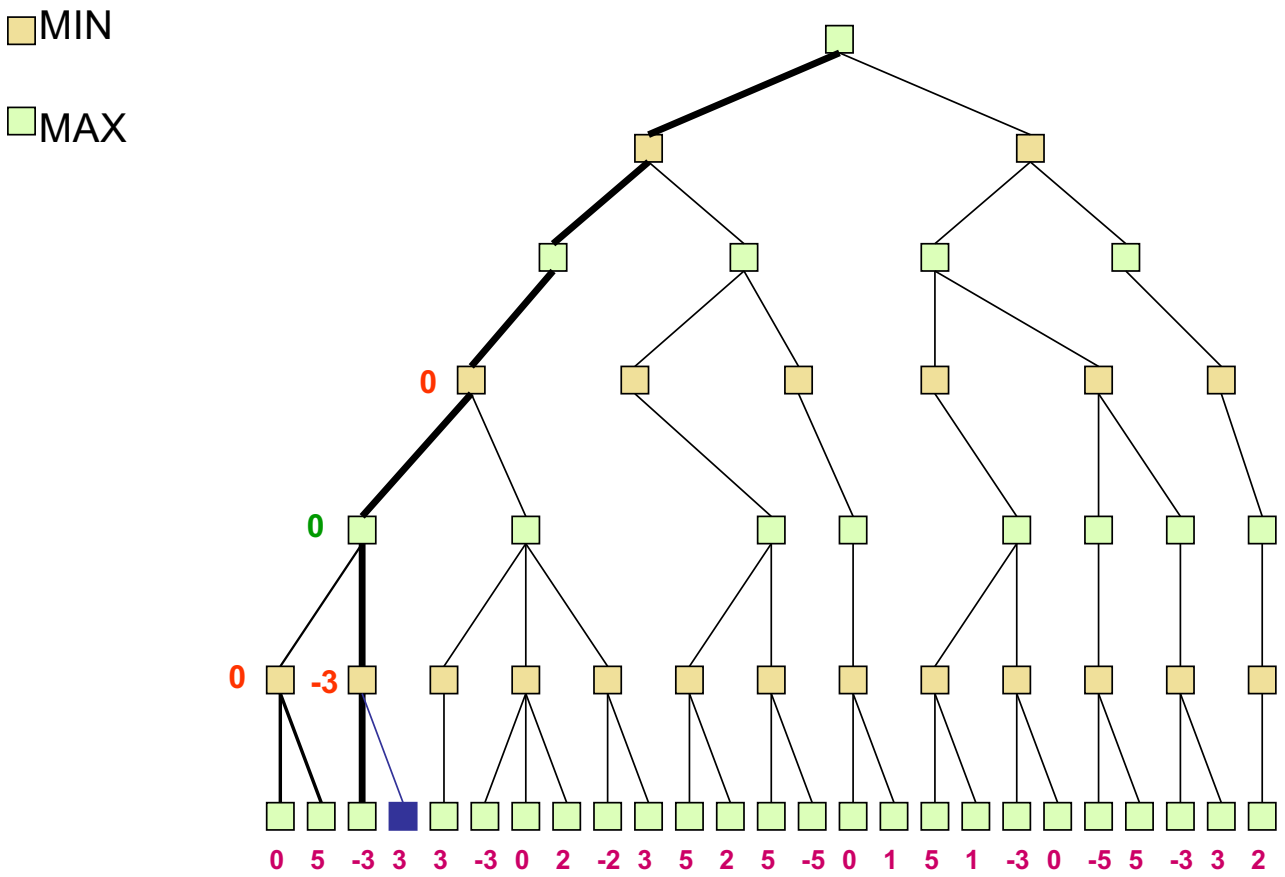


Example: α – β pruning

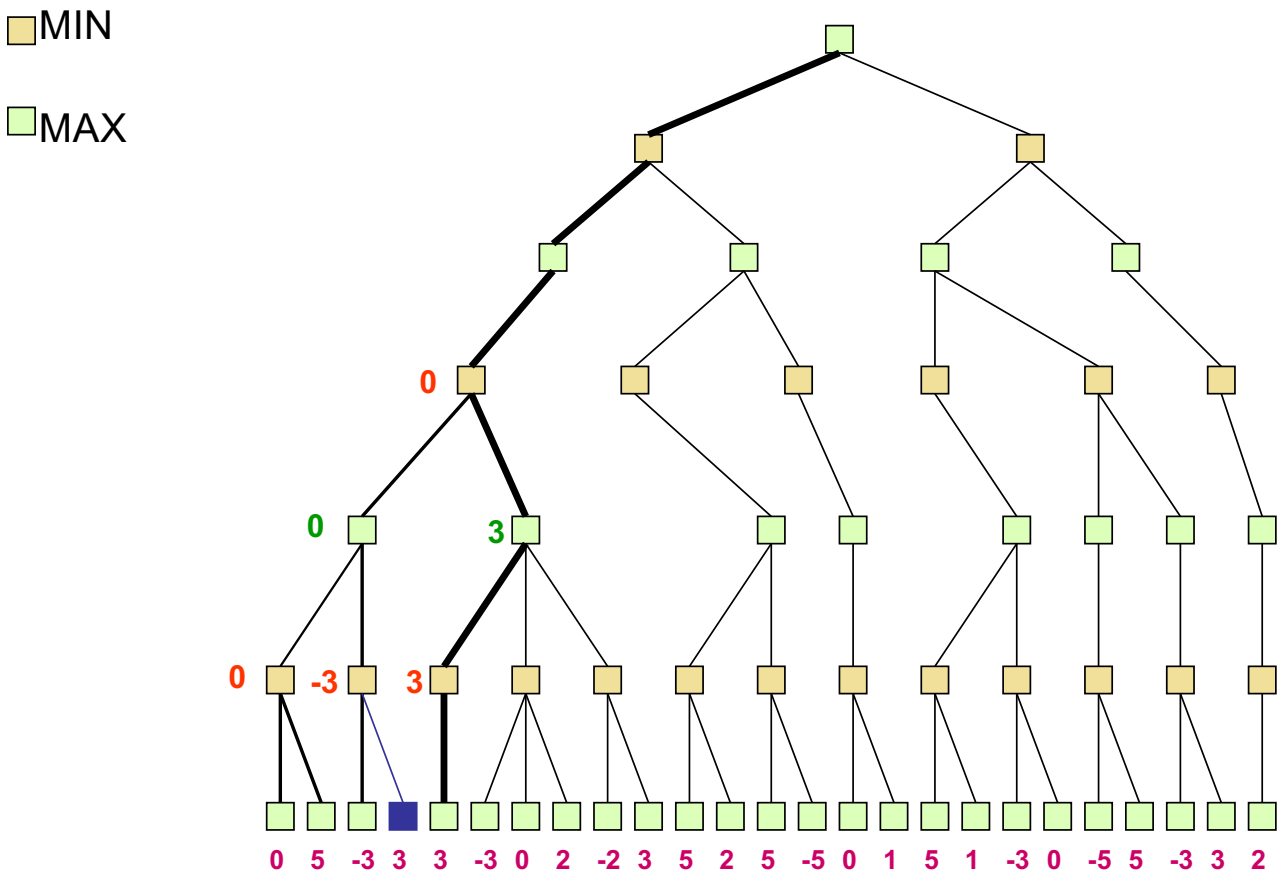


- Prune the sub-tree with a MAX node at its root for which the α value is greater or equal than any β value in its MIN node ancestors
- Prune the sub-tree with a MIN node at its root for which the β value is less or equal than any α value in its MAX node ancestors

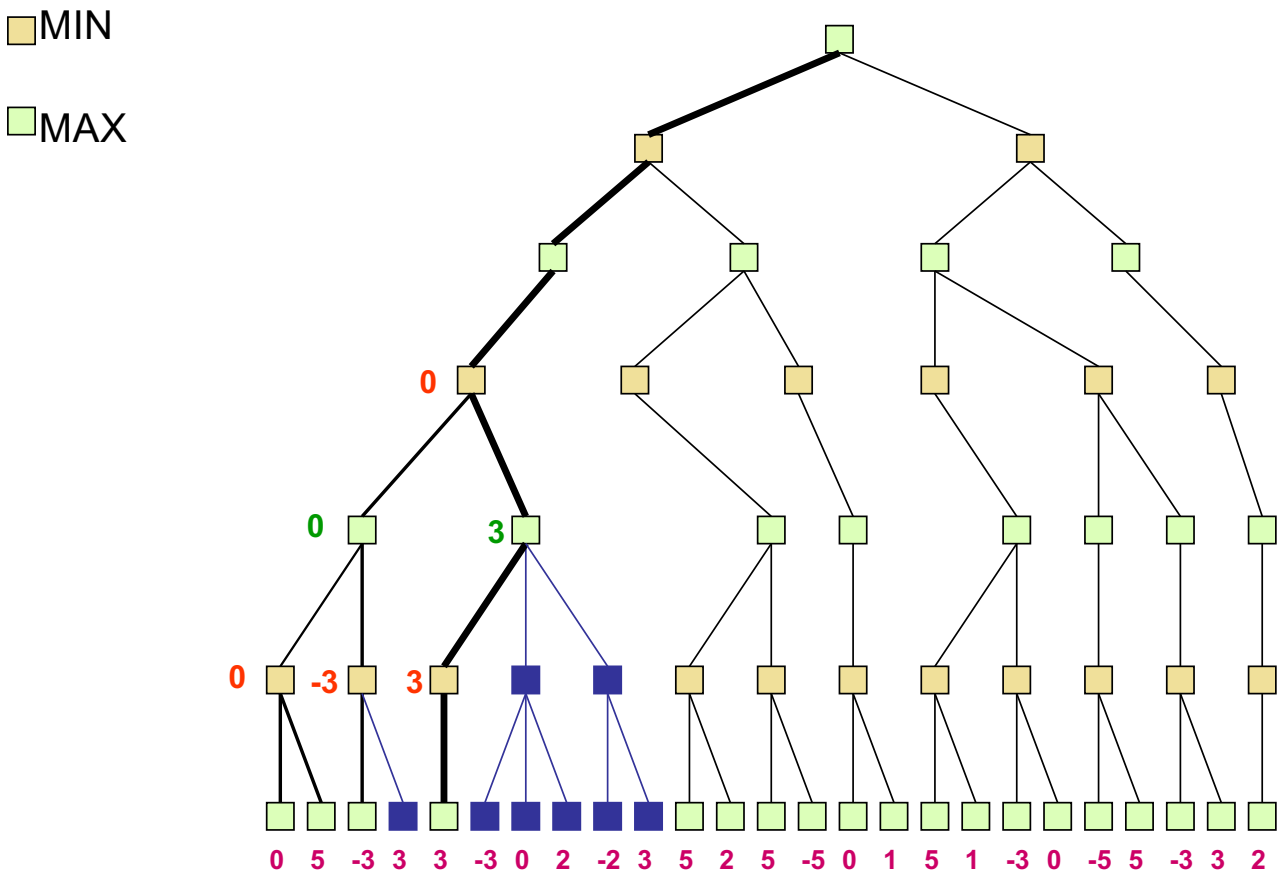
Example: α – β pruning



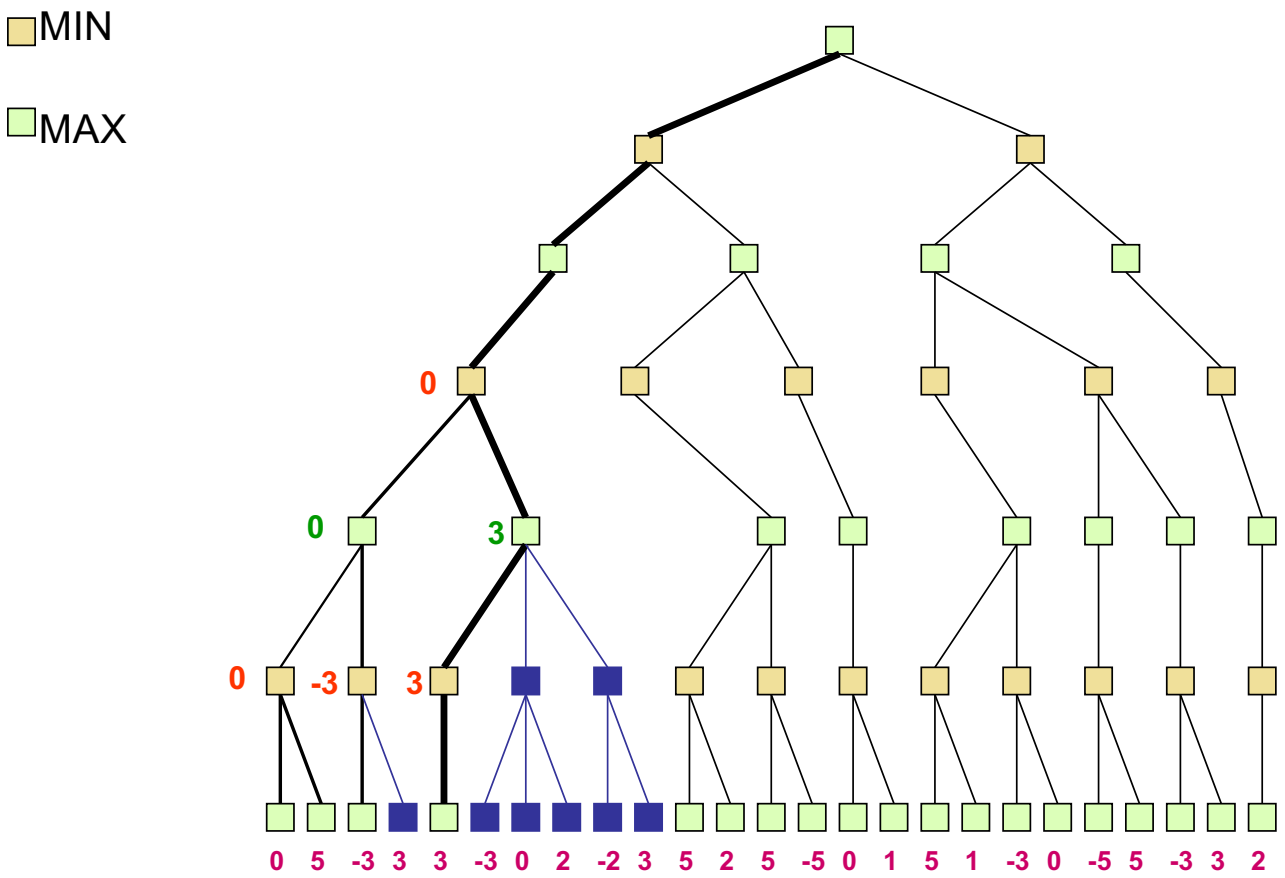
Example: α – β pruning



Example: α – β pruning

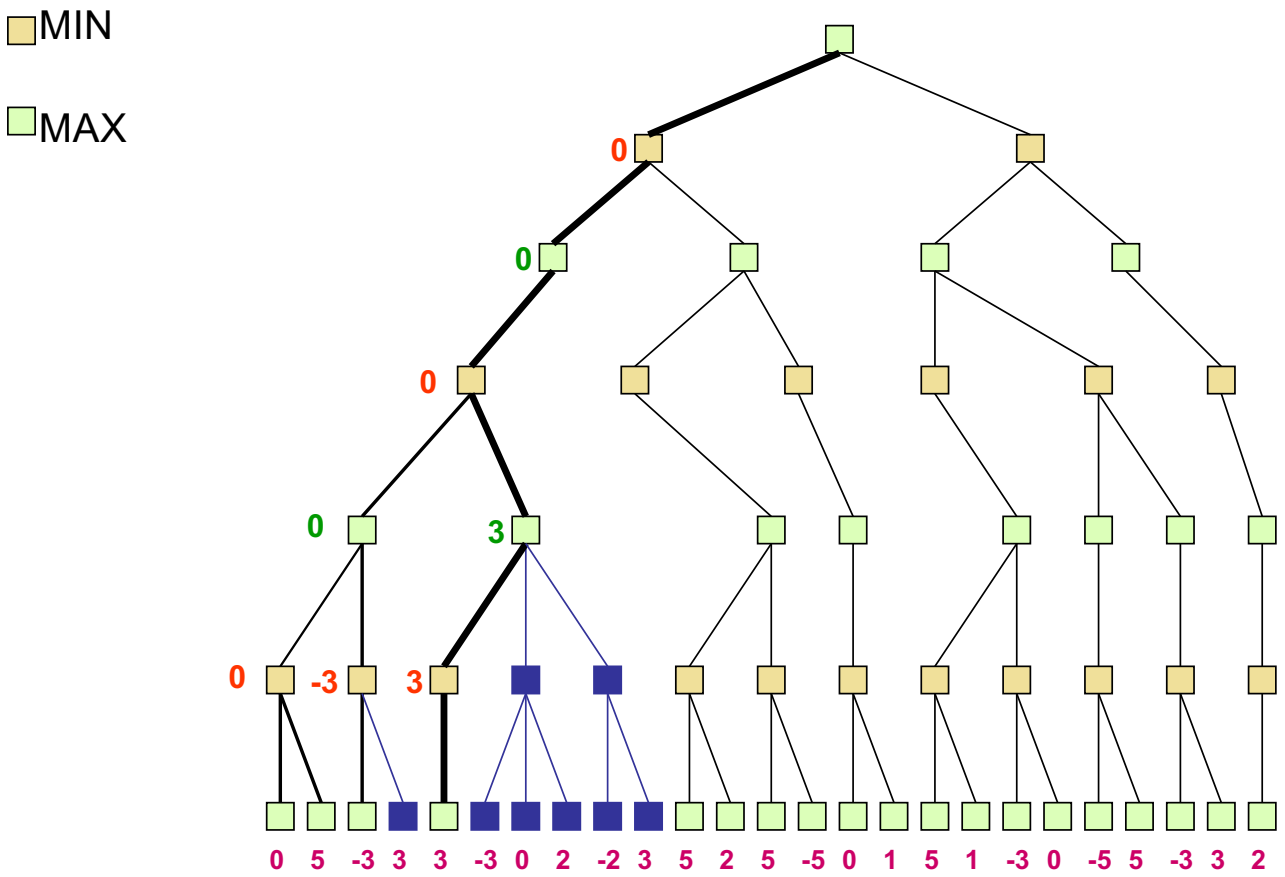


Example: α – β pruning

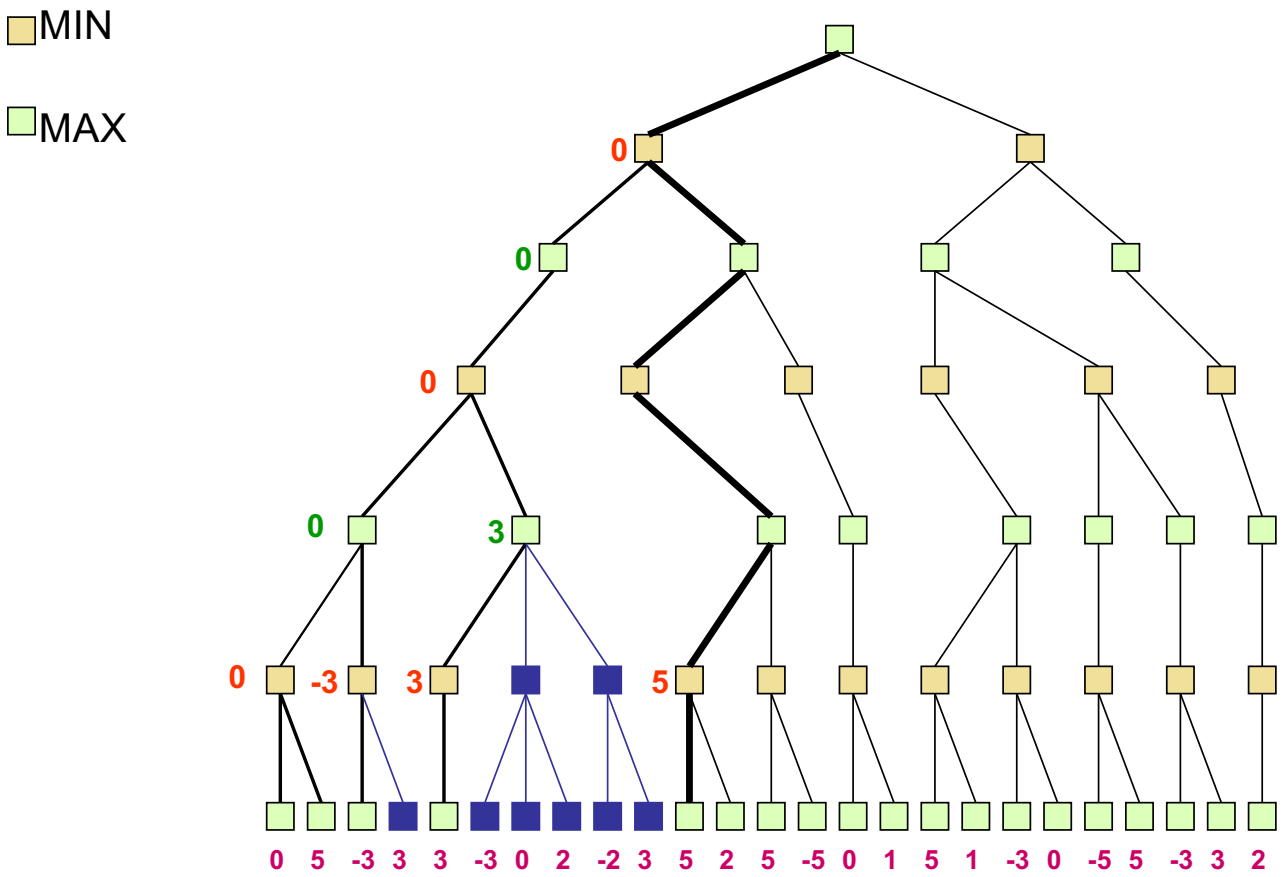


- Prune the sub-tree with a MAX node at its root for which the α value is greater or equal than any β value in its MIN node ancestors
- Prune the sub-tree with a MIN node at its root for which the β value is less or equal than any α value in its MAX node ancestors

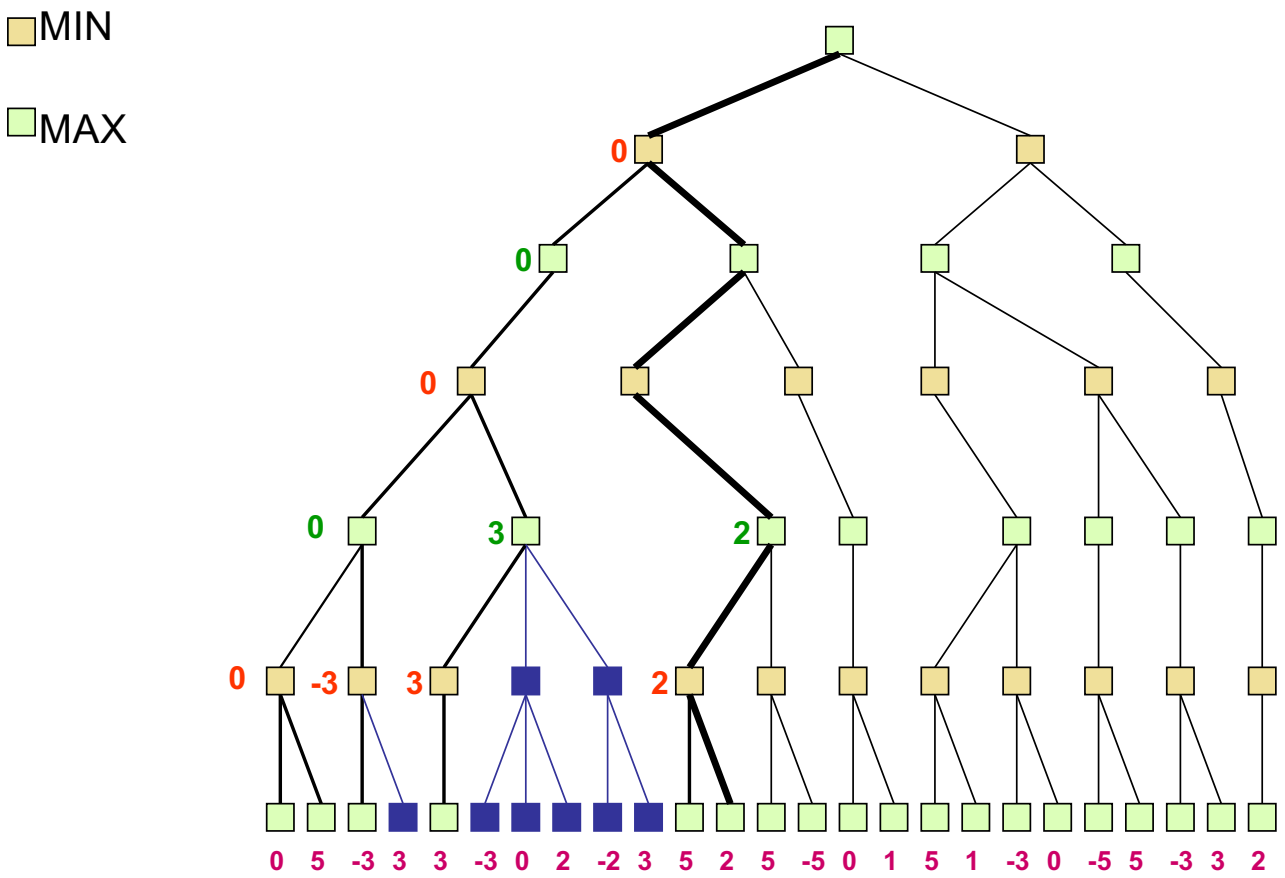
Example: α – β pruning



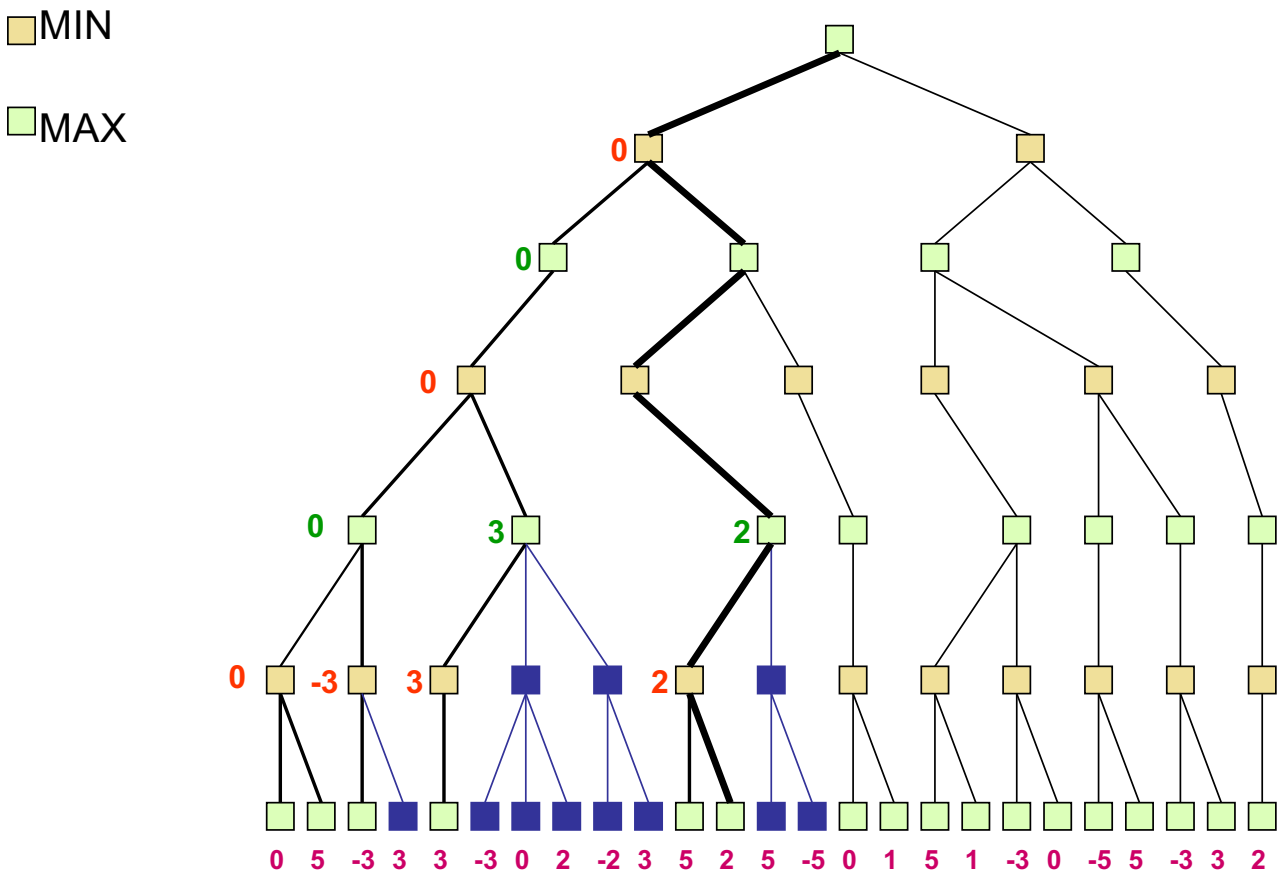
Example: α – β pruning



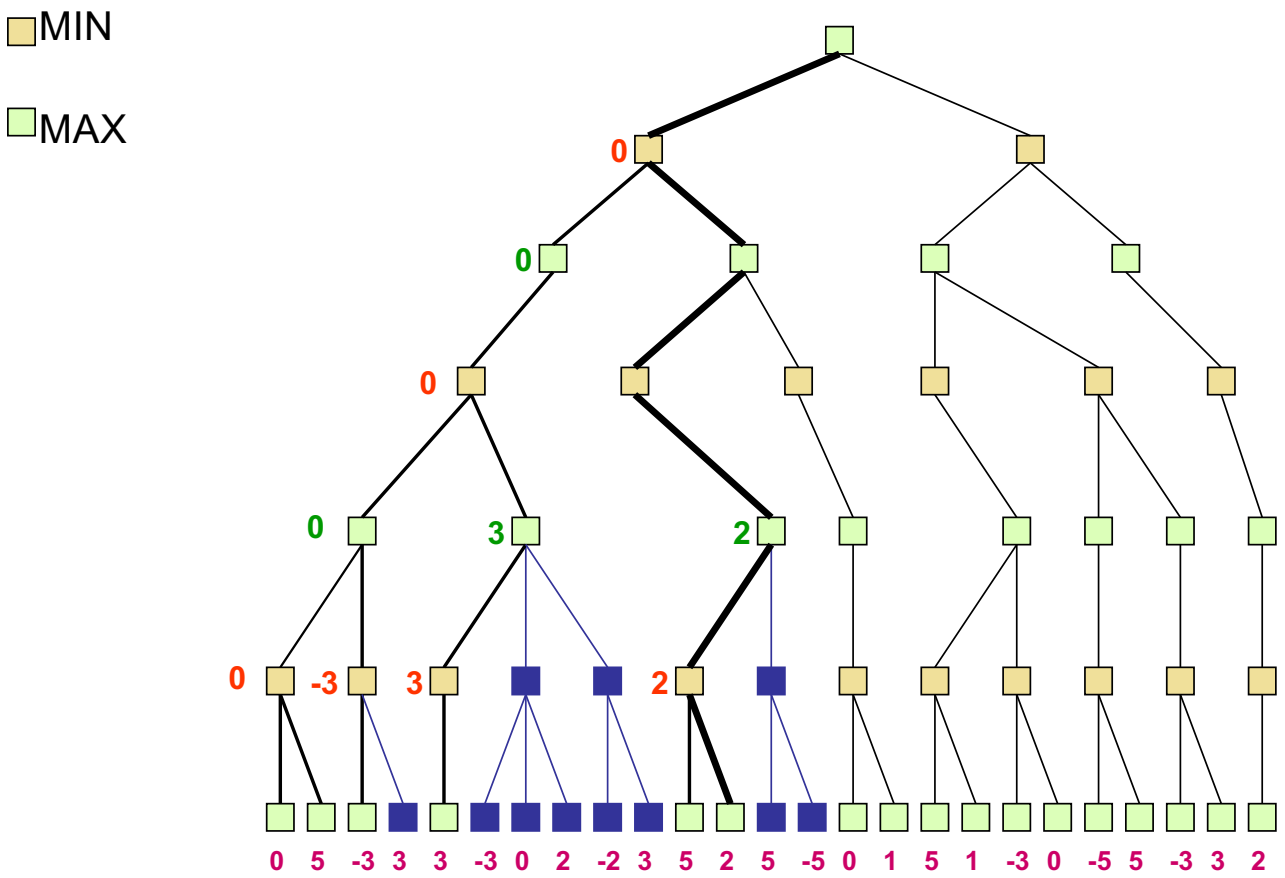
Example: α – β pruning



Example: α – β pruning

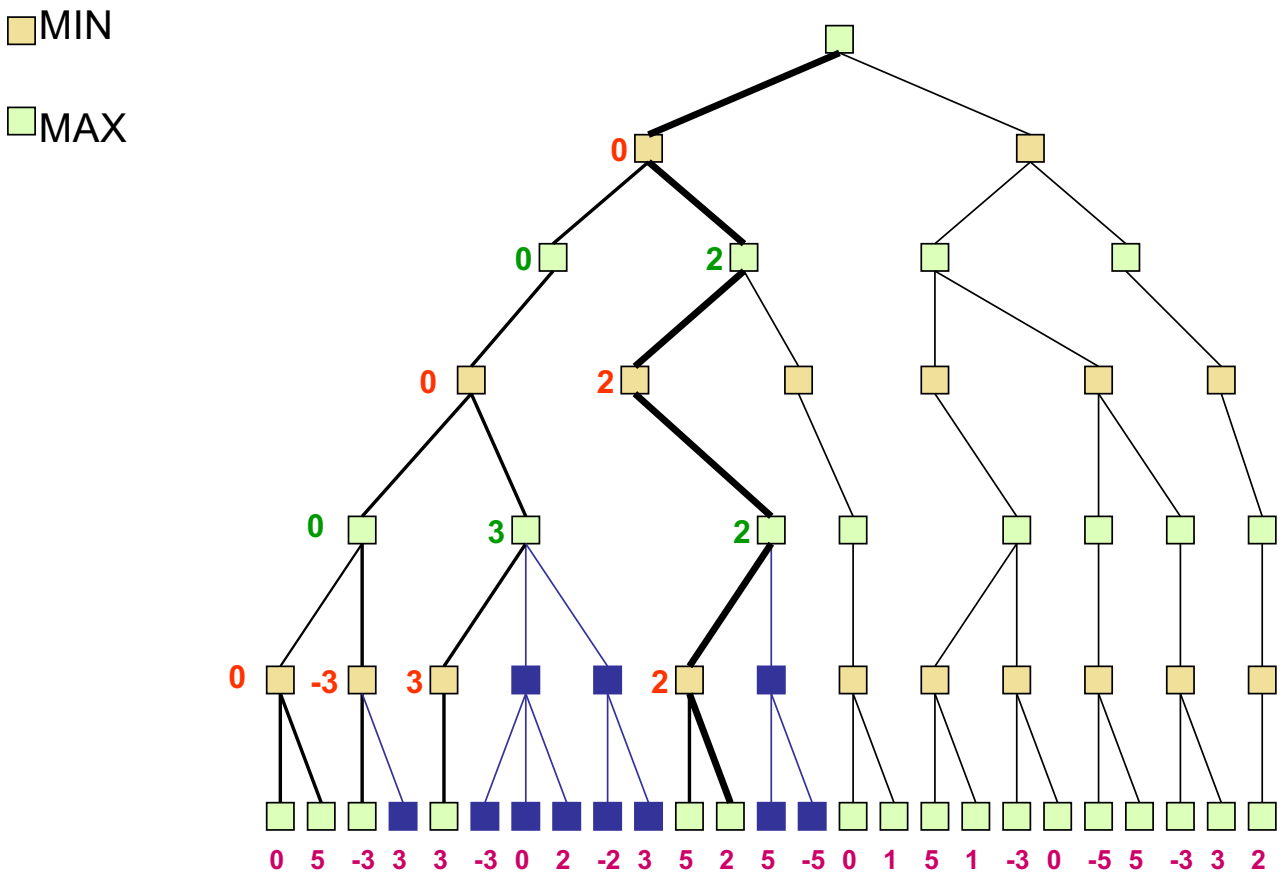


Example: α – β pruning

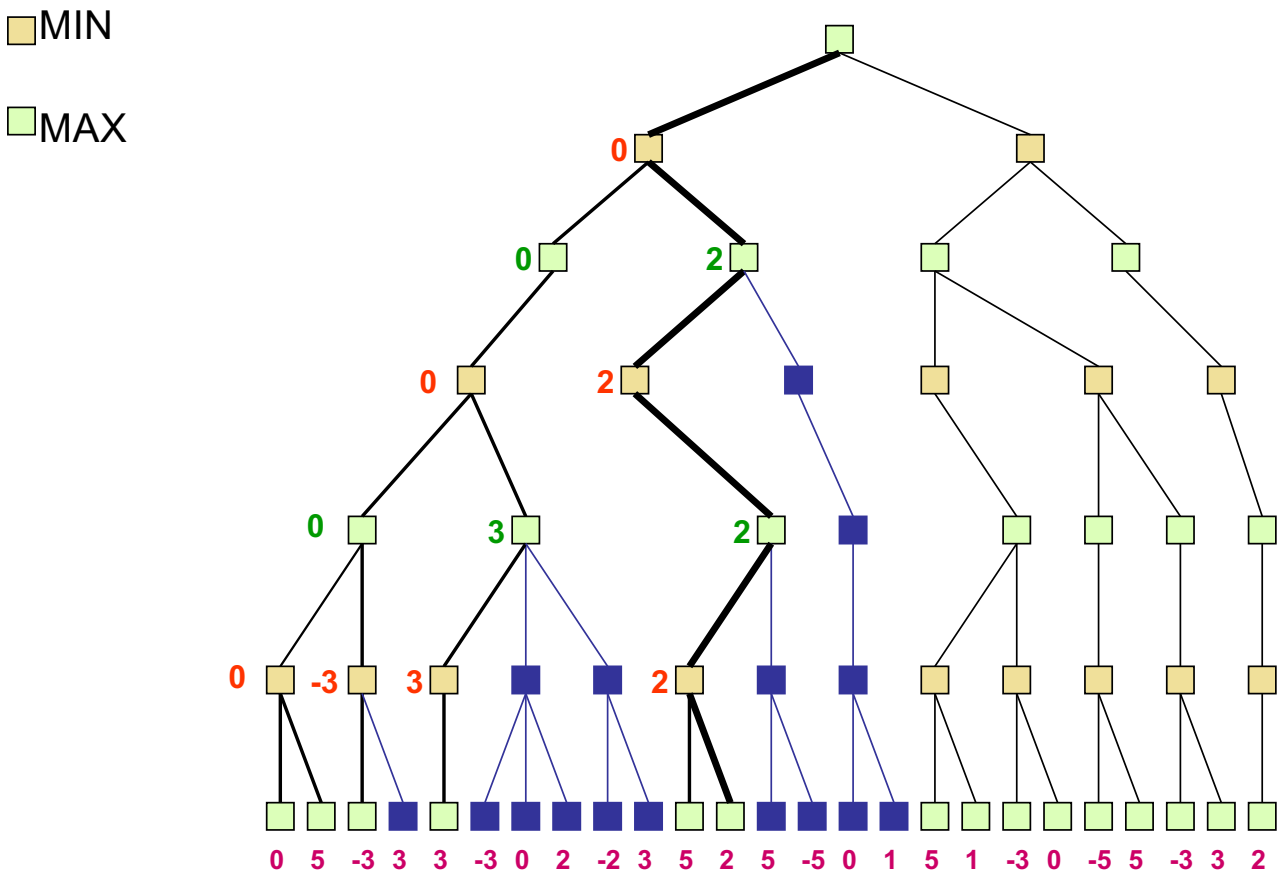


- Prune the sub-tree with a MAX node at its root for which the α value is greater or equal than any β value in its MIN node ancestors
- Prune the sub-tree with a MIN node at its root for which the β value is less or equal than any α value in its MAX node ancestors

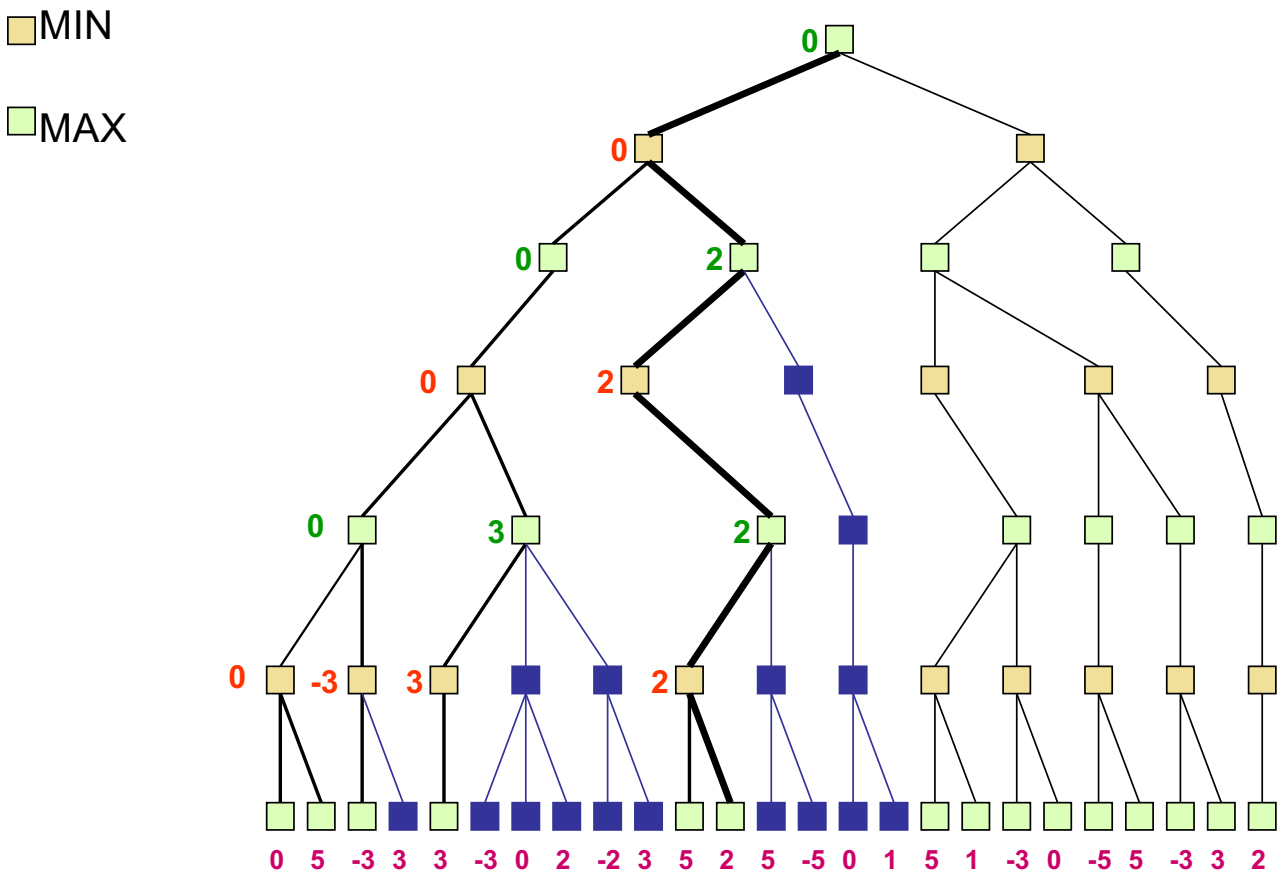
Example: α – β pruning



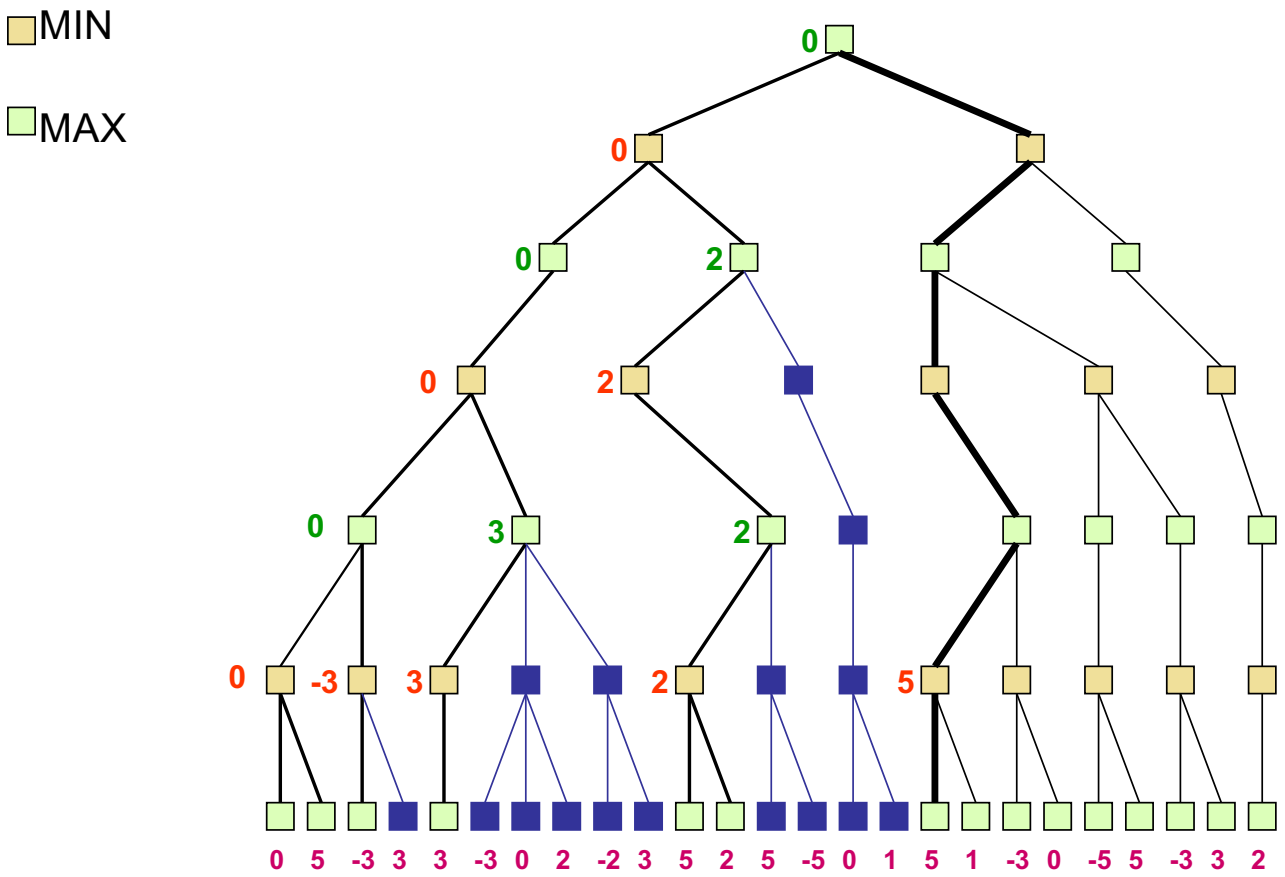
Example: α – β pruning



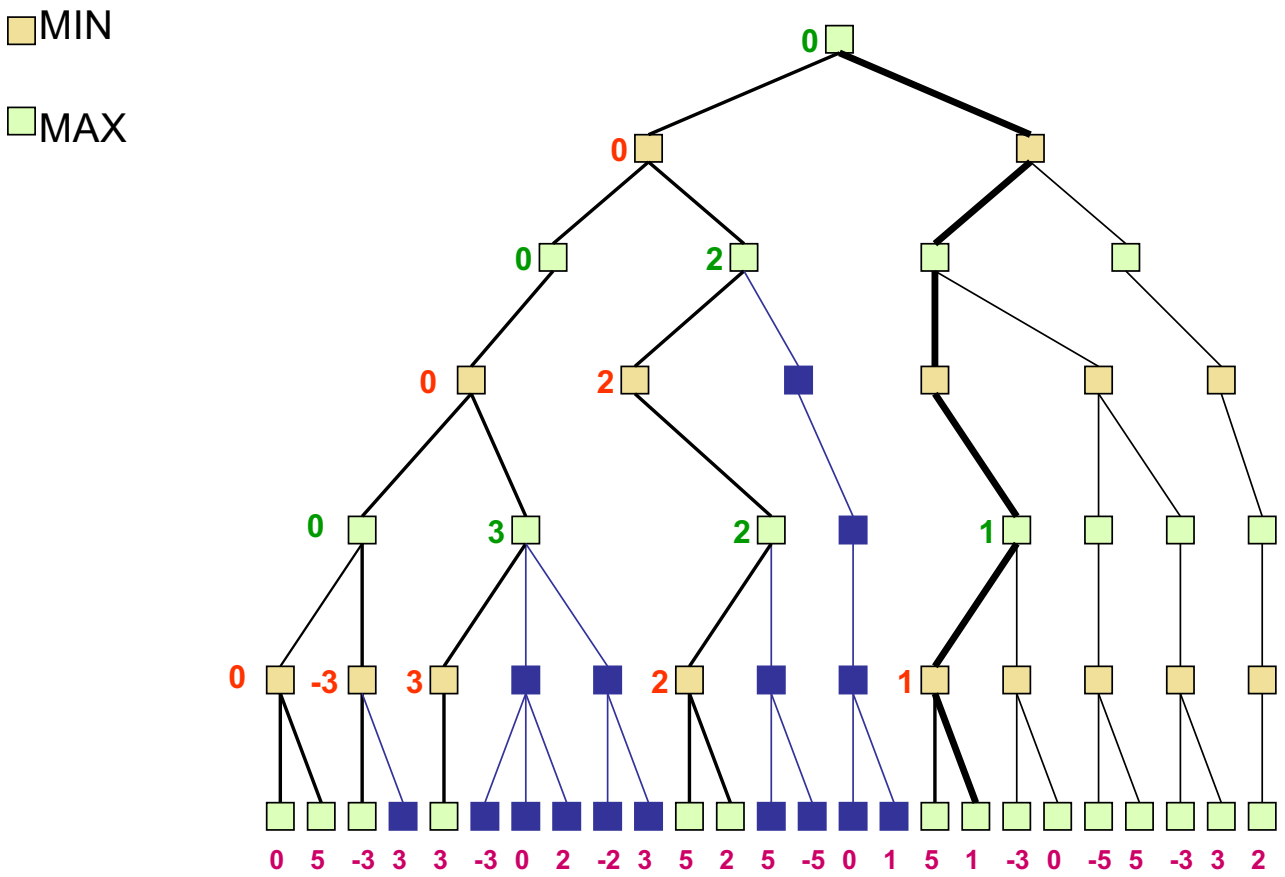
Example: α – β pruning



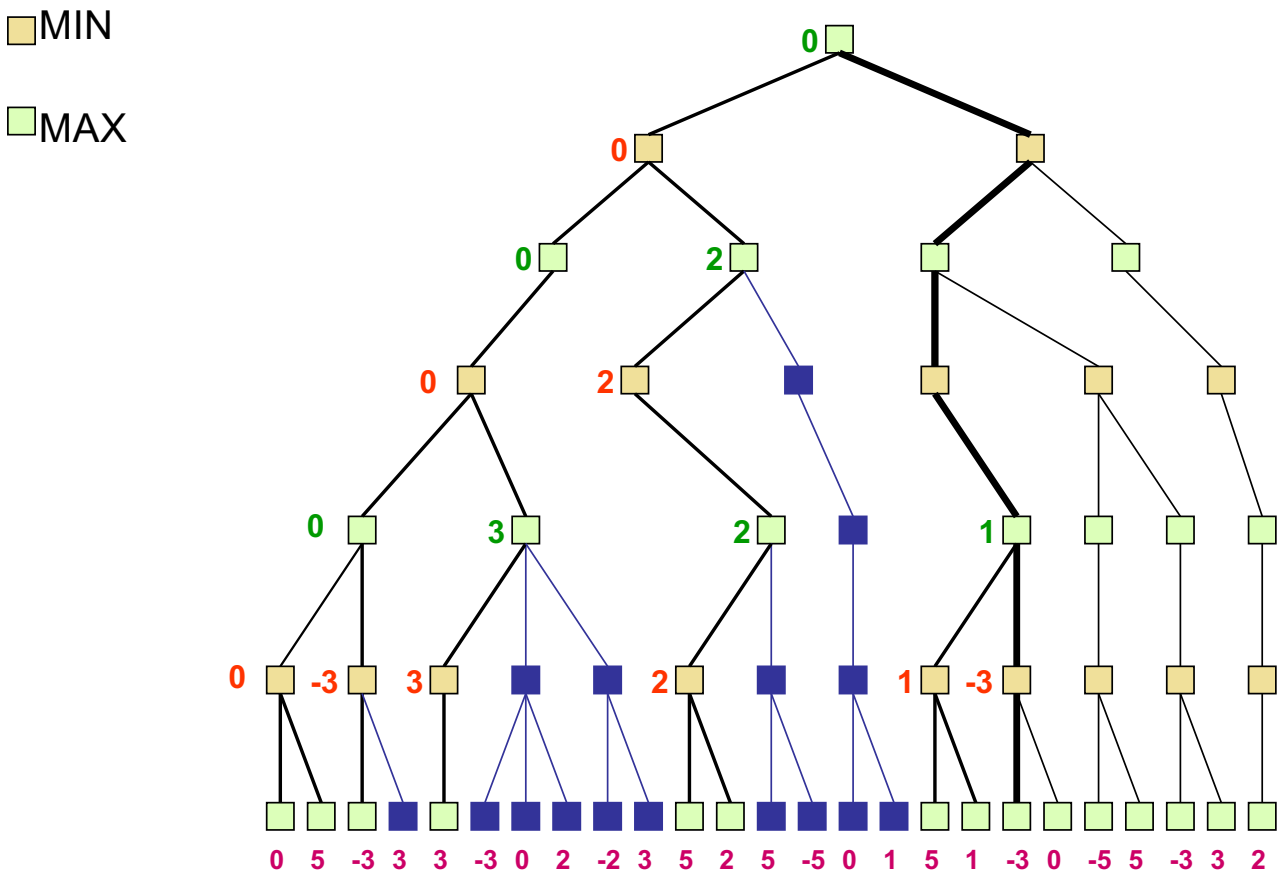
Example: α – β pruning



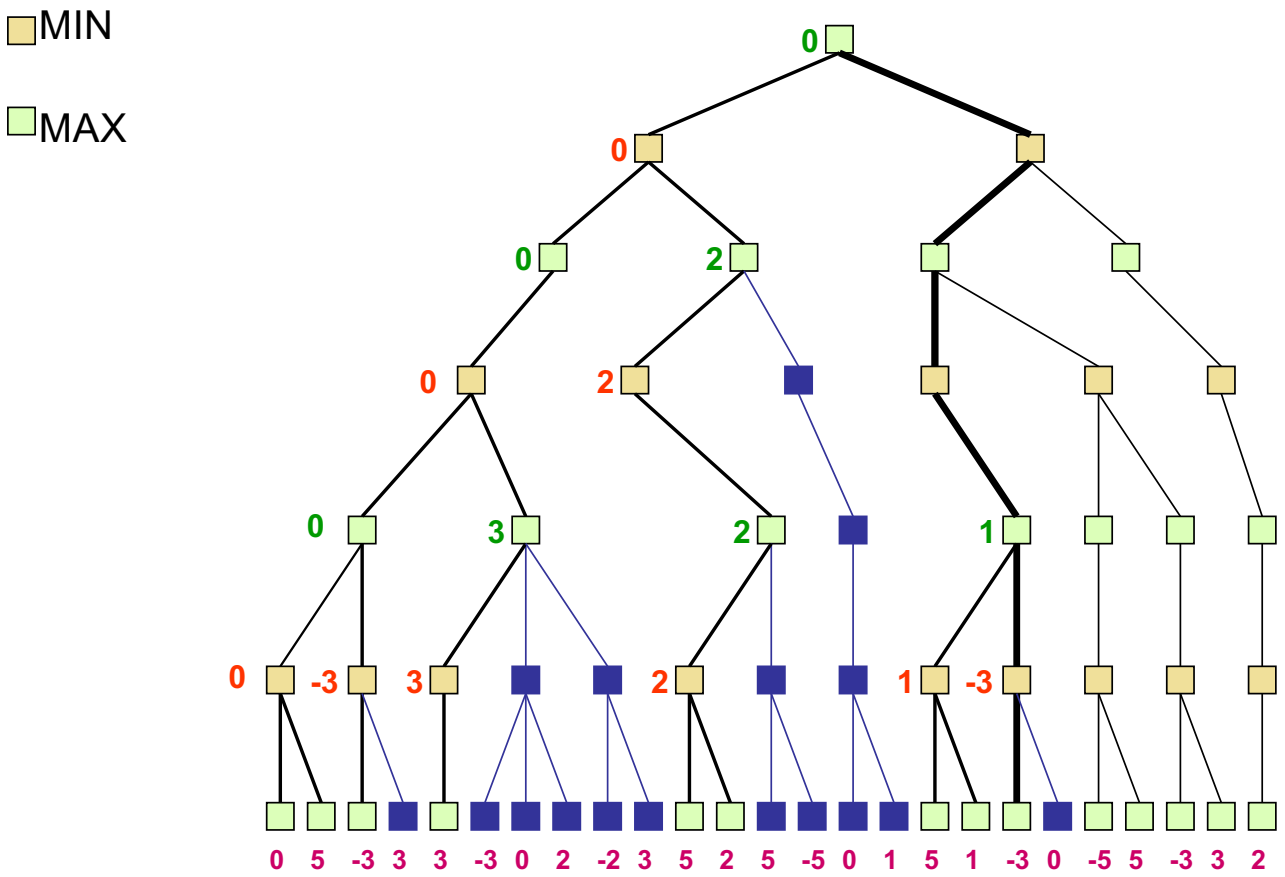
Example: α – β pruning



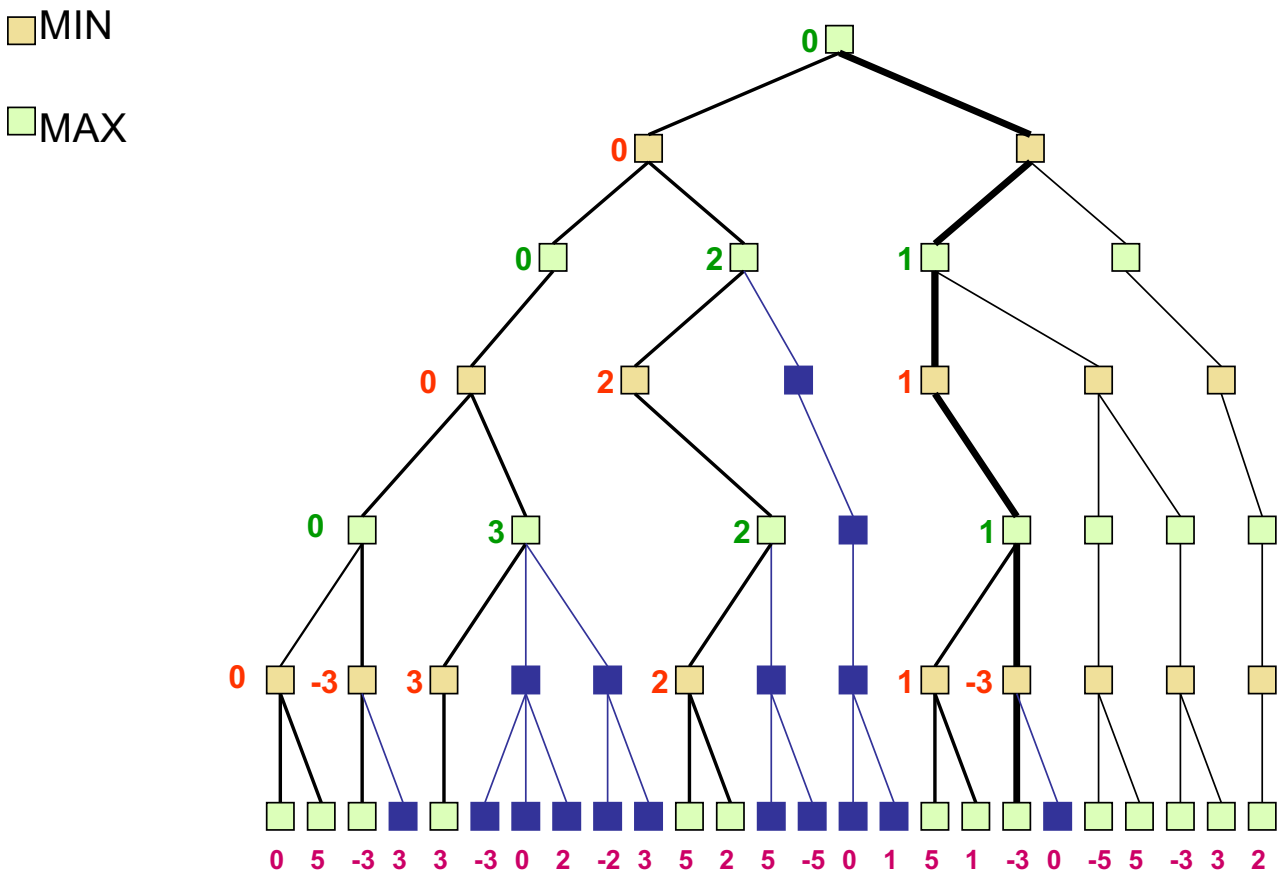
Example: α – β pruning



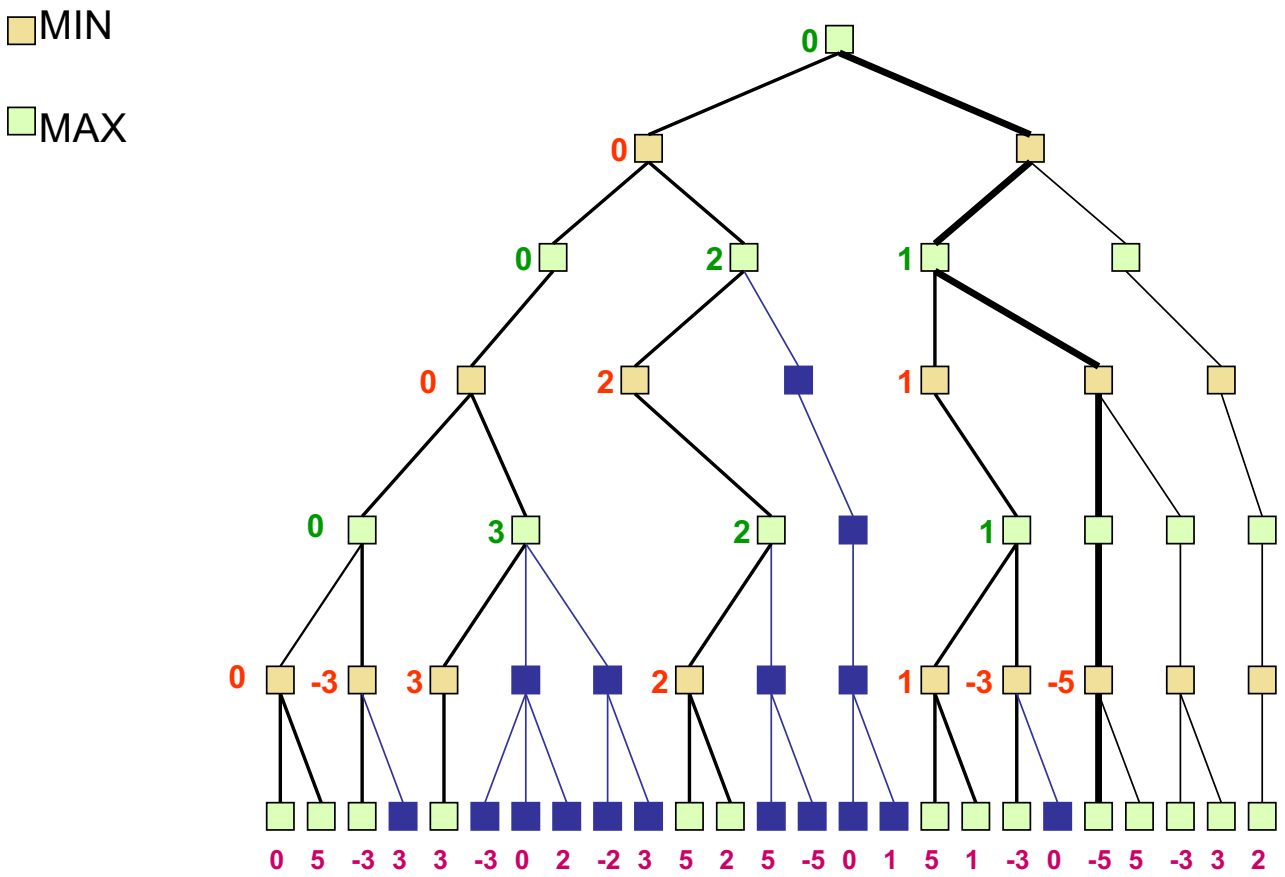
Example: α – β pruning



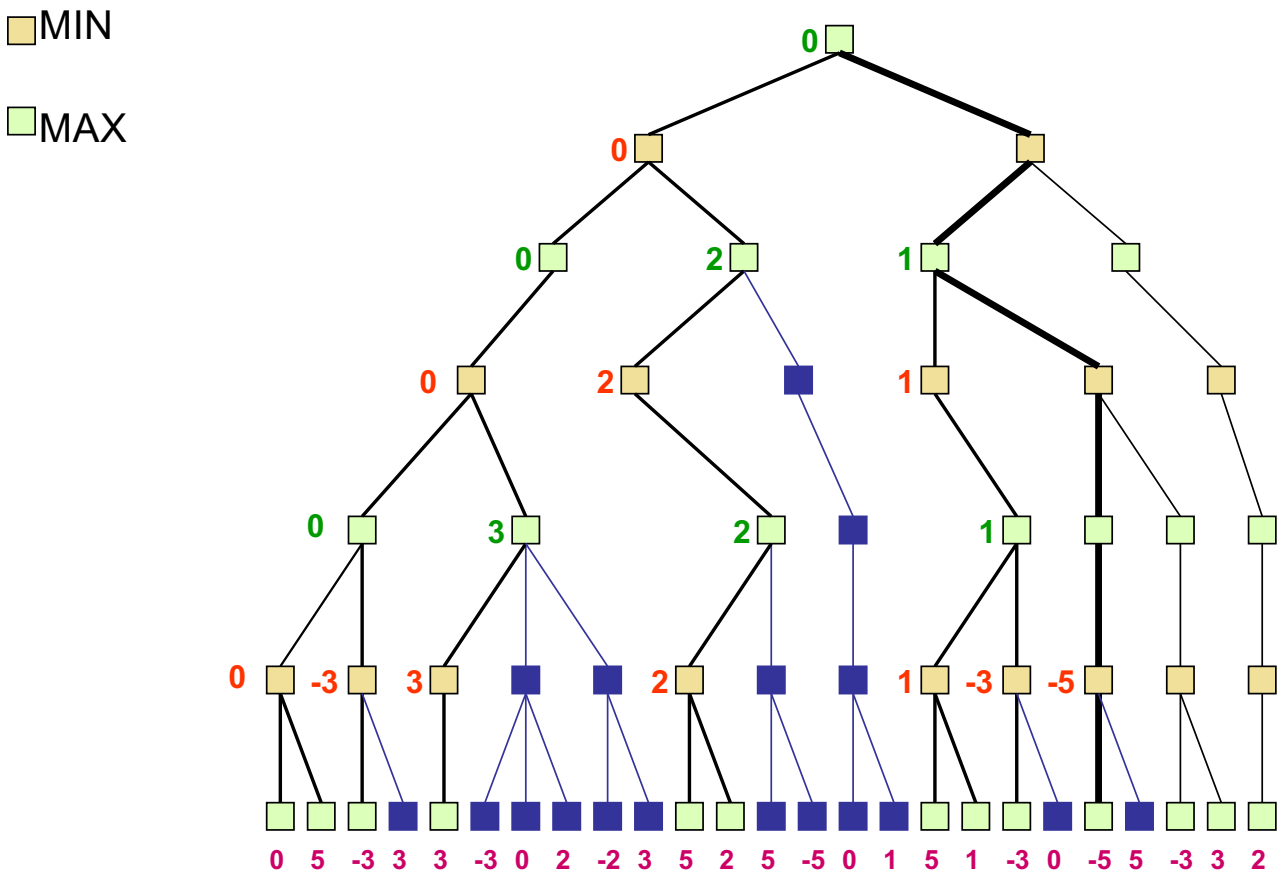
Example: α – β pruning



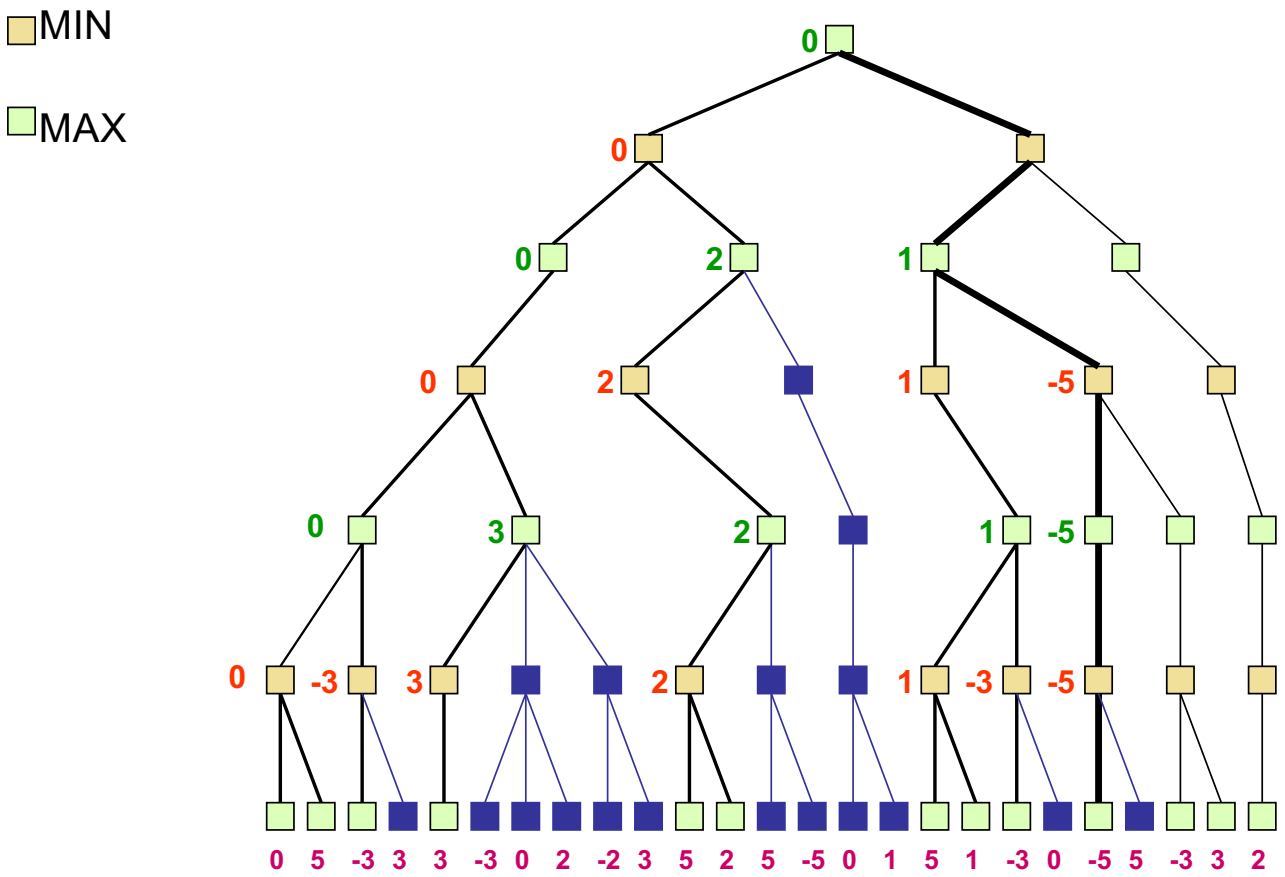
Example: α – β pruning



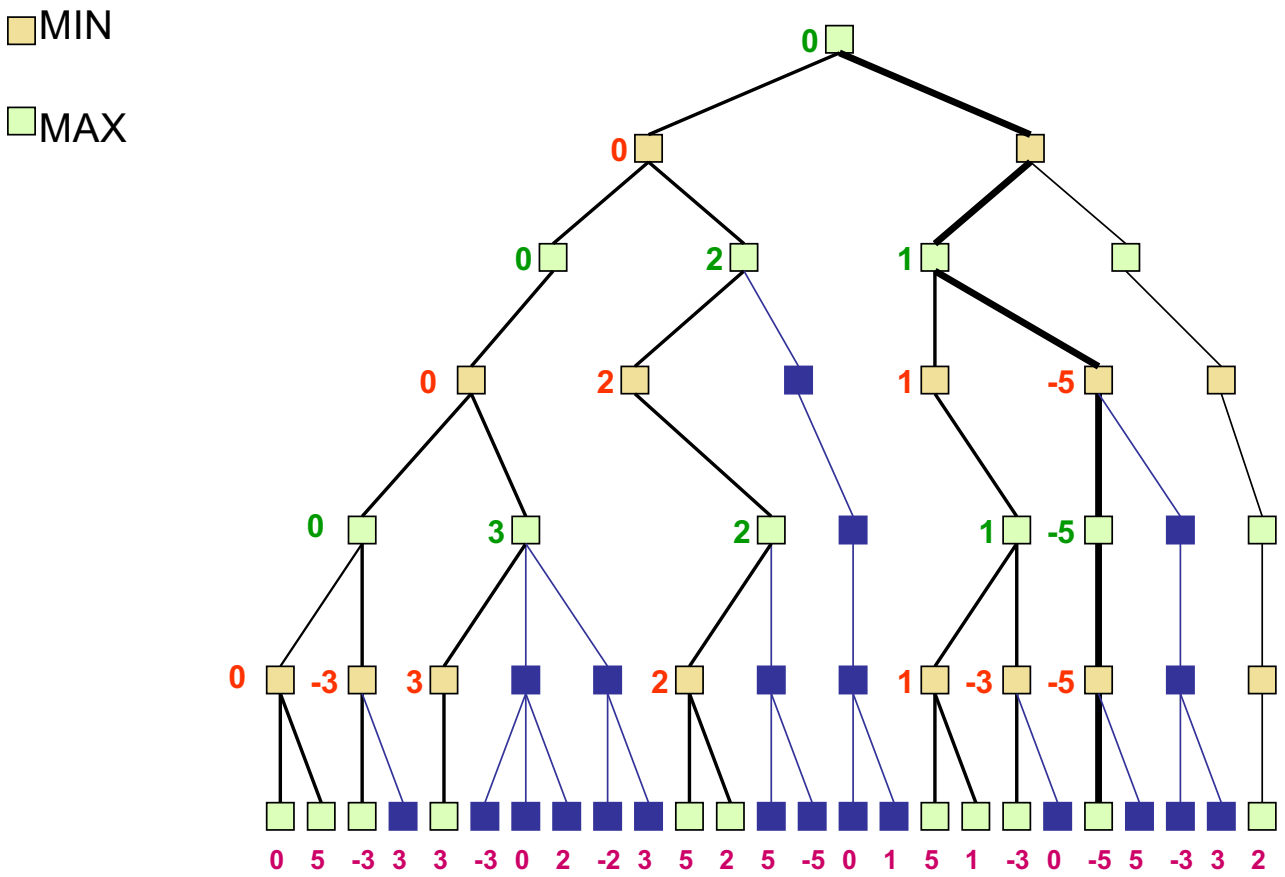
Example: α – β pruning



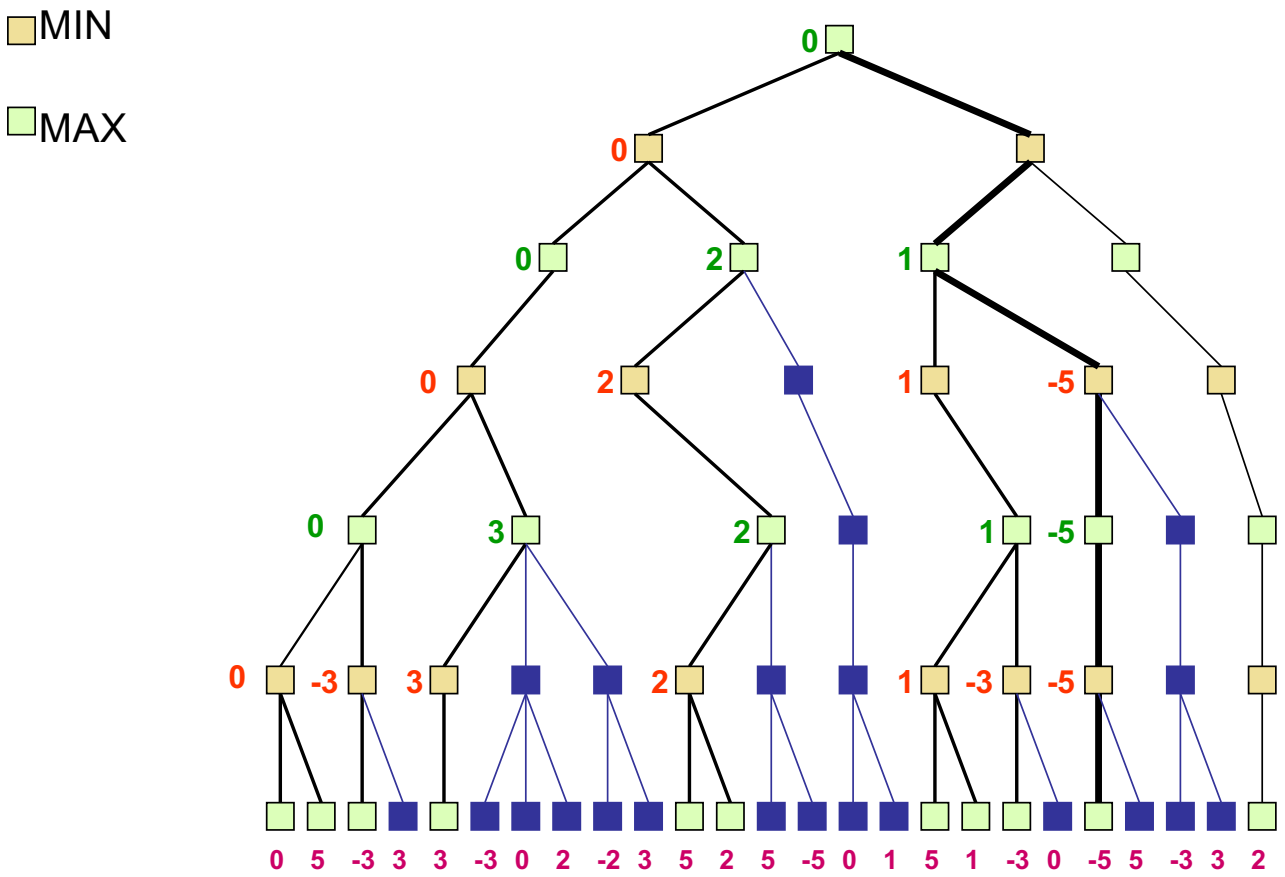
Example: α – β pruning



Example: α – β pruning

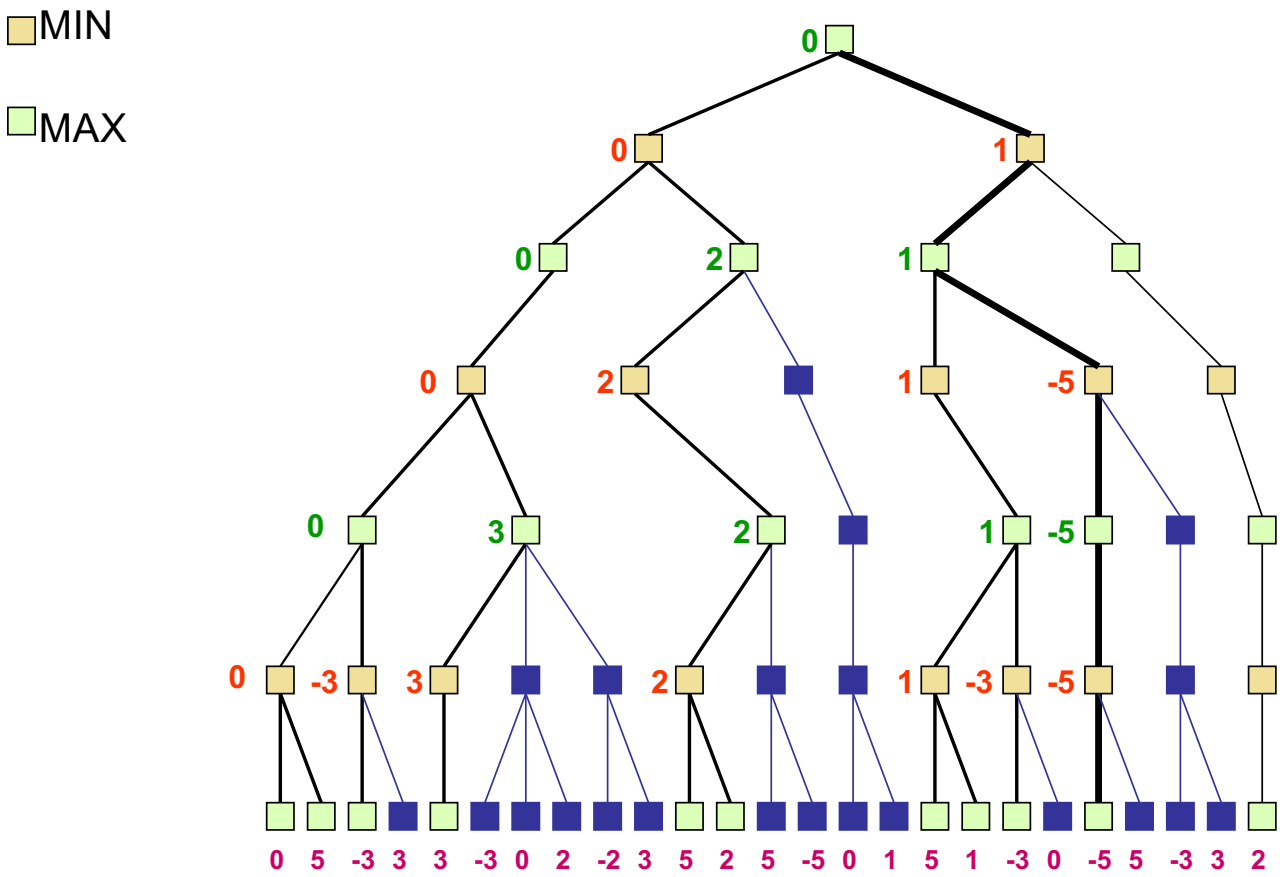


Example: α – β pruning

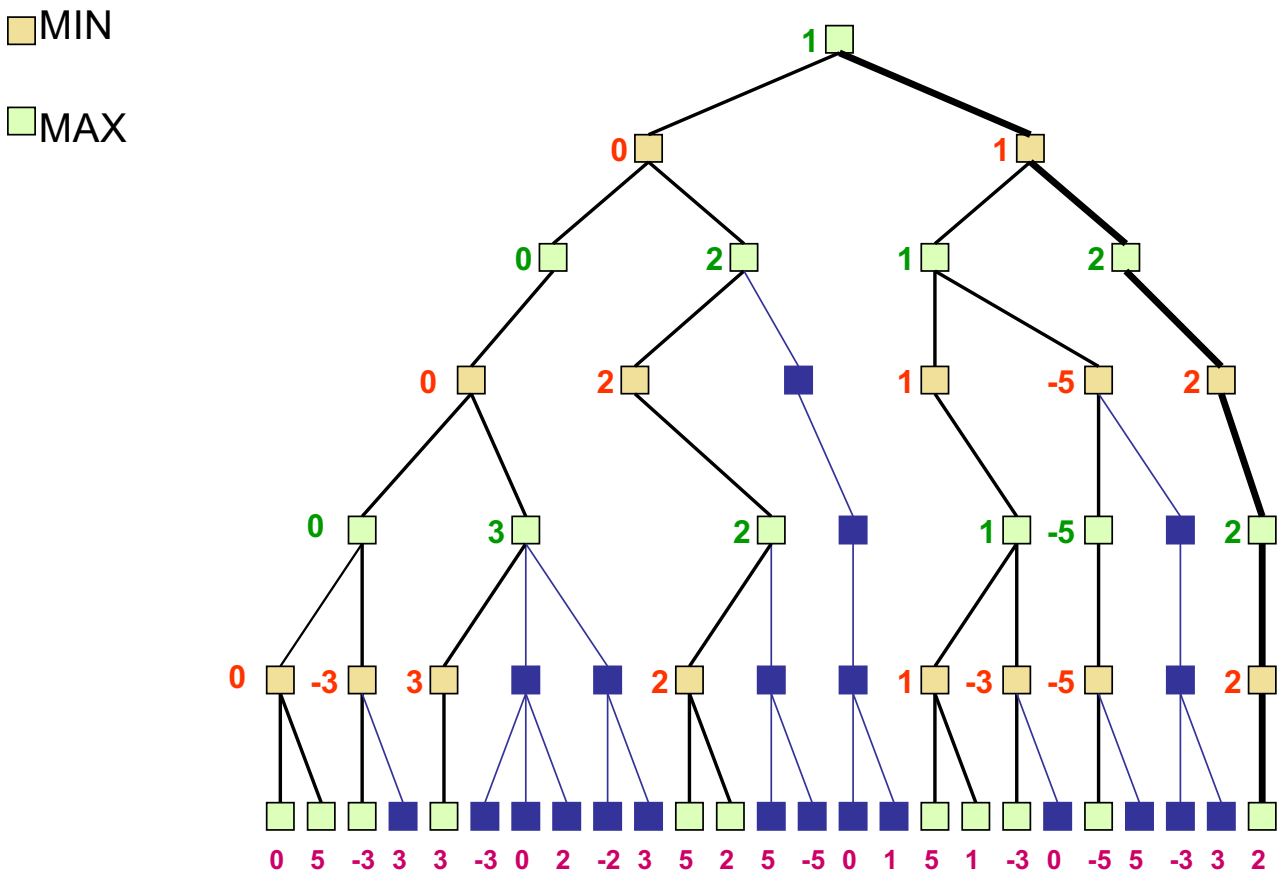


- Prune the sub-tree with a MAX node at its root for which the α value is greater or equal than any β value in its MIN node ancestors
- Prune the sub-tree with a MIN node at its root for which the β value is less or equal than any α value in its MAX node ancestors

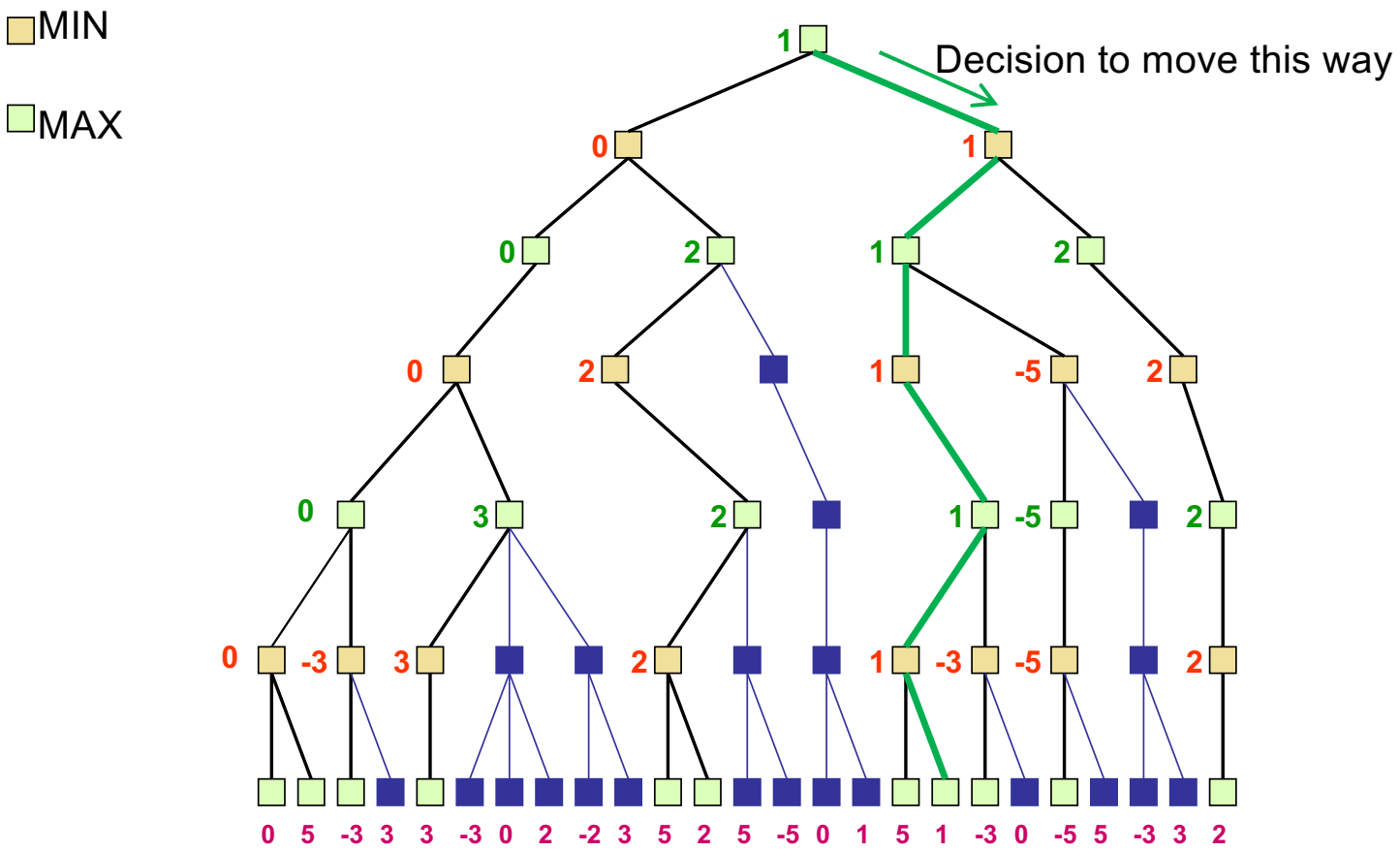
Example: α – β pruning



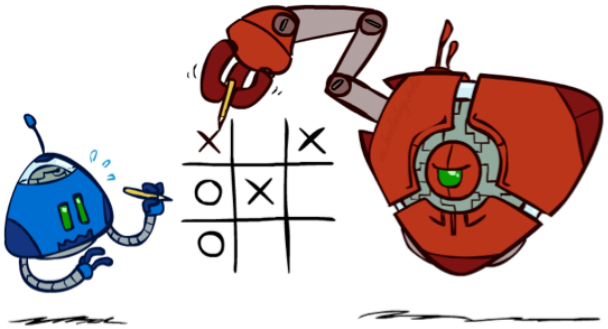
Example: α – β pruning

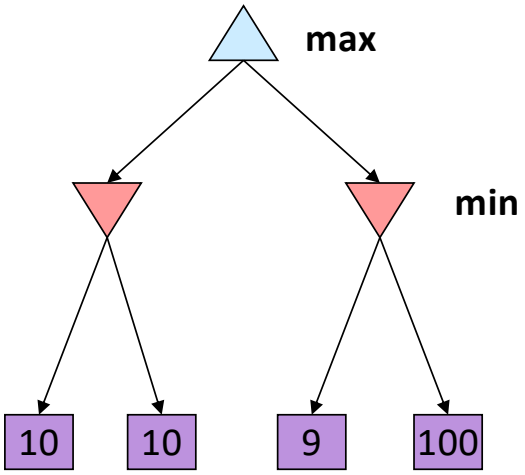


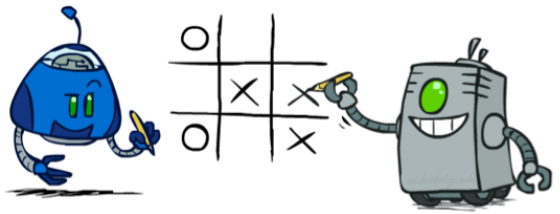
Example: α – β pruning



Minimax Properties







Optimal against a perfect player. Otherwise?

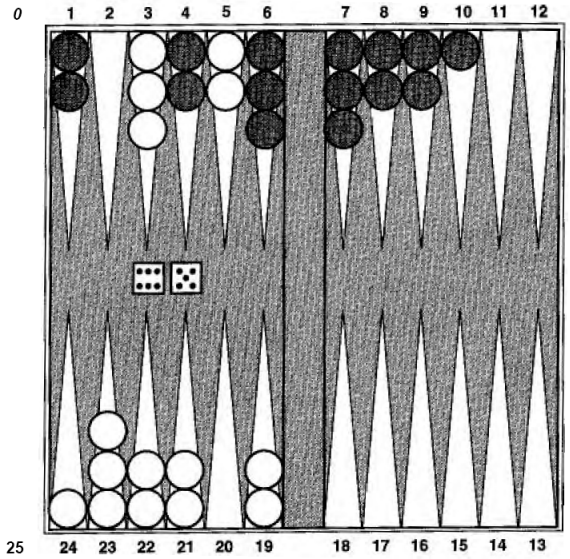
How to play games with uncertainty?

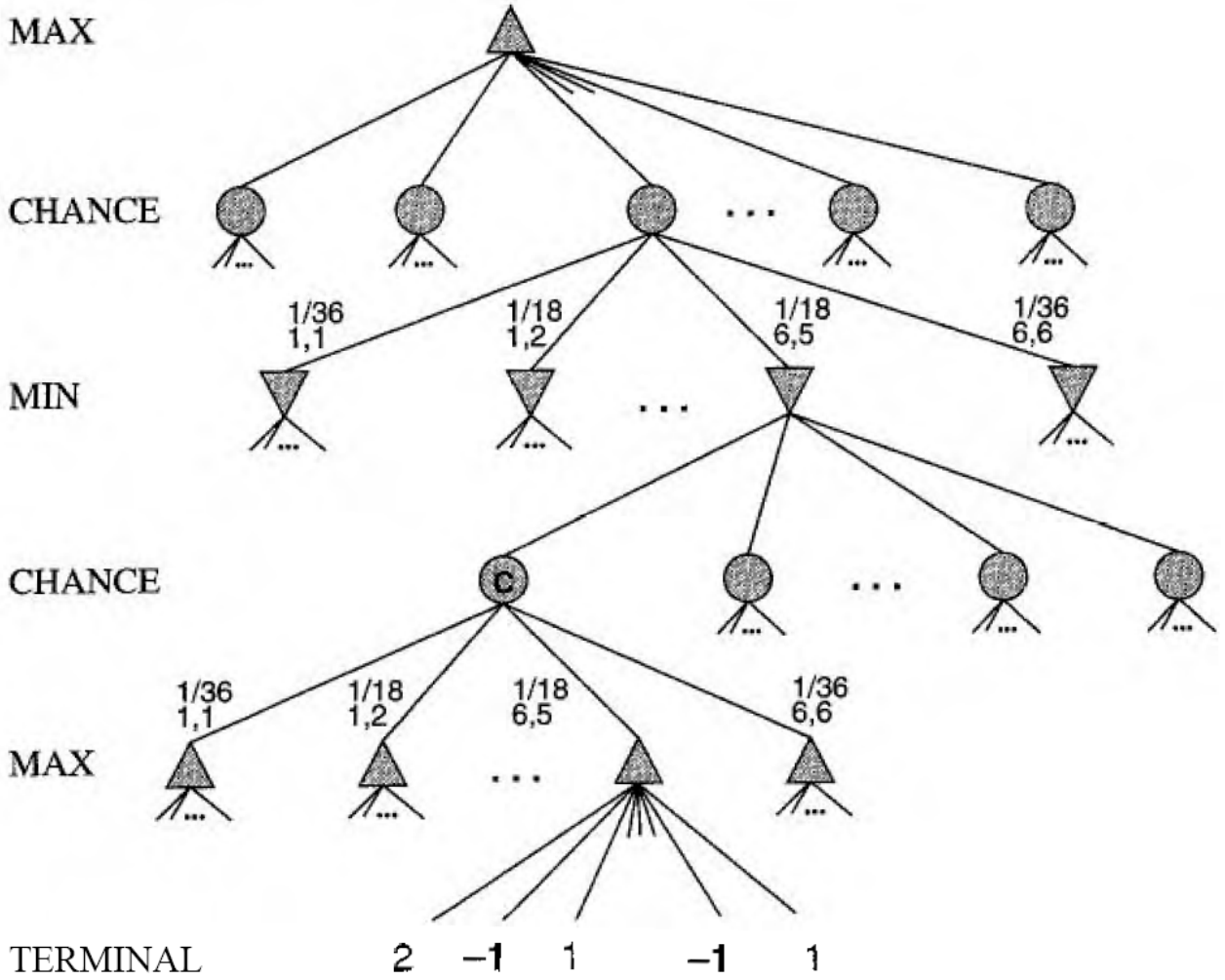
- Card games
- We introduce reasoning for the current and next states for each of the players for which is believed to be the most expected
- This means the node includes average probability that the situation (node) will occur

How to play games which include chance?

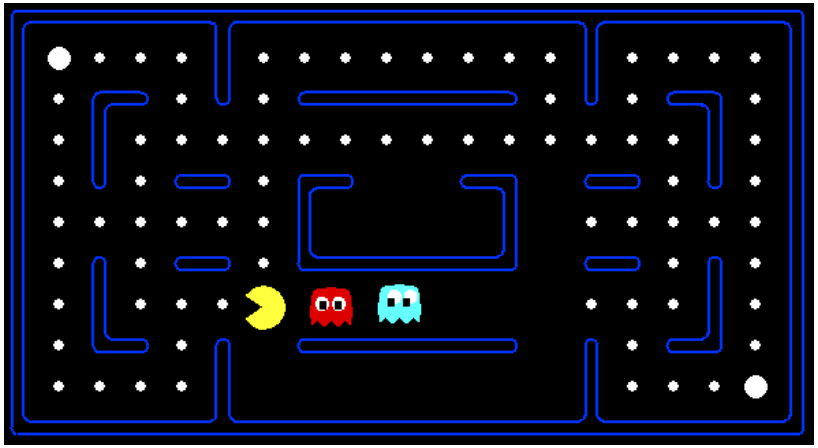
- Dice games
- The strategy remains unchanged, but when defining the evaluation function, we include the chance factor
- That means that between the Min and Max nodes we introduce an element of uncertainty, and then the average expected value is determined

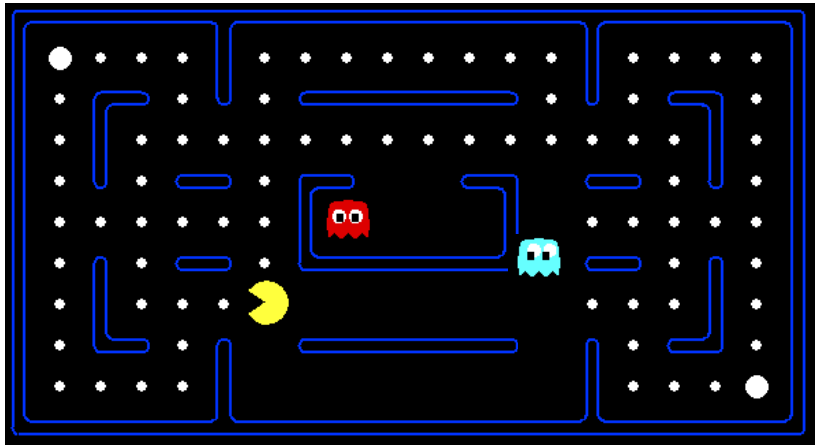
Game including chance

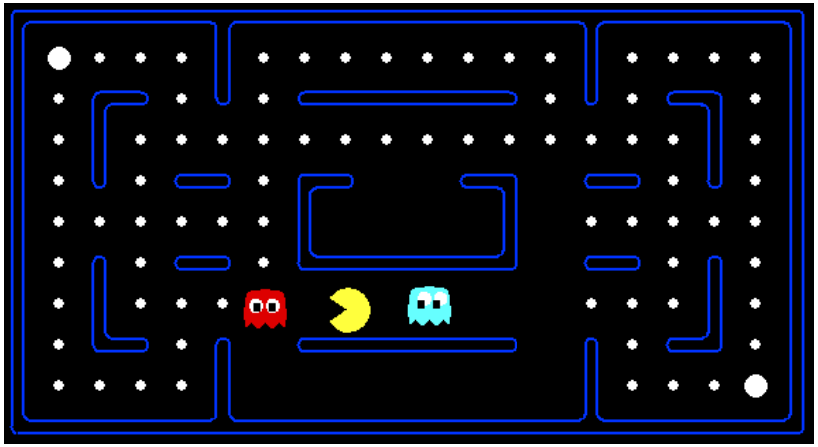


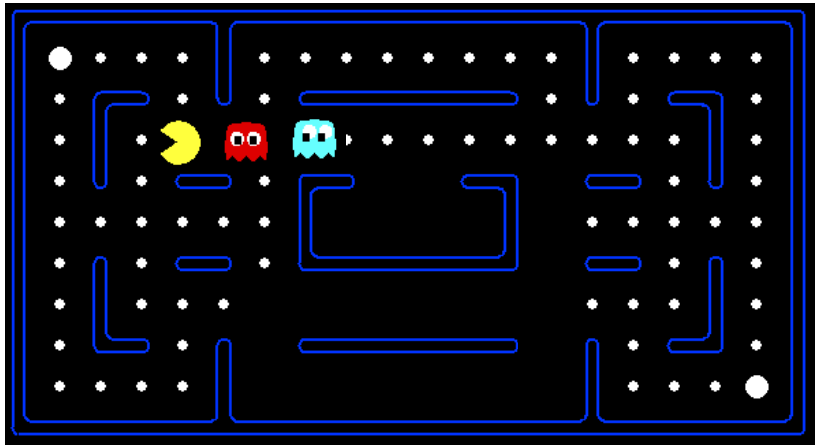


Another game including chance

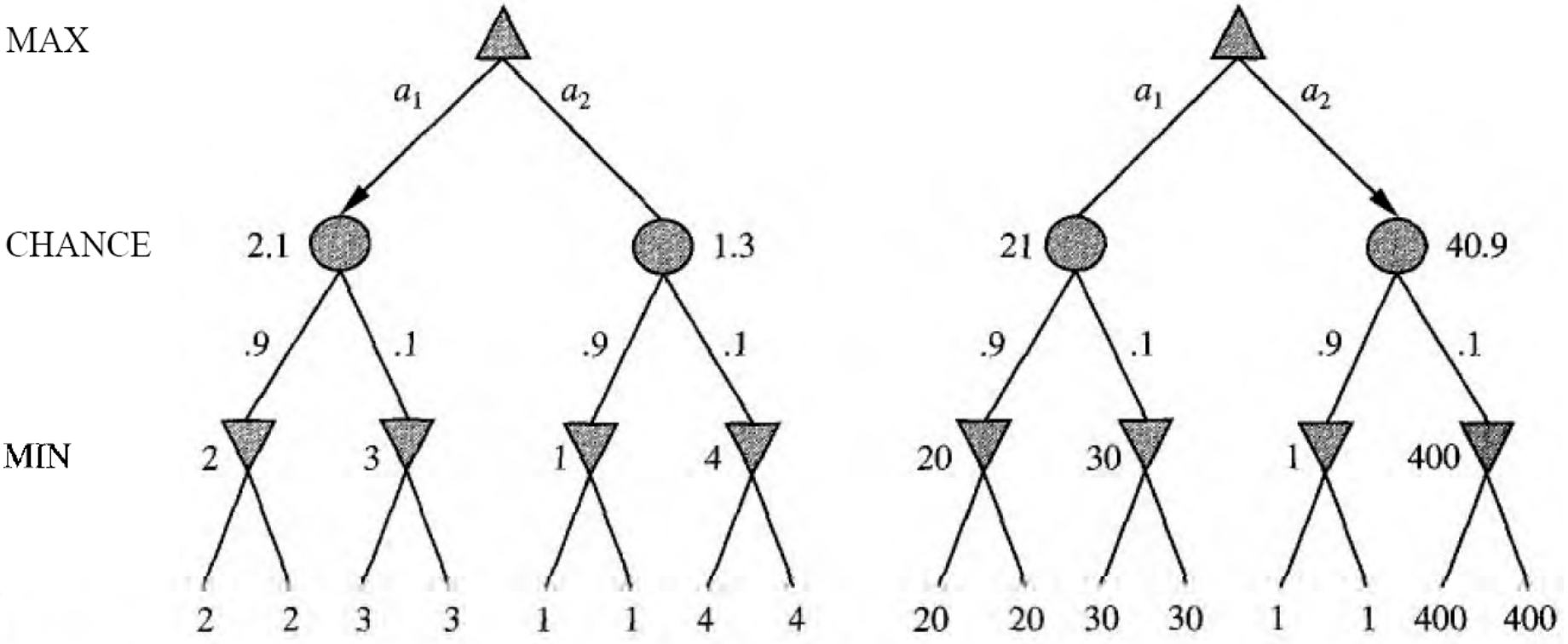




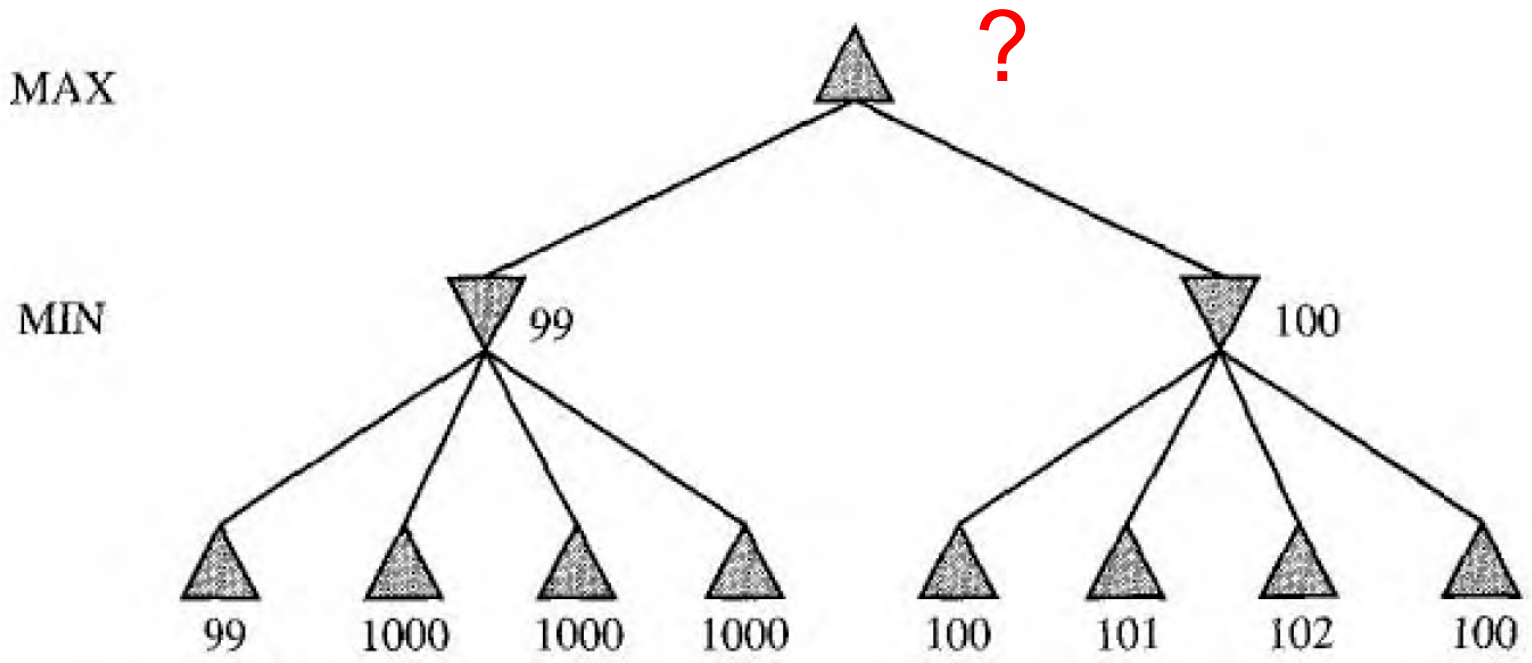




In games including chance the evaluation function value is important



But also in MINIMAX



We can include choice probabilities and hope for a mistake by the opponent

Expected value for minimax in games including chance

For $s \in$ Next_moves(n)

Expectiminimax(n)=

$$\begin{cases} \text{Utility(n), if n is a terminal state} \\ \max \text{ Expectiminimax(s), if n is MAX node} \\ \min \text{ Expectiminimax(s), if n is MIN node} \\ \sum P(s) \cdot \text{Expectiminimax(s), if n is a chance node} \end{cases}$$

How to define Utility(n)?

- Utility function determination is a complex problem and various utility functions are used also in business sector, for example in applications in systems for crediting, investment, insurance etc.

Bibliography

- Artificial Intelligence, A Modern Approach 2nd edition, Russel and Norvig
- Artificial Intelligence, A New Synthesis, Nils J. Nilsson

Questions?