Adversarial Search

Games

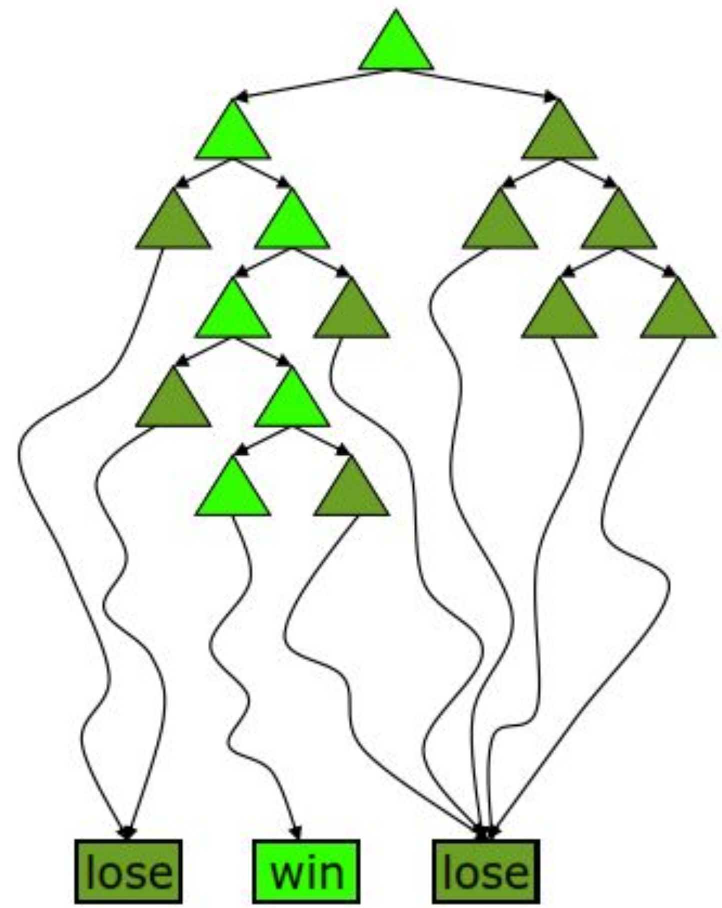
- Multi agent environments : any given agent will need to consider the actions of other agents and how they affect its own welfare.
- The unpredictability of these other agents can introduce many possible contingencies
- There could be *competitive* or *cooperative* environments
- Competitive environments, in which the agent's goals are in conflict require adversarial search – these problems are called as games

Games vs. search problems

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

Deterministic Single-Player?

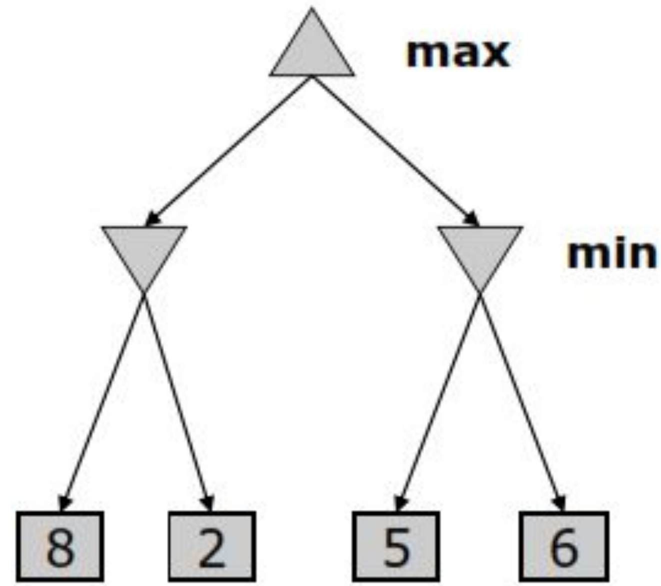
- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- **… it's just search!**
- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don't have path sums as before (utilities at end)
- After search, can pick move that leads to best node



Slide adapted from Macskassy

Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result
- Minimax search
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves
  - Choose move to position with highest minimax value = best achievable utility against best play



Two-player Games

- A game formulated as a search problem:

  - Initial state: board position and turn
  - Operators: definition of legal moves
  - Terminal state: conditions for when game is over
  - Utility function: a <u>numeric</u> value that describes the outcome of the game. E.g., -1, 0, 1 for loss, draw, win. (AKA **payoff function**)

Slide adapted from Macskassy

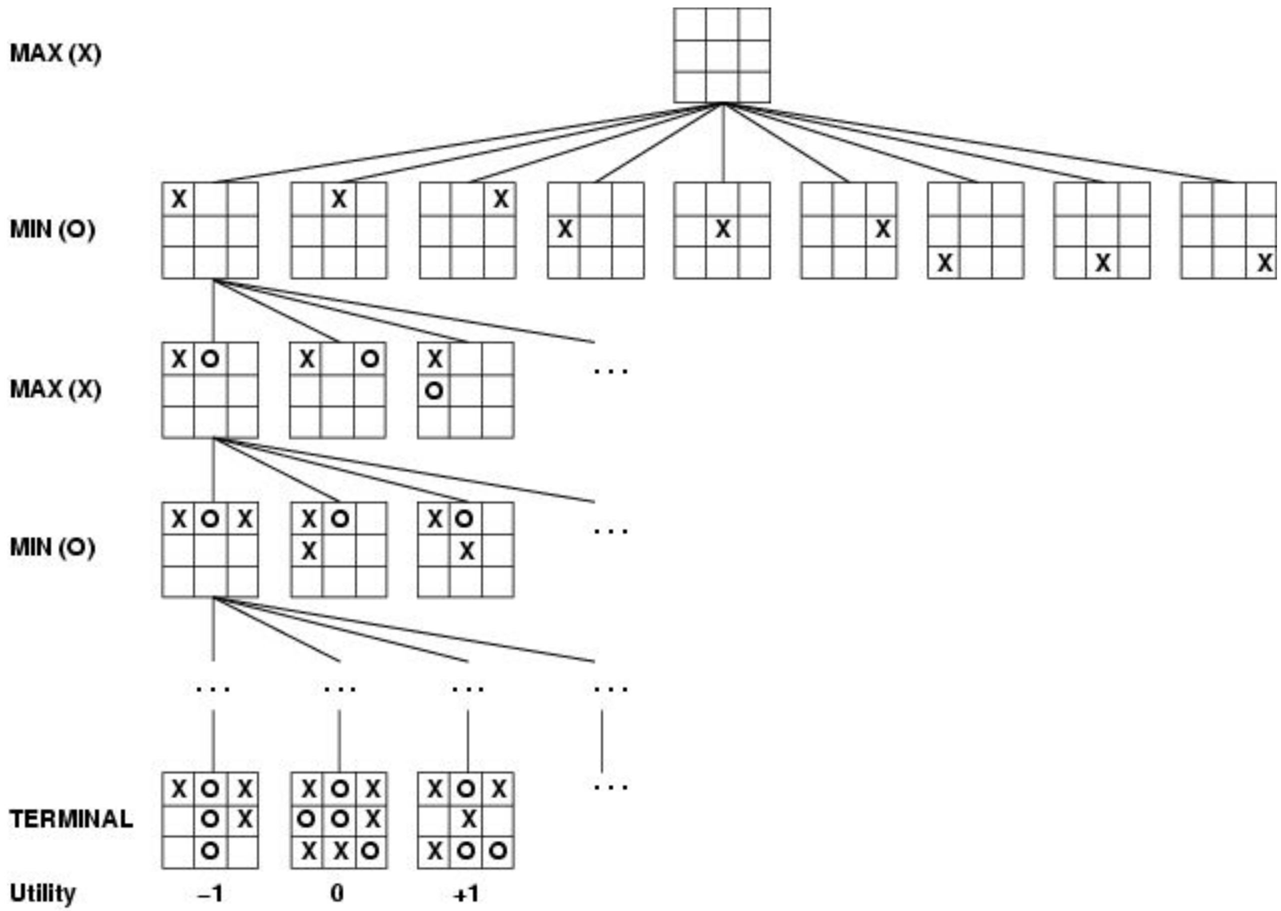
Game search

- Game-playing programs developed by AI researchers since the beginning of the modern AI era
  - Programs playing chess, checkers, etc (1950s)
- Specifics:
  - Sequences of player's decisions we control
  - Decisions of other player(s) we do not control
- Contingency problem: many possible opponent's moves must be "covered" by the solution
- Opponent's behavior introduces uncertainty
- Rational opponent – maximizes its own utility (payoff) function

Game Search Problem

- Problem formulation
  - Initial state: initial board position + whose move it is
  - Operators: legal moves a player can make
  - Goal (terminal test): game over?
  - Utility (payoff) function: measures the outcome of the game and its desirability
- Search objective:
  - Find the sequence of player's decisions (moves) maximizing its utility (payoff)
  - Consider the opponent's moves and their utility

Game tree (2-player, deterministic, turns)



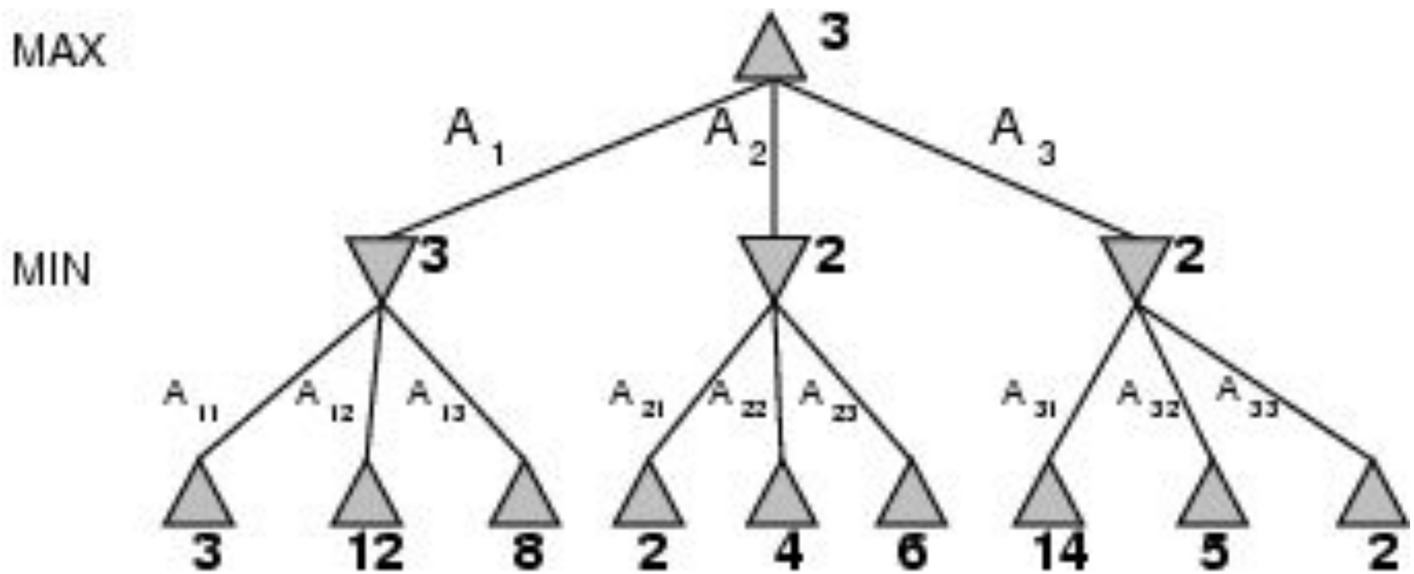
Minimax Algorithm

- How to deal with the contingency problem?
  - Assuming the opponent is always rational and always optimizes its behavior (opposite to us), we consider the best opponent's response
  - Then the minimax algorithm determines the best move

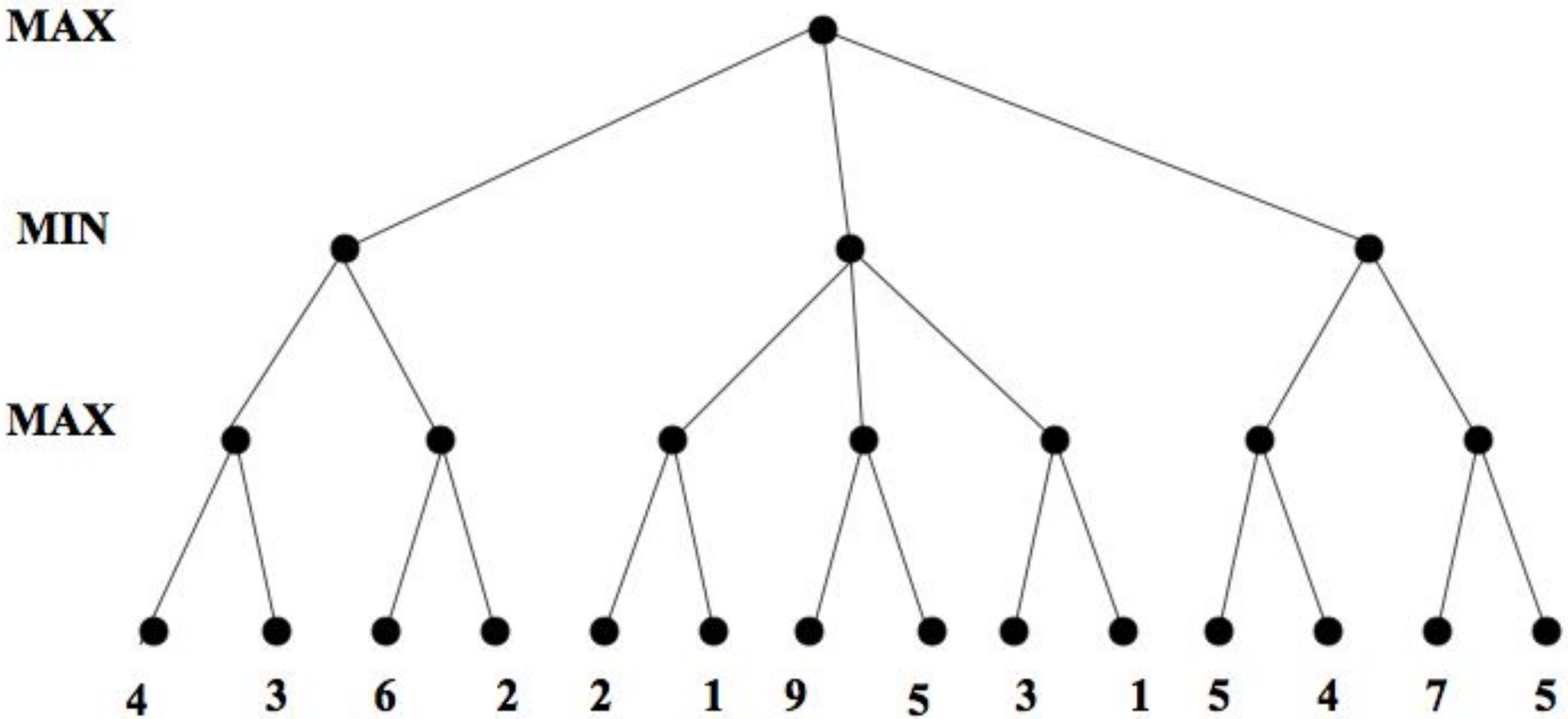
The agent doesn't know what effect its actions will have

Minimax

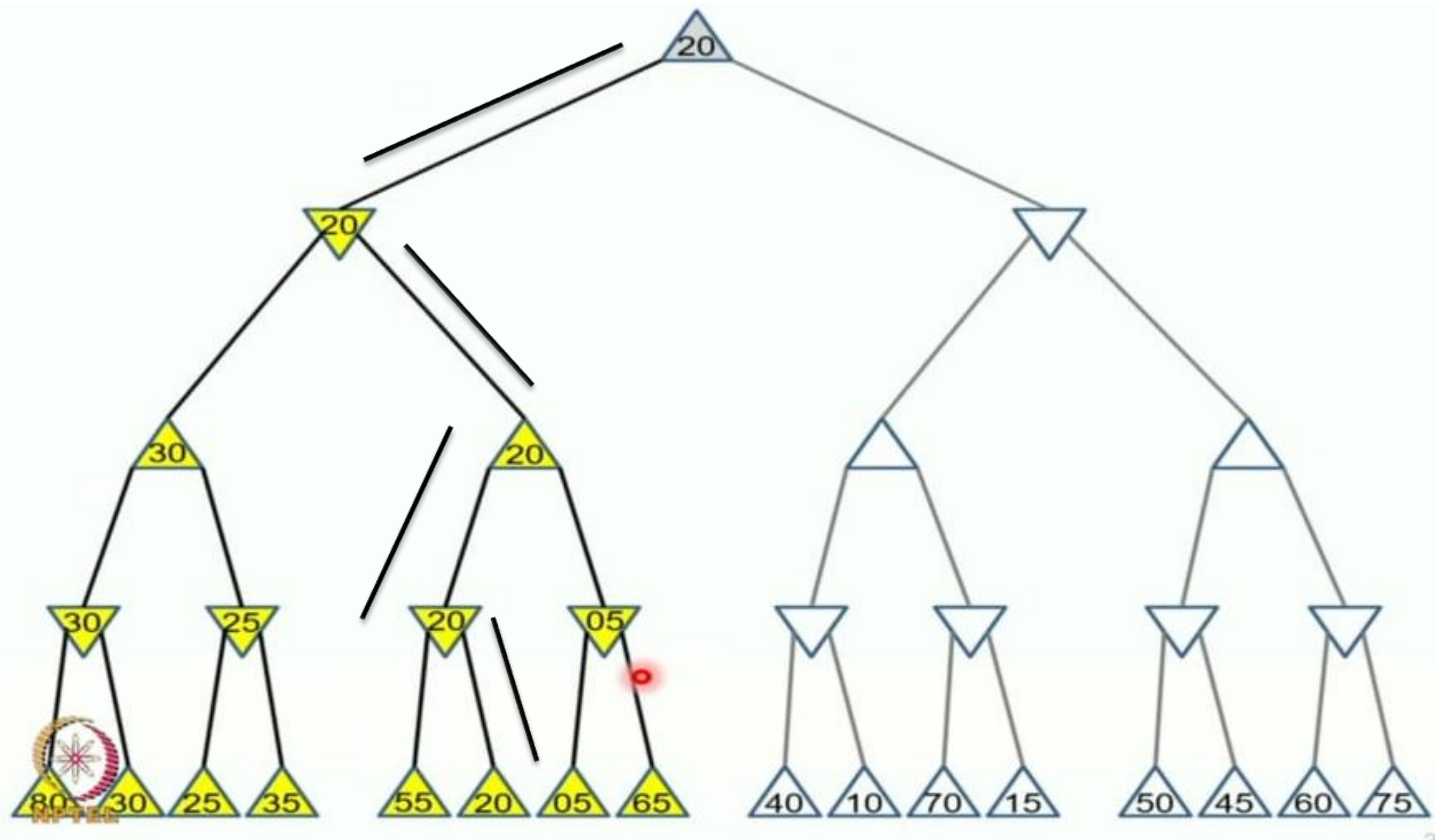
- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game: [will go through another eg in lecture]



Minimax. Example 1



Ex 2



```
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
```

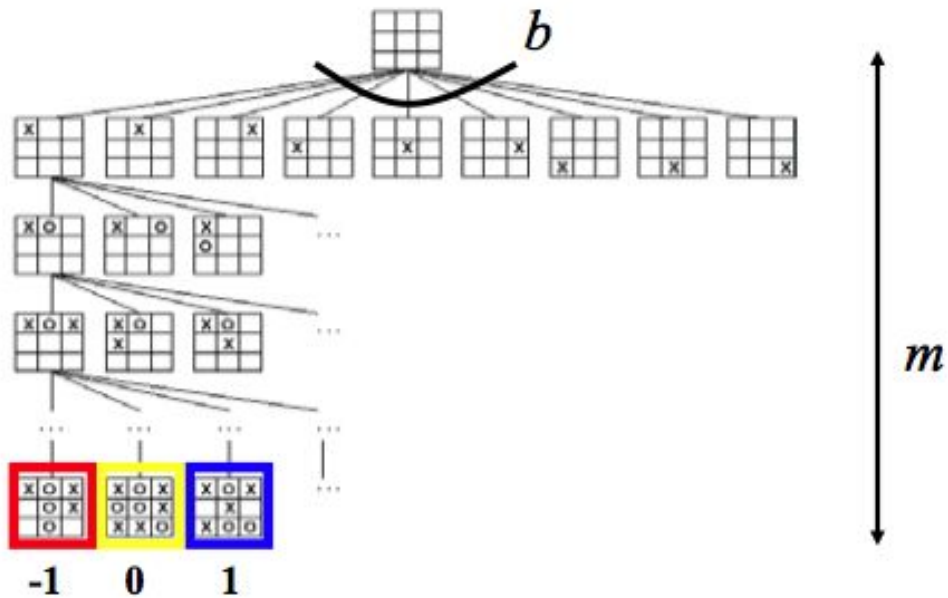
```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

```
function MINIMAX-DECISION(game) returns an operator

  for each op in OPERATORS[game] do
      VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
  end
  return the op with the highest VALUE[op]
```

```
function MINIMAX-VALUE(state, game) returns a utility value

  if TERMINAL-TEST[game](state) then
      return UTILITY[game](state)
  else if MAX is to move in state then
      return the highest MINIMAX-VALUE of SUCCESSORS(state)
  else
      return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Complexity of the minimax algorithm

- We need to explore the complete game tree before making the decision



Complexity:

$O(b^m)$

- Impossible for large games
  - Chess: 35 operators, game can have 50 or more moves

Solution to the complexity problem

Two solutions:

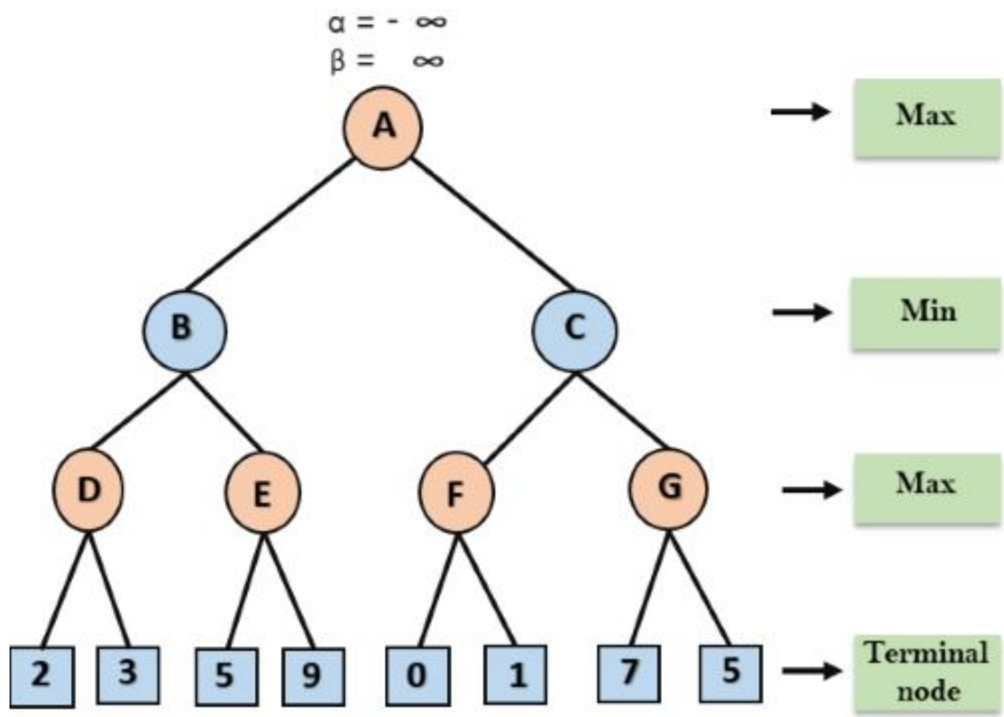
1. **Dynamic pruning of redundant branches** of the search tree
   - identify a provably suboptimal branch of the search tree before it is fully explored
   - Eliminate the suboptimal branch

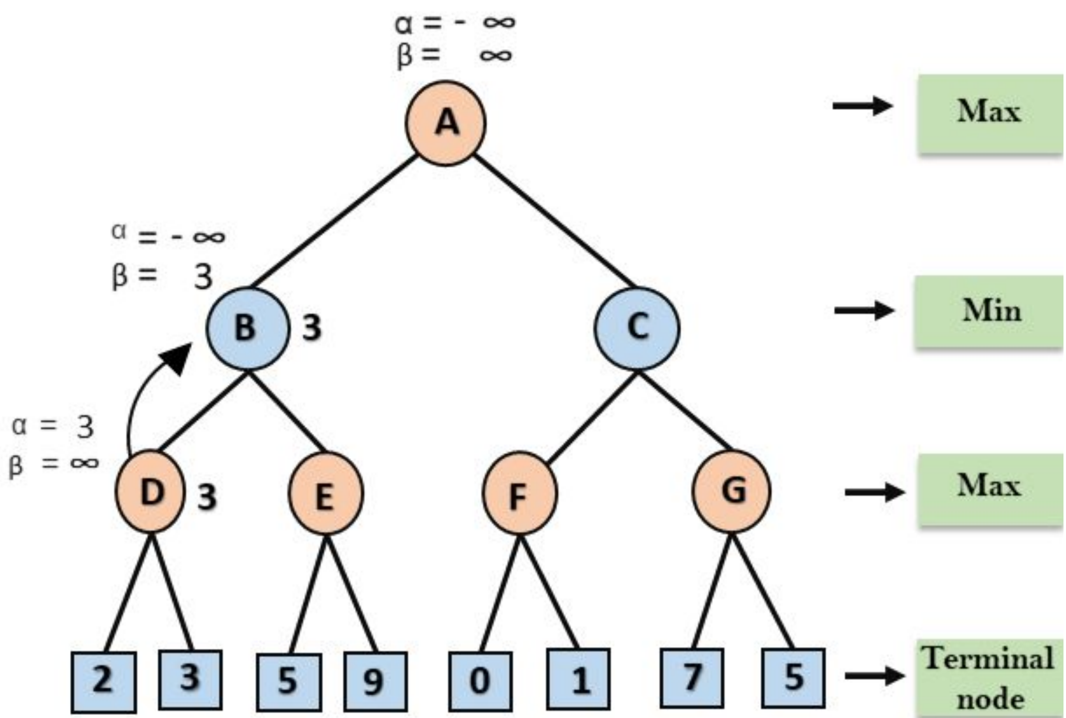
   Procedure: Alpha-Beta pruning

2. **Early cutoff of the search tree**
   - uses imperfect minimax value estimate of non-terminal states (positions)

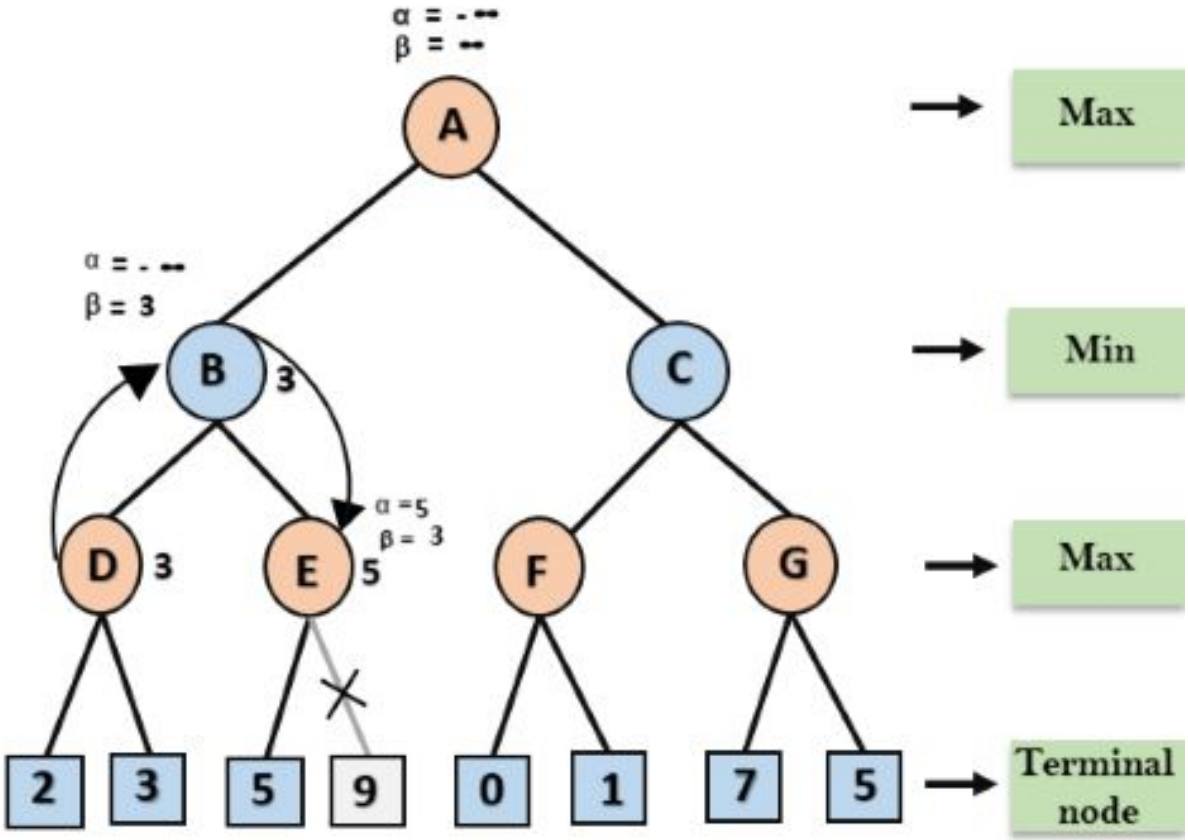
Alpha Beta Pruning

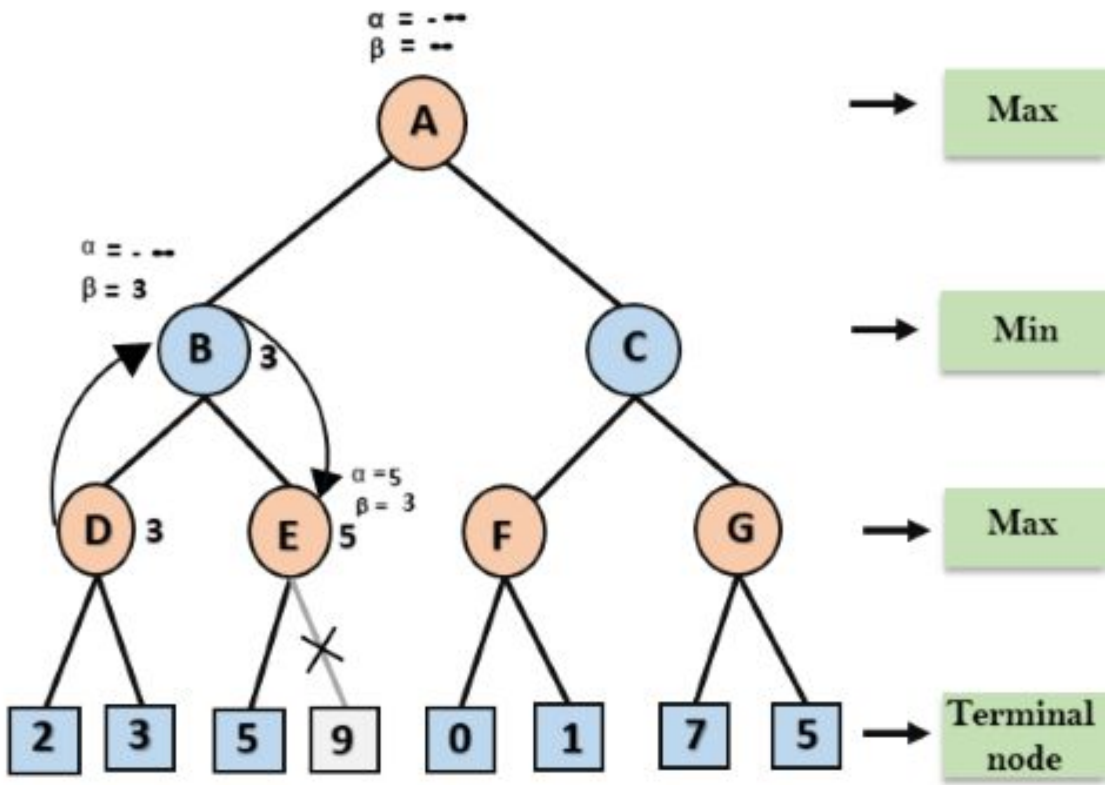
- Some branches will never be played by rational players since they include sub-optimal decisions for either player
- First, we will see the idea of Alpha Beta Pruning
- Then, we’ll introduce the algorithm for minimax with alpha beta pruning, and go through the example again, showing the book-keeping it does as it goes along


α = - ∞
β = ∞
A
B
C
D
E
F
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node

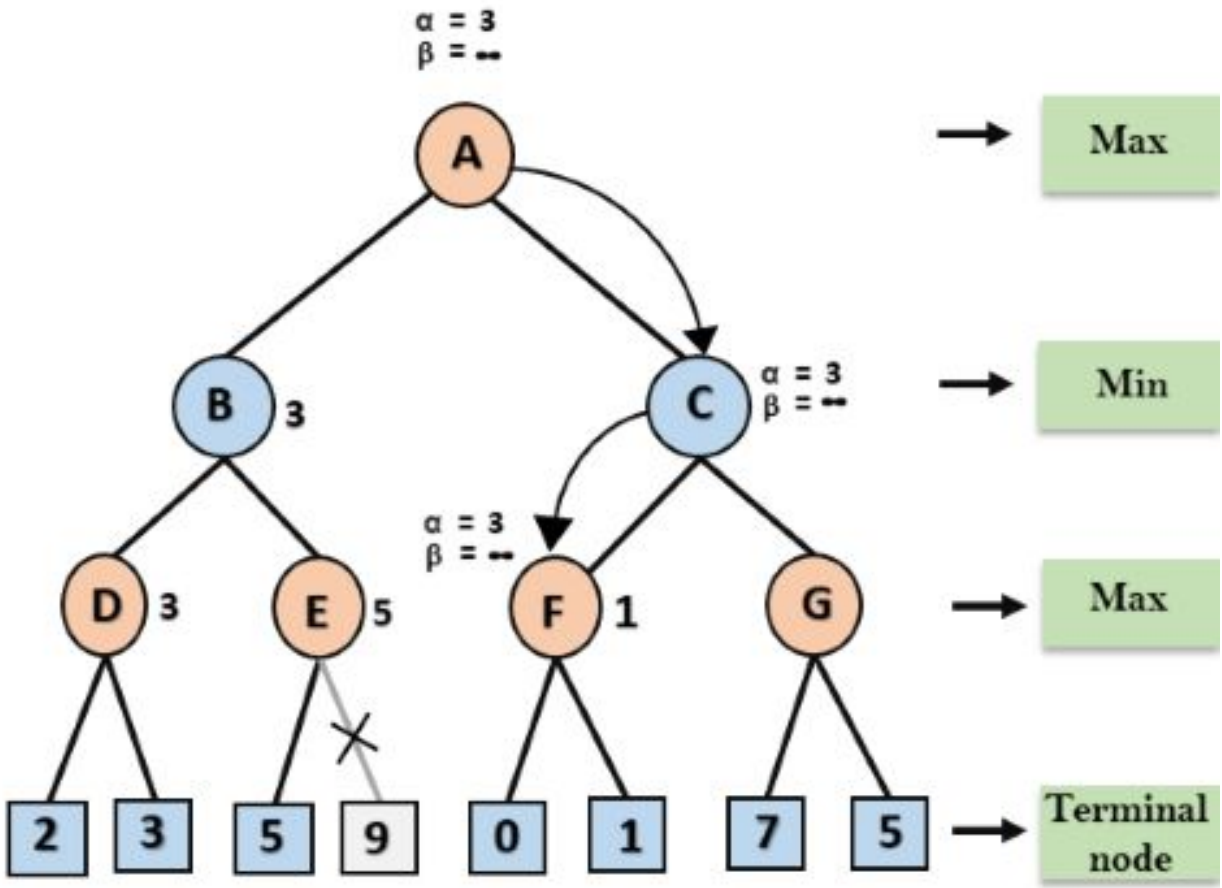


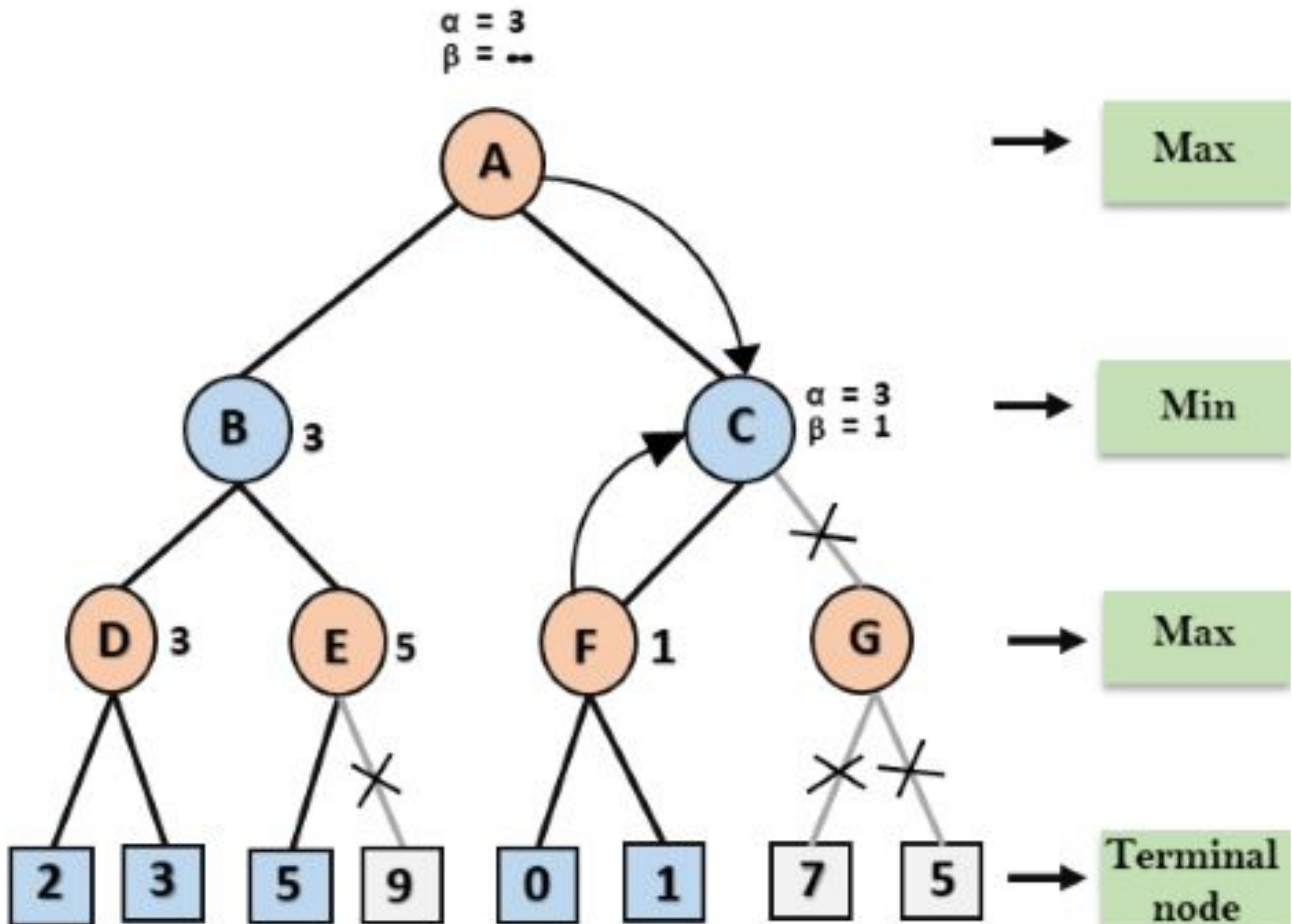
α = - ∞
β = ∞
A
Max
α = - ∞
β = 3
B
3
C
Min
α = 3
β = ∞
D
3
E
F
G
Max
2
3
5
9
0
1
7
5
Terminal node


α = -∞
β = ∞
A
α = -∞
β = 3
B
3
C
D
3
E
5
α =5
β = 3
F
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node

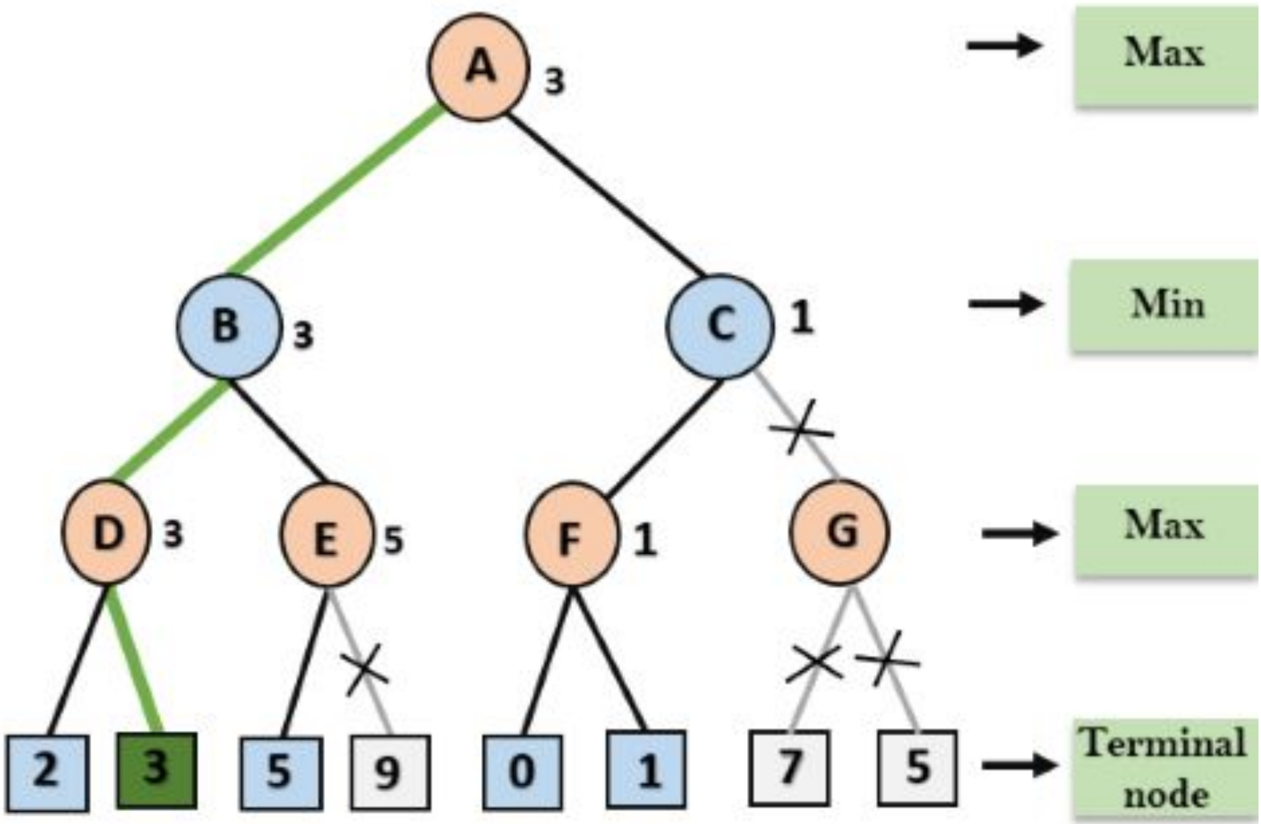


α = -∞
β = ∞
A
Max
α = -∞
β = 3
B
3
C
Min
α = 5
β = 3
D
3
E
5
F
G
Max
2
3
5
9
0
1
7
5
Terminal node


α = 3
β = -∞
A
B
3
C
α = 3
β = -∞
α = 3
β = -∞
D
3
E
5
F
1
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node



α = 3
β = ∞
A
Max
B
3
C
α = 3
β = 1
Min
D
3
E
5
F
1
G
Max
2
3
5
9
0
1
7
5
Terminal node


A 3
B 3
C 1
D 3
E 5
F 1
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node

Move Ordering in Alpha-Beta pruning:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

α-β pruning

- Alpha-beta 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 known to be worse than the current α or β value for MAX or MIN, respectively.

- The effectiveness of alpha-beta pruning is highly dependent on the order in which the successors are examined.

Properties of α-β

- Pruning does not affect final result

- Good move ordering improves effectiveness of pruning

- With "perfect ordering," time complexity = $O(b^{m/2})$
  - doubles depth of search

- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

The α-β algorithm

```
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
```

The α-β algorithm

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*)
  $v \leftarrow +\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Properties of α-β

- Pruning does not affect final result

- Good move ordering improves effectiveness of pruning

- With "perfect ordering," time complexity = $O(b^{m/2})$
   - doubles depth of search

- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)