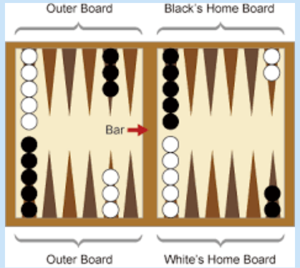
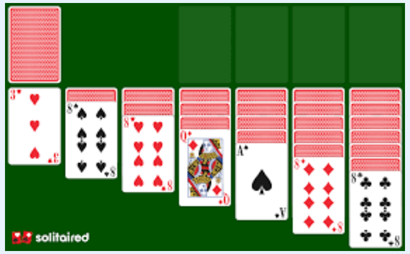
Adversarial Search And Games

الدكتور/ مصطفى شبل

Introduction

Games types

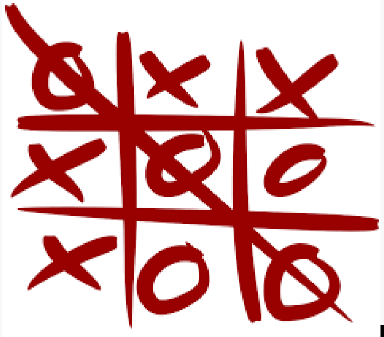
	Deterministic	*Chance* 
Perfect information	*Chess , go*  	*Monopoly , backgammon*  
Imperfect information	*battleship* 	*Solitaire* 

Definitions

- More than one player game
- Players alternate moves
- **Zero-sum**: one player's loss is the other's gain
- **Perfect information**: both players have access to complete information about the state of the game. No information is hidden from either player.
- No chance (e.g., using dice) involved
- Examples: Tic-Tac-Toe, Checkers, Chess,

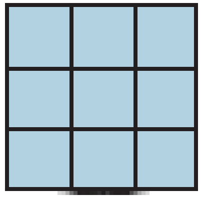
1

2



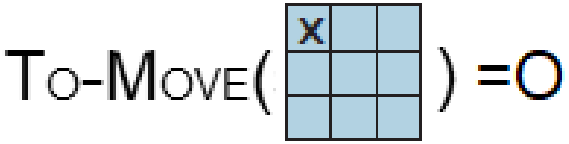
3

Definitions II

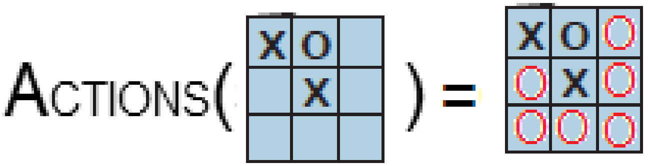


S_0: The initial state, which specifies how the game is set up at the start.

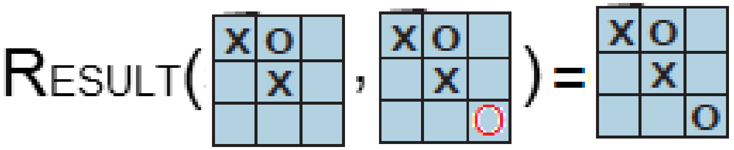
- To-Move(s): The player whose turn it is to move in state s (sometimes called player(s)).



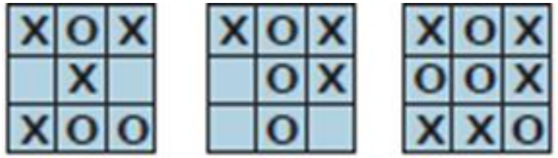
- Actions(s): The set of legal moves in state s.



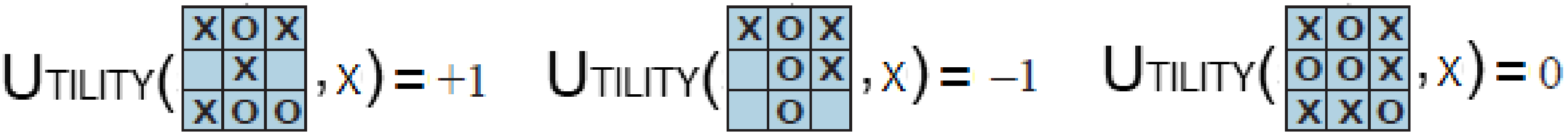
- Result(s, a): The transition model, which defines the state resulting from taking action a in state s.



- Is-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):defines the final numeric value to player p when the game ends in terminal state s.



State Space Graph (Game Tree)

a graph where the vertices are states, the edges are moves and a state might be reached by multiple paths

MAX has 9 possible moves

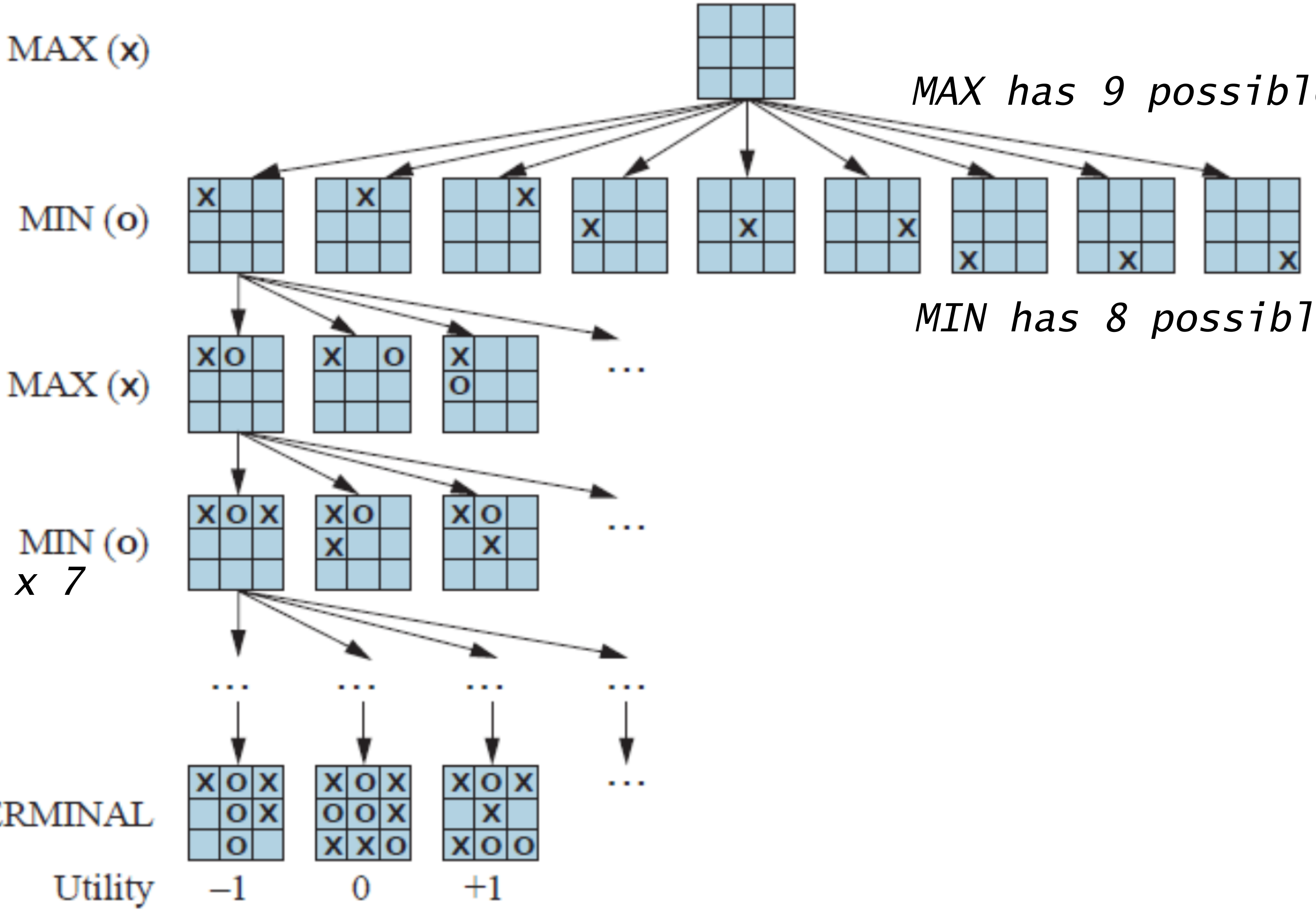
MIN has 8 possible moves

State space= 9

State space= 9 x 8

State space= 9 x 8 x 7

State space= 9!



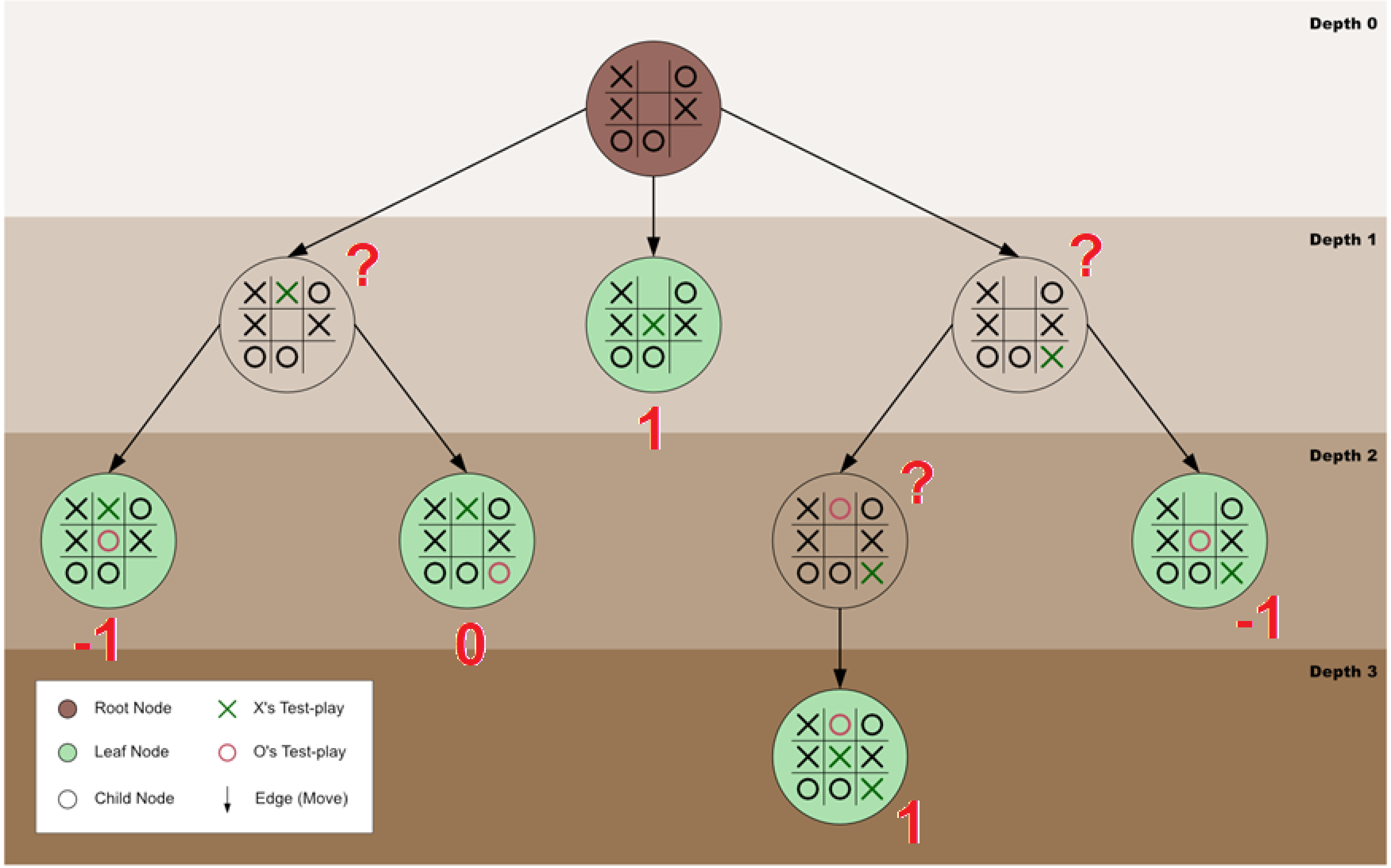
Optimal Decisions in Games

- Search – no adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics techniques can find *optimal* solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

- Games – adversary
  - Solution is strategy
    - strategy specifies move for every possible opponent reply.
  - Time limits force an *approximate* solution
  - Evaluation function: evaluate “goodness” of game position
  - Examples: chess, checkers

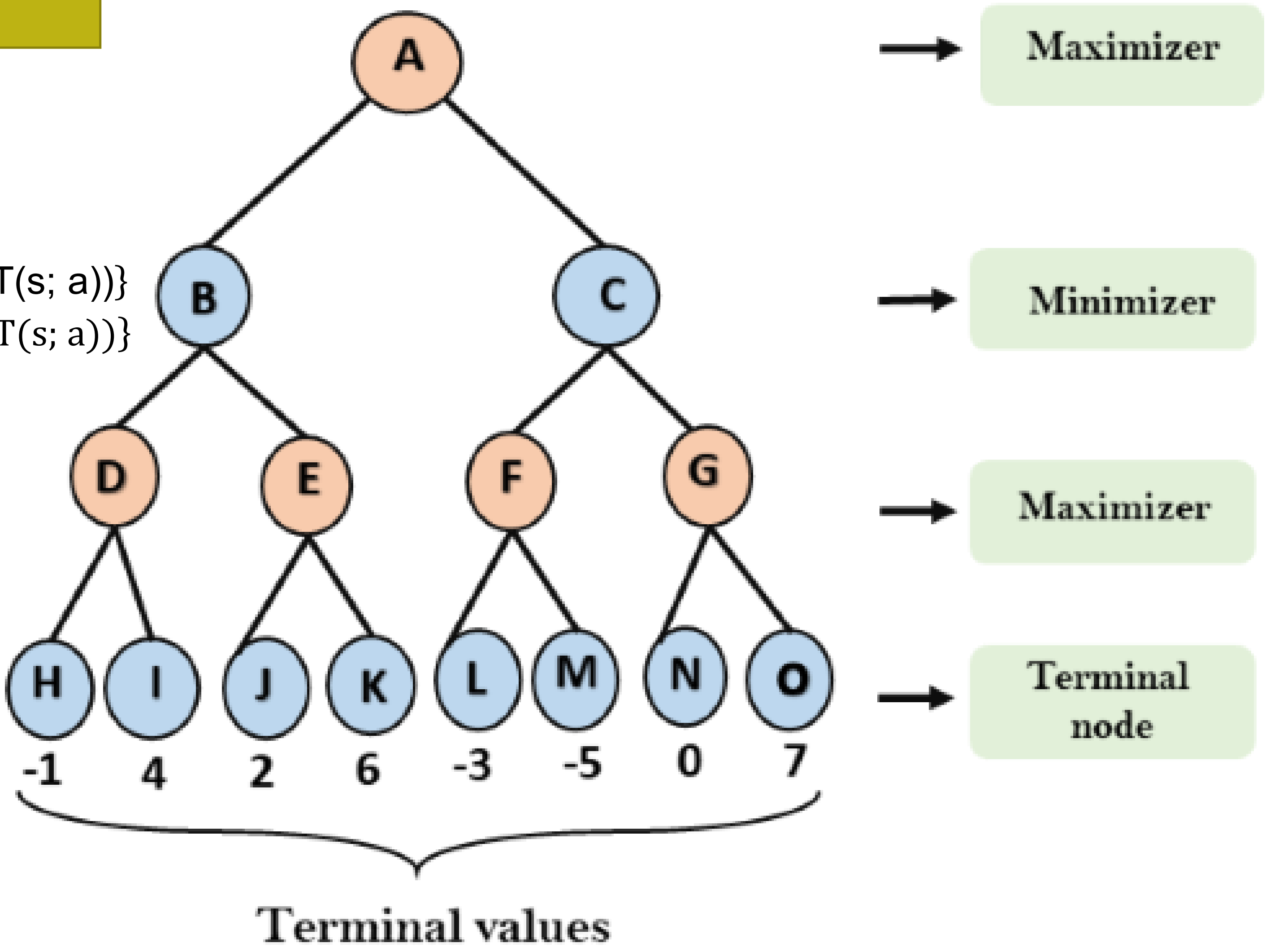
Minimax algorithm (Minimax value)

Given a game tree, the optimal strategy can be determined by working out the minimax value of each state in the tree.



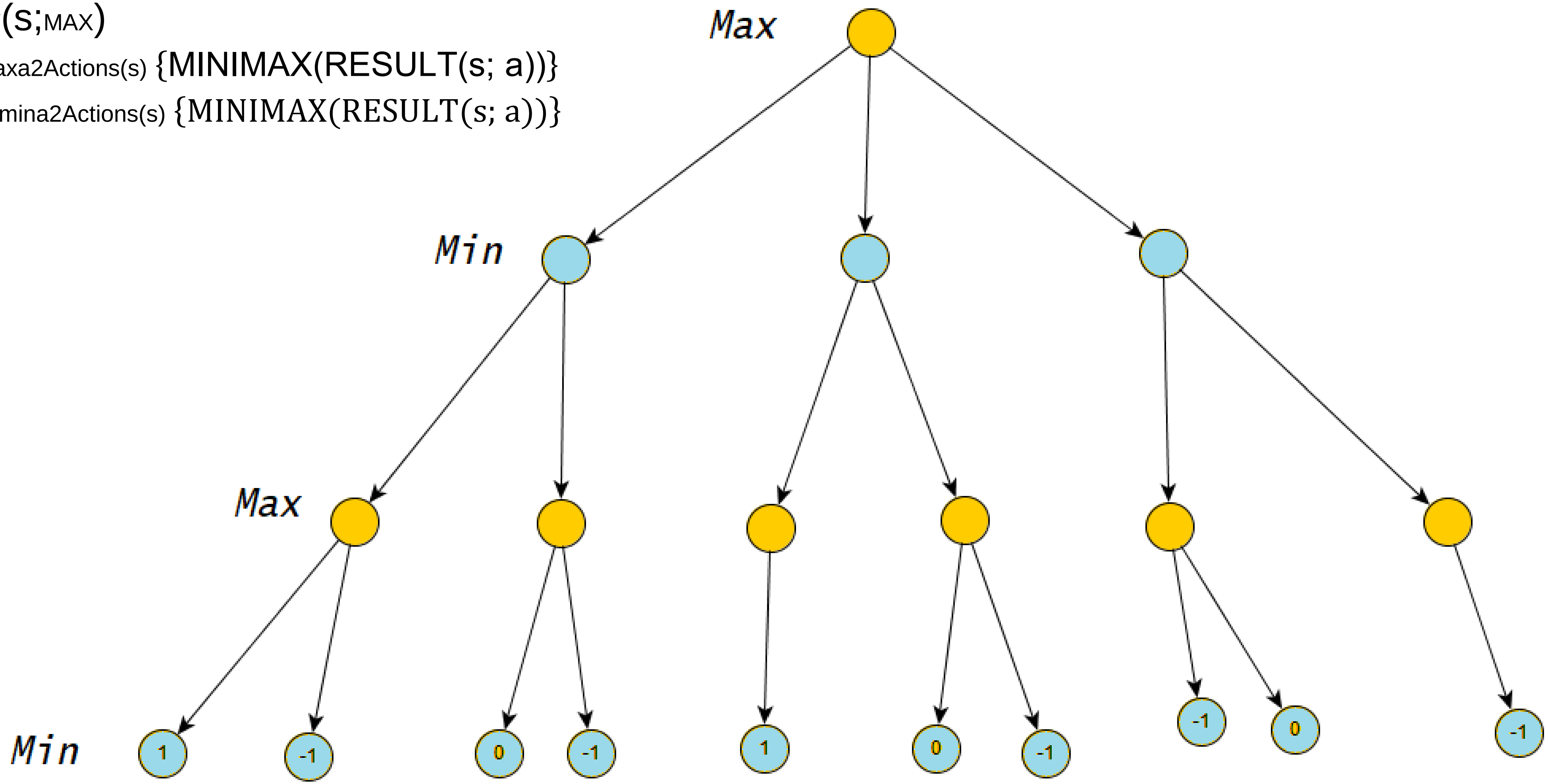
Minimax pseudocode

MINIMAX(s) = 8<
:
if (IS-TERMINAL(s)) UTILITY(s;MAX)
if (TO-MOVE(s)== MAX) maxa2Actions(s) {MINIMAX(RESULT(s; a))}
if (TO-MOVE(s)== min) mina2Actions(s) {MINIMAX(RESULT(s; a))}



Minimax value

MINIMAX(s) = 8<
:
if (IS-TERMINAL(s)) UTILITY(s;MAX)
if (TO-MOVE(s)== MAX) maxa2Actions(s) {MINIMAX(RESULT(s; a))}
if (TO-MOVE(s)== min) mina2Actions(s) {MINIMAX(RESULT(s; a))}



The Minimax Search Algorithm

if (Is-Terminal(s)) Utility(s;max)
if (To-Move(s)== max) maxa2Actions(s) {MINIMAX(RESULT(s; a))}
if (To-Move(s)== min) mina2Actions(s) {MINIMAX(RESULT(s; a))}

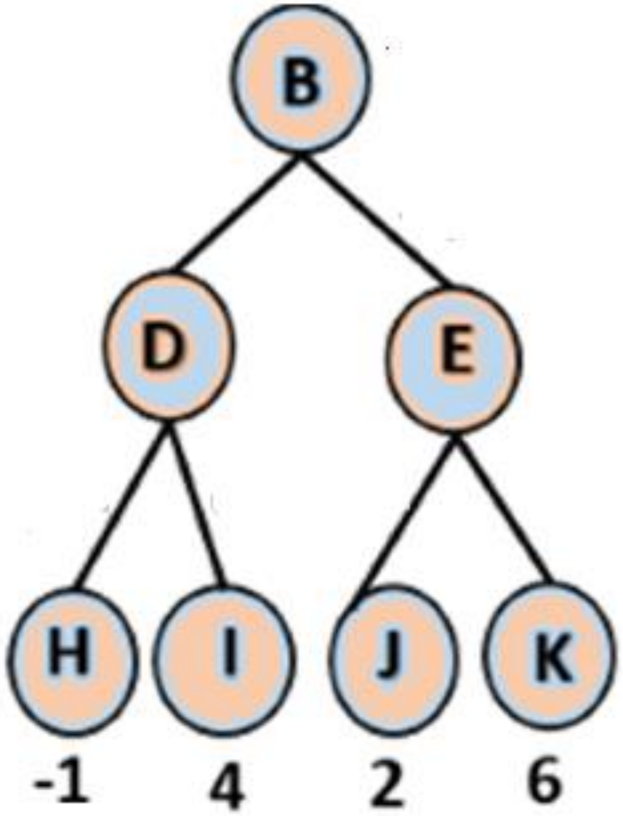
function MINIMAX-DECISION(*state*) **returns** *an action*
  return arg $\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(s, a))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

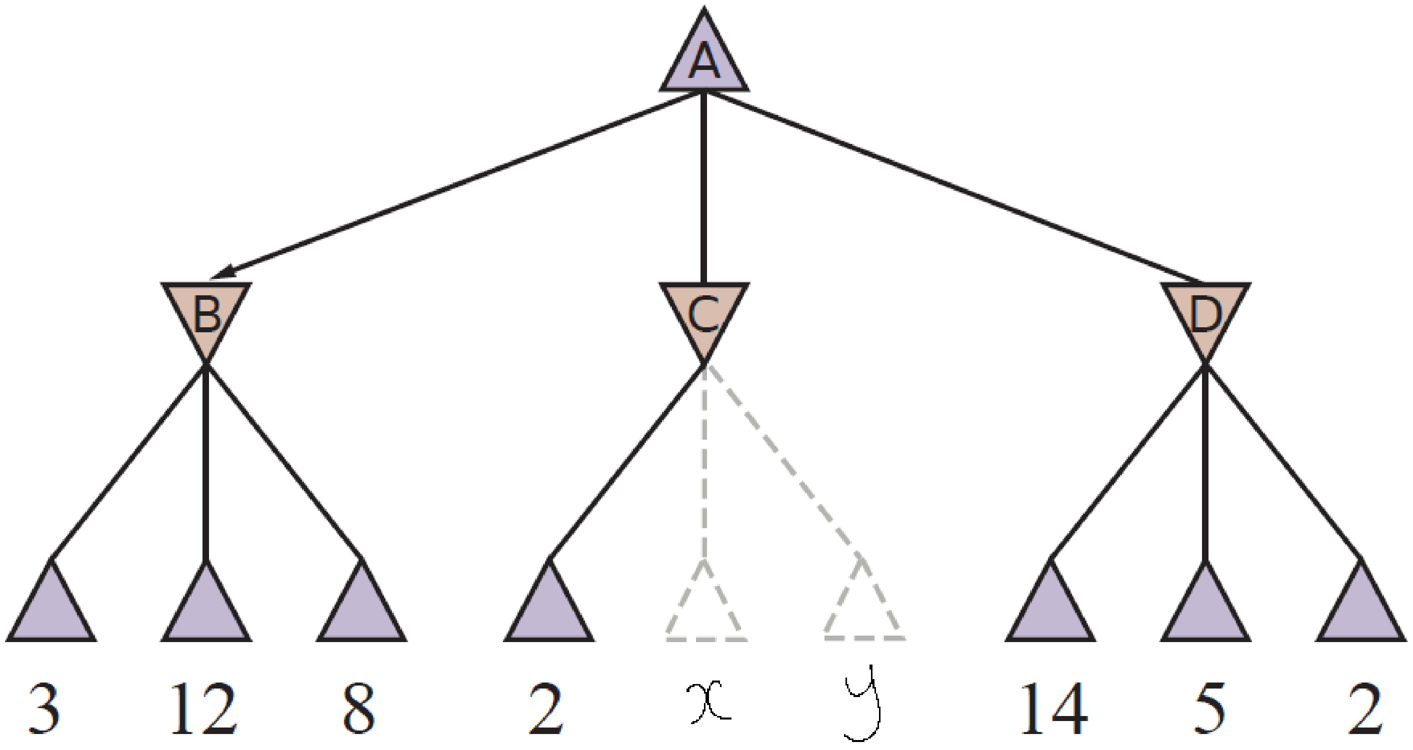
---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v



Processing time is crucial

MINIMAX(root) = max(min(3;12;8);min(2;x;y);min(14;5;2))

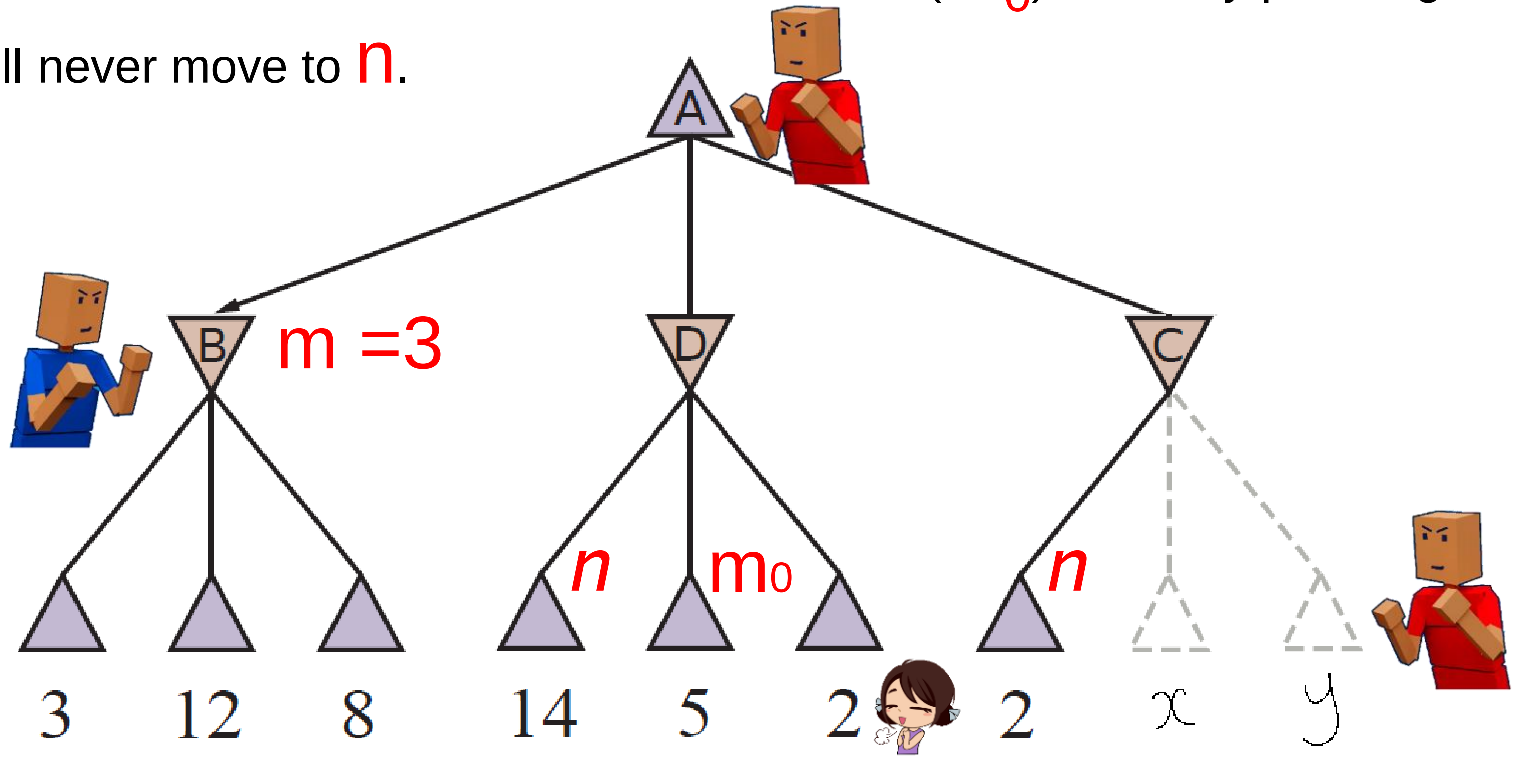


Is the minimax decision are dependent on the values x and y?

Principal of pruning

The general principle is this:

1. consider a node n somewhere in the tree, such that Player has a choice of moving to n.
2. If Player has a better choice either at the same level (m_0) or at any point higher up in the tree (m), then Player will never move to n.



Which node we have to calculate ??

Alpha-Beta Pruning

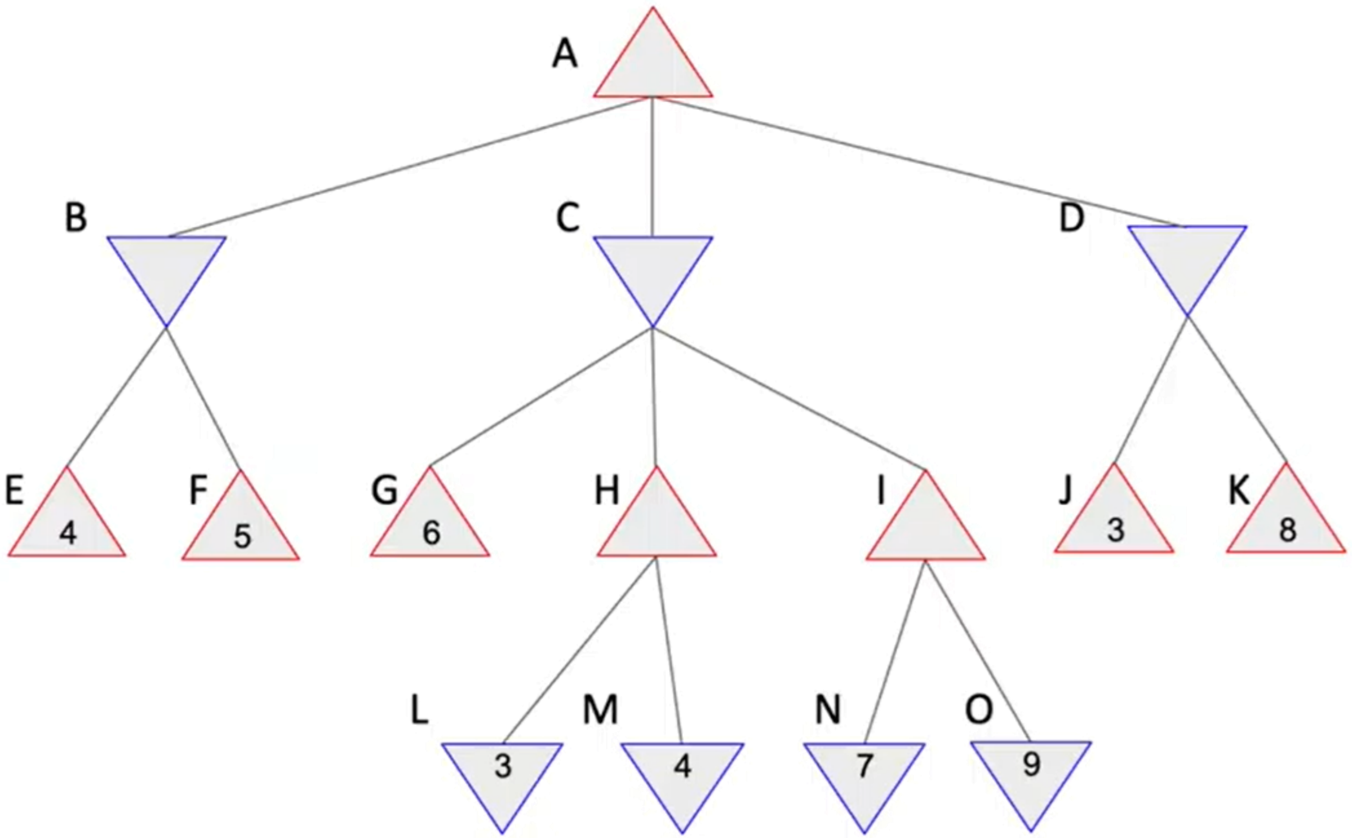
function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

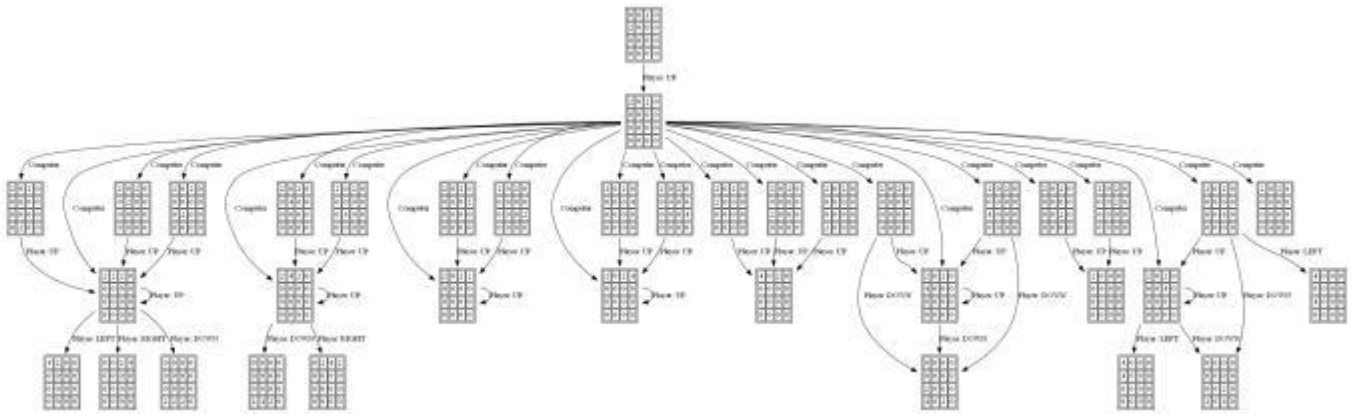
---

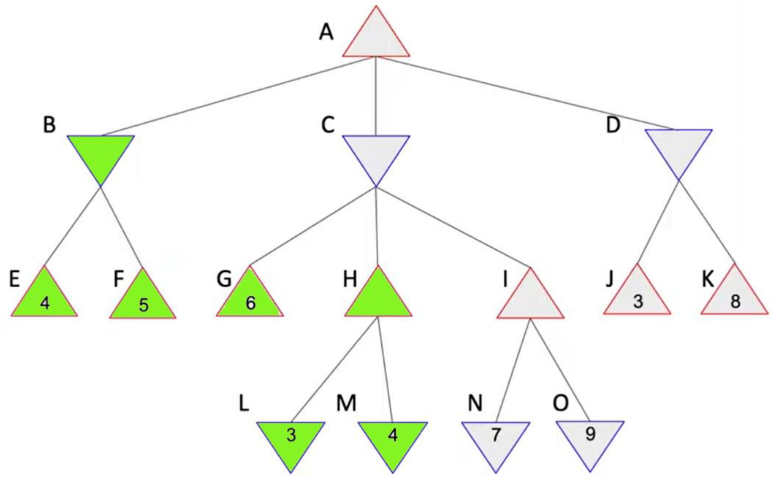
function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

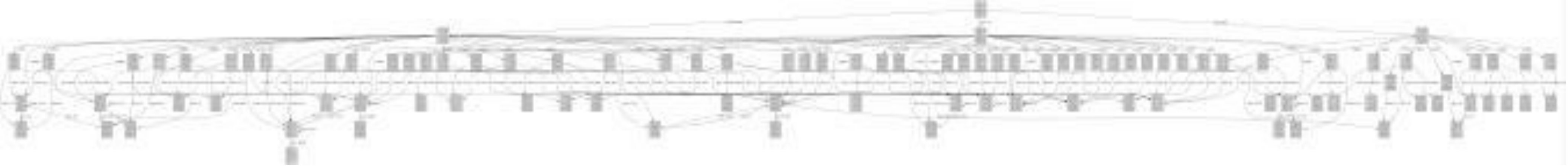


Heuristic Alpha-Beta Tree Search

cut off the search early and apply a heuristic evaluation function to states, effectively treating nonterminal nodes as if they were terminal.

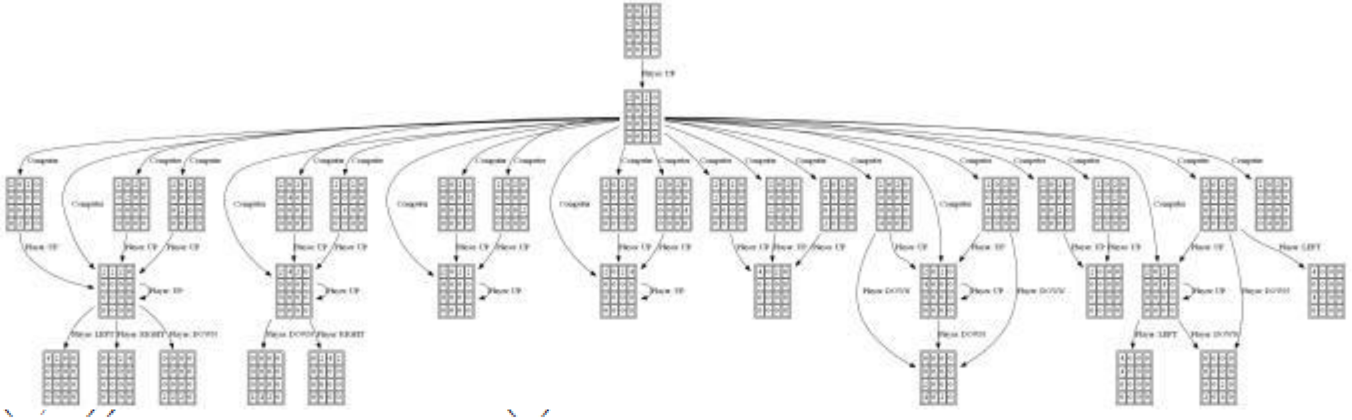






Evaluation Function

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT} \end{cases}$$



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

EVAL(s; p)=UTILITY(s; p) @ terminal state states

UTILITY(loss; p) ≤ EVAL(s; p) ≤ UTILITY(win; p) @ non- terminal state

Conditions of Evaluation Function

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

1. The computation must not take too long! (The whole point is to search faster.)
2. The evaluation function should be strongly correlated with the actual chances of winning.

Example of Evaluation function

one category might contain all two pawn versus one-pawn endgames.

82% win , 2% lose, 16% draw

Expected Value: (0.82x1)+(0.02x0)+ (0.16x0.5) = 0.90

Material Value : pawn is worth 1, a knight or bishop is worth 3, a rook 5, and the queen 9.

Weighted Linear Function

$$\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

Where do the features and weights come from?

1. Culture of human chess-playing experience.
2. Machine learning techniques

Thank You

http://drshiple-courses.weebly.com/autonomous-multiagent-systems.html