Inteligenta Artificiala

Universitatea Politehnica Bucuresti
Anul universitar 2010-2011

Adina Magda Florea

http://turing.cs.pub.ro/ia_10 si
curs.cs.pub.ro

- Cautare cu actiuni nedeterministe
- Strategii de cautare in jocuri

1. Cautare cu actiuni nedeterministe

Problema aspiratorului determinist

- Locatii A,B care pot fi curate (C) sau murdare (M)
- Actiuni agent: St, Dr, Aspira, (nimic)
- 2×2^2 stari posibile (2×2^n)
- M,M, $Agent^A \rightarrow_{Dr}$ M,M, $Agent^B$
- M,M, $Agent^A \rightarrow_{St}$ M,M, $Agent^A$
- M,M, $Agent^A \rightarrow_{Aspira}$ C,M,$Agent^A$
- Stare initiala (M,M, $Agent^A$)
- Plan = [Aspira, Dr, Aspira]

Problema aspiratorului nedeterminist

Aspira nedeterminist

- daca Aspira in M atunci (C) sau (C si C patrat alaturat)
- daca Aspira in C atunci (C) sau (M)

- Stare initiala (M,M, $Agent^A$)
- Plan contingent =

  [Aspira,

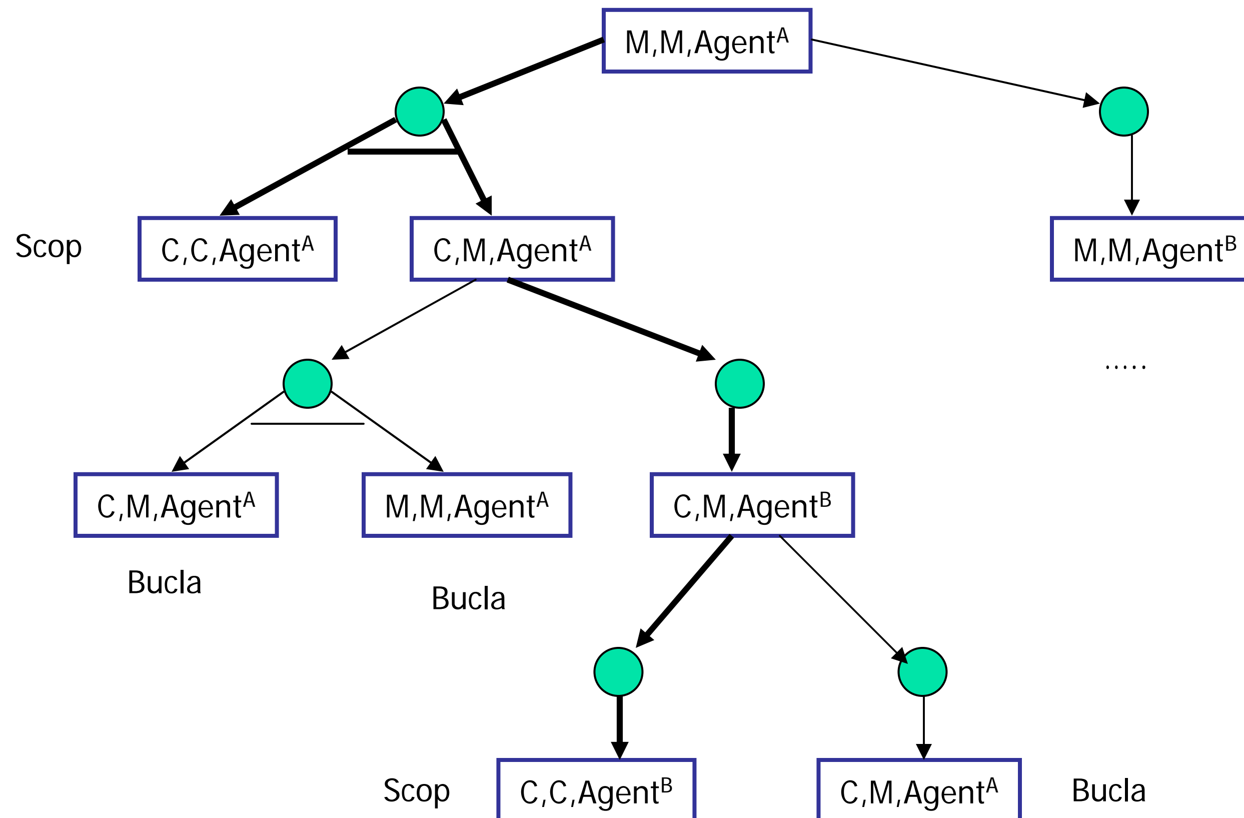
  daca Stare = (C,M, $Agent^A$) **atunci** Dr, Aspira

  altfel nimic]

- Planul – arbore SI/SAU

Problema aspiratorului nedeterminist

- Solutie – un arbore SI/SAU:
  - stare scop in fiecare frunza
  - o actiune dintr-o ramura a unui nod SAU
  - toate actiunile din ramurile unui nod SI



Algoritm Plan: **Determina graf SI/SAU de actiuni**

1. Inspec-SAU(S_i,[])

/* intoarce plan contingent sau INSUCCES */

Inspec-SAU(S, Cale)

1. **daca** S este stare finala
   atunci intoarce Planul vid
2. **daca** $S \in$ Cale **atunci intoarce** INSUCCES
3. **pentru** fiecare actiune A_i posibil de executat din S **executa**
   3.1 Plan $\leftarrow$ Inspec-SI(Stari(S,A_i), [S|Cale])
   3.2 **daca** Plan $\neq$ INSUCCES **atunci intoarce** [A_i|Plan]
4. **intoarce** INSUCCES

sfarsit

Plan contingent

Inspec-SI(Stari, Cale)

1. **pentru** fiecare $S_i \in$ Stari **executa**

    1.1 $Plan_i \leftarrow$ Inspec-SAU(S_i, Cale)

    1.2 **daca** $Plan_i$ = INSUCCES

    atunci intoarce INSUCCES

2. **intoarce**

[**if** S_1 **then** $plan_1$ **else** …**if** S_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

sfarsit

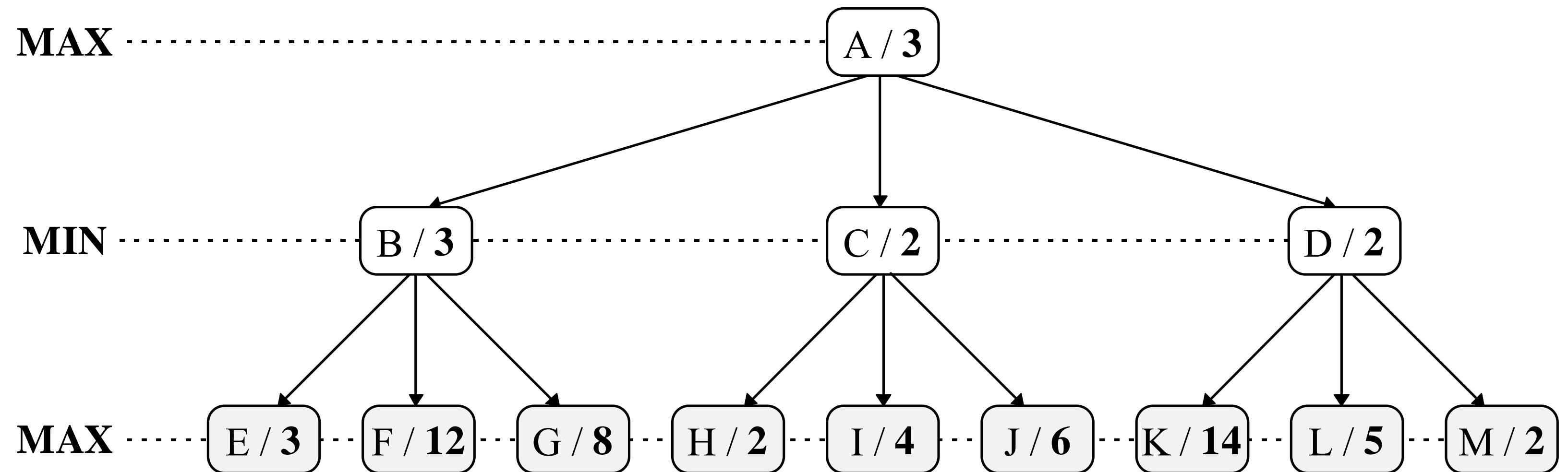
2. Strategii de cautare in jocuri

- Jocuri ce implică doi adversari
  - jucator
  - adversar
- Jocuri in care spatiul de cautare poate fi investigat exhaustiv
- Jocuri in care spatiul de cautare nu poate fi investigat complet deoarece este prea mare.

2.1 Minimax pentru spatii de cautare investigate exhaustiv

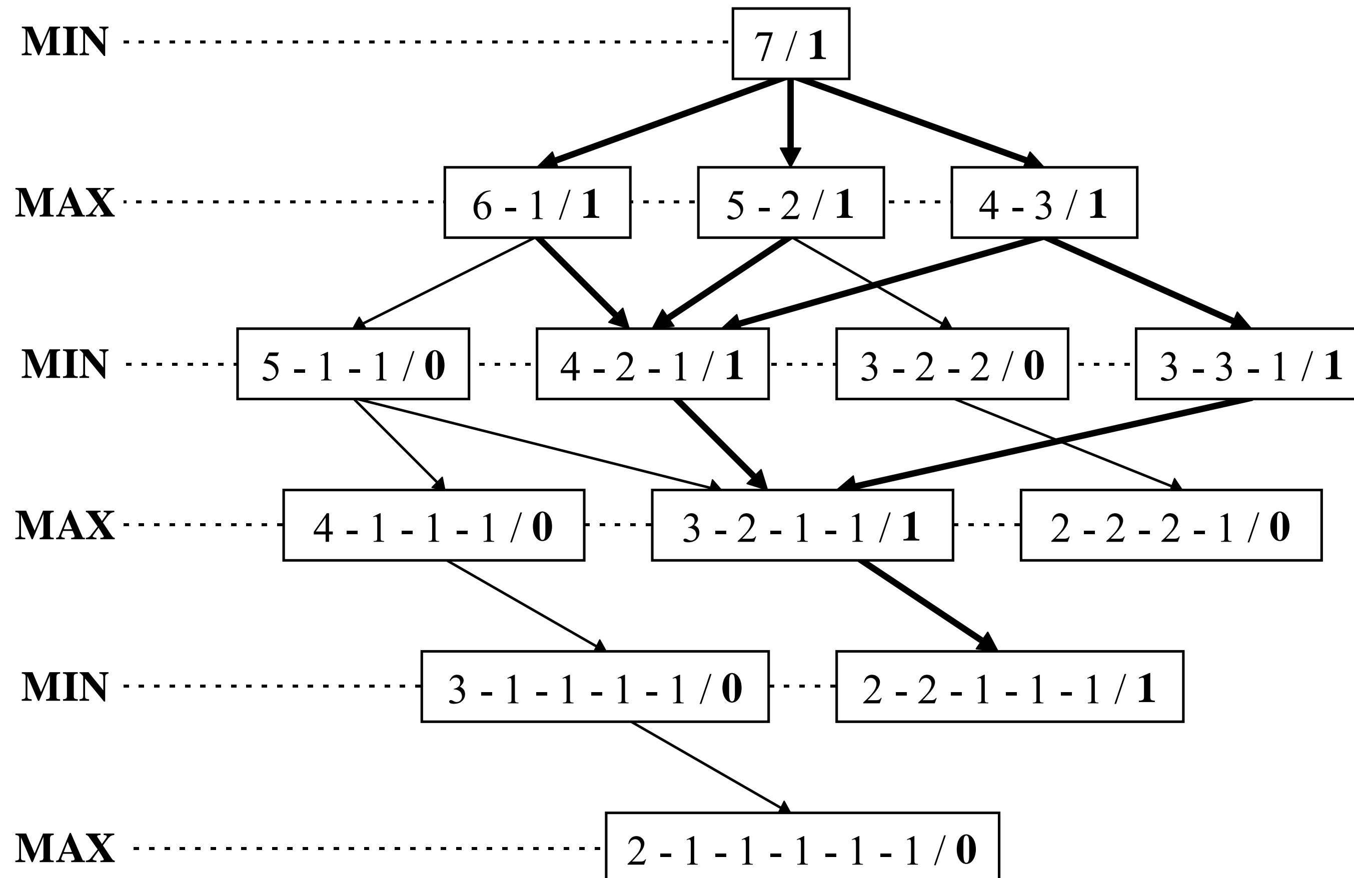
- Jucator – MAX
- Adversar – MIN
- Principiu Minimax
- Etichetez fiecare nivel din AJ cu **MAX** (jucator) si **MIN** (adversar)
- Etichetez frunzele cu scorul jucatorului
- Parcurg AJ
  - daca nodul parinte este **MAX** atunci i se atribuie valoarea maxima a succesorilor sai;
  - daca nodul parinte este **MIN** atunci i se atribuie valoarea minima a succesorilor sai.

Spatiu de cautare Minimax (AJ)



Spatiu de cautare Minimax (AJ)

Nim cu 7 bete



Algoritm: **Minimax cu investigare exhaustiva**

AMinimax(S)

1. pentru fiecare succesor S_j al lui S (obtinut printr-o mutare op_j) **executa**

val(S_j) $\leftarrow$ Minimax(S_j)

2. aplica op_j pentru care val(S_j) este maxima

sfarsit

Minimax(S)

1. daca S este nod final **atunci intoarce** scor(S)

2. altfel

2.1 **daca** MAX muta in S **atunci**

2.1.1 **pentru** fiecare succesor S_j al lui S **executa**

val(S_j) $\leftarrow$ Minimax(S_j)

2.1.2 **intoarce max**(val(S_j), $\forall$j)

2.2 **altfel** { MIN muta in S }

2.2.1 **pentru** fiecare succesor S_j al lui S **executa**

val(S_j) $\leftarrow$ Minimax(S_j)

2.2.2 **intoarce min**(val(S_j), $\forall$j)

sfarsit

2.2 Minimax pentru spatii de cautare investigate pana la o adancime *n*

- Principiu Minimax
- Algoritmul Minimax pana la o adancime *n*
- *nivel(S)*
- O functie euristica de evaluare a unui nod *eval(S)*

Algoritm: Minimax cu adancime finita *n*

AMinimax(S)

1. pentru fiecare succesor S_j al lui S (obtinut printr-o mutare op_j) **executa**

val(S_j) $\leftarrow$ Minimax(S_j)

2. aplica op_j pentru care val(S_j) este maxima

sfarsit

Minimax(S) { intoarce o estimare a starii S }

0. daca S este nod final **atunci intoarce** scor(S)

1. daca nivel(S) = n **atunci intoarce** **eval(S)**

2. altfel

2.1 **daca** MAX muta in S **atunci**

2.1.1 **pentru** fiecare succesor S_j al lui S **executa**

val(S_j) $\leftarrow$ Minimax(S_j)

2.1.2 **intoarce** **max**(val(S_j), $\forall j$)

2.2 **altfel** { MIN muta in S }

2.2.1 **pentru** fiecare succesor S_j al lui S **executa**

val(S_j) $\leftarrow$ Minimax(S_j)

2.2.2 **intoarce** **min**(val(S_j), $\forall j$)

sfarsit

Implementare Prolog

```
play:-
	initialize(Position,Player),
	display_game(Position,Player),
	play(Position,Player,Result).

% play(+Position,+Player,-Result)
play(Position, Player, Result) :-
	game_over(Position,Player,Result), !, write(Result),nl.

play(Position, Player, Result) :-
	choose_move(Position,Player,Move),
	move(Move,Position,Position1),
	next_player(Player,Player1),
	display_game(Position1,Player1),
	!, play(Position1,Player1,Result).

% apel   ?-play.
```

```
move(a1,a,b).
move(a2,a,c).
move(a3,a,d).
move(b1,b,e).
move(b2,b,f).
move(b3,b,g).
move(c1,c,h).
move(c2,c,i).
move(c3,c,j).
move(d1,d,k).
move(d2,d,l).
move(d3,d,m).

next_player(max,min).
next_player(min,max).

initialize(a,max).
display_game(Position,Player):-
        write(Position),nl,write(Player),nl.
```

```
game_over(e,max,3).
game_over(f,max,12).
game_over(g,max,8).
game_over(h,max,2).
game_over(i,max,4).
game_over(j,max,6).
game_over(k,max,14).
game_over(l,max,5).
game_over(m,max,2).

% move(+Move,+Position,-Position1)

% game_over(+Position,+Player,
                        -Result).

% next_player(+Player, - Player1)
```

```
% choose_move(+Position, +Player, -BestMove)
choose_move(Position, Player, BestMove):-
    get_moves(Position,Player,Moves),
    evaluate_and_choose(Moves,Position,10,Player,Record,[BestMove,_]).

% get_moves(+Position, +Player, -Moves)
get_moves(Position, Player, Moves):-
    findall(M,move(M,Position,_),Moves).

% evaluate_and_choose(+Moves, +Position,+D,+MaxMin,
                                        +Record,-BestRecord).
evaluate_and_choose([Move|Moves],Position,D,MaxMin,Record,BestRecord)
    :-
    move(Move,Position,Position1),
    next_player(MaxMin, MinMax),
    minimax(D,Position1,MinMax,Value),
    update(MaxMin,Move,Value,Record,Record1),
    evaluate_and_choose(Moves,Position,D,MaxMin,Record1,BestRecord).

evaluate_and_choose([], Position, D, MaxMin, BestRecord, BestRecord).
```

```
% minimax(+Depth,+Position,+MaxMin,-Value)

minimax(_, Position, MaxMin, Value) :-
        game_over(Position,MaxMin,Value),!.

minimax(0, Position, MaxMin, Value) :-
        eval(Position,Value),!.

minimax(D, Position, MaxMin, Value) :-
        D > 0, D1 is D-1,
        get_moves(Position,MaxMin,Moves),
       evaluate_and_choose(Moves, Position, D1, MaxMin, Record,
                                                   [BestMove,Value]).
```

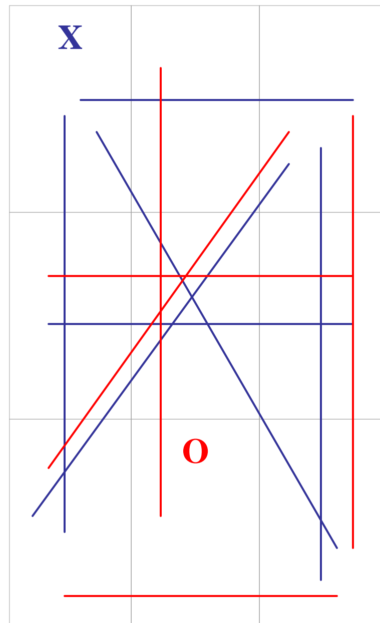
```
% update(+MaxMin, +Move, +Value, +Record, -Record1)

update(_, Move, Value, Record, [Move,Value]) :-
        var(Record),!.

update(max, Move, Value, [Move1,Value1], [Move1,Value1]) :-
         Value =< Value1.
update(max, Move, Value, [Move1,Value1],  [Move,Value]) :-
        Value > Value1.

update(min, Move, Value, [Move1,Value1], [Move1,Value1]) :-
         Value > Value1.
update(min, Move, Value, [Move1,Value1], [Move,Value]) :-
        Value =< Value1.
```

Exemplu de functie de evaluare

Jocul de Tic-Tac-Toe (X si O)

- Functie de estimare euristica **eval(S) -** conflictul existent in starea **S**.
- eval(**S**) = numarul total posibil de linii castigatoare ale lui **MAX** in starea **S** - numarul total posibil de linii castigatoare ale lui **MIN** in starea **S**.
- Daca **S** este o stare din care **MAX** poate face o miscare cu care castiga, atunci eval(**S**) = ∞ (o valoare foarte mare)
- Daca **S** este o stare din care **MIN** poate castiga cu o singura mutare, atunci eval(**S**) = -∞ (o valoare foarte mica).

eval(S) in Tic-Tac-Toe



X are 6 linii castigatoare posibile

O are 5 linii castigatoare posibile

eval(**S**) = 6 - 5 = 1

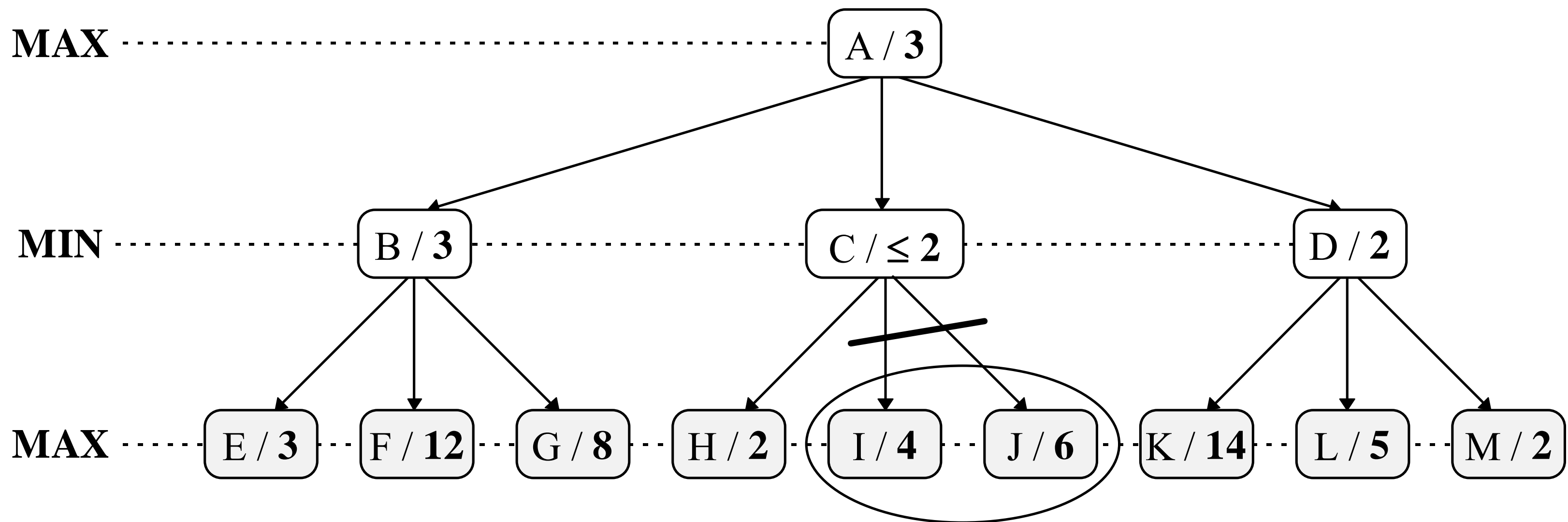
2.3 Algoritmul taierii alfa-beta

- Este posibil sa se obtină decizia corecta a algoritmului **Minimax** fara a mai inspecta toate nodurile din spatiului de cautare pana la un anumit nivel.
- Procesul de eliminare a unei ramuri din arborele de cautare se numeste *taierea arborelui de cautare* (*pruning*).

Algoritmul taierii alfa-beta

- Fie α cea mai buna valoare (cea mai mare) gasita pentru **MAX** si β cea mai buna valoare (cea mai mica) gasita pentru **MIN**.
- Algoritmul **alfa-beta** actualizeaza α si β pe parcursul parcurgerii arborelui si elimina investigarile subarborilor pentru care α sau β sunt mai proaste.
- Terminarea cautarii (taierea unei ramuri) se face dupa doua reguli:
  - Cautarea se opreste sub orice nod **MIN** cu o valoare β mai mica sau egala cu valoarea α a oricaruia dintre nodurile **MAX** predecesoare nodului **MIN** in cauza.
  - Cautarea se opreste sub orice nod **MAX** cu o valoare α mai mare sau egala cu valoarea β a oricaruia dintre nodurile **MIN** predecesoare nodului **MAX** in cauza.

Tăierea alfa-beta a spaţiului de căutare



Algoritm: **Alfa-beta**

MAX(S, α, β) { intoarce valoarea maxima a unei stari. }

0. daca S este nod final **atunci intoarce** scor(S)

1. daca nivel(S) = n **atunci** intoarce eval(S)

2. altfel

2.1 **pentru** fiecare succesor S_j al lui S **executa**

2.1.1 $\alpha \leftarrow \max(\alpha, MIN(S_j, \alpha, \beta))$

2.1.2 **daca** $\alpha \geq \beta$ **atunci** intoarce β

2.2 **intoarce** α

sfarsit

MIN(S, α, β) { intoarce valoarea minima a unei stari. }

0. daca S este nod final **atunci intoarce** scor(S)

1. daca nivel(S) = n **atunci intoarce** eval(S)

2. altfel

2.1 **pentru** fiecare succesor S_j al lui S **executa**

2.1.1 $\beta \leftarrow \min(\beta, MAX(S_j, \alpha, \beta))$

2.1.2 **daca** $\beta \leq \alpha$ **atunci intoarce** α

2.2 **intoarce** β

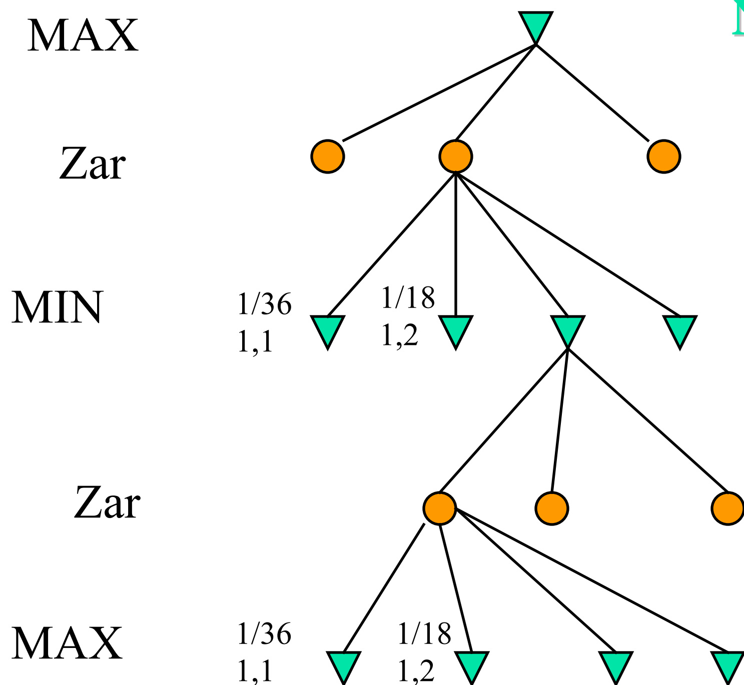
sfarsit

2.4 Jocuri cu elemente de sansa

- Jucatorul nu cunoaste miscarile legale ale oponentului
- 3 tipuri de noduri:
  - MAX
  - MIN
  - Sansa (chance nodes)

- 36 rez pt 2 zaruri
- 21 noduri distincte
- Zaruri egale (6 dist) - > 1/36
- Zaruri diferite (15 dist) -> 1/18

- Valoarea estimata pt noduri sansa
- **$\text{Suma}_{S_j \text{ suc } S}[\, P(S_j) * \text{EMinimax}(S_j)]$**



Noduri de decizie

- **Functia de evaluare**
- scor – nod terminal
- max din EMinimax succesori - MAX
- min din EMinimax succesori - MIN
- Suma [$P(S_j) * \text{EMinimax}(S_j)$] succesori - SANSA

Noduri sansa