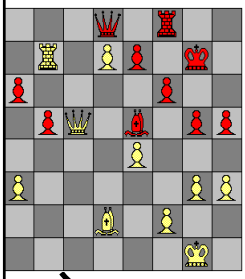
Proiectarea Algoritmilor

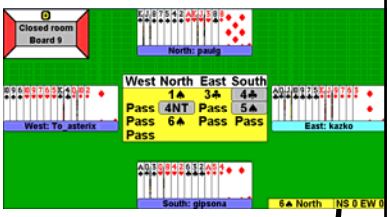
Curs 4 – Algoritmi pentru jocuri
Minimax, α-β

Bibliografie

- Giumale – Introducere in Analiza Algoritmilor cap 7.6
- http://www.dwheeler.com/chess-openings/#Sicilian%20Defense
- http://mouserunner.com/MozllaTicTacToe/Mozilla_Tic_Tac_Toe.htm

Problema

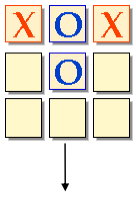




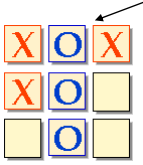
PA

10

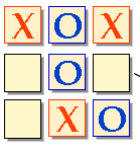
Cum gândim noi?



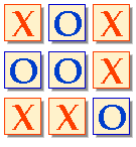
Analizăm posibilitățile și evaluăm
fiecare mutare în funcție de consecințe.



Am pierdut! :(



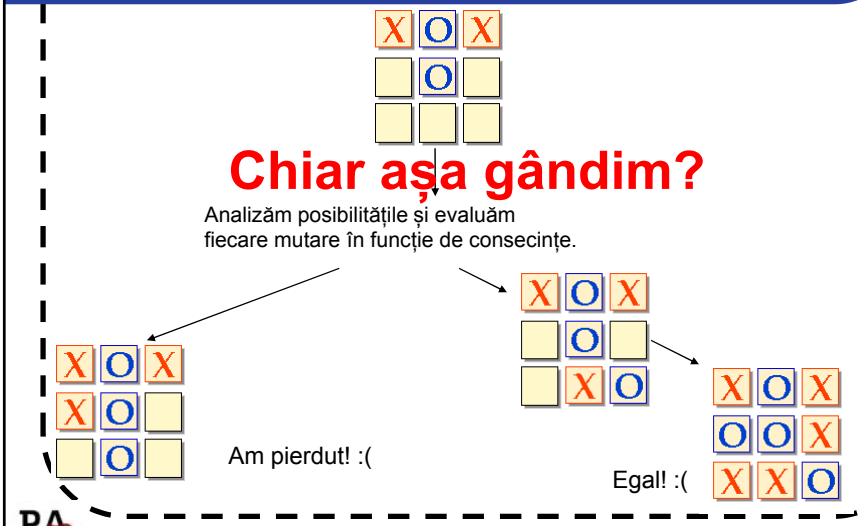
Egal! :(



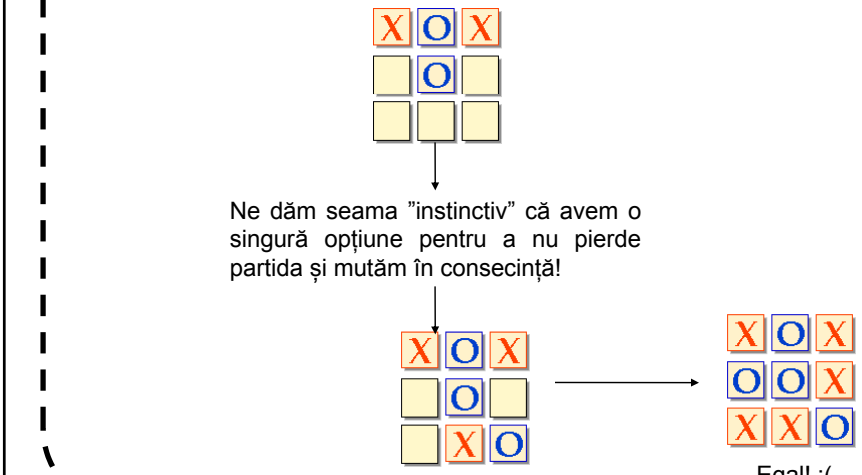
PA

Proiectarea Algoritmilor 2010

Cum gândim noi?



Cum gândim noi?

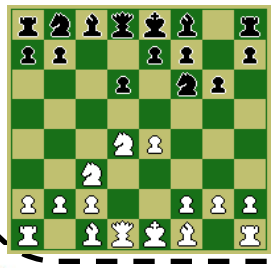


Cum gândim noi?

http://www.dwheeler.com/chess-openings/#Sicilian%20Defense

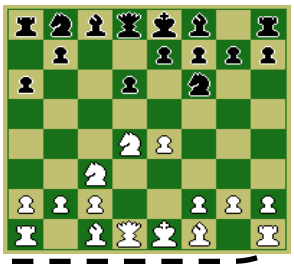
Apărarea siciliană!

Varianta Najdorf



Când avem foarte multe posibilități la dispoziție încercăm să folosim poziții (pattern-uri) cunoscute.

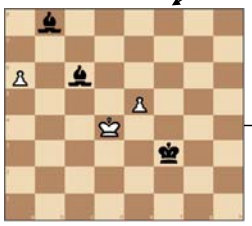
Varianta Dragon



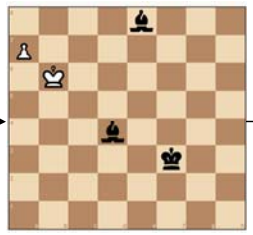
Cum gândim noi?

Albul la mutare
-11 posibilități de mutare;
- le putem încerca pe toate să vedem ce se întâmplă.

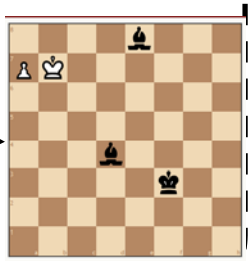
Circa 15.000 de mutări de analizat – ușor pentru calculator. Noi eliminăm mutările ce ni se par fără sens (mai mult de jumătate).



Numărul mutărilor posibile se reduce la 6.



Doar 4 mutări posibile.

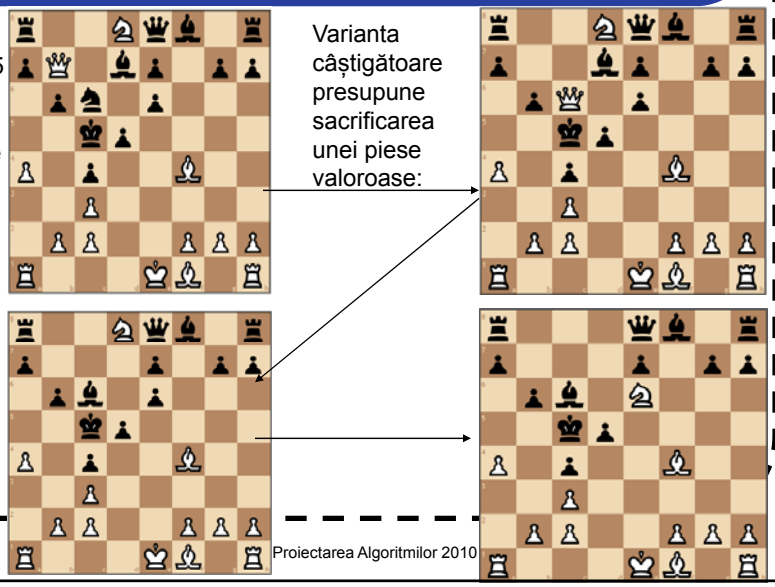


Remiză asigurată!

Cum gândim noi?

Cam 35 de mutări posibile

Varianta câștigătoare presupune sacrificarea unei piese valoroase:



Cum gândim noi?

- Evaluăm amenințările:
  - Căutăm mutări care să minimizeze pierderile;
  - Căutăm mutări care să maximizeze câștigul.
- Alegem mutările ce ni se ”par” cele mai bune pe moment:
  - Explorăm în adâncime graful mutărilor;
  - Numărul de niveluri = minim dintre:
    - Terminarea jocului;
    - Obținerea unui avantaj consistent fără pericol aparent de a-l pierde;
    - Nivelul maxim al capacității noastre de calcul.

Abordări posibile pentru calculator

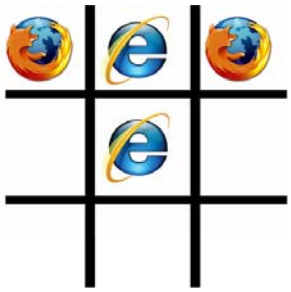
- Șabloane pentru poziții standard.
- Căutare în spațiul de poziții.
- Utilizare euristici pentru evaluarea poziției curente.
- Ne vom concentra asupra căutărilor.

Metoda minimax

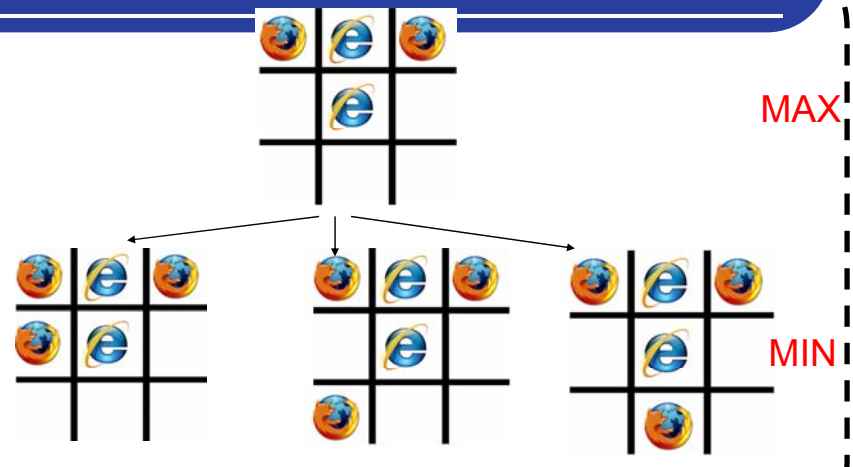
- 2 jucători: Max si Min care mută pe rând (Max muta primul).
- Max urmărește să-și maximizeze câștigul.
- Min urmărește să-și minimizeze pierderea.
- Se construiește un arbore AND-OR:
- Nivelele impare → mutările jucătorului Max.
- Nivelele pare → mutările jucătorului Min.
- Frunzele desemneaza castigul/pierderea lui Max.
- Arcele reprezinta mutarile propriu-zise.

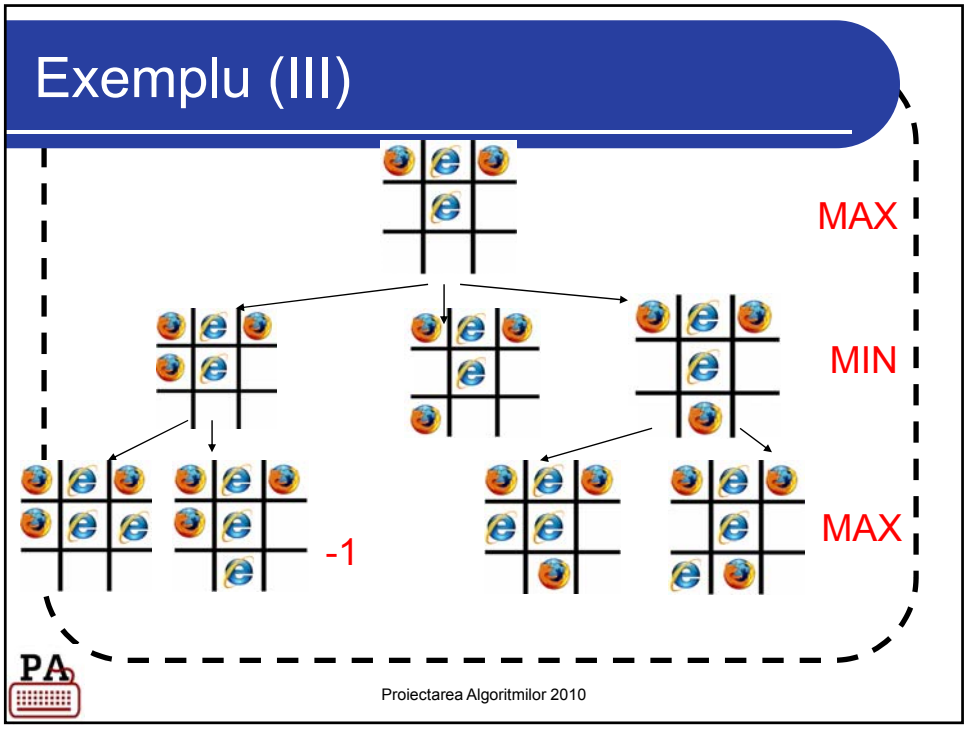
Exemplu (I)

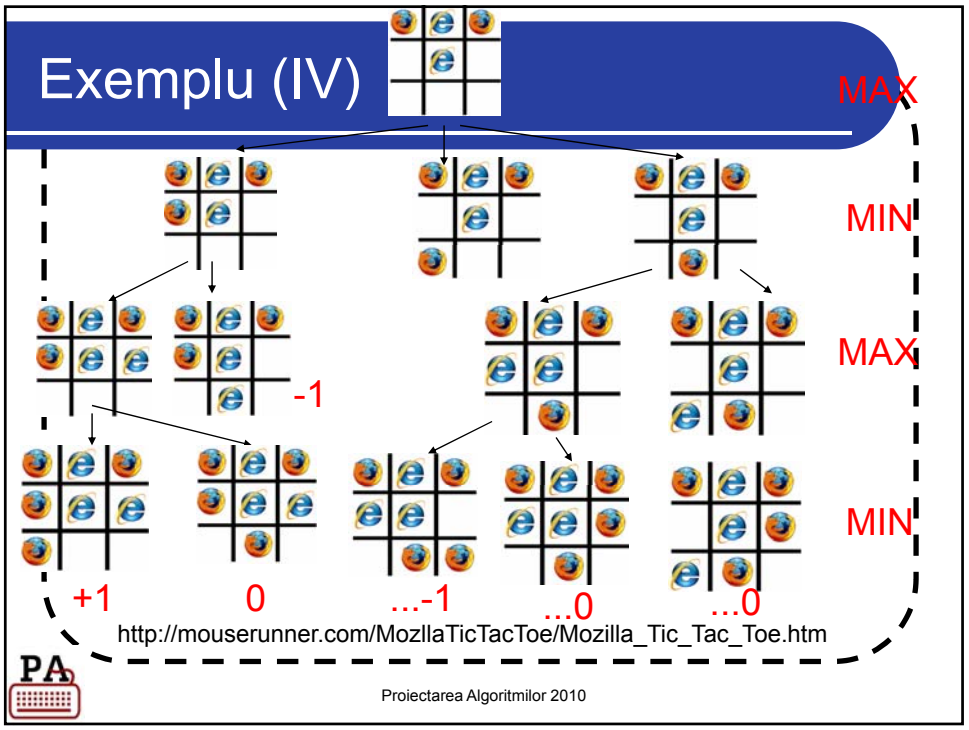
MAX (Firefox) trebuie să mute:



Exemplu (II)



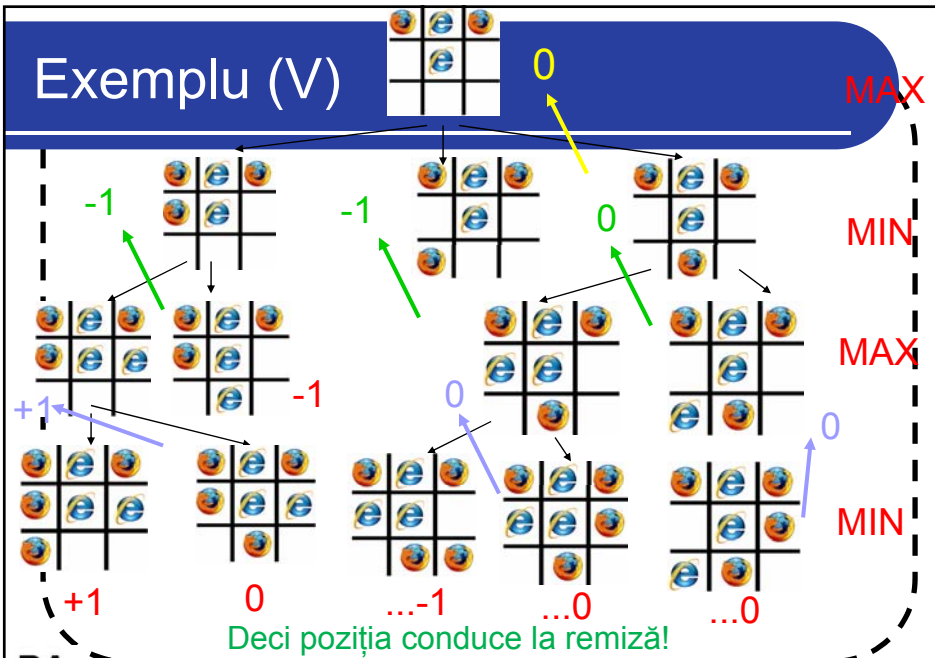

Exemplu (III)
MAX
MIN
MAX
-1
PA
Proiectarea Algoritmilor 2010


Exemplu (IV)
MAX
MIN
MAX
MIN
-1
+1
0
...-1
...0
...0
http://mouserunner.com/MozllaTicTacToe/Mozilla_Tic_Tac_Toe.htm
PA
Proiectarea Algoritmilor 2010

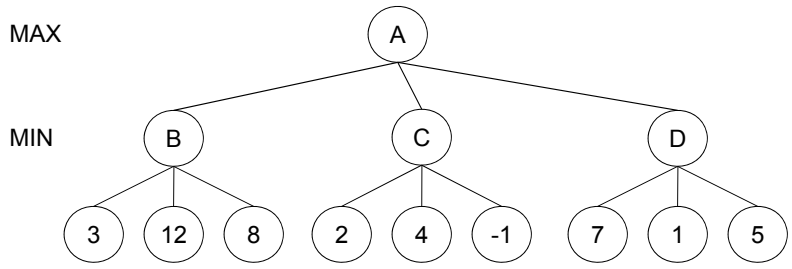
Functionare Minimax

- 1) Se generează întregul arbore;
- 2) Se evaluează frunzele și li se asociază valori;
- 3) Se propagă rezultatele dinspre frunze spre rădăcină astfel:
  - Nivelul MIN alege cea mai mică valoare dintre cele ale copiilor.
  - Nivelul MAX alege cea mai mare valoare dintre cele ale copiilor.

Exemplu (V)



Alt exemplu (I)



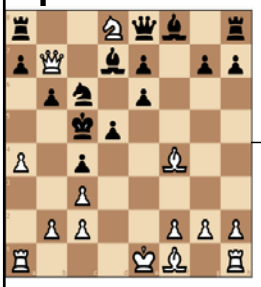
Probleme

- Dimensiunea arborelui pentru "X și 0" e < 9!
- Pentru Șah fiecare nod are în medie 35 copii!
- Pentru Go ramificarea este de cca. 150 – 250!
- Complexitatea arborelui:
  - pentru Șah – 10^{123} noduri;
  - pentru Go – 10^{360} noduri.
- Limitări:→ Nu putem să construim întregul arbore → Nu putem ajunge de fiecare dată la stările finale pentru a le putea evalua.

Optimizări minimax

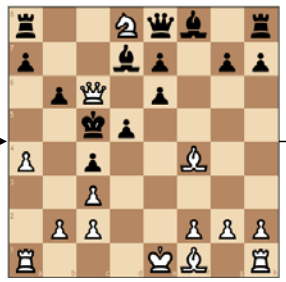
- Limitarea adâncimii căutării
  - Trebuie să construim o funcție euristică care să estimeze șansele de câștig pentru o poziție dată.
    - Ex. pentru șah:
      - Regină:10p; Turn: 5p; Cal, nebun: 3p; Pion: 1p;
      - Ex: Funcție de evaluare a poziției = suma pieselor proprii – suma pieselor adversarului.
  - Oprirea căutării:
    - Limitare statică: după un număr maxim de nivele/interval de timp.
    - Limitare dinamică: când profitul obținut din continuarea căutării devine foarte mic (scade sub o valoare fixata).
  - Se estimează valoarea funcției de evaluare la nivelul respectiv.
  - Apoi propagăm valorile conform principiului enunțat anterior.

Exemplu și contraexemplu



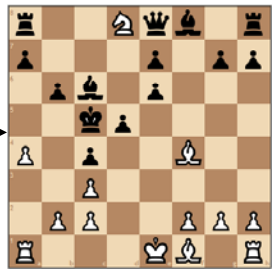
Eval: 36-37=-1

Funcția nu ține cont de poziție – albul are o poziție net superioară dar funcția de evaluare o ignoră



Eval: 36-34=2

Dacă căutarea se oprește la acest nivel atunci aparent albul iese în câștig material ignorându-se faptul că la mutarea următoare se pierde dama



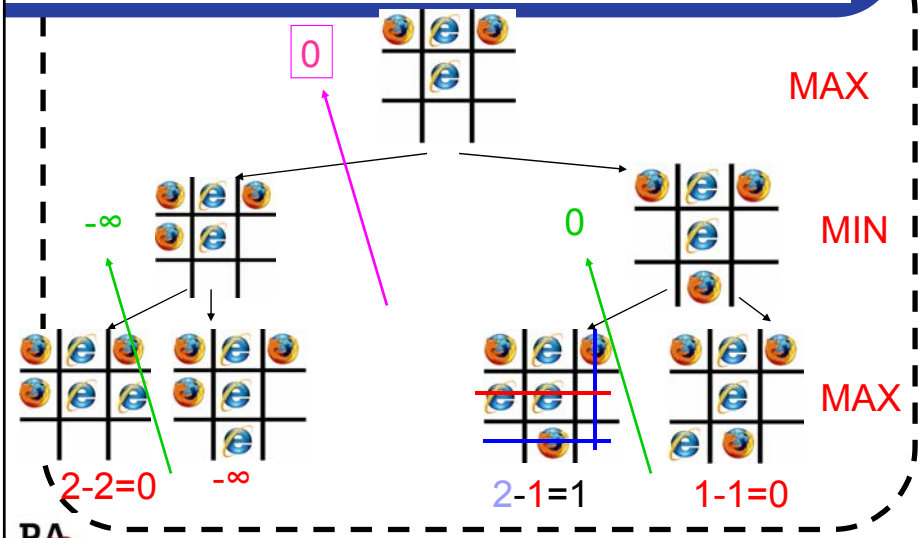
Eval: 26-34=-8

Dacă căutarea se oprește la acest nivel aparent albul iese in dezavantaj deoarece a pierdut dama

Exemplu funcție euristică X și 0

- F = numărul de linii/coloane/diagonale posibil câștigătoare pentru MAX – numărul de linii/coloane/diagonale posibil câștigătoare pentru MIN.
- Dacă MAX poate să mute și să câștige atunci F = +∞; dacă MIN poate să mute și să câștige F = -∞.

Exemplu funcție euristică X și 0



Minimax – funcții de evaluare

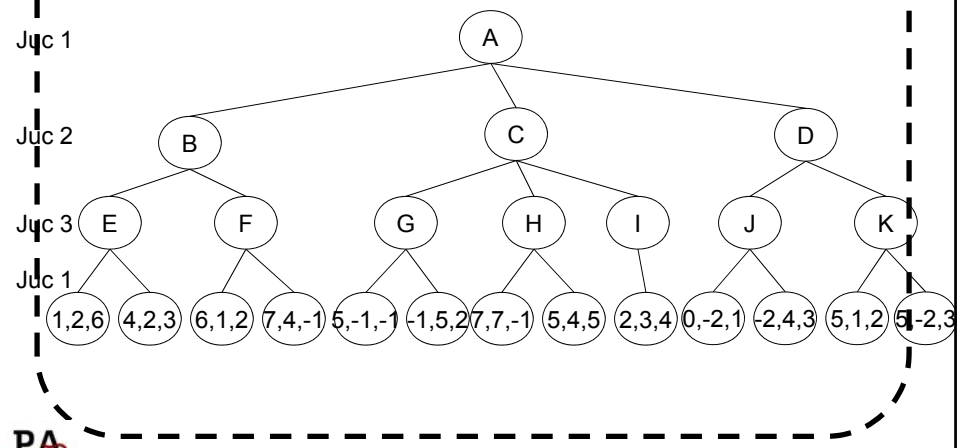
- Funcția euristică trebuie să cuantifice ”poziția”
  - Chiar în dauna avantajului material.
- Trebuie să ia în calcul potențialele amenințări!

Algoritm MINIMAX

- MINIMAX_limitat (n, nivel_limita)
  - *Pentru fiecare* n’ ∈ succs(n) // pentru toate mutările
    - Fie m = mutarea corespunzătoare arcului (n,n’)
    - VAL(m) = w(n’, nivel_limita, 1) // determin valoarea mutării
  - *Întoarce* m a.î. VAL(m) = max {VAL(x) | x ∈ mutări(n)}
- W(n, limita,nivel)
  - *Dacă* n este frunză *întoarce* cost(n)
  - *Dacă* nivel ≥ limită *întoarce* euristică(n)
  - *Dacă* jucătorul MAX este la mutare *întoarce*
    - max {w(n’, limita, nivel + 1) | n’ ∈ succs(n)}
  - *Dacă* jucătorul MIN este la mutare *întoarce*
    - min {w(n’, limita, nivel + 1) | n’ ∈ succs(n)}

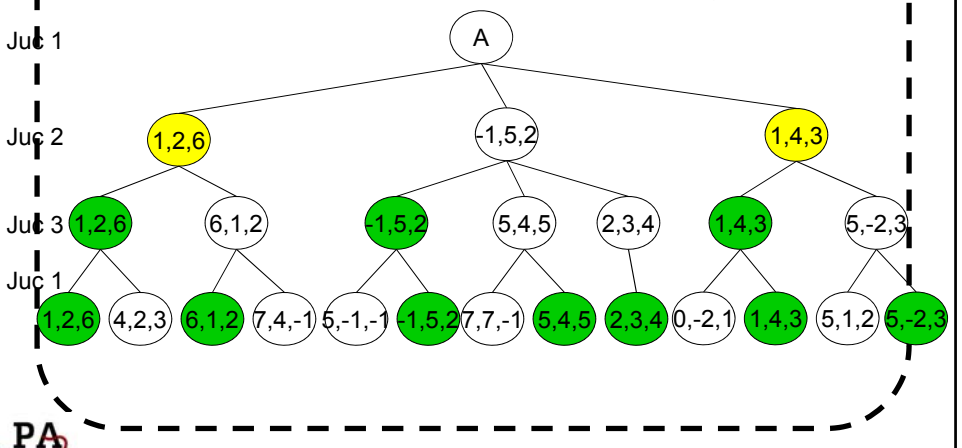
Caz special - Minimax 3 jucatori

Jucătorii vor alege pe rând valoarea care le maximizează câștigul propriu



Caz special - Minimax 3 jucatori (2)

Jucătorii vor alege pe rând valoarea care le maximizează câștigul propriu

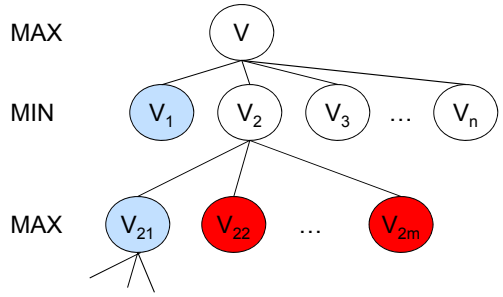


Caz special (2) – Minimax Probabilistic

- La unele jocuri, mutările sunt guvernate de șansă.
- Ex: Jocul de Table – mulțimea mutărilor este limitată de:
  - starea curentă a jocului;
  - combinația zarurilor in starea curentă.
- Arborele MINIMAX este completat cu noduri suplimentare (noduri șansă) plasate intre nodurile MIN/MAX (MIN – șansă – MAX si MAX – șansă – MIN).
- Valorile se calculează ca sumă ponderată intre probabilitatea nodului si evaluarea acestuia (prin cost sau euristică).

Tăiere α-β

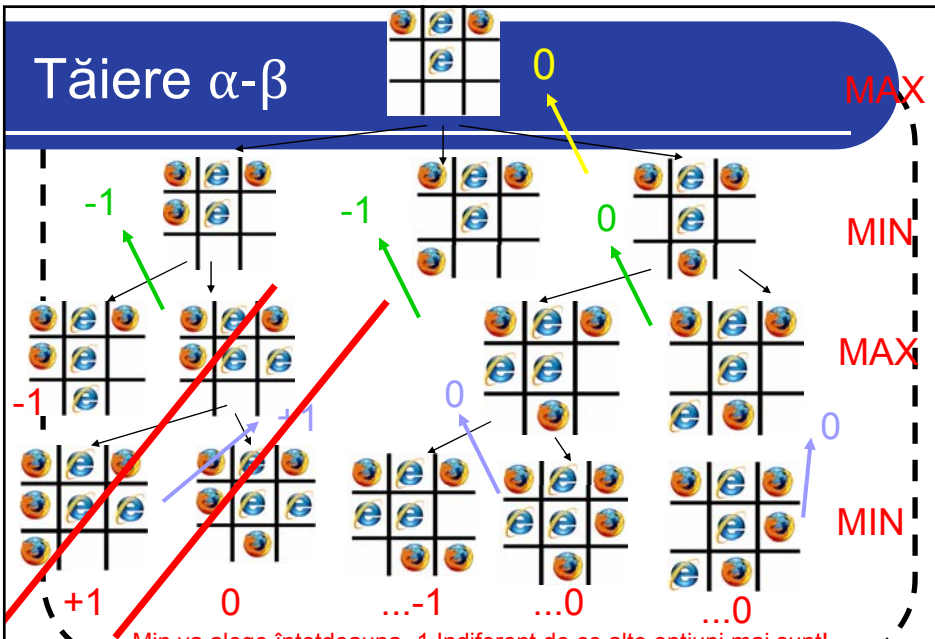
- Încercăm să limităm spațiul de căutare prin eliminarea variantelor ce nu au cum să fie alese.
- Idee:
  - Dacă $V_{21}<V_1$ toată ramura V_2 poate fi Ignorată.



Tăiere α-β

- α = max dintre valorile găsite pentru un nod MAX
- β = min dintre valorile găsite pentru un nod MIN
- Tăiem o ramură dacă:
  - am găsit un nod pe nivelul MAX cu valoare β <= oricare din valorile α calculate anterior;
  - am găsit un nod pe nivelul MIN cu valoare α >= oricare din valorile β calculate anterior.
- Teorema α-β. Fie J un nod din arborele MINIMAX explorat. Daca α(J) ≥ β(J), atunci explorarea nodului J nu este necesară.

Tăiere α-β

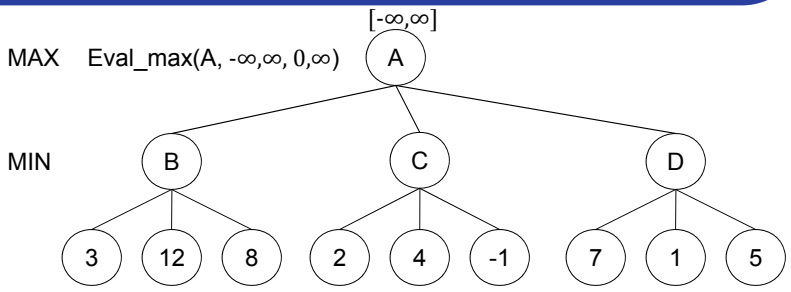


Min va alege întotdeauna -1 Indiferent de ce alte opțiuni mai sunt!

Algoritm α-β

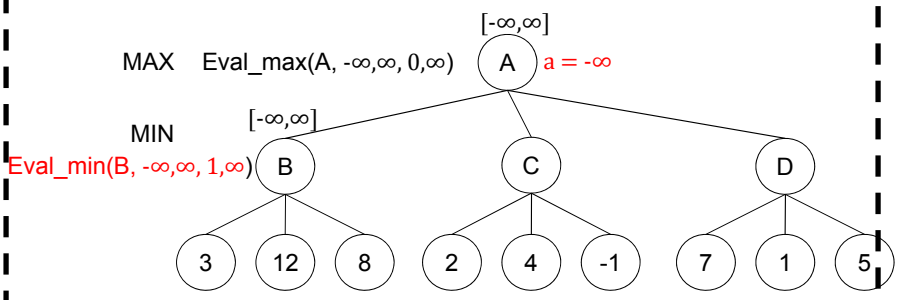
- α-β(n, limită)
  - w = eval_max(n, -∞, ∞, 0, limită)
  - *întoarce* m ∈ mutări(n) a.i. VAL(m) = w
- eval_max(n, α, β, nivel, limită)
  - *Dacă* (tip(n) == terminal) // am ajuns la frunză
    - *întoarce* cost(n)
  - *Dacă* (nivel ≥ limita) *întoarce* euristică(n) // sunt limitat
  - a = -∞ // valoarea curenta a nodului de tip max
  - *pentru fiecare* (n' ∈ succs(n)) {
    - a = max(a, eval_min(n', max(α,a), β, nivel+1, limită); // propag
    - if (a ≥ β) break; }
  - *întoarce* a
- similar eval_min

Alt exemplu (II)



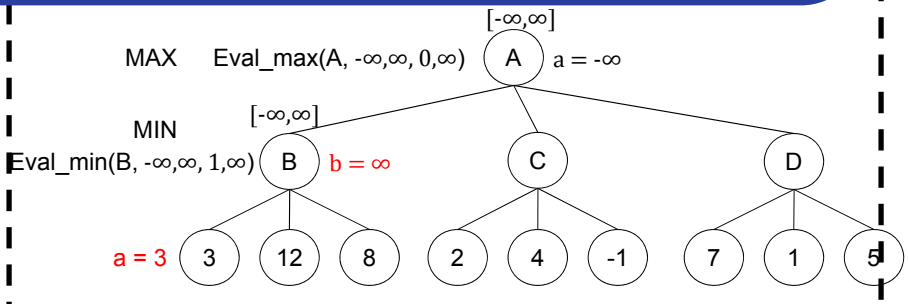
eval_max(n, α, β, nivel, limită)
Dacă (tip(n) == terminal) // am ajuns la frunză
întoarce cost(n)
Dacă (nivel ≥ limita) *întoarce* euristică(n) // sunt limitat
a = -∞ // valoarea curenta a nodului de tip max
pentru fiecare (n' ∈ succs(n)) {
a = max(a, eval_min(n', max(α,a), β, nivel+1, limită); // propag
if (a ≥ β) break; }
întoarce a

Alt exemplu (III)



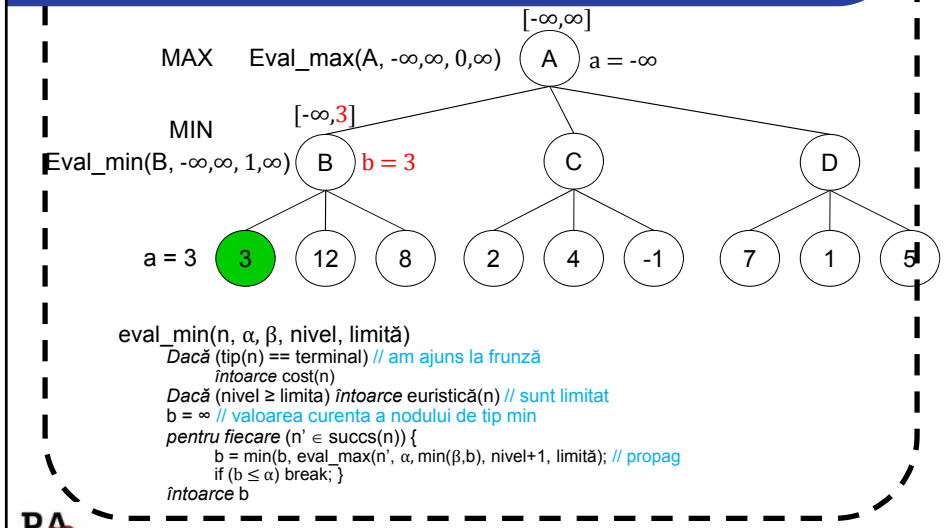

```
eval_max(n, α, β, nivel, limită)
    Dacă (tip(n) == terminal) // am ajuns la frunză
        întoarce cost(n)
    Dacă (nivel ≥ limita) întoarce euristică(n) // sunt limitat
    a = -∞ // valoarea curenta a nodului de tip max
    pentru fiecare (n' ∈ succs(n)) {
        a = max(a, eval_min(n', max(α,a),  β, nivel+1, limită); // propag
        if (a ≥ β) break; }
    întoarce a
```


Alt exemplu (IV)



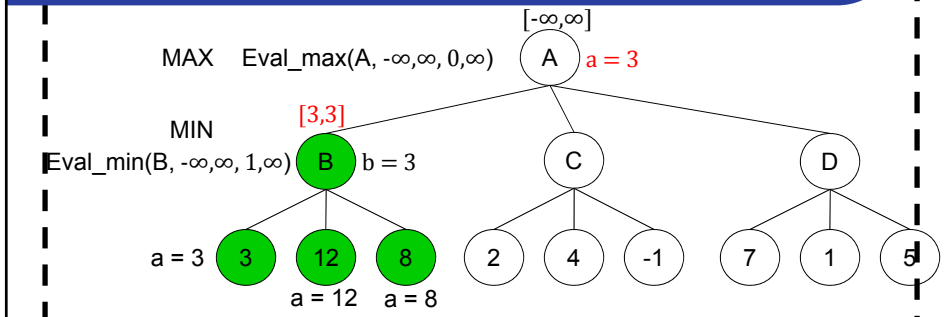

```
eval_min(n, α, β, nivel, limită)
    Dacă (tip(n) == terminal) // am ajuns la frunză
        întoarce cost(n)
    Dacă (nivel ≥ limita) întoarce euristică(n) // sunt limitat
    b = ∞ // valoarea curenta a nodului de tip min
    pentru fiecare (n' ∈ succs(n)) {
        b = min(b, eval_max(n', α, min(β,b), nivel+1, limită); // propag
        if (b ≤ α) break; }
    întoarce b
```

Alt exemplu (V)



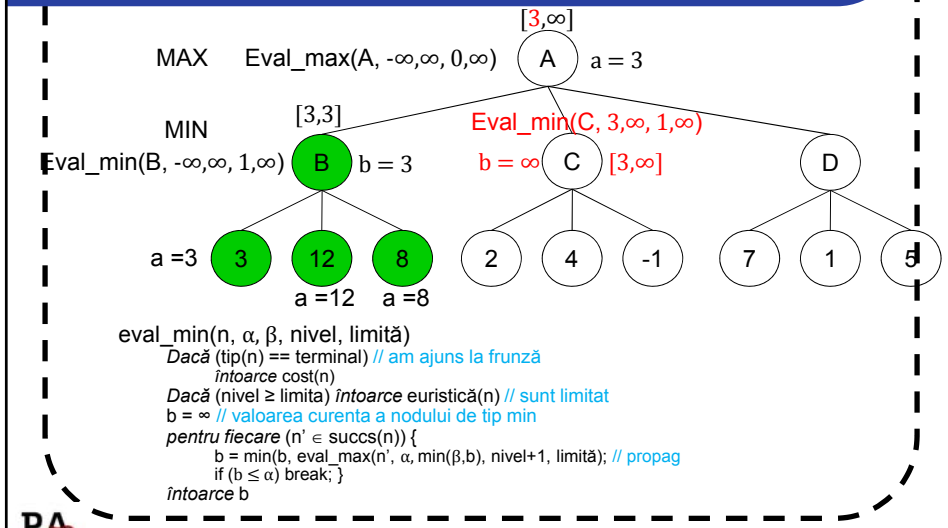

```
eval_min(n, α, β, nivel, limită)
    Dacă (tip(n) == terminal) // am ajuns la frunză
        întoarce cost(n)
    Dacă (nivel ≥ limita) întoarce euristică(n) // sunt limitat
    b = ∞ // valoarea curenta a nodului de tip min
    pentru fiecare (n' ∈ succs(n)) {
        b = min(b, eval_max(n', α, min(β,b), nivel+1, limită); // propag
        if (b ≤ α) break; }
    întoarce b
```


Alt exemplu (VI)



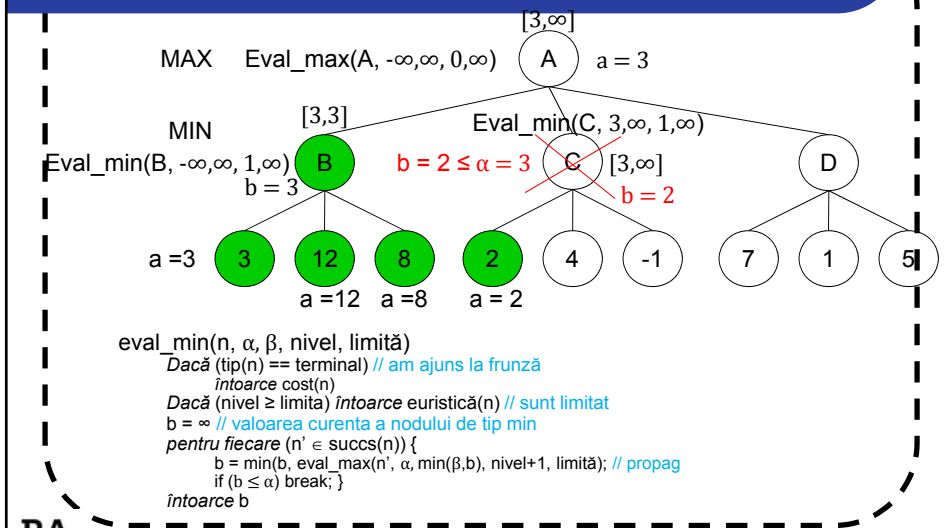

```
eval_max(n, α, β, nivel, limită)
    Dacă (tip(n) == terminal) // am ajuns la frunză
        întoarce cost(n)
    Dacă (nivel ≥ limita) întoarce euristică(n) // sunt limitat
    a = -∞ // valoarea curenta a nodului de tip max
    pentru fiecare (n' ∈ succs(n)) {
        a = max(a, eval_min(n', max(α,a), β, nivel+1, limită); // propag
        if (a ≥ β) break; }
    întoarce a
```

Alt exemplu (VII)



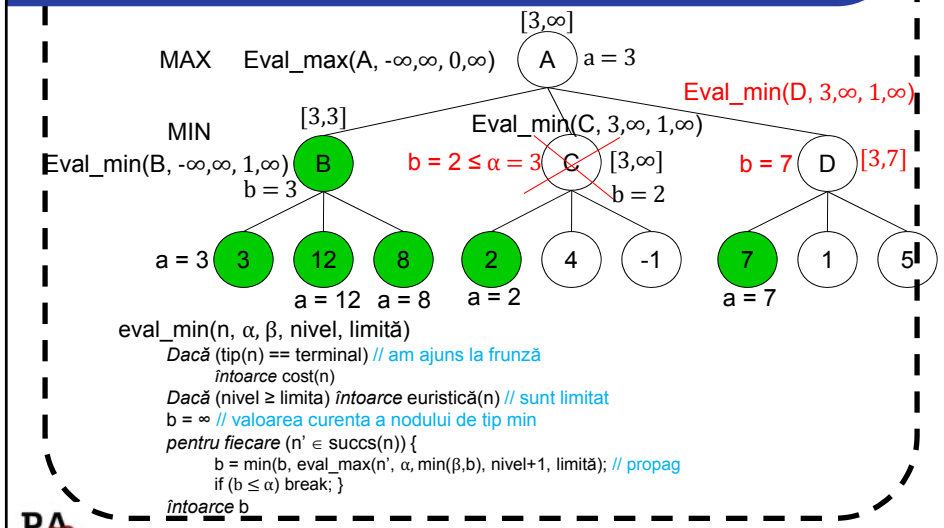

```
eval_min(n, α, β, nivel, limită)
    Dacă (tip(n) == terminal) // am ajuns la frunză
        întoarce cost(n)
    Dacă (nivel ≥ limita) întoarce euristică(n) // sunt limitat
    b = ∞ // valoarea curenta a nodului de tip min
    pentru fiecare (n' ∈ succs(n)) {
        b = min(b, eval_max(n', α, min(β,b), nivel+1, limită); // propag
        if (b ≤ α) break; }
    întoarce b
```


Alt exemplu (VIII)



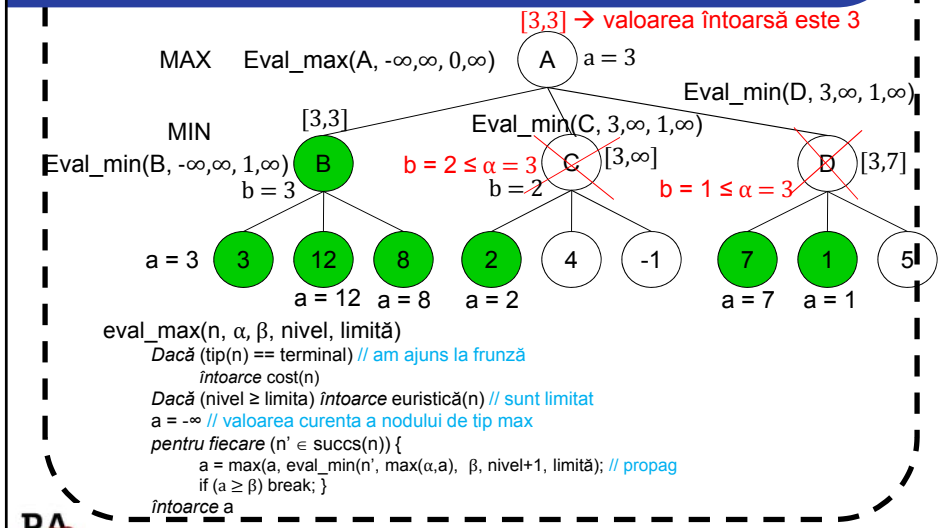

```
eval_min(n, α, β, nivel, limită)
    Dacă (tip(n) == terminal) // am ajuns la frunză
        întoarce cost(n)
    Dacă (nivel ≥ limita) întoarce euristică(n) // sunt limitat
    b = ∞ // valoarea curenta a nodului de tip min
    pentru fiecare (n' ∈ succs(n)) {
        b = min(b, eval_max(n', α, min(β,b), nivel+1, limită); // propag
        if (b ≤ α) break; }
    întoarce b
```

Alt exemplu (IX)




```
eval_min(n, α, β, nivel, limită)
    Dacă (tip(n) == terminal) // am ajuns la frunză
        întoarce cost(n)
    Dacă (nivel ≥ limita) întoarce euristică(n) // sunt limitat
    b = ∞ // valoarea curenta a nodului de tip min
    pentru fiecare (n' ∈ succs(n)) {
        b = min(b, eval_max(n', α, min(β,b), nivel+1, limită); // propag
        if (b ≤ α) break; }
    întoarce b
```


Alt exemplu (X)




```
eval_max(n, α, β, nivel, limită)
    Dacă (tip(n) == terminal) // am ajuns la frunză
        întoarce cost(n)
    Dacă (nivel ≥ limita) întoarce euristică(n) // sunt limitat
    a = -∞ // valoarea curenta a nodului de tip max
    pentru fiecare (n' ∈ succs(n)) {
        a = max(a, eval_min(n', max(α,a), β, nivel+1, limită); // propag
        if (a ≥ β) break; }
    întoarce a
```

Observații α-β

- Reduce complexitatea minimax în cazul ideal de la
  - Număr_ramificări^{număr_nivele} la Număr_ramificări$^{număr_nivele/2}$
- Contează foarte mult ordinea în care analizăm mutările!
  - Sortarea mutărilor după un criteriu dat nu este costisitoare comparativ cu costul exponențial al algoritmului.
- Se folosesc euristici pentru a alege mutările examinate mai întâi:
  - ex: la șah se aleg întâi mutările in care se iau piese;
  - sau se aleg mai întâi mutările cu scor bun in parcurgeri precedente;
  - sau se aleg mutările care au mai generat tăieri.

Observații MINIMAX și α-β

- Algoritmi de căutare în adâncime.
- Pot cauza probleme când avem un timp limită.
- → soluție posibilă IDDFS (căutare în adâncime mărind iterativ adâncimea maximă până la care căutăm).

Concluzii

- Algoritmi cu complexitate foarte mare.
- Soluții euristice pentru limitarea complexității.
- Recomandabil să se combine cu alte strategii – baze de date cu poziții, pattern-matching.

- **function** alphabeta(node, depth, α, β)
  - *(* β represents previous player best choice - doesn't want it if α would worsen it *)*
  - **if** depth = 0 "or" node is a terminal node
    - **return** the heuristic value of node
  - **foreach** child of node
    - α := max(α, -alphabeta(child, depth-1, -β, -α))
    - *(* use symmetry, -β becomes subsequently pruned α *)*
    - **if** β≤α
      - **break** *(* Beta cut-off *)*
  - **return** α
- alphabeta(origin, depth, -infinity, +infinity)