Implementarea de jocuri in Prolog

Ruxandra Stoean
http://inf.ucv.ro/~rstoean
ruxandra.stoean@inf.ucv.ro

Jocuri

- Prolog-ul ofera modalitati foarte eficiente in implementarea de cadre pentru teoria jocurilor.

- Vom studia doua exemple de implementari Prolog pentru strategiile a doua jocuri:
  - Mastermind.
  - X si 0.

- Mastermind este static din punctul de vedere al utilizatorului, in timp ce X si 0 presupune dinamicitate.

Mastermind

- Exista mai multe variante, cea prezentata aici este una jucata de obicei cu hartia si creionul.

- Avem 2 jucatori:
  - unul static, A, care se gandeste la o secventa de (4) cifre distincte.
  - celalalt dinamic, B, care trebuie sa ghiceasca secventa respectiva.

- Jucatorul B poate sa interogheze pe A, intr-un numar finit de intrebari, asupra apropierii codului sau de secventa adevarata.

Mastermind

- Cand i se prezinta un cod posibil, jucatorul A trebuie sa precizeze:
  - cate cifre se gasesc pe pozitii identice in codul ghicit si in secventa data(numite **boi**)
  - si cate cifre se gasesc in ambele coduri, dar pe pozitii diferite (numite **vaci**).

- Incercati o instanta de Mastermind cu un coleg vecin! ☺

Strategia pentru Mastermind

- Un cod posibil apare prin generarea unei submultimi a multimii celor 10 cifre posibile.
- Se verifica codul nou cu toate cele intrebate anterior – trebuie sa minimizam numarul de interogari ale utilizatorului!
- Daca acest cod nou este **consistent** cu toata informatia de pana acum (nu e inconsistent cu nici unul din codurile anterior intrebate), se intreaba aceasta noua secventa si se retine raspunsul in memoria Prolog-ului.

Strategia - continuare

- Consistenta cu toate codurile anterior chestionate presupune echivalenta numarului de boi si vaci.
- Daca raspunsul utilizatorului contine cei 4 boi – deci toate cifrele sunt corecte si ca entitate si ca pozitie – se anunta faptul ca secventa a fost ghicita si se specifica numarul de pasi intreprins.
- In caz contrar, se repeta procedeul cu o noua generare de cod.

Generarea unui cod nou

- Se specifica faptul ca este vorba de un o submultime de 4 elemente, fiindca avem un cod de 4 cifre distincte:

```
ghiceste(Cod):-Cod = [_, _, _, _],
  selectSublista(Cod, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).
```

- Se creeaza un predicat care genereaza submultimi ale unei multimi.

Generare de submultimi

- Se selecteaza cate un element din lista data.

- Introducerea backtracking-ului la un anumit punct in program va determina generari de submultimi distincte.

```
selectSublista([X|Xs], Ys):-selecteaza(X, Ys, Ys1),
   selectSublista(Xs, Ys1).
selectSublista([], _).

selecteaza(X, [X|Xs], Xs).
selecteaza(X, [Y|Ys], [Y|Zs]):-selecteaza(X, Ys, Zs).
```

Verificarea consistentei cu trecutul

- Un cod nou este inconsistent cu unul intrebat anterior daca numarul de boi si vaci dintre cele doua este diferit de aceste valori ale codului anterior stocat in memoria Prolog-ului:

```
inconsistent(Cod):-intrebare(CodVechi, Boi, Vaci),
   not(potrivire_boi_si_vaci(CodVechi, Cod, Boi,
   Vaci)).
potrivire_boi_si_vaci(CodVechi, Cod, Boi, Vaci):-
   potriviri_exacte(CodVechi, Cod, N1), Boi == N1,
membri_comuni(CodVechi, Cod, N2), Dif is N2 -
   Boi, Vaci ==  Dif.
```

Potriviri exacte, membri comuni

```
potriviri_exacte(L1,L2,N):-potriviri_exacte(L1, L2, 0, N).
potriviri_exacte([], [], N, N).
potriviri_exacte([X|L1], [X|L2], K, N):-K1 is K + 1,
   potriviri_exacte(L1, L2, K1, N).
potriviri_exacte([X|L1], [Y|L2], K, N):-X\=Y,
   potriviri_exacte(L1, L2, K, N).

membri_comuni(Xs, Ys, N):-membri_comuni(Xs, Ys, 0, N).
membri_comuni([],_, N, N):-!.
membri_comuni([X|L1], L2, K, N):-member(X, L2), K1 is K +
   1, membri_comuni(L1, L2, K1, N).
membri_comuni([_|L1], L2, K, N):-membri_comuni(L1, L2, K,
   N).
```

Adresarea unei interogari

- Daca acel cod este consistent cu intrebarile anterioare, se intreaba utilizatorul asupra numarului de boi si vaci fata de secventa sa si se inregistreaza intrebarea + boi/vaci in memoria Prolog:

```
verifica(Cod):-not(inconsistent(Cod)), intreaba(Cod).

intreaba(Cod):-repeat, write('Cati boi si cate vaci sunt in '),
   write(Cod), writeln(' ?'), read(Boi), read(Vaci),
   verifica(Boi, Vaci), !, assert(intrebare(Cod, Boi, Vaci)), Boi
   == 4.

verifica(Boi, Vaci):-integer(Boi), integer(Vaci), Adun is Boi +
   Vaci, Adun =< 4.
```

Anuntul codului final

- Se colecteaza toate inregistrarile memorate de intrebari + numar de boi/vaci si se numara de cate incercari a fost vorba.

```
anunta:-findall(X, intrebare(X, _, _), L), length(L, N), write('Am gasit raspunsul dupa un numar de incercari egal cu '), writeln(N).
```

Programul principal

- Programul se apeleaza cu un parametru in care se va intoarce codul ghicit de calculator.
- Se va sterge memoria unor rulari anterioare si se va executa ciclul ghiceste-verifica.
- Cand codul a trecut testul (boi = 4), se anunta codul si numarul de incercari.

```
:-dynamic intrebare/3.

mastermind(Cod):-retractall(intrebare(_,_,_)),
  ghiceste(Cod), verifica(Cod), anunta.
```

Observatii

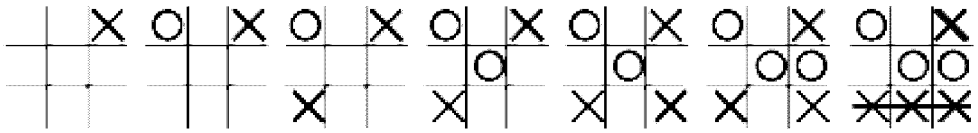
- Este important de remarcat momentul in care backtracking-ul se apeleaza implicit pentru a genera o noua solutie:

```
mastermind(Cod):-retractall(intrebare(_,_,_)), ghiceste(Cod),
    verifica(Cod), anunta.

verifica(Cod):-not(inconsistent(Cod)), intreaba(Cod). % aici
    da fail pentru a merge la urmatorul cod cu
    predicatul ghiceste

inconsistent(Cod):-intrebare(CodVechi, Boi, Vaci),
    not(potrivire_boi_si_vaci(CodVechi, Cod, Boi, Vaci)). %
    aici da fail pentru a trece la urmatoarea intrebare
    stocata in memorie – toate trebuie verificate!
```

X si 0



- Consideram jocul pe 9 patratele -3x.
- Tabla de joc va fi dinamica – fiecare mutare este inregistrata prin retract(vechea configuratie) / assert(noua configuratie).
- Utilizatorul si calculatorul vor actiona fiecare dinamic de aceasta data.
- Vom folosi strategia minimax de la teoria jocurilor, imbunatatita prin reducerea α-β.

Tabla de joc

- Va fi data de predicatul:

(board([_Z1,_Z2,_Z3,_Z4,_Z5,_Z6,_Z7,_Z8,_Z9])

- Vectorul cu 9 pozitii este reprezentarea liniara a matricii tablei de joc.

- Primele 3 pozitii reprezinta prima linie, urmatoarele 3 a doua si ultimele 3, ultima linie.

Tabla de joc

- Algoritmul incepe prin a inregistra tabla de joc goala:

```
assert(board([_Z1,_Z2,_Z3,_Z4,_Z5,_Z6,_Z7,_Z8,_Z
  9]))
```

- Astfel, toate pozitiile sale sunt neinitializate.

- Cand utilizatorul sau calculatorul marcheaza x, respectiv 0, pozitia respectiva este initializata corespunzator.

Marcarea unei pozitii

- O pozitie (X, Y) considerata de jucator - utilizator (x) sau calculator (o) - pentru marcare, se modifica corespunzator prin predicatul:

```
mark(Player, [X|_],1,1) :- var(X), X=Player.
mark(Player, [_,X|_],1,2) :- var(X), X=Player.
mark(Player, [_,_,X|_],1,3) :- var(X), X=Player.
mark(Player, [_,_,_,X|_],2,1) :- var(X), X=Player.
mark(Player, [_,_,_,_,X|_],2,2) :- var(X), X=Player.
mark(Player, [_,_,_,_,_,X|_],2,3) :- var(X), X=Player.
mark(Player, [_,_,_,_,_,_,X|_],3,1) :- var(X), X=Player.
mark(Player, [_,_,_,_,_,_,_,X|_],3,2) :- var(X), X=Player.
mark(Player, [_,_,_,_,_,_,_,_,X|_],3,3) :- var(X), X=Player.
```

Inregistrarea unei mutari in memorie

- Efectul mutarii (X, Y) a jucatorului - utilizatorul (x) sau calculatorul (o) – duce la retragerea vechii configuratii a tablei de joc si asertarea noii forme.
- Prin faptul ca folosim o tabla cu pozitii neinitializate, simbolul (x sau o) este automat unificat cu variabila de pe pozitia data de mutare:

```
record(Player,X,Y) :- retract(board(B)),
                      mark(Player,B,X,Y),
                      assert(board(B)).
```

Afisarea tablei de joc

```
showBoard :-
   board([Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8,Z9]),write('
   '),mark(Z1),write(' '),mark(Z2),write('
   '),mark(Z3),nl,write(' '),mark(Z4),write('
   '),mark(Z5),write(' '),mark(Z6),nl,write('
   '),mark(Z7),write(' '),mark(Z8),write('
   '),mark(Z9),nl.

mark(X) :- var(X),write('#').
mark(X) :- not(var(X)),write(X).
```

Starea terminala de castig

- Jucatorul P castiga jocul daca tabla se afla in una din urmatoarele situatii:

```
win([Z1,Z2,Z3|_],P) :- Z1==P, Z2==P, Z3==P.
win([_,_,_,Z1,Z2,Z3|_],P) :- Z1==P, Z2==P, Z3==P.
win([_,_,_,_,_,_,Z1,Z2,Z3],P) :- Z1==P, Z2==P, Z3==P.
win([Z1,_,_,Z2,_,_,Z3,_,_],P) :- Z1==P, Z2==P, Z3==P.
win([_,Z1,_,_,Z2,_,_,Z3,_],P) :- Z1==P, Z2==P, Z3==P.
win([_,_,Z1,_,_,Z2,_,_,Z3],P) :- Z1==P, Z2==P, Z3==P.
win([Z1,_,_,_,Z2,_,_,_,Z3],P) :- Z1==P, Z2==P, Z3==P.
win([_,_,Z1,_,Z2,_,Z3,_,_],P) :- Z1==P, Z2==P, Z3==P.
```

Starea terminala de joc indecis

- Daca toata tabla se completeaza – nu mai am variabile – inseamna ca jocul este blocat.

```
complete([]).
complete([X|Rest]):-not(var(X)), complete(Rest).
```

- Conditiile de terminare sunt asadar:

```
stop:-board(B), win(B,x), write('Tu ai castigat! Felicitari!').
stop:-board(B), win(B,o), write('Eu am castigat!').
stop:-board(B), complete(B), write('Scor indecis!').
```

Liniile deschise ale unui jucator

- Un jucator va cauta, intotdeauna, sa mute pe o linie (coloana, diagonala) care sa il avantajeze spre castig.
- Acestea sunt liniile deschise.
- Aceste linii (coloane, diagonale) sunt acelea care fie nu contin niciun simbol, fie contin simbol jucatorului.

Liniile deschise ale unui jucator

```
open([Z1,Z2,Z3|_],Player) :- (var(Z1) | Z1 == Player),(var(Z2) | Z2 ==
    Player),(var(Z3) | Z3 == Player).
open([_,_,_,Z1,Z2,Z3|_],Player) :- (var(Z1) | Z1 == Player),(var(Z2) | Z2 ==
Player), (var(Z3) | Z3 == Player).
open([_,_,_,_,_,_,Z1,Z2,Z3],Player) :- (var(Z1) | Z1 == Player),(var(Z2) | Z2
== Player), (var(Z3) | Z3 == Player).
open([Z1,_,_,Z2,_,_,Z3,_,_],Player) :- (var(Z1) | Z1 == Player),(var(Z2) | Z2
== Player), (var(Z3) | Z3 == Player).
open([_,Z1,_,_,Z2,_,_,Z3,_],Player) :- (var(Z1) | Z1 == Player),(var(Z2) | Z2
== Player), (var(Z3) | Z3 == Player).
open([_,_,Z1,_,_,Z2,_,_,Z3],Player) :- (var(Z1) | Z1 == Player),(var(Z2) | Z2
== Player), (var(Z3) | Z3 == Player).
open([Z1,_,_,_,Z2,_,_,_,Z3],Player) :- (var(Z1) | Z1 == Player),(var(Z2) | Z2
== Player), (var(Z3) | Z3 == Player).
open([_,_,Z1,_,Z2,_,Z3,_,_],Player) :- (var(Z1) | Z1 == Player),(var(Z2) | Z2
== Player), (var(Z3) | Z3 == Player).
```

Mutarile utilizatorului si calculatorului

- Utilizatorul (x) muta primul, apoi calculatorul (o), s.a.m.d. pana cand se ajunge la una din conditiile terminale.

```
play:- init, showBoard, repeat, human, computer, stop, !.

human:-write('Mutarea ta: '), read(X), read(Y), human(X,Y).
human(X,Y) :- record(x,X,Y).

computer :- board(B), (not(complete(B))), alpha_beta(o,2,B,-
   200,200,(X,Y),_Value),record(o,X,Y), showBoard.
computer.
```

Strategia calculatorului – minimax cu reducerea α-β

- Pentru jocul calculatorului, vom implementa strategia minimax, prin care se presupune ca, avand mai multe posibilitati de mutare, utilizatorul va lua cea mai buna decizie pentru el, adica alegerea de utilitate minima pentru calculator.
- Scopul calculatorului este sa faca mutarea care maximizeaza valoarea configuratiei dupa ce adversarul a facut cea mai buna mutare a sa, adica aceea care ii minimizeaza valoarea acestuia.

Strategia calculatorului – minimax cu reducerea α-β

- Anticipatia se face cu cate mutari sunt posibile in avans in functie de restrictiile computationale.
- Se ajunge la nodurile frunza a caror utilitate se calculeaza si utilitatile starilor parinte se stabilesc prin strategia minimax.
- Se va aplica reducerea α-β pentru eficientizarea calculului, atunci cand parti ale arborelui de deductie sunt redundante.

Evaluarea unei stari (nod) frunza

- Performanta starii (nodului) frunza este data, din punctul de vedere al jucatorului, de numarul de pozitii deschise jocului pentru simbolul sau minus numarul de pozitii deschise jocului pentru adversar.

- Daca x (utilizatorul) castiga, utilitatea starii este -100, daca o (calculatorul) castiga, utilitatea este 100.

Evaluarea pentru starea curenta a jocului

```
value(Board,100) :- win(Board,o), !.
value(Board,-100) :- win(Board,x), !.
value(Board,E) :- findall(o,open(Board,o),MAX),
   length(MAX,Emax),
   findall(x,open(Board,x),MIN),
   length(MIN,Emin), E is Emax - Emin.
```

Strategia calculatorului

- α este valoarea cele mai bune (adica cea mai mare) alegeri gasita pana la momentul curent la orice punct de-a lungul unui drum pentru MAX (o).
- β este definit in mod similar pentru MIN, adica cea mai mica valoare gasita la orice punct de-a lungul unui drum pentru MIN (x).
- De-a lungul reducerii, cautarea se continua actualizand valorile pentru α si β si nemaiparcurgand acei subarbori atunci cand valoarea unei mutari este minimul sau maximul posibil.

Strategia calculatorului

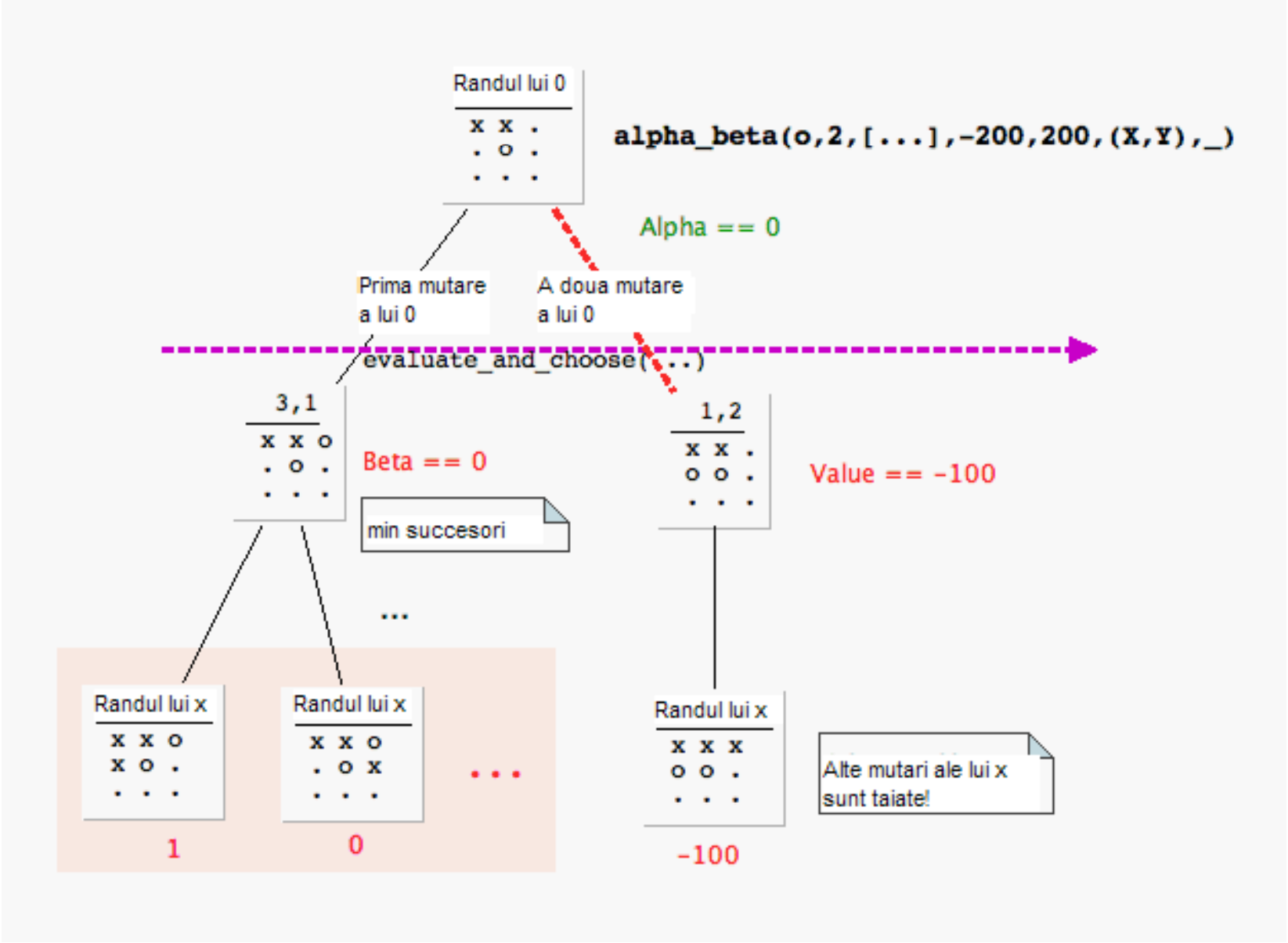
- Mutarile posibile pentru cei doi jucatori se codifica prin liste, unde pozitia dorita a tablei de joc se instantiaza.
- Pentru a schimba pe rand nivelurile - discutam cand de minimizare, cand de maximizare - valorile pentru mutari, α si β se inverseaza ca semn.
- Adancimea cautarii este considerata 2.

Mutarea dintr-o pozitie in alta pe tabla de joc

- Daca jucatorul P face mutarea (X, Y), urmatoarea configuratie a tablei este data de a doua lista:

```
move(P,(1,1),[X1|R],[P|R]) :- var(X1).
move(P,(1,2),[X1,X2|R],[X1,P|R]) :- var(X2).
move(P,(1,3),[X1,X2,X3|R],[X1,X2,P|R]) :- var(X3).
move(P,(2,1),[X1,X2,X3,X4|R],[X1,X2,X3,P|R]) :- var(X4).
move(P,(2,2),[X1,X2,X3,X4,X5|R],[X1,X2,X3,X4,P|R]) :- var(X5).
move(P,(2,3),[X1,X2,X3,X4,X5,X6|R],[X1,X2,X3,X4,X5,P|R]) :-
    var(X6).
move(P,(3,1),[X1,X2,X3,X4,X5,X6,X7|R],[X1,X2,X3,X4,X5,X6,P|R
    ]) :-var(X7).
move(P,(3,2),[X1,X2,X3,X4,X5,X6,X7,X8|R],[X1,X2,X3,X4,X5,X6,
    X7,P|R]) :-var(X8).
move(P,(3,3),[X1,X2,X3,X4,X5,X6,X7,X8,X9|R],[X1,X2,X3,X4,X5,
    X6,X7,X8,P|R]) :-var(X9).
```

Vizualizare



Reducerea α-β

```
alpha_beta(_Player,0,Position,_Alpha,_Beta,_NoMove,Value) :-
    value(Position,Value).

alpha_beta(Player,D,Position,Alpha,Beta,Move,Value) :- D > 0,
    findall((X,Y),mark(Player,Position,X,Y),Moves),
Alpha1 is -Beta, Beta1 is -Alpha, D1 is D-1,
evaluate_and_choose(Player,Moves,Position,D1,Alpha1,Beta1,nil,(Move,
    Value)).

evaluate_and_choose(Player,[Move|Moves],Position,D,Alpha,Beta,Reco
    rd,BestMove):- move(Player,Move,Position,Position1),
    other_player(Player,OtherPlayer),
alpha_beta(OtherPlayer,D,Position1,Alpha,Beta,_OtherMove,Value),
Value1 is -Value,
    cutoff(Player,Move,Value1,D,Alpha,Beta,Moves,Position,Record,Best
    Move).

evaluate_and_choose(_Player,[],_Position,_D,Alpha,_Beta,Move,(Mov
    e,Alpha)).
```

Reducerea α-β - continuare

```
cutoff(_Player,Move,Value,_D,_Alpha,Beta,_Moves,_Position,_Record,(Move,Value)):- Value >= Beta, !.

cutoff(Player,Move,Value,D,Alpha,Beta,Moves,Position,_Record,BestMove) :-Alpha < Value, Value < Beta, !,
evaluate_and_choose(Player,Moves,Position,D,Value,Beta,Move,BestMove).

cutoff(Player,_Move,Value,D,Alpha,Beta,Moves,Position,Record,BestMove) :-Value =< Alpha, !,
evaluate_and_choose(Player,Moves,Position,D,Alpha,Beta,Record,BestMove).
```

Concluzii

- Incercati sa jucati contra calculatorului in cele doua programe.
- Strategiile celor doua jocuri prezentate nu sunt foarte performante – crearea de strategii optime pentru jocul calculatorului o problema de mare actualitate!

Pe saptamana viitoare!