Sergi Magret Goy i Ibrahima Sory Diallo Diallo

MUTA

Per realitzar aquest exercici el que hem plantejat és anar provant les paraules de la "base de dades", però només s'acceptaran com a candidates les que tinguin una lletra de diferència amb la paraula original. Un cop acceptada aquesta paraula com a candidata es provarà de trobar un camí de paraules entre la nova paraula acceptada i la paraula final.

Per tal de que no vagi provant paraules que s'allunyen de la paraula final i reduir més la cerca, primer intentarem les paraules que s'apropen més a la paraula final, és a dir, que tenen més paraules semblants amb la final, si té més lletres diferents que la paraula anterior ja no es deixa continuar. Si no es troba cap paraula que s'apropi a la paraula final, se'n busca una que al menys tingui el mateix nombre de lletres diferents.

Per mirar la diferència entre dos paraules utilitzem el predicat `string_chars()`, on el segon paràmetre és una llista amb els caràcters de la paraula, i després utilitzem el predicat `diferencia()` per mirar quantes lletres diferents hi ha entre les dues paraules.

```
% diferencia(X, Y, N). Es compleix quan N és el nombre d'elements diferents entre X i Y. Pressuposa que les dues llistes tenen la mateixa mida.
diferencia([], [], 0).
diferencia([X|XS], [Y|YS], N):- X == Y, diferencia(XS, YS, N), !.
diferencia([X|XS], [Y|YS], N):- X \== Y, diferencia(XS, YS, Np), N is Np+1, !.
```

Per fer el predicat `muta()` el primer que fem és mirar la diferència entre les dues paraules i cridar a `i_muta()` amb les dues paraules passades, una llista buida i la diferència que hi ha entre les dues paraules.

```
muta(PI, PF):- string_chars(PI, CPI), string_chars(PF, CPF), diferencia(CPI, CPF, N), i_muta(PI, PF, [], N), !.
```

Sergi Magret Goy i Ibrahima Sory Diallo Diallo


```
% i_muta(PI, PF, LP, LastDif). Es compleix quan PI és la paraula inicial de la mutació, PF la paraula final de la
% mutació, LP és la llista de paraules per les que ja s'ha passat i LastDif és la última diferència entre PI i PF
i_muta(PI, PF, LP, _):- string_chars(PI, CPI),
                        string_chars(PF, CPF),
                        diferencia(CPI, CPF, N),
                        N =:= 0, % Si la diferència és 0 significa que s'ha arribat a la paraula final
                        reverse([PF|LP], LF), length(LF, Llarg),
                        write("Canvis: "), writeln(LF), write("Llargada: "), writeln(Llarg).


% Es pot considerar com un "if" amb la condició de que la nova paraula tingui diferència 1 i la diferència amb la paraula
% final sigui més petita (ens apropem a ella)
i_muta(PI, PF, LP, LastDif):- string_chars(PI, CPI),
                              w4(X),
                              not(member(X, LP)), % Assegurar que la nova paraula triada no forma part de la llista de
                              string_chars(X, CX), % paraules per la que s'ha passat, sinó segurament s'entraria en un
                              diferencia(CPI, CX, N), % bucle entre dues paraules
                              N =:= 1,
                              string_chars(PF, CPF),
                              diferencia(CX, CPF, Np),
                              Np < LastDif, % Assegurar que la diferencia entre la nova paraula i la final és mes petita
                              append([PI], LP, L), % Es dona la paraula com a bona i s'afegeix a la llista
                              i_muta(X, PF, L, Np). % Finalment es torna a cridar al predicat amb la paraula acceptada
                                                    % com a la nova paraula inicial, la mateixa paraula final, la nova
                                                    % llista i la nova diferència entre paraules
```

```
% Es pot considerar com un "if" amb la condició de que la nova paraula tingui diferència 1 i la diferència amb la paraula
% final sigui igual a l'anterior paraula, el predicat és idèntic a l'anterior excepte LastDif =:= Np
i_muta(PI, PF, LP, LastDif):- string_chars(PI, CPI),
                              w4(X),
                              not(member(X, LP)),
                              string_chars(X, CX),
                              diferencia(CPI, CX, N),
                              N =:= 1,
                              string_chars(PF, CPF),
                              diferencia(CX, CPF, Np),
                              LastDif =:= Np,
                              append([PI], LP, L),
                              i_muta(X, PF, L, Np).
```

La solució que dóna aquest predicat no és la millor, és podria considerar un algorisme *greedy*, ja que sempre intenta apropar-se el màxim possible cap a la paraula final.

Per fer un algorisme que trobi la millor solució només se'ns ha ocorregut aprofitar el predicat `findall()` per trobar totes les solucions i després mirar quina és la millor comptant el nombre de passes que es fa. El que si hem fet és posar la restricció de que un cop es trobi un camí, cap altre camí que es trobi podrà ser major que aquest primer camí trobat entre les dues paraules, per exemple si es troba un camí entre les dues paraules de 6, qualsevol que sigui més gran de 6 serà automàticament descartat.

Aquest predicat l'hem anomenat `fmuta()` i utilitza el `trobaUn()` que bàsicament és el mateix que `i_muta()`, excepte que en aquest cas es "retorna" la llista trobada i s'utilitza per determinar la màxima llargada que podran tenir la resta de llistes. Després s'utilitza el `trobaMesCurt()` juntament amb el `findall()` per trobar totes les més curtes, finalment el `mesCurta()` per trobar la més curta de totes les retornades per el `findall()`.

```
% mesCurta(X, LlCurta). Es compleix quan X és una llista de llistes i LlCurta és la llista més curta de X
mesCurta([X], X).
mesCurta([X,Y|XS], LlCurta):- length(X, LX), length(Y, LY), LX =< LY, mesCurta([X|XS], LlCurta).
mesCurta([X,Y|XS], LlCurta):- length(X, LX), length(Y, LY), LY < LX, mesCurta([Y|XS], LlCurta).


fmuta(PI, PF):- string_chars(PI, CPI),
                string_chars(PF, CPF),
                diferencia(CPI, CPF, Dif),
                trobaUn(PI, PF, [], Dif, LFTmp), % Es troba la primera solució
                length(LFTmp, MaxLlarg), % I es fa servir la llargada per la resta de possibles llistes
                findall(LF, trobaMesCurt(PI, PF, [], Dif, MaxLlarg-1, LF), Llistes),
                mesCurta([LFTmp|Llistes], MC),
                write("Canvis: "), writeln(MC), write("Llargada:"), length(MC, X), writeln(X), !.


% trobaUn(PI, PF, LP, LastDif, LF). Es compleix quan PI és la paraula inicial de la mutació, PF la paraula final de la
% mutació, LP és la llista de paraules per les que ja s'ha passat, LastDif és la última diferència entre PI i PF i LF és
% la llista final trobada.
trobaUn(PI, PF, LP, _, LF):- string_chars(PI, CPI),
                             string_chars(PF, CPF),
                             diferencia(CPI, CPF, N),
                             N =:= 0,
                             reverse([PF|LP], LF), !.
```

```
trobaUn(PI, PF, LP, LastDif, LF):- string_chars(PI, CPI),
                            w4(X),
                            not(member(X, LP)),
                            string_chars(X, CX),
                            diferencia(CPI, CX, N),
                            N =:= 1,
                            string_chars(PF, CPF),
                            diferencia(CX, CPF, Np),
                            Np < LastDif, % Assegures que la diferencia entre la nova paraula i la final sigui mes petita
                            append([PI], LP, L),
                            trobaUn(X, PF, L, Np, LF).

trobaUn(PI, PF, LP, LastDif, LF):- string_chars(PI, CPI),
                            w4(X),
                            not(member(X, LP)),
                            string_chars(X, CX),
                            diferencia(CPI, CX, N),
                            N =:= 1,
                            string_chars(PF, CPF),
                            diferencia(CX, CPF, Np),
                            LastDif =:= Np,
                            append([PI], LP, L),
                            trobaUn(X, PF, L, Np, LF).
```

Sergi Magret Goy i Ibrahima Sory Diallo Diallo


```
% trobaMesCurt(PI, PF, LP, LastDif, MaxLlarg LF). Es compleix quan PI és la paraula inicial de la mutació, PF la paraula
% final de la mutació, LP és la llista de paraules per les que ja s'ha passat, LastDif és la última diferència entre PI i
% PF, MaxLlarg és la màxima llargada que pot tenir la llista i LF és la llista final trobada.
trobaMesCurt(PI, PF, LP, _, MaxLlarg, LF):- string_chars(PI, CPI),
                            string_chars(PF, CPF),
                            diferencia(CPI, CPF, N),
                            N =:= 0,
                            length(LP, Len),
                            Len < MaxLlarg, % Assegurar que la nova llista és més petita que el màxim
                            reverse([PF|LP], LF).

trobaMesCurt(PI, PF, LP, LastDif, MaxLlarg, LF):- string_chars(PI, CPI),
                            w4(X),
                            not(member(X, LP)),
                            string_chars(X, CX),
                            diferencia(CPI, CX, N),
                            N =:= 1,
                            string_chars(PF, CPF),
                            diferencia(CX, CPF, Np),
                            Np < LastDif, % Assegures que la diferencia entre la nova paraula i la final sigui mes petita
                            length(LP, Len),
                            Len < MaxLlarg, % Assegurar que la nova llista és més petita que el màxim
                            append([PI], LP, L),
                            trobaMesCurt(X, PF, L, Np, MaxLlarg, LF).
```

```
trobaMesCurt(PI, PF, LP, LastDif, MaxLlarg, LF):- string_chars(PI, CPI),
                            w4(X),
                            not(member(X, LP)),
                            string_chars(X, CX),
                            diferencia(CPI, CX, N),
                            N =:= 1,
                            string_chars(PF, CPF),
                            diferencia(CX, CPF, Np),
                            LastDif =:= Np,
                            length(LP, Len),
                            Len < MaxLlarg, % Assegurar que la nova llista és més petita que el màxim
                            append([PI], LP, L),
                            trobaMesCurt(X, PF, L, Np, MaxLlarg, LF).
```

La idea del `trobaMesCurt()` era que si trobava una llista més curta, actualitzés el MaxLlarg per així anar reduint el màxim de la llista i reduir el nombre d'opcions però no hem aconseguit fer aquesta actualització perquè això és molt idea de variable i no unificava.

Exemples d'execució:

```
?- muta("wish", "hope").

Canvis: [wish,wise,wipe,pipe,pope,hope]

Llargada: 6

true.


?- muta("wilt","lang").
```

```
Canvis: [wilt,lilt,lint,lent,lena,lana,lang]
Llargada: 7
true.

?- muta("font","slub").
Canvis: [font,fond,bond,band,baud,saud,saul,shul,shun,shut,slut,slub]
Llargada: 12
true.

?- fmuta("what","when").
Canvis: [what,whet,when]
Llargada: 3
true.

?- fmuta("acta","acid").
Canvis: [acta,acts,aces,aced,acid]
Llargada: 5
true.
```

BATUTS

La base de dades sobre la que estem treballant és la següent:

```
establiment( % -> Promig 2.66666
    best_batuts, [alan,john,mary],
    [
        batut(berry,    [orange, blueberry, strawberry], 2),
        batut(tropical, [orange, banana, mango, guava], 3),
        batut(blue,     [banana, blueberry], 3)
    ]
).

establiment( % -> Promig 2.5
    all_batuts, [keith,mary],
    [
        batut(pinacolada,   [orange, pineapple, coconut], 2),
        batut(green,        [orange, banana, kiwi], 5),
        batut(purple,       [orange, blueberry, strawberry], 2),
        batut(smooth,       [orange, banana, mango], 1)
    ]
).

establiment( % -> Promig 2.2
    batuts_galore, [heath,john,michelle],
    [
        batut(combo1, [strawberry, orange, banana], 2),
        batut(combo2, [banana, orange], 5),
        batut(combo3, [orange, peach, banana], 2),
        batut(combo4, [guava, mango, papaya, orange], 1),
```

```
        batut(combo5, [grapefruit, banana, pear], 1)
    ]
).

establiment( % -> Promig 3.0
    roses_batuts, [marc,roger,carmen],
    [
        batut(fresc1, [peach, lemon, milk], 2.5),
        batut(fresc2, [cherry, orange], 3),
        batut(fresc3, [apple, strawberry, orange, milk], 4),
        batut(fresc4, [chocolate, banana, milk], 2),
        batut(fresc5, [watermelon, pear, yogurt, peach], 3.5)
    ]
).

establiment( % -> Promig 1.94
    batuts_barats, [pep,laia,rosana],
    [
        batut(fresc1, [peach, lemon, milk], 2),
        batut(fresc2, [cherry], 1),
        batut(fresc3, [strawberry, orange, milk], 1.2),
        batut(fresc4, [chocolate, milk, strawberry], 1.5),
        batut(fresc5, [watermelon, pear, yogurt, peach], 4)
    ]
).

establiment( % -> Promig 3.54
    weird_batuts, [josep,alex,sergi,cristina],
    [
```

```
        batut(weird1, [peach, lemon, milk, cherry], 3),
        batut(weird2, [cherry, watermelon, chocolate, yogurt], 2.7),
        batut(weird3, [strawberry, orange, milk, peach], 2.5),
        batut(weird4, [chocolate, milk, strawberry, apple], 3.6),
        batut(weird5, [watermelon, pear, yogurt, peach, lemon, milk], 6.2),
        batut(weird6, [orange, pear, peach, lemon, yogurt], 3.6),
        batut(weird7, [cherry, orange, milk, strawberry, lemon], 3.2)
    ]
).


% mesDe(+N, E). Es satisfà si l'establiment E te més de N batuts
mesDe(N,E):- establiment(E,_,L), length(L,Len), Len > N.


% elFa(B, E). Es satisfà si l'establiment E fa el batut B
elFa(B,E):- establiment(E,_,L), member(batut(B,_,_),L).


% ratio(E, R). Es satisfà si l'establiment E te un ratio d'empleats per batuts d'R
ratio(E,R):- establiment(E, Empleats, Batuts), length(Empleats, LEmp), length(Batuts, LBat), R is LEmp/LBat.


% suma(LL, M). Es satisfà quan M és la suma dels preus de la llista de batuts LL.
suma([],0).
suma([batut(_,_,P)|LS], M):- suma(LS, Z), M is Z + P.

% promig(E, P). Es satisfà si el promig del preu dels batuts a l'establiment E és P
promig(E,P):- establiment(E,_,L), length(L,Len), suma(L,M), Z is M/Len, Z = P.
```

```
% mesBarat(E). Es satisfà si l'establiment E te els batuts més barats en promig
mesBarat(E):- establiment(E, _, _), promig(E, Pe), establiment(X, _, _), X \= E, promig(X, Px), Pe > Px, !, fail. % A la mínima que n'hi hagi un de mes barat fallar
mesBarat(E):- establiment(E, _, _). % Si provant tots els altres establiment no s'ha complert l'anterior, aquesta retornarà true, NO ES POT CANVIAR D'ORDRE
```

Per fer el `trobaBatuts()` hem utilitzat els predicats auxiliars descrits a continuació:

```
% notInList(LX, D). Es satisfà quan algun dels elements de la llista LX està a la llista D
notInList([],_).
notInList([L|LX],D):- notInList(LX,D), not(member(L,D)).

% inList(LX, D). Es satisfà quan tots els elemnts de la llista LX estan a la llista D
inList([],_).
inList([L|LX],D):- inList(LX,D), member(L,D).

% fillList(E, Lb, D, I, L). Es satisfà quan L és una llista formada per parelles (E, NomBatut) que contenen
% els ingredients de la llista D i cap de la llista I. Lb és la llista de batuts de l'establiment E.
% Fa un recorregut de tots els batuts de l'establiment i comprovant si el batut compleix els requisits D i I.
fillList(_,[],_,_,[]).
fillList(E,[batut(M,List,_)|Lb],D,I,[E,M|L]):- fillList(E,Lb,D,I,L), notInList(List,I), inList(D,List), !.
fillList(E,[batut(_,_,_)|Lb],D,I,L):- fillList(E,Lb,D,I,L).

% recurse(Establiments,L,D,I). Es satisfà quan L és una llista formada per llistes de parells (Establiment, NomBatut)
% de la llista d'establiments Establiments que contenen els ingredients que es demanan a la llista D
% i cap dels que es demanen a la llista I
recurse([],[],_,_).
recurse([X|Establiments],[Z|L],D,I):- recurse(Establiments,L,D,I), establiment(X,_,Lb), fillList(X,Lb,D,I,Z).
```

Finalment per al `trobaBatuts()` el que fem és construir una llista amb tots els Establiments i després recórrer aquesta llista per anar comprovant quins dels batuts de cada establiment compleixen les condicions

```
% trobaBatuts(L,D,I). Es satisfà si L és una llista formada pels parells (Establiment,NomBatut)
% de tots els establiments-batuts que contenen els ingredients que es demanen a la llista D
% i cap dels que es diuen a la llista I.
trobaBatuts(L,D,I):- findall(Establiment, establiment(Establiment,_,_), List),
                     recurse(List,Z,D,I),
                     flatten(Z,L). % Converteix llista de llistes en una unica llista
```

Exemples d'execució

```
?- mesDe(5, X).

X = weird_batuts.


?- elFa(fresc1, X).

X = roses_batuts ;

X = batuts_barats ;

false.


?- elFa(X, roses_batuts).

X = fresc1 ;

X = fresc2 ;
```

```
X = fresc3 ;
X = fresc4 ;
X = fresc5.

?- trobaBatuts(X, [peach,lemon],[orange]).
X = [roses_batuts, fresc1, batuts_barats, fresc1, weird_batuts, weird1, weird_batuts, weird5] ;
false.
?- trobaBatuts(X, [peach,lemon],[orange,cherry]).
X = [roses_batuts, fresc1, batuts_barats, fresc1, weird_batuts, weird5] ;
false.
```

4LINIA

Per realitzar aquest exercici el que hem fet primerament es definir l'estructura de dades que es de tipus Llista de Llistes creat dinàmicament, on cada fila correspon a una columna en el tauler i després les columnes són les files del tauler.

També hem definit una llista dinàmica per anar guardant les possibles jugades del tauler, així en cada moment podrem accedir a la llista sense haver-ho de re-calcular.

```
:- set_prolog_flag(answer_write_options,[max_depth(0)]).
% Predicats que es satifan automaticament quan s'inicia el programa.
:-dynamic board/1. % Iniciem un tauler dynamic, que anirem modifican durant el joc.
:-retractall(board(_)). % Eliminem el tauler si ja existia.
:-dynamic moves/1. % Predicat que anira guardant les possibles jugades del tauler en tot moment.
:-retractall(moves(_)).
:-assertz(moves([])).
:-assertz(board([/*A*/ ['X','X','_','_','_','_'],
                 /*B*/ ['X','X','_','_','_','_'],
                 /*C*/ ['O','X','O','_','_','_'],
                 /*D*/ ['X','O','_','_','_','_'],
                 /*E*/ ['X','_','_','_','_','_'],
                 /*F*/ ['_','_','_','_','_','_'],
                 /*G*/ ['X','_','_','_','_','_'] ])).
```

```
% Es satisfà el predicat initBoard() si s'ha creat dinàmicament el tauler mitjançant els mètodes
% retract() per eliminar el tauler ja existent, i després amb el predicat <assertz()> crear-ne un de nou buit.
initBoard():-retract(board(_)),assertz(board([
    /*A*/ ['_','_','_','_','_','_','_'],  %A
    /*B*/ ['_','_','_','_','_','_','_'],  %B
    /*C*/ ['_','_','_','_','_','_','_'],  %C
    /*D*/ ['_','_','_','_','_','_','_'],  %D
    /*E*/ ['_','_','_','_','_','_','_'],  %E
    /*F*/ ['_','_','_','_','_','_','_'],  %F
    /*G*/ ['_','_','_','_','_','_','_']])).%G

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% PREDICATS QUE MOSTREN PER PANATALLA EL TAULER   %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Es satisfà el predicat display_game() si es mostra correctament el tauler per
% pantalla amb el format requerit.
display_game():-board(Board), write("   A  B  C  D  E  F  G"), nl, display(Board,6).
display_game.


% Grid -> El tauler
% N -> Nombre de columnes
% Es satisfa si es mostra totes les columnes del tauler començant per l'element en la posició N
% de cada fila.
display(Grid,N) :-
    maplist(nth1(N),Grid, Column), % Per cada fila del tauler guarda l'element N en la llista Columna.
    write(N),disp(Column),nl,fail.
```

```
display(Grid,N) :-
    N > 0,
    N1 is N-1,
    display(Grid,N1).

% L -> Llista d'elements.
% Es satisfà si es mostra tots els elements de la llista L.
disp([]).
disp([X|L]):-write("  "),write(X),disp(L),nl,fail.


% opponent/computer -> Jugador huma/ ordinador
% Es satisfà si s'ha anunciat correctament el resultat.
announceResult(opponent):- write("YOU WON THE GAME!"),nl.
announceResult(computer):- write("THE COMPUTER WON THE GAME!"),nl.

% Move -> jugada
% opponent/computer -> Jugador huma/ ordinador
% Es satisfa si s'ha escrit correctament el movie.
dispMove(Move,opponent):- nth1(1,Move,I),nth1(I,['A','B','C','D','E','F','G'],E),write("tria: "),write(E),nl.
dispMove(Move,computer):- nth1(1,Move,I),nth1(I,['A','B','C','D','E','F','G'],E),write("robot: "),write(E),nl.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%&&&&&&& PREDICATS PER COMPROBAR SI HA ACABAT EL JOC  %%%%%%%&&&&&&
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Board -> Tauler
% opponent/computer -> Jugador Huma / Ordinador
% Es satisfà si el jugador humà o l'ordinador aconsegueix alinear 4 peces horizontalment;verticalment;
% o diagonalment en el tauler.
game_over(Board,opponent,_):-checkHori(Board,'X',7); checkVert(Board,'X',6); checkdiagonals('X',Board),!.
game_over(Board,computer,_):-checkHori(Board,'O',7); checkVert(Board,'O',6); checkdiagonals('O',Board),!.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% PREDICATS PER ESCOLLIR ELS MOVIMENTS  %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Board -> El tauler
% E -> '_' codifica el una posicio buida en el tauler
% X -> fila
% Y -> columna
% Es satisfà si la posició X, Y del tauler es una posició buida.
legal(Board,E,X,Y):-member(E,['A','B','C','D','E','F','G']),
            indexOf(['A','B','C','D','E','F','G'],E,T), X is T+1, % Trobar l'index de l'element E i guardar-lo a T
            nth1(X, Board, Row),
            findFirstEmpty(Row,'_',1,Y),!.  % Trobar la primera posició buida de la fila.
```

```
% Es satisfa el predicat possibles_moves , si en encara es poden
% fer moviments en el tauler, i posteriorment unificar els moviments en la
% llista dinàmica <moves/1>
possibles_moves(_):- assertz(moves([])),
                     board(Board),between(1,7,I), nth1(I,Board,Row),
                     findFirstEmpty(Row,'_',1,Index), % Troba la primera posició buida de la fila
                     moves(Moves), % Agafar els moviments vàlids en el tauler
                     retract(moves(_)),
                     append([[I,Index]],Moves,List), % Afageix el nou moviment trobat en la llista dinàmica de moviments
                     assertz(moves(List)),fail. % Guarda la llista modificada amb el nou moviment.
possibles_moves(L):- moves(L).


% opponent -> Jugador huma
% Move -> Jugada que fara el judador huma amb posicio [x,y]
% Es satisfa si el jugador huma ha escollit una jugada valida.
choose_move(opponent,Move):- board(Board),write("tria :"),repeat,
                             get_char(E), % Demanem al jugador la posició on vol tirar i guardem el resultat a E
                             legal(Board,E,X,Y),Move=[X,Y|[]],!. % Comprovem si la jugada es vàlida i posteriorment
unificar la jugada a Move

% computer -> Ordinador
% Move -> Paràmetre on unificarem la jugada escollida per l'ordinador
% Es satisfà si l'ordinador acconsegueix fer una jugada correctament
choose_move(computer,Move):- possibles_moves(Moves),winingMove(Moves,Move,computer),!,dispMove(Move,computer).
choose_move(computer,Move):- possibles_moves(Moves),winingMove(Moves,Move,opponent),!,dispMove(Move,computer). % BLOCK OP
PONENT FROM WINNING
choose_move(computer,Move):- possibles_moves(Moves),length(Moves,N),random_between(1,N,I),nth1(I,Moves,Move),!,dispMove(M
ove,computer),nl.
```

```
% Moves -> Llista de possibles moviments en el tauler
% Move -> Variable on unificarem la jugada guanyadora per l'ordinador o que bloqueja l'opponent de guanyar.
% Player -> Jugador
% Es satisfà si es troba jugada que fagi guanyar l'ordinador altrament que eviti l'oponent guanyar
% i si no es compleixent tots els objectius anteriors s'escollira una jugada aleatòria dels possibles moviments.
winingMove([],_,_):-!,fail.

% Simulem una jugada amb un dels moviments de la llista de moviments i seguidament comprobar si es una jugada guanyadora
% i en cas que ho sigui acabem la cerca i unifiquem la jugada a Move
winingMove([WiningMove|_],Move,Player):- fakeMove(WiningMove,Player,Result),game_over(Result,Player,_),Move=WiningMove,!.
winingMove([_|Moves],Move,Player):- winingMove(Moves,Move,Player),!.

% Move -> Juga que volem simular
% computer/opponent -> Judador amb qui volem fer la simulació
% Result -> Estat del tauler després de fer la simulació amb el jugador computer/opponent
% Es satisfà si s'ha pogut dur a terme la jugada
fakeMove([X,Y|_],computer,Result):-board(Board),replace_row_col(Board,X,Y,'O',Result),!.
fakeMove([X,Y|_],opponent,Result):-board(Board),replace_row_col(Board,X,Y,'X',Result),!.

% Move -> Jugada que volem dur a terme en el tauler
% opponent -> Jugador humà
% Es satisfà si s'ha pogut dur a terme el moviment correctament en el tauler
move([X,Y|_],opponent):-board(Board),
                replace_row_col(Board,X,Y,'X',Result),
                assertz(board(Result)),
                retract(board(Board)).
```

```
% Move -> Jugada que volem dur a terme en el tauler
% compuer -> Ordinador
% Es satisfà si s'ha pogut dur a terme el moviment correctament en el tauler
move([X,Y|_],computer):- board(Board),
                         replace_row_col(Board,X,Y,'O',Result),
                         assertz(board(Result)),
                         retract(board(Board)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% ESCOLLIM EL SEGÜENT JUGADOR  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% opponent/Computer -> Jugador HumàOrdinador
% Es satisfà si s'ha pogut escollir un jugador correctament.
next_player(opponent,computer). % Torn ordinador.
next_player(computer,opponent). % Torn jugador humà.
```

```
                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         %%%%%%%%%%%%%%%%%%%% MAIN  %%%%%%%%%%%%%%%%%%%%%%
                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Result -> Unificat l'estat final de tauler després de finalitzar la partida
% Es satisfà si s'ha inicialitzat correctament la partida
init_game(Result):-initBoard,display_game,nl,nl,play(opponent),board(Result).

% Player -> Jugador
% Es satisfà si s'ha trobat un guanyador o empat.
% Comprovem si ha hagut un guanyador amb el jugador anterior
play(Player):- board(Board),
               next_player(Player,AntPlayer), % Jugador anterior
               game_over(Board,AntPlayer,_),!, % Comprovem si s'ha acabat la partida
               announceResult(AntPlayer),!. % Anunciem el resulat en cas de que hi hagi un guanyador

% Es satisfà si hi ha un empat
play(Player):- possibles_moves(Moves),length(Moves,Len),Len=0,write("TIE!").

% Es satisfà en el cas que no hi ha un guanyador ni un empat, llavors es crida recursivament
% per passar al següent jugador.
play(Player):- choose_move(Player,Move), % Escollim un moviment
               move(Move,Player), % Fem el moviment
               display_game,nl,nl, % Mostrem per pantalla l'estat del tauler
               next_player(Player,Player1),!, % Passem el següent jugador
               play(Player1). % Crida recursiva.
```

```
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        %%%%%%%%   PREDICATS AUXILIARS    %%%%%%
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% L -> llista
% E -> Element al qual volem trobar el seu índex a la llista L
% Index -> Índex de l'element E
% Es satisfà el predicat si trobem l'index de l'element E dins de la llista L
indexOf([Element|_], Element, 0):- !.
indexOf([_|Tail], Element, Index):- indexOf(Tail, Element, Index1), !, Index is Index1+1.

% Sublist -> Llista d'elements que volem trobar en la llista <List>
% List -> Llista d'elements
% Es satisfà el predicat si <Sublist> és sublllista de la llista <Llista> amb el mateix ordre.
sublist(SubList, List):-append(Prefix, _, List), append(_, SubList, Prefix).

replace_nth(N,I,V,O):- nth1(N,I,_,T), nth1(N,O,V,T).

% M -> Tauler
% Row -> Fila
% Col -> Columna
% Cell -> Element
% N -> Tauler final despres de fer la substitució en la posició (Fila,Columna) amb l'element Cell
% Es satisfà si s'ha pogut remplaçar l'element de la posició Row,Columna amb l'element <Element>
replace_row_col(M,Row,Col,Cell,N) :-
    nth1(Row,M,Old),
    replace_nth(Col,Old,Cell,Upd),
    replace_nth(Row,M,Upd,N).
```

```
% Grid -> tauler
% J -> Peça del jugador
% N -> Mida de les columnes del tauler
% Es satisfà el predicat si es detecten quatre elements J consecutius horitzontalment
checkHori(Grid, J, N) :- maplist(nth1(N),Grid, Column), sublist([J,J,J,J],Column),!.
checkHori(Grid, J, N) :- N > 0, N1 is N-1, checkHori(Grid, J, N1),!.


% Grid -> Tauler
% J -> Peça jugador
% N -> Mida columnes tauler
% Es satisfà el predicat  si es detecten quatre elements J consecutius verticalment
checkVert(Grid, J, N) :- nth1(N,Grid,L), sublist([J,J,J,J], L),!.
checkVert(Grid, J, N) :- N > 0, N1 is N-1,  checkVert(Grid, J, N1),!.

% T -> Tauler
% X -> Peça jugador
% Es satisfà el predicat si es detecten quatre elements X consecutius diagonalment
checkdiagonals(X,T):- append(_,[C1,C2,C3,C4|_],T),
                      append(I1,[X|_],C1),
                      append(I2,[X|_],C2),
                      append(I3,[X|_],C3),
                      append(I4,[X|_],C4),
                      length(I1,M1), length(I2,M2), length(I3,M3), length(I4,M4),
                      M2 is M1-1, M3 is M2-1, M4 is M3-1,!.
```

```
% List -> Llista d'elements
% Index -> Comptador
% E -> Element posició buida
% Z -> Index posició buida
% Es satisfa si es retorna la primera posició buida d'una una fila <List>
findFirstEmpty([E|_],E,Index,Index):-!.
findFirstEmpty([X|List],Ele,Index,Z):- K is Index+1,findFirstEmpty(List,Ele,K,N),Z=N,!.
```

PROVES LININA:
Guanya l'ordinador.

```
A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  _  _  _  _  _  _
1  _  _  _  _  _  _  _
```

```
tria :A
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  _  _  _  _  _  _
1  X  _  _  _  _  _  _
```

```
robot: E
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  _  _  _  _  _  _
1  X  _  _  _  O  _  _
```

```
tria  A
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  X  _  _  _  _  _  _
1  X  _  _  _  O  _  _
```

```
robot: A
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  O  _  _  _  _  _  _
2  X  _  _  _  _  _  _
1  X  _  _  _  O  _  _
```

```
tria B
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  O  _  _  _  _  _  _
2  X  _  _  _  _  _  _
1  X  X  _  _  O  _  _
```

```
robot: F
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  O  _  _  _  _  _  _
2  X  _  _  _  _  _  _
1  X  X  _  _  O  O  _
```

```
tria C
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  O  _  _  _  _  _  _
2  X  _  _  _  _  _  _
1  X  X  X  _  O  O  _
```

```
robot: D
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  O  _  _  _  _  _  _
2  X  _  _  _  _  _  _
1  X  X  X  O  O  O  _
```

```
tria B
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  O  _  _  _  _  _  _
2  X  X  _  _  _  _  _
1  X  X  X  O  O  O  _
```

```
robot: G
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  O  _  _  _  _  _  _
2  X  X  _  _  _  _  _
1  X  X  X  O  O  O  O
```

```
THE COMPUTER WON THE GAME!
true.
```

Guanya L’huma.

```
init_game(_).
```

```
A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  _  _  _  _  _  _
1  _  _  _  _  _  _  _
```

```
tria B
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  _  _  _  _  _  _
1  _  X  _  _  _  _  _
```

```
robot: B

   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  O  _  _  _  _  _
1  _  X  _  _  _  _  _
```

```
tria C
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  O  _  _  _  _  _
1  _  X  X  _  _  _  _
```

```
robot: C

   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  O  O  _  _  _  _
1  _  X  X  _  _  _  _
```

```
tria  D
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  O  O  _  _  _  _
1  _  X  X  X  _  _  _
```

```
robot: E
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  O  O  _  _  _  _
1  _  X  X  X  O  _  _
```

```
tria  A
   A  B  C  D  E  F  G
6  _  _  _  _  _  _  _
5  _  _  _  _  _  _  _
4  _  _  _  _  _  _  _
3  _  _  _  _  _  _  _
2  _  O  O  _  _  _  _
1  X  X  X  X  O  _  _
```

```
YOU WON THE GAME!
true.
```