%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% INICIALITSACIO DEL TAULER  %%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- 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','\_','\_','\_','\_','\_'] ])).

/\*\*

 \* @Parametres

 \* Es satisfa el predicat initBoard() si s'ha creat dinanmicament el tauler mitjançant els methodes

 \* retract() per eliminar el tauler ja exitent, i despres amb el predicat < assertz()> crear 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 NOSTREN PER PANATALLA EL TAULER DEL TAULER  %%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/\*

\*@Parametres

\*Es satisfa 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 comencent per l'element en la posicio 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).

/\*\*

 \* @Parameters

 \* L -> Llista d'elements.

 \* Es satisfa 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 satisfa 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 satisfa si el jugador huma 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),!.

%hola

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/\*\*

 \* Board -> El tauler

 \* E -> '\_' codifica el una posicio buida en el tauler

 \* X -> fila

 \* Y -> columna

 \* Es satisfa si en la posicio x,y del tauler es una posicio 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, %troba l'index de l'element E i guarda el resultat en T

            nth1(X, Board, Row),

            findFirstEmpty(Row,'\_',1,Y),!.  %Troba la prima posicio buida de la fila.

/\*\*

 \* Es satisfa el predicat possibles\_moves , si en encara es poden

 \* fer moviments en el tauler, i posteriorment guardar els moviments en la

 \* llista dynamica <moves/1>

 \*

 \* \*\*/

possibles\_moves(\_):- assertz(moves([])),

                        board(Board),between(1,7,I), nth1(I,Board,Row),

                        findFirstEmpty(Row,'\_',1,Index), % troba la primera posicio buida de la fila

                        moves(Moves), % agafa el moviments els moviments valids en el tauler

                        retract(moves(\_)),

                        append([[I,Index]],Moves,List), % afageix el nou moviment trobat en la llista dynamica 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 el jugador la posicio on vol tirar i guardem el resultat en E

                            legal(Board,E,X,Y),Move=[X,Y|[]],!. %comprovem si la jugada es valida i posteriorment unificar la jugada a Move

/\*\*

 \* computer -> Ordinador

 \* Move -> Parametre on unificarem la jugada escollida per l'ordinador

 \* Es satisfa 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). % FOR BLOCK OPPONENT TO WIN

choose\_move(computer,Move):- possibles\_moves(Moves),length(Moves,N),random\_between(1,N,I),nth1(I,Moves,Move),!,dispMove(Move,computer),nl.

/\*\*

 \* Moves -> Llista de possibles moviments en el tauler

 \* Move -> Variable on unificarem la jugada ganadora per l'ordinador o que bloqueja l'opponent en guanyar.

 \* Player -> Jugador

 \* Es satisfa si s'aconseguix si es troba jugada que fagi guanyar l'ordinador altrament un eviti l'opponent guanyar

 \* i si no es compleixent tots els objectius anteriors es s'escollira una jugada random d'intre dels possibles moviments.

 \* \*\*/

winingMove([],\_,\_):-!,fail.

% simulem una jugada amb un dels moviments de la llista de moviments i seguidament comprobar si es una jugada guanyador

% 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 simulacio

 \* Result -> Esta del tauler despres de fer la simulacio amb el jugador computer/opponent

 \* Es satisfa si s'ha pogut dur a terme la judada

 \* \*\*/

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 -> Move que volem dur a terme en el tauler

 \* opponent -> Jugador huma

 \* Es satisfa 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 -> Move que volem dur a terme en el tauler

\* compuer -> Ordinador

\* Es satisfa 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 SEGUENT JUGADOR  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/\*\*

 \* opponent/Computer -> Jugador Huma/ Ordinador

 \* Es satisfa si s'ha pogut escollir un jugador correctament.

 \* \*\*/

next\_player(opponent,computer). % torn ordinador.

next\_player(computer,opponent). % torn jugador huma.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%% MAIN  %%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/\*\*

 \* Result -> unificat l'stat final de tauler despres de finalitzar la partida

 \* Es satisfa si s'ha inicialitzat correctament la partida

 \* \*\*/

init\_game(Result):-initBoard,display\_game,nl,nl,play(opponent),board(Result).

/\*\*

 \* Player -> Jugador

 \* Es satisfa 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 ha acabat la partida

                announceResult(AntPlayer),!. % anunciem el resulat en cas de que hi hagi un guanyador

% Es satisfa si hi ha un empat

play(Player):-possibles\_moves(Moves),length(Moves,Len),Len=0,write("TIE!").

% Es satisfa en el cas que no hi ha un guanyador ni un empat, llavors la crida recursiva

% per passar al seguent 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 seguent jugador

                play(Player1). % crida recursiva.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%     HELPER METHODS    %%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/\*\*

 \* L -> llista

 \* E -> Element a la qual volem trobar el seu index

 \* Index -> Index de l'element E

 \* Es satisfa el predicat si trobem l'index de l'element E dintre 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 satisfa el predicat si <Sublist> es subllista 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 substitucio en la posicio (Fila,Columna) amb l'element

 \* Es satisfa si s'ha pogut remplaçar l'element de la posicio 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 jugador

 \* N -> Mida columnes tauler

 \* Es satisfa el predicat  si es detecta quatre elements J consequtius Horitzaontalment

 \* \*\*/

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 satisfa el predicat  si es detecta quatre elements J consequtius 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 satisfa el predicat  si es detecta quatre elements X consequtius 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 -> Countador

 \* E -> Element posicio buida

 \* Z -> Index posicio buida

 \* Es satisfa si es retorna la primera posicio buida dona 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,!.