```
/* -------------------------------------------------------
    CSE 3401 F12 Assignment 4 file

% Surname:
% First Name:
% Student Number:

  ------------------------------------------------------- */

%do not chagne the follwoing line!
:- ensure_loaded('play.pl').

% /* -------------------------------------------------------- */
%               IMPORTANT! PLEASE READ THIS SUMMARY:
%       This files gives you some useful helpers (set &get).
%       Your job is to implement several predicates using
%       these helpers (feel free to add your own helpers if needed,
%       MAKE SURE to write comments for all your helpers, marks will
%       be deducted for bad style!).
%
%       Implement the following predicates at their designated space
%       in this file (we suggest to have a look at file ttt.pl  to
%       see how the implementations is done for game tic-tac-toe.
%
%          * initialize(InitialState,InitialPlyr).
%          * winner(State,Plyr)
%          * tie(State)
%          * terminal(State)
%          * moves(Plyr,State,MvList)
%          * nextState(Plyr,Move,State,NewState,NextPlyr)
%          * validmove(Plyr,State,Proposed)
%          * h(State,Val)  (see question 2 in the handout)
%          * lowerBound(B)
%          * upperBound(B)
% /* -------------------------------------------------------- */









% /* -------------------------------------------------------- */

% We use the following State Representation:
% [Row0, Row1 ... Rown] (ours is 6x6 so n = 5 ).
% each Rowi is a LIST of 6 elements '.' or '1' or '2' as follows:
%    . means the position is  empty
%    1 means player one has a stone in this position
%    2 means player two has a stone in this position.



% given helper: Inital state of the board
initBoard([ [.,.,.,.,.,.],
            [.,.,.,.,.,.],
            [.,.,1,2,.,.],
            [.,.,2,1,.,.],
            [.,.,.,.,.,.],
            [.,.,.,.,.,.] ]).

%%%%%%%%%%%%%%%%%%% IMPLEMENT: initialize(...)%%%%%%%%%%%%%%%%%%%%%%
%%% Using initBoard define initialize(InitialState,InitialPlyr).
%%%  holds iff InitialState is the initial state and
%%%  InitialPlyr is the player who moves first.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%winner(...)%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% define winner(State,Plyr) here.
%     - returns winning player if State is a terminal position and
%     Plyr has a higher score than the other player




%%%%%%%%%%%%%%%%%%%%%%%%%%%%tie(...)%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% define tie(State) here.
%    - true if terminal State is a "tie" (no winner)




%%%%%%%%%%%%%%%%%%%%%%%%%%%%terminal(...)%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% define terminal(State).
%   - true if State is a terminal




%%%%%%%%%%%%%%%%%%%%%%%%%%%%showState(State)%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% given helper. DO NOT  change this. It's used by play.pl
%%
showState( G ) :-
        printRows( G ).

printRows( [] ).
printRows( [H|L] ) :-
        printList(H),
        nl,
        printRows(L).

printList([]).
printList([H | L]) :-
        write(H),
        write(' '),
        printList(L).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%moves(Plyr,State,MvList)%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% define moves(Plyr,State,MvList).
%   - returns list MvList of all legal moves Plyr can make in State
%




%%%%%%%%%%%%%%%%nextState(Plyr,Move,State,NewState,NextPlyr)%%%%%%%%%%%%%%%%%%%%
```

```
%%
%% define nextState(Plyr,Move,State,NewState,NextPlyr).
%   - given that Plyr makes Move in State, it determines NewState (i.e. the next
%     state) and NextPlayer (i.e. the next player who will move).
%




%%%%%%%%%%%%%%%%%%%%%%%%%%%%validmove(Plyr,State,Proposed)%%%%%%%%%%%%%%%%%%%%%%
%%
%% define validmove(Plyr,State,Proposed).
%   - true if Proposed move by Plyr is valid at State.




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%h(State,Val)%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% define h(State,Val).
%   - given State, returns heuristic Val of that state
%   - larger values are good for Max, smaller values are good for Min
%   NOTE1. If State is terminal h should return its true value.
%   NOTE2. If State is not terminal h should be an estimate of
%          the value of state (see handout on ideas about
%          good heuristics.




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%lowerBound(B)%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% define lowerBound(B).
%   - returns a value B that is less than the actual or heuristic value
%     of all states.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%upperBound(B)%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% define upperBound(B).
%   - returns a value B that is greater than the actual or heuristic value
%     of all states.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                                                              %
%                Given   UTILITIES                                             %
%                   do NOT change these!                                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% get(Board, Point, Element)
%    : get the contents of the board at position column X and row Y
% set(Board, NewBoard, [X, Y], Value):
%    : set Value at column X row Y in Board and bind resulting grid to NewBoard
%
% The origin of the board is in the upper left corner with an index of
% [0,0], the upper right hand corner has index [5,0], the lower left
% hand corner has index [0,5], the lower right hand corner has index
% [5,5] (on a 6x6 board).
%
```

```
% Example
% ?- initBoard(B), showState(B), get(B, [2,3], Value).
%. . . . . .
%. . . . . .
%. . 1 2 . .
%. . 2 1 . .
%. . . . . .
%. . . . . .
%
%B = [['.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.'],
%     ['.', '.', 1, 2, '.', '.'], ['.', '.', 2, 1, '.'|...],
%     ['.', '.', '.', '.'|...], ['.', '.', '.'|...]]
%Value = 2
%Yes
%?-
%
% Setting values on the board
% ?- initBoard(B),  showState(B),set(B, NB1, [2,4], 1), set(NB1, NB2, [2,3], 1),  showState(
NB2).
%
% . . . . . .
% . . . . . .
% . . 1 2 . .
% . . 2 1 . .
% . . . . . .
% . . . . . .
%
% . . . . . .
% . . . . . .
% . . 1 2 . .
% . . 1 1 . .
% . . 1 . . .
% . . . . . .
%
%B = [['.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.'], ['.', '.',
%1, 2, '.', '.'], ['.', '.', 2, 1, '.'|...], ['.', '.', '.', '.'|...], ['.', '.',
% '.'|...]]
%NB1 = [['.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.'], ['.', '.'
%, 1, 2, '.', '.'], ['.', '.', 2, 1, '.'|...], ['.', '.', 1, '.'|...], ['.', '.
%', '.'|...]]
%NB2 = [['.', '.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', '.'], ['.', '.'
%, 1, 2, '.', '.'], ['.', '.', 1, 1, '.'|...], ['.', '.', 1, '.'|...], ['.',
%'.', '.'|...]]

% get(Board, Point, Element): get the value of the board at position
% column X and row Y (indexing starts at 0).
get( Board, [X, Y], Value) :-
        nth0( Y, Board, ListY),
        nth0( X, ListY, Value).

% set( Board, NewBoard, [X, Y], Value)

set( [Row|RestRows], [NewRow|RestRows], [X, 0], Value)
    :- setInList(Row, NewRow, X, Value).

set( [Row|RestRows], [Row|NewRestRows], [X, Y], Value) :-
        Y > 0,
        Y1 is Y-1,
        set( RestRows, NewRestRows, [X, Y1], Value).

% setInList( List, NewList, Index, Value)

setInList( [_|RestList], [Value|RestList], 0, Value).
```

```
setInList( [Element|RestList], [Element|NewRestList], Index, Value) :-
        Index > 0,
        Index1 is Index-1,
        setInList( RestList, NewRestList, Index1, Value).
```