CSPC 54 : Prolog Assignment-7

Name : Rajneesh Pandey,

Roll no. 106119100,

Class : CSE-B

## Instructions 1

**Text

Description automatically generated**

**CODE :**

:- dynamic ([

         agent\_location/1,

         gold\_location/1,

         pit\_location/1,

         time\_taken/1,

         score/1,

         visited/1,

         visited\_cells/1,

         world\_size/1,

         wumpus\_location/1,

             isPit/2,

             isWumpus/2,

             isGold/2

        ]).

solve\_question1 :-

    format('I am ready with the given configuration...~n ~n', []),

    init,

    format('Game Starts Now !!!~n ~n', []),

    take\_steps([[1,1]]).

step\_pre(VisitedList) :-

    agent\_location(AL),

    gold\_location(GL),

    wumpus\_location(WL),

    score(S),

    time\_taken(T),

    ( AL=GL -> writeln('WON!'), format('Total Points achieved = ~p,~n Time: ~p', [S,T])

    ; AL=WL -> format('Lost: Wumpus kills me!!!~n', []),

               format('Total Points: ~p,~n Time: ~p', [S,T])

    ; take\_steps(VisitedList)

    ).

take\_steps(VisitedList) :-

    make\_percept\_sentence(Perception),

    agent\_location(AL),

    format('I\'m in ~p, seeing: ~p~n', [AL,Perception]),

    update\_knowledge\_base(Perception),

    ask\_knowledge\_base(VisitedList, Action),

    format('I\'m going to: ~p~n', [Action]),

    update\_time,

    update\_score,

    agent\_location(Aloc),

    VL = [Aloc|VisitedList],

    standing,

    step\_pre(VL).

update\_time :-

    time\_taken(T),

    NewTime is T+1,

    retractall( time\_taken(\_) ),

    assert( time\_taken(NewTime) ).

update\_score :-

    agent\_location(AL),

    gold\_location(GL),

    wumpus\_location(WL),

    update\_score(AL, GL, WL).

update\_score(P) :-

    score(S),

    NewScore is S+P,

    retractall( score(\_) ),

    assert( score(NewScore) ).

update\_score(AL, AL, \_) :-

    update\_score(1000).

update\_score(\_,\_,\_) :-

    update\_score(-1).

update\_agent\_location(NewAL) :-

    retractall( agent\_location(\_) ),

    assert( agent\_location(NewAL) ).

is\_pit(no,  X) :-

    \+ pit\_location(X).

is\_pit(yes, X) :-

    pit\_location(X).

standing :-

    wumpus\_location(WL),

    gold\_location(GL),

    agent\_location(AL),

    ( is\_pit(yes, AL) -> format('Agent was fallen into a pit!~n~n', []),

      fail

    ; stnd(AL, GL, WL)

      %\+ pit\_location(yes, Al),

    ).

stnd(\_, \_, \_) :-

    format('I should proceed now !!!~n~n', []).

stnd(AL, \_, AL) :-

    format('YIKES! You\'re eaten by the wumpus!', []),

    fail.

stnd(AL, AL, \_) :-

    format('AGENT FOUND THE GOLD!!', []),

    true.

%------------------------------------------------------------------

% Perceptotion

make\_perception([\_Stench,\_Bleeze,\_Glitter]) :-

    agent\_location(AL),

    isStinky(AL),

    isBleezie(AL),

    isGlittering(AL).

test\_perception :-

    make\_percept\_sentence(Percept),

    format('I feel ~p, ',[Percept]).

make\_percept\_sentence([Stench,Bleeze,Glitter]) :-

    smelly(Stench),

    bleezy(Bleeze),

    glittering(Glitter).

init :-

    init\_game,

    init\_land\_fig72,

    init\_agent,

    init\_wumpus.

init\_game :-

    retractall( time\_taken(\_) ),

    assert( time\_taken(0) ),

    retractall( score(\_) ),

    assert( score(0) ),

    retractall( visited(\_) ),

    assert( visited(1) ),

    retractall( isWumpus(\_,\_) ),

    retractall( isGold(\_,\_) ),

    retractall( visited\_cells(\_) ),

    assert( visited\_cells([]) ).

init\_land\_fig72 :-

    retractall( world\_size(\_) ),

    assert( world\_size(4) ),

    retractall( gold\_location(\_) ),

    assert( gold\_location([3,2]) ),

    retractall( pit\_location(\_) ),

    assert( pit\_location([4,4]) ),

    assert( pit\_location([3,3]) ),

    assert( pit\_location([1,3]) ).

init\_agent :-

    retractall( agent\_location(\_) ),

    assert( agent\_location([1,1]) ),

    visit([1,1]).

init\_wumpus :-

    retractall( wumpus\_location(\_) ),

    assert( wumpus\_location([4,1]) ).

visit(Xs) :-

    visited\_cells(Ys),

    retractall( visited\_cells(\_) ),

    assert( visited\_cells([Ys|Xs]) ).

%adj(X,Y) :-

%    world\_size(WS),

%    ( X is Y+1, Y   < WS

%    ; X is Y-1, Y-1 > 0

%    ).

adj(1,2).

adj(2,1).

adj(2,3).

adj(3,2).

adj(3,4).

adj(4,3).

adjacent( [X1, Y1], [X2, Y2] ) :-

    ( X1 = X2, adj( Y1, Y2 )

    ; Y1 = Y2, adj( X1, X2 )

    ).

%adjacent([X1,Y],[X2,Y]) :-

%    adj(X1,X2).

%adjacent([X,Y1],[X,Y2]) :-

%    adj(Y1,Y2).

isSmelly(Ls1) :-

    wumpus\_location( Ls2 ),

    adjacent( Ls1, Ls2 ).

isBleezy(Ls1) :-

    pit\_location( Ls2 ),

    adjacent( Ls1, Ls2 ).

isGlittering( [X1, Y1] ) :-

    gold\_location( [X2, Y2] ),

    X1 = X2,

    Y1 = Y2.

bleezy(yes) :-

    agent\_location(AL),

    isBleezy(AL).

bleezy(no).

smelly(yes) :-

    agent\_location(AL),

    isSmelly(AL).

smelly(no).

glittering(yes) :-

    agent\_location(AL),

    isGlittering(AL).

glittering(no).

%-------------------------------------------------------------

% Knowledge Base:

update\_knowledge\_base( [Stench,Bleeze,Glitter] ) :-

    add\_wumpus\_knowledge\_base(Stench),

    add\_pit\_knowledge\_base(Bleeze),

    add\_gold\_knowledge\_base(Glitter).

% if it would be 'yes' -> it would mean the player is eaten ;]

add\_wumpus\_knowledge\_base(no) :-

    %agent\_location(L1),

    %adjacent(L1, L2),

    %assume\_wumpus(no, L2).

    agent\_location([X,Y]),

    world\_size(\_),

    % Checking needed!!

    % adj will freeze for (4,\_) !!

    Z1 is Y+1, assume\_wumpus(no,[X,Z1]),

    Z2 is Y-1, assume\_wumpus(no,[X,Z2]),

    Z3 is X+1, assume\_wumpus(no,[Z3,Y]),

    Z4 is X-1, assume\_wumpus(no,[Z4,Y]).

add\_pit\_knowledge\_base(no) :-

    agent\_location([X,Y]),

    Z1 is Y+1, assume\_pit(no,[X,Z1]),

    Z2 is Y-1, assume\_pit(no,[X,Z2]),

    Z3 is X+1, assume\_pit(no,[Z3,Y]),

    Z4 is X-1, assume\_pit(no,[Z4,Y]).

% Checking needed!! If its not already in the knowledge\_base !!!

add\_pit\_knowledge\_base(yes) :-

    agent\_location([X,Y]),

    Z1 is Y+1, assume\_pit(yes,[X,Z1]),

    Z2 is Y-1, assume\_pit(yes,[X,Z2]),

    Z3 is X+1, assume\_pit(yes,[Z3,Y]),

    Z4 is X-1, assume\_pit(yes,[Z4,Y]).

add\_gold\_knowledge\_base(no) :-

    gold\_location(GL),

    assume\_gold(no, GL).

add\_gold\_knowledge\_base(yes) :-

    gold\_location([X1,Y1]),

    agent\_location([X2,Y2]),

    X1 = X2, Y1 = Y2,

    assume\_gold(yes, [X1,Y1]).

assume\_wumpus(no, L) :-

    retractall( isWumpus(\_, L) ),

    assert( isWumpus(no, L) ),

    format('knowledge\_base learn ~p - no Wumpus there!~n', [L]).

assume\_wumpus(yes, L) :-

    %wumpus\_healthy, % Will be included ...

    retractall( isWumpus(\_, L) ),

    assert( isWumpus(yes, L) ),

    format('knowledge\_base learn ~p - possibly the Wumpus is there!~n', [L]).

assume\_pit(no, L) :-

    retractall( isPit(\_, L) ),

    assert( isPit(no, L) ),

    format('knowledge\_base learn ~p - there\'s no Pit there!~n', [L]).

assume\_pit(yes, L) :-

    retractall( isPit(\_, L) ),

    assert( isPit(yes, L) ),

    format('knowledge\_base learn ~p - its a Pit!~n', [L]).

assume\_gold(no, L) :-

    retractall( isGold(\_, L) ),

    assert( isGold(no, L) ),

    format('knowledge\_base learn ~p - there\'s no gold here!~n', [L]).

assume\_gold(yes, L) :-

    retractall( isGold(\_, L) ),

    assert( isGold(yes, L) ),

    format('knowledge\_base learn ~p - GOT THE GOLD!!!~n', [L]).

permitted([X,Y]) :-

    world\_size(WS),

    0 < X, X < WS+1,

    0 < Y, Y < WS+1.

ask\_knowledge\_base(VisitedList, Action) :-

    isWumpus(no, L),

    isPit(no, L),

    permitted(L),

    not\_member(L, VisitedList),

    update\_agent\_location(L),

    Action = L.

%------------------------------------------------------------------------------

% Utils

not\_member(\_, []).

not\_member([X,Y], [[U,V]|Ys]) :-

    ( X=U,Y=V -> fail

    ; not\_member([X,Y], Ys)

    ).

**INPUT:**

solve\_question1.

**OUTPUT:**

I am ready with the given configuration...  
   
Game Starts Now !!!  
   
I'm in [1,1], seeing: [no,no,no]  
knowledge\_base learn [1,2] - no Wumpus there!  
knowledge\_base learn [1,0] - no Wumpus there!  
knowledge\_base learn [2,1] - no Wumpus there!  
knowledge\_base learn [0,1] - no Wumpus there!  
knowledge\_base learn [1,2] - there's no Pit there!  
knowledge\_base learn [1,0] - there's no Pit there!  
knowledge\_base learn [2,1] - there's no Pit there!  
knowledge\_base learn [0,1] - there's no Pit there!  
knowledge\_base learn [3,2] - there's no gold here!  
I'm going to: [1,2]  
I should proceed now !!!  
  
I'm in [1,2], seeing: [no,yes,no]  
knowledge\_base learn [1,3] - no Wumpus there!  
knowledge\_base learn [1,1] - no Wumpus there!  
knowledge\_base learn [2,2] - no Wumpus there!  
knowledge\_base learn [0,2] - no Wumpus there!  
knowledge\_base learn [1,3] - its a Pit!  
knowledge\_base learn [1,1] - its a Pit!  
knowledge\_base learn [2,2] - its a Pit!  
knowledge\_base learn [0,2] - its a Pit!  
knowledge\_base learn [3,2] - there's no gold here!  
I'm going to: [2,1]  
I should proceed now !!!  
  
I'm in [2,1], seeing: [no,no,no]  
knowledge\_base learn [2,2] - no Wumpus there!  
knowledge\_base learn [2,0] - no Wumpus there!  
knowledge\_base learn [3,1] - no Wumpus there!  
knowledge\_base learn [1,1] - no Wumpus there!  
knowledge\_base learn [2,2] - there's no Pit there!  
knowledge\_base learn [2,0] - there's no Pit there!  
knowledge\_base learn [3,1] - there's no Pit there!  
knowledge\_base learn [1,1] - there's no Pit there!  
knowledge\_base learn [3,2] - there's no gold here!  
I'm going to: [2,2]  
I should proceed now !!!  
  
I'm in [2,2], seeing: [no,no,no]  
knowledge\_base learn [2,3] - no Wumpus there!  
knowledge\_base learn [2,1] - no Wumpus there!  
knowledge\_base learn [3,2] - no Wumpus there!  
knowledge\_base learn [1,2] - no Wumpus there!  
knowledge\_base learn [2,3] - there's no Pit there!  
knowledge\_base learn [2,1] - there's no Pit there!  
knowledge\_base learn [3,2] - there's no Pit there!  
knowledge\_base learn [1,2] - there's no Pit there!  
knowledge\_base learn [3,2] - there's no gold here!  
I'm going to: [3,1]  
I should proceed now !!!  
  
I'm in [3,1], seeing: [yes,no,no]  
I'm in [3,1], seeing: [no,no,no]  
knowledge\_base learn [3,2] - no Wumpus there!  
knowledge\_base learn [3,0] - no Wumpus there!  
knowledge\_base learn [4,1] - no Wumpus there!  
knowledge\_base learn [2,1] - no Wumpus there!  
knowledge\_base learn [3,2] - there's no Pit there!  
knowledge\_base learn [3,0] - there's no Pit there!  
knowledge\_base learn [4,1] - there's no Pit there!  
knowledge\_base learn [2,1] - there's no Pit there!  
knowledge\_base learn [3,2] - there's no gold here!  
I'm going to: [2,3]  
I should proceed now !!!  
  
I'm in [2,3], seeing: [no,yes,no]  
knowledge\_base learn [2,4] - no Wumpus there!  
knowledge\_base learn [2,2] - no Wumpus there!  
knowledge\_base learn [3,3] - no Wumpus there!  
knowledge\_base learn [1,3] - no Wumpus there!  
knowledge\_base learn [2,4] - its a Pit!  
knowledge\_base learn [2,2] - its a Pit!  
knowledge\_base learn [3,3] - its a Pit!  
knowledge\_base learn [1,3] - its a Pit!  
knowledge\_base learn [3,2] - there's no gold here!  
I'm going to: [3,2]  
I should proceed now !!!  
  
WON!  
Total Points achieved = 995,  
 Time: 6

*1***true**

Next101001,000 Stop

**Screenshots:**

Graphical user interface, text, application

Description automatically generated

## Instructions 2 :

Text

Description automatically generated

**Code:**

is\_valid(8).

is\_valid(7).

is\_valid(6).

is\_valid(5).

is\_valid(4).

is\_valid(3).

is\_valid(2).

is\_valid(1).

is\_valid(0).

kill\_one\_point(4, 3).

kill\_one\_point(3, 2).

kill\_one\_point(2, 1).

kill\_one\_point(1, 0).

kill\_one\_point(8, 7).

kill\_one\_point(7, 6).

kill\_one\_point(6, 5).

kill\_one\_point(5, 4).

nonempty([[\_|\_] | \_]).

nonempty([[] | Rest]) :- nonempty(Rest).

empty([]).

empty([[] | Rest]) :- empty(Rest).

first\_column([], [], []).

first\_column([[] | Rows], [[] | Remainder], Column) :-

    first\_column(Rows, Remainder, Column).

first\_column([[Element | Row] | Rows], [Row | Remainder], [Element | Column]) :-

    first\_column(Rows, Remainder, Column).

transpose(All, []) :- empty(All).

transpose(All, [NewRow | Result]) :-

    nonempty(All),

    first\_column(All, YetToBeTransposed, NewRow),

    transpose(YetToBeTransposed, Result).

tile\_reduces(\*, \_).

tile\_reduces(0, []).

tile\_reduces(Number, [\* | Neighbours]) :-

    kill\_one\_point(Number, NewNumber),

    tile\_reduces(NewNumber, Neighbours).

tile\_reduces(Number, [NeighbourNumber | Neighbours]) :-

    is\_valid(Number),

    is\_valid(NeighbourNumber),

    tile\_reduces(Number, Neighbours).

%% Row reduces iff all its tiles reduce

row\_reduces([], \_).

row\_reduces([Tile | Row], [Neighbours | Others]) :-

    tile\_reduces(Tile, Neighbours),

    row\_reduces(Row, Others).

%% grid\_reduces: true iff all numbers reduce to 0

% Initiate

grid\_reduces([FirstRow, SecondRow | Tail]) :-

    grid\_reduces(none, FirstRow, SecondRow, Tail).

% Initiate for single-row board

grid\_reduces([Row]) :-

    grid\_reduces(none, Row, none, []).

% First row

grid\_reduces(none, ThisRow, SecondRow, [NextRow | Tail]) :-

    [\_ | SecondRowTail] = SecondRow,

    [\_ | ThisRowTail] = ThisRow,

    transpose([

        SecondRow,

        [0 | SecondRow],

        SecondRowTail,

        [0 | ThisRow],

        ThisRowTail

    ], Neighbours),

    row\_reduces(ThisRow, Neighbours),

    grid\_reduces(ThisRow, SecondRow, NextRow, Tail).

% First row if there are only two rows

grid\_reduces(none, ThisRow, SecondRow, []) :-

    [\_ | SecondRowTail] = SecondRow,

    [\_ | ThisRowTail] = ThisRow,

    transpose([

        SecondRow,

        [0 | SecondRow],

        SecondRowTail,

        [0 | ThisRow],

        ThisRowTail

    ], Neighbours),

    row\_reduces(ThisRow, Neighbours),

    grid\_reduces(ThisRow, SecondRow, none, []).

% First row if there is only one row

grid\_reduces(none, ThisRow, none, []) :-

    [\_ | ThisRowTail] = ThisRow,

    transpose([

        ThisRowTail, [0 | ThisRow]

    ], Neighbours),

    row\_reduces(ThisRow, Neighbours).

% Last row

grid\_reduces(PenultimateRow, ThisRow, none, []) :-

    [\_ | PenultimateRowTail] = PenultimateRow,

    [\_ | ThisRowTail] = ThisRow,

    transpose([

        [0 | PenultimateRow], PenultimateRow, PenultimateRowTail,

        [0 | ThisRow], ThisRowTail

    ], Neighbours),

    row\_reduces(ThisRow, Neighbours).

% Second to last row

grid\_reduces(AboveRow, ThisRow, BelowRow, []) :-

    [\_ | AboveRowTail] = AboveRow,

    [\_ | BelowRowTail] = BelowRow,

    [\_ | ThisRowTail] = ThisRow,

    transpose([

        AboveRow, [0 | AboveRow], AboveRowTail,

        BelowRow, [0 | BelowRow], BelowRowTail,

        [0 | ThisRow], ThisRowTail

    ], Neighbours),

    row\_reduces(ThisRow, Neighbours),

    grid\_reduces(ThisRow, BelowRow, none, []).

% Inner row /4

grid\_reduces(AboveRow, ThisRow, BelowRow, [NextRow | Tail]) :-

    [\_ | AboveRowTail] = AboveRow,

    [\_ | BelowRowTail] = BelowRow,

    [\_ | ThisRowTail] = ThisRow,

    transpose([

        AboveRow, [0 | AboveRow], AboveRowTail,

        BelowRow, [0 | BelowRow], BelowRowTail,

        [0 | ThisRow], ThisRowTail

    ], Neighbours),

    row\_reduces(ThisRow, Neighbours),

    grid\_reduces(ThisRow, BelowRow, NextRow, Tail).

valid\_board(Board) :- grid\_reduces(Board).

print\_board([]).

print\_board([Row | Rows]) :- print\_row(Row), print\_board(Rows).

print\_row([]) :- nl.

print\_row([Tile | Tiles]) :- write(Tile), write(" "), print\_row(Tiles).

solve\_question2(Board) :- valid\_board(Board), print\_board(Board).

solve\_question2(Board, Result) :- solve\_question2(Board), Result = Board.

Input :

*solve\_question2*([ [1,1,1,0], [2,\*,3,1], [3,\*,\_,\_], [2,\*,4,\_], [1,1,2,\_]]).

Output :

1 1 1 0  
2 \* 3 1  
3 \* \* 2  
2 \* 4 \*  
1 1 2 1

*1***true**

Input :

*solve\_question2*([ [\_,\_,\_], [\_,8,\_], [\_,\_,\_]]).

Output :

\* \* \*  
\* 8 \*  
\* \* \*

*1***true**

ScreenShot :

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated