```
/* 8_puzzle.pl */

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%    A* Algorithm
%%%
%%%
%%%    Nodes have form    S#D#F#A
%%%         where S describes the state or configuration
%%%               D is the depth of the node
%%%               F is the evaluation function value
%%%               A is the ancestor list for the node

:- op(400,yfx,'#').    /* Node builder notation */

solve(State,Soln) :- f_function(State,0,F),
              search([State#0#F#[]],S), reverse(S,Soln).

f_function(State,D,F) :- h_function(State,H),
                  F is D + H.

search([State#_#_#Soln|_], Soln) :- goal(State).
search([B|R],S) :- expand(B,Children),
              insert_all(Children,R,Open),
              search(Open,S).

insert_all([F|R],Open1,Open3) :- insert(F,Open1,Open2),
                        insert_all(R,Open2,Open3).
insert_all([],Open,Open).

insert(B,Open,Open) :- repeat_node(B,Open), ! .
insert(B,[C|R],[B,C|R]) :- cheaper(B,C), ! .
insert(B,[B1|R],[B1|S]) :- insert(B,R,S), !.
insert(B,[],[B]).

repeat_node(P#_#_#_, [P#_#_#_|_]).

cheaper( _#_#F1#_ , _#_#F2#_ ) :- F1 < F2.

expand(State#D#_#S,All_My_Children) :-
    bagof(Child#D1#F#[Move|S],
         (D1 is D+1,
          move(State,Child,Move),
          f_function(Child,D1,F)),
         All_My_Children).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%   8-puzzle solver
%%%
%%%
%%%   State have form   A/B/C/D/E/F/G/H/I
%%%        where {A,...,I} = {0,...,8}
%%%          0 represents the empty tile
%%%

goal(1/2/3/8/0/4/7/6/5).

  %%% The puzzle moves

left( A/0/C/D/E/F/H/I/J , 0/A/C/D/E/F/H/I/J ).
left( A/B/C/D/0/F/H/I/J , A/B/C/0/D/F/H/I/J ).
left( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/0/H/J ).
left( A/B/0/D/E/F/H/I/J , A/0/B/D/E/F/H/I/J ).
left( A/B/C/D/E/0/H/I/J , A/B/C/D/0/E/H/I/J ).
left( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/F/H/0/I ).

up( A/B/C/0/E/F/H/I/J , 0/B/C/A/E/F/H/I/J ).
up( A/B/C/D/0/F/H/I/J , A/0/C/D/B/F/H/I/J ).
up( A/B/C/D/E/0/H/I/J , A/B/0/D/E/C/H/I/J ).
up( A/B/C/D/E/F/0/I/J , A/B/C/0/E/F/D/I/J ).
up( A/B/C/D/E/F/H/0/J , A/B/C/D/0/F/H/E/J ).
up( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/0/H/I/F ).

right( A/0/C/D/E/F/H/I/J , A/C/0/D/E/F/H/I/J ).
right( A/B/C/D/0/F/H/I/J , A/B/C/D/F/0/H/I/J ).
right( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/H/J/0 ).
right( 0/B/C/D/E/F/H/I/J , B/0/C/D/E/F/H/I/J ).
right( A/B/C/0/E/F/H/I/J , A/B/C/E/0/F/H/I/J ).
right( A/B/C/D/E/F/0/I/J , A/B/C/D/E/F/I/0/J ).

down( A/B/C/0/E/F/H/I/J , A/B/C/H/E/F/0/I/J ).
down( A/B/C/D/0/F/H/I/J , A/B/C/D/I/F/H/0/J ).
down( A/B/C/D/E/0/H/I/J , A/B/C/D/E/J/H/I/0 ).
down( 0/B/C/D/E/F/H/I/J , D/B/C/0/E/F/H/I/J ).
down( A/0/C/D/E/F/H/I/J , A/E/C/D/0/F/H/I/J ).
down( A/B/0/D/E/F/H/I/J , A/B/F/D/E/0/H/I/J ).

  %%% the heuristic function
h_function(Puzz,H) :- p_fcn(Puzz,P),
```

```
            s_fcn(Puzz,S),
            H is P + 3*S.


  %%% the move
move(P,C,left) :-  left(P,C).
move(P,C,up) :-  up(P,C).
move(P,C,right) :-  right(P,C).
move(P,C,down) :-  down(P,C).

  %%% the Manhattan distance function
p_fcn(A/B/C/D/E/F/G/H/I, P) :-
    a(A,Pa), b(B,Pb), c(C,Pc),
    d(D,Pd), e(E,Pe), f(F,Pf),
    g(G,Pg), h(H,Ph), i(I,Pi),
    P is Pa+Pb+Pc+Pd+Pe+Pf+Pg+Ph+Pg+Pi.

a(0,0). a(1,0). a(2,1). a(3,2). a(4,3). a(5,4). a(6,3). a(7,2). a(8,1).
b(0,0). b(1,1). b(2,0). b(3,1). b(4,2). b(5,3). b(6,2). b(7,3). b(8,2).
c(0,0). c(1,2). c(2,1). c(3,0). c(4,1). c(5,2). c(6,3). c(7,4). c(8,3).
d(0,0). d(1,1). d(2,2). d(3,3). d(4,2). d(5,3). d(6,2). d(7,2). d(8,0).
e(0,0). e(1,2). e(2,1). e(3,2). e(4,1). e(5,2). e(6,1). e(7,2). e(8,1).
f(0,0). f(1,3). f(2,2). f(3,1). f(4,0). f(5,1). f(6,2). f(7,3). f(8,2).
g(0,0). g(1,2). g(2,3). g(3,4). g(4,3). g(5,2). g(6,2). g(7,0). g(8,1).
h(0,0). h(1,3). h(2,3). h(3,3). h(4,2). h(5,1). h(6,0). h(7,1). h(8,2).
i(0,0). i(1,4). i(2,3). i(3,2). i(4,1). i(5,0). i(6,1). i(7,2). i(8,3).

  %%% the out-of-cycle function
s_fcn(A/B/C/D/E/F/G/H/I, S) :-
    s_aux(A,B,S1), s_aux(B,C,S2), s_aux(C,F,S3),
    s_aux(F,I,S4), s_aux(I,H,S5), s_aux(H,G,S6),
    s_aux(G,D,S7), s_aux(D,A,S8), s_aux(E,S9),
    S is S1+S2+S3+S4+S5+S6+S7+S8+S9.

s_aux(0,0) :- !.
s_aux(_,1).

s_aux(X,Y,0) :- Y is X+1, !.
s_aux(8,1,0) :- !.
s_aux(_,_,2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%    8-puzzle animation -- using VT100 character graphics
%%%
```

```
%%%
%%%

puzzle(P) :- solve(P,S),
          animate(P,S),
          message.

animate(P,S) :- initialize(P),
             cursor(1,2), write(S),
             cursor(1,22), write('Hit ENTER to step solver.'),
             get0(_X),
             play_back(S).

:- dynamic location/3.   %%% So that location of a tile
                    %%%  can be retracted/asserted.
                    %%% Location(s) asserted and retracted
                    %%%  by puzzle animator below

initialize(A/B/C/D/E/F/H/I/J) :-
 cls,
 retractall(location(_,_,_)),
 assert(location(A,20,5)),
 assert(location(B,30,5)),
 assert(location(C,40,5)),
 assert(location(F,40,10)),
 assert(location(J,40,15)),
 assert(location(I,30,15)),
 assert(location(H,20,15)),
 assert(location(D,20,10)),
 assert(location(E,30,10)), draw_all.

draw_all :- draw(1), draw(2), draw(3), draw(4),
         draw(5), draw(6), draw(7), draw(8).

  %%% play_back([left,right,up,...]).
play_back([M|R]) :- call(M), get0(_X), play_back(R).
play_back([]) :- cursor(1,24).  %%% Put cursor out of the way

message :- nl,nl,
  write('   ****************************************'), nl,
  write('   *  Enter 8-puzzle goals in the form ...   *'), nl,
  write('   *     ?-  puzzle(0/8/1/2/4/3/7/6/5).      *'), nl,
  write('   *   Enter goal ''message'' to reread this.   *'), nl,
  write('   ****************************************'), nl, nl.
```

```
cursor(X,Y) :- put(27), put(91), %%% ESC [
            write(Y),
            put(59),        %%%  ;
            write(X),
            put(72).        %%%  M

  %%% clear the screen, quickly
cls :-  put(27), put("["), put("2"), put("J").

  %%% video attributes -- bold and blink not working
plain       :- put(27), put("["), put("0"), put("m").
reverse_video :- put(27), put("["), put("7"), put("m").


  %%% Tile objects, character map(s)
  %%% Each tile should be drawn using the character map,
  %%%  drawn at 'location', which is asserted and retracted
  %%%  by 'playback'.
character_map(N, [ [' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],
                   [' ',' ',' ',' ', N ,' ',' ',' ',' ',' '],
                   [' ',' ',' ',' ',' ',' ',' ',' ',' ',' '] ]).


  %%% move empty tile (spot) to the left
left :- retract(location(0,X0,Y0)),
       Xnew is X0 - 10,
       location(Tile,Xnew,Y0),
       assert(location(0,Xnew,Y0)),
       right(Tile),right(Tile),right(Tile),
       right(Tile),right(Tile),
       right(Tile),right(Tile),right(Tile),
       right(Tile),right(Tile).

up :- retract(location(0,X0,Y0)),
     Ynew is Y0 - 5,
     location(Tile,X0,Ynew),
     assert(location(0,X0,Ynew)),
     down(Tile),down(Tile),down(Tile),down(Tile),down(Tile).

right :- retract(location(0,X0,Y0)),
        Xnew is X0 + 10,
        location(Tile,Xnew,Y0),
        assert(location(0,Xnew,Y0)),
        left(Tile),left(Tile),left(Tile),left(Tile),left(Tile),
        left(Tile),left(Tile),left(Tile),left(Tile),left(Tile).
```

```
down :- retract(location(0,X0,Y0)),
      Ynew is Y0 + 5,
      location(Tile,X0,Ynew),
      assert(location(0,X0,Ynew)),
      up(Tile),up(Tile),up(Tile),up(Tile),up(Tile).


draw(Obj) :- reverse_video, character_map(Obj,M),
         location(Obj,X,Y),
         draw(X,Y,M), plain.

 %%% hide tile
hide(Obj) :- character_map(Obj,M),
         location(Obj,X,Y),
         hide(X,Y,M).

hide(_,_,[]).
hide(X,Y,[R|G]) :- hide_row(X,Y,R),
              Y1 is Y + 1,
              hide(X,Y1,G).

hide_row(_,_,[]).
hide_row(X,Y,[_|R]) :- cursor(X,Y),
                 write(' '),
                 X1 is X + 1,
                 hide_row(X1,Y,R).

  %%% draw tile
draw(_,_,[]).
draw(X,Y,[R|G]) :- draw_row(X,Y,R),
              Y1 is Y + 1,
              draw(X,Y1,G).

draw_row(_,_,[]).
draw_row(X,Y,[P|R]) :- cursor(X,Y),
                 write(P),
                 X1 is X + 1,
                 draw_row(X1,Y,R).

  %%% Move an Object up
up(Obj)   :- hide(Obj),
         retract(location(Obj,X,Y)),
         Y1 is Y - 1,
         assert(location(Obj,X,Y1)),
         draw(Obj).
```

```
down(Obj)  :- hide(Obj),
        retract(location(Obj,X,Y)),
        Y1 is Y + 1,
        assert(location(Obj,X,Y1)),
        draw(Obj).

left(Obj)  :- hide(Obj),
        retract(location(Obj,X,Y)),
        X1 is X - 1,
        assert(location(Obj,X1,Y)),
        draw(Obj).

right(Obj) :- hide(Obj),
        retract(location(Obj,X,Y)),
        X1 is X + 1,
        assert(location(Obj,X1,Y)),
        draw(Obj).

:- message.
```