```
Beginning tests for haonanl5 Sat May 26 05:01:15 AEST 2018

Compiling sources

Running a few simple tests
TESTING easy1.txt from 1,1
....
....
....
...W

Starting at (1,1)
Successfully killed wumpus with plan: [south,shoot,south,south,east,shoot,north,
shoot,north,north,east,shoot,south,shoot,south,south,east,north,shoot,north,nort
h]
Feedback: [empty,miss,empty,smell,smell,hit]
ATTEMPTS = 1
ENERGY = 45

TESTING medium1.txt from 2,2
#.P.#
#...#
#.P.#
...W#
#####

Starting at (2,2)
Successfully killed wumpus with plan: [south,shoot,west,shoot,south,shoot,south,
east,shoot,north,shoot,east,shoot,north,shoot,north,east,shoot,north,shoot,east,
shoot,south,shoot,south,west,shoot,south,south,west,east,east,north,south,west,n
orth,south,west,north,south,west,north,south,west,north,shoot,east,east]
Feedback: [smell,miss,wall,miss,smell,miss,wall,stench,hit]
ATTEMPTS = 2
ENERGY = 179

TESTING medium3.txt from 5,6
.......
.P.....
...P...
...W...
.....P.
..P....
.......

Starting at (5,6)
Successfully killed wumpus with plan: [north,south,west,north,shoot]
Feedback: [smell,smell,smell,stench,hit]
ATTEMPTS = 3
ENERGY = 209

TESTING medium4.txt from 1,1
.......
P#...P.
.....P.
...W...
.P.#...
.....##
.......
```

```
Starting at (1,1)
Successfully killed wumpus with plan: [east,east,east,shoot,south,shoot,south,so
uth,west,shoot,west,east,shoot,south,shoot,south,south,east,shoot,north,north,ea
st,shoot,north,shoot,north,north,north,east,south,shoot,south,south,south,south,
west,shoot,south,east,east,north,shoot,north,north,north,north,north]
Feedback: [empty,empty,smell,miss,smell,hit]
ATTEMPTS = 4
ENERGY = 384

TESTING hard1.txt from 4,4
....
....
.P..
....
.W.#

Starting at (4,4)
Successfully killed wumpus with plan: [west,shoot,west,west,north,shoot,north,no
rth,east,shoot,south,shoot,south,east,shoot,north,shoot,north,east,south,shoot,s
outh,south,south,west,shoot,west,west]
Feedback: [smell,miss,stench,smell,smell,miss,empty,empty,empty,miss,smell,hit]
ATTEMPTS = 1
ENERGY = 60


Running formal tests with hidden results
Completed tests Sat May 26 05:01:24 AEST 2018
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author:   Haonan Li <haonanl5@student.unimelb.edu.au>
%
% Intro:    Wumpus is a planning problem. This program is about finding and
%           kill a Wumpus hiding in an unknown maze. The player sends in a
%           series of disposable robots each with a fixed list of instructions
%           to follow. Each robot follows its instructions until it is
%           destroyed or it finishes the instructions, or it kills the Wumpus.
%           After a robot is finished the player gets feedback on what the
%           robot sensed following the instructions. The aim is to explore the
%           maze and find and shoot the Wumpus.
%
% Strategy: a) The main idea of the program is try to explore more positions
%           and if wumpus is detected, find a path to kill it.
%           b) Save map information with several lists, each list saves one
%           kind of position, for example: Unknown list save the unknown
%           positions it will be initialized with all positions except start
%           position.
%           c) To generate a guess, first check if the wumpus's position is
%           known. If yes, find a path to the position that is in the same
%           horizontal or vertical line with wumpus and there is no wall
%           between the position and wumpus, and shoot. If the wumpus is
%           still in unknown positions, explore map.
%           d) To explore the map, first check whether all unkonwn positions
%           not reachable. That is, if there is one unkonwn position whose
%           neighbor is already in Empty set, the position should be reachable
%           although it might be wall or pit.
%           e) Add shoot in the path, each time check if there are positions
%           in front of robot that was not covered by shoot and add a shoot
%           if the answer is yes.
%           f) For very hard map, the wumpus is unreachable and its position
%           can not be inferd from smell or stench informations, shoot the
%           unknown positions whose neighbor is a pit. In this way, all
%           positions hiden behind pits will be covered by shoot.
%           g) Each time when we get feedback, add the positions to the
%           corresponding list in State.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%

:- module(wumpus,[initialState/5, guess/3, updateState/4]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State:    Save all informations about the map and guess histories
% State [
%           Map size and Current POS and direction,
%           Unknown Pos,
%           Empty Pos,
%           Pit Pos,
%           Wall Pos,
%           Shoot Pos,
%           Damp,
%           Wumpus,
%           Smell,
%           Stench
%       ]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% initialState(+R,+C,+X,+Y,-State0)
% Input:    The number of rows R and number of columns C in the game, and the
%           starting position (X,Y)
% Output:   State0, an initial state representation for the game.
initialState(R,C,X,Y,State0) :−
    all_pos(R−C,C−R,All_pos),
    subtract(All_pos,[X−Y],All_pos1),
    State0 = [R−C−X−Y−north,All_pos1,[X−Y],[],[],[],[],[],[],[]].

% all_pos(+R−C,+X−Y,−Pos)
% Input:    The the number of rows and cols R−C, and current processing
%           position
% Output:   All positions in the map
all_pos(R−_,0−R,[]).
all_pos(R−C,X−0,Pos) :−
    X > 0, X1 is X−1,
    all_pos(R−C,X1−R,Pos).
all_pos(R−C,X−Y,[X−Y|Pos]) :−
    Y > 0, Y1 is Y−1,
    all_pos(R−C,X−Y1,Pos).

% explore_path(+State,−State1,−Path)
% Input:    Current state
% Output:   The state after explore State1 and Path with least distance and
%           the end position has not been explored before.
explore_path(State,State1,Path) :−
    State = [_−_−X−Y−D,Unknown,Empty,_,_,_,_,_,_,_],
    reach_able(Unknown,Empty),
    length(Empty,Dis),
    explore_dis_path(Dis,State,[X−Y−D],State1,Path1),
    reverse(Path1,[_|Path]).

% explore_dis_path(+Dis,+State,+Visited,−State,−Path)
% Input:    Distance limit of path Dis, current state State, positions that
%           have been visited in current explore.
% Output:   Path in which each position transient at most once and end with a
%           unknown position, updated state State.
explore_dis_path(Dis,State,Visited,State1,Path) :−
    Dis > 0,
    State = [R−C−_−_−_,Unknown,Empty,_,_,_,_,_,_,_],
    Visited = [X−Y−D1|_],
    stench_around(X−Y,Range),
    (   intersection(Range,Unknown,[U−V|_]) −>
            move(R−C,X−Y−D1,U−V−D2),
            subtract(Unknown,[U−V],Unknown1),
            Attrs = [info,R−C−U−V−D2,empty,[U−V|Empty],unknown,Unknown1],
            set_elements(State,State1,Attrs),
            Path = [U−V−D2|Visited]
    ;   move(R−C,X−Y−D1,U−V−D2),
        (   memberchk(U−V,Unknown) −>
                subtract(Unknown,[U−V],Unknown1),
                Attrs = [info,R−C−U−V−D2,empty,[U−V|Empty],unknown,Unknown1],
                set_elements(State,State1,Attrs),
                Path = [U−V−D2|Visited]
        ;   \+ memberchk(U−V−D2,Visited),
            memberchk(U−V,Empty),
            Dis1 is Dis−1,
            explore_dis_path(Dis1,State,[U−V−D2|Visited],State1,Path)
        )
    ).
```

```
% kill_path(+State,−Path)
% Input:    Current state State.
% Output:   The path Path to kill wumpus if wumpus have been found and all
%           positions in the path are accesible.
kill_path(State,Path) :−
    State = [_−_−X−Y−D1,_,Empty,_,_,Shoot,_,[_],_,_],
    shoot_pos(State,Goal),
    \+ memberchk(Goal,Shoot),
    length(Empty,Dis),
    kill_dis_path(Dis,State,Goal,[X−Y−D1],Path1),
    Path2 = [shoot|Path1],
    reverse(Path2,[_|Path]).

% kill_dis_path(+Dis,+State,+Goal,+Visited,−Path)
% Input:    Limitaion of path length Dis, current state State, Goal which is
%           position and direction expect to achieve, and Visited the contains
%           all visited position in current path.
% Output:   Path: path to the Goal.
kill_dis_path(Dis,State,Goal,Visited,Path) :−
    Dis > 0,
    State = [R−C−_−_−_,_,Empty,_,_,_,_,_,_,_],
    Visited = [X−Y−D1|_],
    move(R−C,X−Y−D1,S−T−D2),
    (   Goal = S−T−D2 −>
            Path = [Goal|Visited]
    ;   \+ memberchk(S−T−D2,Visited),
        memberchk(S−T,Empty),
        Dis1 is Dis−1,
        kill_dis_path(Dis1,State,Goal,[S−T−D2|Visited],Path)
    ).

% shoot_pos(+State,−Goal)
% Input:    Current state State.
% Output:   Goal: good shoot position and direction.
shoot_pos(State,Goal) :−
    State = [R−C−_−_−_,Unknown,Empty,_,Wall,_,_,[U−V],_,_],
    shoot_pos(R−C,Empty,Empty,Unknown,Wall,U−V,Goal).
shoot_pos(R−C,Empty,[X−Y|_],Unknown,Wall,U−V,Goal) :−
    move(R−C,X−Y−east,M−N−D),
    check_shoot(M−N−D,U−V,Wall,Unknown),
    memberchk(M−N,Empty),
    Goal = M−N−D.
shoot_pos(R−C,Empty,[_|Candidate],Unknown,Wall,U−V,Goal) :−
    shoot_pos(R−C,Empty,Candidate,Unknown,Wall,U−V,Goal).

% attempt_kill_path(+State,−State1,−Path)
% Input:    Current state State
% Output:   Path: a path that attempt to kill the wumpus, based on the fact
%           that wumpus position can not be find.
attempt_kill_path(State,State1,Path) :−
    State = [R−C−_−_−_,Unknown,Empty,Pit,_,_,_,[],_,_],
    unknown_shootable_pos(R−C,Pit,Empty,Unknown,Shootable),
    Shootable = [U−V|_],
    set_elements(State,State1,[wumpus,[U−V]]),
    kill_path(State1,Path).

% unknown_shootable_pos(+R−C,+Pit,+Empty,+Unknown,−Shootable)
% Input:    Rows and Cols of the map R−C, Pit, Empty, Unknwon are pit set,
%           empty set and unknown set separately.
% Output:   Shootable, all unknown positions that can be shoot passing a pit.
unknown_shootable_pos(_,[],_,_,[]).
unknown_shootable_pos(R−C,[X−Y|Pit],Empty,Unknown,Shootable) :−
    move(R−C,X−Y−D,U−V−D),
    (   memberchk(U−V,Unknown) −>
            rev_dir(D,D1),
            (   move(R−C,X−Y−D1,M−N−D1) −>
                memberchk(M−N,Empty),
                Shootable = [U−V|Shootable1],
                unknown_shootable_pos(R−C,Pit,Empty,Unknown,Shootable1)
            ;   unknown_shootable_pos(R−C,Pit,Empty,Unknown,Shootable)
            )
    ;   unknown_shootable_pos(R−C,Pit,Empty,Unknown,Shootable)
    ).


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


% guess(+State0,−State,−Guess)
% Input:    Current state State0.
% Output:   new state State, and a Guess which is a list of north, east, south,
%           west, shoot which are instructions for the robot.
guess(State0, State, Guess) :−
    State = State0,
    (   kill_path(State0,Path) −>
        path_to_dir(Path,Guess)
    ;   guess_path(State0,100,Path),
        (   \+ Path = [] −>
                add_shoot(State,Path,Path1),
                path_to_dir(Path1,Path2),
                limit_energy(Path2,100,Guess)
        ;   attempt_kill_path(State0,_,Path),
            path_to_dir(Path,Guess)
        )
    ).

%guess_path(+State,+Energy,−Guess)
% Input:    Current state State, current energy Energy.
% Output:   Guess which is a guessed path without shoot.
guess_path(_,Energy,[]) :−
    Energy =< 0.
guess_path(State,_,_) :−
    \+ explore_path(State,_,_),
    \+ kill_path(State,_).
guess_path(State,Energy,Guess) :−
    Energy > 0,
    (   explore_path(State,State1,Path1) −>
            length(Path1,Dis),
            Energy1 is Energy − Dis
    ;   kill_path(State,Path1) −>
        State1 = State,
        Energy1 = 0
    ),
    append(Path1,Guess1,Guess),
    guess_path(State1,Energy1,Guess1).
```

```
% add_shoot(+State,+Path,-Path1)
% Input:   Current state State, and Path which is a path without shoot.
% Output:   Path1, the path with shoot added.
add_shoot(State,Path,Path1) :-
    (   State = [_,_,_,_,_,_,_,[],_,_] ->
            add_shoot_random(State,Path,Path1)
    ;   add_shoot_wumpus(State,Path,Path1)
    ).

% add_shoot_random(+State,+Path,-Dpath)
% Input:   Current state State, current path Path.
% Output:   Dpath: path with shoot added randomly if the position of wumpus
%           is unknown.
add_shoot_random(_,[],[]).
add_shoot_random(State,[X-Y-D|Path],[X-Y-D,shoot|Dpath]) :-
    State = [R-C-_-_-_,_,_,_,_,_,Shoot,_,_,_,_],
    way_to_edge(R-C,X-Y-D,Pos),
    \+ memberchk(X-Y-D,Shoot),
    \+ subtract(Pos,Shoot,[]),
    shoot_range(R-C,X-Y-D,Shoot,Shoot1),
    set_elements(State,State1,[shoot,Shoot1]),
    add_shoot_random(State1,Path,Dpath).
add_shoot_random(State,[X-Y-D|Path],[X-Y-D|Dpath]) :-
    State = [_,_,_,_,_,Shoot,_,_,_,_],
    memberchk(X-Y-D,Shoot),
    add_shoot_random(State,Path,Dpath).
add_shoot_random(State,[X-Y-D|Path],[X-Y-D|Dpath]) :-
    State = [R-C-_-_-_,_,_,_,_,_,Shoot,_,_,_,_],
    \+ memberchk(X-Y-D,Shoot),
    way_to_edge(R-C,X-Y-D,Pos),
    subtract(Pos,Shoot,[]),
    add_shoot_random(State,Path,Dpath).


% add_shoot_wumpus(+State,+Path,-Dpath)
% Input:   Current state State, current path Path.
% Output:   Dpath which is the result of adding shoot to the path.
add_shoot_wumpus(_,[],[]).
add_shoot_wumpus(State,[P|Path],Dpath) :-
    State = [_,_,_,_,Wall,_,_,[U-V],_,_],
    (   P = X-Y-D ->
        (   check_shoot(X-Y-D,U-V,Wall,[]) ->
                Dpath = [X-Y-D,shoot|Dpath1]
        ;   Dpath = [X-Y-D|Dpath1]
        )
    ;   Dpath = Dpath1
    ),
    add_shoot_wumpus(State,Path,Dpath1).

% limit_energy(+Guess,+Energy,-Guess1)
% Input:   Current guess Guess and Limitation of energy Energy.
% Output:   New guess Guess1 whose energy is no larger than Energy.
limit_energy(_,0,[]).
limit_energy([],_,[]).
limit_energy([shoot|_],Energy,[]) :-
    Energy < 5.
limit_energy([X|Path],Energy,[X|Guess]) :-
    X \= shoot,
    Energy > 0,
    Energy1 is Energy−1,
    limit_energy(Path,Energy1,Guess).
limit_energy([shoot|Path],Energy,[shoot|Guess]) :−
    Energy >= 5,
    Energy1 is Energy−5,
    limit_energy(Path,Energy1,Guess).


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


% updateState(+State0,+Guess,+Feedback,−State)
% Input:    Current state State0, previous guess Guess and the feedback from
%           the guess Feedback.
% Output:   A new updated state State.
updateState(State0, Guess, Feedback, State) :−
    State0 = [Info,_,_,_,_,_,_,_,_,_],
    update_all(Info,Guess,Feedback,State0,State0,State1),
    infer_pit(State1,State2),
    infer_wumpus(State2,State).

% infer_pit(+State,−State1)
% Input:    Current state State
% Output:   State after infering pits, if one position in Damp whose 3
%           neighbors are known and not pit, the last one neighbor will be
%           infered a pit.
infer_pit(State,State1) :−
    State = [_,_,_,_,_,_,Damp,_,_,_],
    infer_pit(Damp,State,State1).

% infer_pit(+Damp,+State,−State1)
% Input:    Current damp set Damp, Current state State.
% Output:   State after infering pits.
infer_pit([],State,State).
infer_pit([X−Y|Remain],State,State1) :−
    State = [_,Unknown,_,Pit,_,_,_,_,_,_],
    stench_around(X−Y,Nearby),
    (   intersection(Nearby,Pit,[]) −>
        (   intersection(Nearby,Unknown,[U−V]) −>
            subtract(Unknown,[U−V],Unknown1),
            Attrs = [unknown,Unknown1,pit,[U−V|Pit]],
            set_elements(State,State2,Attrs),
            infer_pit(Remain,State2,State1)
        ;   infer_pit(Remain,State,State1)
        )
    ;   infer_pit(Remain,State,State1)
    ).

% infer_pit(+State,−State1)
% Input:    Current state State
% Output:   Updated state State1 with infered wumpus if possible.
infer_wumpus(State,State) :−
    State = [_,_,_,_,_,_,_,[_],_,_].
infer_wumpus(State,State1) :−
    State = [_,Unknown,_,_,_,_,_,[],Smell,Stench],
    smell_infer(Smell,Infer),
    stench_infer(Stench,Infer1),
    merge_infer(Infer,Infer1,Infer2),
    intersection(Infer2,Unknown,Wumpus1),
    (   \+ Wumpus1 = [_−_] −>
            State1 = State
    ;   subtract(Unknown,Wumpus1,Unknown1),
        Attrs = [unknown,Unknown1,wumpus,Wumpus1],
        set_elements(State,State1,Attrs)
    ).

% merge_infer(+Range1,+Range2,−Range)
% Input:    Two range R1, R2 contains possible position of wumpus.
% Output:   Infered position of wumpus R.
merge_infer(R1,[],R1).
merge_infer([],R2,R2).
merge_infer(R1,R2,R) :−
    \+ R1 = [],
    \+ R2 = [],
    intersection(R1,R2,R).

% stench_infer(+Stench,−Range)
% Input:    Current stench set Stench.
% Output:   Range: Possible position of wumpus based on stench information.
stench_infer([],[]).
stench_infer([S],Range) :−
    stench_around(S,Range).
stench_infer([S,S1|Stench],Range) :−
    stench_around(S,Range1),
    stench_infer([S1|Stench],Range2),
    intersection(Range1,Range2,Range).

% smell_infer(+Stench,−Range)
% Input:    Current stench set Stench.
% Output:   Range: Possible position of wumpus based on smell information.
smell_infer([],[]).
smell_infer([S],Range) :−
    smell_around(S,Range).
smell_infer([S,S1|Smell],Range) :−
    smell_around(S,Range1),
    smell_infer([S1|Smell],Range2),
    intersection(Range1,Range2,Range).

% update_all(+Info,+Guess,+Feedback,+State0,+State1,−State)
% Input:    Basic map information Info, previous guess Guess and the feedback
%           from the guess FeedBack, current state State0, current updated
%           state State1.
% Output:   Final updated state State.
update_all(_,_,[],_,State,State).
update_all(R−C−X−Y−D1,[Gue|Guess],[Fee|Feedback],State0,State1,State) :−
    update_one(R−C−X−Y−D1,Gue,Fee,U−V−D2,State1,State2),
    update_all(R−C−U−V−D2,Guess,Feedback,State0,State2,State).

% update_onea(+Info,+Guess,+Feedback,+Position,+State0,−State)
% Input:    Basic map informations Info, one guess Guess, one feedback Feedback
%           and current state State0.
% Output:   State1 which is the updated state use one step guess.
update_one(_−_−X−Y−D1,shoot,miss,X−Y−D1,State0,State) :−
    State0 = [_,_,_,_,_,Shoot,_,_,_,_],
    Attrs = [shoot,[X−Y−D1|Shoot]],
    set_elements(State0,State,Attrs).
update_one(R−C−X−Y−D1,Gue,wall,X−Y−Gue,State0,State) :−
```

```
    move(R−C,X−Y−D1,U−V−Gue),
    State0 = [_,Unknown,_,_,Wall,_,_,_,_,_],
    subtract(Unknown,[U−V],Unknown1),
    Attrs = [unknown,Unknown1,wall,[U−V|Wall]],
    set_elements(State0,State,Attrs).
update_one(R−C−X−Y−D1,Gue,wall,X−Y−Gue,State0,State0) :−
    \+ move(R−C,X−Y−D1,_−_−Gue).
update_one(R−C−X−Y−D1,Gue,Fee,U−V−Gue,State0,State) :−
    \+ memberchk(Fee,[miss,wall]),
    move(R−C,X−Y−D1,U−V−Gue),
    get_element(State0,unknown,Unknown),
    subtract(Unknown,[U−V],Unknown1),
    (   memberchk(Fee,[pit,wumpus,empty]) −>
            get_element(State0,Fee,Attr),
            add_to_set(Attr,U−V,Attr1),
            Attrs = [unknown,Unknown1,Fee,Attr1]
    ;   get_element(State0,empty,Empty),
        add_to_set(Empty,U−V,Empty1),
        get_element(State0,Fee,Attr),
        add_to_set(Attr,U−V,Attr1),
        Attrs = [unknown,Unknown1,empty,Empty1,Fee,Attr1]
    ),
    set_elements(State0,State,Attrs).


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


% reverse direction
rev_dir(east,west).
rev_dir(west,east).
rev_dir(north,south).
rev_dir(south,north).

% check if the shoot is aim to wumpus
check_shoot(X−Y−D,U−Y,Wall,_) :−
    pos_between(U−Y,X−Y−D,Pos),
    intersection(Pos,Wall,[]).
check_shoot(X−Y−D,X−V,Wall,_) :−
    pos_between(X−V,X−Y−D,Pos),
    intersection(Pos,Wall,[]).

% move within the map
move(R−C,X−Y−_,U−V−D) :−
    move_dist(A,B,D),
    U is X+A, U>0, U=<C,
    V is Y+B, V>0, V=<R.

% move to one particular direction
move_dist(0,1,south).
move_dist(1,0,east).
move_dist(0,−1,north).
move_dist(−1,0,west).

% make sure there are reachable unknown positions
reach_able([X−Y|_],Empty) :−
    stench_around(X−Y,Neighbor),
    \+ intersection(Neighbor,Empty,[]).
```

```
reach_able([_|Unknown],Empty) :-
    reach_able(Unknown,Empty).

% check one position and move to another in one step
moveable(R-C,X-Y,U-V-D,[]) :-
    \+ move(R-C,X-Y-east,U-V-D).
moveable(R-C,X-Y,U-V-D,[U-V]) :-
    move(R-C,X-Y-east,U-V-D).

% transform from positions to directions
path_to_dir([],[]).
path_to_dir([shoot|Path],[shoot|Dpath]) :-
    path_to_dir(Path,Dpath).
path_to_dir([_-_-D|Path],[D|Dpath]) :-
    path_to_dir(Path,Dpath).

% set particular elements and return a new state.
set_elements(State,State,[]).
set_elements(State,State1,[Attr,Value|Other]) :-
    index_attr(Attr,Index),
    set_element(State,State2,1,Index,11,Value),
    set_elements(State2,State1,Other).

% set one attribute through index
set_element([],[],11,_,11,_).
set_element([A|State],[B|State1],Cur,Obj,End,Value) :-
    Cur < End,
    (   Cur = Obj ->
            B = Value
    ;   B = A
    ),
    Cur1 is Cur+1,
    set_element(State,State1,Cur1,Obj,End,Value).

% get one attribute of the state
get_element(State,Attr,Value) :-
    index_attr(Attr,Index),
    get_element(State,1,Index,Value).
get_element([A|State],Cur,Obj,Value) :-
    (   Cur \= Obj ->
            Cur1 is Cur+1,
            get_element(State,Cur1,Obj,Value)
    ;   Value = A
    ).

% from start pos and go straight to the edge.
way_to_edge(_,X-Y-west,Pos) :-
    pos_between(1-Y,X-Y-west,Pos).
way_to_edge(_-C,X-Y-east,Pos) :-
    pos_between(C-Y,X-Y-east,Pos).
way_to_edge(_,X-Y-north,Pos) :-
    pos_between(X-1,X-Y-north,Pos).
way_to_edge(R-_,X-Y-south,Pos) :-
    pos_between(X-R,X-Y-south,Pos).

% positions between current and aim
pos_between(U-V,U-V-_,[U-V]).
pos_between(U-V,X-Y-west,[X-Y|Pos]) :-
    X > U,
```

```prolog
    X1 is X-1,
    pos_between(U-V,X1-Y-west,Pos).
pos_between(U-V,X-Y-east,[X-Y|Pos]) :-
    X < U,
    X1 is X+1,
    pos_between(U-V,X1-Y-east,Pos).
pos_between(U-V,X-Y-north,[X-Y|Pos]) :-
    Y > V,
    Y1 is Y-1,
    pos_between(U-V,X-Y1-north,Pos).
pos_between(U-V,X-Y-south,[X-Y|Pos]) :-
    Y < V,
    Y1 is Y+1,
    pos_between(U-V,X-Y1-south,Pos).

% add shoot range to the shooted set
shoot_range(R-C,X-Y-D,Shoot,Shoot1) :-
    way_to_edge(R-C,X-Y-D,Pos),
    append(Pos,Shoot,Shoot1).

% range of stench
stench_around(X-Y,Range) :-
    X1 is X-1, X3 is X+1,
    Y1 is Y-1, Y3 is Y+1,
    Range = [X1-Y,X-Y1,X-Y3,X3-Y].

% range of smell
smell_around(X-Y,Range) :-
    X0 is X-3, X1 is X-2, X2 is X-1, X4 is X+1, X5 is X+2, X6 is X+3,
    Y0 is Y-3, Y1 is Y-2, Y2 is Y-1, Y4 is Y+1, Y5 is Y+2, Y6 is Y+3,
    Range = [X0-Y,X1-Y2,X1-Y,X1-Y4,X2-Y1,X2-Y2,X2-Y,X2-Y4,X2-Y5,X-Y0,X-Y1,X-Y2,
        X-Y4,X-Y5,X-Y6,X4-Y1,X4-Y2,X4-Y,X4-Y4,X4-Y5,X5-Y2,X5-Y,X5-Y4,X6-Y].

% add an element to a set
add_to_set(Set,Elm,Set1) :-
    (   memberchk(Elm,Set) ->
        Set1 = Set
    ;   append(Set,[Elm],Set1)
    ).

% map of attributes and its index
index_attr(info,1).
index_attr(unknown,2).
index_attr(empty,3).
index_attr(pit,4).
index_attr(wall,5).
index_attr(shoot,6).
index_attr(damp,7).
index_attr(wumpus,8).
index_attr(smell,9).
index_attr(stench,10).
```