Possible answer code for V-World Workshop 2

Define is_bumble_hungry/1

so Bumble becomes hungry (assert(hungry)) when his Strength drops below 1000, and he ceases to be hungry (retractall(hungry)) when his Strength goes above 3000. When his Strength is between 1000 and 2999, his hunger state should not change. So sometimes he will be hungry and sometimes he won't when his Strength is in this range.

Define is_bumble_hurt/1

so Bumble becomes hurt (assert(hurt)) when his Damage goes above 50, and he ceases to be hurt (retractall(hurt)) when his Damage goes down to 0. When his Damage is between 0 and 50, his hurt state should not change. This will not matter in the risk worlds because the only way Bumble can reduce his Damage in these worlds is to press a red cross, and this drops his Damage all the way to 0. But in other worlds, there may be actions that will reduce Bumble's Damage by some fixed amount without reducing it all the way to 0.

Your code could now look like this:

```
agent(Perceptions,Action) :-
        set_bumble_states(Perceptions),
        bumble(Perceptions,Action).

set_bumble_states([Strength,Damage|_]):-
        is_bumble_hungry(Strength),
        is_bumble_hurt(Damage).

is_bumble_hungry(Strength):-
        Strength < 1000,            % if Strength is less than 1000
        assert(hungry).             % then bumble is hungry

is_bumble_hungry(Strength):-
        Strength > 3000,            % if Strength is more than 3000
        retractall(hungry).         % then bumble is not hungry

is_bumble_hungry(Strength).         % otherwise no change


is_bumble_hurt(Damage):-
        Damage > 50,                % if Damage is more than 50
        assert(hurt).               % then bumble is hurt

is_bumble_hurt(Damage):-
        Damage = 0,                 % if Damage goes down to 0
        retractall(hurt).           % then bumble is not hurt

is_bumble_hurt(Damage).             % otherwise no change 1-49
```

Changing Bumble's behaviour (possible) code

1. If Bumble is not hungry, he ignores fruit. Simply modify the clause that relates to consuming fruit to test for 'hungry' as below
2. If Bumble is hungry and he is beside fruit, he eats it.

```
% Consume any nearby fruit.

bumble([_,_,_,_,_,
        [_,NW,N,NE,_],[_,W,_,E,_],[_,SW,S,SE,_],_],move(Dir)) :-
        member(fruit,[NW,N,NE,W,E,SW,S,SE],Num),
        hungry,                         % only consume fruit if hungry
        member(Dir,[nw,n,ne,w,e,sw,s,se],Num).
```

3. If Bumble is hungry and he is beside a tree but he is not beside fruit, he presses on the tree. Remember, the 'consume fruit' related clause comes before the 'push interesting objects' ones, i.e. this will be tested & auctioned first

```
% Try pushing nearby interesting objects (but not tree: below)
% pushing trees is associated with being hungry


bumble([_,_,_,_,_,[_,NW,N,NE,_],[_,W,_,E,_],[_,SW,S,SE,_],_],move(Dir)) :-

        \+ retract(pushed),
        member(Obj,[NW,N,NE,W,E,SW,S,SE],Num),
        interesting(Obj),
        \+ Obj = tree,          % not if a tree – special case
        member(Dir,[nw,n,ne,w,e,sw,s,se],Num),
        assert(pushed).

% Try pushing nearby interesting objects if it is a tree
% trees are interesting & should be pushed only if hungry

bumble([_,_,_,_,_,[_,NW,N,NE,_],[_,W,_,E,_],[_,SW,S,SE,_],_],move(Dir)) :-

        \+ retract(pushed),
        member(Obj,[NW,N,NE,W,E,SW,S,SE],Num),
        interesting(Obj),
        Obj = tree,     % only if a tree – special case
        hungry,                 % and only if hungry
        member(Dir,[nw,n,ne,w,e,sw,s,se],Num),
        assert(pushed).

% Trees, red croses, and birds are interesting.

interesting(tree).
interesting(cross).
interesting(bird).
```

4. If Bumble is not hurt, he ignores hornets and red crosses. If Bumble is hurt, and he is beside a red cross, he presses it.

```
% Try pushing nearby interesting objects
% (but not trees or crosses - see below)
% pushing trees is associated with being hungry
% pushing crosses only when hurt

bumble([_,_,_,_,_,[_,NW,N,NE,_],[_,W,_,E,_],[_,SW,S,SE,_],_],move(Dir)) :-

        \+ retract(pushed),
        member(Obj,[NW,N,NE,W,E,SW,S,SE],Num),
        interesting(Obj),
        \+ Obj = tree,
        \+ Obj = cross,
        member(Dir,[nw,n,ne,w,e,sw,s,se],Num),
        assert(pushed).

% Try pushing nearby interesting objects if it is a tree
% trees are interesting & should be pushed only if hungry

bumble([_,_,_,_,_,[_,NW,N,NE,_],[_,W,_,E,_],[_,SW,S,SE,_],_],move(Dir)) :-

        \+ retract(pushed),
        member(Obj,[NW,N,NE,W,E,SW,S,SE],Num),
        interesting(Obj),
        Obj = tree,
        hungry,                 % but only if hungry
        member(Dir,[nw,n,ne,w,e,sw,s,se],Num),
        assert(pushed).

% Try pushing nearby interesting objects if it is a red cross
% red crosses are interesting & should be pushed only if hurt

bumble([_,_,_,_,_,[_,NW,N,NE,_],[_,W,_,E,_],[_,SW,S,SE,_],_],move(Dir)) :-

        \+ retract(pushed),
        member(Obj,[NW,N,NE,W,E,SW,S,SE],Num),
        interesting(Obj),
        Obj = cross,
        hurt,               % but only if hurt
        member(Dir,[nw,n,ne,w,e,sw,s,se],Num),

        assert(pushed).
```