Possible Code for implementing the behaviours suggested in Workshop 3

Suggested behaviour in the original workshop:

Notice that while a 5 x 5 piece of its world is visible to it, Bumble 1.1 only uses information about the smaller 3 x 3 piece of the world immediately adjacent to it. So it does not use all the information available to it. It seems likely that Bumble could survive longer if it used some of this information it is ignoring. Produce a new version of Bumble, Bumble 1.2 that displays the following behaviours.

1.0 If Bumble is hungry and there is no tree next to Bumble, but there is a tree two squares away from Bumble, and Bumble can move to an empty space next to the tree, then Bumble moves next to the tree.

2.0 If Bumble is hurt and there is no cross next to Bumble, but there is a cross two squares away from Bumble, and Bumble can move to an empty space next to the cross, then Bumble moves next to the cross.

3.0 If none of the above applies and Bumble is within two moves of something interesting or desirable (a key, bug spray, etc.,) then Bumble moves toward the desirable object.

Ok, we know that the predicate bumble(Perceptions,Actions) has a 5x5 matrix of perceptions around itself, and we are told to now use it. Remember that the full argument list is as below:

```
bumble([Strength, Damage, Itinerary, Last_move,
                [NWNW,NNW,NN,NNE,NENE],
                [WNW,NW,N,NE,ENE],
                [WW,W,HERE,E,EE],
                [WSW,SW,S,SE,ESE],
                [SWSW,SSW,SS,SSE,SESE]
        ]
        move(Dir)).
```

In the simple (bumble_1.0/ 1.1) agent, only the immediately adjacent tiles were instantiated and used. Bumble's behaviour was based purely on adjacency. The code for the pushing tree behaviour (behaviour 1.0 above) from bumble_1.1.agt is shown below:

```
% 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).
```

Now we are told to incorporate the full 5x5 array. In my attempt at this, I have created another tree-pushing behaviour that Prolog will try if the simple adjacency clause fails. I have therefore used anonymous variables (underscores) for where the previous behaviour has already been tried, i.e. I am not now interested in adjacent tiles, but tiles on the periphery of the array:

```
% but if a tree is 2 tiles away, move towards it

bumble([_,_,_,_,[NWNW,NNW,NN,NNE,NENE],[WNW,_,_,_,ENE],
            [WW,_,_,_,EE],[WSW,_,_,_,ESE],
            [SWSW,SSW,SS,SSE,SESE]],move(Dir)) :-

%       \+ retract(pushed),
       member(Obj,[NWNW,NNW,NN,NNE,NENE,
                   SWSW,SSW,SS,SSE,SESE,
                   WNW,WW,WSW,ENE,EE,ESE],Num),
       interesting(Obj),
       Obj = tree,
       hungry,              % but only if hungry
       member(GoToward,[nwnw,nnw,nn,nne,nene,
                        swsw,ssw,ss,sse,sese,
                        wnw,ww,wsw,ene,ee,ese],Num),
       move_toward_x(GoToward,Dir).
% may need some test here for if it is possible to occupy
% this cell (does it have a nasty thing there?)
```

This looks very similar to the previous clause. Notice that I have in fact copy/ pasted it from the original, commented out ‘retract(pushed)’ and deleted ‘assert(pushed).’ This is because this clause is not actually able to do that in one turn – it can only move towards the tree here.

I have also kept to the original sequence of finding a member of the list of (now peripheral) tiles for an object, checking if it is ‘interesting’ and then if it is a tree. I could easily have tested directly for a tree. The next line after checking for hungry is important. From the first member/3 clause and its subsequent choices, we know if there is a tree on the periphery of bumble’s visual perceptions, and in which tile it is… Num holds the list position. Nute suggests a table of directions & so I have created one. Remember that a search algorithm could (& perhaps should) be used here – but in order to follow the original workshop, my version of the table of directions follows. You might like to draw this as graphs

```
/* Table of directions as specified in Nute's V-World
... Our Workshop 3
*/
move_toward_x(nwnw,nw). % top left corner
move_toward_x(nwnw,n).
move_toward_x(nwnw,w).
%
move_toward_x(nene,ne). % top right corner
move_toward_x(nene,n).
move_toward_x(nene,e).
%
move_toward_x(swsw,sw). % bottom left corner
move_toward_x(swsw,s).
move_toward_x(swsw,w).
%
```

```
move_toward_x(sese,se). % botton right corner
move_toward_x(sese,s).
move_toward_x(sese,e).
% a - top left of centre
move_toward_x(nnw,w).
move_toward_x(nnw,nw).
% b - top centre
move_toward_x(nn,n).
move_toward_x(nn,nw).
move_toward_x(nn,ne).
% c - top right of centre
move_toward_x(nne,n).
move_toward_x(nne,ne).
% d - bottom left of centre
move_toward_x(ssw,s).
move_toward_x(ssw,sw).
% e - bottom centre
move_toward_x(ss,s).
move_toward_x(ss,sw).
move_toward_x(ss,se).
% f - bottom right of centre
move_toward_x(sse,s).
move_toward_x(sse,se).
% g - left up from centre
move_toward_x(wnw,w).
move_toward_x(wnw,nw).
% h - left centre
move_toward_x(ww,w).
move_toward_x(ww,nw).
move_toward_x(ww,sw).
% i - left below centre
move_toward_x(wsw,w).
move_toward_x(wsw,sw).
% j - right up from centre
move_toward_x(ene,e).
move_toward_x(ene,ne).
% k - right centre
move_toward_x(ee,e).
move_toward_x(ee,ne).
move_toward_x(ene,se).
% l - right below centre
move_toward_x(ese,e).
move_toward_x(ese,se).

%%%%%%%%%%%%% end of table of directions %%%%
```

From the first member/3 clause we know the position in the list that the tree is, so in the second member/3 clause, the actual tile is identified (nwnw, say). The subsequent ‘move_toward_x/2 clause will access the table of directions and instantiate Dir with the tile that bumble should move to.

The same concepts could be used for the suggested behaviours 2.0 & 3.0 above.

Testing

You will want to verify that the new behaviours actually work. Here is a suggestion.

1. Save the new agent code – export it as ‘All Files’ with a .agt suffix & save it as bumble_1.2 (or whatever you like)
2. Load a world – say test5.vw
3. Load your new agent (bumble_1.2.agt if you have named it thus)
4. Put bumble in the left hand room at the top ... Options/ Change Agent Location
5. Change to manual test mode… Options/ Manual Testing Mode
6. Reduce bumble’s strength to less that 1000 (you can just hit the up arrow to force bumble to constantly hit against the wall until this happens. I have to find a better way to accomplish this in code (later)
7. Put bumble in the furthest (bottom right say) corner of the central room that contains the tree – this is 2 tiles away from the tree
8. Creep & see that the agent moves toward the tree. Try putting bumble in other places 2 tiles away & note its behaviour – verify that it is as expected
9. Put a spypoint on move_toward_x/2 and creep again from 2 tiles away with bumble hungry
10. Creep some more until bumble has eaten enough to make it not hungry. Try the above experiments again.