V-World Workshop 2: Watch Bumble Go! (Using test1 – test7)

Exercise 2 *(Continued from Workshop 1)*

Start V-World, load the test1 world, and load bumble_1.0.agt.
This time, instead of playing the game manually we will let Bumble play the game. Click Leap on the menu bar. When Bumble finally dies, note how many moves he made, and his final Strength and Damage. If Bumble should survive for more than 5,000 moves, click Abort to stop him and record the number of moves as 5,000.

Note: the following exercises are Nute's experiments. They are a little long-winded, so I don't suggest you do them. The approach is valid in terms of experimental procedure, but for the purposes of this workshop, perhaps a little too much *Reload test1 and Bumble, and run him again. Do it a third time. Record the number of moves and the causes of death (starved or killed) for the three runs.*

Run Bumble for three times in each of the worlds test2--test7, recording the number of moves and the cause of death for each run in each world. Prepare your results in a table.

Bumble moves very fast. To better see what he is doing, select Creep from the menu bar. This will cause Bumble to make one move. By doing this repeatedly, you can get a better idea of how Bumble behaves.

2.1 Evaluate Bumble's behaviour

After you have conducted these experiments, and before you look at the code in bumble_1.0.agt, write a description of how Bumble behaves. From this description, you should be able to predict Bumble's behaviour reasonably well in different situations.

Now look over the code in bumble_1.0.agt carefully. Notice that Bumble is defined as a function from perceptions to actions.

Load Bumble into test1 and move him into different situations using Options/Change Agent Location. Look at the code in bumble_1.0.agt and try to predict how Bumble will move before you click Creep. Run Bumble for one step and see how good your prediction is.

Discussion: The worlds test1--test7 contain only walls, trees, fruit, red crosses, hornets, snails, bug spray, keys, doors, and empty space. Bumble does well in test1 because this environment is resource-rich and the threat level is low (lots more trees and red crosses than hornets.) In test2 and test3, it becomes increasingly difficult for Bumble to survive because resources become scarcer compared to threats. The hornets usually do not kill Bumble directly, but they keep him away from the fruit so that he eventually starves. test4--test7 are very dangerous for Bumble because it is difficult for him to find the resources he needs to stay alive. Bumble should do better in test2 and test3 if he were less timid and used the food resources more efficiently.

Exercise 3: Bumble 1.1 – changing the agent's behaviour

Bumble 1.0 (the agent defined in bumble_1.0.agt) does not modify its behaviour in response to changes in its Strength or Damage. We will begin by introducing states that our agent may be in and that affect his behaviour. He should act differently when he is hungry or hurt. You will modify Bumble so that he becomes hungry when his Strength drops below 1000, and he ceases to be hungry when his Strength goes above 3000. And you will modify Bumble so that he is hurt when his Damage goes above 50, and he ceases to be hurt when his Damage drops back to 0.

Look at the first line of code in bumble_1.0.agt, that is, the first line in the file after the comments that begin with %. It says

:- dynamic [agent/2,tried/0,last/1,pushed/0].

This line tells WIN-PROLOG that these predicates are dynamic, that is, that clauses for these predicates can be added to the program, deleted from the program, or examined during program execution using the built-in Prolog predicate clause/2. (The V-World program operates by using clause/2 to look at the definition of agent/2 in an agent file. The other predicates in this list are used by Bumble to keep track of what it did last.).

Change this line to read:

:- dynamic [agent/2,tried/0,last/1,pushed/0,hungry/0,hurt/0].

Now our agent program can add the clauses hungry. or hurt. to itself as it runs and use the presence or absence of these clauses to help it decide what to do in any situation.

The next few lines of code in Bumble 1.0 are

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

In V-World, an agent is defined as a function from perceptions to actions. Our sample agent computes this function by calling the function bumble/2. We will modify Bumble so that before he decides on what action to take, he first determines what state he is in.

Change the definition of agent/2 to

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

Then add the following code immediately below the definition of agent/2.

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

You may want to add more conditions to this code later as you find more useful states to use in steering Bumble's behaviour, but these will do for now. Notice that **Bumble's Perceptions consist of a list of values**, and the first two of these are his Strength and his Damage. We only need to use Strength to determine whether Bumble is hungry, and we only need to use Damage to determine whether he is hurt.

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.

A set of possible modifications to the code can be found in the file:

V-World answer code for Workshop2

Take a look at this code if needed. It is expected that you will need this information at this stage in the learning process

Once you have states established, you can use them to modify Bumble's behaviour. If you want him to behave in a certain way if he is hurt, then add the condition hurt to the rule defining that particular behaviour; if you want him to act some other way when he is not hurt, then add the condition **\+ hurt** to that rule. Note:

\+ Goal is the standard (ISO) for **negation by failure**. It succeeds if the given Goal fails and fails if the given Goal succeeds. It is very similar to **not Goal** (but not exactly the same – see WIN-PROLOG Help).

You can also use conditions like this to define is_bumble_hungry/1 and is_bumble_hurt/1.

Here are the changes in behaviour you should build into Bumble 1.1.

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.

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 & actioned first.

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.

Again, a set of possible modifications to the code can be found in the file:

V-World answer code for Workshop2

Take a look at this code if needed. It is expected that you will need this information at this stage in the learning process

Other than these changes, Bumble 1.1 should behave just like Bumble 1.0.

Export your code to a file called bumble_1.1.agt and test it in the worlds test1--test7. **Important**: You cannot just 'Save' agent files. WIN-PROLOG will only allow you to save standard Prolog file (or Flex ones if you are in Flex). In order to save agent files, it is necessary to EXPORT them instead.

Refer to the separate file:

Developing and saving Agent Files in V-World

Discussion: Bumble 1.1 should perform better than Bumble 1.0in test2 and test3. In fact, Bumble 1.1 should routinely survive 5,000 moves in test2, and at least occasionally survive 5,000 moves in test3. When Bumble 1.1 dies in test3, it will usually be because he was stung to death rather than because he starved. The hornets have a tough time keeping Bumble 1.1 away from food. However, Bumble 1.1 does not perform better than Bumble 1.0 in test4--test7 because both versions have the same problems locating the resources they need to survive.