Agents with Memories

1 Adding a memory of previous percepts: motivation

At this point, we put aside multi-agent systems and return to our consideration of environments in which there is only one agent. But, we now move away from reactive agents and begin to consider agents that are more intelligent. Most important will be to start to consider agents that 'plan ahead'. But, before we do that, we'll make a smaller change: we'll give our agent a memory of previous percepts.

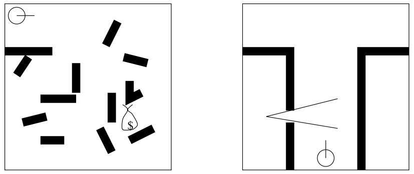
The need for memory may seem obvious. The more complicated the task, the more likely it is that the agent needs memory in order to accomplish the task. But we'll try to argue this more precisely.

Class exercise. Why is it helpful to remember previous percepts at all?

Before we look at memory proper, let's make one more observation. Agents can sometimes avoid using an (internal) memory: the world can act as their memory. When they need to 'store' something, they might be able to carry out an action on the world whose effects can later be sensed when they need to 'recall' what they 'stored'.

Class exercise. Here are two tasks that seem to require memory. But both can be solved by agents with no memory. In each case, how could you build a purely reactive agent that could solve the task?

1. In the left-hand diagram, an agent must randomly traverse an object-strewn world until it finds some treasure. Then, it must pick up the treasure and retrace its steps back to its starting point.
2. In the right-hand diagram, an agent must travel down a corridor. Halfway down the corridor, a light might (or might not) be flashed at the agent. Later, when the agent reaches the T-junction, it must turn left if, earlier, the light had been flashed; it must turn right if, earlier, the light had not been flashed.

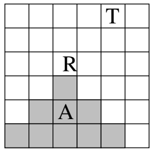


Using the world as your 'memory' can be cumbersome. An agent might be reluctant to modify the world, just for the sake of 'storing' information; when the time comes for 'recall', the part of the environment where the information is 'stored' might not be immediately reachable by the agent's sensors; and, if there are other agents in the world, they can corrupt the information that an agent has 'stored' in the world. These problems do not arise (or arise to a lesser extent) with 'internal' memory.

2 Belief states

If the environment is only partially observable, then the agent knows only a certain amount about the actual state of the world. One way to think about this is that, from the agent's point-of-view, its current percepts are consistent with a *set* of states of the world. We call such a set of states a *belief state*. A belief state represents the agent's current belief about which actual states it might currently be in.

Suppose we are programming a game and we are implementing an AI agent, visually depiected as a robot. Suppose our agent knows that, in this game, there is an alien and a tree. Finally, suppose this is the current state of the world (with our agent's visual field highlighted in grey):



R = Robot; A = Alien; T = Tree

Our agent can use its vision sensors to detect the position of the alien: $\mathit{position}(\mathit{alien}) = \langle 2, 1\rangle$. But, because the environment is only partially observable, our agent does not know where the tree is. The agent's belief state (the set of all world states that are consistent with the agent's current percepts) is:

$$\begin{aligned}\{\quad &\{\mathit{position}(\mathit{alien}) = \langle 2,1\rangle, \mathit{position}(\mathit{tree}) = \langle 0,5\rangle\},\\ &\{\mathit{position}(\mathit{alien}) = \langle 2,1\rangle, \mathit{position}(\mathit{tree}) = \langle 1,5\rangle\},\\ &\{\mathit{position}(\mathit{alien}) = \langle 2,1\rangle, \mathit{position}(\mathit{tree}) = \langle 2,5\rangle\}, \ldots \quad \}\end{aligned}$$

Class exercise. What can you say about the belief states of an agent whose environment is fully observable?

One way to reduce the number of states in a belief state is to use a memory. Memorised percepts may fill in some of the gaps left by the current percepts: the number of states that are consistent with the current percepts and the memorised percepts is likely to be smaller than the number of states that are consistent with the current percepts alone.

In the example above, if our agent had previously seen that the tree was in position $\langle 4, 5\rangle$, then the belief state is reduced to:

$$\{\{\mathit{position}(\mathit{alien}) = \langle 2,1\rangle, \mathit{position}(\mathit{tree}) = \langle 4,5\rangle\}\}$$

Of course, there is a risk in a dynamic environment that the memorised percepts become out-of-date. In the example above, if the tree is a magic tree that can move, then it is not safe to assume that the tree is still in position $\langle 4, 5\rangle$, where we saw it earlier. But, if we know things about the speed and ease with which magic trees can move, we can make predictions about where the magic tree now might be: our belief state would contain states consistent with our current percepts and predictions based on the current and memorised percepts. We'll discuss this issue further in the next lecture.

3 Adding a memory of previous percepts

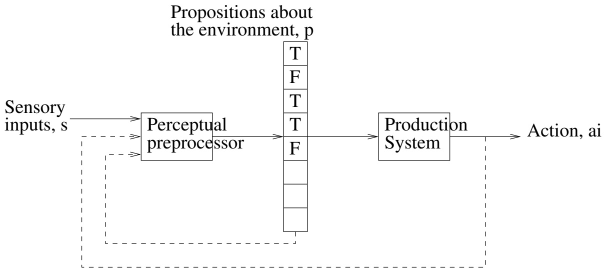
So we now want to consider agents that have a different action function from those that we have been considering. In our reactive agents, the action function maps

- from $\mathbf{s}$ (the immediate sensory inputs)
- to $\mathbf{a}$ (the actions the agent can perform).

Now, we're considering agents whose action function maps

- from $\mathbf{s}$ (the immediate sensory inputs) *and* $\mathbf{m}$ (what's in its memory) *and* a_i (the agent's previous action)
- to $\mathbf{a}$ (the actions the agent can perform).

There are many ways of organising such an agent. The diagram shows one of these, in the case of an agent whose action function is implemented using a production system:



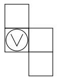
The inputs to the perceptual preprocessor are the agent's sensory inputs, $\mathbf{s}$, the previous vector of truth-values, $\mathbf{p}$, and the action the agent has just executed. In pseudocode, the agent does the following at each point in time:

$\mathbf{s} := \text{SENSE}()$;
$\mathbf{p} := \text{PREPROCESS}(\mathbf{s}, \mathbf{p}, \mathit{action})$;
$\mathit{conflictSet} := \text{FINDALLMATCHES}(\mathbf{p}, \mathit{rules})$;
$\mathit{rule} := \text{CHOOSEARULE}(\mathit{conflictSet})$;
$\mathit{action} :=$ the action part of rule;
$\text{EXECUTE}(\mathit{action})$;

As usual, the conflict resolution strategy we will adopt is the one that chooses the *first* matching rule.

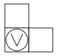
3.1 Example A

Consider the softbot that walks anticlockwise around the walls of grid-based rooms. We know it only needs three touch sensors:



These three sensors are sufficient for the task in hand. So, if it has these three sensors, then we will say that the environment is *fully observable*.

However, here we are going to deprive it of one of the sensors, leaving it with just two:



These two are not sufficient. This agent now operates in a *partially observable* environment. (Think about its belief states.) But, with a memory, the agent can still successfully walk anticlockwise around the walls of the room!

The agent will use three propositions, which we will refer to as p_0, p_2 and p_3 for consistency with previous lectures, and we will assume each is initially F. The perceptual preprocessor will set the truth-values of these propositions as follows:

- p_0 will be T iff sensor s_0 (the one at the agent's front) returns 1;
- p_2 will be T iff sensor s_2 (the one at the agent's right-hand side) returns 1;
- p_3 will be T iff proposition p_2 was T in the previous step.

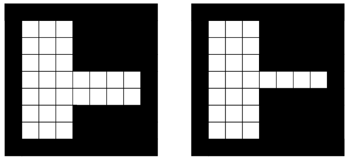
One possible production system is:

if $\neg p_2 \wedge p_3$ **then** `Turn(RIGHT, 2)`
if p_0 **then** `Turn(LEFT, 2)`
if $\neg p_0$ **then** `Move`

3.2 Example B

You might recall that, even if we equip this agent with all 8 touch sensors, if the agent operates purely reactively, there are rooms it cannot successfully circumnavigate. In particular, if there are 'tight' dead-end corridors (only one cell wide), it cannot reliably walk the walls of the room:

Our agent can cope with this... but not with this.



The reason is that there are two states of this world that look identical to the agent but which require different actions.

Let's see how we might solve this using a memory. One solution is to use a proposition q which will be set to T on the agent's first visit to the mouth of a tight dead-end corridor, and set to F on the agent's second visit, when it has returned to the mouth of the tight dead-end corridor. The agent will turn right when q is T, and move forward when q is F.

Here is what the perceptual preprocessor would do:

- p_0 will be T iff sensor s_0 (the one at the agent's front) returns 1;
- p_1 will be T iff sensor s_1 (the one at the agent's front right) returns 1;
- p_2 will be T iff sensor s_2 (the one at the agent's right-hand side) returns 1;
- p_3 will be T iff sensor s_3 (the one at the agent's back right) returns 1;
- q will be set to T iff the previous action was a `Move` action.

Here are the condition-action rules:

if $q \wedge \neg p_2 \wedge p_3$ **then** `Turn(RIGHT, 2)`
if $\neg q \wedge p_1 \wedge \neg p_2$ **then** `Move`
if p_0 **then** `Turn(LEFT, 2)`
if $\neg p_0$ **then** `Move`

Rule ordering is critical here.

Note that this even works if there are corridors that lead off other corridors.