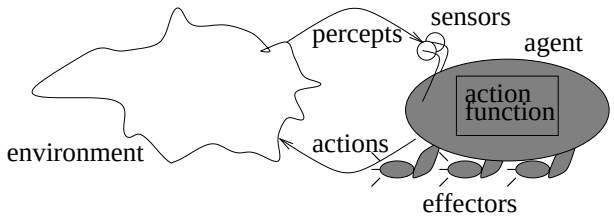
Agents

1 Sense, Plan, Act

The word 'agent' is heavily used in AI and Computer Science nowadays. Here we attempt to explain what the word means in AI.

> *"An* agent *is anything that can be viewed as* perceiving *its* environment *through* sensors *and* acting *upon that environment through* effectors*."*



Robots are *embodied* agents, *situated* in physical environments, e.g. planetary rovers, warehouse part pickers.

Softbots are *software agents situated* in virtual environments such as the Internet, a game environment, a software simulation or just taking input from the keyboard and displaying output on the screen, e.g. WWW bargain hunting agents, diary management agents, theorem-provers, medical consultancy systems, careers advisory systems.

At some level of abstraction, agents execute a *sense/plan/act cycle*:

- *Sense:* Use sensors to find things out about the environment (including the effects of previous actions).
- *Plan:* Decide on the next action(s).
- *Act:* Use effectors to carry out the chosen action(s).

In AI, our focus is the *Plan* phase.

The task of the Plan phase is to implement an *action function* that maps

- from *percept sequences* (the sequence of things the agent has perceived so far)
- to the actions the agent can perform (to give the next action(s)).

2 Intelligent Agents

It is really the *Plan* phase that determines whether we have an *intelligent agent*:

- *Rationality:* For each percept sequence, a rational agent chooses the action that it expects will maximize success.

- *Autonomy:* Three kinds of computation contribute to the definition of an agent function:
  1. Some of the computation about what action to do when is done by the agent's designers or other controlling agents;
  2. some of the computation is done by the agent itself on receipt of the latest percept; and
  3. some of the computation is done by the agent when it modifies the definition of its action function, i.e. when it learns from experience.

  The more an agent relies on the first of these, we say that it lacks autonomy.

3 Different Types of Environments and Agents

3.1 Environments

Here we draw some distinctions about different types of environments. Note, however, in some cases the distinctions need to be drawn from the perspective of a particular agent. Different agents in the same environment might categorise the environment differently.

- **Fully observable vs. partially observable**

  Fully observable: The agent's sensors give it access to the complete state of the environment at each point in time or, at least, to all aspects of the state of the environment that are relevant to the agent's decisions.

  Partially observable: Not all aspects of the state of the environment are available to the agent. Perhaps the sensors are noisy or inaccurate; perhaps they are limited in range or direction. In particular, an agent cannot sense now what happened in the past; if this is relevant to decision making, then the agent needs some form of memory. In particular also, an agent cannot in general sense what is going on in another agent's head; if this is relevant to decision making, some form of inference is needed.

- **Deterministic vs. stochastic**

  Deterministic: The next state of the environment is completely determined by the current state and the action executed by the agent.

  Stochastic: The next state is not completely determined. Obviously the next state isn't completely determined when you roll a dice, when you test whether a door is locked or not, or when you carry out any action that you might bungle! Note from these examples that, even if the world is deterministic but only partially observable, then it may *appear* to be stochastic to the agent, and this poses much the same set of difficulties in agent design.

- **Single-step vs. sequential**

  Single-step: In a single-step environment, the agent senses its environment and then chooses and performs an action. Crucially, subsequent sense-plan-act cycles are independent of previous ones: the choice of action depends only on the current cycle. Many *classification* tasks are single-step. For example, consider an agent that has to spot defective parts on an assembly line (i.e. it looks at each part that passes along the assembly line and, on the basis of what it senses about this part, it classifies the part as *defective* or *not defective*). Presumably, the agent bases each decision on the current part alone; the current decision doesn't affect whether the next part is defective.

  Sequential: The decision taken in the current sense-plan-act cycle could affect all future decisions. Navigating through a maze, playing chess, driving a vehicle, controlling a factory production line, and conducting a conversation, e.g., with a patient who requires a diagnosis or a customer who seeks a product are all (likely) examples of sequential environments.

- **Static vs. dynamic**

  Static: While the agent is choosing its action (the 'plan' step), the environment cannot change. Choosing chess moves (when not done against the clock) is an example.

  Dynamic: The environment can change while the agent is 'thinking'. Grass keeps growing; the patient's condition keeps deteriorating; other soccer players on the pitch keep playing; ...

- **Discrete vs. continuous**

  Discrete: An environment may be discrete in space or time, in which case variables that describe the environment take on values of fixed finite precision.

  Continuous: Variables which describe the environment can theoretically take on values of any precision. Of course, all digital equipment represents values with only finite precision. This means that, strictly speaking, for agents built from digital devices, all environments are discrete.

- **Single agent vs. multi-agent**

  Single agent: There is only one agent, 'our' agent.

  Multi-agent: There is more than one agent. There are subtleties here. Which other objects should 'our' agent regard as being other agents? Clearly, the chess opponent is another agent. But is the wind another agent? What about all other drivers on the road? 'Our' agent must treat another object as an agent if (a) the other object has some measure of success that is is trying to maximise; (b) the other object's behaviour affects 'our' agent's success; and (c) 'our' agent's behaviour affects the other object's success. Hence, the wind is not an agent, since it cannot be said to be maximising its success at anything. We would want to treat nearby drivers as other agents; we might regard more distant drivers as just stochastically behaving objects. (Note that this definition allows the other agents to be cooperative, competitive or anything in between.)

3.2 Reactive Agents through to Deliberative Agents

Reactive agents: A reactive agent is one that chooses its actions on the basis of its *current percept only*. In other words, its next action is chosen solely using what its sensors tell it the world is like now. It doesn't use any memory to keep track of earlier percepts, and it does not think ahead. Reactive agents therefore typically have very simple *Plan* phases. Despite this, reactive agents can exhibit surprisingly sophisticated behaviour.

Reactive agents with internal state: This kind of agent uses memory to keep track of parts of the percept history, but still does not think ahead. Often, the agent will use the percept history to build and maintain a mental *model* of the world. Of course, the model needs to be updated to reflect not just the current percept but also to reflect the effects of 'our' agent's actions, other agents' actions and the way the world changes independently of all these agents. A model provides a way to handle partial observability: the model can be consulted for information about parts of the world that cannot currently be sensed.

Deliberative agents: A deliberative agent thinks ahead. It must keep a mental model of the world, and it 'simulates' the effects of actions on that mental model. It will often explore alternative 'simulations'. It chooses its actions on the basis of the outcomes of these 'simulations': it can choose the actions that best achieve its goals or that bring greatest utility. It needs a memory and a reasoning capability, hence its *Plan* phase is much more complicated.

The above categorisation is not intended to deny that many intelligent agents, such as humans, are hybrids that combine reactive and deliberative capabilities.

In the rest of this module, we will trace a story from reactive agents to reactive agents with internal state to deliberative agents, adding ever more capabilities to our agent as we move through the module.

But, for a little concreteness, let's look now at one way of building a reactive agent.

4 Table-Driven Agents: A Simple Implementations of Reactive Agents

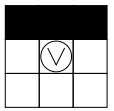
A table-driven agent does the following at each point in time:

```
percept := SENSE();
action := LOOKUP(percept, table);
EXECUTE(action);
```

4.1 Class exercise

By way of a simple example, consider a softbot who lives in a virtual room. The room is rectangular and a grid divides it into equal-sized squares. Some of the cells contain bricks, and so the softbot cannot pass through those cells. In particular, a wall of bricks marks the boundaries of the room. (We will assume the softbot always starts off facing either North, East, South or West.)

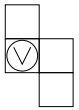
1. Suppose the softbot has eight touch sensors. These return 1 if there's an object in the corresponding cell, and 0 otherwise. For example, in this configuration,

   the sensors return 11000001 (starting from the softbot's mouth and moving clockwise).

   How many table entries would there be, i.e. how many different percepts are there?

2. In fact, only three sensors are needed in this exercise. They detect the cells highlighted here:

   How many table entries would there be?

3. The actions are `Move` and `Turn`. `Move` moves the agent one pace forward in the direction that it is facing. For `Turn`, we supply two arguments. The first argument is either `LEFT` or `RIGHT`. The second argument is an integer that indicates how many 45° rotations make up the turn. For example, `Turn(RIGHT, 2)` is a turn action that turns this agent 90 degrees to the right (e.g. if it was facing North, it is now facing East).

   Fill in the table, to define a table-driven agent that walks the walls of the room in an anticlockwise direction.

Percept	Action
000	
001	
010	
011	
100	
101	
110	
111	

4.2 Discussion

The softbot we designed above can hardly be called an autonomous agent. All the computation was done by us, the designers, in advance. This is one criticism of the table-driven approach. (There may be ways of learning or evolving the table, and we will return to these issues later in the module.)

The table-driven approach is *possible* only if the number of distinct percepts is finite.

The approach is *practicable* only if the number of distinct percepts is not just finite but also quite small. Otherwise, (a) the table would require too much space; and (b) for us, the designers, to fill in the entries would be too time-consuming (or too many experiences would be needed for the agent to learn the entries).

Exercise (Past-Exam Question)

This question is about TLUs and neural networks. Throughout, assume that the *activation function* of the TLUs, g, is defined as follows:

$$g(x) =_{\text{def}} \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

where θ is the threshold of the TLU.

1. Design a TLU which has two inputs s_1 and s_2. The inputs can take values of 0 or 1. The TLU should compute $s_1 \downarrow s_2$, where $\downarrow$ is the NOR operator, i.e. $s_1 \downarrow s_2 \equiv \neg(s_1 \vee s_2)$.

2. Your answer to part 1 is to be converted into a TLU that has a threshold of 0 but it has an extra input s_0. At what value will you fix the input s_0 and what weight will you use on s_0's input wire to continue to compute $s_1 \downarrow s_2$?

3. The kind of conversion that you carried out in part 2 is a useful precursor to training a neural net using a learning algorithm. Why is it useful?

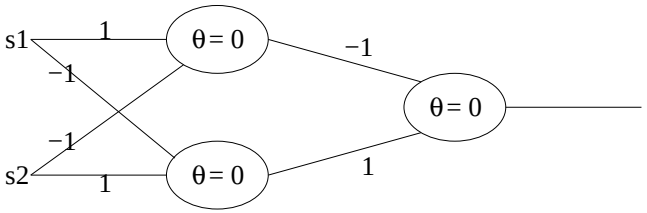
4. Below we have tabulated eight Boolean-valued functions:

s_1	s_2	$s_1 \boxtimes s_2$	$s_1 \boxdot s_2$	$s_1 \boxplus s_2$	$s_1 \boxminus s_2$	$s_1 \otimes s_2$	$s_1 \odot s_2$	$s_1 \oplus s_2$	$s_1 \ominus s_2$
0	0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1

   Suppose you intend to design eight TLUs, each having two inputs, s_1 and s_2, that can take values of only 0 or 1.

   For how many of the eight functions can such TLUs be designed? Explain your answer convincingly and in detail. (There is no need to give any actual TLUs.)

5. Here is a fully connected, layered, feedforward neural network:



   What will the output of this network be when the values of its inputs, s_1 and s_2, are both 1. Show your working.

6. Design a fully connected, layered, feedforward neural network for the following. There are four inputs, each of which can take a value of 0 or 1. When any two of the inputs are 0 and the other two inputs are 1, the output of the network is 1. Otherwise the output is 0.