INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Brief History of AI

1943–56:

- McCulloch & Pitts (1943)
  artificial neural net — proved equivalent to Turing machine;
- Shannon, Turing (1950)
  chess playing programs
- Marvin Minsky (1951)
  first neural net computer — SNARC
- Dartmouth College (1956)
  term "AI" coined by John McCarthy
  Newell & Simon presented LOGIC THEORIST program

1956-70:

- Programs written that could:

  plan, learn, play games, prove theorems, *solve problems*.

- Major centres established:
  - Minsky — MIT
  - McCarthy — Stanford
  - Newell & Simon — CMU
- Major feature of the period were *microworlds* — toy problem domains.

  Example: blocks world.

  "It'll scale, honest…"

1970s:

- 1970s period of recession for AI
  (Lighthill report in UK)
- Techniques developed on microworlds *would not* scale.
- Implications of *complexity theory* developed in late 1960s, early 1970s began to be appreciated:
  - *brute force techniques will not work.*
  - *works in principle does not mean works in practice.*

1980s:

- General purpose, brute force techniques don't work, so use *knowledge rich* solutions.
- Early 1980s saw emergence of *expert systems* as systems capable of exploiting knowledge about tightly focussed domains to solve problems normally considered the domain of experts.
- Ed Feigenbaum's *knowledge principle*.

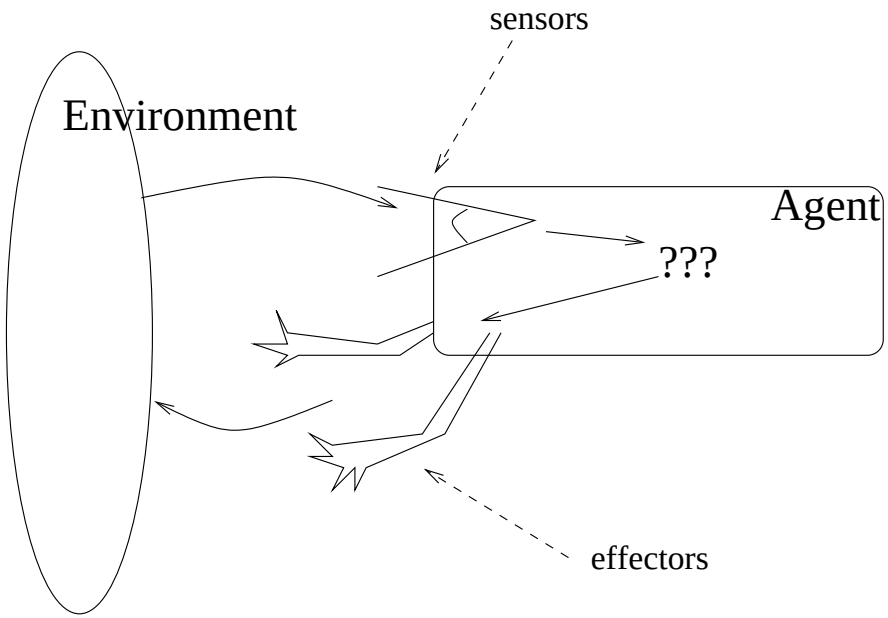
- Expert systems success stories:
  - MYCIN — blood diseases in humans;
  - DENDRAL — interpreting mass spectrometers;
  - R1/XCON — configuring DEC VAX hardware;
  - PROSPECTOR — finding promising sites for mineral deposits;
- Expert systems emphasised *knowledge representation*: rules, frames, semantic nets.
- Problems:
  - the knowledge elicitation bottleneck;
  - marrying expert system & traditional software;
  - breaking into the mainstream.

1990s:

- Most companies set up to commercialise expert systems technology went bust.
- "AI as homeopathic medicine" viewpoint common.
- 1990s: emphasis on understanding the interaction between *agents* and *environments*.
- AI as *component*, rather than as end in itself.
  - "Useful first" paradigm — Etzioni (NETBOT, US$35m)
  - "Raisin bread" model — Winston.

Intelligent Agents

- An *agent* is a system that is *situated* in an environment, and which is capable of *perceiving* its environment and *acting* in it to satisfy its design objectives.
- Pictorially:



- Human "agent":
  - *environment*: physical world;
  - *sensors*: eyes, ears, ...
  - *effectors*: hands, legs, ...
- Software agent:
  - *environment*: (e.g.) UNIX operating system;
  - *sensors*: `ls`, `ps`, ...
  - *effectors*: `rm`, `chmod`, ...
- Internet agent:
  - *environment*: the Internet;
  - *sensors*: `http` requests;
  - *effectors*: `http` commands.

What to do?

Those who do not reason
Perish in the act.
Those who do not act
perish for that reason
(W H Auden)

- The key problem we have is *knowing the right thing to do*.
- Knowing what to do can *in principle* be easy: consider all the alternatives, and choose the "best".
- But Auden's quote! In any time-constrained domain, we have to make a decision *in time for that decision to be useful*!
- A tradeoff.

- Need to know *how* and *when* to evaluate success.
- How:
  - an objective performance measure;
  - application specific;
- When:
  - in discrete *episodes*, or over long periods?
- Don't confuse *omniscience* with rationality.
- Real agents don't know enough to always make the best choice. (We often fall into this trap when making judgements about history.)
- Rationality concerned with *expected success* given *information available*.

- *Ideal rational agent*:

  For each percept sequence, an ideal rational agent will act to maximise its expected performance measure, on the basis of information provided by percept sequence plus any information built in to agent.

- Note that this does not preclude performing actions to *find things out*.

- More precisely, we can view an agent as a function:

$$f : P^* \rightarrow A$$

  from sequences of percepts P to actions A.

- For example, a quadratic agent:

Percept	Action
0	0
1	1
2	4
3	9
4	16
...	...

- This table can be viewed as a *specification* of the agent.
- We don't have to *implement* agent as table lookup:

```
int agent(int n)
{
     return n * n;
}
```

- *Autonomy* a crucial concern for agents.

  Means behaviour is based on *own* experience. Implies *learning*, or *adaptation*.

Structure of Agents

- Two components:
  - *program*: the thing which defines the mapping from percept sequences to actions;
  - *architecture*: the "shell" into which the agent program fits.

  Agent = program + architecture.
- An appropriate architecture can make design of programs much easier.

Classifying Environments

- The PAGE approach:
  - percepts;
  - actions;
  - goals;
  - environment.
- Example: Refinery controller.
  - percepts: temp, pressure readings;
  - actions: open, close valves, switch on, off heaters...;
  - goals: maximise purity, yield, safety;
  - environment: refinery.

- Example: Medical diagnosis system.
  - percepts: symptoms, findings, patient answers;
  - actions: questions, tests, treatments;
  - goals: healthy patient, minimise costs;
  - environment: patient, hospital.

- Example: Email manager.
  - percepts: email arrived, headers, content of email;
  - actions: delete email, reorder email, obtain user attention;
  - goals: present important email first; hide junk email;
  - environment: mail reader, operating system.

Accessible *vs* inaccessible

An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state.

Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible.

The more accessible an environment is, the simpler it is to build agents to operate in it.

Deterministic *vs* non-deterministic

A deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action.

The physical world can to all intents and purposes be regarded as non-deterministic.

Non-deterministic environments present greater problems for the agent designer.

Episodic *vs* non-episodic

In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios.

An example of an episodic environment would be a mail sorting system.

Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to perform based only on the current episode — it need not reason about the interactions between this and future episodes.

Static *vs* dynamic

A static environment is one that can be assumed to remain unchanged except by the performance of actions by the agent.

A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control. The physical world is a highly dynamic environment.

Discrete *vs* continuous

An environment is discrete if there are a fixed, finite number of actions and percepts in it.

Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.