INTELLIGENT AGENTS

Intelligent Agents

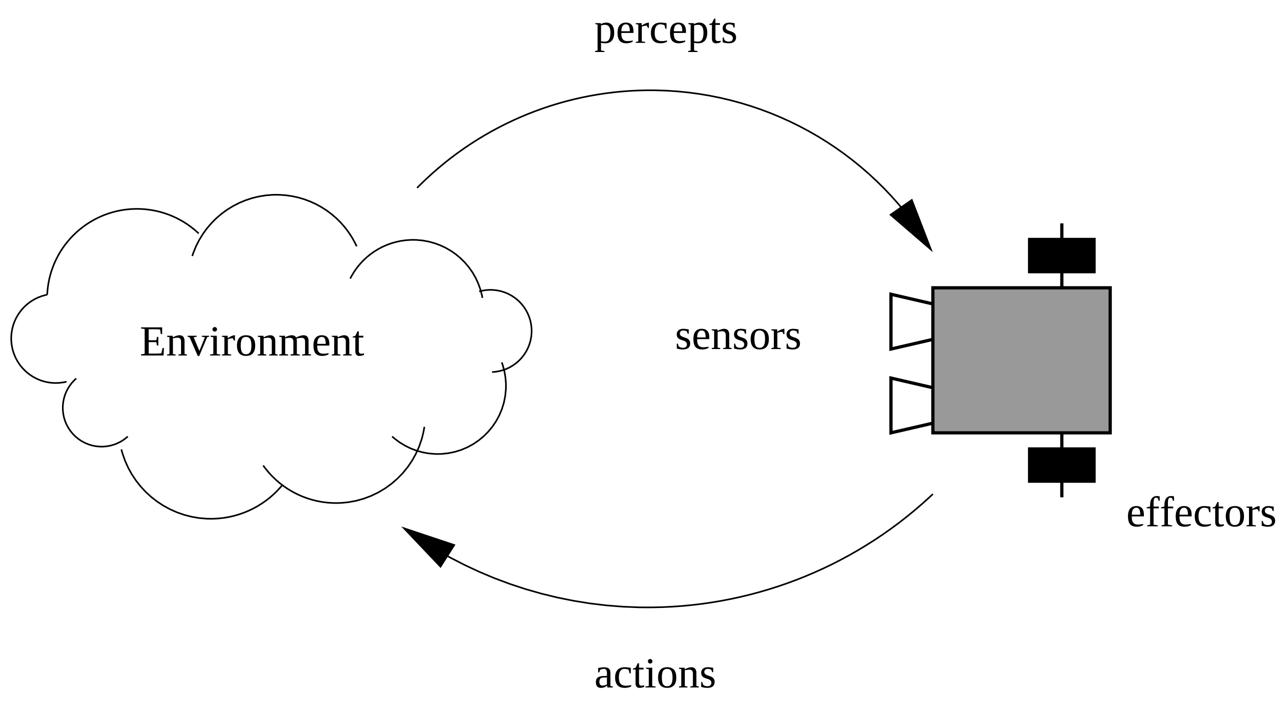
- We are going to take a very agent-based perspective on artificial intelligence.
- One of the main ways that AI is viewed at the moment.

Intelligent Agents

- An *agent* is a system that:
  - is *situated* in an environment,
  - is capable of *perceiving* its environment, and
  - is capable of *acting* in its environment

  with the goal of satisfying its design objectives.

- Pictorially:



- The task is to program the agent to convert percepts to actions.

- 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`, . . .
- Robot:
  - *environment*: physical world;
  - *sensors*: sonar, camera;
  - *effectors*: wheels.

Google
intel
Red Bull
MDV
MOHR DAVIDOW VENTURES
Powered by Diesel.
NXP

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 any time-constrained domain, we have to make a decision *in time for that decision to be useful*!
- A tradeoff.

- *Ideal rational agent*:

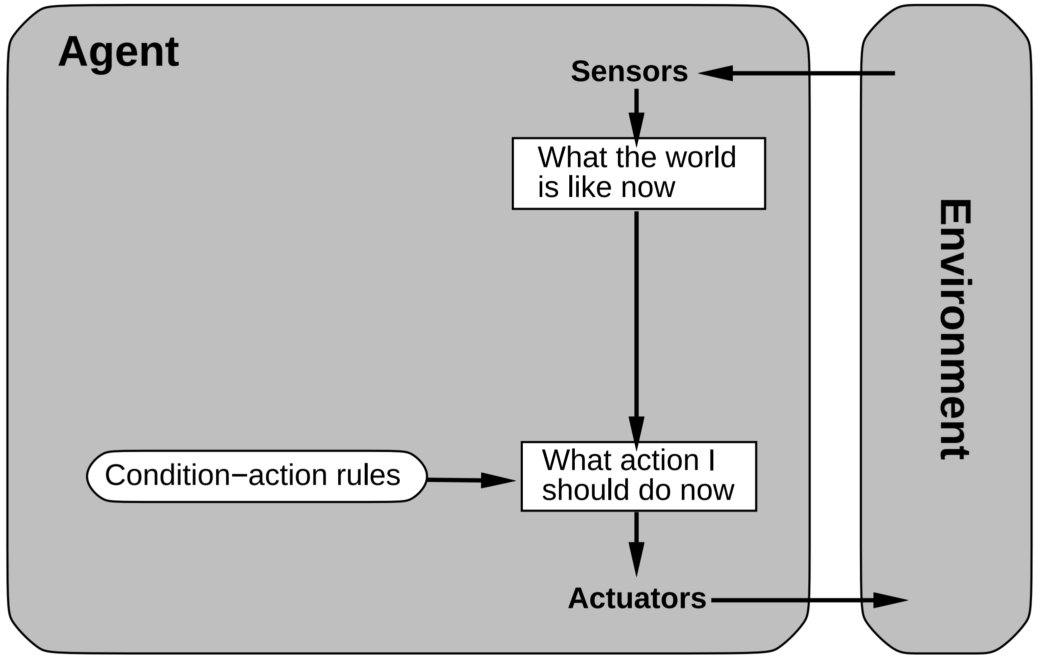
  For each percept sequence, an ideal rational agent will act to maximise its expected performance measure, on the basis of information provided by percept sequence plus any information built in to agent.
- Note that this does not preclude performing actions to *find things out*.
- More precisely, we can view an agent as a function:

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

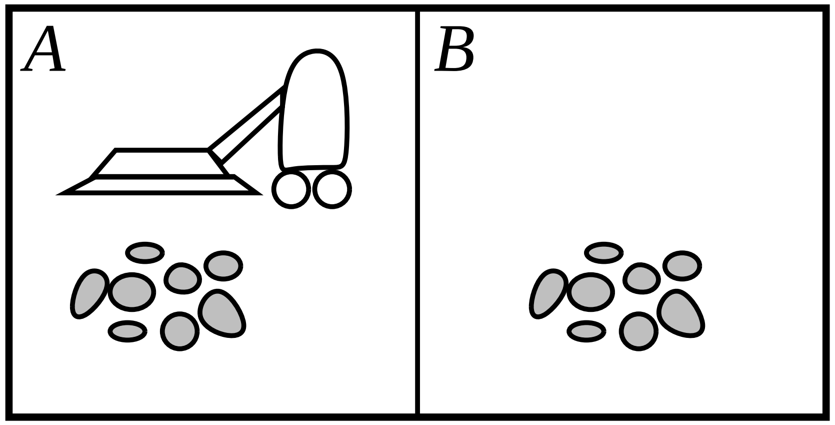
  from sequences of percepts P to actions A.

Simple reflex agent

- A simple agent maps percepts directly to actions:



- In the "vacuum world":



- A simple agent may suffice:

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

- A more general version of this program, which works for the agent architecture given above, is on the next slide.

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
    static: rules, a set of condition-action rules
    state ← INTERPRET-INPUT(percept)
    rule ← RULE-MATCH(state, rules)
    action ← rule.action
    return action
```

Fully observable versus partially observable

- A *fully observable* environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state.
- Such an environment is also called *accessible*.
- Most moderately complex environments (including, for example, the everyday physical world and the Internet) are only *partially observable*.
- Such environments are also known as *non-accessible*
- The more observable an environment is, the simpler it is to build agents to operate in it.

Deterministic versus 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.
- The textbook calls non-deterministic environments *stochastic* if we quantify the non-determinism using probability theory.
- Non-deterministic environments present greater problems for the agent designer.

Episodic versus sequential

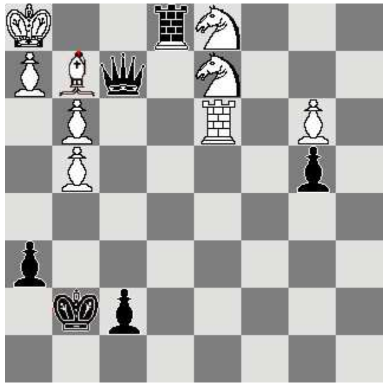
- In an *episodic* environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios.
- An example of an episodic environment would be an assembly line where an agent had to spot defective parts.
- Episodic 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.
- Environments that are not episodic are called either *non-episodic* or *sequential*. Here the current decision affects future decisions.
- Driving a car is sequential.

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.
- One reason an environment may be dynamic is the presence of other agents.

Discrete *vs* continuous

- An environment is *discrete* if there are a fixed, finite number of actions and percepts in it.
- The textbook *gives* a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.



Abstract Architectures for Agents

- Assume the environment may be in any of a finite set E of discrete, instantaneous states:

$$E = \{e, e', \ldots\}.$$

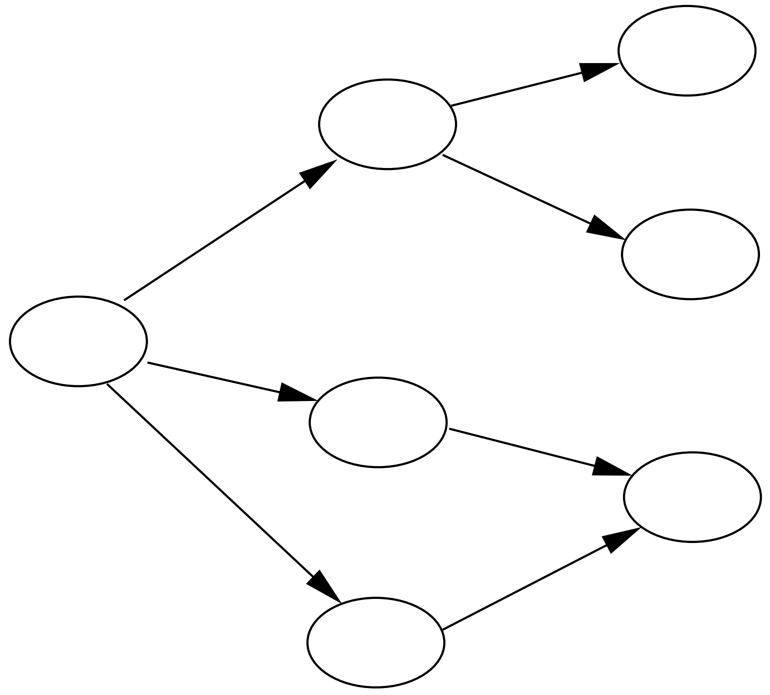
- Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the environment.

$$Ac = \{\alpha, \alpha', \ldots\}$$

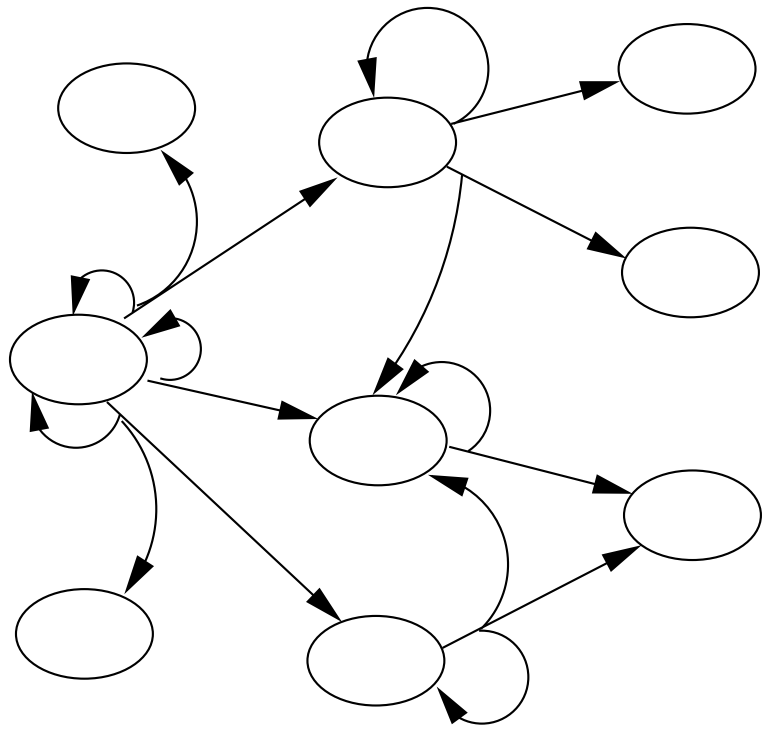
- Actions can be non-deterministic, but only one state ever results from and action.
- A *run*, r, of an agent in an environment is a sequence of interleaved environment states and actions:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u$$

- When actions are deterministic each state has only one possible successor.



- When actions are non-deterministic a run (or *trajectory*) is the same, but the set of possible runs is more complex.



- Let:
  - $\mathcal{R}$ be the set of all such possible finite sequences (over E and Ac);
  - $\mathcal{R}^{Ac}$ be the subset of these that end with an action; and
  - $\mathcal{R}^{E}$ be the subset of these that end with an environment state.
- We will use $r, r', \ldots$ to stand for the members of $\mathcal{R}$

Environments

- A *state transformer* function represents behaviour of the environment:

$$\tau : \mathcal{R}^{Ac} \rightarrow \wp(E)$$

- Note that environments are…
  - *history dependent*.
  - *non-deterministic*.
- If $\tau(r) = \{ \}$, there are no possible successor states to r, so we say the run has *ended*. ("Game over.")
- An environment *Env* is then a triple $Env = \langle E, e_0, \tau \rangle$ where E is set of environment states, $e_0 \in E$ is initial state; and τ is state transformer function.

Agents

- We can think of an agent as being a function which maps runs to actions:

$$Ag : \mathcal{R}^E \rightarrow Ac$$

- Thus an agent makes a decision about what action to perform based on the history of the system that it has witnessed to date.
- Let $\mathcal{AG}$ be the set of all agents.

Systems

- A *system* is a pair containing an agent and an environment.
- Any system will have associated with it a set of possible runs; we denote the set of runs of agent *Ag* in environment *Env* by $\mathcal{R}(Ag, Env)$.
- Assume $\mathcal{R}(Ag, Env)$ contains only runs that have ended.

- Formally, a sequence

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)$$

represents a run of an agent *Ag* in environment $Env = \langle E, e_0, \tau \rangle$ if:

1. e_0 is the initial state of *Env*
2. $\alpha_0 = Ag(e_0)$; and
3. for $u > 0$,

$$\begin{aligned} e_u &\in \tau((e_0, \alpha_0, \ldots, \alpha_{u-1})) \quad \text{and} \\ \alpha_u &= Ag((e_0, \alpha_0, \ldots, e_u)) \end{aligned}$$

- Two agents are said to be *behaviorally equivalent* with respect to *Env* iff $\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$.

Simple reflex agents

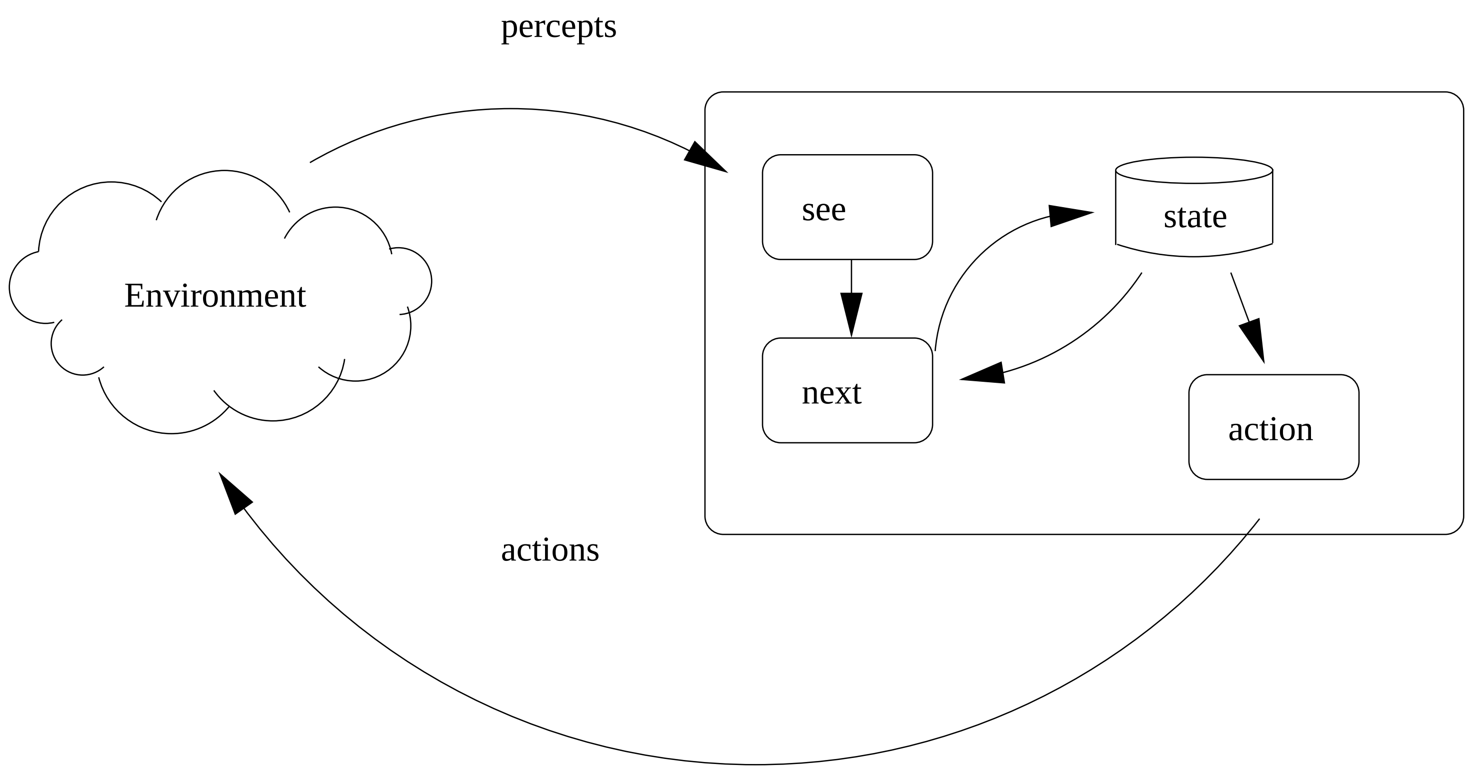
- Given this notation we can write the simple reflex agents we met above as having an action function that maps state to action:

$$action : E \rightarrow Ac$$

- A thermostat is a simple reflex agent:

$$action(e) = \begin{cases} \text{off} & \text{if } e = \text{temperature OK} \\ \text{on} & \text{otherwise.} \end{cases}$$

Agents with State



- The *see* function is the agent's ability to observe its environment, whereas the *action* function represents the agent's decision making process.
- *Output* of the *see* function is a *percept*:

  $$see : E \to Per$$

  which maps environment states to percepts.
- The agent has some internal data structure, which is typically used to record information about the environment state and history.
- Let I be the set of all internal states of the agent.

- The action-selection function *action* is now defined as a mapping

$$action : I \rightarrow Ac$$

  from internal states to actions.
- An additional function *next* is introduced, which maps an internal state and percept to an internal state:

$$next : I \mathbf{x} Per \rightarrow I$$

- This says how the agent updates its view of the world when it gets a new percept.

1. Agent starts in some initial internal state i_0.
2. Observes its environment state e, and generates a percept $see(e)$.
3. Internal state of the agent is then updated via *next* function, becoming $next(i_0, see(e))$.
4. The action selected by the agent is $action(next(i_0, see(e)))$.

   This action is then performed.
5. Goto (2).

Tasks for Agents

- We build agents in order to carry out *tasks* for us.
- The task must be *specified* by us…
- But we want to tell agents what to do *without* telling them how to do it.

Utility Functions

- One possibility: associate *utilities* with individual states — the task of the agent is then to bring about states that maximise utility.
- A task specification is a function

  $$u : E \to \mathbb{R}$$

  which associated a real number with every environment state.

- But what is the value of a *run*…
  - minimum utility of state on run?
  - maximum utility of state on run?
  - sum of utilities of states on run?
  - average?
- Disadvantage: difficult to specify a *long term* view when assigning utilities to individual states.
- One possibility: a *discount* for states later on. This is what we do in *reinforcement learning*.

Utilities over Runs

- Another possibility: assigns a utility not to individual states, but to runs themselves:

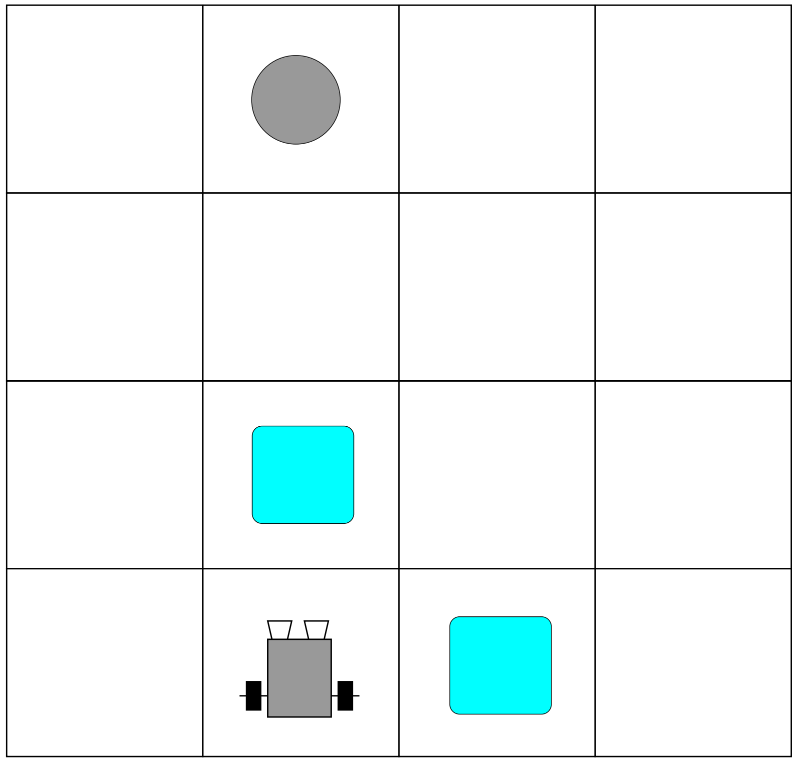
$$u : \mathcal{R} \to \mathbb{R}$$

- Such an approach takes an inherently *long term* view.
- Other variations: incorporate probabilities of different states emerging.

Utility in the Tileworld

- Simulated two dimensional grid environment on which there are agents, tiles, obstacles, and holes.
- An agent can move in four directions, up, down, left, or right, and if it is located next to a tile, it can push it.
- Holes have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, with the aim being to fill as many holes as possible.

- The agent starts to push a tile towards the hole.



- TILEWORLD changes with the random appearance and disappearance of holes.
- Utility function defined as follows:

$$u(r) \mathrel{\hat{=}} \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

- TILEWORLD captures the need for *reactivity* and for the advantages of exploiting opportunities.

Expected Utility

- Write $P(r \mid Ag, Env)$ to denote probability that run r occurs when agent Ag is placed in environment Env.
- In a non-deterministic environment, for example, this can be computed from the probability of each step.
- For a run $r = (e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)$:

$$\Pr(r \mid Ag, Env) = \Pr(e_1, \mid e_0, \alpha_0) \Pr(e_2 \mid e_1, \alpha_1) \ldots$$

  and clearly:

$$\sum_{r \in \mathcal{R}(Ag, Env)} P(r \mid Ag, Env) = 1.$$

- The *expected utility* of agent Ag in environment Env (given P, u), is then:

$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r \mid Ag, Env).$$

An Example

Consider the environment $Env_1 = \langle E, e_0, \tau \rangle$ defined as follows:

$$E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$$

$$\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2\}$$

$$\tau(e_0 \xrightarrow{\alpha_1}) = \{e_3, e_4, e_5\}$$

There are two agents possible with respect to this environment:

$$Ag_1(e_0) = \alpha_0$$

$$Ag_2(e_0) = \alpha_1$$

The probabilities of the various runs are as follows:

$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_1, Env_1) = 0.4$$

$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_1, Env_1) = 0.6$$

$$P(e_0 \xrightarrow{\alpha_1} e_3 \mid Ag_2, Env_1) = 0.1$$

$$P(e_0 \xrightarrow{\alpha_1} e_4 \mid Ag_2, Env_1) = 0.2$$

$$P(e_0 \xrightarrow{\alpha_1} e_5 \mid Ag_2, Env_1) = 0.7$$

Assume the utility function u_1 is defined as follows:

$$u_1(e_0 \xrightarrow{\alpha_0} e_1) = 8$$

$$u_1(e_0 \xrightarrow{\alpha_0} e_2) = 11$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_3) = 70$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_4) = 9$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_5) = 10$$

What are the expected utilities of the agents for this utility function?

Optimal Agents

- The optimal agent Ag_{opt} in an environment Env is the one that *maximizes expected utility*:

$$Ag_{opt} = \arg \max_{Ag \in \mathcal{AG}} EU(Ag, Env) \tag{1}$$

- Of course, the fact that an agent is optimal does not mean that it *will* be best; only that *on average*, we can expect it to do best.
- To see the difference between these two ideas, remember the Patriots 4th down against the Colts the other season.
- Also note that though this characterises an optimal agent, it does not tell us how to build an optimal agent.

Bounded Optimal Agents

- Some agents cannot be implemented on some computers
- A function $Ag : \mathcal{R}^E \rightarrow Ac$ may need more than available memory to implement.
- Write $\mathcal{AG}_m$ to denote the agents that can be implemented on machine m:

$$\mathcal{AG}_m = \{Ag \mid Ag \in \mathcal{AG} \text{ and } Ag \text{ can be implemented on } m\}.$$

- The *bounded optimal* agent, Ag_{bopt}, with respect to m is then…

$$Ag_{bopt} = \arg \max_{Ag \in \mathcal{AG}_m} EU(Ag, Env) \tag{2}$$

Predicate Task Specifications

- A special case of assigning utilities to histories is to assign 0 (false) or 1 (true) to a run.
- If a run is assigned 1, then the agent *succeeds* on that run, otherwise it *fails*.
- Call these *predicate task specifications*.
- Denote predicate task specification by Ψ:

$$\Psi : \mathcal{R} \to \{0, 1\}$$

Task Environments

- A *task environment* is a pair $\langle Env, \Psi \rangle$, where Env is an environment, and

$$\Psi : \mathcal{R} \to \{0, 1\}$$

  is a predicate over runs.
- Let $\mathcal{TE}$ be the set of all task environments.
- A task environment specifies:
  - the properties of the system the agent will inhabit;
  - the criteria by which an agent will be judged to have either failed or succeeded.

- Write $\mathcal{R}_\Psi(Ag, Env)$ to denote set of all runs of the agent Ag in environment Env that satisfy Ψ:

$$\mathcal{R}_\Psi(Ag, Env) = \{r \mid r \in \mathcal{R}(Ag, Env) \text{ and } \Psi(r) = 1\}.$$

- We then say that an agent Ag succeeds in task environment $\langle Env, \Psi\rangle$ if

$$\mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env)$$

- In other words, an agent succeeds if every run satisfies the specification of the agent.

- We might write this as:

$$\forall r \in \mathcal{R}(Ag, Env), \text{we have} \Psi(r) = 1$$

- This is a bit pessimistic.
- If the agent fails on a single run, we say it has failed overall.
- A more optimistic idea of success is:

$$\exists r \in \mathcal{R}(Ag, Env), \text{we have} \Psi(r) = 1$$

  which counts an agent as successful as soon as it completes a single successful run.

The Probability of Success

- If the environment is non-deterministic, the τ returns a *set* of possible states.
- We can define a probability distribution across the set of states.
- Let $P(r \mid Ag, Env)$ denote probability that run r occurs if agent Ag is placed in environment Env.
- Then the probability $P(\Psi \mid Ag, Env)$ that Ψ is satisfied by Ag in Env would then simply be:

$$P(\Psi \mid Ag, Env) = \sum_{r \in \mathcal{R}_{\Psi}(Ag, Env)} P(r \mid Ag, Env)$$

Summary

- This lecture has looked at:
  - The notion of intelligent agents
  - A classification of agent environments.
  - A simple model of agents and environments.
- Broadly speaking, the rest course will cover the some of the more advanced techniques of AI, with special reference to agents.
- The techniques we'll look at will start with those applicable to simple environments and move towards those suitable for more complex environments.