Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili

LECTURE 2: Intelligent Agents – Properties & Abstract Architecture

Introduction to Multi-Agent Systems (MESIIA, MIA)

URV

What is an Agent?

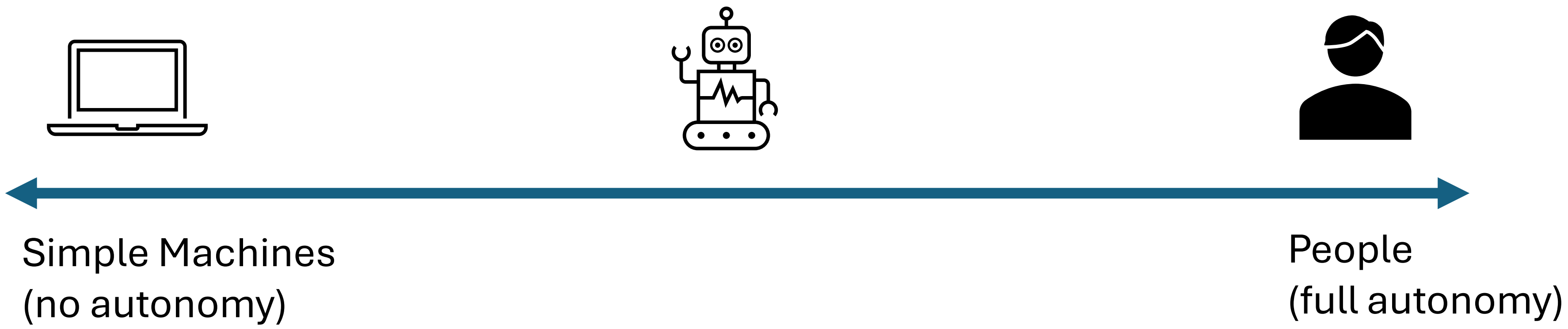
- There is no commonly accepted definition of the term *Agent*.
- The central point about agents is they are *autonomous*.

“An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous actions* in this environment in order to meet its delegated objectives.”

- It is about decisions
  - An agent must:
    - Choose *what* action to perform.
    - Decide *when* to perform an action.

Autonomy

- Autonomy is a *spectrum*



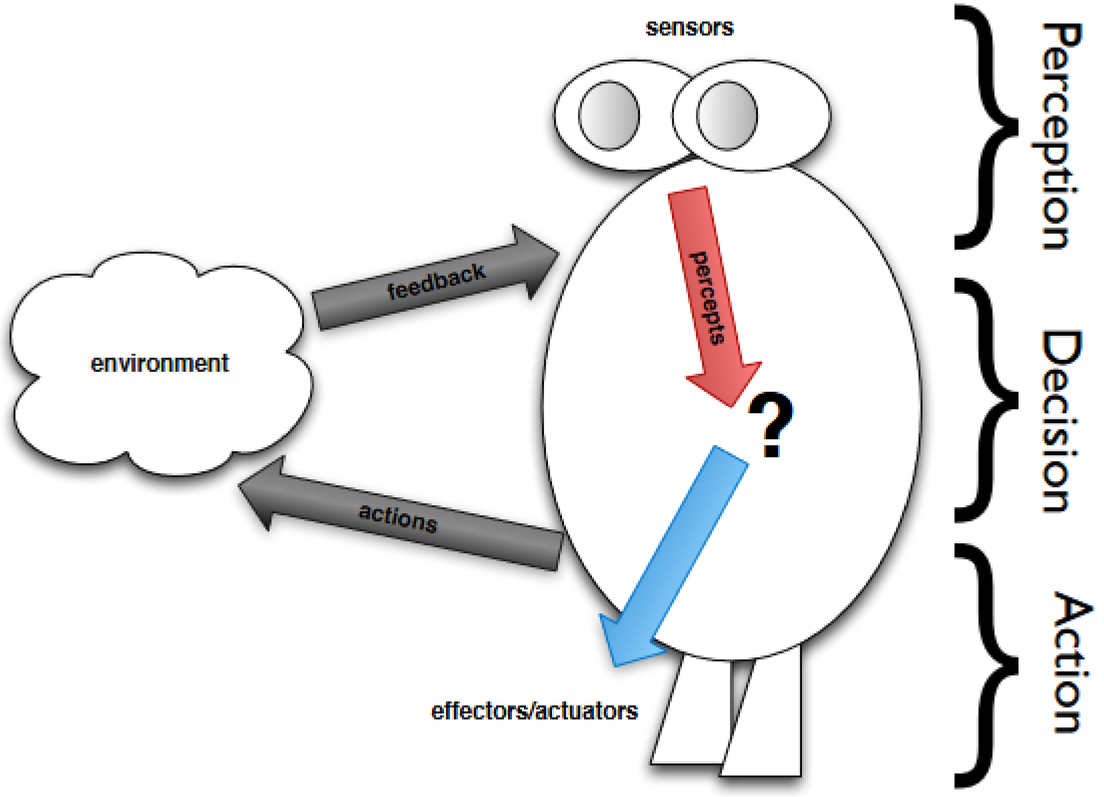
- Autonomy is adjustable
    - Human will make a decision with a remarkable higher benefit
    - High degree of uncertainty about the environment
    - The decision might cause harm, or
    - The agent lacks the capabilities to decide itself

Simple Agents

- Control Systems
  - Example: *Thermostat* (physical environment)
  - Delegated goal: maintain room temperature
  - Actions: switch on/off
  - Can be adjusted?

- Software Demons
  - Example: *UNIX biff program* (software/virtual environment)
  - Delegated goal: monitor for incoming email and flag it
  - Actions: display GUI icons

Agent and Environment



Properties of Environments

- Why do we need the properties of the environment?
  - Affect the agents, and how we build them
- Common to categorise environments along some different dimensions.
  - Fully observable vs partially observable
  - Deterministic vs non-deterministic
  - Static vs dynamic
  - Discrete vs continuous
  - Episodic vs non-episodic

Properties of Environments

- Fully observable vs partially observable
  - An accessible or fully observable 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, or partially observable.
  - The more accessible an environment is, the simpler it is to build agents to operate in it.

Properties of Environments

- Fully observable vs partially observable
  - Example – 1: Consider a multi-agent system for playing a cooperative card game, such as bridge or spades. The agents are the players, and their goal is to maximize their team's score. The agents *can* ***see*** *their own cards*, but not the cards of the other players. The agents can ***communicate*** with their partner, but not with their opponents. Is this environment accessible or inaccessible for the agents? Explain your answer.

  - Example – 2: Consider a multi-agent system for playing a tic-tac-toe game. The agents are the players, and their goal is to win the game or avoid losing. The agents can ***see*** the entire board, and they ***know*** the rules of the game. The agents take turns to place their symbol (X or O) on an empty cell of the board. Is this environment accessible or inaccessible for the agents? Explain your answer.

Properties of Environments

- 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.
  - Fully observable, deterministic environment => No worry about uncertainty.
  - We'll follow Russell and Norvig in calling environments stochastic if we quantify the non-determinism using probability theory.
  - Non-deterministic environments present greater problems for the agent designer.

Properties of Environments

- 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.

Properties of Environments

- Discrete vs continuous
  - An environment is discrete if there are a fixed, finite number of actions and percepts in it.
  - Otherwise, it is continuous
  - Examples:
    - Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.
  - We often assume that the environment is discrete for simplicity.

Properties of Environments

- Episodic vs non-episodic
  - An ***episodic*** environment is one in which the current decision of the agent affects only the current episode, and not the future episodes.
    - An episode is a single experience that the agent has with the environment
    - No need to consider the long-term consequences
    - Act based on the current percept
  - A ***non-episodic*** (sequential) environment is one in which the current decision of the agent affects not only the current episode, but also the future episodes.
    - Agent must consider the long-term consequences of its actions
    - Act based on the history of percepts and actions

Intelligent Agents

- What is an intelligent agent?
- "What is Intelligence?" itself.
- Need to list the capabilities (properties) that we might expect an intelligent agent to have.
- We typically think of as intelligent agent as exhibiting 3 types of behaviour:
  - Reactive (environment aware)
  - Pro-active (goal-driven)
  - Social Ability

Reactivity

- If a program's environment is guaranteed to be fixed, the program need never worry about its own success or failure
- Program just executes blindly.
  - Example of fixed environment: compiler
- The real world is not like that: most environments are dynamic and information is incomplete
- Software is hard to build for dynamic domains: program must take into account possibility of failure
- A reactive system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful)

Proactiveness

- Reacting to an environment is easy
  - e.g., stimulus → response rules
- But we generally want agents to *do things for us*.
  - Hence goal directed behaviour
- *Pro-activeness* = generating and attempting to achieve goals; not driven solely by events; taking the initiative.

Social Ability

- The real world is a *multi-agent* environment: we cannot go around attempting to achieve goals without taking others into account.
  - Some goals can only be achieved by interacting with others.
- *Social ability* in agents is the ability to interact with other agents (and possibly humans) via *cooperation*, *coordination* and *negotiation.*
  - At the very least, it means the ability to communicate …

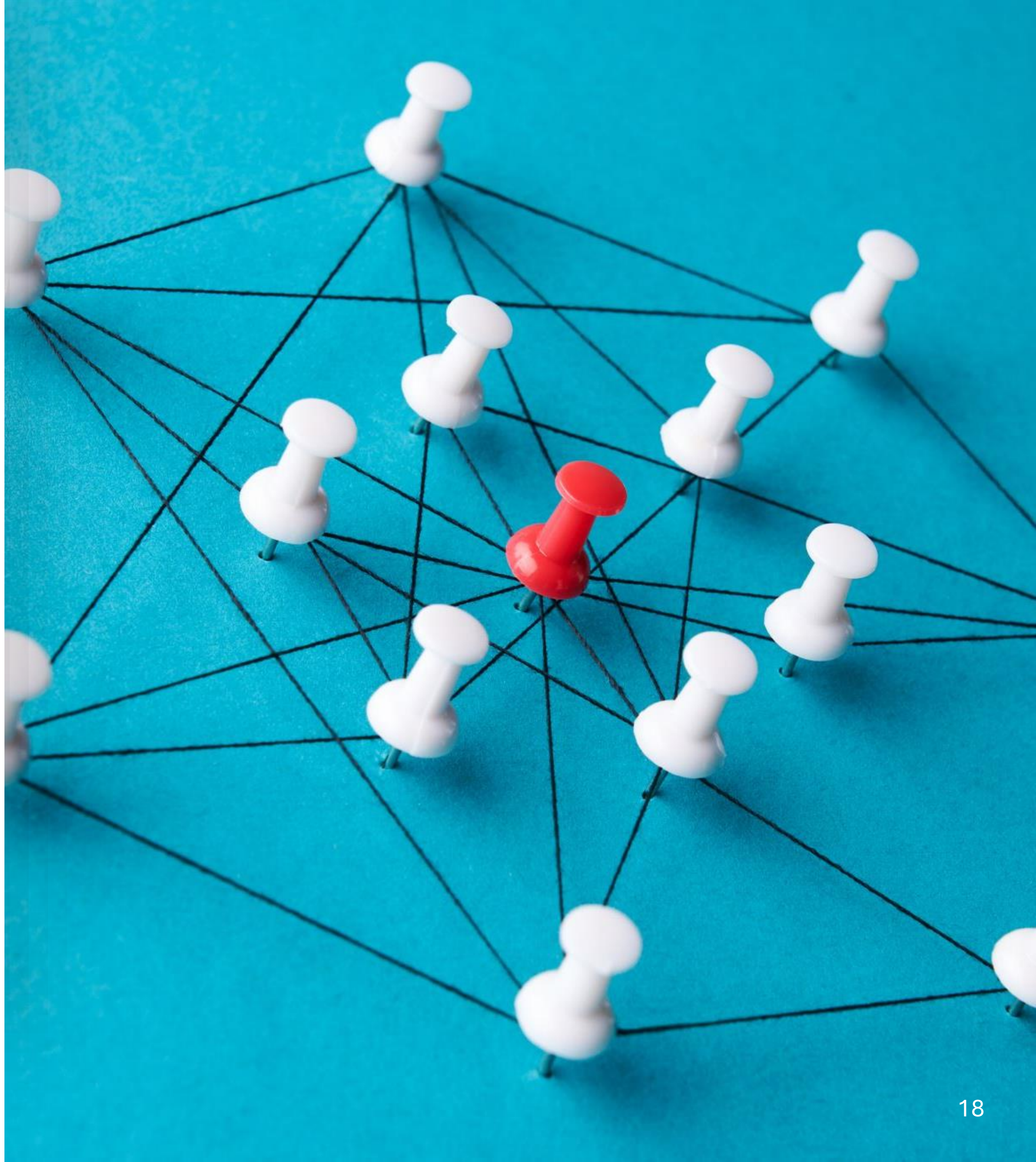
Social Ability: Cooperation

- Cooperation is working together as a team to achieve a shared goal.
- Often prompted either by the fact that no one agent can achieve the goal alone, or that cooperation will obtain a better result (e.g., get result faster)

Social Ability: Coordination

- Coordination is managing the interdependencies between activities.

- For example, if there is a non-sharable resource that you want to use and I want to use, then we need to coordinate.



Social Ability: Negotiation

- Negotiation is the ability to reach agreements on matters of common interest.

- For example:
  - You have one TV in your house; you want to watch a movie, your housemate wants to watch football.
  - A possible deal: watch football tonight, and a movie tomorrow.

- Typically involves offer and counter-offer, with compromises made by participants.

Some Other Properties...

- Mobility
  - The ability of an agent to move. For software agents this movement is around an electronic network.
- Rationality
  - Whether an agent will act in order to achieve its goals, and will not deliberately act so as to prevent its goals being achieved.
- Learning
  - Whether agents improve performance over time.
- Veracity
  - Whether an agent will knowingly communicate false information.

Abstract Architectures for Agents

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

$$E = \{e, e', \dots\}$$

- Agents are assumed to have a collection of possible actions, Ac, available to them, which transform the state of the environment:

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

- A *run*,r, of an agent in an environment is a sequence of environment states and actions:

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

Abstract Architectures for Agents - Runs

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

Abstract Architectures for Agents – Env.

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

$$\tau : R^{Ac} \rightarrow 2^{E}$$

- Assumptions on the environments:
  - history dependent: the next state not only dependent on the action of the agent, but an earlier action may be significant
  - non-deterministic: There is some uncertainty about the result
- If $\tau(r) = \phi$, then there are no possible successor states to r. In this case, we say that the system has *ended* its run

Abstract Architectures for Agents – Agents

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

$$Ag: R^E \to Ac$$

- Thus, an agent makes a decision about what action to perform
  - based on the history of the system that it has witnessed to date
- Environments assumed to be non-deterministic, what about agents?
- We can now define AG to be the set of all (possible) agents.

Abstract Architectures for Agents – System

- A system is a pair containing an agent and an environment, $\langle Ag, Env \rangle$.
- 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:

$$\boxed{R(Ag, Env)}$$

  - Assume that this only contains *terminated* runs.

Abstract Architectures for Agents – Systems

Formally, a sequence

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

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

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

   $e_u \in \tau((e_0, \alpha_0, \dots, \alpha_{u-1}))$, and

   $\alpha_u = Ag((e_0, \alpha_0, \dots, e_u))$

Why the notation?

- It allows us to get a precise handle on some ideas about agents.
  - For example, we can tell when two agents are the same
- Of course, there are different meanings for “same”. Here is one specific one.

Two agents, Ag_1 and Ag_2 are said to be behaviorally equivalent with respect to Env iff $R(Ag_1, Env) = R(Ag_2, Env)$

- We won’t be able to tell two such agents apart by watching what they do.

Purely Reactive Agents

- These agents decide what to do without reference to their history
  - they base their decision making entirely on the present, with no reference at all to the past
- Then, we can formally define such an agent as:

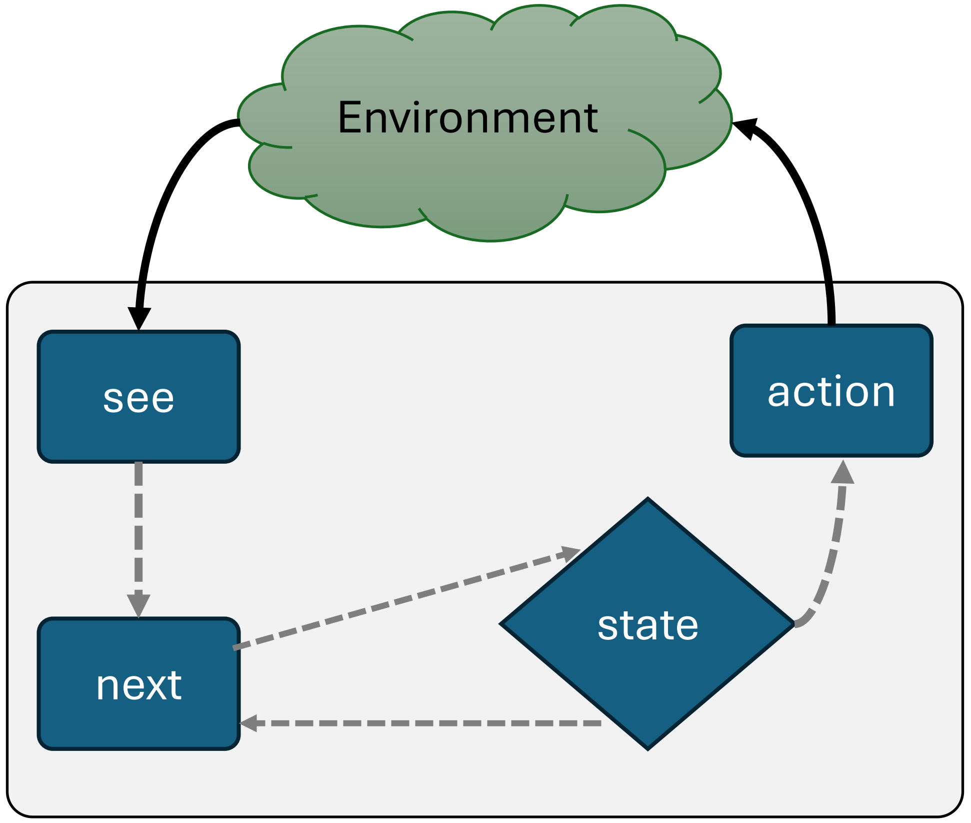
$$\boxed{Ag : E \rightarrow Ac}$$

- Example: A *thermostat* is a purely reactive agent

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

Agents with state

- Such kind of agents have some internal data structure, known as ***state***.



Perception

- 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:

$$\boxed{see: E \rightarrow Per}$$

Actions and Next State Functions

- An additional function next is introduced, which maps an internal state and percept to an internal state:

$$\boxed{next: I \times Per \to I}$$

- This says how the agent updates its view of the world *when it gets a new percept*
- The action-selection function action is now defined as a mapping from internal states to actions:

$$\boxed{action: I \to Ac}$$

Agent Control Loop

1. Agent starts at 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 $i_0 = next\left(i_0, see(e)\right)$
4. The action selected by the agent is $action(i_0)$. 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.
  - How can we make this happen???

Utility functions

- One idea:
  - associated rewards with states that we want agents to bring about.
  - We associate utilities with individual states (t he task of the agent is then to bring about states that maximise utility)

- A task specification is then a function which associates a real number with every environment state:

$$\boxed{u : E \rightarrow \mathbb{R}}$$

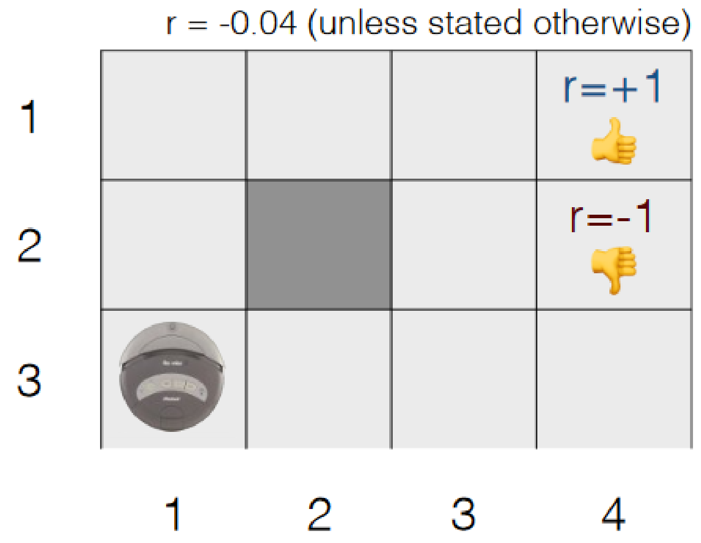
Local Utility Functions

- 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

Example of local utility function

- Goal is to select actions to maximise future rewards
  - Each action results in moving to a state with some assigned reward
  - Allocation of that reward may be immediate or delayed (e.g. until the end of the run)
  - It may be better to sacrifice immediate reward to gain more long-term reward
  - What actions maximise the reward?

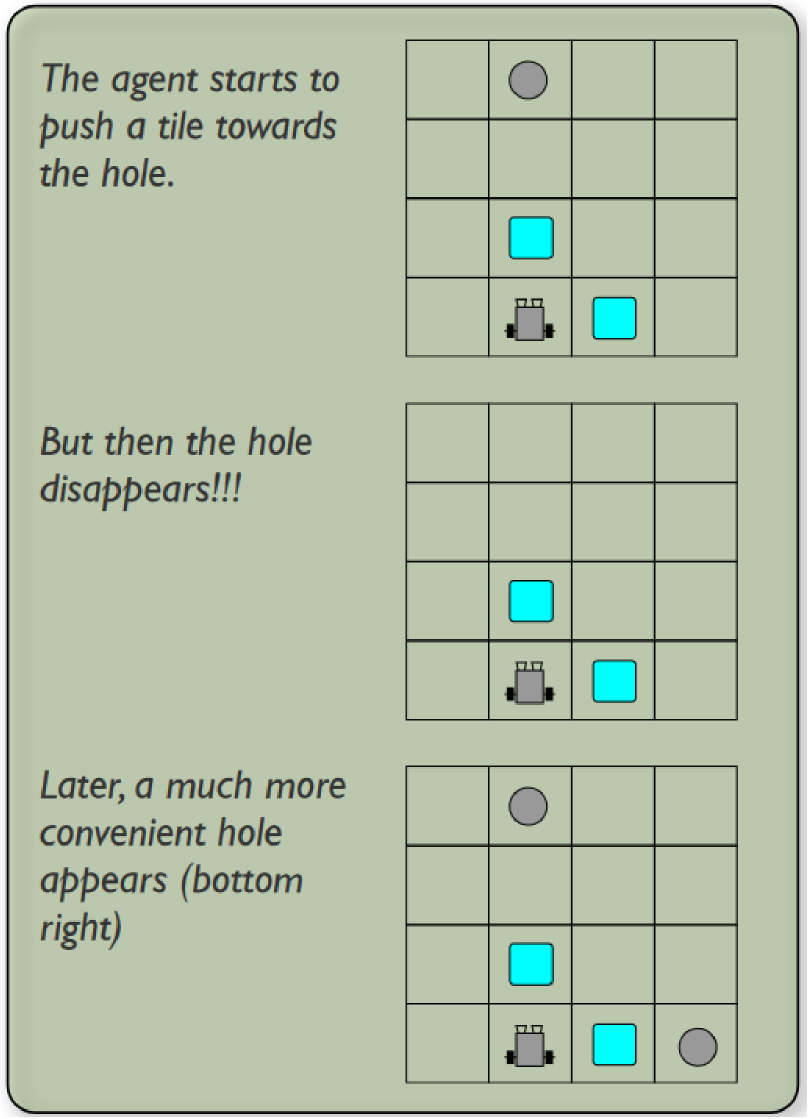


Utilities over Runs

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

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

- Such an approach takes an inherently *long-term view*.
- To see where utilities might come from, let's look at an example


The agent starts to push a tile towards the hole.
But then the hole disappears!!!
Later, a much more convenient hole appears (bottom right)

Example: 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
  - 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.
- TILEWORLD changes with the random appearance and disappearance of holes.

Example: Utilities in the Tileworld

- Utilities are associated over runs, so that more holes filled is a higher utility.
- Utility function defined as follows:
- Thus:
  - if agent fills all holes, utility = 1
  - if agent fills no holes, utility = 0

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

Expected Utility

- Let $P(r|Ag, Env)$ be the probability that run r occurs when agent Ag is placed in environment Env.

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

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

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

Optimal Agents

- The optimal agent $Ag_{opt} \in AG$ in an environment Env is the one that maximizes expected utility, i.e., EU:

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

Example 1

- Consider the environment $Env = \langle E, e_0, \tau \rangle$ defined as follows:
  - $E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$
  - $\tau((e_{0,} \alpha_0)) = \{e_1, e_2\}, \tau((e_{0,} \alpha_1)) = \{e_3, e_4, e_5\}$
- There are two agents possible with respect to this environment
  - $Ag_1(e_0) = \alpha_0$, $Ag_2(e_0) = \alpha_1$
- The probabilities and utility function of the various runs are as follows:
  - $P\big((e_0, \alpha_0, e_1) \mid Ag_1, Env\big) = 0.4,\ u\big((e_0, \alpha_0, e_1)\big) = 8$
  - $P\big((e_0, \alpha_0, e_2) \mid Ag_1, Env\big) = 0.6,\ u\big((e_0, \alpha_0, e_2)\big) = 11$
  - $P\big((e_0, \alpha_1, e_3) \mid Ag_2, Env\big) = 0.1,\ u\big((e_0, \alpha_1, e_3)\big) = 70$
  - $P\big((e_0, \alpha_1, e_4) \mid Ag_2, Env\big) = 0.2,\ u\big((e_0, \alpha_1, e_4)\big) = 9$
  - $P\big((e_0, \alpha_1, e_5) \mid Ag_2, Env\big) = 0.7,\ u\big((e_0, \alpha_1, e_5)\big) = 10$

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

Example 2

- Consider the environment $Env = \langle E, e_0, \tau \rangle$ defined as follows:
  - $E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$
  - $\tau((e_{0,} \alpha_0)) = \{e_1, e_2\}, \tau((e_{1,} \alpha_1)) = \{e_3\} , \tau((e_{2,} \alpha_2)) = \{e_4, e_5\}$
- There are two agents possible with respect to this environment
  - $Ag_1(e_0) = \alpha_0 , Ag_1(e_1) = \alpha_1$
  - $Ag_2(e_0) = \alpha_0 , Ag_2(e_2) = \alpha_2$
- The probabilities and utility function of the various runs are as follows:
  - $P\big((e_0, \alpha_0, e_1) \mid Ag_1, Env\big) = 0.5,$ $u\big((e_0, \alpha_0, e_1)\big) = 4$
  - $P\big((e_0, \alpha_0, e_2) \mid Ag_1, Env\big) = 0.5,$ $u\big((e_0, \alpha_0, e_2)\big) = 3$
  - $P\big((e_1, \alpha_1, e_3) \mid Ag_1, Env\big) = 1,$ $u\big((e_1, \alpha_1, e_3)\big) = 7$
  - $P\big((e_0, \alpha_0, e_1) \mid Ag_2, Env\big) = 0.1$ $u\big((e_2, \alpha_2, e_4)\big) = 3$
  - $P\big((e_0, \alpha_0, e_2) \mid Ag_2, Env\big) = 0.9$ $u\big((e_2, \alpha_2, e_5)\big) = 2$
  - $P\big((e_2, \alpha_2, e_4) \mid Ag_2, Env\big) = 0.4$
  - $P\big((e_2, \alpha_2, e_5) \mid Ag_2, Env\big) = 0.6$
- What are the expected utilities of the agents for this utility function?

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

$$\Psi : R \rightarrow \{0, 1\}$$

Task Environments

- A task environment is a pair $\langle Env, \Psi \rangle$ where Env is an environment and, Ψ is the task specification.

- Let the set of all task environments be defined by: $\mathrm{T}\mathcal{E}$

- 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.

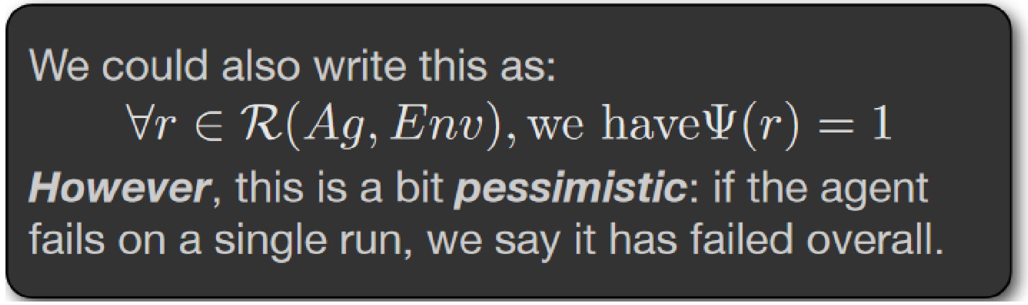
Task Environments

- To denote set of all runs of the agent Ag in environment Env that satisfy Ψ, we write:

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

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

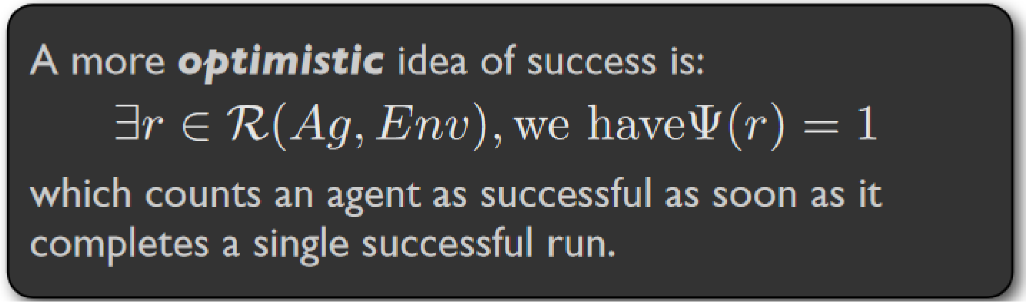
$$R_{\Psi}(Ag, Env) = R(Ag, Env)$$



We could also write this as:

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

However, 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 $P(r|Ag, Env)$ denote probability that run r occurs if agent Ag is placed in environment Env.
  - Then the probability $P(\Psi|Ag, Env)$ that Ψ is satisfied by Ag in Env would then simply be:

$$P(\Psi|Ag, Env) = \sum_{r \in R_{\Psi}(Ag,Env)} P(r|Ag, Env)$$

Achievement and Maintenance Tasks

- The idea of a predicate task specification is admittedly abstract.
- It generalises two common types of tasks, *achievement* tasks and *maintenance* tasks

Achievement and Maintenance Tasks

- An ***achievement task*** is specified by a set G of "good" or "goal" states: $G \subseteq E$.
  - The agent succeeds if it is guaranteed to bring about at least one of these states (we don't care which, as all are considered good).
  - The agent *succeeds* if in an achievement task it can *force the environment* into one of the goal states $g \in G$

- An ***maintenance task*** is specified by a set B of "bad" or "undesirable" states: $B \subseteq E$.
  - The agent succeeds if it manages to avoid all states in B - if it never performs actions which result in any state b $\in B$ occurring

Readings for this week

- M.Wooldridge: An introduction to MultiAgent Systems – Ch. 2 Intelligent Agents
- Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents”, Stan Franklin and Art Graesser. ECAI ‘96 Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages. pp 21-35