[DT0834]
Fundamentals and Applications of Artificial Intelligence
(*Fondamenti e Applicazioni dell'Intelligenza Artificiale*)
[F3I] B.Sc. Computer Science (L31 - *Laurea in Informatica*)

[2024/25] [Fall semester] [Teams: 9c7i3vd]

2.1 – Intelligent agents

Prof. Francesco Gullo

University of L'Aquila, DISIM Department

francesco.gullo@univaq.it https://fgullo.github.io/

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

2.1 Intelligent agents

~~1. Introduction to AI (6h)~~

- ~~main notions, history, examples, applications~~

2. Agent-centric AI and symbolic AI (20h)

- intelligent agents, problem solving, solving problems by searching, constraint satisfaction problems, adversarial/multi-agent search, knowledge representation, automated reasoning, logic programming, examples and applications (game playing, solving NP-hard problems, robotics, autonomous cars, natural language processing, automated theorem proving, planning, expert systems)

3. Machine learning and connectionist AI (14h)

- supervised learning, regression and classification, neural networks and deep learning, examples and applications (generative AI, large language models, computer vision), other supervised-learning techniques (hints), other learning paradigms: unsupervised learning, semi-supervised learning, reinforcement learning (hints)

4. Neuro-symbolic AI (4h)

5. Philosophy, ethics and safety of AI (4h)

2.1 Intelligent agents - Outline

- Intelligent agents and agent-based AI tasks
- Rational agents
- Agent's Performance, Environment, Actuators, Sensors (PEAS)
- Agent function and agent program
- Main types of agent
- Example: paper buying agent

2.1 Intelligent agents
Materials for this lesson

Mostly:

- [BOOK] Russell, Norvig, 2021 - Artificial Intelligence: A Modern Approach, 4th ed. (https://aima.cs.berkeley.edu/global-index.html), Chapter 2

Further materials:

- [BOOK] Poole, Mackworth, 2023 - Artificial Intelligence: Foundations of Computational Agents, 3rd ed. (https://artint.info/3e//html/ArtInt3e.html), Chapters 1-2
- [COURSE] Introduction to Artificial Intelligence @UCBerkeley (https://inst.eecs.berkeley.edu/~cs188/su24/), Lectures 1-2
- [BOOK] Aggarwal, 2021 - Artificial Intelligence: A Textbook (http://www.charuaggarwal.net), Section 1.4

2.1 Intelligent agents
Materials for this lesson

Code:

- Companion Python code repository of Russell and Norvig's book ("Artificial Intelligence: A Modern Approach")
  - https://github.com/aimacode/aima-python
- Companion Python code repository of Poole and Mackworth's book ("Artificial Intelligence: Foundations of Computational Agents")
  - Web: https://artint.info/AIPython/
  - Code document: https://artint.info/AIPython/aipython/aipython.pdf
  - Code base: https://artint.info/AIPython/aipython.zip

Intelligent agents and agent-based AI tasks

Intelligent agents

Earlier AI approaches to action tended to focus on single, isolated software systems that acted in a relatively inflexible way, automatically following pre-set rules.

Modern technologies and software applications have created a need for artificial entities that are more autonomous, flexible, and adaptive, and that operate as social entities by interacting with each other and with the humans or the environment.

- Such artificial entities are termed “intelligent agents” or simply “agents”

Intelligent agents

The concept of agent is used often in AI, when a sequence of decisions ("actions") need to be made in order to achieve a particular goal.

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators:

- A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators.
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
- A software agent receives file contents, network packets, and human input (keyboard/mouse/touchscreen/voice) as sensory inputs and acts on the environment by writing files, sending network packets, and displaying information or generating sounds.

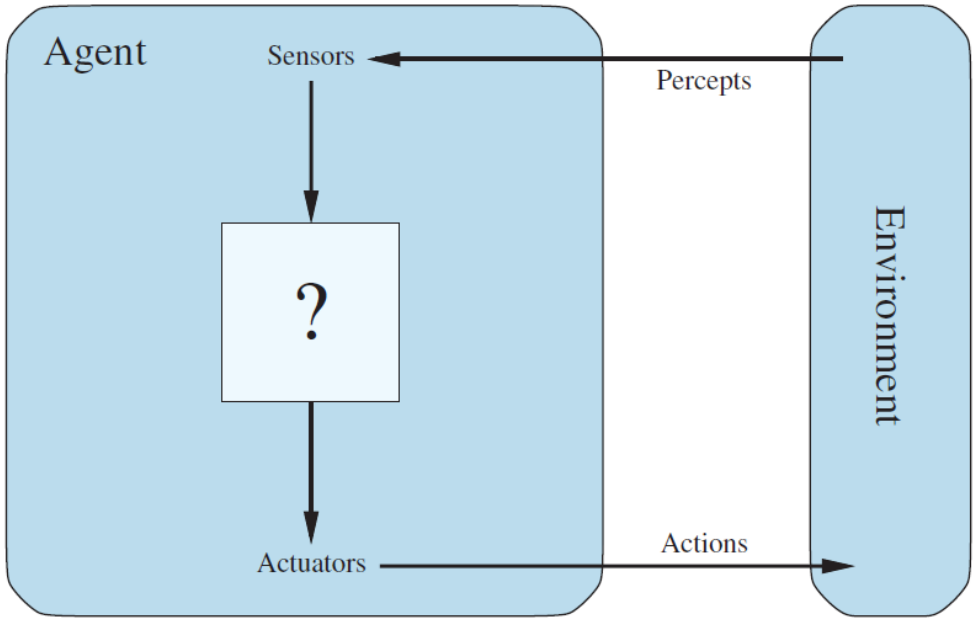


image source

Intelligent agents: environment and perception

The environment which intelligent agents interact with could be everything—even the entire universe!

In practice it is just that part of the universe whose state we care about when designing this agent: the part that affects what the agent perceives and that is affected by the agent's actions.

The term "percept" is used to refer to the content an agent's sensors are perceiving.

An agent's percept sequence is the complete history of everything the agent has ever perceived.

In general, an agent's choice of action at any given instant can depend on its built-in knowledge and on the entire percept sequence observed to date, but not on anything it hasn't perceived.

Intelligent agents: examples

Tasks such as playing a game of chess, performing a robotic task, finding a path in a maze, replying to natural questions posed by a human, all require a sequence of decisions from an intelligent agent.

An example of an agent might be a robot, a chatbot, a chessplaying entity, or a route planner.

Intelligent agents: examples

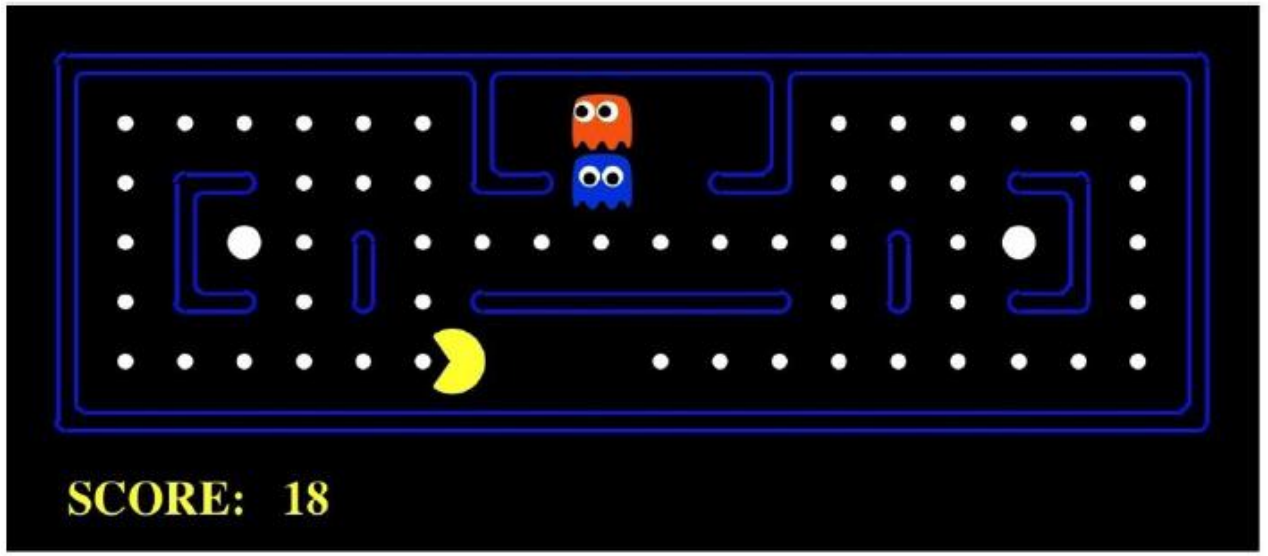
Table 1.2: Agent examples

Agent	Sensor	Actuator	Objective	State	Environment
Cleaning Robot	Camera Joint sensor	Limbs Joints	Cleanliness evaluation	Object/joint positions	Cleaned space
Chess agent	Board input interface	Move output interface	Position evaluation	Chess position	Chess board
Self-driving car	Camera Sound sensor	Car control interface	Driving safety and goals	Speed/Position in traffic	Traffic conditions
Chatbot	Keyboard	Screen	Chat evaluation	Dialog history	Chat participants

image source

Intelligent agents: examples

Pac-Man is a registered trademark of Namco-Bandai Games, used here for educational purposes



Pac-Man is an agent!

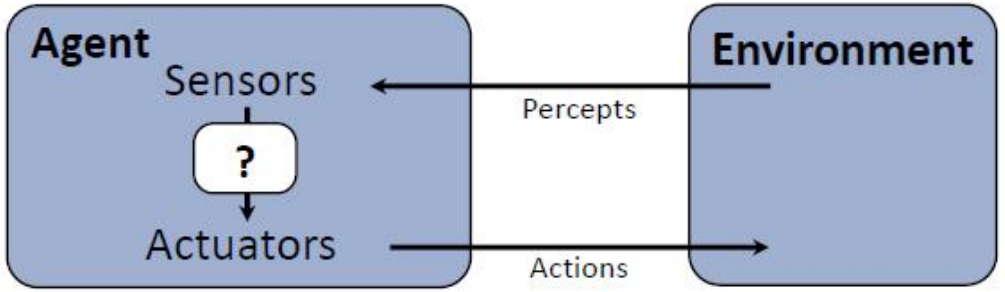


image source

Agent-based AI tasks

It is only in the case of problems associated with a sequence of decisions that it becomes important to use the concept of agent and a state of the system.

Such settings are referred to as sequential.

An intelligent agent plays the same role that a human plays in a biological system, based on a sequence of choices that achieve a desired goal.

Non-agent-based AI tasks

One does not always encounter the notion of an agent in all forms of AI.

For example, the concept of agent rarely arises for single-decision problems in machine learning; an example is that of classification, where one is trying to categorize an instance into one of many classes (e.g., categorize whether an image contains animals or not).

Such environmental settings are referred to as episodic, and even though the notion of agent is implicit, it is rarely used in practice.

In these cases, the solution to this problem is a single-step process, and therefore the question of system state or environment is not as important as it would be in a sequential process where an action affects subsequent states.

Therefore, in such cases, the implicit agent receives a single percept and performs a single action. The next percept-action pair is independent of the current one. In other words, the individual episodes are independent of one another.

Rational agents

Rationality and performance measure

A rational agent is one that "does the right thing".

In the AI landscape, "doing the right thing" for an agent is typically recognized as corresponding to how good the consequences yielded by the agent are.

- This is called consequentialism.

More precisely, a sequence of actions performed by an agent causes the environment to go through a sequence of states. If such a sequence of states is desirable, then the agent has performed well (it has "done the right thing").

This notion of desirability is captured by an ad-hoc defined performance measure that evaluates any given sequence of environment states.

Rationality vs. humanity

Humans have desires and preferences of their own, so the notion of rationality as applied to humans has to do with their success in choosing actions that produce sequences of environment states that are desirable from their point of view.

Machines, on the other hand, do not have desires and preferences of their own; the performance measure is, initially at least, in the mind of the designer of the machine, or in the mind of the users the machine is designed for.

Definition of a rational agent

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Designing performance measures

It can be quite hard to formulate a performance measure correctly.

For instance, consider a simple robotic vacuum-cleaning agent in a world consisting of squares that can be either dirty or clean.

- Measure performance by the amount of dirt cleaned up in a single eight-hour shift => a rational agent can maximize this performance measure by cleaning up the dirt, then dumping it all on the floor, then cleaning it up again, and so on => not exactly a good behavior!
- A more suitable performance measure would reward the agent for having a clean floor. For example, one point could be awarded for each clean square at each time step (perhaps with a penalty for electricity consumed and noise generated).

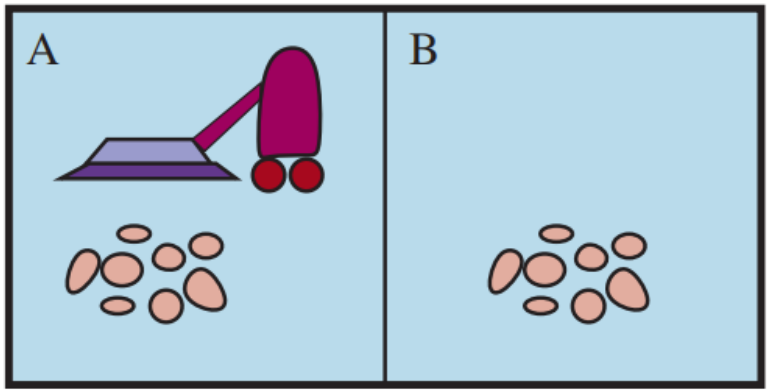


image source

As a general rule, it is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave.

Example of rational agent

Consider again the simple vacuum-cleaner agent that cleans a square if it is dirty and moves to the other square if not.

Assume the following:

- The performance measure awards one point for each clean square at each time step, over a "lifetime" of 1000 time steps, minus a penalty of <1 point for every action (for electricity consumed and noise generated).
- The available actions are Right, Left, Suck, Do nothing.
- The "geography" of the environment is known a priori but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The Right and Left actions move the agent one square except when this would take the agent outside the environment, in which case the agent remains where it is.
- The agent correctly perceives its location and whether that location contains dirt.

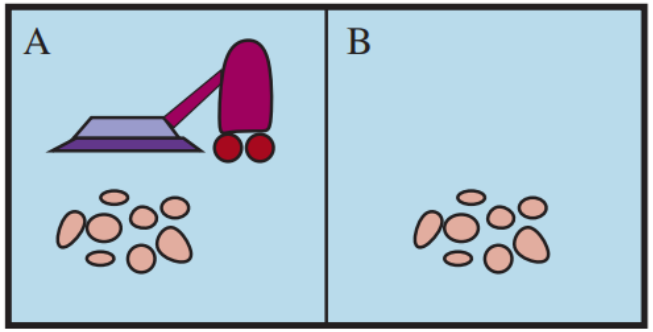


image source

A vacuum-cleaner agent that operates according to the above rules and tries to maximize the above performance measure is a rational agent.

Learning

It is desirable that a rational agent not only gather information but also to learn as much as possible from what it perceives.

The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented.

There are cases in which the environment is completely known a priori and completely predictable. In such cases, the agent need not perceive or learn; it simply acts correctly. But these are extreme and rather unlikely cases.

Designing an agent: Performance, Environment, Actuators, Sensors (PEAS)

PEAS: Performance, Environment, Actuators, Sensors

To design an agent we need to specify a “PEAS”, that is:

- performance measure,
- environment,
- actuators,
- sensors.

PEAS for automated taxi

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

Figure 2.4 PEAS description of the task environment for an automated taxi driver.

image source

PEAS for other agents

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments	Touchscreen/voice entry of symptoms and findings
Satellite image analysis system	Correct categorization of objects, terrain	Orbiting satellite, downlink, weather	Display of scene categorization	High-resolution digital camera
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, tactile and joint angle sensors

image source

PEAS for other agents

Refinery controller	Purity, yield, safety	Refinery, raw materials, operators	Valves, pumps, heaters, stirrers, displays	Temperature, pressure, flow, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, feedback, speech	Keyboard entry, voice

Figure 2.5 Examples of agent types and their PEAS descriptions.

image source

Types of environment

- Fully observable vs. partially observable
- Single agent vs. multi-agent
- Deterministic vs. nondeterministic
- Episodic vs. sequential
- Static vs. dynamic
- Discrete vs. continuous
- Known vs. unknown

Types of environment

Fully observable vs. partially observable

- Fully observable: if agent's sensors detect all aspects that are relevant to the choice of action; relevance, in turn, depends on the performance measure.
- An environment might be partially observable because of noisy or inaccurate sensors or because parts of the state are simply missing from the sensor data
  - For example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares, and an automated taxi cannot see what other drivers are thinking.

Single agent vs. multi-agent

- An agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two-agent environment.

Types of environment

Deterministic vs. nondeterministic

- Deterministic: if the next state of the environment is completely determined by the current state and the action executed by the agent(s).
- Environment in taxi driving is nondeterministic, because one can never predict the behavior of traffic exactly; moreover, one's tires may blow out unexpectedly and one's engine may seize up without warning.
- The vacuum-cleaning world as we described is deterministic, but variations can include nondeterministic elements such as randomly appearing dirt and an unreliable suction mechanism.

"Stochastic" is sometimes used as a synonym for "nondeterministic".

Here we make a distinction between these two terms:

- Stochastic: probabilities explicitly present (e.g., "there's a 25% chance of rain tomorrow")
- Nondeterministic: possibilities are listed without being quantified (e.g., "there's a chance of rain tomorrow").

Types of environment

Episodic vs. sequential

- Episodic: agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes.
  - Many classification tasks are episodic. For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions; moreover, the current decision doesn't affect whether the next part is defective.
- Sequential: the current decision could affect all future decisions.
  - Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences.

Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

Types of environment

Discrete vs. continuous

- The discrete/continuous distinction applies to the state of the environment, to the way how time is handled, and to the percepts and actions of the agent.
- For example, the chess environment has a finite number of distinct states (excluding the clock).
- Chess also has a discrete set of percepts and actions.
- Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time.
- Taxi-driving actions are also continuous (steering angles, etc.). Input from digital cameras is discrete, strictly speaking, but is typically treated as representing continuously varying intensities and locations.

Types of environment

Known vs. unknown

- Strictly speaking, this distinction refers not to the environment itself but to the agent's (or designer's) state of knowledge about the "laws of physics" of the environment.
- In a known environment, the outcomes (or outcome probabilities if the environment is nondeterministic) for all actions are given.
- If the environment is unknown, the agent will have to learn how it works in order to make good decisions.

The distinction between known and unknown environments is not the same as the one between fully and partially observable environments.

- It is quite possible for a known environment to be partially observable—for example, in solitaire card games, I know the rules but am still unable to see the cards that have not yet been turned over.
- Conversely, an unknown environment can be fully observable—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them

Types of environment

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

image source

Figure 2.6 Examples of environments and their characteristics.

Perhaps the hardest case is *partially observable*, *multiagent*, *nondeterministic*, *sequential*, *dynamic*, *continuous*, and *unknown*.

Designing an agent: agent program

Agent function and agent program

Formally speaking, the behavior of an agent behavior is described by an agent function that maps any given percept sequence to an action.

The agent function is an abstract mathematical description of agent's behavior.

Internally to an agent, the agent function is implemented by what is called agent program.

- The agent program is a concrete implementation of the agent function, running within some physical system.

Agent architecture

The agent program is assumed to run on some sort of computing device with physical sensors and actuators: this is called “the agent architecture”

The architecture might be just an ordinary PC, or it might be a robotic car with several onboard computers, cameras, and other sensors.

In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds program’s action choices to the actuators as they are generated.

The job of AI is mainly to design agent programs, while agent architectures are somehow given for granted.

Agent program and agent architecture

The job of AI is to design an agent program that implements the agent function, i.e., the mapping from percepts to actions.

The agent program is assumed to run on some sort of computing device with physical sensors and actuators: this is called "the agent architecture"

Simply speaking, an agent is a combination of its program and its architecture:

agent = program + architecture

Table-driven agent

Perhaps the simplest way to design an agent program is to build a lookup table that maps every possible percept sequence to the corresponding most desirable action to be performed.

function TABLE-DRIVEN-AGENT(*percept*) **returns** an action
  persistent: *percepts*, a sequence, initially empty
    table, a table of actions, indexed by percept sequences, initially fully specified

  append *percept* to the end of *percepts*
  action ← LOOKUP(*percepts*, *table*)
  return *action*

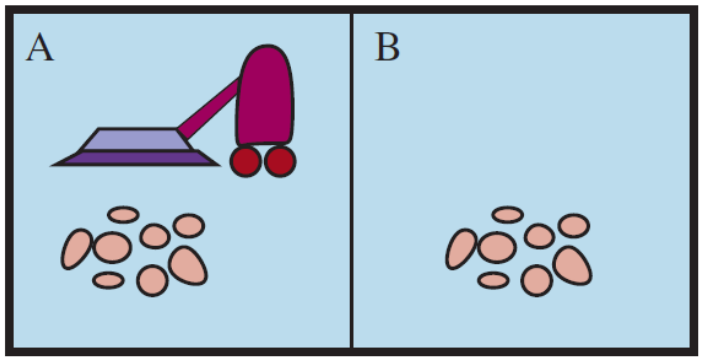
Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

image source

Table-driven agent

A simple table-driven vacuum-cleaner agent:

image source



A vacuum-cleaner environment with just two locations. Each location can be clean or dirty, and the agent can move left or right and can clean the square that it occupies.

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Dirty]$	*Suck*
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Clean], [A, Dirty]$	*Suck*
⋮	⋮

image source

Partial tabulation of a simple agent function for the vacuum-cleaner world shown on the left. The agent cleans the current square if it is dirty, otherwise it moves to the other square. Note that the table is of unbounded size unless there is a restriction on the length of possible percept sequences.

Table-driven agent

Building a table-driven agent is feasible only in a very few simple cases.

In most cases, the lookup table is going to be of huge (or even infinite) size.

Let P be the set of possible percepts and let T be the lifetime of the agent (the total number of percepts it will receive).

The lookup table will contain $\sum_{t=1}^{T} |P|^t$ entries.

The lookup table of chess has at least 10^{150} entries.

- Much more than the total number of atoms in the observable universe, which is estimated to be $<10^{82}$!

Going beyond table-driven agents

The key challenge for AI is to find out how to write agent programs that, to the maximum extent possible, produce rational behavior from a smallish program rather than from a vast table.

For instance, a non-table-driven vacuum-cleaner agent program might be:

```
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
```

image source

Types of agent

Simple reflex agent

- takes actions based on the current percept only

Model-based reflex agent

- maintains internal state to track aspects of the world that are not evident in the current percept; takes actions based on the current percept and their internal state

Goal-based agent

- acts to achieve their goals

Utility-based agent

- tries to maximize their own expected “happiness”

All agents can improve their performance through learning.

Types of agent: simple reflex agent

Taxi-driving agent:

It brakes based on the condition-action rule "*if car-in-front-is-braking then initiate-braking*".

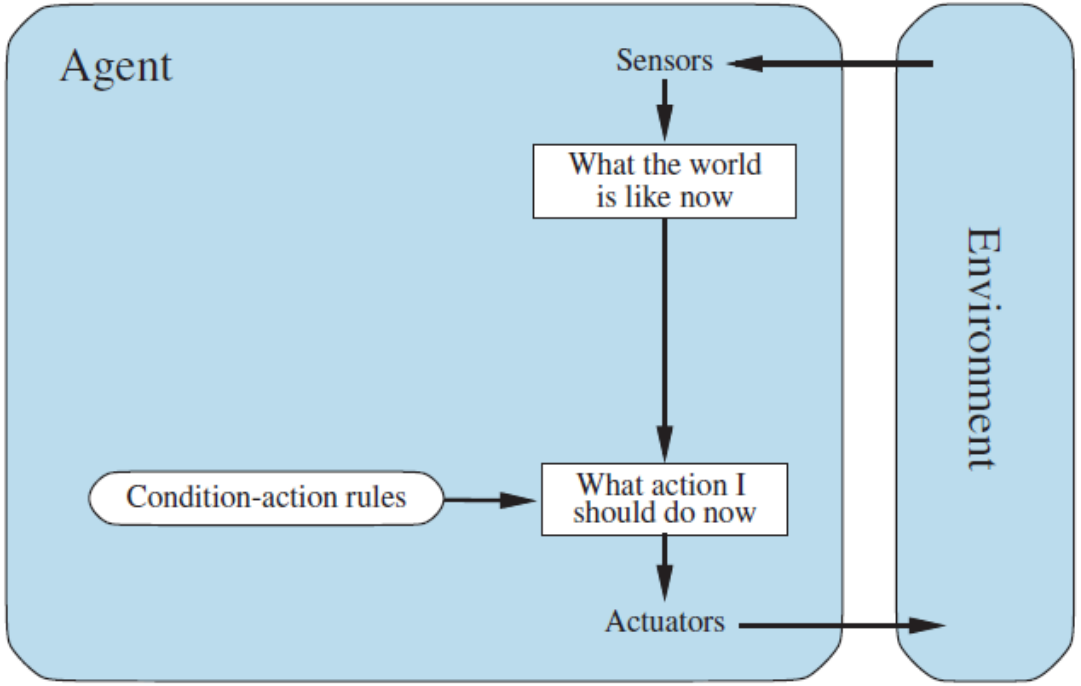


Figure 2.9 Schematic diagram of a simple reflex agent. We use rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process.

image source

Types of agent: model-based reflex agent

Taxi-driving agent:

For driving tasks other than braking, such as changing lanes, the agent needs to keep track of where the other cars are (if it can't see them all at once).

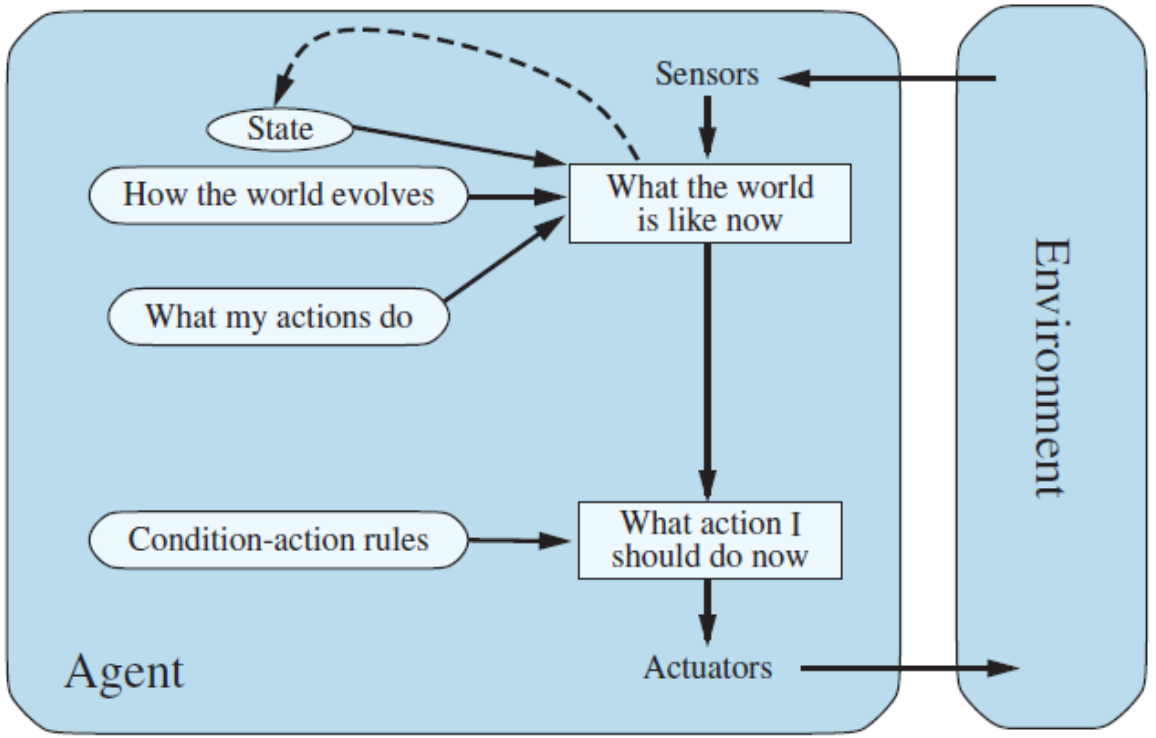


Figure 2.11 A model-based reflex agent.

image source

Types of agent: goal-based agent

Taxi-driving agent:

Knowing something about the current state of the environment is not always enough to decide what to do.

For example, at a road junction, the taxi can turn left, turn right, or go straight on.

The correct decision depends on its goal, i.e., on where the taxi is trying to get to.

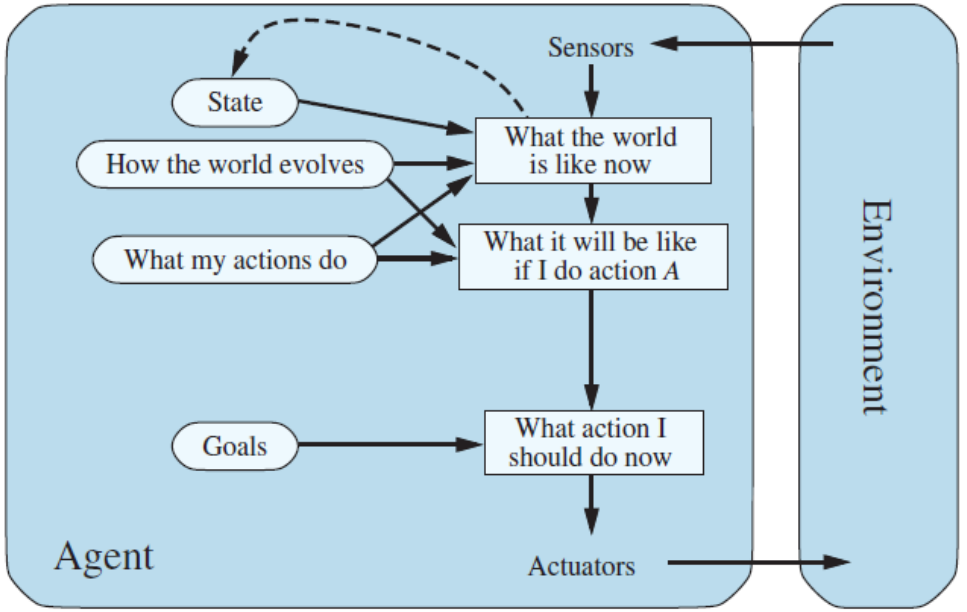


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

image source

Sometimes goal-based action selection is straightforward, i.e., when goal satisfaction results immediately from a single action.

Sometimes it will be more tricky, i.e., when the agent has to consider long sequences of twists and turns in order to find a way to achieve the goal.

Search and planning are the subfields of AI devoted to finding action sequences that achieve the agent's goals.

Types of agent: utility-based agent

Taxi-driving agent:

Goals alone are not enough to generate high-quality behavior in most environments: many action sequences will get the taxi to its destination (thereby achieving the goal), but some are quicker, safer, more reliable, or cheaper than others.

Goals just provide a crude binary distinction between "happy" and "unhappy" states.

A more general performance measure should allow a comparison of different environment states according to exactly how happy they would make the agent.

The measure of "happiness" of an agent is mathematically expressed by a so-called utility function.

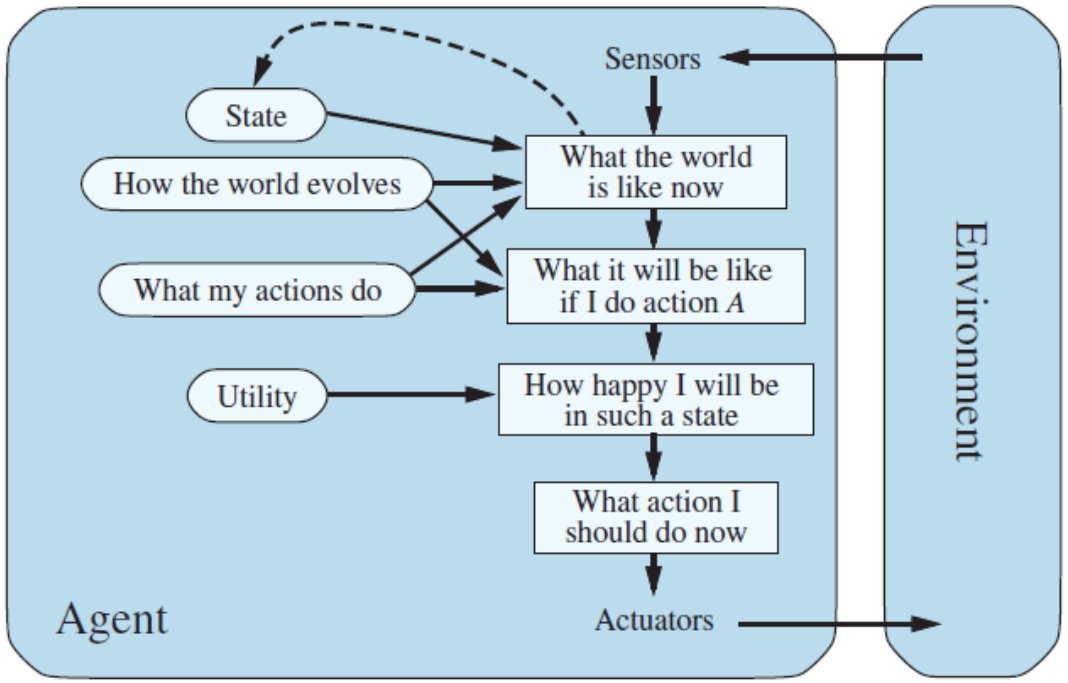


Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

image source

Types of agent: utility-based agent

Utility function vs. performance measure:

- Agent's utility function is essentially an internalization of the performance measure.
- Provided that the internal utility function and the external performance measure are in agreement, an agent that chooses actions to maximize its utility will be rational according to the external performance measure.
  - Designing an agent like this is not the only way of achieving rationality!

Types of agent: learning agents

A learning agent in AI is the type of agent that can learn from its past experiences.

It starts to act with basic knowledge and then is able to act and adapt automatically through learning.

Any type of agent (model-based, goal-based, utility-based, etc.) can be built as a learning agent (or not).

Historical note:

Turing, in 1950, considered the idea of actually programming his intelligent machines by hand. He estimates how much work this might take and concludes, "*Some more expeditious method seems desirable*".

The method he proposes is to build learning machines and then to teach them. In many areas of AI, this is now the preferred method for creating state-of-the-art systems.

Types of agent: learning agents

Taxi-driver agent:

Suppose it receives no tips from passengers who have been thoroughly shaken up during the trip.

The external performance standard must inform the agent that the loss of tips is a negative contribution to its overall performance.

Then the agent might be able to learn that violent maneuvers do not contribute to its own utility.

In a sense, the performance standard distinguishes part of the incoming percept as a reward (or penalty) that provides direct feedback on the quality of the agent's behavior.

This reward/penalty can be used by the agent to learn about good and bad behaviors.

Example: paper buying agent

Paper buying agent

Source:

- Example 2.1 of Poole and Mackworth's book ("*Artificial Intelligence: Foundations of Computational Agents*"), 3rd ed.
- **`utilities.py`**, **`display.py`**, **`agents.py`**, and **`agentBuying.py`** files in companion Python code repository of Poole and Mackworth's book
  - https://artint.info/AIPython/aipython.zip
  - Section 2.2 of https://artint.info/AIPython/aipython/aipython.pdf

Paper buying agent

Agent responsible for guaranteeing that the available amount of some household commodity (e.g., toilet paper) is adequate.

It monitors (on a daily basis):

- online deals for buying paper;
- how much paper the household has in stock.

It decides (on a daily basis) whether to order more paper and how much to order.

Percepts:

- price of the paper, and
- amount of paper currently in stock.

Action: number of units of paper to be ordered (zero if the agent decides not to order any).

Paper buying agent

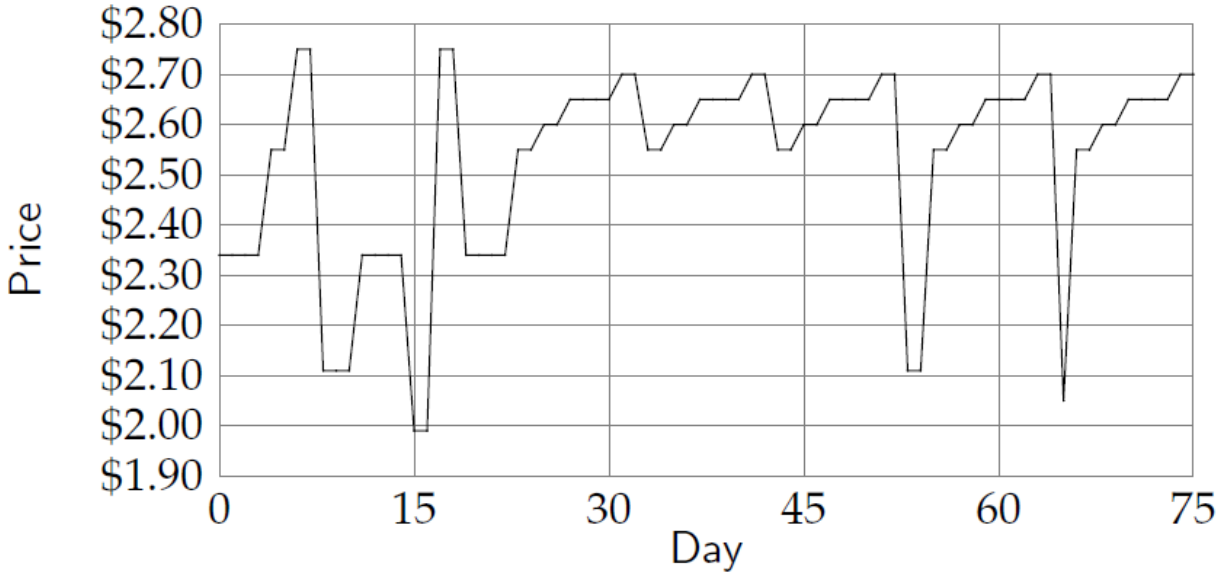
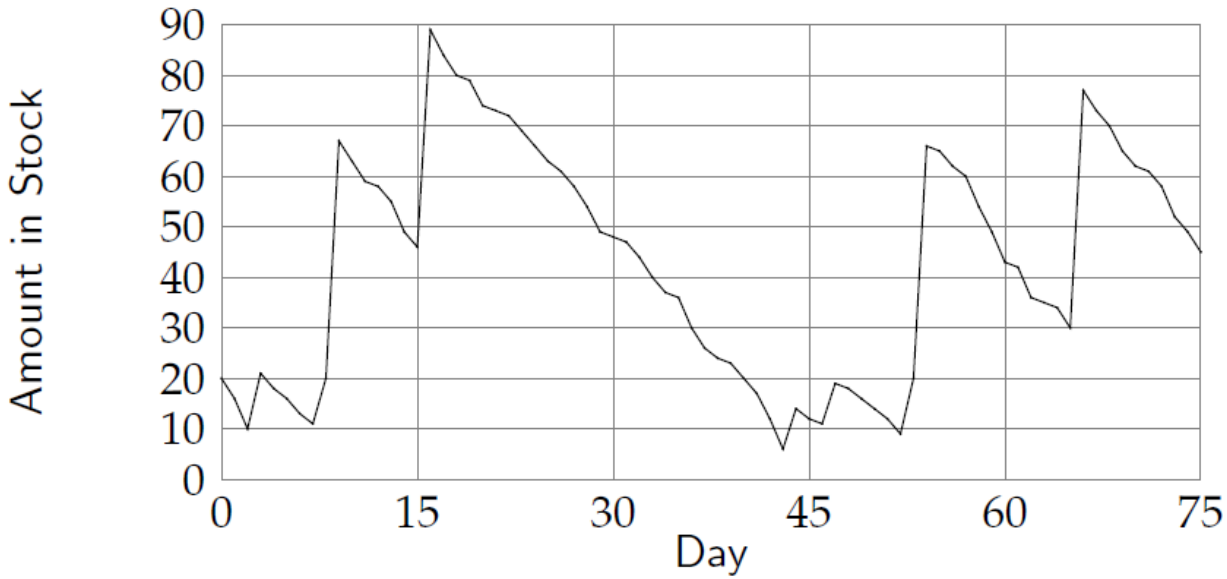


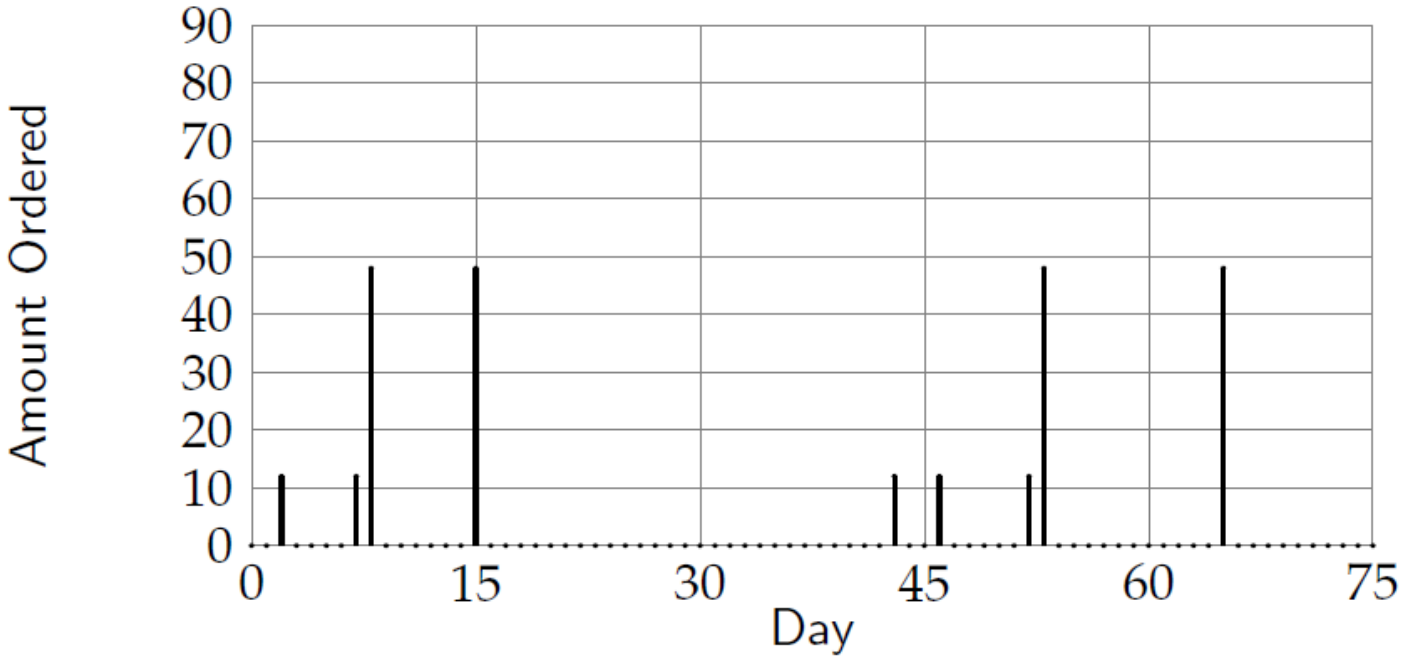
image source

A possible percept sequence
(for the two percepts,
namely price and amount in stock)



Paper buying agent

image source



A possible sequence of actions performed

Paper buying agent: `display.py`

```
# display.py - A simple way to trace the intermediate steps of algorithms.
class Displayable(object):
    """Class that uses 'display'.
    The amount of detail is controlled by max_display_level
    """
    max_display_level = 1   # can be overridden in subclasses or instances

    def display(self,level,*args,**nargs):
        """print the arguments if level is less than or equal to the
        current max_display_level.
        level is an integer.
        the other arguments are whatever arguments print can take.
        """
        if level <= self.max_display_level:
            print(*args, **nargs)  ##if error you are using Python2 not Python3
```

More on *args and **nargs: https://www.linkedin.com/pulse/decoding-python-functions-default-positional-keyword-benjamin/

Paper buying agent: `agents.py`

```
11  from display import Displayable
12
13  class Agent(Displayable):
14
15      def initial_action(self, percept):
16          """return the initial action."""
17          return self.select_action(percept)   # same as select_action
18
19      def select_action(self, percept):
20          """return the next action (and update internal state) given percept
21          percept is variable:value dictionary
22          """
23          raise NotImplementedError("go")   # abstract method
```

An agent takes the percept, updates its internal state (if any), and output its next action.
An agent implements the method **`select_action`** that takes percept and returns its next action.

Paper buying agent: `agents.py`

```
24
25  class Environment(Displayable):
26      def initial_percept(self):
27          """returns the initial percept for the agent"""
28          raise NotImplementedError("initial_percept")   # abstract method
29
30      def do(self, action):
31          """does the action in the environment
32          returns the next percept """
33          raise NotImplementedError("Environment.do")   # abstract method
```

An environment takes in actions of the agents, updates its internal state and returns the next percept, using the method **`do`**.

The environment implements a **`do(action)`** method where action is a variable-value dictionary. This returns a percept, which is also a variable-value dictionary. The use of dictionaries allows for structured actions and percepts.

Paper buying agent: `agents.py`

```
34
35  class Simulate(Displayable):
36      """simulate the interaction between the agent and the environment
37      for n time steps.
38      Returns a pair of the agent state and the environment state.
39      """
40      def __init__(self,agent, environment):
41          self.agent = agent
42          self.env = environment
43          self.percept = self.env.initial_percept()
44          self.percept_history = [self.percept]
45          self.action_history = []
46
47      def go(self, n):
48          for i in range(n):
49              action = self.agent.select_action(self.percept)
50              self.display(2,f"i={i} action={action}")
51              self.percept = self.env.do(action)
52              self.display(2,f"       percept={self.percept}")
53
```

The methods **`do`** and **`select_action`** are chained together to build a simulator.

In order to start this, we need either an action or a percept.

There are two variants used:

- An agent implements the **`initial_action`** method which is used initially.
- The environment implements the **`initial_percept`** method which gives the initial percept. This is the method used here.

Paper buying agent: `agentBuying.py`

```python
import random
from agents import Agent, Environment, Simulate
from utilities import select_from_dist

class TP_env(Environment):
    price_delta = [0, 0, 0, 21, 0, 20, 0, -64, 0, 0, 23, 0, 0, 0, -35,
        0, 76, 0, -41, 0, 0, 0, 21, 0, 5, 0, 5, 0, 0, 0, 5, 0, -15, 0, 5,
       0, 5, 0, -115, 0, 115, 0, 5, 0, -15, 0, 5, 0, 5, 0, 0, 0, 5, 0,
       -59, 0, 44, 0, 5, 0, 5, 0, 0, 0, 5, 0, -65, 50, 0, 5, 0, 5, 0, 0,
       0, 5, 0]
    sd = 5  # noise standard deviation

    def __init__(self):
        """paper buying agent"""
        self.time=0
        self.stock=20
        self.stock_history = []  # memory of the stock history
        self.price_history = []  # memory of the price history
```

The environment state is given in terms of the time and the amount of paper in stock.

It also remembers the in-stock history and the price history.

The percept consists of the price and the amount of paper in stock.

The action of the agent is the number of items to buy.

Paper buying agent: `agentBuying.py`

```
29
30      def initial_percept(self):
31          """return initial percept"""
32          self.stock_history.append(self.stock)
33          self.price = round(234+self.sd*random.gauss(0,1))
34          self.price_history.append(self.price)
35          return {'price': self.price,
36                  'instock': self.stock}
37
38      def do(self, action):
39          """does action (buy) and returns percept consisting of price and instock"""
40          used = select_from_dist({6:0.1, 5:0.1, 4:0.1, 3:0.3, 2:0.2, 1:0.2})
41          # used = select_from_dist({7:0.1, 6:0.2, 5:0.2, 4:0.3, 3:0.1, 2:0.1}) # uses more paper
42          bought = action['buy']
43          self.stock = self.stock+bought-used
44          self.stock_history.append(self.stock)
45          self.time += 1
46          self.price =  round(self.price
47                              + self.price_delta[self.time%len(self.price_delta)] # repeating pattern
48                              + self.sd*random.gauss(0,1)) # plus randomness
49          self.price_history.append(self.price)
50          return {'price': self.price,
51                  'instock': self.stock}
```

Paper buying agent: `agentBuying.py`

```
52
53  class TP_agent(Agent):
54      def __init__(self):
55          self.spent = 0
56          percept = env.initial_percept()
57          self.ave = self.last_price = percept['price']
58          self.instock = percept['instock']
59          self.buy_history = []
60
```

Paper buying agent: `agentBuying.py`

```
61      def select_action(self, percept):
62          """return next action to carry out
63          """
64          self.last_price = percept['price']
65          self.ave = self.ave+(self.last_price-self.ave)*0.05
66          self.instock = percept['instock']
67          if self.last_price < 0.9*self.ave and self.instock < 60:
68              tobuy = 48
69          elif self.instock < 12:
70              tobuy = 12
71          else:
72              tobuy = 0
73          self.spent += tobuy*self.last_price
74          self.buy_history.append(tobuy)
75          return {'buy': tobuy}
```

Paper buying agent: `agentBuying.py`

```
76
77   env = TP_env()
78   ag = TP_agent()
79   sim = Simulate(ag,env)
80   sim.go(90)
81   ag.spent/env.time   ## average spent per time period
```

Paper buying agent: `utilities.py`

```
# utilities.py - AIPython useful utilities
# AIFCA Python code Version 0.9.13 Documentation at https://aipython.org
# Download the zip file and read aipython.pdf for documentation

# Artificial Intelligence: Foundations of Computational Agents https://artint.info
# Copyright 2017-2024 David L. Poole and Alan K. Mackworth
# This work is licensed under a Creative Commons
# Attribution-NonCommercial-ShareAlike 4.0 International License.
# See: https://creativecommons.org/licenses/by-nc-sa/4.0/deed.en
```

```
def select_from_dist(item_prob_dist):
    """ returns a value from a distribution.
    item_prob_dist is an item:probability dictionary, where the
        probabilities sum to 1.
    returns an item chosen in proportion to its probability
    """
    ranreal = random.random()
    for (it,prob) in item_prob_dist.items():
        if ranreal < prob:
            return it
        else:
            ranreal -= prob
    raise RuntimeError(f"{item_prob_dist} is not a probability distribution")
```

Questions?