INTELLIGENT AGENTS

2

*In which we discuss the nature of agents, perfect or otherwise, the diversity of environments, and the resulting menagerie of agent types.*

Chapter 1 identiﬁed the concept of **rational agents** as central to our approach to artiﬁcial intelligence. In this chapter, we make this notion more concrete. We will see that the concept of rationality can be applied to a wide variety of agents operating in any imaginable environ- ment. Our plan in this book is to use this concept to develop a small set of design principles for building successful agents—systems that can reasonably be called **intelligent**.

We begin by examining agents, environments, and the coupling between them. The observation that some agents behave better than others leads naturally to the idea of a rational agent—one that behaves as well as possible. How well an agent can behave depends on the nature of the environment; some environments are more difﬁcult than others. We give a crude categorization of environments and show how properties of an environment inﬂuence the design of suitable agents for that environment. We describe a number of basic “skeleton” agent designs, which we ﬂesh out in the rest of the book.

## AGENTS AND ENVIRONMENTS

ENVIRONMENT

An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and

SENSOR acting upon that environment through **actuators**. This simple idea is illustrated in Figure 2.1. ACTUATOR 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 keystrokes, ﬁle contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing ﬁles, and sending network packets.

PERCEPT

PERCEPT SEQUENCE



We use the term **percept** to refer to the agent’s perceptual inputs at any given instant. 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 the entire percept sequence observed to date, but not on anything it hasn’t perceived.* By specifying the agent’s choice of action for every possible percept sequence, we have said more or less everything

34

Section 2.1. Agents and Environments 35

**Figure 2.1** Agents interact with environments through sensors and actuators.

Actions

Actuators

Sensors

Percepts

Agent

?

Environment

AGENT FUNCTION

AGENT PROGRAM



there is to say about the agent. Mathematically speaking, we say that an agent’s behavior is described by the **agent function** that maps any given percept sequence to an action.

We can imagine *tabulating* the agent function that describes any given agent; for most agents, this would be a very large table—inﬁnite, in fact, unless we place a bound on the length of percept sequences we want to consider. Given an agent to experiment with, we can, in principle, construct this table by trying out all possible percept sequences and recording which actions the agent does in response.1 The table is, of course, an *external* characterization of the agent. *Internally*, the agent function for an artiﬁcial agent will be implemented by an **agent program**. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.

To illustrate these ideas, we use a very simple example—the vacuum-cleaner world shown in Figure 2.2. This world is so simple that we can describe everything that happens; it’s also a made-up world, so we can invent many variations. This particular world has just two locations: squares *A* and *B*. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square. A partial tabulation of this agent function is shown in Figure 2.3 and an agent program that implements it appears in Figure 2.8 on page 48.

Looking at Figure 2.3, we see that various vacuum-world agents can be deﬁned simply by ﬁlling in the right-hand column in various ways. The obvious question, then, is this: *What is the right way to ﬁll out the table?* In other words, what makes an agent good or bad, intelligent or stupid? We answer these questions in the next section.

1 If the agent uses some randomization to choose its actions, then we would have to try each sequence many times to identify the probability of each action. One might imagine that acting randomly is rather silly, but we show later in this chapter that it can be very intelligent.

36 Chapter 2. Intelligent Agents



**Figure 2.2** A vacuum-cleaner world with just two locations.

B

A

|  |  |
| --- | --- |
| Percept sequence | Action |
| [*A, Clean* ]  [*A, Dirty* ]  [*B, Clean*]  [*B, Dirty* ]  [*A, Clean* ], [*A, Clean* ]  [*A, Clean* ], [*A, Dirty* ] | *Right Suck Left Suck Right Suck* |
| .  [*A, Clean* ], [*A, Clean* ], [*A, Clean* ]  [*A, Clean* ], [*A, Clean* ], [*A, Dirty* ] | .  *Right Suck* |
| . | . |
| **Figure 2.3** Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2. | |

Before closing this section, we should emphasize that the notion of an agent is meant to be a tool for analyzing systems, not an absolute characterization that divides the world into agents and non-agents. One could view a hand-held calculator as an agent that chooses the action of displaying “4” when given the percept sequence “2 + 2 =,” but such an analysis would hardly aid our understanding of the calculator. In a sense, all areas of engineering can be seen as designing artifacts that interact with the world; AI operates at (what the authors consider to be) the most interesting end of the spectrum, where the artifacts have signiﬁcant computational resources and the task environment requires nontrivial decision making.

## GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

RATIONAL AGENT

A **rational agent** is one that does the right thing—conceptually speaking, every entry in the table for the agent function is ﬁlled out correctly. Obviously, doing the right thing is better than doing the wrong thing, but what does it mean to do the right thing?

Section 2.2. Good Behavior: The Concept of Rationality 37

PERFORMANCE MEASURE



DEFINITION OF A RATIONAL AGENT



We answer this age-old question in an age-old way: by considering the *consequences* of the agent’s behavior. When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well. This notion of desirability is captured by a **performance measure** that evaluates any given sequence of environment states.

Notice that we said *environment* states, not *agent* states. If we deﬁne success in terms of agent’s opinion of its own performance, an agent could achieve perfect rationality simply by deluding itself that its performance was perfect. Human agents in particular are notorious for “sour grapes”—believing they did not really want something (e.g., a Nobel Prize) after not getting it.

Obviously, there is not one ﬁxed performance measure for all tasks and agents; typically, a designer will devise one appropriate to the circumstances. This is not as easy as it sounds. Consider, for example, the vacuum-cleaner agent from the preceding section. We might propose to measure performance by the amount of dirt cleaned up in a single eight-hour shift. With a rational agent, of course, what you ask for is what you get. 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. 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). *As a general rule, it is better to design performance measures according to what one actually wants in the environment, rather than according to how one thinks the agent should behave.*

Even when the obvious pitfalls are avoided, there remain some knotty issues to untangle. For example, the notion of “clean ﬂoor” in the preceding paragraph is based on average cleanliness over time. Yet the same average cleanliness can be achieved by two different agents, one of which does a mediocre job all the time while the other cleans energetically but takes long breaks. Which is preferable might seem to be a ﬁne point of janitorial science, but in fact it is a deep philosophical question with far-reaching implications. Which is better— a reckless life of highs and lows, or a safe but humdrum existence? Which is better—an economy where everyone lives in moderate poverty, or one in which some live in plenty while others are very poor? We leave these questions as an exercise for the diligent reader.

# Rationality

What is rational at any given time depends on four things:

* + - * The performance measure that deﬁnes the criterion of success.
      * The agent’s prior knowledge of the environment.
      * The actions that the agent can perform.
      * The agent’s percept sequence to date. This leads to a **deﬁnition of a rational agent**:

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

38 Chapter 2. Intelligent Agents

OMNISCIENCE

Consider the simple vacuum-cleaner agent that cleans a square if it is dirty and moves to the other square if not; this is the agent function tabulated in Figure 2.3. Is this a rational agent? That depends! First, we need to say what the performance measure is, what is known about the environment, and what sensors and actuators the agent has. Let us assume the following:

* The performance measure awards one point for each clean square at each time step, over a “lifetime” of 1000 time steps.
* The “geography” of the environment is known *a priori* (Figure 2.2) but the dirt distri- bution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The *Left* and *Right* actions move the agent left and right except when this would take the agent outside the environment, in which case the agent remains where it is.
* The only available actions are *Left* , *Right* , and *Suck* .
* The agent correctly perceives its location and whether that location contains dirt.

We claim that *under these circumstances* the agent is indeed rational; its expected perfor- mance is at least as high as any other agent’s. Exercise 2.2 asks you to prove this.

One can see easily that the same agent would be irrational under different circum- stances. For example, once all the dirt is cleaned up, the agent will oscillate needlessly back and forth; if the performance measure includes a penalty of one point for each movement left or right, the agent will fare poorly. A better agent for this case would do nothing once it is sure that all the squares are clean. If clean squares can become dirty again, the agent should occasionally check and re-clean them if needed. If the geography of the environment is un- known, the agent will need to explore it rather than stick to squares *A* and *B*. Exercise 2.2 asks you to design agents for these cases.

# Omniscience, learning, and autonomy

We need to be careful to distinguish between rationality and **omniscience**. An omniscient agent knows the *actual* outcome of its actions and can act accordingly; but omniscience is impossible in reality. Consider the following example: I am walking along the Champs Elyse´es one day and I see an old friend across the street. There is no trafﬁc nearby and I’m not otherwise engaged, so, being rational, I start to cross the street. Meanwhile, at 33,000 feet, a cargo door falls off a passing airliner,2 and before I make it to the other side of the street I am ﬂattened. Was I irrational to cross the street? It is unlikely that my obituary would read “Idiot attempts to cross street.”

This example shows that rationality is not the same as perfection. Rationality max- imizes *expected* performance, while perfection maximizes *actual* performance. Retreating from a requirement of perfection is not just a question of being fair to agents. The point is that if we expect an agent to do what turns out to be the best action after the fact, it will be impossible to design an agent to fulﬁll this speciﬁcation—unless we improve the performance of crystal balls or time machines.

2 See N. Henderson, “New door latches urged for Boeing 747 jumbo jets,” *Washington Post*, August 24, 1989.

Section 2.2. Good Behavior: The Concept of Rationality 39

INFORMATION GATHERING

EXPLORATION

LEARNING

AUTONOMY

Our deﬁnition of rationality does not require omniscience, then, because the rational choice depends only on the percept sequence *to date*. We must also ensure that we haven’t inadvertently allowed the agent to engage in decidedly underintelligent activities. For exam- ple, if an agent does not look both ways before crossing a busy road, then its percept sequence will not tell it that there is a large truck approaching at high speed. Does our deﬁnition of rationality say that it’s now OK to cross the road? Far from it! First, it would not be rational to cross the road given this uninformative percept sequence: the risk of accident from cross- ing without looking is too great. Second, a rational agent should choose the “looking” action before stepping into the street, because looking helps maximize the expected performance. Doing actions *in order to modify future percepts*—sometimes called **information gather- ing**—is an important part of rationality and is covered in depth in Chapter 16. A second example of information gathering is provided by the **exploration** that must be undertaken by a vacuum-cleaning agent in an initially unknown environment.

Our deﬁnition requires a rational agent not only to 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 extreme cases in which the environment is completely known *a priori*. In such cases, the agent need not perceive or learn; it simply acts correctly. Of course, such agents are fragile. Consider the lowly dung beetle. After digging its nest and laying its eggs, it fetches a ball of dung from a nearby heap to plug the entrance. If the ball of dung is removed from its grasp *en route*, the beetle continues its task and pantomimes plug- ging the nest with the nonexistent dung ball, never noticing that it is missing. Evolution has built an assumption into the beetle’s behavior, and when it is violated, unsuccessful behavior results. Slightly more intelligent is the sphex wasp. The female sphex will dig a burrow, go out and sting a caterpillar and drag it to the burrow, enter the burrow again to check all is well, drag the caterpillar inside, and lay its eggs. The caterpillar serves as a food source when the eggs hatch. So far so good, but if an entomologist moves the caterpillar a few inches away while the sphex is doing the check, it will revert to the “drag” step of its plan and will continue the plan without modiﬁcation, even after dozens of caterpillar-moving interventions. The sphex is unable to learn that its innate plan is failing, and thus will not change it.

To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks **autonomy**. A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowl- edge. For example, a vacuum-cleaning agent that learns to foresee where and when additional dirt will appear will do better than one that does not. As a practical matter, one seldom re- quires complete autonomy from the start: when the agent has had little or no experience, it would have to act randomly unless the designer gave some assistance. So, just as evolution provides animals with enough built-in reﬂexes to survive long enough to learn for themselves, it would be reasonable to provide an artiﬁcial intelligent agent with some initial knowledge as well as an ability to learn. After sufﬁcient experience of its environment, the behavior of a rational agent can become effectively *independent* of its prior knowledge. Hence, the incorporation of learning allows one to design a single rational agent that will succeed in a vast variety of environments.

40 Chapter 2. Intelligent Agents

## THE NATURE OF ENVIRONMENTS

TASK ENVIRONMENT

PEAS

Now that we have a deﬁnition of rationality, we are almost ready to think about building rational agents. First, however, we must think about **task environments**, which are essen- tially the “problems” to which rational agents are the “solutions.” We begin by showing how to specify a task environment, illustrating the process with a number of examples. We then show that task environments come in a variety of ﬂavors. The ﬂavor of the task environment directly affects the appropriate design for the agent program.

# Specifying the task environment

In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent’s actuators and sensors. We group all these under the heading of the **task environment**. For the acronymically minded, we call this the **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors) description. In designing an agent, the ﬁrst step must always be to specify the task environment as fully as possible.

The vacuum world was a simple example; let us consider a more complex problem: an automated taxi driver. We should point out, before the reader becomes alarmed, that a fully automated taxi is currently somewhat beyond the capabilities of existing technology. (page 28 describes an existing driving robot.) The full driving task is extremely *open-ended*. There is no limit to the novel combinations of circumstances that can arise—another reason we chose it as a focus for discussion. Figure 2.4 summarizes the PEAS description for the taxi’s task environment. We discuss each element in more detail in the following paragraphs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Taxi driver | Safe, fast, legal, comfortable trip, | Roads, other trafﬁc, | Steering, accelerator, | Cameras, sonar, speedometer, |
|  | maximize proﬁts | pedestrians, customers | brake, signal, horn, display | GPS, odometer,  accelerometer, engine sensors, |
|  |  |  |  | keyboard |
| **Figure 2.4** PEAS description of the task environment for an automated taxi. | | | | |

First, what is the **performance measure** to which we would like our automated driver to aspire? Desirable qualities include getting to the correct destination; minimizing fuel con- sumption and wear and tear; minimizing the trip time or cost; minimizing violations of trafﬁc laws and disturbances to other drivers; maximizing safety and passenger comfort; maximiz- ing proﬁts. Obviously, some of these goals conﬂict, so tradeoffs will be required.

Next, what is the driving **environment** that the taxi will face? Any taxi driver must deal with a variety of roads, ranging from rural lanes and urban alleys to 12-lane freeways. The roads contain other trafﬁc, pedestrians, stray animals, road works, police cars, puddles,

Section 2.3. The Nature of Environments 41

SOFTWARE AGENT

and potholes. The taxi must also interact with potential and actual passengers. There are also some optional choices. The taxi might need to operate in Southern California, where snow is seldom a problem, or in Alaska, where it seldom is not. It could always be driving on the right, or we might want it to be ﬂexible enough to drive on the left when in Britain or Japan. Obviously, the more restricted the environment, the easier the design problem.

The **actuators** for an automated taxi include those available to a human driver: control over the engine through the accelerator and control over steering and braking. In addition, it will need output to a display screen or voice synthesizer to talk back to the passengers, and perhaps some way to communicate with other vehicles, politely or otherwise.

The basic **sensors** for the taxi will include one or more controllable video cameras so that it can see the road; it might augment these with infrared or sonar sensors to detect dis- tances to other cars and obstacles. To avoid speeding tickets, the taxi should have a speedome- ter, and to control the vehicle properly, especially on curves, it should have an accelerometer. To determine the mechanical state of the vehicle, it will need the usual array of engine, fuel, and electrical system sensors. Like many human drivers, it might want a global positioning system (GPS) so that it doesn’t get lost. Finally, it will need a keyboard or microphone for the passenger to request a destination.

In Figure 2.5, we have sketched the basic PEAS elements for a number of additional agent types. Further examples appear in Exercise 2.4. It may come as a surprise to some read- ers that our list of agent types includes some programs that operate in the entirely artiﬁcial environment deﬁned by keyboard input and character output on a screen. “Surely,” one might say, “this is not a real environment, is it?” In fact, what matters is not the distinction between “real” and “artiﬁcial” environments, but the complexity of the relationship among the behav- ior of the agent, the percept sequence generated by the environment, and the performance measure. Some “real” environments are actually quite simple. For example, a robot designed to inspect parts as they come by on a conveyor belt can make use of a number of simplifying assumptions: that the lighting is always just so, that the only thing on the conveyor belt will be parts of a kind that it knows about, and that only two actions (accept or reject) are possible. In contrast, some **software agents** (or software robots or **softbots**) exist in rich, unlim-

SOFTBOT ited domains. Imagine a softbot Web site operator designed to scan Internet news sources and show the interesting items to its users, while selling advertising space to generate revenue. To do well, that operator will need some natural language processing abilities, it will need to learn what each user and advertiser is interested in, and it will need to change its plans dynamically—for example, when the connection for one news source goes down or when a new one comes online. The Internet is an environment whose complexity rivals that of the physical world and whose inhabitants include many artiﬁcial and human agents.

# Properties of task environments

The range of task environments that might arise in AI is obviously vast. We can, however, identify a fairly small number of dimensions along which task environments can be catego- rized. These dimensions determine, to a large extent, the appropriate agent design and the applicability of each of the principal families of techniques for agent implementation. First,

42 Chapter 2. Intelligent Agents

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Medical diagnosis system | Healthy patient, reduced costs | Patient, hospital, staff | Display of questions, tests, diagnoses, treatments, referrals | Keyboard entry of symptoms, ﬁndings, patient’s answers |
| Satellite image analysis system | Correct image categorization | Downlink from orbiting satellite | Display of scene categorization | Color pixel arrays |
| Part-picking robot | Percentage of parts in correct bins | Conveyor belt with parts; bins | Jointed arm and hand | Camera, joint angle sensors |
| Reﬁnery controller | Purity, yield, safety | Reﬁnery, operators | Valves, pumps, heaters, displays | Temperature, pressure, chemical sensors |
| Interactive English tutor | Student’s score on test | Set of students, testing agency | Display of exercises, suggestions, corrections | Keyboard entry |
| **Figure 2.5** Examples of agent types and their PEAS descriptions. | | | | |

FULLY OBSERVABLE

PARTIALLY OBSERVABLE

UNOBSERVABLE

SINGLE AGENT MULTIAGENT

we list the dimensions, then we analyze several task environments to illustrate the ideas. The deﬁnitions here are informal; later chapters provide more precise statements and examples of each kind of environment.

**Fully observable** vs. **partially observable**: If an agent’s sensors give it access to the complete state of the environment at each point in time, then we say that the task environ- ment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are *relevant* to the choice of action; relevance, in turn, depends on the performance measure. Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world. An environment might be partially observable because of noisy and 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. If the agent has no sensors at all then the environment is **unobserv- able**. One might think that in such cases the agent’s plight is hopeless, but, as we discuss in Chapter 4, the agent’s goals may still be achievable, sometimes with certainty.

**Single agent** vs. **multiagent**: The distinction between single-agent and multiagent en-

Section 2.3. The Nature of Environments 43

COMPETITIVE

COOPERATIVE

DETERMINISTIC

vironments may seem simple enough. For example, 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. There are, however, some subtle issues. First, we have described how an entity *may* be viewed as an agent, but we have not explained which entities *must* be viewed as agents. Does an agent *A* (the taxi driver for example) have to treat an object *B* (another vehicle) as an agent, or can it be treated merely as an object behaving according to the laws of physics, analogous to waves at the beach or leaves blowing in the wind? The key distinction is whether *B*’s behavior is best described as maximizing a performance measure whose value depends on agent *A*’s behavior. For example, in chess, the opponent entity *B* is trying to maximize its performance measure, which, by the rules of chess, minimizes agent *A*’s per- formance measure. Thus, chess is a **competitive** multiagent environment. In the taxi-driving environment, on the other hand, avoiding collisions maximizes the performance measure of all agents, so it is a partially **cooperative** multiagent environment. It is also partially com- petitive because, for example, only one car can occupy a parking space. The agent-design problems in multiagent environments are often quite different from those in single-agent en- vironments; for example, **communication** often emerges as a rational behavior in multiagent environments; in some competitive environments, **randomized behavior** is rational because it avoids the pitfalls of predictability.

**Deterministic** vs. **stochastic**. If the next state of the environment is completely deter-

STOCHASTIC mined by the current state and the action executed by the agent, then we say the environment is deterministic; otherwise, it is stochastic. In principle, an agent need not worry about uncer- tainty in a fully observable, deterministic environment. (In our deﬁnition, we ignore uncer- tainty that arises purely from the actions of other agents in a multiagent environment; thus, a game can be deterministic even though each agent may be unable to predict the actions of the others.) If the environment is partially observable, however, then it could *appear* to be stochastic. Most real situations are so complex that it is impossible to keep track of all the unobserved aspects; for practical purposes, they must be treated as stochastic. Taxi driving is clearly stochastic in this sense, because one can never predict the behavior of trafﬁc exactly; moreover, one’s tires blow out and one’s engine seizes up without warning. The vacuum world as we described it is deterministic, but variations can include stochastic elements such as randomly appearing dirt and an unreliable suction mechanism (Exercise 2.13). We say an

UNCERTAIN

NONDETERMINISTIC

EPISODIC

environment is **uncertain** if it is not fully observable or not deterministic. One ﬁnal note: our use of the word “stochastic” generally implies that uncertainty about outcomes is quan- tiﬁed in terms of probabilities; a **nondeterministic** environment is one in which actions are characterized by their *possible* outcomes, but no probabilities are attached to them. Nonde- terministic environment descriptions are usually associated with performance measures that require the agent to succeed for *all possible* outcomes of its actions.

**Episodic** vs. **sequential**: In an episodic task environment, the agent’s experience is

SEQUENTIAL 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

44 Chapter 2. Intelligent Agents

STATIC

defective. In sequential environments, on the other hand, the current decision could affect all future decisions.3 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.

**Static** vs. **dynamic**: If the environment can change while an agent is deliberating, then

DYNAMIC we say the environment is dynamic for that agent; otherwise, it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. Dynamic environments, on the other hand, are continuously asking the agent what it wants to do; if it hasn’t decided yet, that counts as deciding to do nothing. If the environment itself does not change with the passage of time but the agent’s performance score does, then we say the environment is

SEMIDYNAMIC

DISCRETE

**semidynamic**. Taxi driving is clearly dynamic: the other cars and the taxi itself keep moving while the driving algorithm dithers about what to do next. Chess, when played with a clock, is semidynamic. Crossword puzzles are static.

**Discrete** vs. **continuous**: The discrete/continuous distinction applies to the *state* of the

CONTINUOUS environment, to the way *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 speak- ing, but is typically treated as representing continuously varying intensities and locations.

KNOWN

**Known** vs. **unknown**: Strictly speaking, this distinction refers not to the environment

UNKNOWN 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 stochastic) for all actions are given. Obviously, if the environment is unknown, the agent will have to learn how it works in order to make good decisions. Note that 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.

As one might expect, the hardest case is *partially observable*, *multiagent*, *stochastic*, *sequential*, *dynamic*, *continuous*, and *unknown*. Taxi driving is hard in all these senses, except that for the most part the driver’s environment is known. Driving a rented car in a new country with unfamiliar geography and trafﬁc laws is a lot more exciting.

Figure 2.6 lists the properties of a number of familiar environments. Note that the answers are not always cut and dried. For example, we describe the part-picking robot as episodic, because it normally considers each part in isolation. But if one day there is a large

3 The word “sequential” is also used in computer science as the antonym of “parallel.” The two meanings are largely unrelated.

Section 2.3. The Nature of Environments 45

ENVIRONMENT CLASS

|  |  |
| --- | --- |
| Task Environment | Observable Agents Deterministic Episodic Static Discrete |
| Crossword puzzle Chess with a clock | Fully Single Deterministic Sequential Static Discrete Fully Multi Deterministic Sequential Semi Discrete |
| Poker Backgammon | Partially Multi Stochastic Sequential Static Discrete Fully Multi Stochastic Sequential Static Discrete |
| Taxi driving Medical diagnosis | Partially Multi Stochastic Sequential Dynamic Continuous Partially Single Stochastic Sequential Dynamic Continuous |
| Image analysis Part-picking robot | Fully Single Deterministic Episodic Semi Continuous Partially Single Stochastic Episodic Dynamic Continuous |
| Reﬁnery controller Interactive English tutor | Partially Single Stochastic Sequential Dynamic Continuous Partially Multi Stochastic Sequential Dynamic Discrete |
| **Figure 2.6** Examples of task environments and their characteristics. | |

batch of defective parts, the robot should learn from several observations that the distribution of defects has changed, and should modify its behavior for subsequent parts. We have not included a “known/unknown” column because, as explained earlier, this is not strictly a prop- erty of the environment. For some environments, such as chess and poker, it is quite easy to supply the agent with full knowledge of the rules, but it is nonetheless interesting to consider how an agent might learn to play these games without such knowledge.

Several of the answers in the table depend on how the task environment is deﬁned. We have listed the medical-diagnosis task as single-agent because the disease process in a patient is not proﬁtably modeled as an agent; but a medical-diagnosis system might also have to deal with recalcitrant patients and skeptical staff, so the environment could have a multiagent aspect. Furthermore, medical diagnosis is episodic if one conceives of the task as selecting a diagnosis given a list of symptoms; the problem is sequential if the task can include proposing a series of tests, evaluating progress over the course of treatment, and so on. Also, many environments are episodic at higher levels than the agent’s individual actions. For example, a chess tournament consists of a sequence of games; each game is an episode because (by and large) the contribution of the moves in one game to the agent’s overall performance is not affected by the moves in its previous game. On the other hand, decision making within a single game is certainly sequential.

The code repository associated with this book (aima.cs.berkeley.edu) includes imple- mentations of a number of environments, together with a general-purpose environment simu- lator that places one or more agents in a simulated environment, observes their behavior over time, and evaluates them according to a given performance measure. Such experiments are often carried out not for a single environment but for many environments drawn from an **en- vironment class**. For example, to evaluate a taxi driver in simulated trafﬁc, we would want to run many simulations with different trafﬁc, lighting, and weather conditions. If we designed the agent for a single scenario, we might be able to take advantage of speciﬁc properties of the particular case but might not identify a good design for driving in general. For this

46 Chapter 2. Intelligent Agents

ENVIRONMENT GENERATOR

reason, the code repository also includes an **environment generator** for each environment class that selects particular environments (with certain likelihoods) in which to run the agent. For example, the vacuum environment generator initializes the dirt pattern and agent location randomly. We are then interested in the agent’s average performance over the environment class. A rational agent for a given environment class maximizes this average performance. Exercises 2.8 to 2.13 take you through the process of developing an environment class and evaluating various agents therein.

## THE STRUCTURE OF AGENTS

AGENT PROGRAM

ARCHITECTURE

So far we have talked about agents by describing *behavior*—the action that is performed after any given sequence of percepts. Now we must bite the bullet and talk about how the insides work. The job of AI is to design an **agent program** that implements the agent function— the mapping from percepts to actions. We assume this program will run on some sort of computing device with physical sensors and actuators—we call this the **architecture**:

*agent* = *architecture* + *program .*

Obviously, the program we choose has to be one that is appropriate for the architecture. If the program is going to recommend actions like *Walk*, the architecture had better have legs. 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 the program’s action choices to the actuators as they are generated. Most of this book is about designing agent programs, although Chapters 24 and 25 deal directly with the sensors and actuators.

# Agent programs

The agent programs that we design in this book all have the same skeleton: they take the current percept as input from the sensors and return an action to the actuators.4 Notice the difference between the agent program, which takes the current percept as input, and the agent function, which takes the entire percept history. The agent program takes just the current percept as input because nothing more is available from the environment; if the agent’s actions need to depend on the entire percept sequence, the agent will have to remember the percepts. We describe the agent programs in the simple pseudocode language that is deﬁned in Appendix B. (The online code repository contains implementations in real programming languages.) For example, Figure 2.7 shows a rather trivial agent program that keeps track of the percept sequence and then uses it to index into a table of actions to decide what to do. The table—an example of which is given for the vacuum world in Figure 2.3—represents explicitly the agent function that the agent program embodies. To build a rational agent in

4 There are other choices for the agent program skeleton; for example, we could have the agent programs be **coroutines** that run asynchronously with the environment. Each such coroutine has an input and output port and consists of a loop that reads the input port for percepts and writes actions to the output port.

Section 2.4. The Structure of Agents 47

**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.

**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*

this way, we as designers must construct a table that contains the appropriate action for every possible percept sequence.

It is instructive to consider why the table-driven approach to agent construction is doomed to failure. 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 T |P|t

t =1

entries. Consider the automated taxi: the visual input from a single camera comes in at the rate of roughly 27 megabytes per second (30 frames per second, 640 × 480 pixels with 24 bits of color information). This gives a lookup table with over 10250,000,000,000 entries for an hour’s driving. Even the lookup table for chess—a tiny, well-behaved fragment of the real world—would have at least 10150 entries. The daunting size of these tables (the number of atoms in the observable universe is less than 1080) means that (a) no physical agent in this universe will have the space to store the table, (b) the designer would not have time to create the table, (c) no agent could ever learn all the right table entries from its experience, and (d)

even if the environment is simple enough to yield a feasible table size, the designer still has no guidance about how to ﬁll in the table entries.

Despite all this, TABLE-DRIVEN-AGENT *does* do what we want: it implements the desired agent function. The key challenge for AI is to ﬁnd out how to write programs that, to the extent possible, produce rational behavior from a smallish program rather than from a vast table. We have many examples showing that this can be done successfully in other areas: for example, the huge tables of square roots used by engineers and schoolchildren prior to the 1970s have now been replaced by a ﬁve-line program for Newton’s method running on electronic calculators. The question is, can AI do for general intelligent behavior what Newton did for square roots? We believe the answer is yes.

In the remainder of this section, we outline four basic kinds of agent programs that embody the principles underlying almost all intelligent systems:

* + - * Simple reﬂex agents;
      * Model-based reﬂex agents;
      * Goal-based agents; and
      * Utility-based agents.

Each kind of agent program combines particular components in particular ways to generate actions. Section 2.4.6 explains in general terms how to convert all these agents into *learning*

48 Chapter 2. Intelligent Agents

**Figure 2.8** The agent program for a simple reﬂex agent in the two-state vacuum environ- ment. This program implements the agent function tabulated in Figure 2.3.

**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*

SIMPLE REFLEX AGENT

CONDITION–ACTION RULE

*agents* that can improve the performance of their components so as to generate better actions. Finally, Section 2.4.7 describes the variety of ways in which the components themselves can be represented within the agent. This variety provides a major organizing principle for the ﬁeld and for the book itself.

# Simple reﬂex agents

The simplest kind of agent is the **simple reﬂex agent**. These agents select actions on the basis of the *current* percept, ignoring the rest of the percept history. For example, the vacuum agent whose agent function is tabulated in Figure 2.3 is a simple reﬂex agent, because its decision is based only on the current location and on whether that location contains dirt. An agent program for this agent is shown in Figure 2.8.

Notice that the vacuum agent program is very small indeed compared to the correspond- ing table. The most obvious reduction comes from ignoring the percept history, which cuts down the number of possibilities from 4T to just 4. A further, small reduction comes from the fact that when the current square is dirty, the action does not depend on the location.

Simple reﬂex behaviors occur even in more complex environments. Imagine yourself as the driver of the automated taxi. If the car in front brakes and its brake lights come on, then you should notice this and initiate braking. In other words, some processing is done on the visual input to establish the condition we call “The car in front is braking.” Then, this triggers some established connection in the agent program to the action “initiate braking.” We call such a connection a **condition–action rule**,5 written as

**if** *car-in-front-is-braking* **then** *initiate-braking*.

Humans also have many such connections, some of which are learned responses (as for driv- ing) and some of which are innate reﬂexes (such as blinking when something approaches the eye). In the course of the book, we show several different ways in which such connections can be learned and implemented.

The program in Figure 2.8 is speciﬁc to one particular vacuum environment. A more general and ﬂexible approach is ﬁrst to build a general-purpose interpreter for condition– action rules and then to create rule sets for speciﬁc task environments. Figure 2.9 gives the structure of this general program in schematic form, showing how the condition–action rules allow the agent to make the connection from percept to action. (Do not worry if this seems

5 Also called **situation–action rules**, **productions**, or **if–then rules**.

Section 2.4. The Structure of Agents 49

**Figure 2.9** Schematic diagram of a simple reﬂex agent.

Actuators

Condition-actio~~n rules~~

Sensors

Agent

What action I should do now

What the world is like now

Environment

**Figure 2.10** A simple reﬂex agent. It acts according to a rule whose condition matches the current state, as deﬁned by the percept.

**function** SIMPLE-REFLEX-AGENT( *percept* ) **returns** an action

**persistent**: *rules*, a set of condition–action rules

*state* ← INTERPRET-INPUT( *percept* ) *rule* ← RULE-MATCH(*state*, *rules*) *action* ← *rule*.ACTION

**return** *action*

trivial; it gets more interesting shortly.) 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. The agent program, which is also very simple, is shown in Figure 2.10. The INTERPRET-INPUT function generates an abstracted description of the current state from the percept, and the RULE-MATCH function returns the ﬁrst rule in the set of rules that matches the given state description. Note that the description in terms of “rules” and “matching” is purely conceptual; actual implementations can be as simple as a collection of logic gates implementing a Boolean circuit.

Simple reﬂex agents have the admirable property of being simple, but they turn out to be of limited intelligence. The agent in Figure 2.10 will work *only if the correct decision can be made on the basis of only the current percept—that is, only if the environment is fully observ- able.* Even a little bit of unobservability can cause serious trouble. For example, the braking rule given earlier assumes that the condition *car-in-front-is-braking* can be determined from the current percept—a single frame of video. This works if the car in front has a centrally mounted brake light. Unfortunately, older models have different conﬁgurations of taillights,



50 Chapter 2. Intelligent Agents

RANDOMIZATION

INTERNAL STATE

MODEL-BASED AGENT

brake lights, and turn-signal lights, and it is not always possible to tell from a single image whether the car is braking. A simple reﬂex agent driving behind such a car would either brake continuously and unnecessarily, or, worse, never brake at all.

We can see a similar problem arising in the vacuum world. Suppose that a simple reﬂex vacuum agent is deprived of its location sensor and has only a dirt sensor. Such an agent has just two possible percepts: [*Dirty* ] and [*Clean*]. It can *Suck* in response to [*Dirty*]; what should it do in response to [*Clean*]? Moving *Left* fails (forever) if it happens to start in square *A*, and moving *Right* fails (forever) if it happens to start in square *B*. Inﬁnite loops are often unavoidable for simple reﬂex agents operating in partially observable environments.

Escape from inﬁnite loops is possible if the agent can **randomize** its actions. For ex- ample, if the vacuum agent perceives [*Clean* ], it might ﬂip a coin to choose between *Left* and *Right* . It is easy to show that the agent will reach the other square in an average of two steps. Then, if that square is dirty, the agent will clean it and the task will be complete. Hence, a randomized simple reﬂex agent might outperform a deterministic simple reﬂex agent.

We mentioned in Section 2.3 that randomized behavior of the right kind can be rational in some multiagent environments. In single-agent environments, randomization is usually *not* rational. It is a useful trick that helps a simple reﬂex agent in some situations, but in most cases we can do much better with more sophisticated deterministic agents.

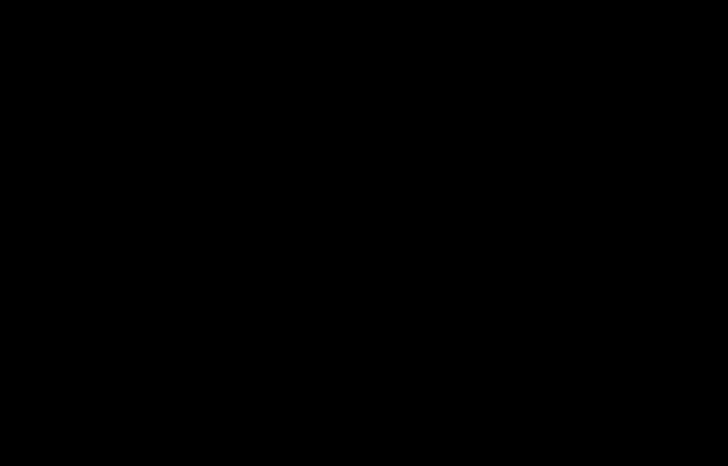
# Model-based reﬂex agents

The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can’t see now*. That is, the agent should maintain some sort of **internal state** that depends on the percept history and thereby reﬂects at least some of the unobserved aspects of the current state. For the braking problem, the internal state is not too extensive— just the previous frame from the camera, allowing the agent to detect when two red lights at the edge of the vehicle go on or off simultaneously. For other driving tasks 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. And for any driving to be possible at all, the agent needs to keep track of where its keys are.

Updating this internal state information as time goes by requires two kinds of knowl- edge to be encoded in the agent program. First, we need some information about how the world evolves independently of the agent—for example, that an overtaking car generally will be closer behind than it was a moment ago. Second, we need some information about how the agent’s own actions affect the world—for example, that when the agent turns the steering wheel clockwise, the car turns to the right, or that after driving for ﬁve minutes northbound on the freeway, one is usually about ﬁve miles north of where one was ﬁve minutes ago. This knowledge about “how the world works”—whether implemented in simple Boolean circuits or in complete scientiﬁc theories—is called a **model** of the world. An agent that uses such a model is called a **model-based agent**.

Figure 2.11 gives the structure of the model-based reﬂex agent with internal state, show- ing how the current percept is combined with the old internal state to generate the updated description of the current state, based on the agent’s model of how the world works. The agent program is shown in Figure 2.12. The interesting part is the function UPDATE-STATE , which

Section 2.4. The Structure of Agents 51



**Figure 2.11** A model-based reﬂex agent.

Actuators

Agent

What action I should do now

Condition-action rules

What my actions do

What the world is like now

State

How the world evolves

Sensors

Environment

**Figure 2.12** A model-based reﬂex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reﬂex agent.

**function** MODEL-BASED-REFLEX-AGENT( *percept* ) **returns** an action

**persistent**: *state*, the agent’s current conception of the world state

*model* , a description of how the next state depends on current state and action

*rules*, a set of condition–action rules

*action*, the most recent action, initially none

*state* ← UPDATE-STATE(*state*, *action* , *percept* , *model* ) *rule* ← RULE-MATCH(*state*, *rules*)

*action* ← *rule*.ACTION

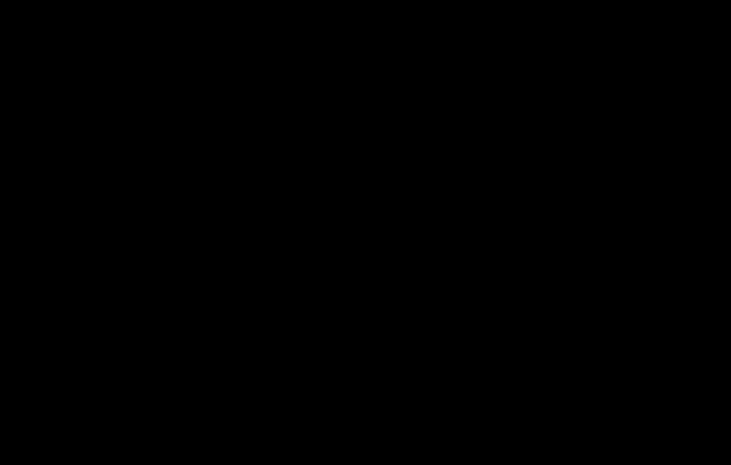
**return** *action*

is responsible for creating the new internal state description. The details of how models and states are represented vary widely depending on the type of environment and the particular technology used in the agent design. Detailed examples of models and updating algorithms appear in Chapters 4, 12, 11, 15, 17, and 25.

Regardless of the kind of representation used, it is seldom possible for the agent to determine the current state of a partially observable environment *exactly*. Instead, the box labeled “what the world is like now” (Figure 2.11) represents the agent’s “best guess” (or sometimes best guesses). For example, an automated taxi may not be able to see around the large truck that has stopped in front of it and can only guess about what may be causing the hold-up. Thus, uncertainty about the current state may be unavoidable, but the agent still has to make a decision.

A perhaps less obvious point about the internal “state” maintained by a model-based agent is that it does not have to describe “what the world is like now” in a literal sense. For

52 Chapter 2. Intelligent Agents



**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.

Actuators

Agent

What action I should do now

Goals

What it will be like if I do action *A*

What my actions do

What the world is like now

State

How the world evolves

Sensors

Environment

example, the taxi may be driving back home, and it may have a rule telling it to ﬁll up with gas on the way home unless it has at least half a tank. Although “driving back home” may *seem* to an aspect of the world state, the fact of the taxi’s *destination* is actually an aspect of the agent’s internal state. If you ﬁnd this puzzling, consider that the taxi could be in exactly the same place at the same time, but intending to reach a different destination.

GOAL

# Goal-based agents

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 where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable—for example, being at the passenger’s destination. The agent program can combine this with the model (the same information as was used in the model- based reﬂex agent) to choose actions that achieve the goal. Figure 2.13 shows the goal-based agent’s structure.

Sometimes goal-based action selection is straightforward—for example, when goal sat- isfaction results immediately from a single action. Sometimes it will be more tricky—for example, when the agent has to consider long sequences of twists and turns in order to ﬁnd a way to achieve the goal. **Search** (Chapters 3 to 5) and **planning** (Chapters 10 and 11) are the subﬁelds of AI devoted to ﬁnding action sequences that achieve the agent’s goals.

Notice that decision making of this kind is fundamentally different from the condition– action rules described earlier, in that it involves consideration of the future—both “What will happen if I do such-and-such?” and “Will that make me happy?” In the reﬂex agent designs, this information is not explicitly represented, because the built-in rules map directly from

Section 2.4. The Structure of Agents 53

UTILITY

UTILITY FUNCTION

EXPECTED UTILITY

percepts to actions. The reﬂex agent brakes when it sees brake lights. A goal-based agent, in principle, could reason that if the car in front has its brake lights on, it will slow down. Given the way the world usually evolves, the only action that will achieve the goal of not hitting other cars is to brake.

Although the goal-based agent appears less efﬁcient, it is more ﬂexible because the knowledge that supports its decisions is represented explicitly and can be modiﬁed. If it starts to rain, the agent can update its knowledge of how effectively its brakes will operate; this will automatically cause all of the relevant behaviors to be altered to suit the new conditions. For the reﬂex agent, on the other hand, we would have to rewrite many condition–action rules. The goal-based agent’s behavior can easily be changed to go to a different destination, simply by specifying that destination as the goal. The reﬂex agent’s rules for when to turn and when to go straight will work only for a single destination; they must all be replaced to go somewhere new.

# Utility-based agents

Goals alone are not enough to generate high-quality behavior in most environments. For example, 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 world states according to exactly how happy they would make the agent. Because “happy” does not sound very scientiﬁc, economists and computer scientists use the term **utility** instead.6

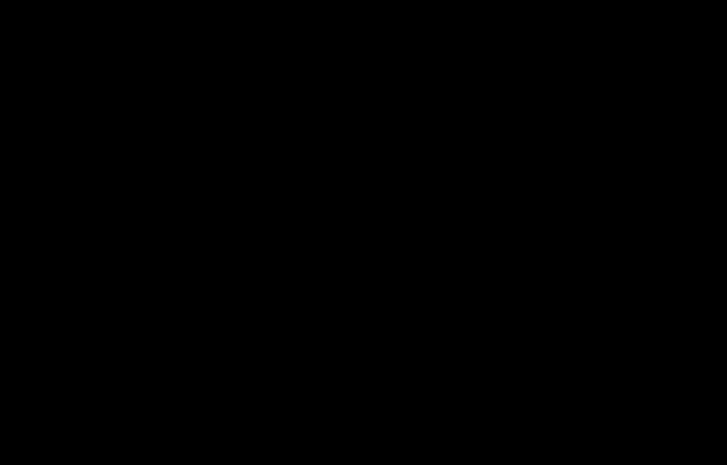
We have already seen that a performance measure assigns a score to any given sequence of environment states, so it can easily distinguish between more and less desirable ways of getting to the taxi’s destination. An agent’s **utility function** is essentially an internalization of the performance measure. If the internal utility function and the external performance measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure.

Let us emphasize again that this is not the *only* way to be rational—we have already seen a rational agent program for the vacuum world (Figure 2.8) that has no idea what its utility function is—but, like goal-based agents, a utility-based agent has many advantages in terms of ﬂexibility and learning. Furthermore, in two kinds of cases, goals are inadequate but a utility-based agent can still make rational decisions. First, when there are conﬂicting goals, only some of which can be achieved (for example, speed and safety), the utility function speciﬁes the appropriate tradeoff. Second, when there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed against the importance of the goals.

Partial observability and stochasticity are ubiquitous in the real world, and so, therefore, is decision making under uncertainty. Technically speaking, a rational utility-based agent chooses the action that maximizes the **expected utility** of the action outcomes—that is, the utility the agent expects to derive, on average, given the probabilities and utilities of each

6 The word “utility” here refers to “the quality of being useful,” not to the electric company or waterworks.

54 Chapter 2. Intelligent Agents



**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.

Actuators

Agent

How happy I will be in such a state

What action I should do now

Utility

What it will be like if I do action *A*

What my actions do

What the world is like now

State

How the world evolves

Sensors

Environment

outcome. (Appendix A deﬁnes expectation more precisely.) In Chapter 16, we show that any rational agent must behave *as if* it possesses a utility function whose expected value it tries to maximize. An agent that possesses an *explicit* utility function can make rational decisions with a general-purpose algorithm that does not depend on the speciﬁc utility function being maximized. In this way, the “global” deﬁnition of rationality—designating as rational those agent functions that have the highest performance—is turned into a “local” constraint on rational-agent designs that can be expressed in a simple program.

The utility-based agent structure appears in Figure 2.14. Utility-based agent programs appear in Part IV, where we design decision-making agents that must handle the uncertainty inherent in stochastic or partially observable environments.

At this point, the reader may be wondering, “Is it that simple? We just build agents that maximize expected utility, and we’re done?” It’s true that such agents would be intelligent, but it’s not simple. A utility-based agent has to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning. The results of this research ﬁll many of the chapters of this book. Choosing the utility-maximizing course of action is also a difﬁcult task, requiring ingenious algorithms that ﬁll several more chapters. Even with these algorithms, perfect rationality is usually unachievable in practice because of computational complexity, as we noted in Chapter 1.

# Learning agents

We have described agent programs with various methods for selecting actions. We have not, so far, explained how the agent programs *come into being*. In his famous early paper, Turing (1950) considers the idea of actually programming his intelligent machines by hand.

Section 2.4. The Structure of Agents 55

**Figure 2.15** A general learning agent.

Actuators

Agent

learning

goals

changes

feedback

Sensors

Performance standard

Problem generator

knowledge

Performance element

Learning element

Critic

Environment

LEARNING ELEMENT

PERFORMANCE ELEMENT

CRITIC

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. Learning has another advantage, as we noted earlier: it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow. In this section, we brieﬂy introduce the main ideas of learning agents. Throughout the book, we comment on opportunities and methods for learning in particular kinds of agents. Part V goes into much more depth on the learning algorithms themselves.

A learning agent can be divided into four conceptual components, as shown in Fig- ure 2.15. The most important distinction is between the **learning element**, which is re- sponsible for making improvements, and the **performance element**, which is responsible for selecting external actions. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions. The learning element uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modiﬁed to do better in the future.

The design of the learning element depends very much on the design of the performance element. When trying to design an agent that learns a certain capability, the ﬁrst question is not “How am I going to get it to learn this?” but “What kind of performance element will my agent need to do this once it has learned how?” Given an agent design, learning mechanisms can be constructed to improve every part of the agent.

The critic tells the learning element how well the agent is doing with respect to a ﬁxed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent’s success. For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so. It is important that the performance

56 Chapter 2. Intelligent Agents

PROBLEM GENERATOR

standard be ﬁxed. Conceptually, one should think of it as being outside the agent altogether because the agent must not modify it to ﬁt its own behavior.

The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and informative experiences. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. But if the agent is willing to explore a little and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run. The problem generator’s job is to suggest these exploratory actions. This is what scientists do when they carry out experiments. Galileo did not think that dropping rocks from the top of a tower in Pisa was valuable in itself. He was not trying to break the rocks or to modify the brains of unfortunate passers-by. His aim was to modify his own brain by identifying a better theory of the motion of objects.

To make the overall design more concrete, let us return to the automated taxi example. The performance element consists of whatever collection of knowledge and procedures the taxi has for selecting its driving actions. The taxi goes out on the road and drives, using this performance element. The critic observes the world and passes information along to the learning element. For example, after the taxi makes a quick left turn across three lanes of traf- ﬁc, the critic observes the shocking language used by other drivers. From this experience, the learning element is able to formulate a rule saying this was a bad action, and the performance element is modiﬁed by installation of the new rule. The problem generator might identify certain areas of behavior in need of improvement and suggest experiments, such as trying out the brakes on different road surfaces under different conditions.

The learning element can make changes to any of the “knowledge” components shown in the agent diagrams (Figures 2.9, 2.11, 2.13, and 2.14). The simplest cases involve learning directly from the percept sequence. Observation of pairs of successive states of the environ- ment can allow the agent to learn “How the world evolves,” and observation of the results of its actions can allow the agent to learn “What my actions do.” For example, if the taxi exerts a certain braking pressure when driving on a wet road, then it will soon ﬁnd out how much deceleration is actually achieved. Clearly, these two learning tasks are more difﬁcult if the environment is only partially observable.

The forms of learning in the preceding paragraph do not need to access the external performance standard—in a sense, the standard is the universal one of making predictions that agree with experiment. The situation is slightly more complex for a utility-based agent that wishes to learn utility information. For example, suppose the taxi-driving agent 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. Hard-wired performance standards such as pain and hunger in animals can be understood in this way. This issue is discussed further in Chapter 21.

In summary, agents have a variety of components, and those components can be repre- sented in many ways within the agent program, so there appears to be great variety among

Section 2.4. The Structure of Agents 57

learning methods. There is, however, a single unifying theme. Learning in intelligent agents can be summarized as a process of modiﬁcation of each component of the agent to bring the components into closer agreement with the available feedback information, thereby improv- ing the overall performance of the agent.

# How the components of agent programs work

We have described agent programs (in very high-level terms) as consisting of various compo- nents, whose function it is to answer questions such as: “What is the world like now?” “What action should I do now?” “What do my actions do?” The next question for a student of AI is, “How on earth do these components work?” It takes about a thousand pages to begin to answer that question properly, but here we want to draw the reader’s attention to some basic distinctions among the various ways that the components can represent the environment that the agent inhabits.

Roughly speaking, we can place the representations along an axis of increasing com- plexity and expressive power—**atomic**, **factored**, and **structured**. To illustrate these ideas, it helps to consider a particular agent component, such as the one that deals with “What my actions do.” This component describes the changes that might occur in the environment as the result of taking an action, and Figure 2.16 provides schematic depictions of how those transitions might be represented.



**Figure 2.16** Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real- valued, or one of a ﬁxed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

(b) Structured

(b) Factored

(a) Atomic

B

B

C

C

ATOMIC REPRESENTATION

In an **atomic representation** each state of the world is indivisible—it has no internal structure. Consider the problem of ﬁnding a driving route from one end of a country to the other via some sequence of cities (we address this problem in Figure 3.2 on page 68). For the purposes of solving this problem, it may sufﬁce to reduce the state of world to just the name of the city we are in—a single atom of knowledge; a “black box” whose only discernible property is that of being identical to or different from another black box. The algorithms

58 Chapter 2. Intelligent Agents

FACTORED REPRESENTATION

VARIABLE ATTRIBUTE VALUE

STRUCTURED REPRESENTATION

EXPRESSIVENESS

underlying **search** and **game-playing** (Chapters 3–5), **Hidden Markov models** (Chapter 15), and **Markov decision processes** (Chapter 17) all work with atomic representations—or, at least, they treat representations *as if* they were atomic.

Now consider a higher-ﬁdelity description for the same problem, where we need to be concerned with more than just atomic location in one city or another; we might need to pay attention to how much gas is in the tank, our current GPS coordinates, whether or not the oil warning light is working, how much spare change we have for toll crossings, what station is on the radio, and so on. A **factored representation** splits up each state into a ﬁxed set of **variables** or **attributes**, each of which can have a **value**. While two different atomic states have nothing in common—they are just different black boxes—two different factored states can share some attributes (such as being at some particular GPS location) and not others (such as having lots of gas or having no gas); this makes it much easier to work out how to turn one state into another. With factored representations, we can also represent *uncertainty*—for example, ignorance about the amount of gas in the tank can be represented by leaving that attribute blank. Many important areas of AI are based on factored representations, including **constraint satisfaction** algorithms (Chapter 6), **propositional logic** (Chapter 7), **planning** (Chapters 10 and 11), **Bayesian networks** (Chapters 13–16), and the **machine learning** al- gorithms in Chapters 18, 20, and 21.

For many purposes, we need to understand the world as having *things* in it that are *related* to each other, not just variables with values. For example, we might notice that a large truck ahead of us is reversing into the driveway of a dairy farm but a cow has got loose and is blocking the truck’s path. A factored representation is unlikely to be pre-equipped with the attribute *TruckAheadBackingIntoDairyFarmDrivewayBlockedByLooseCow* with

value *true* or *false*. Instead, we would need a **structured representation**, in which ob-

jects such as cows and trucks and their various and varying relationships can be described explicitly. (See Figure 2.16(c).) Structured representations underlie **relational databases** and **ﬁrst-order logic** (Chapters 8, 9, and 12), **ﬁrst-order probability models** (Chapter 14), **knowledge-based learning** (Chapter 19) and much of **natural language understanding** (Chapters 22 and 23). In fact, almost everything that humans express in natural language concerns objects and their relationships.

As we mentioned earlier, the axis along which atomic, factored, and structured repre- sentations lie is the axis of increasing **expressiveness**. Roughly speaking, a more expressive representation can capture, at least as concisely, everything a less expressive one can capture, plus some more. Often, the more expressive language is *much* more concise; for example, the rules of chess can be written in a page or two of a structured-representation language such as ﬁrst-order logic but require thousands of pages when written in a factored-representation language such as propositional logic. On the other hand, reasoning and learning become more complex as the expressive power of the representation increases. To gain the beneﬁts of expressive representations while avoiding their drawbacks, intelligent systems for the real world may need to operate at all points along the axis simultaneously.

Section 2.5. Summary 59

## SUMMARY

This chapter has been something of a whirlwind tour of AI, which we have conceived of as the science of agent design. The major points to recall are as follows:

* An **agent** is something that perceives and acts in an environment. The **agent function**

for an agent speciﬁes the action taken by the agent in response to any percept sequence.

* The **performance measure** evaluates the behavior of the agent in an environment. A **rational agent** acts so as to maximize the expected value of the performance measure, given the percept sequence it has seen so far.
* A **task environment** speciﬁcation includes the performance measure, the external en- vironment, the actuators, and the sensors. In designing an agent, the ﬁrst step must always be to specify the task environment as fully as possible.
* Task environments vary along several signiﬁcant dimensions. They can be fully or partially observable, single-agent or multiagent, deterministic or stochastic, episodic or sequential, static or dynamic, discrete or continuous, and known or unknown.
* The **agent program** implements the agent function. There exists a variety of basic agent-program designs reﬂecting the kind of information made explicit and used in the decision process. The designs vary in efﬁciency, compactness, and ﬂexibility. The appropriate design of the agent program depends on the nature of the environment.
* **Simple reﬂex agents** respond directly to percepts, whereas **model-based reﬂex agents** maintain internal state to track aspects of the world that are not evident in the current percept. **Goal-based agents** act to achieve their goals, and **utility-based agents** try to maximize their own expected “happiness.”
* All agents can improve their performance through **learning**.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

CONTROLLER

The central role of action in intelligence—the notion of practical reasoning—goes back at least as far as Aristotle’s *Nicomachean Ethics*. Practical reasoning was also the subject of McCarthy’s (1958) inﬂuential paper “Programs with Common Sense.” The ﬁelds of robotics and control theory are, by their very nature, concerned principally with physical agents. The concept of a **controller** in control theory is identical to that of an agent in AI. Perhaps sur- prisingly, AI has concentrated for most of its history on isolated components of agents— question-answering systems, theorem-provers, vision systems, and so on—rather than on whole agents. The discussion of agents in the text by Genesereth and Nilsson (1987) was an inﬂuential exception. The whole-agent view is now widely accepted and is a central theme in recent texts (Poole *et al.*, 1998; Nilsson, 1998; Padgham and Winikoff, 2004; Jones, 2007).

Chapter 1 traced the roots of the concept of rationality in philosophy and economics. In AI, the concept was of peripheral interest until the mid-1980s, when it began to suffuse many

60 Chapter 2. Intelligent Agents

AUTONOMIC COMPUTING

MULTIAGENT SYSTEMS

discussions about the proper technical foundations of the ﬁeld. A paper by Jon Doyle (1983) predicted that rational agent design would come to be seen as the core mission of AI, while other popular topics would spin off to form new disciplines.

Careful attention to the properties of the environment and their consequences for ra- tional agent design is most apparent in the control theory tradition—for example, classical control systems (Dorf and Bishop, 2004; Kirk, 2004) handle fully observable, deterministic environments; stochastic optimal control (Kumar and Varaiya, 1986; Bertsekas and Shreve, 2007) handles partially observable, stochastic environments; and hybrid control (Henzinger and Sastry, 1998; Cassandras and Lygeros, 2006) deals with environments containing both discrete and continuous elements. The distinction between fully and partially observable en- vironments is also central in the **dynamic programming** literature developed in the ﬁeld of operations research (Puterman, 1994), which we discuss in Chapter 17.

Reﬂex agents were the primary model for psychological behaviorists such as Skinner (1953), who attempted to reduce the psychology of organisms strictly to input/output or stim- ulus/response mappings. The advance from behaviorism to functionalism in psychology, which was at least partly driven by the application of the computer metaphor to agents (Put- nam, 1960; Lewis, 1966), introduced the internal state of the agent into the picture. Most work in AI views the idea of pure reﬂex agents with state as too simple to provide much leverage, but work by Rosenschein (1985) and Brooks (1986) questioned this assumption (see Chapter 25). In recent years, a great deal of work has gone into ﬁnding efﬁcient algo- rithms for keeping track of complex environments (Hamscher *et al.*, 1992; Simon, 2006). The Remote Agent program (described on page 28) that controlled the Deep Space One spacecraft is a particularly impressive example (Muscettola *et al.*, 1998; Jonsson *et al.*, 2000).

Goal-based agents are presupposed in everything from Aristotle’s view of practical rea- soning to McCarthy’s early papers on logical AI. Shakey the Robot (Fikes and Nilsson, 1971; Nilsson, 1984) was the ﬁrst robotic embodiment of a logical, goal-based agent. A full logical analysis of goal-based agents appeared in Genesereth and Nilsson (1987), and a goal-based programming methodology called agent-oriented programming was developed by Shoham (1993). The agent-based approach is now extremely popular in software engineer- ing (Ciancarini and Wooldridge, 2001). It has also inﬁltrated the area of operating systems, where **autonomic computing** refers to computer systems and networks that monitor and con- trol themselves with a perceive–act loop and machine learning methods (Kephart and Chess, 2003). Noting that a collection of agent programs designed to work well together in a true multiagent environment necessarily exhibits modularity—the programs share no internal state and communicate with each other only through the environment—it is common within the ﬁeld of **multiagent systems** to design the agent program of a single agent as a collection of autonomous sub-agents. In some cases, one can even prove that the resulting system gives the same optimal solutions as a monolithic design.

The goal-based view of agents also dominates the cognitive psychology tradition in the area of problem solving, beginning with the enormously inﬂuential *Human Problem Solv- ing* (Newell and Simon, 1972) and running through all of Newell’s later work (Newell, 1990). Goals, further analyzed as *desires* (general) and *intentions* (currently pursued), are central to the theory of agents developed by Bratman (1987). This theory has been inﬂuential both in

Exercises 61

natural language understanding and multiagent systems.

Horvitz *et al.* (1988) speciﬁcally suggest the use of rationality conceived as the maxi- mization of expected utility as a basis for AI. The text by Pearl (1988) was the ﬁrst in AI to cover probability and utility theory in depth; its exposition of practical methods for reasoning and decision making under uncertainty was probably the single biggest factor in the rapid shift towards utility-based agents in the 1990s (see Part IV).

The general design for learning agents portrayed in Figure 2.15 is classic in the machine learning literature (Buchanan *et al.*, 1978; Mitchell, 1997). Examples of the design, as em- bodied in programs, go back at least as far as Arthur Samuel’s (1959, 1967) learning program for playing checkers. Learning agents are discussed in depth in Part V.

Interest in agents and in agent design has risen rapidly in recent years, partly because of the growth of the Internet and the perceived need for automated and mobile **softbot** (Etzioni and Weld, 1994). Relevant papers are collected in *Readings in Agents* (Huhns and Singh, 1998) and *Foundations of Rational Agency* (Wooldridge and Rao, 1999). Texts on multiagent systems usually provide a good introduction to many aspects of agent design (Weiss, 2000a; Wooldridge, 2002). Several conference series devoted to agents began in the 1990s, including the International Workshop on Agent Theories, Architectures, and Languages (ATAL), the International Conference on Autonomous Agents (AGENTS), and the International Confer- ence on Multi-Agent Systems (ICMAS). In 2002, these three merged to form the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). The journal *Autonomous Agents and Multi-Agent Systems* was founded in 1998. Finally, *Dung Beetle Ecology* (Hanski and Cambefort, 1991) provides a wealth of interesting information on the behavior of dung beetles. YouTube features inspiring video recordings of their activities.

EXERCISES

* 1. Suppose that the performance measure is concerned with just the ﬁrst *T* time steps of the environment and ignores everything thereafter. Show that a rational agent’s action may depend not just on the state of the environment but also on the time step it has reached.
  2. Let us examine the rationality of various vacuum-cleaner agent functions.
     1. Show that the simple vacuum-cleaner agent function described in Figure 2.3 is indeed rational under the assumptions listed on page 38.
     2. Describe a rational agent function for the case in which each movement costs one point. Does the corresponding agent program require internal state?
     3. Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from its experience in these cases? If so, what should it learn? If not, why not?
  3. For each of the following assertions, say whether it is true or false and support your answer with examples or counterexamples where appropriate.

**a**. An agent that senses only partial information about the state cannot be perfectly rational.