CHAPTER 19

Intelligent Agents

An active line on a walk, moving freely without a goal. A walk for a walk's sake. The agent is a point that shifts position.

—Paul Klee, *Pedagogical Sketchbook*

My team had written a number of programs to control swarms of agents. These programs were modeled on behavior of bees. The programs had many useful characteristics. Because swarms were composed of many agents, the swarm could respond to the environment in a robust way. Faced with new and unexpected conditions, the swarm programs didn't crash; they just sort of flowed around the obstacles, and kept going.

—Michael Crichton, *Prey*

For I also am a man set under authority, having under me soldiers, and I say unto one, Go, and he goeth; and to another, Come, and he cometh; and to my servant, Do this, and he doeth it.

—The Gospel according to St Luke, Chapter 7, Verse 8

19.1 Introduction

An **agent** is an entity that is able to carry out some task, usually to help a human user. Agents can be biologic (people or animals, for example), robotic, or computational. This chapter is primarily concerned with the latter type, in particular with **software agents**. A software agent is a computer program designed to carry out some task on behalf of a user.

As we will see, there are a number of ways in which software agents can be built and a number of properties that they can have. One property with which we are particularly concerned is **intelligence**. We will discuss in more detail what is meant by intelligence, in the context of agents, in Section 19.2.1.

This chapter also introduces other important properties that agents may or may not have, including **autonomy**, **benevolence**, the ability to **collaborate** (with other agents, for example), and the ability to learn.

A number of architectures that can be used to build agents are discussed.

This chapter also introduces a number of types of agents, such as **reactive agents**, **interface agents**, **information agents**, and **multiagent systems**, which use a number of agents together to solve a single problem.

Finally, the chapter briefly introduces the ideas behind robotic agents and discusses a particular type of robot, known as a Braitenberg vehicle, which is used to discuss the nature of intelligence and our interpretation of behavior.

In many ways, the field of Artificial Intelligence as a whole can be seen as the study of methods that can be used to build intelligent agents. For example, the techniques discussed in Chapters 3 through 6 can be thought of as methods that intelligent agents can use to enable them to search or to play games. Each of the methods explained in this book can be used by an intelligent agent or to build intelligent agent systems.

19.2 Properties of Agents

19.2.1 Intelligence

An agent is a tool that carries out some task or tasks on behalf of a human. For example, a simple agent might be set up to buy a particular stock when its price fell below a particular level. A simple Internet search agent might be designed to send queries to a number of search engines and collate the results.

Intelligent agents have additional domain knowledge that enables them to carry out their tasks even when the parameters of the task change or when unexpected situations arise. For example, an intelligent agent might be designed to buy books for a user on the Internet at the lowest possible price. The agent would need to be able to interact with a set of online bookstores but would also need to be able to learn how to deal with new bookstores or with individuals who were offering secondhand books. These

kinds of agents that perform tasks on behalf of people are called **interface agents**, which are discussed in Section 19.5.

Many intelligent agents are able to learn, from their own performance, from other agents, from the user, or from the environment in which they are situated. The ways in which agents can learn have been covered in some detail in Part 4 of this book, and the way in which some of these ideas can be applied by intelligent agents are introduced in Section 19.12.

19.2.2 Autonomy

In addition to intelligence, an important feature of many intelligent agents is **autonomy**—the ability to act and make decisions independently of the programmer or user of the agent. For example, an intelligent buying agent that is designed to buy goods on behalf of a user needs to be able to make decisions about what items to purchase without checking back with the user. This autonomy is what sets intelligent agents aside from many other Artificial Intelligence techniques.

19.2.3 Ability to Learn

Many agents have an ability to learn. In other words, when presented with new information, such an agent is able to store that new information in a useful form. For example, agents can learn from a user by observing actions or by being given instruction. We see how interface agents use these kinds of learning in Section 19.5. Agents can also learn from other agents in **multiagent systems**, which are described in Section 19.8.

Learning allows agents to improve their performance at carrying out a particular task over time. If a human user tells an agent that it has carried out a task poorly, it is useful for that agent to be able to learn from this experience to avoid making the same mistakes in the future.

19.2.4 Cooperation

In multiagent systems, agents usually **cooperate** with each other. This cooperation implies some form of **social interaction** between agents. For example, a buying agent may negotiate with selling agents to make purchases. As has been mentioned, agents can also learn from each other. To use the buying agent example again, a buying agent may be informed by another buying agent of a new shopping portal that the agent may find useful.

Of course, it is also useful for agents to cooperate with the humans who use them. Although in most agent systems, this cooperation is in the form of simple inputs and instructions, the manner in which agents cooperate with people can be very important, as we see in Section 19.5 when we discuss interface agents.

19.2.5 Other Agent Properties

Agents can have a number of other properties. A **versatile** agent is one that is able to carry out many different tasks. Most agents are **benevolent**, but some can be **competitive** or **nonhelpful**. Similarly, agents may be **altruistic** or **antagonistic**. Some agents can have the ability to lie to other agents, or to users, whereas other agents are always truthful (this property is known as **veracity**).

Other properties of agents include the extent to which they can be trusted with delegated tasks and whether or not they **degrade gracefully** (i.e., when the agent encounters a new problem that it is unable to solve, does it fail completely, or is it able to make some progress?).

An agent's **mobility** is defined by its ability to move about on the Internet or another network.

19.3 Agent Classifications

As has been discussed in Section 19.2, agents can be classified according to a number of parameters. We will now discuss a variety of types of agents that are classified according to these, and other, parameters.

The types of agents that we will look at are not mutually exclusive: an interface agent can be reactive or utility based. It can also be versatile or nonversatile.

The main classes of agents are defined as follows:

- reactive agents
- collaborative agents
- interface agents
- mobile agents
- information-gathering agents

We also look at the difference between reactive agents and goal-based and utility-based agents, which are defined by the ways in which they are motivated. Reactive agents simply respond to inputs they receive, whereas goal-based and utility-based agents have an ability to reason about their positions and make decisions on the basis of that reasoning.

Some agents are **hybrids**, which exhibit properties of more than one of the categories listed above. The eventual aim of most intelligent agent research is to develop **smart agents**, which would be fully autonomous and able to learn and cooperate with other agents. Smart agents do not yet exist and are not covered by this book.

19.4 Reactive Agents

A simple **reactive agent** (also known as a **reflex agent**) is a production system where inputs from the environment are compared with rules to determine which actions to carry out. In other words, reactive agents simply react to events in their environment according to predetermined rules.

A simple example of a reactive agent is the automatic mail filter that many e-mail systems now possess. This mail filter examines each e-mail as it arrives and compares it against a set of rules, or templates, and classifies it accordingly. A common use for such systems is to reject so-called "junk mail" or "spam." More complex systems are used to route e-mails within an organization, so that a consumer can send an e-mail to a central mail address, and the system will determine to which department within the company to send the mail, based on its contents.

In the case of the e-mail–filtering agent, the environment is simply an e-mail inbox and the contents of that inbox.

A reactive agent does not tend to perform well when its environment changes or when something happens that it has not been told about. For example, an e-mail–filtering system might have problems when it receives an e-mail that is entirely in Chinese. New rules can of course be written to deal with such situations, but it might be more desirable to have an agent that can learn to adapt to new situations.

A more complex reactive agent can be developed that combines inputs from its environment with information about the state of the world and information about how its actions affect the world.

Hence, a scheduling system might be based on the e-mail–filtering agent system, which assigns tasks to employees based on the content of e-mails as they arrive.

For example, when an e-mail arrives from a customer, reporting a bug in the company's software system, the agent might assign a task to the engineering department to fix the bug. The agent would then wait for further information from the engineering department. If it did not receive assurance that the bug had been fixed within a reasonable amount of time, it might contact the engineering department again. The agent's ability to do this derives from the fact that it is able to store information about the state of the world (such as "engineering department working to fix bug number 36,234,120") and about how its actions affect the state of the world (such as "when I send this e-mail to engineering, they will start to work on fixing the bug").

If a subsequent e-mail arrives from a different customer, reporting the same bug, the agent would not need to report the bug again because it knows that it has already reported it. Instead, it might reply to the customer saying something like

> *Thank you for your email—we are already aware of this problem, and our engineers are working to fix it now.*

19.4.1 Goal-based Agents

Goal-based agents are more complex than reactive agents. Rather than following a predetermined set of rules, a goal-based agent acts to try to achieve a goal. This is often done by using **search** (see Part 2) or **planning** (see Part 5).

A goal-based agent might, for example, be given the goal of finding pages on the Internet that are of interest to an Artificial Intelligence researcher. The agent will be designed so that it is capable of carrying out actions (such as loading a web page, examining it, and following links from one web page to another). It is also able to identify when it has reached a goal (for example, by matching the pages it finds against a set of keywords whose presence indicates relevance to Artificial Intelligence).

This goal based agent would search the Internet looking for pages that matched its criteria and would presumably report those pages to its owner or to a client. This kind of agent does not take into account how efficiently

it is searching or how relevant the pages are that it is finding. In other words, its aim is simply to satisfy its goal; it does not take into account how *well* it has satisfied the goal or how *efficiently*. **Utility-based agents**, which are described in the next section, use these concepts to attempt to provide better results and in a more efficient manner.

19.4.2 Utility-based Agents

A **utility-based agent** is similar to a goal-based agent, but in addition to attempting to achieve a set of goals, the utility-based agent is also trying to maximize some **utility value**. The utility value can be thought of as the happiness of the agent, or how successful it is being. It may also take into account how much work the agent needs to do to achieve its goals.

Let us return to our example from the previous section of an agent that searches for pages on the Internet that are of interest to Artificial Intelligence researchers.

The utility-based agent can use knowledge about the Internet to follow the most worthwhile paths from one page to another. In other words, it can use heuristic-based search techniques to minimize the amount of time it spends examining pages that are not of interest and to maximize the likelihood that if an interesting page exists, it will be found (this combines search concepts from Chapters 4 and 5 with information retrieval techniques, which are discussed in Chapter 20).

The techniques we saw in Chapter 6 for game-playing systems can also be used as part of a utility-based agent. In this case, the agent's utility function is based on how successful it is at playing the game, and its goal is to maximize this utility function by winning the game.

19.4.3 Utility Functions

A utility function maps a set of states to the set of real numbers. In other words, given a particular state of the world, an agent is able to use its utility function to derive a score, or utility value, that tells it how "happy" it is in that state or how successful it has been if it reaches that state.

The static board evaluators that we saw in Chapter 6 are an example of a utility function that is used to evaluate a single position in a board game.

By searching through a tree of possible future states, based on available actions, and selecting a path that maximizes the utility function throughout the tree, a utility-based agent is able to achieve its goals effectively and efficiently.

For example, our Artificial Intelligence research agent might assign a high utility value to pages that are written in English and that appear to be written by a reliable source.

The idea of utility is closely related to the idea of **rationality**. An agent that behaves rationally is one that attempts to maximize its utility function. This utility function may not seem rational to all observers, although a rational agent might be programmed to lose at chess as spectacularly as possible. By losing a game, this agent maximizes its utility function and so, contrary to appearance, it is behaving rationally.

This model of utility is based on economics theory. One utility function for people is money. In general, people tend to prefer to have more money rather than less money. It is not as simple as this though. We might assume that the utility function for a human relating to money (ignoring other aspects of life) is simply based on the amount of money that that person had. This is contradicted by an experiment carried out in 1982 by psychologists, Tversky and Kahneman. In their experiment, they offered subjects two consecutive choices:

1. A or B:

   A = 80% chance of winning \$4000

   B = 100% chance of winning \$3000

2. C or D:

   C = 20% chance of winning \$4000

   D = 25% chance of winning \$3000

Most subjects choose *A*, rather than *B;* and *C*, rather than *D*. Let us consider the utility of these choices. In the choice between *A* and *B*, we have an 80% chance of winning \$4000 or a 100% chance of winning \$3000. The expected values of these two choices are

$$E(A) = 0.8 \times 4000 = 3200$$
$$E(B) = 1.0 \times 3000 = 3000$$

Hence, the most rational choice, using a simple utility function, would be to select *A* rather than *B*. For the choice between *C* and *D*, the expected values are

$$E(C) = 0.2 \times 4000 = 800$$

$$E(D) = 0.25 \times 3000 = 750$$

So in this choice, most people make the more rational decision on the basis of the simple utility function. What this experiment tells us is that people have much more complex utility functions than we might assume.

Similarly, utility-based intelligent agents usually need sophisticated utility functions. In the case of a chess playing agent, for example, a utility function based solely on the number of pieces each player has would not be sufficient. A utility function based on which player wins is fine, but as we saw in Chapter 6, this does not help the agent to play the game because the search tree is usually too large for the agent to reach a position where one player has won.

19.5 Interface Agents

An interface agent can be thought of as a personal assistant. Interface agents are typically autonomous agents, capable of learning in order to carry out tasks on behalf of a human user. Typically, interface agents collaborate with the user, but do not need to collaborate with other agents; although in some cases, interface agents can learn by seeking advice from other agents.

A typical example of an interface agent is a tool that is used to help a user learn to use a new software package. Such an agent has the ability to observe what the user does and make suggestions for better ways to perform those tasks. It is also able to assist the user in carrying out complex tasks, possibly learning as it does so. Interface agents can thus take instructions from users and can also learn from feedback from users about whether they are doing a good job or not, in order to perform better in future.

It is often useful for repetitive tasks to be delegated to an interface agent. The interface agent can learn how to carry out the task by observing the user and then is able to repeat the task as required.

Kozierok and Maes (1993) describe an interface agent that is able to assist a user with scheduling meetings on a calendar. The agent is able to arrange meetings with other people and is also able to accept, reject, and rearrange

meetings on behalf of the user. By observing the user's behavior, it is able to learn, for example, that the user does not like to book meetings on Friday afternoons and so is able to avoid such meetings.

A number of tools exist that filter Usenet postings and new articles for a user. These tools can typically be trained by example: a user can show examples of interesting articles, and examples of uninteresting articles and the agent can learn to identify interesting articles and present those to the user, while avoiding uninteresting ones.

19.6 Mobile Agents

Mobile agents are those capable of "moving" from one place to another. In the case of mobile robots, this literally means moving in physical space. In the case of mobile software agents, this mobility usually refers to the Internet or other network. An agent that is not mobile is **static**.

Mobile agents travel from one computer to another, gathering information and performing actions as needed on the basis of that information. A computer virus can be thought of as a form of mobile agent, although most viruses are not intelligent, merely autonomous. That is, they are able to act without being given direct instruction from a human, but they do not adapt intelligently to their surroundings—they simply follow a fixed set of rules that tells them how to infect a computer and how to reproduce.

For mobile agents to run on remote computers, a suitable environment must of course be provided that allows the agent to run on that machine. An example of a system that provides such an environment is Telescript, developed by General Magic. The Java programming language, developed by Sun, can also be used for developing mobile agents.

The idea that a mobile agent can be sent from one computer across the Internet to run on another computer raises many security questions.

The main advantages of mobile agents are in efficiency. An agent that has to communicate with a number of remote servers and request large quantities of information in order to make a decision uses a large amount of bandwidth, which can be avoided if the agent is able to physically move to the remote server and query it locally.

Similarly, the mobile agent may be able to take advantage of superior computing power or the existence of particular functional abilities at the remote machine that are not present locally.

In this way, mobile agents can be used to generate a **distributed computing architecture**, where computation takes place on multiple computers at arbitrary locations.

A further advantage of mobile agents is that they can carry out their tasks asynchronously: the user can set a mobile agent off on a particular task and can then get on with other work, or maybe even switch the computer off. When the user is ready to receive the results, the agent can be recalled.

19.7 Information Agents

Information agents, also known as **information-gathering agents**, are usually used on the Internet and so are also sometimes called **Internet agents**. An information agent is used to help a user find, filter, and classify information from the vast array of sources available on the Internet.

Information agents may be static or mobile. Some information agents are capable of learning, whereas the behavior of others is fixed. Additionally, information agents can be collaborative or can work independently of other agents. The distinctive feature of an information agent is the function that it provides, rather than the way it works.

There is an overlap between information agents and other kinds of agents described in this chapter. The interface agents described in Section 19.5, which monitor Usenet postings or online news articles, are examples of information agents.

Information agents know how to search the Internet, usually using a number of search tools. In this way, they are able to cover as much content as possible and thus maximize their **recall** (see Chapter 20). The real challenge is usually **precision**. This is heavily dependent on the ability of the agent to receive input instructions from the user. Some agents learn by example: the user shows the agent examples of pages that are relevant and pages that are not relevant, and the system learns to differentiate the two groups. Other agents are directed by keywords or more sophisticated information retrieval techniques (see Chapter 20) to identify relevant material for the user.

The Internet provides some unique challenges to these agents. Internet data is very **dirty**: most of the information on the Internet is not organized in any way; much of it includes misspellings, incorrect grammar, and incorrect facts. Additionally, the Internet is global in nature, and so material is available in almost every language.

The sheer quantity of the data and the dirty nature of the data make it very difficult for many information agent systems to provide adequate precision in identifying relevant documents.

Of course, this is one of the reasons that information agents are so useful. It is even harder for humans to locate the data they want than it is for the agents. Agents have the advantage of speed and of being able to examine pages asynchronously, delivering results to a user, perhaps by e-mail, once they are available.

More sophisticated information agents are able to monitor the browsing habits of users to identify the kinds of material they are interested in and to use that information to improve the performance of future searches.

19.8 Multiagent Systems

In many situations, simple reactive agents are sufficient. The fact that they do not have the ability to learn means that they are not suited to operating in complex, dynamic environments. Also, because such an agent is based on a set of rules, the number of tasks and situations that it can deal with is limited by the number of rules it has. In fact, most agents do not exist in isolation.

Multiagent systems are a common way of exploiting the potential power of agents by combining many agents in one system. Each agent in a multiagent system has incomplete information and is incapable of solving the entire problem on its own, but combined together, the agents form a system that has sufficient information and ability to solve the problem. The system does not have a centralized control mechanism for solving the problem.

An example of how many simple agents can combine together to produce complex behavior can be seen by examining the way that ant colonies function. Each ant has very little intelligence and very little ability to learn. Taken as a whole, however, the ant colony is able to deal with complex situations and in some ways behaves as a single living entity.

In much the same way, many "dumb" agents can be combined together to produce a more intelligent system. For example, the legs of a robot might be controlled by a set of agents. Each leg is controlled by a simple reactive robot that has instructions for how to move the leg according to what the leg encounters.

Communication and **collaboration** are desirable properties of multiagent systems. Communication means, for example, that agents can inform each other of changes in the environment or of new discoveries they have made. Collaboration means that agents can work together to solve a common goal.

In fact, multiagent systems often involve relatively simple interactions between agents, and as we have seen with systems like Reynolds' Boids (Chapter 13), the system as a whole is able to solve complex problems without the individual agents necessarily knowing anything about the overall problem. Such emergent behavior is a valuable property of multiagent systems.

Multiagent systems can be given the ability to learn to solve new problems using genetic algorithms (see Chapter 14). In this way, robots have been successfully developed whose limbs are controlled by individual agents, each of which has been developed using a genetic algorithm. The robots are able to walk in a way that mimics the locomotion of insects (Gary Parker 1997, 1998).

Agents in a multiagent system can be collaborative or competitive. Agents designed to play chess against other agents would clearly be competitive, whereas agents that traverse the Internet searching for specific material may find it advantageous to cooperate with other similar agents.

An **agent team** is a group of agents that collaborate together to achieve some common goal. It is often the case that an agent team consists of agents that operate in different ways and have different goals to accomplish. For example, a team of agents might be used to arrange travel for a businessman: one agent might book flights, another agent arranges hotel accommodation, a third agent arranges meetings with business associates, while a fourth agent arranges meals and entertainments.

In some situations, these agents will be competing with other agents, bidding for purchases, but the agents within the team will cooperate with each other (e.g., the meal-booking agent will inform the meeting booking agent if it changes its restaurant bookings, which might affect a meeting that has been arranged in that restaurant).

19.9 Collaborative Agents

Collaborative agent systems are multiagent systems in which the agents collaborate with each other to accomplish goals. This property, of cooperating to achieve a common goal, is known as **benevolence**.

Collaborative agents typically do not have the ability to learn, although some have simple learning abilities. As with multiagent systems, the idea is that a combination of many simple agents can solve a problem that each agent individually would not be able to solve.

Collaborative agent systems are able to take advantage of their parallel nature in order to solve problems faster than would otherwise be possible. They are also more reliable than traditional systems because additional agents can be added to provide redundancy: if one agent fails, or provides incorrect information, this will not affect the overall performance of the system because other agents will provide corrective information.

19.10 Agent Architectures

In this section, we will look at a number of **architectures** that can be used to build intelligent agents. The architecture of an agent is the way in which its various processing modules are connected together and the way in which those modules are connected to the environment in which the agent operates.

19.10.1 Subsumption Architecture

There are a number of architectures suitable for reactive agents. One of the most commonly used is Brooks' **subsumption architecture** (Brooks 1985). The subsumption architecture is a layered architecture that was designed for implementing physical robots, which does not involve any centralized intelligence or control mechanism.

The agent in this architecture has a set of inputs, a possible set of actions, and a layered set of modules, each of which is designed to control some aspect of the agent's behavior. Each layer is able to inhibit the behavior of layers below it.

The modules are **augmented finite state machines** (AFSMs), which are similar to the finite state automata we saw in Chapter 13. AFSMs are often based on production rules, as used by expert systems, which take the form

```
input → action
```

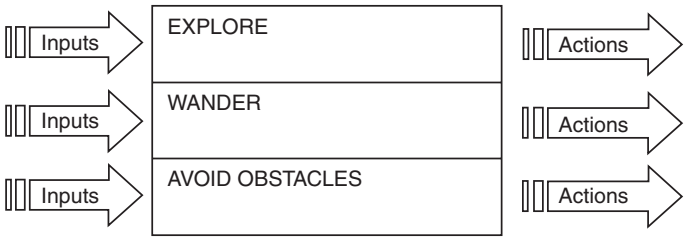

Figure 19.1
A three-layer subsumption architecture

These rules are called **situated action rules** or **situation action rules** because they map situations to actions. An agent that uses such rules is said to be **situated**, in that it is affected by where it is in its environment.

An AFSM is triggered when its inputs exceed a threshold. Each AFSM also has inhibitor inputs that can prevent it from triggering.

Rather than having a centralized representation, the subsumption architecture relies on lower-level modules that combine together. From these combined modules emerges intelligent behavior.

A simple subsumption architecture is shown in Figure 19.1.

This architecture was proposed by Brooks as a control mechanism for a robot. Each layer in the architecture is designed to handle one type of behavior: exploring, wandering, or avoiding obstacles. The modules act **asynchronously**, but each module can affect the behavior of the other modules.

The WANDER module will take into account the instructions generated by the AVOID OBSTACLES module, but it is also able to suppress the instructions generated by the AVOID OBSTACLES module, in order to ensure that while avoiding collisions, the robot still wanders around. This is to ensure that the robot does not simply focus on avoiding obstacles to the exclusion of everything else.

More important than wandering, for this robot, is exploration. Hence, the EXPLORE module is able to suppress instructions from the WANDER module to ensure that the robot continues to explore new territory, rather than simply wandering aimlessly.

Further layers can be added to the architecture to generate more sophisticated behavior—for example, Brooks describes a system that is able to wander around among desks in an office, looking for empty drink cans. This system has an architecture with additional layers for identifying drink cans, identifying desks, and so on (Brooks 1993).

19.10.2 BDI Architectures

BDI architectures, or **Belief Desire Intention** architectures, are based on the three concepts of belief, desire, and intention. A belief is a statement about the environment that the agent considers to be true. BDI agents have a set of beliefs that are similar to the set of facts contained in a rule-based production system. A desire is a goal state that the agent would like to reach, and the agent's intentions are the plans it has for how to behave in order to achieve its desires.

An agent can have an intention to carry out a particular action, in which case it will probably do so. Alternatively, an agent can have an intention to bring about a particular state.

When an agent **commits** to carrying out a particular action, or achieving a particular goal, it 'promises' that it will do so. Hence, a BDI agent has a set of beliefs that lead it to establish a set of desires. To achieve its desires, the BDI agent considers a number of **options** and commits to one or more of them. These options now become the agent's intentions.

Intentions persist until the goals are achieved, or until it becomes unreasonable to continue to attempt to achieve them (e.g., if it becomes obvious that the goals can never be achieved or if new beliefs are developed that lead the agent to change its desires).

A **bold** agent is one that establishes a set of intentions and then aims to carry them out without ever stopping to consider whether it should change its intentions. A **cautious** agent is one that considers its intentions continually. Kinny and Georgeff (1991) found that bold agents perform better than cautious agents in worlds where the environment does not change very frequently and that cautious agents perform better than bold agents in worlds that change quickly.

19.10.3 Other Architectures

A number of other agent architectures exist. Logic-based agents apply rules of logical deduction to a **symbolic representation** of their environment. The state of such an agent is usually represented using first-order predicates, and its behavior is determined by a set of deduction rules, usually expressed in first-order predicate logic.

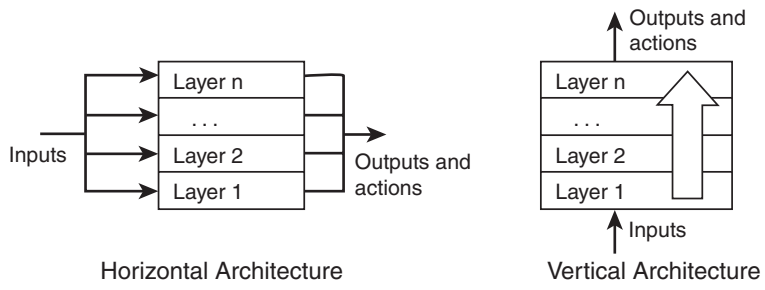

Figure 19.2
Horizontal and vertical agent architectures compared

In contrast to logic-based architectures, purely reactive agents do not perform any symbol manipulation and rely on a simple mapping from inputs to actions.

A number of layered architectures exist other than the subsumption architecture. The subsumption architecture is an example of a **horizontal layered architecture**, where each layer receives inputs and contributes to the actions and outputs of the agent. In a **vertical layered architecture**, input is passed to one layer, which then passes information on to a further layer. Actions and outputs are eventually produced by the final layer. These two architecture types are illustrated in Figure 19.2.

TouringMachines is an example of a horizontal architecture, which is based on three layers:

- *Reactive layer:* This layer uses situation rules to react to changes in the agent's environment.
- *Planning layer:* This layer uses a library of plans (called **schemas**) to determine the behavior of the agent, in order to achieve particular goals. In most situations, this is the layer that decides the main behavior of the agent.
- *Modeling layer:* This layer contains a model of the agent and any other agents in the world, in order to avoid conflicts with other agents.

InteRRaP is an example of a vertical layered architecture, which has three layers with very similar functions to the layers of the TouringMachines architecture. Each layer in the InteRRap architecture has a database of relevant knowledge: the reactive layer has a database of knowledge about the world the agent inhabits; the planning layer has a database of planning

knowledge that contains information about the agent's plans; the cooperation layer (similar to the modeling layer in TouringMachines) has social knowledge about the other agents and their interactions.

In the TouringMachines architecture, each layer interacts with the environment, directly receiving inputs and producing actions and outputs. In the InteRRap architecture, only the bottom layer (the reactive, behavior layer) interacts directly with the world. If it is unable to deal with a particular situation, it passes the information on to the next layer, the planning layer. Similarly, if this layer cannot deal with the current situation, it passes the information on to the final layer, the cooperation layer. Outputs are passed back to the behavior layer, which turns them into actions or outputs.

19.11 Accessibility

When playing a game such as chess, each player knows what position he will be in after making any given move. What he does not usually know is what move his opponent will make and, thus, what position he will reach after his opponent's move.

In some cases an agent's state after carrying out a particular action can be deterministically predicted. In many situations, however, this is not the case, and the outcome is unpredictable, or **stochastic**. Given that an agent usually has a certain degree of knowledge about the world and the way its actions affect its state, we can make certain predictions. For example, an agent can say that if it is in state S_1 and it takes action A, then it will move into state S_2 with probability p. These probabilities are contained within a **transition model**, which enables the agent to make predictions about what effect its actions will have on it and its environment.

If an agent is able to determine all relevant facts about the environment in which it operates, then that environment is described as being **accessible**. If it is **inaccessible**, then certain facts are hidden from the agent, although it may be able to deduce them by maintaining internal information about the state of the environment. For example, if an agent is in an environment in which it is unable to determine the temperature, it may have a rule that says "if you turn up the heating, the temperature will increase."

We could consider two types of agents that play chess. One agent might have the ability to examine the board at each move of the game and make decisions about what move to make from that point. The agent does not have the

ability to remember moves that have been made in the past, and thus the only way it can determine the current position is by examining the board.

This agent acts in an accessible environment because, at any given point, it has access to all the information it needs to be able to play the game. If we imagine that this agent is playing a game where half of the board is covered up, and it is unable to see what happens there, then we can see that the agent would have great difficulties because it would have no way of determining what was happening on that side of the board apart from a few limited facts it could deduce, such as "my king is on this side of the board, so I know I do not have a king on the other side of the board."

A different type of agent might play the game without any direct access to the board at all. This agent stores information about the moves that have been made in the past and is able to use this information to determine the current position of the board. This agent would play equally well whether the board were entirely visible or entirely covered up.

This agent operates in an inaccessible environment, but, in fact, because the environment it operates in is entirely deterministic, it is able to derive complete knowledge about the board at all times.

An agent that played a game such as poker would need to be able to act in an inaccessible, stochastic environment because the cards the opponent has are neither visible nor deterministically allocated.

In an accessible, stochastic environment, agents use **Markov decision processes** (MDPs) to determine the best course of action. In an inaccessible, stochastic environment, agents use **partially observable Markov decision processes** (POMDPs). Clearly, POMDPs must operate with far less information and so tend to be more complex than MDPs.

19.12 Learning Agents

Machine learning is covered in more detail in Part 4 of this book. An agent that is capable of **learning** (a **learning agent**) is able to acquire new knowledge and skills and is able to use the new knowledge and skills to improve its performance.

One common way to provide agents with the ability to learn is to use neural networks, which are covered in more detail in Chapter 11. A neural network is designed to learn in a similar manner to the way a human brain

learns. Another method for enabling agents to learn is to use genetic algorithms. One way to use genetic algorithms in this way is to have the genetic algorithm breed populations of agents, with the aim of breeding a highly successful agent. Another way is to have each agent use a genetic algorithm to develop suitable strategies for dealing with particular problems.

19.12.1 Multiagent Learning

Multiagent systems are often required to solve problems in dynamic and unpredictable environments. In these circumstances, a learning ability is particularly important because the environment can change too quickly for predetermined behaviors to be effective.

Multiagent learning can in many ways be more impressive than the learning carried out by individual agents. Each agent in a learning multiagent system can learn independently of the other agents and can also learn from the other agents.

In this way, the agents can explore multiple potential strategies in parallel, and when one agent discovers a particularly effective strategy, it can pass this knowledge on to other agents. For this reason, when the environment changes, multiagent learning systems are able to adapt much more quickly than nonlearning systems, or even individual learning agents.

In **centralized learning**, the agents learn on an individual and distinct basis, whereas in **decentralized learning**, the actions of the individual agents lead to the whole system learning. The classifier systems described in Chapter 13 are an example of a decentralized multiagent learning system, where each rule can be thought of as a separate agent, and where the whole system learns by experience how best to solve a problem.

19.13 Robotic Agents

The agents described in this chapter so far have been software agents—they exist only in a virtual world. **Robotic agents**, or **robots**, are artificial agents that exist physically in the real world.

Mobile robotic agents controlled by Brooks' subsumption architecture have been briefly described in Section 19.10.1.

Robotic agents operate in an inaccessible, stochastic environment. The real world has many properties that make the tasks of robotic agents much

harder than those of many software agents. An ability to deal with **uncertainty** is clearly important, as is robustness in the face of extremely unpredictable and potentially dangerous environments.

Robots have been designed that build cars, using robotic arms and conveyer belts.

More sophisticated are the robots that are designed to explore other planets and collect samples for scientific analysis. Such robots, of course, require autonomy: they cannot be controlled directly by human input because they would be too far away from the earth. One important aspect of such robots is their ability to walk: this involves not just knowing how to move legs in such a way as to move forward, but also how to navigate over hills and rocks, around pot-holes and through valleys. Agents such as Atilla and Genghis, designed by the MIT Mobot Lab (Mobot means "mobile robot"), have these abilities and are modeled on insects.

Genghis has six legs and a number of sensors that enable it to determine certain facts about its inaccessible environment. The interesting thing about Genghis is that nobody ever told it how to walk or steer around obstacles. Its brain consists of 57 augmented finite state machines, each of which is responsible for a simple piece of behavior, such as lifting a leg or wandering. Using these AFSMs and feedback from its sensors, Genghis was able to learn to walk from the experience of trying and failing to do so.

19.14 Braitenberg Vehicles

Braitenberg vehicles were invented by a neuroscientist, Valentino Braitenberg, in the 1980s. Braitenberg vehicles are imaginary robots used by Braitenberg in thought experiments on the nature of intelligence. There are 14 different classes of vehicles, ranging from extremely simple to fairly complex. We will consider just the six simplest types.

Even the simplest of his vehicles can exhibit interesting behaviors and tell us a great deal about our assumptions concerning intelligence and thought.

The simplest type of Braitenberg vehicle, known as vehicle 1, simply has one motor and a sensor. The sensor is wired directly to the motor, such that the more of whatever the sensor is designed to sense there is, the faster the motor turns. For example, if the sensor were a light sensor, then the motor would turn faster when the sensor could detect more light.

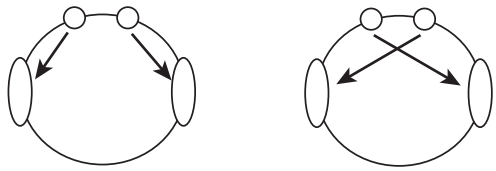

Figure 19.3
Two varieties of Braitenberg vehicles type 2, seen from above

The behavior of this vehicle is very simple: the more light there is, the faster it moves. It would normally move in a straight line, although imperfections in its environment (such as friction and obstacles) might cause it to deviate.

The second type of Braitenberg vehicle has two sensors and two motors. The motors and sensors are placed symmetrically around the vehicle, as shown in Figure 19.3.

In the first vehicle shown in Figure 19.3, the left-hand sensor (the sensors are on the front of the vehicle) is connected to the left-hand motor, and the right-hand sensor to the right-hand motor. In the second vehicle shown, the sensors and motors are connected the other way around. The first vehicle will tend to move away from the source that its sensors detect, whereas the second vehicle will move toward it.

These vehicles can be thought of as **timid** (the one that moves away from the source) and **bold** (the one that moves toward the source).

Let us now consider a type of the timid vehicle, which has a sensor for proximity and where its motors have a built-in tendency to move even without any stimulation to the sensors. When placed in a simple maze, this vehicle will navigate through the maze without bumping into the walls. Clearly, apparently complex behavior can emerge from very simple concepts. This timid vehicle was certainly not designed to traverse a maze, and it does not have any knowledge of mazes or the world. An observer who did not know how the vehicle worked might conclude that it relied on a very sophisticated form of Artificial Intelligence.

It is interesting to note at this point some of the words that we have been using to describe agents: *timid*, *bold*, *cautious*, and so on. There is a tendency to anthropomorphize the behaviors of agents, which is at least partly due to the impression that agents can give of having almost human-like intelligence.

The third type of vehicle is similar to the second type except that the sensors are wired in such a way that they inhibit the motors: the more stimula-

tion they receive, the slower the motors turn. These types of vehicles will tend to move toward a source of stimulation but will end up near the source, either facing it or turned away from it, depending on which way its sensors are wired to the motors.

Braitenberg vehicles can have more than one type of sensor—for example, a vehicle might have light sensors and proximity detectors for objects. These sensors can be connected to motors in different ways, producing more and more complex behaviors.

The fourth type of Braitenberg vehicle has a nonlinear relationship between input to the sensors and the speed of the motors. For example, one of these vehicles might move slowly toward a light source and speed up as it gets closer, then slow down again as it gets very close to the source.

The fifth type of vehicle has a primitive memory that can be used to store information about events that happened in the past.

The sixth type of Braitenberg vehicle is evolved using artificial evolution, as described in Chapters 13 and 14.

Braitenberg vehicles teach us the following principle, which Braitenberg called the principle of "Uphill Analysis and Downhill Invention": It is easier to invent something than to analyze it. Fully functioning Braitenberg vehicles can be built using easily available components, and yet their behavior can be extremely complex and, in some cases, impossible to analyze or explain.

19.15 Chapter Summary

- An agent is an entity that carries out a task on behalf of a human user.
- A software agent is an agent that exists solely as a computer program.
- Intelligent agents have more knowledge or understanding of their environment than simple agents and are able to use this intelligence to carry out their tasks more effectively.
- Autonomous agents are able to carry out their tasks without direct input from a human.
- Some agents are able to learn from their user, from other agents, from the environment, or by observing the consequences of their own actions.

- Reactive agents simply react to the environment they are in, using situated action rules, which provide an action for each situation.
- Goal-based agents seek to achieve some goal, whereas utility-based agents seek to maximize some utility function.
- Interface agents are automated personal assistants.
- Mobile agents are able to travel over a network, such as the Internet.
- An information agent collects information (often from the Internet) on behalf of its owner.
- Multiagent systems use a number of agents that usually collaborate together to achieve some common goal.
- The subsumption architecture is an example of a vertically layered architecture for controlling robots.
- BDI architectures use beliefs, desires, and intentions to control agents.
- An accessible environment is one in which all necessary facts are available to the agent. Many agents must be able to operate in inaccessible environments and often in stochastic ones, where the changes in the environment are unpredictable.
- Robotic agents operate in the real world.

19.16 Review Questions

19.1 "A computer virus is a kind of intelligent agent." Discuss this statement. Consider the various agent properties that have been discussed in this chapter. Which of these properties do computer viruses have?

19.2 Explain what is meant by the following terms in the context of agents:

- intelligence
- autonomy
- learning
- collaboration
- utility

19.3 Explain the idea behind the BDI architecture. Why do you think this architecture is particularly appealing to human researchers?

19.4 Explain the nature of the first six types of Braitenberg vehicles. Discuss how these vehicles can help us to understand the nature of intelligence.

19.5 Think of a real-world interface agent. Discuss to what extent this agent has autonomy, learning abilities, and intelligence.

19.6 What do Braitenberg vehicles teach us about intelligence? Do you think the intelligence given to Braitenberg vehicles could be put to some practical use?

19.7 In Michael Crichton's novel, *Prey*, he postulates a multiagent system consisting of millions of tiny robotic agents. The system evolves over a period of days to develop human-like intelligence, and a belligerent desire to destroy life. Discuss how plausible you think this idea is, in the context of the subjects introduced in this chapter.

19.17 Exercises

19.1 Implement an intelligent agent system to carry out a simple task for you in the programming language of your choice.

19.2 Investigate a software agent that comes with your computer, or find one that you can download for free. Explore its limitations and its capabilities. To what extent would you describe it as "intelligent"? What simple improvements would you suggest for the agent? Which of the following properties does the agent exhibit:

- intelligence
- autonomy
- ability to learn
- cooperation
- benevolence
- veracity

To what extent would it still be useful if it did not have the properties that it does have? Which of the above properties might be given to the agent to improve it? How would it be improved?

19.18 Further Reading

Several texts cover the subject of Artificial Intelligence from the perspective of Artificial Agents—in particular, Russell and Norvig (1995) and Pfeifer

and Scheier (1999). Weiss (1999) provides an excellent exploration of multiagent systems.

Brooks' subsumption architecture was introduced in *A Robust Layered Control System For a Mobile Robot* (from *IEEE Journal of Robotics and Automation, RA-2, April,* pp. 14–23), and was also published as *MIT AI Memo 864* (1985).

Braitenberg (1986) provides a fascinating description of his vehicles, as well as providing an absorbing philosophical argument. A good practical explanation of Braitenberg's vehicles is also found in Pfeifer and Scheier (2000)

Behavior-Based Robotics, by Ronald C. Arkin (1998 – MIT Press)

Software Agents, edited by Jeffrey M. Bradshaw (1997 – AAAI Press)

Vehicles: Experiments in Synthetic Psychology, by Valentino Braitenberg (1986 – MIT Press)

Intelligent Agents for Mobile and Virtual Media, edited by Rae Earnshaw, John Vince, and Margaret A. Arden (2002 – Springer Verlag)

Commitment and Effectiveness of Situated Agents, by D. Kinny and M. Georgeff (1991 – in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 82–88)

Braitenberg Creatures, by David W. Hogg, Fred Martin, and Mitchel Resnick (1991 – originally published as *Epistemology and Learning Memo #13*)

A Learning Interface Agent for Scheduling Meetings, by R. Kozierok and P. Maes (1993 – in *Proceedings of the ACM-SIGCHI International Workshop on Intelligent User Interfaces*)

Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines, by Stefano Nolfi and Dario Floreano (2000 – MIT Press)

Evolving Hexapod Gaits Using a Cyclic Genetic Algorithm, by Gary Parker (1997 – in *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing*, pp. 141–144)

Generating Arachnid Robot Gaits with Cyclic Genetic Algorithms, by Gary Parker (1998 - in Genetic Programming III, pp. 576–583)

Metachronal Wave Gait Generation for Hexapod Robots, by Gary Parker (1998 – in *Proceedings of the Seventh International Symposium on Robotics with Applications*)

Understanding Intelligence, by Rolf Pfeifer and Christian Scheier (2000 – MIT Press)

Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer, by Peter Stone (2000 – MIT Press)

Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, edited by Gerhard Weiss (1999 – MIT Press)

Introduction to MultiAgent Systems, by Michael Wooldridge (2002 – John Wiley & Sons)

Strategic Negotiation in Multiagent Environments, by Sarit Kraus (2001 – MIT Press)

Intelligent Information Agents: The Agentlink Perspective (Lecture Notes in Computer Science, 2586), edited by Matthias Klusch, Sonia Bergamaschi, and Pete Edwards (2003 – Springer Verlag)

An Introduction to AI Robotics, by Robin R. Murphy (2000 – MIT Press)

Real-Time and Multi-Agent Systems, by Ammar Attoui (2000 – Springer Verlag)

Understanding Agent Systems, edited by Mark D'Inverno and Michael Luck (2001 – Springer Verlag)

Agent Technology: Foundations, Applications, and Markets, edited by Nicholas R. Jennings and Michael J. Wooldridge (1998 – Springer Verlag)

Socially Intelligent Agents - Creating Relationships with Computers and Robots, edited by Kerstin Dautenhahn, Alan H. Bond, Lola Canamero, and Bruce Edmonds (2002 – Kluwer Academic Publishers)

This page intentionally left blank