function KE-AGENT(\textit{percept}) \text{ returns} \text{ an action} \\
\text{ persistent: }KB, \text{ a knowledge base} \\
\text{ t, a counter, initially 0, indicating time} \\
\text{ TELL}(KB, \text{ MAKE-PERCEPT-SENTENCE}(\text{percept}, t)) \\
\text{ action} \leftarrow \text{ \textbf{ASK}}(KB, \text{ MAKE-ACTION-QUERY()} ) \\
\text{ TELL}(KB, \text{ MAKE-ACTION-SENTENCE}(\text{action}, t)) \\
\text{ t} \leftarrow \text{ t} + 1 \\
\text{ return } \text{ action} \\

\textbf{Figure 7.1} A generic knowledge-based agent. Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

the knowledge level, where we need specify only what the agent knows and what its goals are, in order to fix its behavior. For example, an automated taxi might have the goal of taking a passenger from San Francisco to Mann County and might know that the Golden Gate Bridge is the only link between the two locations. Then we can expect it to cross the Golden Gate Bridge because it knows that that will achieve its goal. Notice that this analysis is independent of how the taxi works at the implementation level. It doesn't matter whether its geographical knowledge is implemented as linked lists or pixel maps, or whether it reasons by manipulating strings of symbols stored in registers or by propagating noisy signals in a network of neurons.

A knowledge-based agent can be built simply by \textbf{TELLing} it what it needs to \textbf{know}. Starting with an empty knowledge base, the agent designer can \textbf{TELL} sentences one by one until the agent knows how to operate in its environment. This is called the \textbf{declarative} approach to system \textbf{building}. In contrast, the \textbf{procedural} approach encodes desired behaviors directly as program code. In the 1970s and 1980s, advocates of the two approaches engaged in heated debates. We now understand that a successful agent often combines both declarative and procedural elements in its design, and that declarative knowledge can often be compiled into more efficient procedural code.

We can also provide a knowledge-based agent with mechanisms that allow it to learn for itself. These mechanisms, which are discussed in Chapter 18, create general knowledge about the environment from a series of percepts. A learning agent can be fully autonomous.

7.2 THE WUMPSUS WORLD

In this section we describe an environment in which knowledge-based agents can show their worth. The \textbf{wumpus} world is a cave consisting of rooms connected by passageways. Lurking somewhere in the cave is the \textbf{terrible wumpus}, a beast that eats anyone who enters its room. The wumpus can be shot by an agent, but the agent has only one \textbf{arrow}. Some rooms contain
bottomless pits that will trap anyone who wanders into these rooms (except for the wumpus, which is too big to fall in). The only mitigating feature of this bleak environment is the possibility of finding a heap of gold. Although the wumpus world is rather tame by modern computer game standards, it illustrates some important points about intelligence.

A sample wumpus world is shown in Figure 7.2. The precise definition of the task environment is given, as suggested in Section 2.3, by the PEAS description:

- **Performance measure**: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken and –10 for using up the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.

- **Environment**: A 4x4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. In addition, each square other than the start can be a pit, with probability 0.2.

- **Actuators**: The agent can move Forward, TurnLeft by 90°, or TurnRight by 90°. The agent dies a miserable death if it enters a square containing a pit or a live wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.) If an agent tries to move forward and bumps into a wall, then the agent does not move. The action Grab can be used to pick up the gold if it is in the same square as the agent. The action Shoot can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent has only one arrow, so only the first Shoot action has any effect. Finally, the action Climb can be used to climb out of the cave, but only from square [1,1].

- **Sensors**: The agent has five sensors, each of which gives a single bit of information:
  - In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a Stench.
  - In the squares directly adjacent to a pit, the agent will perceive a Breeze.
  - In the square where the gold is, the agent will perceive a Glitter.
  - When an agent walks into a wall, it will perceive a Bump.
  - When the wumpus is killed, it emits a woeful Scream that can be perceived anywhere in the cave.

The percepts will be given to the agent program in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get [Stench, Breeze, None, None, None].

We can characterize the wumpus environment along the various dimensions given in Chapter 2. Clearly, it is discrete, static, and single-agent. (The wumpus doesn't move, fortunately.) It is sequential, because rewards may come only after many actions are taken. It is partially observable, because some aspects of the state are not directly perceivable: the agent's location, the wumpus's state of health, and the availability of an arrow. As for the locations of the pits and the wumpus, we could treat them as unobserved parts of the state that happen to be immutable—in which case, the transition model for the environment is completely