**AI types**

The field of artificial intelligence, or AI, is concerned with not just understanding but also building intelligent entities—machines that can compute how to act effectively and safely in a wide variety of novel situations. AI currently encompasses a huge variety of subfields, ranging from the general (learning, reasoning, perception, and so on) to the specific, such as playing chess, proving mathematical theorems, writing poetry, driving a car, or diagnosing diseases. AI is relevant to any intellectual task; it is truly a universal field.

Historically, researchers have pursued several different versions of AI. Some have defined intelligence in terms of fidelity to human performance, while others prefer an abstract, formal definition of intelligence called rationality—loosely speaking, doing the “right thing”. The subject matter itself also varies: some consider intelligence to be a property of internal thought processes and reasoning, while others focus on intelligent behavior, an external characterization.

From these two dimensions—human vs. rational and thought vs. behavior—there are four possible combinations, and there have been adherents and research programs for all four. The methods used are necessarily different: the pursuit of human-like intelligence must be in part an empirical science related to psychology, involving observations and hypotheses about actual human behavior and thought processes; a rationalist approach, on the other hand, involves a combination of mathematics and engineering, and connects to statistics, control theory, and economics. The various groups have both disparaged and helped each other.

**Acting humanly: The Turing test approach**

The Turing test, proposed by Alan Turing (1950), was designed as a thought experiment that would sidestep the philosophical vagueness of the question “Can a machine think?” A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. Programming a computer to pass a rigorously applied test provides plenty to work on. The computer would need the following capabilities:

* natural language processing to communicate successfully in a human language;
* knowledge representation to store what it knows or hears;
* automated reasoning to answer questions and to draw new conclusions;
* machine learning to adapt to new circumstances and to detect and extrapolate patterns.

There is sometimes a confusion between the terms “artificial intelligence” and “machine learning”. Machine learning is a subfield of AI that studies the ability to improve performance based on experience. Some AI systems use machine learning methods to achieve competence, but some do not.

Turing viewed the physical simulation of a person as unnecessary to demonstrate intelligence. However, other researchers have proposed a total Turing test, which requires interaction with objects and people in the real world. To pass the total Turing test, a robot will also need the following:

* computer vision and speech recognition to perceive the world;
* robotics to manipulate objects and move about.

These six disciplines compose most of AI. Yet AI researchers have devoted little effort to passing the Turing test, believing that it is more important to study the underlying principles of intelligence. The quest for “artificial flight” succeeded when engineers and inventors stopped imitating birds and started using wind tunnels and learning about aerodynamics. Aeronautical engineering texts do not define the goal of their field as making “machines that fly so exactly like pigeons that they can fool even other pigeons”.

**Thinking humanly: The cognitive modeling approach**

To say that a program thinks like a human, we must know how humans think. We can learn about human thought in three ways:

* introspection—trying to catch our own thoughts as they go by;
* psychological experiments—observing a person in action;
* brain imaging—observing the brain in action.

Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. If the program’s input–output behavior matches corresponding human behavior, that is evidence that some of the program’s mechanisms could also be operating in humans.

The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind.

Real cognitive science is necessarily based on experimental investigation of actual humans or animals. In the early days of AI there was often confusion between the approaches. An author would argue that an algorithm performs well on a task and that it is therefore a good model of human performance, or vice versa. Modern authors separate the two kinds of claims; this distinction has allowed both AI and cognitive science to develop more rapidly. The two fields propel each other, most notably in computer vision, which incorporates neurophysiological evidence into computational models. Recently, the combination of neuroimaging methods combined with machine learning techniques for analyzing such data has led to the beginnings of a capability to “read minds” - that is, to ascertain the semantic content of a person’s inner thoughts. This capability could, in turn, shed further light on how human cognition works.

**Thinking rationally: The “laws of thought” approach**

The laws of thought were supposed to govern the operation of the mind; their study initiated the field called logic. Logicians in the 19th century developed a precise notation for statements about objects in the world and the relations among them. (Contrast this with ordinary arithmetic notation, which provides only for statements about numbers.) By 1965, programs could, in principle, solve any solvable problem described in logical notation. The so-called logicist tradition within artificial intelligence hopes to build on such programs to create intelligent systems.

Logic as conventionally understood requires knowledge of the world that is certain - a condition that, in reality, is seldom achieved. We simply don’t know the rules of, say, politics or warfare in the same way that we know the rules of chess or arithmetic. The theory of probability fills this gap, allowing rigorous reasoning with uncertain information. In principle, it allows the construction of a comprehensive model of rational thought, leading from raw perceptual information to an understanding of how the world works and to predictions about the future. What it does not do, is generate intelligent behavior. For that, we need a theory of rational action. Rational thought, by itself, is not enough.

**Acting rationally: The rational agent approach**

An agent is just something that acts. Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, create and pursue goals. A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

In the “laws of thought” approach to AI, the emphasis was on correct inferences. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to deduce that a given action is best and then to act on that conclusion. On the other hand, there are ways of acting rationally that cannot be said to involve inference. For example, in a dangerous situation a reflex action may be more successful than a slower action taken after careful deliberation.

All the skills needed for the Turing test also allow an agent to act rationally. Knowledge representation and reasoning enable agents to reach good decisions. We need to be able to generate comprehensible sentences in natural language to get by in a complex society. We need learning not only for erudition, but also because it improves our ability to generate effective behavior, especially in circumstances that are new.

The rational-agent approach to AI has two advantages over the other approaches. First, it is more general than the “laws of thought” approach because correct inference is just one of several possible mechanisms for achieving rationality. Second, it is more amenable to scientific development. The standard of rationality is mathematically well defined and completely general. We can often work back from this specification to derive agent designs that provably achieve it—something that is largely impossible if the goal is to imitate human behavior or thought processes.

In a nutshell, AI has focused on the study and construction of agents that do the right thing. What counts as the right thing is defined by the objective that we provide to the agent. This general paradigm is so pervasive that we might call it the standard model. It prevails not only in AI, but also in control theory, where a controller minimizes a cost function; in operations research, where a policy maximizes a sum of rewards; in statistics, where a decision rule minimizes a loss function; and in economics, where a decision maker maximizes utility or some measure of social welfare.

However, perfect rationality—always taking the exactly optimal action—is not feasible in complex environments. The computational demands are just too high. Hence, the issue of limited rationality—acting appropriately when there is not enough time to do all the computations one might like. Still, perfect rationality often remains a good starting point for theoretical analysis.

**Intelligent Agents**

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

The term percept refers to the content an agent’s sensors are perceiving. An agent’s percept sequence is the complete history of everything the agent has ever perceived. In general, an agent’s choice of action at any given instant can depend on its built-in knowledge and on the entire percept sequence observed to date, but not on anything it hasn’t perceived. Mathematically speaking, an agent’s behavior is described by the agent function that maps any given percept sequence to an action.

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 expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Doing actions in order to modify future percepts - sometimes called exploration or information gathering - is an important part of rationality. A rational agent is required 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 and completely predictable. In such cases, the agent need not perceive or learn; it simply acts correctly.

A rational agent should be autonomous - it should learn what it can to compensate for partial or incorrect prior knowledge. 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.

**The Nature of Environments**

Task environments are essentially the “problems” to which rational agents are the “solutions”. For any agent we have 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: the PEAS (Performance, Environment, Actuators, Sensors) description. In designing an agent, the first step must always be to specify the task environment as fully as possible.

Let us consider an automated taxi driver for an example and summarize the PEAS description for the taxi’s task environment.

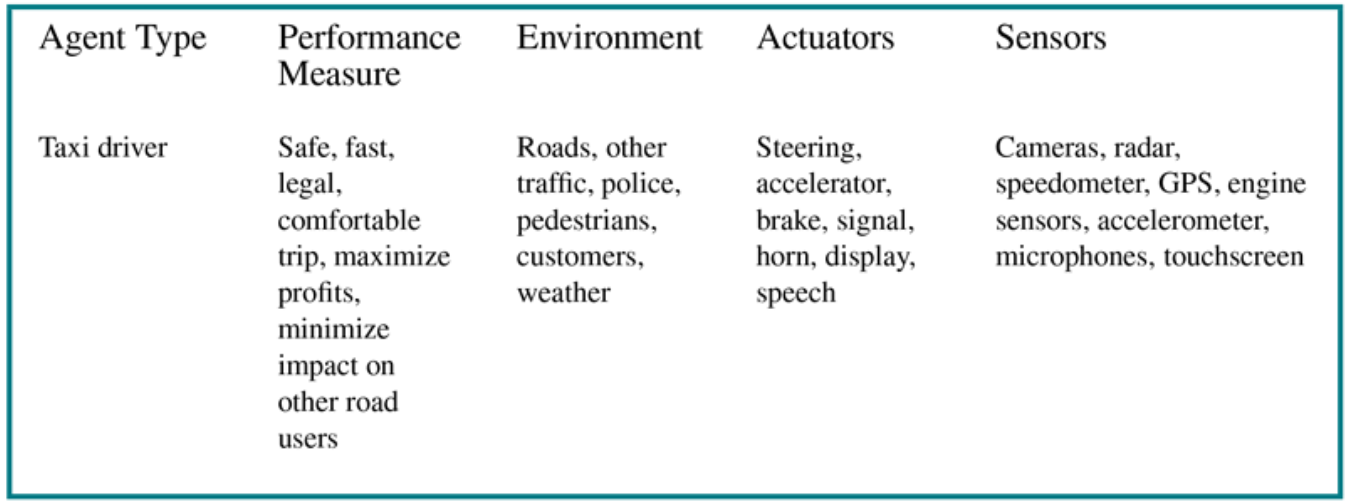


Figure 1 - PEAS description of the task environment for an automated taxi driver

**Performance measure**

Desirable qualities include getting to the correct destination; minimizing fuel consumption 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; maximizing profits. Obviously, some of these goals conflict, so tradeoffs will be required.

**Driving environment**

Any taxi driver must deal with a variety of roads, ranging from rural lanes and urban alleys to 8-lane freeways. The roads contain other traffic, pedestrians, stray animals, road works, police cars, puddles, 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 areas with cold weather, where snow and/or rain is frequently a problem. 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.

**Actuators**

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.

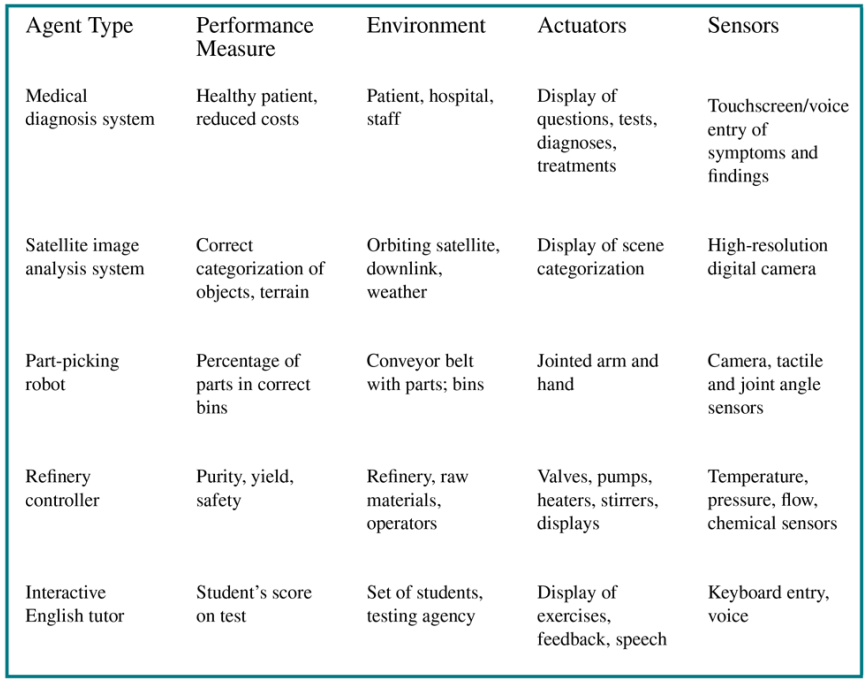
**

Figure 2 - Examples of agent types and their PEAS descriptions

**Sensors**

The basic sensors for the taxi will include one or more video cameras so that it can see, as well as lidar and ultrasound sensors to detect distances to other cars and obstacles. To avoid speeding tickets, the taxi should have a speedometer, 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 to access GPS signals so that it doesn’t get lost. Finally, it will need touchscreen or voice input for the passenger to request a destination.

**Properties of task environments**

Among the vast amount of task environments a fairly small number of dimensions can be categorized. 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.

**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 environment 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. If the agent has no sensors at all then the environment is unobservable. Despite that, the agent’s goals may still be achievable, sometimes with certainty.

**Single-agent vs. multiagent.** 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. However, there are some subtle issues. Does an agent *A* have to treat an object *B* as an agent, or can it be treated merely as an object behaving according to the laws of physics? 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 performance measure. Thus, chess is a competitive multiagent environment. On the other hand, in the taxi-driving environment, avoiding collisions maximizes the performance measure of all agents, so it is a partially cooperative multiagent environment. It is also partially competitive 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 environments; 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. nondeterministic.** If the next state of the environment is completely determined by the current state and the action executed by the agent(s), then the environment is deterministic; otherwise, it is nondeterministic. In principle, an agent need not worry about uncertainty in a fully observable, deterministic environment. If the environment is partially observable, however, then it could appear to be nondeterministic. 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 nondeterministic.

**Episodic vs. sequential.** In an episodic task environment, the agent’s experience is divided into episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. Many classification tasks are episodic. For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions; moreover, the current decision doesn’t affect whether the next part is defective.

In sequential environments, on the other hand, the current decision could affect all future decisions. Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

**Static vs. dynamic.** If the environment can change while an agent is deliberating, then 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. 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 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 speaking, but is typically treated as representing continuously varying intensities and locations.

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

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

The performance measure itself may be unknown, either because the designer is not sure how to write it down correctly or because the ultimate user - whose preferences matter - is not known. For example, a taxi driver usually won’t know whether a new passenger prefers a leisurely or speedy journey, a cautious or aggressive driving style. A virtual personal assistant starts out knowing nothing about the personal preferences of its new owner. In such cases, the agent may learn more about the performance measure based on further interactions with the designer or user. This, in turn, suggests that the task environment is necessarily viewed as a multiagent environment.

The hardest case is partially observable, multiagent, nondeterministic, sequential, dynamic, continuous, and unknown. Taxi driving is hard in all these senses, except that the driver’s environment is mostly known. Driving a rented car in a new country with unfamiliar geography, different traffic laws, and nervous passengers is a lot more exciting.

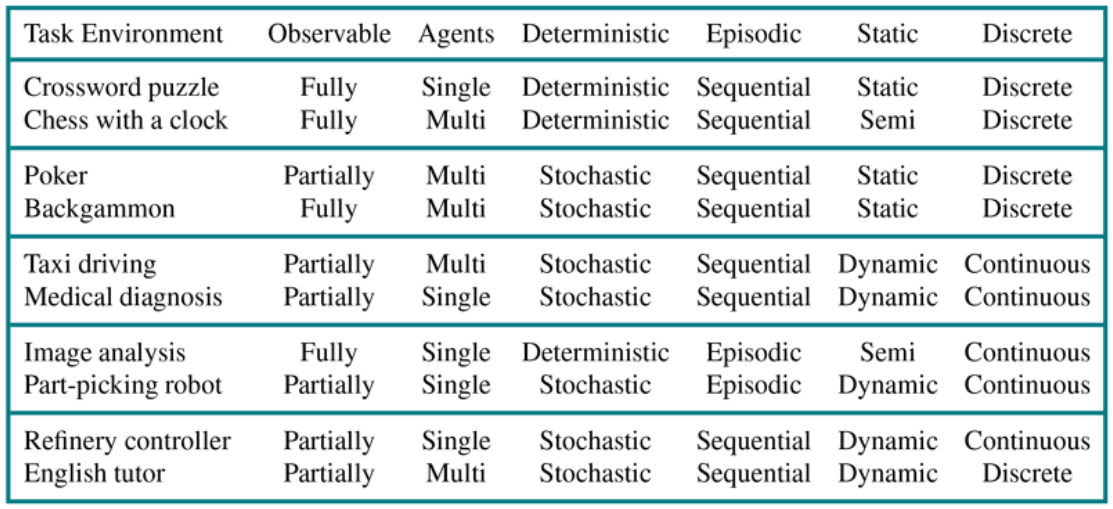
**

Figure 3 - Examples of task environments and their characteristics

**The Structure of Agents**

The job of AI is to design an agent program that implements the agent function - the mapping from percepts to actions. This program supposedly will run on some sort of computing device with physical sensors and actuators - it’s called the agent architecture: agent = architecture + program.

Obviously, the chosen program 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.

There are 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; utility-based agents.

**Simple reflex agents**

These agents select actions on the basis of the current percept, ignoring the rest of the percept history. For example, a simple reflex agent, whose decision is based only on the current location and status:

**function** REFLEX-AGENT([*location*, *status*]) **returns** *Action*

**if** *status* == *Ready* **then return** *Action1*

**else if** *location* == *A* **then return***MoveRight*

**else if** *location* == *B* **then return***MoveLeft*

The program above is specific to one particular 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.

**function** REFLEX-AGENT(*percept*) **returns** *Action*

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

*state ←* INTERPRET\_INPUT(*percept*)

*rule ←*RULE\_MATCH(*state*, *rules*)

*action ← rule.*ACTION

**return** *action*

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.

Simple reflex agents have the admirable property of being simple, but they are of limited intelligence. The agent listed above will work only if the correct decision can be made on the basis of just the current percept - that is, only if the environment is fully observable.

**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 statethat depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.

Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program in some form. First, we need some information about how the world changes over time, which can be divided roughly into two parts: the effects of the agent’s actions and how the world evolves independently of the agent. For example, when the agent turns the steering wheel clockwise, the car turns to the right, and when it’s raining the car’s cameras can get wet. This knowledge about “how the world works” - whether implemented in simple Boolean circuits or in complete scientific theories - is called a transition model of the world.

Second, we need some information about how the state of the world is reflected in the agent’s percepts. For example, when the car in front initiates braking, one or more illuminated red regions appear in the forward-facing camera image, and, when the camera gets wet, droplet-shaped objects appear in the image partially obscuring the road. This kind of knowledge is called a sensor model.

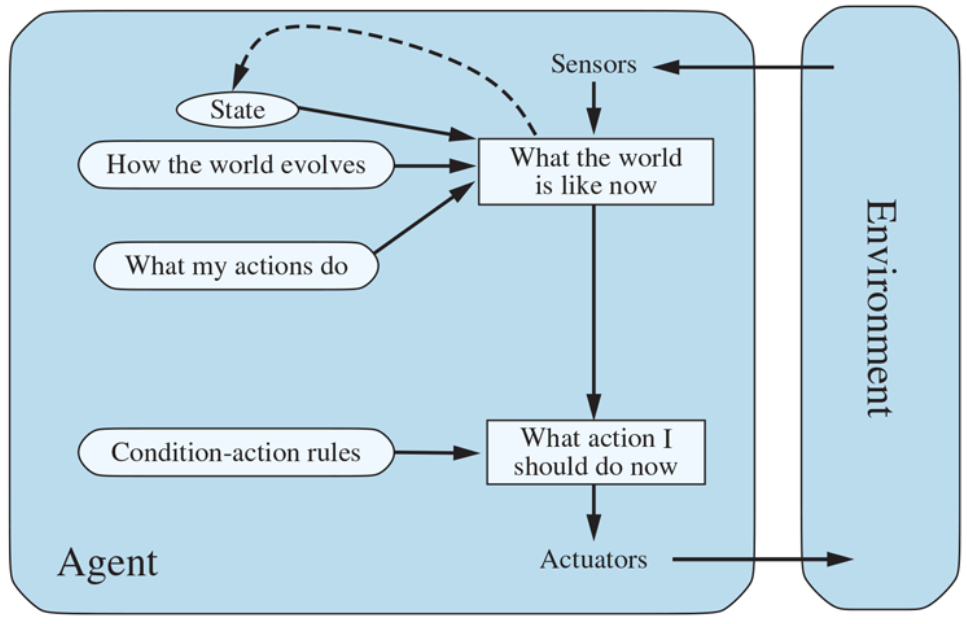
**

Figure 4 - A model-based reflex agent

For example, the structure of the model-based reflex agent with internal state shows 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.

**function** MODEL-BASED-REFLEX-AGENT(*percept*) **returns** *Action*

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

*transition\_model*, a description of how the next state depends   
 on the current state and action

*sensor\_model*, a description of how the current world state is   
 reflected in the agent’s percepts

*rules*, a set of condition-action rules

*action*, the most recent action, initially none

*state ←* UPDATE\_STATE(*state*, *action*, *percept, transition\_model,   
 sensor\_model*)

*rule ←* RULE\_MATCH(*state*, *rules*)

*action ← rule.*ACTION

**return** *action*

The interesting part is the function UPDATE-STATE, which 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.

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. 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.

**Goal-based agents**

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

Sometimes goal-based action selection is straightforward—for example, when goal satisfaction 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 and planning are the subfields of AI devoted to finding action sequences that achieve the agent’s goals.

Decision making of this kind is fundamentally different from the condition–action rules described earlier, in that it involves consideration of the future—“What will happen if I do such-and-such?” In the reflex agent designs, this information is not explicitly represented, because the built-in rules map directly from percepts to actions. The reflex agent brakes when it sees brake lights, period. A goal-based agent brakes when it sees brake lights because that’s the only action that it predicts will achieve its goal of not hitting other cars.

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. For example, a 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.

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. Provided that the internal utility function and the external performance measure are in agreement, an agent that chooses actions to maximize its utility will be rational according to the external performance measure.

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 nondeterminism 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 outcome. 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.

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. 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. Not all utility-based agents are model-based: a model-free agent can learn what action is best in a particular situation without ever learning exactly how that action changes the environment.

**Learning agents**

Any type of agent (model-based, goal-based, utility-based, etc.) can be built as a learning agent (or not). In many areas of AI, this is now the preferred method for creating state-of-the-art systems. Learning allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow.

A learning agent can be divided into four conceptual components. The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting external actions. The performance element 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. Given a design for the performance element, learning mechanisms can be constructed to improve every part of the agent.

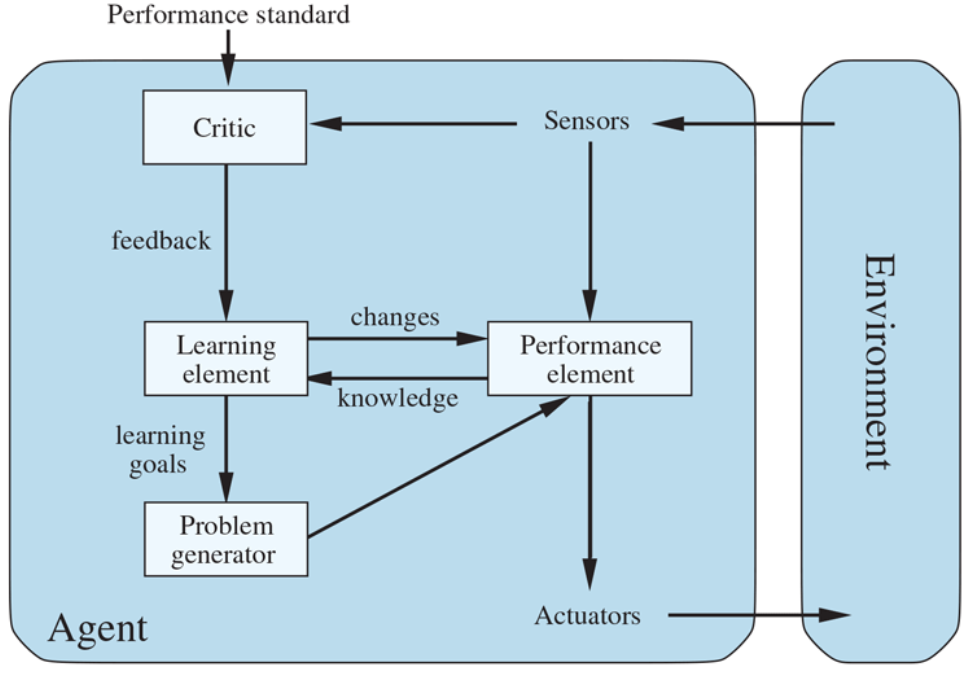


Figure 5 - A general learning agent

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 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. 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.

The learning element can make changes to any of the “knowledge” components of an agent. The simplest cases involve learning directly from the percept sequence. Observation of pairs of successive states of the environment can allow the agent to learn “What my actions do” and “How the world evolves” in response to its actions. *For example, if the automated taxi exerts a certain braking pressure when driving on a wet road, then it will soon find out how much deceleration is actually achieved, and whether it skids off the road. The problem generator might identify certain parts of the model that are in need of improvement and suggest experiments, such as trying out the brakes on different road surfaces under different conditions.*

Information from the external standard is needed when trying to learn a reflex component or a utility function. *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.*

Agents have a variety of components, and those components can be represented in many ways within the agent program, so there appears to be great variety among 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 improving the overall performance of the agent.

**Multiagent systems**

Assuming that only one agent is doing the sensing, planning, and acting represents a huge simplifying assumption, which fails to capture many real-world AI settings. So, let us consider the issues that arise when an agent must make decisions in environments that contain multiple actors. Such environments are called multiagent systems, and agents in such a system face a multiagent planning problem. However, as we will see, the precise nature of the multiagent planning problem—and the techniques that are appropriate for solving it—will depend on the relationships among the agents in the environment.

The first possibility is that while the environment contains multiple actors, it contains only one decision maker. In such a case, the decision maker develops plans for the other agents, and tells them what to do. The assumption that agents will simply do what they are told is called the benevolent agent assumption. However, even in this setting, plans involving multiple actors will require actors to synchronize their actions. Actors A and B will have to act at the same time for joint actions (such as singing a duet), at different times for mutually exclusive actions (such as recharging batteries when there is only one plug), and sequentially when one establishes a precondition for the other (such as A washing the dishes and then B drying them).

One special case is where we have a single decision maker with multiple effectors that can operate concurrently—for example, a human who can walk and talk at the same time. Such an agent needs to do multieffector planning to manage each effector while handling positive and negative interactions among the effectors. When the effectors are physically decoupled into detached units—as in a fleet of delivery robots in a factory—multieffector planning becomes multibody planning.

When there are communication constraints, it is sometimes called a decentralized planning problem; this is perhaps a misnomer, because the planning phase is centralized but the execution phase is at least partially decoupled. In this case, the subplan constructed for each body may need to include explicit communicative actions with other bodies. For example, multiple reconnaissance robots covering a wide area may often be out of radio contact with each other and should share their findings during times when communication is feasible.

The second possibility is that the other actors in the environment are also decision makers: they each have preferences and choose and execute their own plan. We call them counterparts. In this case, we can distinguish two further possibilities.

The first is that, although there are multiple decision makers, they are all pursuing a common goal. This is roughly the situation of workers in a company, in which different decision makers are pursuing, one hopes, the same goals on behalf of the company. The main problem faced by the decision makers in this setting is the coordination problem: they need to ensure that they are all pulling in the same direction, and not accidentally fouling up each other’s plans.

The second possibility is that the decision makers each have their own personal preferences, which they each will pursue to the best of their abilities. It could be that the preferences are diametrically opposed, as is the case in zero-sum games such as chess. But most multiagent encounters are more complicated than that, with more complex preferences.

When there are multiple decision makers, each pursuing their own preferences, an agent must take into account the preferences of other agents, as well as the fact that these other agents are also taking into account the preferences of other agents, and so on. This brings us into the realm of game theory: the theory of strategic decision making. It is this strategic aspect of reasoning—players each taking into account how other players may act—that distinguishes game theory from decision theory. In the same way that decision theory provides the theoretical foundation for decision making in single-agent AI, game theory provides the theoretical foundation for decision making in multiagent systems.

Game theory is the theory of strategic decision making. It is used in decision making situations including the auctioning of oil drilling rights and wireless frequency spectrum rights, bankruptcy proceedings, product development and pricing decisions, and national defense—situations involving billions of dollars and many lives. Game theory in AI can be used in two main ways:

1. AGENT DESIGN: Game theory can be used by an agent to analyze its possible decisions and compute the expected utility for each of these (under the assumption that other agents are acting rationally, according to game theory). In this way, gametheoretic techniques can determine the best strategy against a rational player and the expected return for each player.

2. MECHANISM DESIGN: When an environment is inhabited by many agents, it might be possible to define the rules of the environment (i.e., the game that the agents must play) so that the collective good of all agents is maximized when each agent adopts the game-theoretic solution that maximizes its own utility. For example, game theory can help design the protocols for a collection of Internet traffic routers so that each router has an incentive to act in such a way that global throughput is maximized. Mechanism design can also be used to construct intelligent multiagent systems that solve complex problems in a distributed fashion.