Intelligent Agents

Introduction

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

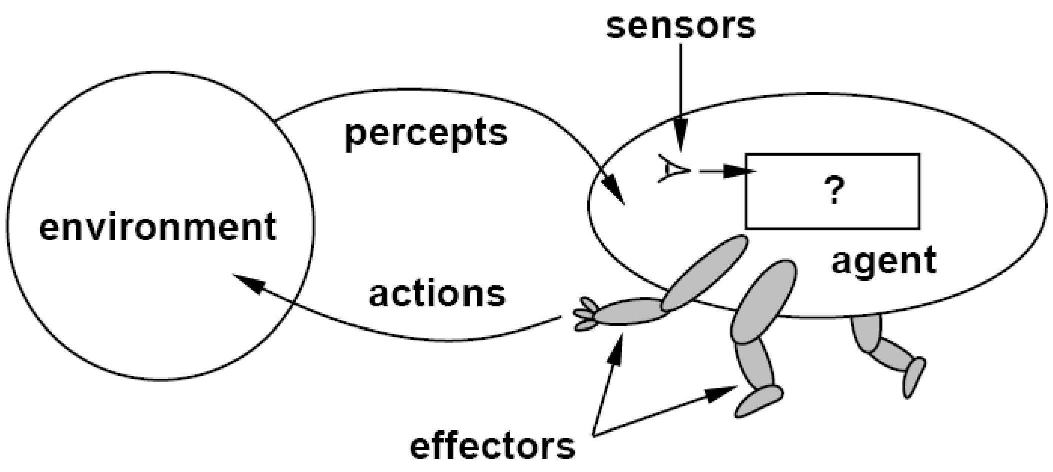


Figure 1: Agents interact with environments through sensors and effectors.

How Agents Should Act

Define : A **rational agent** is one that does the right thing.

Problem : What is a right thing, or right action?

Decided : The one that will cause the agent to be more successful.

Problem again : ***How*** and ***when*** to evaluate the agent‘s success?

How : Use the *objective* **performance measure** -- the criteria that determine how successful an agent is.

When : Measure the performance **over the long run**.

Just remember that :

- a rational agent is not a **omniscience**.
- That is, an agent can not be blamed for failing to take into account something it could not perceive, or for failing to take an action that is incapable of taking.

So, what is rational depends on :

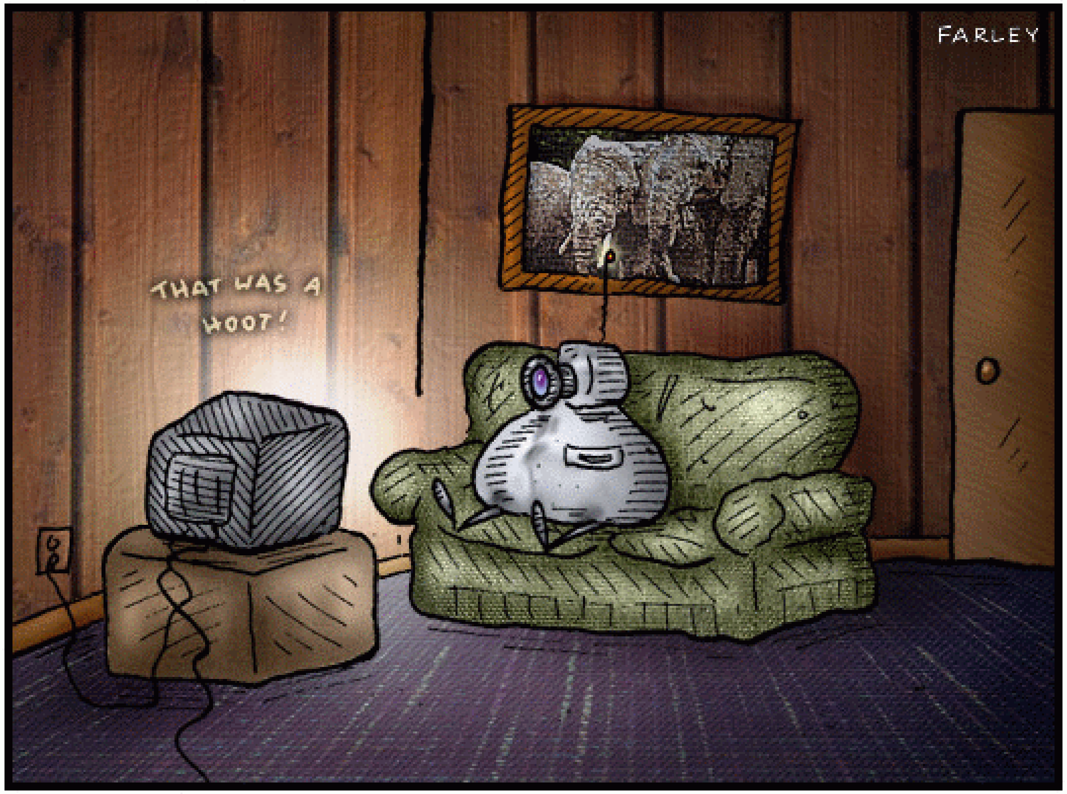
- The performance measure that defines degree of success.
- Everything that the agent has perceived so far. This complete perceptual history is called the **percept sequence**.
- What the agent knows about the environment.
- The actions that the agent can take.

Define an **ideal rational agent** : *For each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

PS. Doing actions in order to obtain useful information is an important part of rationality.

DOCTOR FUN

26 Sep 96



In the world of the future, "intelligent agents" will waste hours watching television for us.

The ideal mapping from percept sequences to actions

A **mapping from percept sequences to actions :**

A table of the actions an agent may take in response to each possible percept sequence.

Ideal mappings describe ideal agents:

Specifying which action an agent ought to take in response to any given percept sequence provides a design for an ideal agent.

Percept x	Action z
1.0	1.000000000000000
1.1	1.048808848170152
1.2	1.095445115010332
1.3	1.401754255099138
1.4	1.183215956619923
1.5	1.224744871391589
1.6	1.264911064067352
1.7	1.303840481040530
1.8	1.341640786499874
1.9	1.378404875209022
.	.
.	.
.	.

```
function SQRT(x)
    $z \leftarrow 1.0$   /*initial guess */
    repeat until $|z^2-x| < 10^{-15}$
        $z \leftarrow z-(z^2-x)/(2z)$
    end
    return $z$
```

Autonomy

An agent is called an **autonomous agent** if its behavior is based on both its own **experience** and the **built-in knowledge** used in constructing the agent.

- A system is to the extent that its behavior is determined by its own experience.
- But, it would be reasonable to provide an artificial intelligent agent with some initial knowledge as well as an ability to learn.
- A truly autonomous intelligent agent should be able to operate successfully in a variety of environments, given sufficient time to adapt.

Structure of Intelligent Agents

The job of AI is to design the **agent program**: a function that implements the agent mapping from percepts to actions.

The program will run on some sort of **architecture**, so:

agent = architecture + program

Makes the percepts from the sensors available to the program, runs the program, and feeds the program‘s action choices to effectors.

In designing an agent, we should have a clear idea of:

- the complexity relationship among the behavior of the agent,
- the percept sequence generated by the environment, and
- the goals the agent is supposed to achieve.

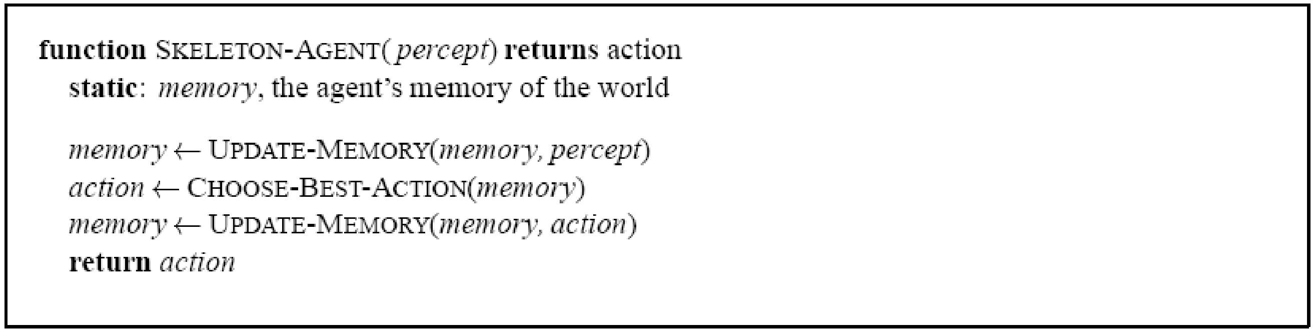
Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

Figure 3: Examples of agent types and their PAGE description.

Agent programs

※**A Skeleton agent** :

- **The memory** is a data structure updated as new percepts arrive. It is the buildup of percept sequences.
- **Performance measure** is not part of the skeleton program. It is applied externally.



```
function SKELETON-AGENT(percept) returns action
   static: memory, the agent's memory of the world

   memory ← UPDATE-MEMORY(memory, percept)
   action ← CHOOSE-BEST-ACTION(memory)
   memory ← UPDATE-MEMORY(memory, action)
   return action
```

Figure 4: A skeleton agent. On each invocation, the agent's memory is updated to reflex the new percept, the best action is chosen, and the fact that the action was taken is also stored in memory. The memory percepts from one invocation to the next.

Why not just look up the answer?

A simple **TABLE-DRIVEN-AGENT** (e.g. the one in ***Figure 5***), although *does* achieve what we want, may have the drawbacks listed below:

- The table might be huge, or virtually impossible to implement.
- The table might take a long time to build.
- The agent will have no autonomy at all, and might be unable to cop with the environment changes.
- Even if there is a learn mechanism, the agent might take forever to learn the entire table(huge) entries.

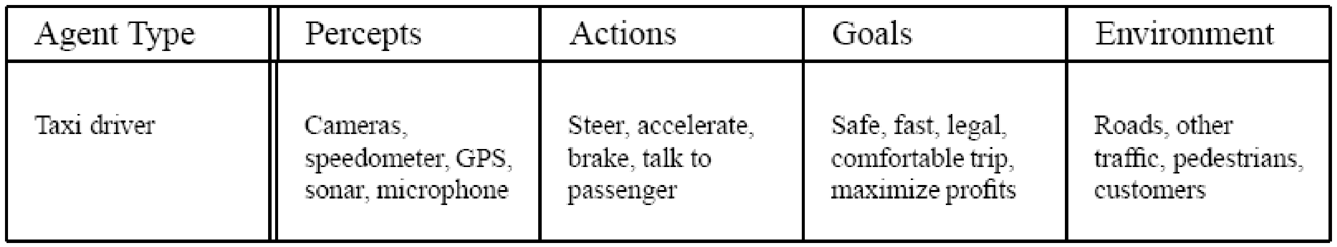
```
function TABLE-DRIVEN-AGENT(percept) returns action
  static: percepts, a sequence, initially empty
          table, a table, indexed by percept sequences, initially fully specified

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

Figure 5: An agent based on a prespecified lookup table. It keeps track of the percept sequence and just looks up the best action.

An example

Consider the job of designing an automated taxi driver, the percepts, actions, goals and environments for the taxi are in (***Figure 6***).

Considering implementing the mapping from percepts to action using four types of agent programs:



Agent Type	Percepts	Actions	Goals	Environment
Taxi driver	Cameras, speedometer, GPS, sonar, microphone	Steer, accelerate, brake, talk to passenger	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers

Figure 6: The taxi driver agent type.

※ Simple reflex agents

- It is impossible to use explicit lookup table.
- Using **condition-action rule**(or **situation-action rule**) is more appropriate. For example:

  if *the car in front is braking* **then** *initiate braking*
- The structure of a simple reflex agent and the agent program is in (***Figure 7***) and (***Figure 8***).
- The **INTERPRET-INPUT** function generates an abstracted description of the current state from the percept.
- The **RULE-MATCH** function returns the first rule in the set of rules that matches the given state description.

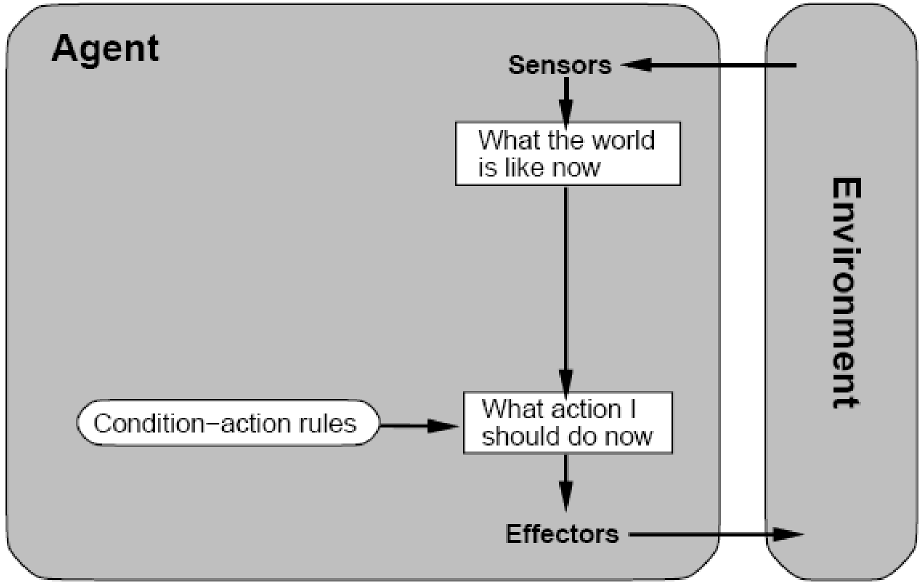


Figure 7: Schematic diagram of a simple reflex agent.

```
function SIMPLE-REFLEX-AGENT(percept) returns action
    static: rules, a set of condition-action rules

    state ← INTERPRET-INPUT(percept)
    rule ← RULE-MATCH(state, rules)
    action ← RULE-ACTION[rule]
    return action
```

Figure 8: A simple reflex agent. It works by finding a rule whose condition matches the current situation (as defined by the percept) and then doing the action associated with that rule.@

※Agents that keep track of the world

- An agent may need to maintain some sort of **internal state** in order to choose an appropriate action.
- In order to update the internal state, two kinds of knowledge need to be encoded in the agent program:
    1. How the world evolves independently of the agent - e.g. an overtaking car is closer than it was one minute ago.
    2. How the agent‘s own actions affect the world.
- The structure of a simple reflex agent and the agent program is in (***Figure 9***) and (***Figure 10***).
- The **UPDATE-STATE** function is responsible for creating the new internal state description.

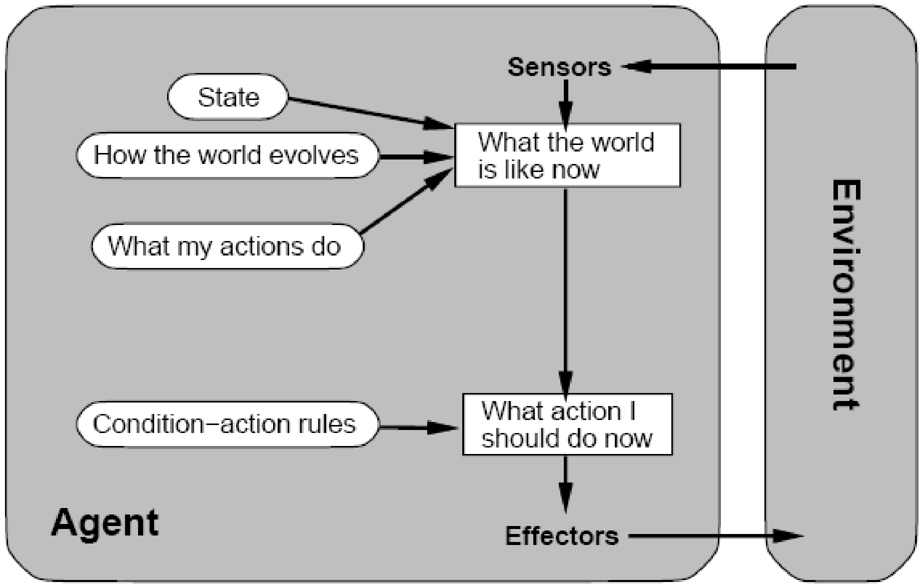
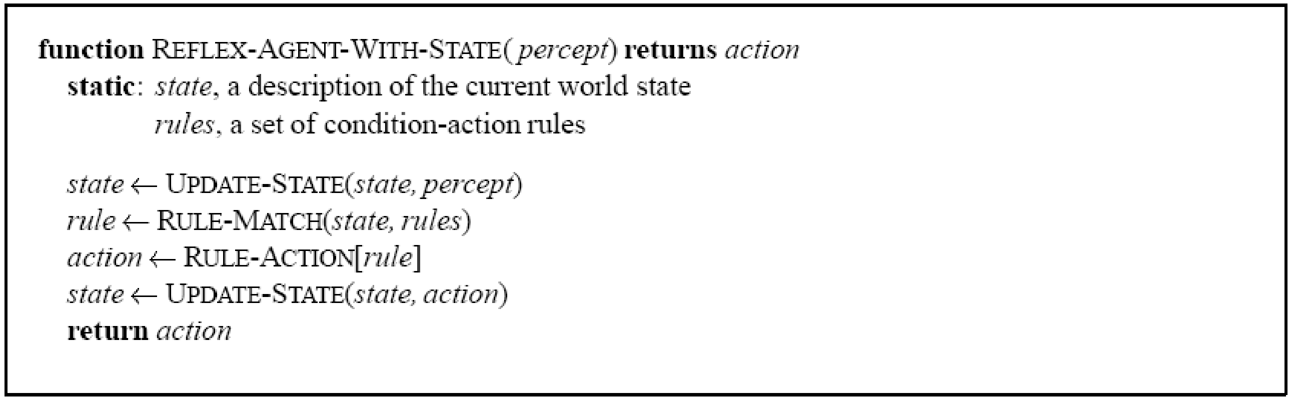


Figure 9: A reflex agent with internal state.



```
function REFLEX-AGENT-WITH-STATE(percept) returns action
   static: state, a description of the current world state
           rules, a set of condition-action rules

   state ← UPDATE-STATE(state, percept)
   rule ← RULE-MATCH(state, rules)
   action ← RULE-ACTION[rule]
   state ← UPDATE-STATE(state, action)
   return action
```

Figure 10: A reflex agent with internal state. It works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state) and then doing the action associated with that rule.

※Goal-based agents

- As well as a current state description, an agent also needs some sort of **goal** information.

- The **goal** information is combined with the **information about the results of possible actions** to choose actions in order to achieve the specified goal.

- **Search** and **planning** techniques are used to find **action sequences** that will achieve the agent‘s goals.

- **Goal** information provide more than condition-action rules do in two aspects:

1. What happen if the agent takes an action?
2. Will that action make the agent approach the goal?

- A Goal-based agent is **less efficient --** e.g. on seeing the brake lights of the car in front, it will reason that *the car in front will slow down and the action that will achieve the goal of not hitting other cars is to brake*.
- A Goal-based agent is, however, **more flexible --** e.g. it will be more flexible with respect to reaching different goals.
- The structure of a goal-based agent is shown in (***Figure 11***).

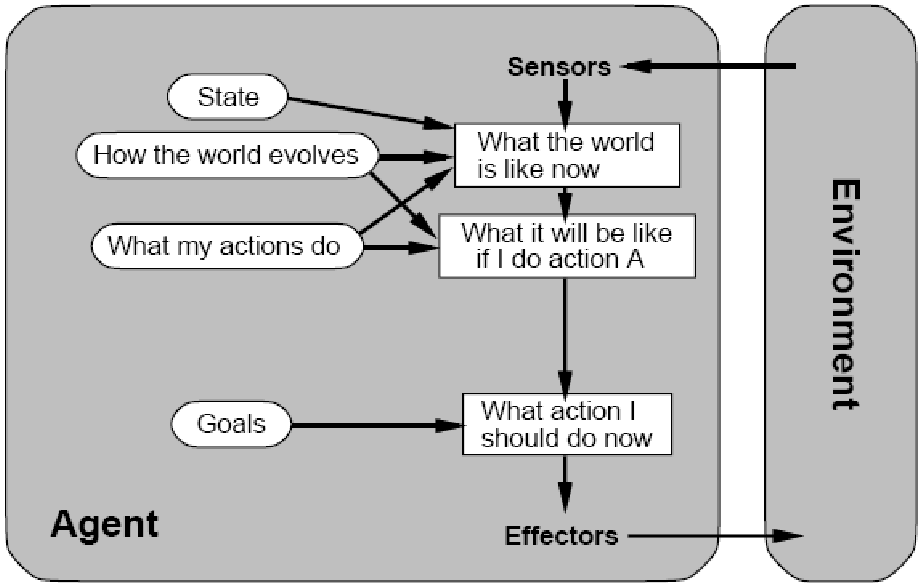

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Effectors
Environment

Figure 11: An agent with explicit goals.

※Utility-based agents

- Goals alone are not really enough to **generate high-quality** behavior, especially when there are many **action sequences** that will make the agent achieve the goal.
- We need to measure **how happy** the agent is when achieving the goal, not just **happy** or **not happy**.
- **Utility** is used to measure if one **world state** is preferred to another in achieving the goal.
- **Utility** is a function that maps a state onto a real number, which describes the associated degree of happiness, i.e. the **performance measure**.

- **Utility** functions also allow rational decisions :
  1. When there are **conflicting** goals, only some of which can be achieved,
  2. When there are several goals that the agent can aim for, none of which can be achieved with **certainty**.
- An agent that possesses an *explicit* **utility** function may have to compare the utilities achieved by different courses of actions before making any rational decisions.
- The structure of a utility-based agent is shown in (***Figure 12***).

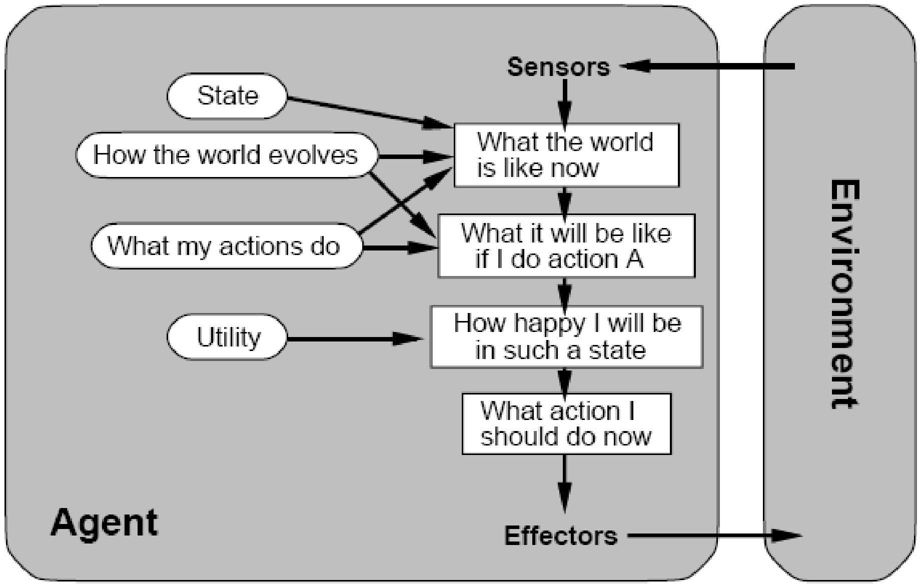


Figure 12: A complete utility-based agentm.

Environments

How to couple an agent to an environment?

Properties of environments

※**Accessible** vs. **inaccessible**

An environment is **effectively accessible** if the sensors detect all aspects that are relevant to the choice of action.

● In an accessible environment, an agent need not maintain any internal state to keep track of the world.

※**Deterministic** vs. **nondeterministic**

An environment is **deterministic** if the next state is completely determined by the current state and the actions selected by the agents.

- If the environment is inaccessible, then it may appear to be nondeterministic to the agent.
- It is often more objective to think of an environment as deterministic or nondeterministic *from the point of view of an agent*.

※**Episodic** vs. **nonepisodic**

In an **episodic** environment, an agent‘s experience is divided into "***episodes***". Each episode consists of the agent perceiving and then acting.

- An episode does not depend on what actions occur in the previous episodes.

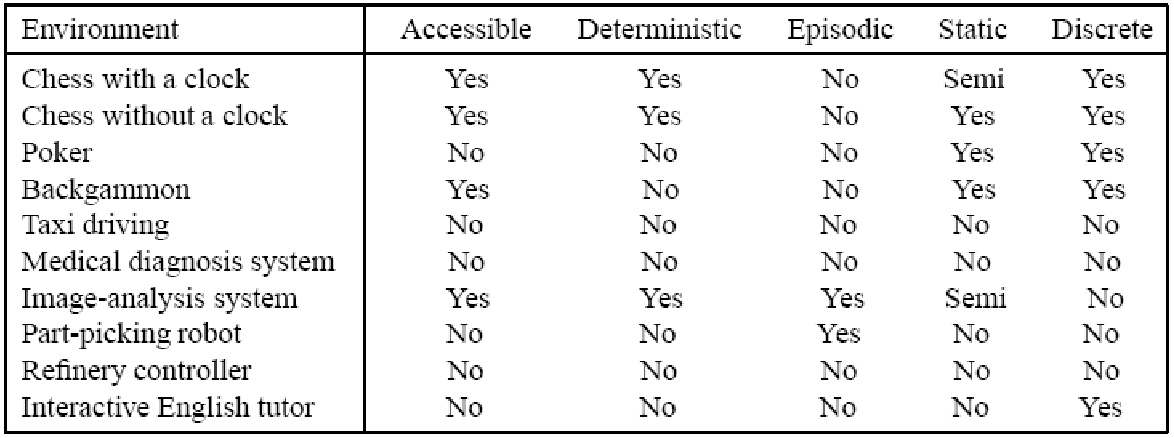
※**Static** vs. **dynamic**

A **static** environment will not change while an agent is deliberating, whereas a **dynamic** one does.

- Agents need not to be worried and keep looking at the static world while they are deciding on actions.
- In a **static** environment, if the agent‘s score of performance changes with the passage of time, then the environment is **semidynamic** to the agent.

※**Discrete** vs. **continuous**

A **discrete** environment allows a limited number of distinct, clearly defined percepts and actions.

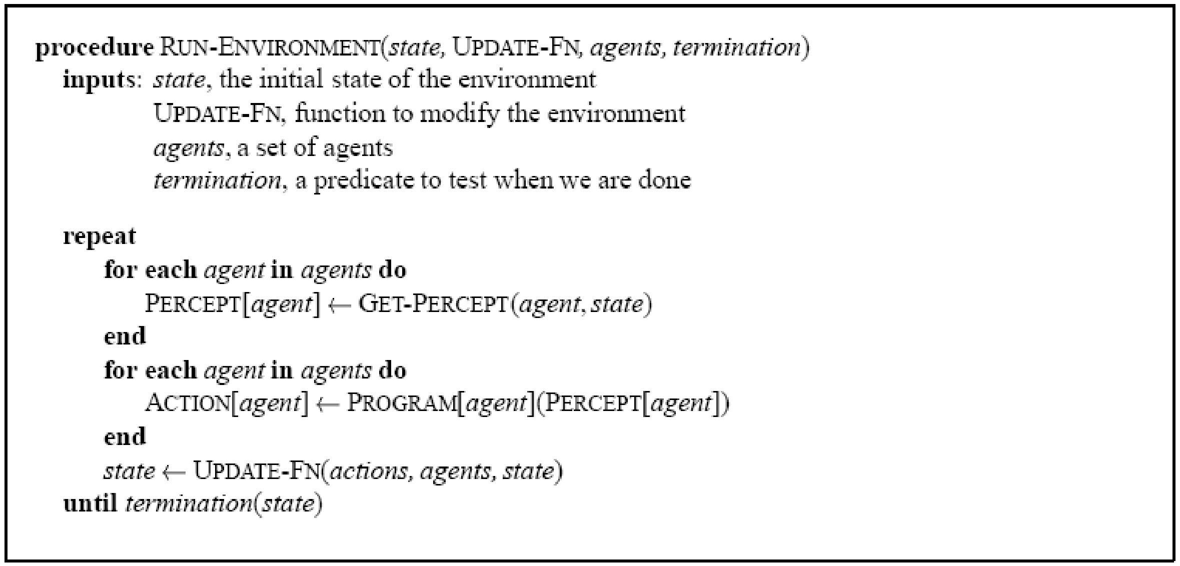


Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

Figure 13: Examples of environments and their characteristics.

Environment programs

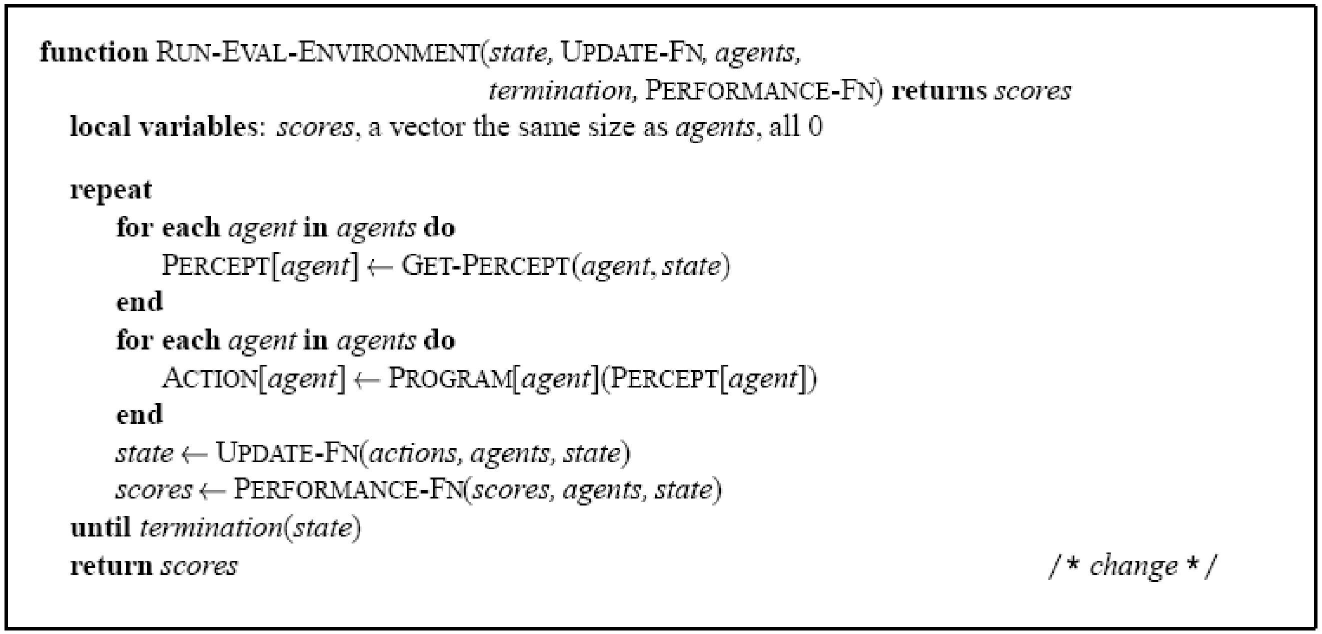
An environment (simulating program) is defined by the **initial state** and the **update function**. *It takes agents as input and arranges to repeatedly give each agent the right percepts and receive back actions.*



```
procedure RUN-ENVIRONMENT(state, UPDATE-FN, agents, termination)
   inputs: state, the initial state of the environment
           UPDATE-FN, function to modify the environment
           agents, a set of agents
           termination, a predicate to test when we are done

   repeat
      for each agent in agents do
         PERCEPT[agent] ← GET-PERCEPT(agent, state)
      end
      for each agent in agents do
         ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
      end
      state ← UPDATE-FN(actions, agents, state)
   until termination(state)
```

Figure 14: The basic environment simulator program. It gives each agent its percept, gets an action from each agent, and then updates the environment.

Performance measure code can be inserted into an environment program. It applies a performance measure to each agent and returns a list of the resulting scores.



```
function RUN-EVAL-ENVIRONMENT(state, UPDATE-FN, agents,
                              termination, PERFORMANCE-FN) returns scores
  local variables: scores, a vector the same size as agents, all 0

  repeat
     for each agent in agents do
        PERCEPT[agent] ← GET-PERCEPT(agent, state)
     end
     for each agent in agents do
        ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
     end
     state ← UPDATE-FN(actions, agents, state)
     scores ← PERFORMANCE-FN(scores, agents, state)
  until termination(state)
  return scores                                          /* change */
```

Figure 15: An environment simulator program that keeps track of the performance measure for each agent.

※Performance of an agent in an environment class

Usually, an agent is designed to work in an **environment class**, a whole set of different environments.

- ***An agent‘s performance, therefore, should be measured on an average over the environment class.***