Artificial Intelligence

By: Mohit Goel
Assistant Professor
Email id- mohit.16907@lpu.co.in
Block-36, Room No.-203

Task Domains of Artificial Intelligence/ Applications of AI

Mundane Tasks

Perception

- Vision
- Speech

Natural language

- Understanding
- Generation
- Translation

Common sense reasoning

Robot control

Task Domains of Artificial Intelligence/ Applications of AI

Expert Tasks

Engineering

- Design
- Fault finding
- Manufacturing planning

Scientific analysis

Medical diagnosis

Financial analysis

Task Domains of Artificial Intelligence/ Applications of AI

Formal Tasks

Games

- Chess
- Backgammon
- Checkers –Go

Mathematics

- Geometry
- Logic
- Integral calculus
- Proving properties of programs

AI Problem

Before embarking on a study of specific AI problems and solution techniques, it is important at least to discuss, if not to answer, the following four questions:

1. What are our underlying assumptions about intelligence?
2. What kinds of techniques will be useful for solving AI problems?
3. At what level of detail, if at all, are we trying to model human intelligence?
4. How will we know when we have succeeded in building an intelligent program?

Knowledge:

One of the few hard and fast results to come out of the first three decades of AI research is that *intelligence requires knowledge*. To compensate for its one overpowering asset, indispensability, knowledge possesses some less desirable properties, including:

1. It is voluminous.
2. It is hard to characterize accurately.
3. It is constantly changing.
4. It differs from data by being organized in a way that corresponds to the ways it will be used.

An AI technique is a method that exploits knowledge
that is represented so that:

- The knowledge captures generalizations that share properties, are grouped together, rather than being allowed separate representation.
- It can be understood by people who must provide it—even though for many programs bulk of the data comes automatically from readings.
- In many AI domains, how the people understand the same people must supply the knowledge to a program.
- It can be easily modified to correct errors and reflect changes in real conditions.
- It can be widely used even if it is incomplete or inaccurate.
- It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must be usually considered.

Tic-Tac-Toe: Program 1

Data Structures

Board: A nine-element vector representing the board, where the elements of the vector correspond to the board positions as follows:

1	2	3
4	5	6
7	8	9

An element contains the value 0 if the corresponding square is blank, 1 if it is filled with an X, or 2 if it is filled with an 0.

Movetable: A large vector of 19,683 elements (3^9), each element of which is a nine-element vector. The contents of this vector are chosen specifically to allow the algorithm to work.

Tic-Tac-Toe: Program 1

The Algorithm

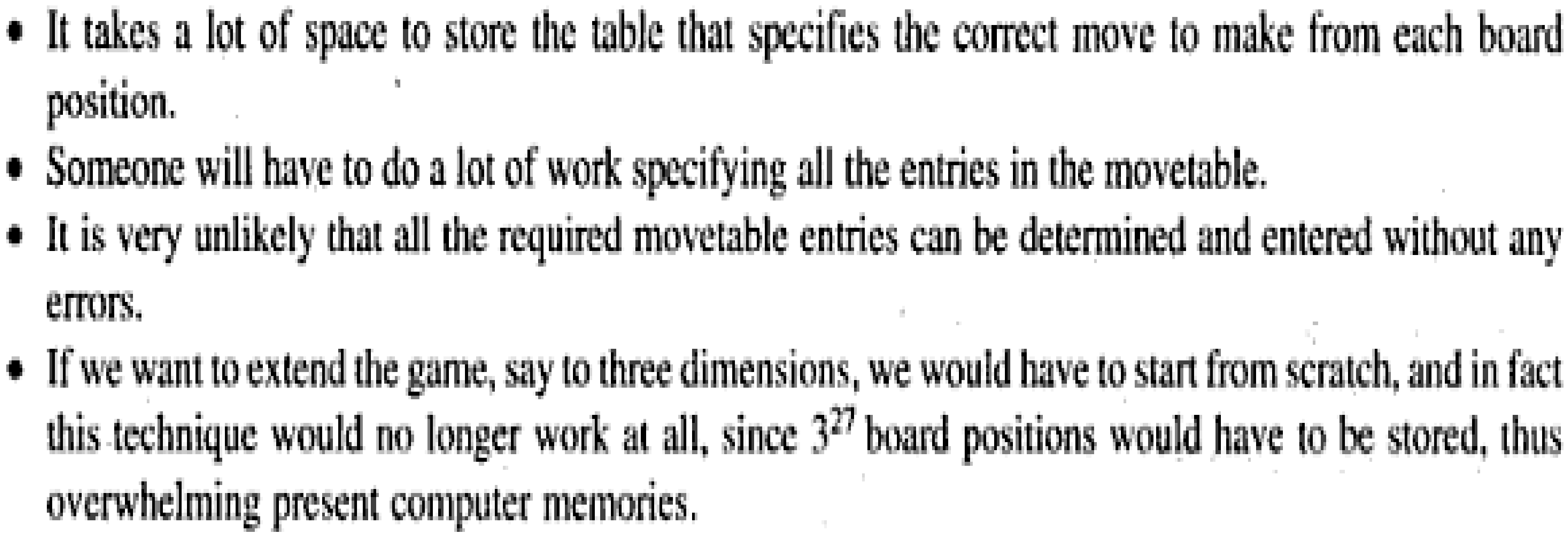
To make a move, do the following:

1. View the vector Board as a ternary (base three) number.
2. Convert it to a decimal number.
3. Use the number computed in step 1 as an index into Movetable and access the vector stored there.
4. The vector selected in step 2 represents the way the board will look after the move that should be made. So set Board equal to that vector.

Tic-Tac-Toe: Program 1

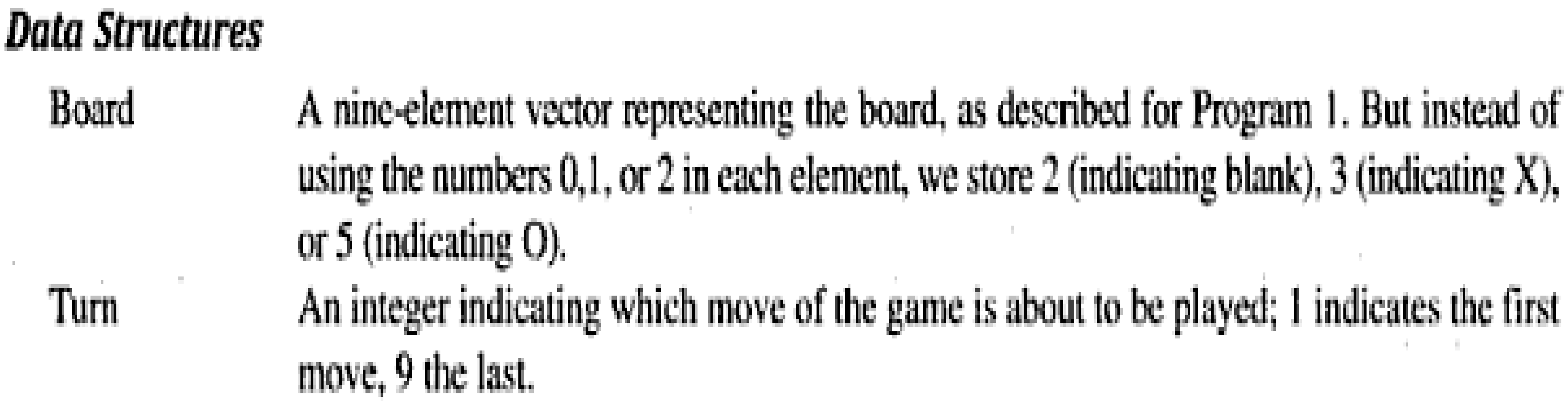
Comments

This program is very efficient in terms of time. And, in theory, it could play an optimal game of tic-tac-toe. But it has several disadvantages:



- It takes a lot of space to store the table that specifies the correct move to make from each board position.
- Someone will have to do a lot of work specifying all the entries in the movetable.
- It is very unlikely that all the required movetable entries can be determined and entered without any errors.
- If we want to extend the game, say to three dimensions, we would have to start from scratch, and in fact this technique would no longer work at all, since 3^{27} board positions would have to be stored, thus overwhelming present computer memories.

Tic-Tac-Toe: Program 2



Data Structures

Board	A nine-element vector representing the board, as described for Program 1. But instead of using the numbers 0,1, or 2 in each element, we store 2 (indicating blank), 3 (indicating X), or 5 (indicating O).
Turn	An integer indicating which move of the game is about to be played; 1 indicates the first move, 9 the last.

Tic-Tac-Toe: Program 2

Tic-Tac-Toe: Program 2

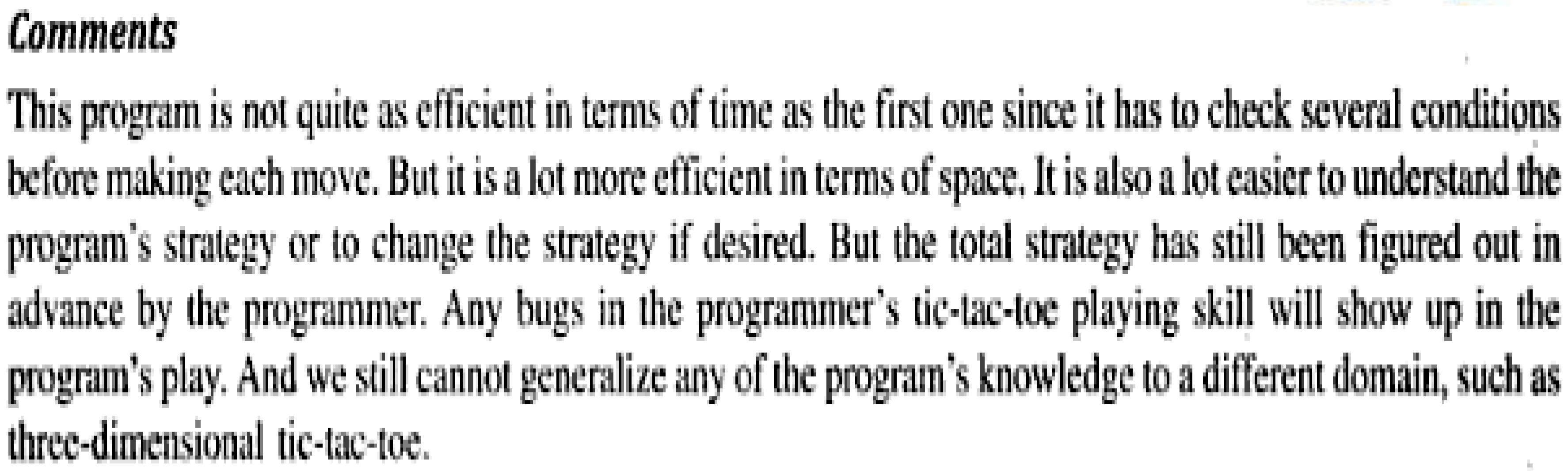
The Algorithm

The main algorithm uses three subprocedures:

Make 2	Returns 5 if the center square of the board is blank, that is, if Board[5] = 2. Otherwise, this function returns any blank noncorner square (2, 4, 6, or 8).
Posswin(*p*)	Returns 0 if player *p* cannot win on his next move; otherwise, it returns the number of the square that constitutes a winning move. This function will enable the program both to win and to block the opponent's win. Posswin operates by checking, one at a time, each of the rows, columns, and diagonals. Because of the way values are numbered, it can test an entire row (column or diagonal) to see if it is a possible win by multiplying the values of its squares together. If the product is 18 ($3 \times 3 \times 2$), then X can win. If the product is 50 ($5 \times 5 \times 2$), then O can win. If we find a winning row, we determine which element is blank, and return the number of that square.
Go(*n*)	Makes a move in square *n*. This procedure sets Board[*n*] to 3 if Turn is odd, or 5 if Turn is even. It also increments Turn by one.

Tic-Tac-Toe: Program 2

Tic-Tac-Toe: Program 2



Comments

This program is not quite as efficient in terms of time as the first one since it has to check several conditions before making each move. But it is a lot more efficient in terms of space. It is also a lot easier to understand the program's strategy or to change the strategy if desired. But the total strategy has still been figured out in advance by the programmer. Any bugs in the programmer's tic-tac-toe playing skill will show up in the program's play. And we still cannot generalize any of the program's knowledge to a different domain, such as three-dimensional tic-tac-toe.

Tic-Tac-Toe: Program 3

This program is identical to Program 2 except for one change in the representation of the board. We again represent the board as a nine-element vector, but this time we assign board positions to vector elements as follows:

8	3	4
1	5	9
6	7	2

Notice that this numbering of the board produces a magic square: all the rows, columns, and diagonals sum up to 15. This means that we can simplify the process of checking for a possible win. In addition to marking the board as moves are made, we keep a list, for each player, of the squares in which he or she has played. To check for a possible win for one player, we consider each pair of squares owned by that player and compute the difference between 15 and the sum of the two squares. If the difference is not positive or if it is greater than 9, then the original two squares were not collinear and so can be ignored. Otherwise, if the square representing the difference is blank, a move there will produce a win. Since no player can have more than four squares at a time, there will be many fewer squares examined using this scheme than there were using the more straightforward approach of Program 2. This shows how the choice of representation can have a major impact on the efficiency of a problem-solving program.

Tic-Tac-Toe: Program 3

Rational Agents in AI

What is an Agent?

- Perceives its environment through sensors
- Acts upon it through actuators
- Humans perceive through 5 senses and act through body parts.
- AI agents perceive through sensors and act through actuators.

What is an Intelligent/Rational Agent?

Acts in way that is expected to maximize to its performance measure

Provided - what it perceived, built-in knowledge

PEAS – Performance + Environment + Actuators + Sensors

Self-Driving cars:

- P – Safety, time legal drive, comfort
- E – Roads, other cars, pedestrians, road signs
- A – Steering, accelerator, brake, signal, horn
- S – Camera, GPS, engine sensors, etc.

PEAS stands for *Performance measure*, *Environment*, *Actuator*, *Sensor*.

1.Performance Measure: Performance measure is the unit to define the success of an agent. Performance varies with agents based on their different precept.

2.Environment: Environment is the surrounding of an agent at every instant. It keeps changing with time if the agent is set in motion. There are 5 major types of environments:

1. Fully Observable & Partially Observable
2. Episodic & Sequential
3. Static & Dynamic
4. Discrete & Continuous
5. Deterministic & Stochastic

3. Actuator: Actuator is a part of the agent that delivers the output of an action to the environment.

4. Sensor: Sensors are the receptive parts of an agent which takes in the input for the agent.

Environment in AI

Environment

An environment in artificial intelligence is the surrounding of the agent. The agent takes input from the environment through sensors and delivers the output to the environment through actuators.

Fully Observable vs Partially-Observable

•When an agent sensor is capable to sense or access the complete state of an agent at each point in time, it is said to be a fully observable environment else it is partially observable.

•Maintaining a fully observable environment is easy as there is no need to keep track of the history of the surrounding.

•Tic Tac Toe/ Chess is a example of fully observable Whereas automated cars are an example of partially observable AI environments.

Deterministic vs Stochastic

When a uniqueness in the agent’s current state completely determines the next state of the agent, the environment is said to be deterministic.

The stochastic environment is random in nature which is not unique and cannot be completely determined by the agent.

Tic-Tac Toe is example of deterministic where as LUDO is example of stochastic problem.

Discrète verses Continuos AI Environnements

Discrete AI environments are the ones where a definite set of possibilities can lead the agent towards the required outcome or goal. In continuous AI environments the systems depend on the fast changing, unknown data sources.

Chess is an example of a discrete AI environment. Whereas, drone systems, automated cars and multi-player video games form examples of continuous AI environments as the environment where they are employed keep on changing rapidly.

Episodic vs. Sequential

In Episodic AI environment a series of one-shot actions are required. These actions are taken on the basis of current percept of the environment only. Whereas, in Sequential AI environment an agent works on the basis of past experiences in order to determine the next best action to be taken in order to accomplish the goal.

Sequential if current decisions affect future decisions, or rely on previous ones

A support bot (agent) answer to a question and then answer to another question and so on. So each question-answer is a single episode.

Non-episodic environment: chess game

Static vs. Dynamic:

Static AI environments work on the basis of knowledge sources which do not change frequently over a period of time. Static environments form an idle environment for most agents. Whereas, when the data source keeps on changing frequently over a period of time it is referred as dynamic AI environment. Agents in dynamic environment require regular training to adapt to the environment.

For example, speech analysis or empty houses are examples of static AI environments. Whereas, the visuals captured by a drone system changes frequently hence they are an example of dynamic AI environment.

Single Agent and Multiple Agent

An environment consisting of only one agent is said to be a single-agent environment.

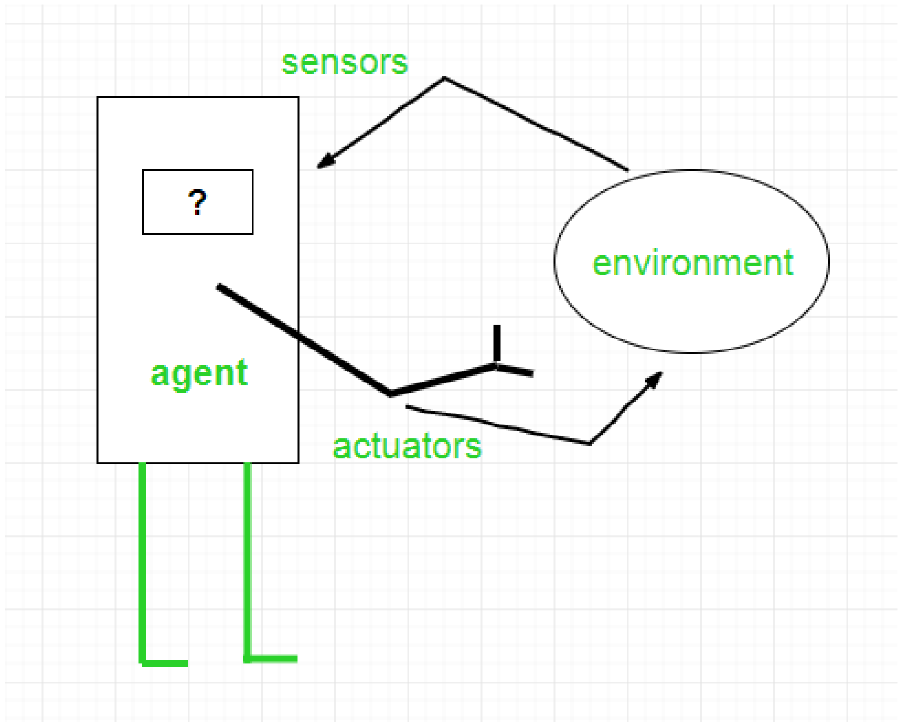
An environment involving more than one agent is a multi-agent environment.
The game of football is multi-agent as it involves 11 players in each team.

Agents in AI

An agent is anything that can be viewed as :

•perceiving its environment through **sensors** and

•acting upon that environment through **actuators**



Types of Agents

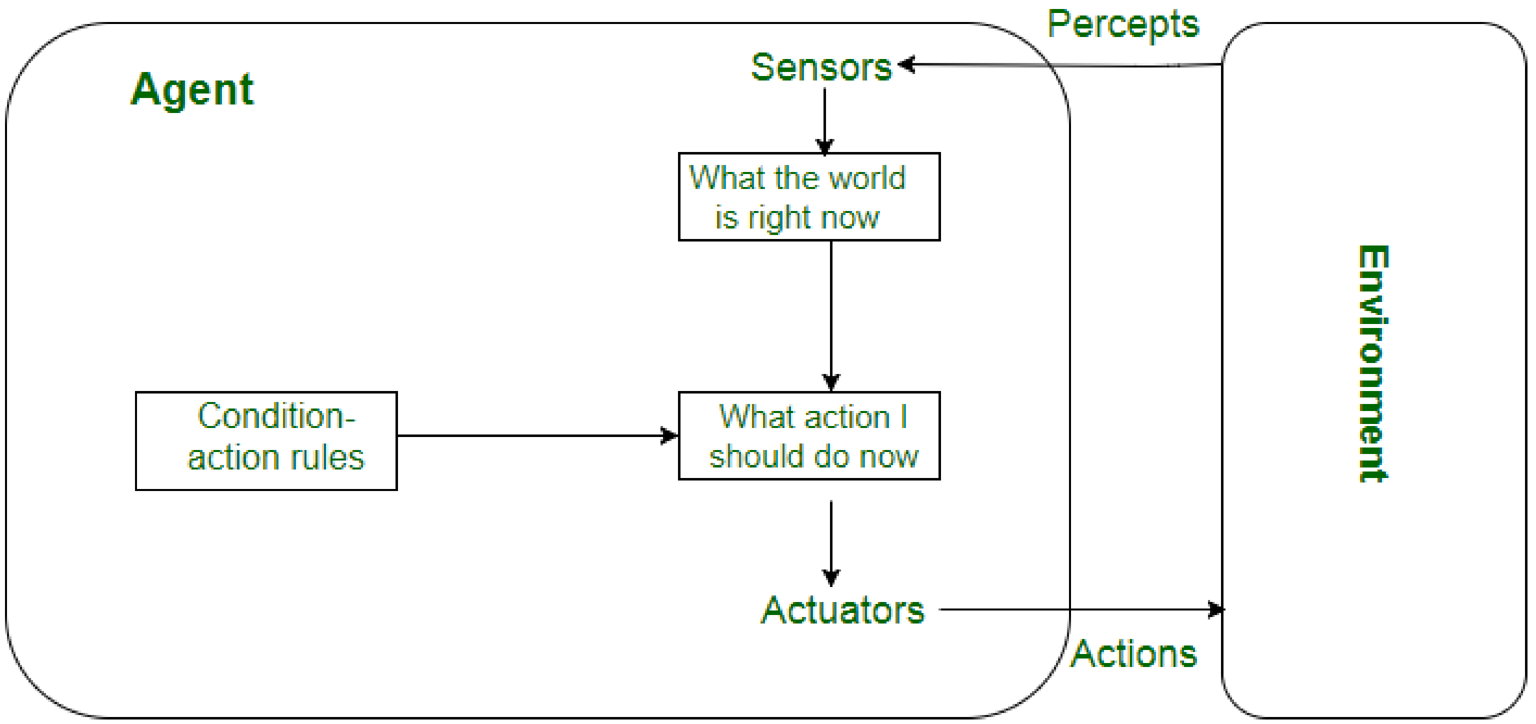
Agents can be grouped into different classes based on their degree of perceived intelligence and capability :

- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents
- Learning Agent

Simple reflex agents

- Simple reflex agents ignore the rest of the percept history and act only on the basis of the current percept.
- The agent function is based on the condition-action rule. A condition-action rule is a rule that maps a state i.e, condition to an action. If the condition is true, then the action is taken, else not.
- *This agent function only succeeds when the environment is fully observable.*

Simple reflex agents



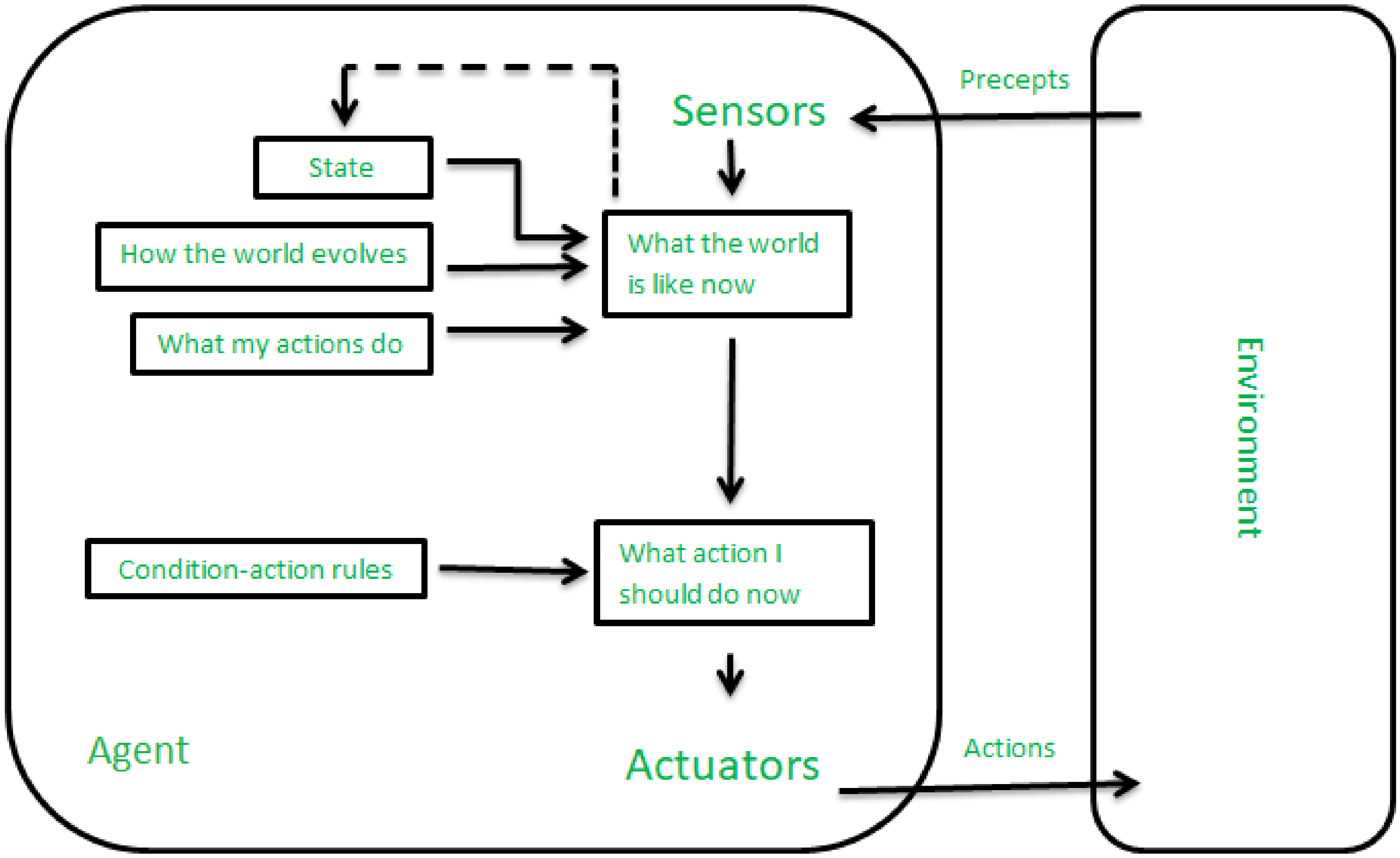
Model-based reflex agents

It works by finding a rule whose condition matches the current situation. A model-based agent can handle partially observable environments by the use of a model about the world. The agent has to keep track of the internal state which is adjusted by each percept and that depends on the percept history. The current state is stored inside the agent which maintains some kind of structure describing the part of the world which cannot be seen.

Updating the state requires information about :

- how the world evolves independently from the agent, and
- how the agent's actions affect the world.

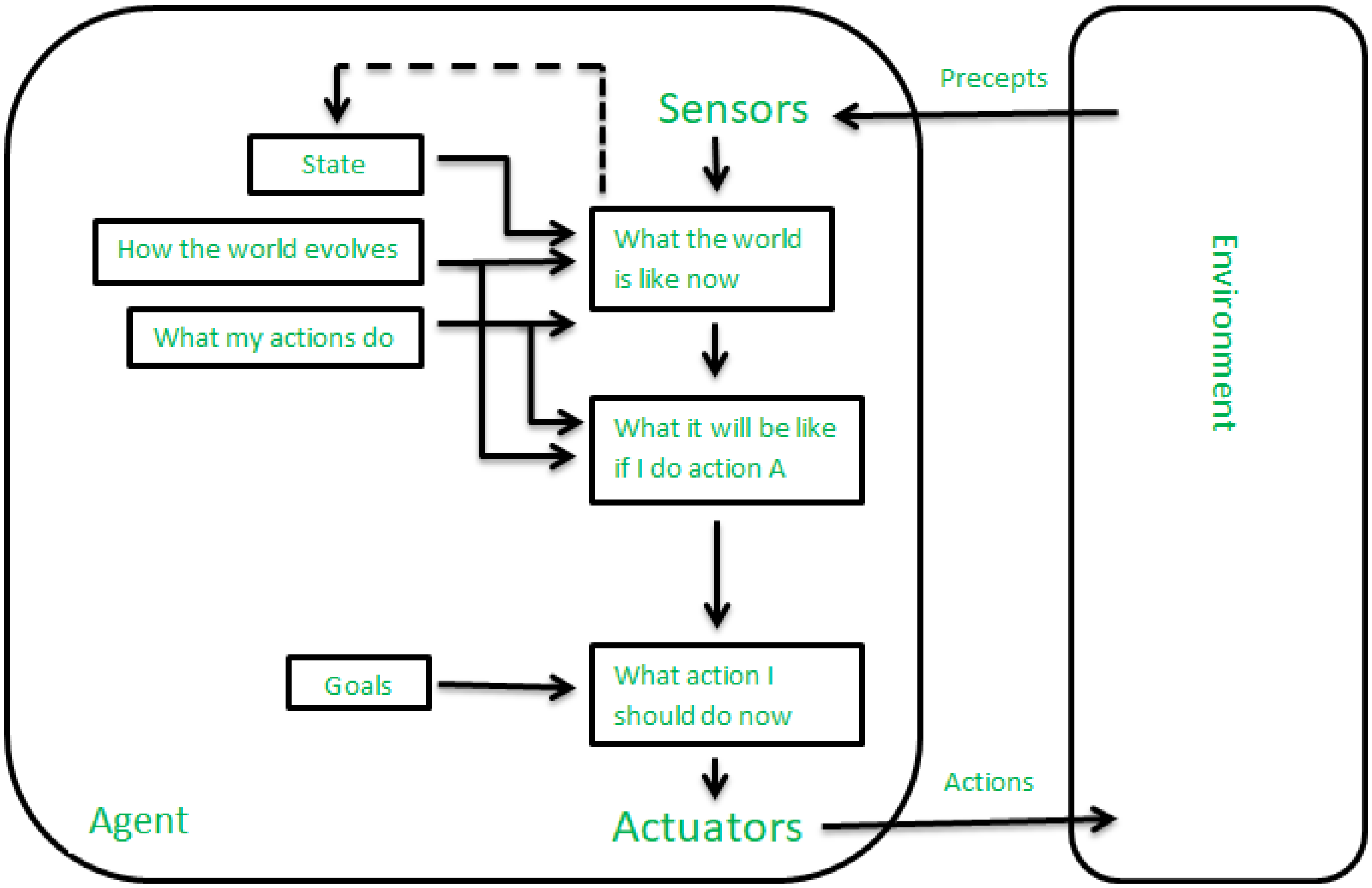
Model-based reflex agents



Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.

Goal-based agents

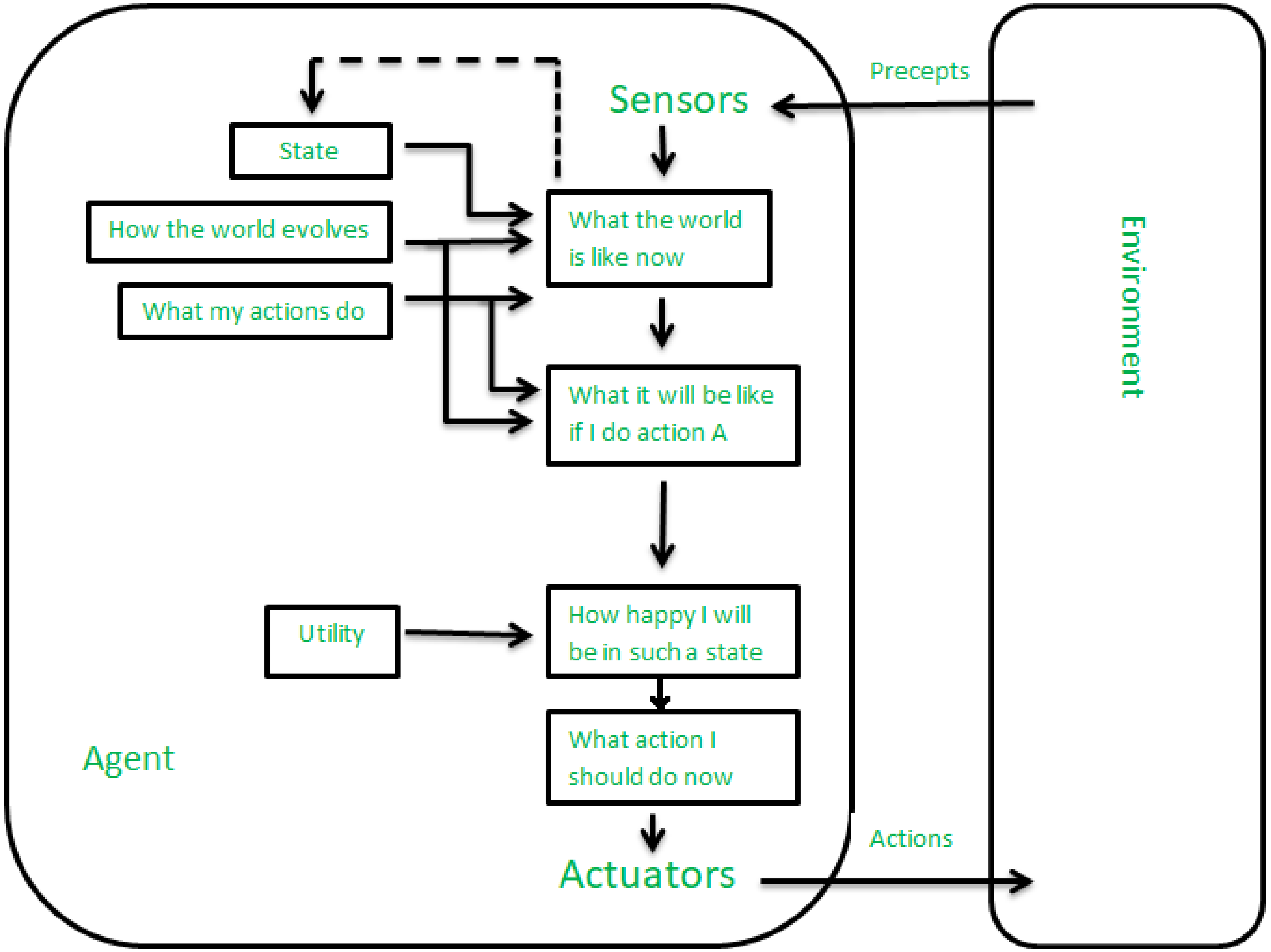


Utility-based agents

These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.

- *Utility-based agent act based not only goals but also the best way to achieve the goal.*
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals

Utility-based agents


Sensors
Precepts
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
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Actions
Environment

Learning Agents

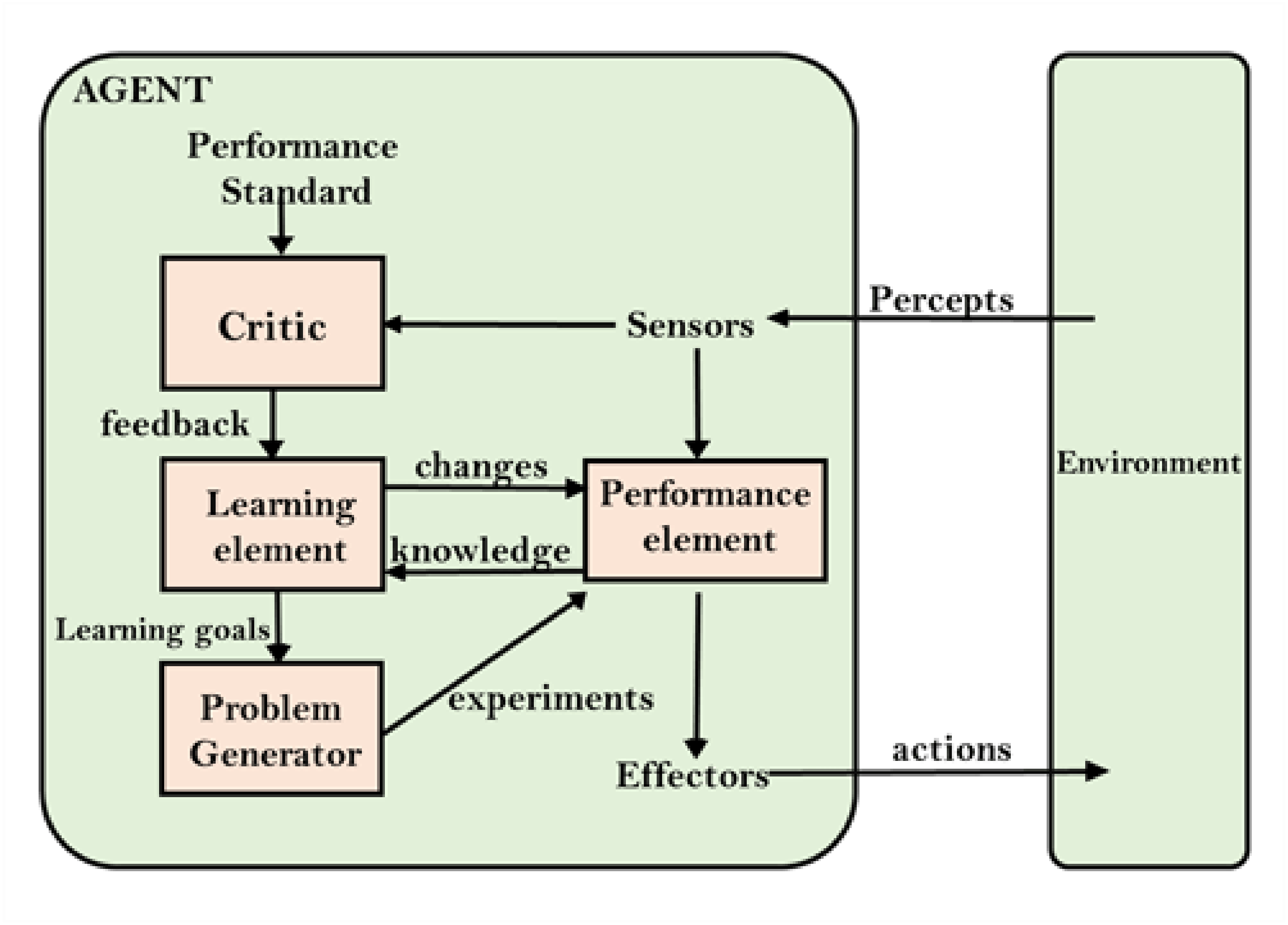
A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities. It starts to act with basic knowledge and then able to act and adapt automatically through learning.

A learning agent has mainly four conceptual components, which are:

- Learning element: It is responsible for making improvements by learning from environment
- Critic: Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
- Performance element: It is responsible for selecting external action
- Problem generator: This component is responsible for suggesting actions that will lead to new and informative experiences.

Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.

Learning Agents



Any Question?