LECTURE NOTES– MODULE I

Introduction to AI and Intelligent Agents

ECS302

ARTIFICIAL INTELLIGENCE

3/4 B.Tech B3,B12

PRANEEL ANKIREDDY
ASSISTANT PROFESSOR
DEPARTMENT OF CSE,GIT,GU

Module I Lecture Notes

Syllabus

Introduction to AI and Intelligent Agents: *What is AI? The foundations of artificial intelligence;* ***Intelligent Agents:*** *Agents and environments.* ***Good Behavior:*** *The concept of rationality, the nature of environments, the structure of agents.*

1. What is AI?

Artificial Intelligence is the branch of computer science concerned with making computers behave like humans. Major AI textbooks define artificial intelligence as "the study and design of intelligent agents," where an intelligent agent is a system that perceives its environment and takes actions which maximize its chances of success. John McCarthy, who coined the term in 1956, defines it as "the science and engineering of making intelligent machines, especially intelligent computer programs."

Definition:

There are two ideas in the definition.

- Intelligence [Ability to understand think & learn]
- Artificial device [Non-Natural]

The definitions of AI can be categorized into four approaches:

Systems that think like humans	*Systems that think rationally*
"The exciting new effort to make computers think … machines with minds, in the full and	"The study of mental faculties through the use of computer models."

literal sense." (Haugeland,1985)	(Charniak and McDermont,1985)
Systems that act like humans "The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil,1990)	*Systems that act rationally* "Computational intelligence is the study of designing intelligent agents." (Poole et al.,1998)

Thinking Humanly: The Cognitive modelling Approach

We can say that given program thinks like a human, we must determine how humans think. i.e., we need to get inside the actual working of human minds.
There are two ways to do this;

- ***Through Introspection (Trying to catch own Thoughts)***
- ***Through psychological experiments***

Allen Newell and Herbert Simon, who developed ***GPS*** (***General problem solver***) tried to trace the reasoning steps to traces of human subjects solving some problems.

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's working.

Acting Humanly: The Turing test Approach

The Turing Test, proposed by Alan Turing (195O), was designed to provide a satisfactory operational definition of intelligence. Here the computer is asking some questions by a human interrogator. The computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not.
The computer would need to possess the following capabilities:

- ***Natural language processing*:** Enable it to communicate successfully in English.
- ***Knowledge representation*:** Store what it knows or hears.
- ***Automated reasoning*:** Use the stored information to answer questions and draw new conclusions.
- ***Machine learning*:** To adapt to new circumstances and to detect and extrapolate patterns.
- ***Computer vision*:** To perceive objects.
- ***Robotics*:** To manipulate objects and move about.

Thinking Rationally: The "Laws of Thought" Approach

The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking," that is, irrefutable reasoning processes. His syllogisms provided patterns for argument structures that always yielded correct conclusions when given proper premises. For example,

"Socrates is a man; all men are mortal; therefore, Socrates is mortal."

These laws of thought were supposed to govern the operation of the mind; their study initiated the field called logic.

Acting Rationally: The Rational Agent Approach

An agent is just something that acts. But computer agents are expected to have other attributes that distinguish them from mere programs, such as operating under autonomous control, perceiving their environment, persisting over a prolonged time period, adapting to change, and being capable of taking on another's goals.

Agent behaviour is described by the "Agent function" that maps any given percept sequence to an agent.

[f: p* → A]

A *rational agent* is one that acts to achieve the best outcome. The right action is the one that will cause the agent to be the most successful.

1.1 Foundations of AI:

It depends on various disciplines that contributed different ideas as given below;

1.1.1 Philosophy:

- *Can formal rules be used to draw valid conclusions?*
- *How does the mental mind arise from a physical brain?*
- *Where does knowledge come from?*
- *How does knowledge lead to action?*

Aristotle was the first person to formulate a precise set of laws governing the mind's rational part. He developed an informal system of ***syllogisms*** for proper reasoning, which allowed one to generate conclusions with the given initial premises.

[Syllogism: It is a Logical Argument consisting of two premises and a conclusion]

- At that time, the study of human intelligence began with no formal expression.
- They Initiated the idea of mind as a machine and its internal operations

1.1.2 Mathematics:

- *What are the formal rules to draw valid conclusions?*
- *What can be computed?*
- *How do we reason with uncertain information*

Mathematics formalizes the three main areas of AI: *computation, logic*, and *probability*

Boole introduced formal language for making logical inference and used *formal logic* methods as Boolean Logic & Fuzzy Logic. At the same time, we consider these bases for most modern approaches that handle *uncertainty* in AI Applications. Besides logic and computation, another contribution of mathematics to AI is the theory of probability used to deal with uncertain measurements and incomplete theories.

1.1.3 Economics:

- *How should we make decisions to maximize payoff?*
- *How should we do this when others may not go along?*
- *How should we do this when the payoff may be far in the future?*

Most people think that economics is being money, but it will say that they study how people make choices that lead to preferred outcomes. So, the mathematical treatment of preferred outcome *"Utility"* was first formalized. Here they considered a "Decision Theory" that combines the Probability Theory & Utility Theory for making decisions.

1.1.4 Neuro Science:

- *How do brain process information?*

It is the study of the Nervous System, particularly the brain. More recent studies used to accurate sensors to correlate brain activity to human thought. This is done by monitoring individual neurons. As we know, there are 1000 times more neurons in a typical human brain than the gates in the CPU of a typical high-end computer. Moore's Law predicts that the CPU gate count will be equal to brains neuron count around 2020.

1.1.5 Psychology:

- *How do Humans and animals think and Act?*

The origins of scientific psychology are usually traced to the German physicist Hermann von Helmholtz's work and his student Wilhelm Wundt. He applied some scientific method to the study of human vision. He also used his view on humans' behaviourism movement and the brain's view as an Information-Processing-Device, Which is a principle characteristic of cognitive psychology.

1.1.6 Computer Engineering:

- *How can we build an efficient computer?*

For artificial intelligence to succeed, we need two things: intelligence and an artifact. AI also requires software side of Computer Science, which has supplied the operating systems, Programming Languages, Tools, etc…

1.1.7 Control Theory and Cybernetics:

- *How can artifacts operate under their own control?*

The artifacts adjust their actions:

- *To do better for the environment over time*
- *Based on an objective function and feedback from the environment*

Machines can modify their behaviour in response to the environment ***(Sense/Action)***. These machines are called self-controlling machines. The goal is to build systems that transition from initial state to goal state with minimum energy. Examples of self-regulating feedback control systems include the

steam engine, created by James Watt and the thermostat, invented by Colnelis Drebbel, who also invented the submarine.

1.1.8 Linguistics:

- How does language relate to thought?

Modem linguistics and AI, then, were "born" at about the same time, and grew up together, intersecting in a hybrid field called computational linguistics or natural language processing. The speech demonstrates so much of human intelligence. Children can create sentences they have never heard before. So language and thought are believed to be tightly intertwined.

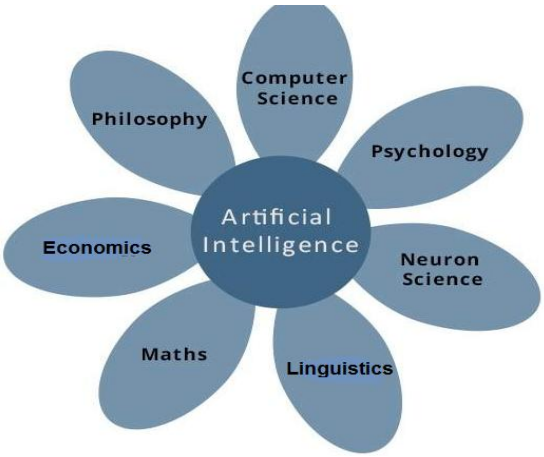

Fig 1.1: Pictorial Representation of AI Foundations

1.2 Goals of AI:

- To understand human intelligence better. We test theories of human intelligence by writing programs which emulate it.
- To create useful smart programs able to do task that would normally require a human expert.

****** EXTRA MATERIAL [start]******

1.3 History of AI:

1.3.1 The Generation of AI (1943-1955)

There were several early examples of work that can be characterized as AI, but it was Alan Turing who first articulated a complete vision of AI in his 1950 article "Computing Machinery and Intelligence." There, he introduced the Turing test, machine learning, etc., in his article in 1950.

1.3.2 Birth of AI: (1956)

In 1956 John McCarthy coined the term Artificial Intelligence. Demonstration of the first running AI program at Carnegie Mellon University.

1.3.3 Early enthusiasm, great expectations (1952-1969)

General problem Solver (**GPS**) was a computer program created in 1957 by Allen Newell and Herbert Simon to build a Universal problem solver Machine. The order considers as sub-goals and possible actions similar to humans which approached the same problems. John McCarthy introduced later LISP in 1958.

1.3.4 A Dose of Reality: (1966-1973)

There is an ability to do different things that will be increased rapidly by the visible range of problems.

1.3.5 AI becomes an Industry: (1980)

In 1981, the Japanese announced the "Fifth Generation" project, a 10-year plan to build intelligent computers running Prolog. Overall, the A1 industry boomed from a few million dollars in 1980 to billions of dollars in 1988.

1.3.6 The return of neural networks (1986-present)

In the mid-1980s, four different groups reinvented the back-propagation learning algorithm first founded in 1969 by Bryson and Ho. Psychologists including David Rumelhart and Geoff Hinton, continued the study of neural-net models of memory.

1.3.7 A1 becomes a science (1987-present)

In recent years, approaches based on Hidden Markov models ***(HMMs)*** have come to dominate the area. Speech technology and the related field of handwritten character recognition are already making the transition to widespread industrial and consumer applications.

1.3.8 The emergence of intelligent agents (1995-present)

One of the most important environments for intelligent agents is the Internet. AI systems have become so common in web-based applications.

1.4 Typical AI problems:

While studying the typical range of tasks that we might expect an "*intelligent entity*" to perform, we need to consider both "*common-place*" tasks as well as expert tasks.

- ***Common-place tasks include:***
  - ✓ *Recognizing people, objects.*
  - ✓ *Communicating (through natural language).*
  - ✓ *Navigating around obstacles on the streets*

- ***Expert tasks include:***
  - ✓ *Medical diagnosis.*
  - ✓ *Mathematical problem solving*
  - ✓ *Playing games like chess*

These tasks cannot be done by all people, and can only be performed by skilled specialists. Now, which of these tasks are easy and which ones are hard? Clearly, tasks of the first type are easy for humans to perform, and almost all are able to master them. The second range of tasks requires skill development and/or intelligence, and only some specialists can perform them well. However, when we look at what computer systems have been able to achieve to date, we see that their achievements include performing sophisticated tasks like medical diagnosis, performing symbolic integration, proving theorems and playing chess.

On the other hand, it has proved to be very hard to make computer systems perform many routine tasks that all humans and a lot of animals can do. Examples of such tasks include navigating our way without running into things, catching prey and avoiding predators. Humans and animals are also capable of interpreting complex sensory information. We are able to recognize objects and people from the visual image that we receive.

1.5 Importance of AI:

Artificial intelligence helps us perform our day-to-day tasks, make decisions, and complete our daily chores. This makes AI a lot popular these days.

1.5.1 AI in Video Games:

The creator can't control other characters in the game. Thus, programmers add artificial intelligence in the game, which controls all the other characters in the game and allows them to run, shoot, follow, and do creative tasks.

1.5.2 AI in the fraud detection system:

Many times we receive fake emails such as login to bank, credit card security, etc. These systems analyze emails, and many banks have security methods which prevent theft of credit card details, etc.

1.5.3 Intelligent personal assistant:

Can perform a lot of tasks with voice commands from the user. Siri in Apple's iPhones is the best example. You have to give a voice command to the Siri, and it can perform tasks like calling a person, setting the alarm, performing a web search, playing music etc.,

1.5.4 Automatic Cars:

With advancements in automobile technology, car manufacturers have also developed cars that can run independently without needing a human being to control it. However, you cannot see these cars on the road yet; developers are working on getting it on the road. These types of cars rely on GPS.

1.5.5 Email Spam Filtering:

There are Millions of Spam Emails send every day. Email filters do great work in keeping your inbox clean. These systems find spam email by sender's address, IP, Geo-Location, words used etc. If they found a match, either they send the email to spam box or delete it completely.

1.5.6 AI in Education:

AI can automate grading, giving educators more time. AI can assess students and adapt to their needs, helping them work at their own pace. AI tutors can provide additional support to students, ensuring they stay on track.

1.5.7 AI in Science:

Used in the development of new drugs and to model clinical studies in animals and humans. They may also use them in Remote controlled surgical procedures.

1.5.8 AI in healthcare:

The biggest bets are on improving patient outcomes and reducing costs. Companies are applying machine learning to make better and faster diagnoses than humans.

1.5.9 AI for Advanced weather modelling:

It can look for past models to predict weather patterns.

1.5.10 AI in advanced financial systems:

i.e., to quickly analyze trends, this allows rapid investment decisions. Etc.,

****** EXTRA MATERIAL [End]******

2. Intelligent Agents:

2.1 Agents and environments:

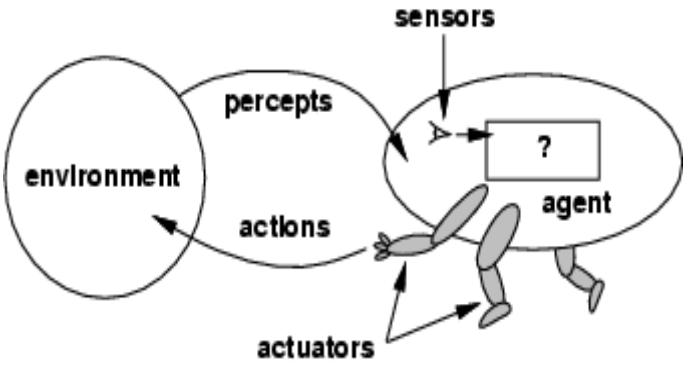

Fig 2.1: Agents and Environments

2.1.1 Agent:

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, mouth, and other body parts for actuators.
- ✓ A *robotic agent* might have cameras and infrared range finders for sensors and various motors for actuators.
- ✓ A *software agent* receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

2.1.2 Percept:

We use the term percept to refer to the agent's perceptual inputs at any given instant.

2.1.3 Percept Sequence:

An agent's percept sequence is the complete history of everything the agent has ever perceived.

2.1.4 Agent function:

Mathematically speaking, we say that an agent's behaviour is described by the agent function that maps any given percept sequence to an action.

2.1.5 Agent program

Internally, the agent function for an artificial agent will be implemented by an agent program. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running on the agent architecture.

To illustrate these ideas, we will use a very simple example-the vacuum-cleaner world shown in **Fig 2.1.5**. This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck, otherwise move to the other square. A partial tabulation of this agent function is shown in **Fig 2.1.6**.

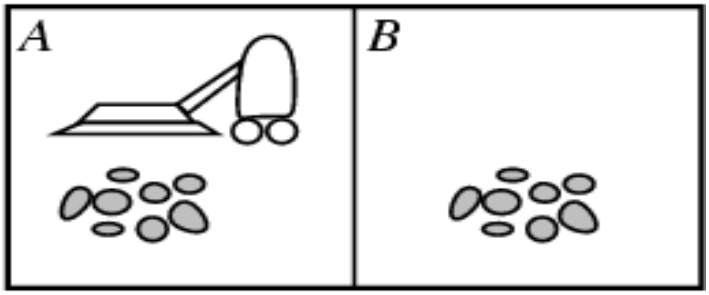

Fig 2.1.5: A vacuum-cleaner world with just two locations.

2.1.6 Agent function

Percept Sequence	**Action**
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
…	

Fig 2.1.6: Partial tabulation of a simple agent function for the example: vacuum-cleaner world shown in the **Fig 2.1.5**

```
Function REFLEX-VACCUM-AGENT ([location, status]) returns an action

If status=Dirty then return Suck

else if location = A then return Right

else if location = B then return Left
```

Fig 2.1.6(i): The REFLEX-VACCUM-AGENT program is invoked for each new percept (location, status) and returns **an** action each time.

3. Good Behavior:

3.1 Concept of Rationality:

A *Rational agent* is one that does the right thing-conceptually speaking; every entry in the table for the agent function is filled out correctly. Obviously, doing the right thing is better than doing the wrong thing. The right action is the one that will cause the agent to be the most successful.

3.1.1 Performance measures:

A performance measure embodies the criterion for the success of an agent's behaviour. When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well.

Consider the vacuum-cleaner agent from the preceding section. We might propose to measure performance by the amount of dirt cleaned up in a single eight-hour shift. With a rational agent, of course, what you ask for is what you get.

- A rational agent can maximize this performance measure by cleaning up the dirt, then dumping it all on the floor, then cleaning it up again, and so on. A more suitable performance measure would reward the agent for having a clean floor.

- For example, one point could be awarded for each clean square at each time. The selection of a performance measure is not always easy. It is based on average cleanliness over time. Yet the same average cleanliness can be achieved by two different agents, one of which does a mediocre job all the time: while the other cleans energetically but takes long breaks.

"As a general rule, it is better to design performance measures according to what one actually wants in the environment, rather than according to how one thinks the agent should behave."

3.1.2 Rationality:

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.

Consider the simple vacuum-cleaner agent that cleans a square if it is dirty and moves to the other square if not; this is the agent function tabulated in **Fig 2.1.6** Is this a rational agent? That depends! First, we need to say what the performance measure is, what is known about the environment, and what sensors and actuators the agent has. Let us assume the following:

- The performance measure awards one point for each clean square at each time step, over a "lifetime" of 1000 time steps.
- The *"geography"* of the environment is known *a priori* **(Fig *2.1.5*)**, *but the dirt distribution and the agent's initial location* are not. Clean squares stay clean and sucking cleans the current square. The *Left* and *Right* actions move the agent left and right except when this would take the agent outside the environment, in which case the agent remains where it is.

- The only available actions are *Left; Right, Suck, and NoOp* (do nothing).
- The agent correctly perceives its location and whether that location contains dirt.

We claim that under these circumstances, the agent is indeed rational; its expected performance is at least as high as any other agent's.

3.1.3 *Omniscience, learning, and autonomy:*

An *omniscient agent* knows the actual outcome of its actions and can act accordingly, but omniscience is impossible in reality. Doing actions in order to modify future percepts-sometimes called information gathering-is an important part of rationality. Our definition requires a rational agent not only to gather information but also to learn as much as possible from what it perceives. To the extent that an agent relies on its designer's prior knowledge rather than on its own percepts, we say that the agent lacks autonomy. A rational agent should be autonomous-it should learn what it can to compensate for partial or incorrect prior knowledge.

3.2 *The Nature of Environments:*

3.2.1 Task environments:

We must think about task environments, which are essentially the "problems" to which rational agents are the "solutions."

3.2.2 Specifying the task environment

The rationality of the simple vacuum-cleaner agent, needs specification of

- ✓ The performance measure

- ✓ The environment
- ✓ The agent's actuators and sensors.

3.2.3 PEAS:

All these are grouped together under the heading of the ***task environment*****.** We call this the ***PEAS*** (***Performance, Environment, Actuators, and Sensors***) description. In designing an agent, the first step must always be to specify the task environment as fully as possible.

Agent Type	Performance Measure	Environments	Actuators	Sensors
Taxi driver	Safe: fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, Signal, horn, display	Cameras, sonar, Speedometer, GPS, Odometer, engine sensors, keyboards, accelerometer

Fig 3.2.3(i): PEAS description of the task environment for an automated taxi.

Agent Type	Performance Measure	Environment	Acurators	Sensors
Medical Diagnostics	Healthy patient ,minimize costs, lawsuits	Patient, hospital, staff	Display questions, tests, diagnosis, treatments, referrals	Keyboard entry of symptoms, findings, patients answers

Satellite Image Analysis	Correct image categorization	Down linking from orbiting satellite	Display categorization of scene.	Color pixel Array
Part Picking Robot	Percentage of parts in correct bins	Conveyor belt with parts , bins	Jointed arm and hand	Camera, joint angel sensors
Refinery Control	Maximize purity, Yield ,safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English Tutor	Maximize student's score on tests	Set of students, testing agency	Display, Suggestion, Corrections, exercises	Keyboard Entry

Fig 3.2.3 (ii): Examples of agent types and their PEAS descriptions.

3.2.4 Properties of task environments:

- Fully observable ***vs*** partially observable
- Deterministic ***vs*** stochastic
- Episodic ***vs*** sequential
- Static ***vs*** dynamic
- Discrete ***vs*** continuous
- Single-agent ***vs*** Multiagent

3.2.4.1 Fully observable vs partially observable

- If an agent's sensors give its access to the complete state of the environment at each point in time, 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;
- 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, and an automated taxi cannot see what other drivers are thinking.

3.2.4.2 Deterministic vs stochastic

- If the next state of the environment is completely determined by the current state and the agent's action, then we say the environment is deterministic; otherwise, it is stochastic.
- For example, Taxi driving is clearly stochastic because one can never predict traffic behaviour exactly; moreover, one's tires blow out, and one's engine seizes up without warning. The vacuum world, as we described it, is deterministic.

3.2.4.3 Episodic vs sequential

- In an episodic task environment, the agent's experience is divided into atomic episodes.
- Each episode consists of the agent perceiving and then performing a single action. Crucially, the next episode does not depend on the actions taken in previous episodes.
- 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; where as in sequential environments, the current decision could affect all future decisions. Chess and taxi driving are sequential.

3.2.4.4 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 to worry about the passage of time.
- Dynamic environments 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 *semi-dynamic*. 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 semi-dynamic. Crossword puzzles are static.

3.2.4.5 Discrete vs continuous

- The discrete/continuous distinction can be applied to the *state* of the environment, the way time is handled, and the agent's percepts and actions.
- For example, a discrete-state environment such as a chess game has a finite number of distinct states. Chess also has a discrete set of percepts and actions.
- Taxi driving is a continuous state and continuous-time problem: the taxi's speed and location and 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.).

3.2.4.6 Single-agent vs Multiagent.

- 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.
- As one might expect, the most demanding case is *partially observable*, *stochastic*, *sequential*, *dynamic*, *continuous*, and *multiagent*.

- Chess is a *competitive* multiagent environment. On the other hand, in the taxi-driving environment, avoiding collisions maximizes all agents' performance measure, so it is a partially *cooperative* multiagent environment. It is also partially competitive because, for example, only one car can occupy a parking space.

Task Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	Fully	Deterministic	Sequential	Static	Discrete	Single
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Poker	Partially	Stochastic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Image-analysis	Fully	Deterministic	Episodic	Semi	Continuous	Single
Part-picking robot	Partially	Stochastic	Episodic	Dynamic	Continuous	Single
Refinery controller	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Interactive English tutor	Partially	Stochastic	Sequential	Dynamic	Discrete	Multi

Fig 3.2: Figure of the properties of the different environments

3.3 Structure of Agents:

AI's job is to design the *agent program* that implements the agent function mapping percepts to actions. We assume this program will run on some sort of computing device with physical sensors and actuators-we call this the architecture:

Agent = architecture + program

3.3.1 Agent programs

The agent programs that we will design in this book all have the same skeleton: they take the current percept as input from the sensors and return an action to the actuators. Notice the difference between the agent program, which takes the current percept as input, and the agent function, which takes the entire percept history. The agent program takes just the current percept as input because nothing more is available from the environment; if the agent's actions depend on the entire percept sequence, the agent will have to remember the percepts. We will describe the agent programs via the simple pseudo-code language.

```
Function TABLE-DRIVEN_AGENT(percept) returns an action

static: percepts, a sequence initially empty

static: table, a table of actions, indexed by percept sequence, initially fully specified

append percept to the end of percepts

percepts <- APPEND(percept, percepts)

action ← LOOKUP(percepts, table)

return action
```

Fig 3.3.1: The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns **an** action each time.

3.3.2 Drawbacks:

- Table lookup of percept-action pairs defining all possible condition-action rules necessary to interact in an environment
- Problems
  - ✓ Too big to generate and to store (Chess has about 10^120 states, for example)
  - ✓ No knowledge of non-perceptual parts of the current state
  - ✓ Not adaptive to changes in the environment; requires an entire table to be updated if changes occur
  - ✓ Looping: Can't make actions conditional
- Take a long time to build the table
- No autonomy
- Even with learning, need a long time to learn the table entries

3.3.3 Some Agent Types:

- ***Table-driven agents***
- ***Simple reflex agents***
- ***Model-based reflex Agents***
- ***Goal-based Agents***
- ***Utility-based Agents***
- ***Learning Agents***

3.3.3.1 Table Driven Agent

These agents use a percept sequence/action table in memory to find the next action. A (large) lookup table implements them. Refer this from above table represented in **Fig 3.3.1**

3.3.3.2 Simple Reflex Agent

The simplest kind of agent is the simple reflex agent. These agents select actions based on the current percept, ignoring the rest of the percept history. For example, the vacuum agent whose agent function

is tabulated in Figure 1.10 is a simple reflex agent. Its decision is based only on the current location and on whether that contains dirt.

- Select action on the basis of only the current percept.
  E.g. the vacuum-agent
- A considerable reduction in possible percept/action situations.
- Implemented through condition-action rules
- If dirty then suck

Imagine yourself as the driver of the automated taxi. If the car in front brakes and its brake lights come on, you should notice this and initiate braking. In other words, some processing is done on the visual input to establish the condition we call *"The car in front is braking"*. Then, this triggers some established connection in the agent program to the action "initiate braking".

We call such a connection a condition-action rule: written as

If *car-in-front-is-braking* ***then*** *initiate-braking.*

A Simple Reflex Agent: Schema

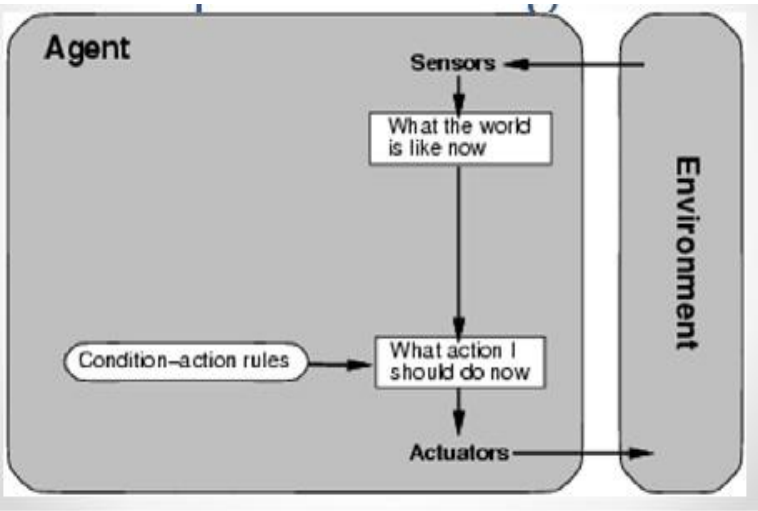

Fig 3.3.3.2: A General model of Simple Reflex Agent

```
function SIMPLE-REFLEX-AGENT(percept) returns an action

static: rules, a set of condition-action rules

static: state, a description of the current world state

static: action, the most recent action.

state ← INTERPRET-INPUT(percept)

rule ← RULE-MATCH(state, rule)

action ← RULE-ACTION[rule]

return action
```

Fig 3.3.3.2 (i): *A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.*

```
function REFLEX-VACUUM-AGENT ([location, status]) return an action

if status == Dirty then return Suck

else if location == A then return Right

else if location == B then return Left
```

Fig3.3.3.2 (i): The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated in the **Fig 2.1.6**

Characteristics:

- Only works if the environment is fully observable.
- Lacking history, easily get stuck in infinite loops.
- One solution is to randomize actions

Gaming example of a simple reflex-based agent

Sometimes we do not need an extensive analysis of the game state to figure out our next move. At times, the game is simple enough that we can make a general strategy and represent it as a set of rules. A rule would specify that if a certain condition is met, a certain action should be taken. An agent built using such rules is called a **Simple Reflex Agent.**

A Simple Reflex Agent is typically employed when all the current game state's information is directly observable, (e.g., Chess, Checkers, Tic Tac Toe, Connect-Four) decision regarding the move only depends on the *current* state. That is when the agent does not need to remember any past state's information to make a decision.

For example, let's consider an agent for Connect-Four. This agent has two objectives: - Stop the enemy from completing four-in-a-row.

For the first aim, some simple rules can be made as follows:

- If any column is topped by two or more of your opponent's pieces, place your piece in that column.
- If you see two or more of opponent's pieces placed adjacently in a row, with an empty *place-able* space on either side, place a piece there.
- If you see two or more of opponent's pieces placed adjacently in a diagonal, with an empty *place-able* space on either side, place a piece there.

Here, *place-able* means that you can place a piece in that location by the rules of the game, that the place directly below it (if it exists) should be filled by a piece.

Each of these "two or more" rules can be broken down into two rules, one for "two" and one for "three", where the rules for "three" will take precedence.

Similarly, we can choose strategies for the second aim: "Completing a Four-in-a-Row" and encoding it in the form of such condition-action rules.

During these lessons, most of our discussions assume we want to build an AI agent that aims to be as smart as the player. When making AI enemies in video games, this is usually not the requirement. So even though not every information is directly observable in a game, a Simple Reflex agent might be a good choice because the AI enemy doesn't *need* to know everything. An enemy can simply function on what it can immediately see. This applies well to retro top-down games. If you're in the enemy's line of sight, it will attack you. Otherwise, it will patrol its pre-defined path or area. This can be achieved by making very simple AI enemies by defining a few simple rules of behaviour.

3.3.3.3 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*. The agent should maintain some sort of *internal state* that 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.

- First, we need some information about how the world evolves independently of the agent. For example, an overtaking car will generally be closer behind than it was a moment ago.

- Second, we need some information about how the agent's own actions affect the world-for example, that when the agent turns the steering wheel clockwise, the car turns to the right or that after driving for five minutes northbound on the freeway one is usually about five miles north of where one was five minutes ago.

- This knowledge about how the world works – whether implemented in simple Boolean circuits or incomplete scientific theories- is called a world model. An agent that uses such a MODEL-BASED model is called a *model-based agent.*

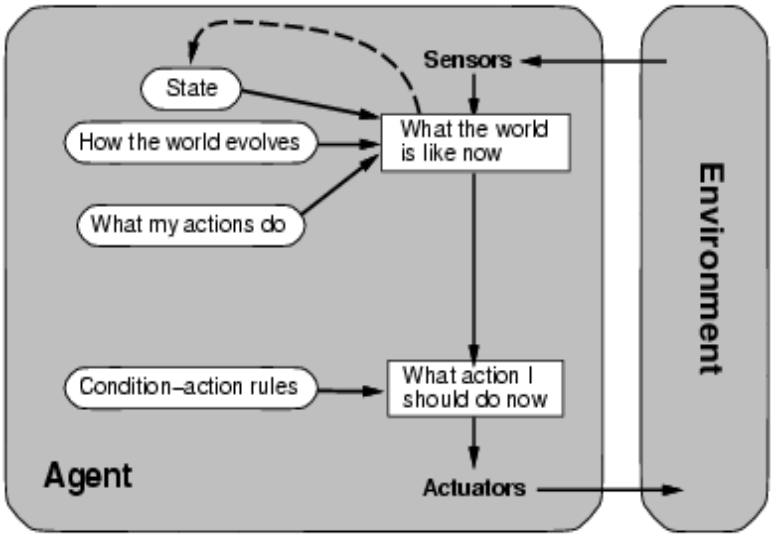

Fig 3.3.3.3: A General model of Model-Based Agent

```
function REFLEX-AGENT-WITH-STATE(percept) returns an action

static: rules, a set of condition-action rules

static: state, a description of the current world state

static: action, the most recent action.

state ← UPDATE-STATE(state, action, percept)

rule ← RULE-MATCH(state, rule)

action ← RULE-ACTION[rule]

return action
```

Fig 3.3.3.3 (i): *Model based reflex agent. It keeps track of the current state of the world using an internal model. It then chooses an action in the same way as the reflex agent.*

Gaming Example of Model based reflex agent

Simple Reflex Agents' problem is that they can only operate on the *current* game state and cannot use information from older states, not even the directly previous one. Model-based Reflex Agents achieve this capability. These agents maintain an internal model of the game world, which they update at every turn using the observations from the current state. So this model is an aggregation of all states the agent observed.

The example from the previous section considers the AI enemy that patrols until it sees the player. If we use a Simple reflex agent, it would only consider what it sees in that very instant. So the moment the player turns a corner, the enemy will stop chasing. So while the Simple Reflex Agent might be good for an automated turret, it's a bad model for enemy guards.

When we build a Model-Based Reflex Agent for an enemy guard, the guard can have an **internal model** of the world and himself. This allows him to use observations from the past. We can use this internal model to store the following information: - The state of the guard (e.g., Patrolling, Chasing, etc.), location where the target was last scene - A cooldown timer (most games have a fixed time before the guard returns to his post) - The path he followed when chasing the enemy. (This is if you want the guard to return to his patrol)

Another use for Model-Based Reflex Agents is when initially your agent does not know the map, but you want it to explore and remember the places. So as it moves around and sees more of the level, it creates an internal map.

3.3.3.4 Goal-based agents

Knowing about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of **goal** information that describes situations that are a desirable-for example, being at the passenger's destination. The agent program can combine this with information about the results of possible actions (the same information as was used to update internal state in the reflex agent) in order to choose actions that achieve the goal.

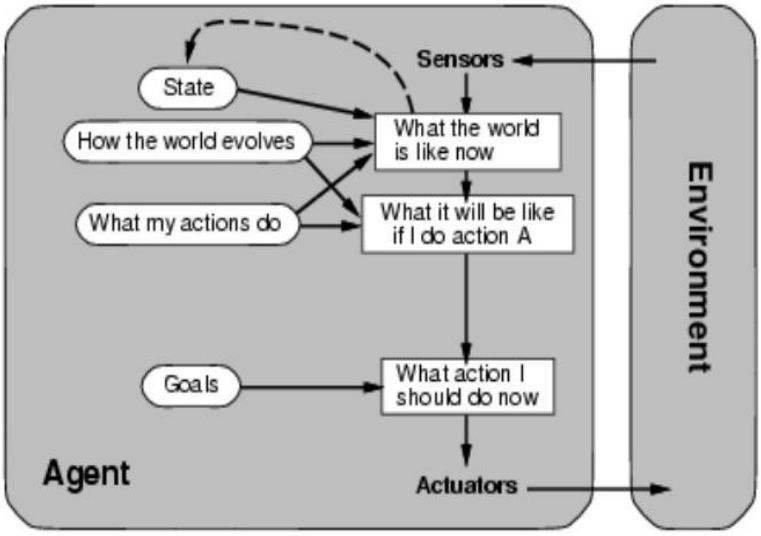

Fig 3.3.3.4: A General model of goal based agent

With reflex agents, the goal is **implicit** in the condition-action rules. With goal-based agents, we make the goal **explicit**. This allows us to

- change the goal easily, and
- Use search and planning in deciding the best action at any given time.

Consider the following architecture for an intelligent agent:

```
function GOAL-BASED-AGENT(percept, goal) returns an action

static: s, an action sequence, initially empty
static: state, a description of the current world state

state <- UPDATE-STATE(state, percept)
if s is empty then
s <- SEARCH(state, goal)

action <- RECOMMENDATION(s, state)
```

```
s <- REMAINDER(s)
return action
```

Fig3.3.3.4 (i): *A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve and chooses an action that will (eventually) lead to the achievement of its goals.*

- The heart of the goal-based agent is the search function.
- It returns a path from the current state to the goal--a plan.
- The agent then returns the first action in the plan.
- A simple enhancement would allow the agent to have multiple goals.

Gaming Example of Goal-Based Agents

Previously we discussed Model-Based Reflex Agents as a way to design simple enemies. We considered a very simple behaviour of the AI enemy, which can be stated in the form of following condition-action rules:

- If **patrolling** and **no enemy in sight** then Patrol pre-defined path
- If **patrolling** and **enemy in sight**, switch-mode from **patrolling** to **chasing**
- If **chasing** and **enemy in sight**, move towards the enemy
- If **chasing** and the **enemy not in sight** move towards the **enemy's last location**
- If at **enemy's last location** and the **enemy not in sight** and **cooldown timer** not started then **start cooldown timer**
- If at **enemy's last location** and the **enemy not in sight** and **cooldown timer expired** then change the mode to **returning to post**
- if **returning to post** and **enemy in sight** then set the mode to **chasing**
- if **returning to post** and reached the **patrolling path** and set the mode to **patrolling**

Now, as you may notice, we are still using **condition-action rules** as we did in the **Simple Reflex Agent**. The differentiating factor here is that we can maintain an **internal state** and model of the

game environment and use it to take decisions. However, note that the decisions are still being taken using **pre-defined** condition action rules.

So what if we want to make our AI guards somewhat smarter? What if we want them to figure out what to do, themselves by understanding their situation and possible paths of action? This might be needed if our guard needs to choose between multiple possible responses like:

- Continue Patrolling
- Investigate Sounds
- Chase Intruder
- Stay in sight of other guards
- Go sound the alarm

Now instead of having pre-defined Condition-Action rules for choosing the actions, our guard should choose the action that best aligns with his **goal**. If we set the goal to be **Capture Intruder**, then the guard will always give precedence to **Chase Intruder** and **Investigate Sounds** as these actions bring the guard to his goal of capturing the intruder. Similarly, if we set the Goal to **Stay Safe**, the guard would always choose to **Continue Patrolling** and **Stay in Sight of Other Guards**.

Ideally, we want an Agent that considers all possibilities and choose the **most useful** one. This means that the agent should choose the one with the highest Utility in case of multiple competing goals. Agents that use *Utility* are also known as **Utility-Based Agents**.

The tree-traversing agents we discussed in previous lessons were actually forms of Utility-Based agents. They explored what actions would lead to what game states, how much that state would be worth (score/utility), and then choose the action that leads the state with the highest Utility.

As you might notice, Goal-Based agents need to understand the game and how the game state changes. Like the Model-Based Agents, Goal-Based agents also have an internal model of the game state. Whereas Model-Based Agents only need to know how to update their internal model of the game state using new observations, Goal-based agents have the additional requirement of knowing how their actions will affect the game state. This is because; Goal-Based Agents use their internal model to test out strategies. When they know how their actions will affect the game state, they can

find out the different states each move will lead to. Then based on the Utility of these, they choose the best strategy.

While Goals can be defined in any way that suits your need and implementation of the game, they are usually defined in terms of an observable (or calculable) variable. For Example:

- **Achieve** *distance to target* = 0
- **Maintain** *Health* > 0

The **Achieve** type of goal aims to reach a certain condition that isn't true initially; the **Maintain** type of goal aims to keep a condition true (which is usually true initially). Now to be able to do this, the agent needs to understand how his actions affect these variables. The agent will then judge if he is getting closer or further from fulfilling a goal.

Your implementation of the agent determines how the agent will behave. You can implement it to be very cautious and avoid even getting *close* to failing a **Maintain** goal. Or you can make it such that it sees Maintain goals as valid ranges in which to function, and thus free moves within the range, even if means coming very close to the border. You can also implement multiple modes, one for each such behavior. This may be needed as you would want the agent to be very conservative with *Health*, but you may need him to be flexible with metrics such as *distance from target*, as long as it leads to achieving other goals.

It also helps to prioritize goals, so the agents know when it has to choose between two goals, whose Utility should have a higher weightage. This usually requires you to normalize your utility values, so that you can compare "Come close to target by 100 units" and "Lose 10 Health" on an equal footing.

3.3.3.5 Utility-based agents

Goals alone are not really enough to generate high-quality behaviour 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, whereas a more general *performance measure*

should allow a comparison of different world states according to exactly how happy they would make the agent if they could be achieved. Because "*happy*" does not sound very scientific, the customary terminology says that if one world state is preferred to another, it has higher *Utility* for the agent.

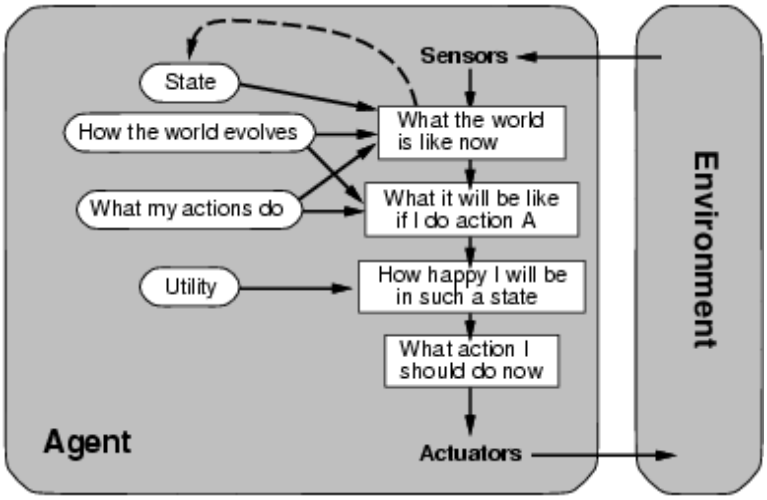

Fig 3.3.3.5: A General model of Utility-based agent

Sometimes agents will have multiple conflicting goals. In this case, a utility function is more appropriate.

A utility function assigns a number proportional to the "happiness" of the agent to each state.

An ideal agent will attempt to maximize the expected value of the utility function, so it must know the probability of each possible outcome of an action in a given state. Consider the following architecture for an intelligent agent:

```
function UTILITY-BASED-AGENT(percept, utility-fn) returns action

static: s, an action sequence, initially empty
static: state, a description of the current world state
```

```
state <- UPDATE-STATE(state, percept)
if s is empty then

s <- SEARCH(state, utility-fn)


action <- RECOMMENDATION(s, state)
s <- REMAINDER(s)
return action
```

Fig3.3.3.5 (i): *A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best-expected Utility, where expected Utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.*

The SEARCH function differs from that of a goal-based agent in that it attempts to find the optimal, reachable state in terms of the expected value of the utility function.

Certain goals can be reached in different ways.

- Some are better, have a higher utility.
  - ✓ Utility function maps a (sequence of) state(s) onto a real number.

- Improves on goals:
  - ✓ Selecting between conflicting goals
  - ✓ Select appropriately between several goals based on likelihood of success.

3.3.3.6 Learning agents:

A learning agent has a performance element that decides which actions to take, and a learning element that can change the performance element to be more efficient as the agents learn. A critic

component is used to evaluate how well the agent is doing and provide feedback to the learning component, and a problem generator that can deviate from the usual routine and explore new possibilities. (The critic and the problem generator can actually be considered as part of the learning component since they both help to learn.)

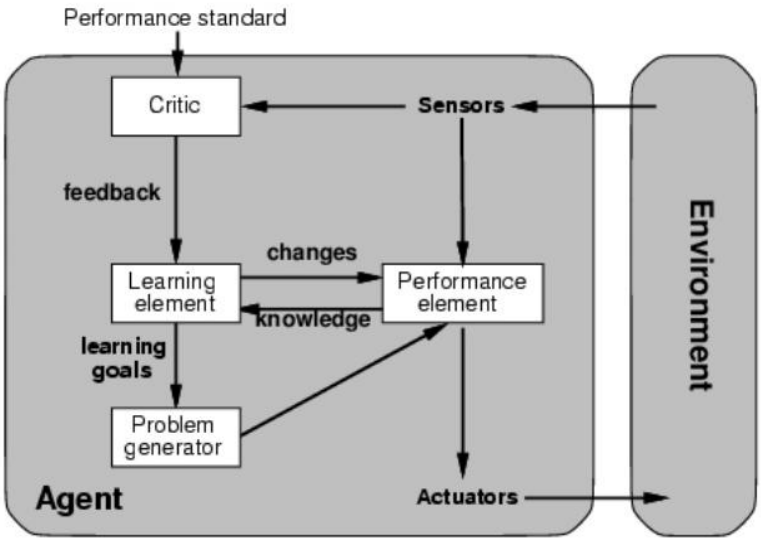

Fig3.3.3.6: *A general model of learning agents.*