

## Personnel

## Lecture 1

### Artificial Intelligence (0270) 2002

In our first lecture, we will have a general overview of the course, and a few essential ideas about Artificial Intelligence in general.

#### References

- Chapter 1: Section 1.1, 1.4
- Chapter 2: The whole chapter

**Lecturer:** Dr. Isaac To Kar Keung ([kkto@csis.hku.hk](mailto:kkto@csis.hku.hk)), CYC407.

Office hour:

#### Tutors:

- Deng Huimin ([hmdeng@csis.hku.hk](mailto:hmdeng@csis.hku.hk)), Office CYC 321.  
Consultation hour:
- Frank, Wong Ho Ting ([htwong@csis.hku.hk](mailto:htwong@csis.hku.hk)), Office CYC 430.  
Consultation hour:

**Web site:** <http://www.csis.hku.hk/~c0270/>

#### Textbook:

- “*Artificial Intelligence, A Modern Approach*”, 2nd Edition. Russell and Norvig, Prentice Hall. Web site: <http://aima.cs.berkeley.edu/>.

AI(0270)

AI(0270)-1.1

## Calling for help

If you have difficulties, you can do one of the followings:

In the order of preference...

- Post a news article to our **newsgroup**, [hku.csis.CSIS0270](mailto:hku.csis.CSIS0270). This is preferred: all of you can answer, and all of you benefit from the answer. Questions asked in any other means may be answered (only) in newsgroup, so **please keep watching the newsgroup**.
- Send us a **mail** at [c0270@csis.hku.hk](mailto:c0270@csis.hku.hk). The mail will end up in the mailbox of me and all tutors.
- Contact us directly. Send a personal mail before visiting our office to make an **appointment**.  
This is the order in which I deal with them. I.e., news are first cleared, and only when all news are dealt with I'll deal with mails to c0270, and only after that I'll look at personal mails.

AI(0270)-1.2

## Lecture conduct

Lectures are free-form, relaxed but serious.

- **Talking** is allowed **only on course matters**.
- **Attend lectures** only if you are willing to learn.  
Nobody force you into lectures anyway.
- **Everything** about the lecture **needs to be understood**.  
Preferably during the lecture, and absolutely no later than the end of the day.
- **Ask** whenever you find something you don't understand.  
Don't hesitate: what you don't understand is probably not understood by many others as well, and you're helping others by just asking.
- **Stop the lecture** if you need time to digest. This is **vitaly** important: it is usually the only way for me to know that you need more time.  
This is the way in which you can understand everything during the lecture.

AI(0270)-1.3

## Assessment

### Assignments (35%)

- 4 programming assignments, with written and programming parts.  
**Language:** you can choose either C++ or Java.  
**Platform:** one of Linux, our departmental servers, or Sun Java development platform.

### Quiz (15%), To be announced

- 1 hour quiz of easy questions. Should be very easy if you have done the assignments.  
Big hint! Quiz is on assignments, not on other materials of the course.

### Exam (50%)

## Advices on learning

- **Learn the concepts first (most important)**. Implementation details are not as important as the concepts.  
We usually show psuedo-code for illustration purpose. But if you are not familiar with computer code, it is okay to skim them (until you need to write programs).
- **Understand, not just memorize**. Find the underlying ideas that give rise to concepts. Find unifying themes.
- **Keep up with lectures**. Each lecture builds on top of the previous ones, so if you miss just one lecture, it will be significantly more difficult to catch up with upcoming lectures.
- **Do the assignments early**. Programming assignments are usually more difficult than what you first think. And assignments clear up your concepts for easy understanding of the coming lectures.
- **Read the book**. Lecture is too short to explain everything in the book, and the function of lectures is simply to guide you to read.

AI(0270)-1.4

AI(0270)-1.5

## What is Artificial Intelligence?

AI is the study of **making intelligent systems**.

Note the word "making": we will write quite some programs.

But what is intelligent? Different approaches:

<b>Think like human (cognitive modelling)</b> Automate human thinking, e.g., decision making, problem solving, learning.	<b>Think rationally (laws of thought)</b> Perform computations that makes it possible to perceive, reason and act.
<b>Act like human (Turing test)</b> Do things that requires intelligence when performed by people.	<b>Act rationally (rational agent)</b> Explain and emulate intelligent behaviour in terms of computational process.

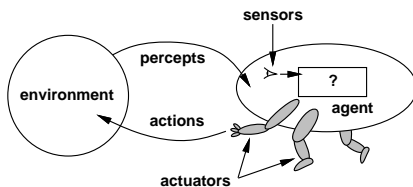
Note that we didn't specify the problem domain. This vagueness is due to the fact that we want **general strategies**, not specific to particular problems.

AI(0270)-1.6

AI(0270)-1.7

## Agent

- We use the term **agent** to mean anything that:
  - perceives** the environment through **sensors**
  - and performs **actions** to the environment through **actuators**.
- Our primary aim: **make good agents**.



AI(0270)-1.8

AI(0270)-1.9

## Rational agents

- We aim at **rational** agents, i.e., one which *does the right thing*, i.e., the action that would make us **successful**. But successful in what?
- We need some **performance measures**: how to judge the system. Must be fix edin advance, and need to be accurate.
- When** to evaluate performance? We will need some averaging.
- So our aim: **optimized** the long term and short term **average** of the **performance measures**.
- However, it is usually not possible to **always** be the best, so we have to make compromises and accept something that is reasonably good. E.g., it might be computationally impossible, or it might takes too much time or memory of the computer.

## Rational is not omniscience

I see my old friend across the street, and want to chat with him. Noting that there is no traffic ,and I've nothing else to do, I started crossing the street... only to get flattened in the middle of the road, by a cargo door falling off from a 747. Am I rational?

- Rationality is not perfection**. The choice of an agent can depend only on percept sequence **to date**.  
Otherwise we want a crystal ball, not a computer!
- But what if, in our environment, cargo door falls every 2 minutes?
- Then we better look up first! It is **too dangerous** to cross the street **without first acquiring the knowledge** about whether doors are falling...

AI(0270)-1.10

## What is rational

Rationality: **expected** success *given what has been perceived*. It is determined by

- The **performance measure**.
- What is known by the agent about the **environment**.  
E.g., is it known that falling cargo door is common?
- What the **actuators** can do to the environment.  
E.g., can it explore for a different action?
- What the **sensors** perceived so far.  
The book calls it PEAS, although we will try to avoid meaningless acronyms.

An **ideal rational agent** will maximize the expected performance measure for **every possible percept sequence**, on the basis of the percepts and its built-in knowledge about the environment.

AI(0270)-1.11

### Example

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, minimize costs, lawsuits	Patient, hospital, staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display categorization of scene	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Maximize purity yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

AI(0270)-1.12

### Let's look it up!

Input is some percept, output is some action. Thus our agent is simple:

```

percepts = []           # An empty percept sequence
table = [ ... ]        # Maps percept sequences to actions
def TableDrivenAgent(p):
    percepts.append(p)
    return table[percepts]

```

Note: We use a syntax that clearly demonstrate the concepts. When you implement it, you will need to deal with the restriction imposed by the actual language. BTW, the syntax looks a bit like a scripting language called Python.

- Note that we don't need to put the environment and the evaluation functions here. These information are **encoded into the table**.

AI(0270)-1.13

### The failure of lookup

The problem is that it **won't work for most situations**.

- The **table** will be **huge**, spending a lot of memory.  
E.g., to play "reverse it", the table will need 60! entries.
- It will take a **long time** for the designer to **fill the table entries**.  
Both because the table is large, and because it requires some intelligence.
- If the environment or the evaluation function **changes** a little bit, the **table needs to be completely redesigned**.  
In other words, the degree of automation is very low.
- It is very **difficult** for the program to **learn** by itself, because it needs to learn all entries of the huge table.

If you say "The program is not intelligent", you better tell why it's **not satisfactory in certain areas**, otherwise you won't know how to improve it.

AI(0270)-1.14

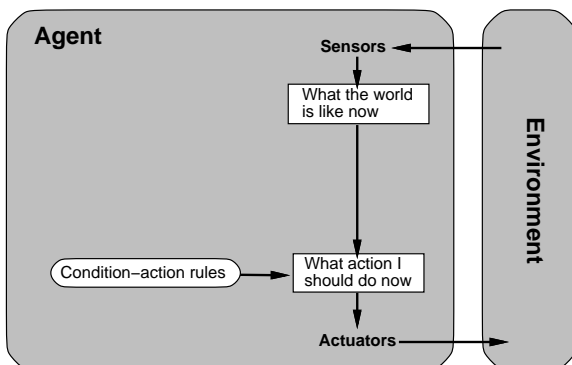
### Reflex agent: making lookup more reasonable

If lookup is infeasible, what can we do to improve the situation?

- One possibility: seek **if-then rules** that govern, in the following form:  
**if car-in-front-is-braking then initiate-braking**
- Such agents are called **reflex agent**: like reflex in neural-science, it is encoded into the program, and effected without reasoning.
- Example: to play a game of tic-tac-toe using table, we would need  $3^9 = 19683$  entries.
- E.g., to play a game of tic-tac-toe, around 80 rules suffice .  
If you write more complicated rules, 5 might suffice .
- But even in such simple situation, it is usually not a good idea to use reflex agents.  
E.g., in TTT, point 2–4 are still quite true, and we can do better.

AI(0270)-1.15

### A schematics of a reflex agent



AI(0270)-1.16

### How to do better?

So what can we do to improve the situation?

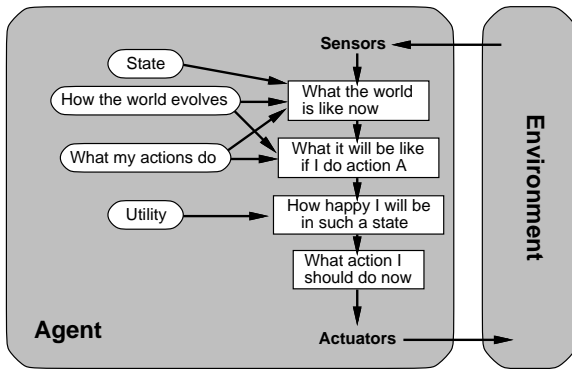
- Write code that **predicts the results** of a possible action.
- Explicitly code the **evaluation function** into our program.
- Write code to **plan action sequences** to maximize performance.
- Find the **intrinsic properties** that can easily change, and write code that **learn** the values of these properties automatically.  
Unluckily, the course does not have enough time to cover this.

E.g.: instead of writing the 80 rules governing the best playing strategy of TTT, we write code to play the game, given the specification of the game.  
We will know how to do that in 4 weeks.

As a result, the program can play any similar game that requires the same level of expertise.

AI(0270)-1.17

### A schematics of a utility-based agent



AI(0270)-1.18

### A better agent

It is instructional to see what is needed by such a Tic-tac-toe program.

Problem specific information hard-coded into the program

Generic functions that can be reused by similar problems

- How to get from percepts
- What are the possible action
- What is the result of actions
- What is considered winning
- How to put actions into effects
- Look for possible actions
- Predict what opponent will do
- Find the expected result
- Find the expected performance
- Find the best strategy

How difficult is it? Indeed, not that difficult. Such a program can be found in the source of the lecture notes, it's around 100 lines.

So easy that it is not usually regarded as AI.

AI(0270)-1.19

### About the environment

There are a few things special about the environment that makes TTT particularly easy:

- It is **fully observable**, i.e., at any time the whole environment can be perceived. This is as opposed to "partially observable".
- It is **deterministic**, i.e., the result of every action is fixed. This is as opposed to "stochastic", i.e., probabilistic.
- It is **discrete**, and has only a **small number** of possible percept and action values, as opposed to "continuous".
- It is **static**, i.e., the environment will not be changing when the agent is thinking. This is as opposed to "dynamic".

Not every AI problem is this simple. Confronting more difficult AI problems, the programmer and the computer need to pay more effort.

And we probably need to accept a solution that is not as good as we want.

AI(0270)-1.20

### What's ahead

- Our primary business in this course is to learn **how reasoning takes place** in the computer.
- The tic-tac-toe example shows one of them: **by searching** for ways to succeed. We will learn how to search efficiently.
- Searching will be combined with a method to **represent knowledge** of the environment: logic.
- We will see how new knowledge can be generated from old ones, and how to combine it with searching to plan actions.
- Finally, we will learn how reasoning has to be done when there is uncertainty in our knowledge, which is common in **expert systems**.

What you should do now: get the book, and skim through the first two chapters.

AI(0270)-1.21

### Course overview

- Wk 1: Introduction, searching
- Wk 2: Informed search (A1)
- Wk 3: Constraint satisfaction problems
- Wk 4: Games (A2)
- Wk 5: Logic
- Wk 6: First Order Logic
- Wk 7: Inference (A3)
- Wk 8: Planning, Quiz
- Wk 9: Hierarchical planning
- Wk 10: Planning in Real World problems
- Wk 11: Probabilistic Reasoning
- Wk 12: Belief Network (A4)
- Wk 13: Markov Model
- Wk 14: Hidden Markov Model

AI(0270)-1.22