LECTURE 3

SUMMARY

1. Methodologies for agent-based systems development

- used in the classical SE
  - i. **Waterfall –** sequential design process
    - 1. **Model Driven Development (MDD)** – can be planned either as a Waterfall project or incrementally
  - ii. **Agile** – incremental software development
    - 1. **Test Driven Development (TDD)**
      - a. **Behavior Driven Development (BDD) –** extension of TDD
    - 2. **Agile MDD**
- developed by FIPA – agent-oriented
  - i. Gaia: based on the organisational abstraction
  - ii. MaSE - Multiagent SE

2. Agent-based development lifecycle

E.g. assuming a sequential development

1. Specification

- MAS specification
  - high-level
    - inputs and outputs to/from the MAS system
    - types of agents
    - specification of agents (inputs, outputs, task)
    - specifying communications among agents
      - WHAT the agents should communicate to each other, not HOW
  - Recommended Practices for the Specification of Multi-Agent Systems
- Languages: CASL (Cognitive Agent Specification Language), AgentSpeak (Prolog like)
- Formal methods/languages
  - SLABS – a formal specification language for agent-based systems

        - Formal specification using the BDI (Belief-Desire-Intention) architecture

2. Analysis

- *Conceptual modeling using agents*
    - **PAGE(S)** – Perception, Action, Goal, Environment (State of the environment)
    - **PEAS** – Performance measure, Environment, Actuators, Sensors
- *Properties of the environment*

3. Design

AUML (Agent Unified Modelling Language) developed by FIPA

- sequence diagrams
- activity diagrams
- …

- agent design (architectures for agents)
- agent network design
    - **coordination** (http://www.cs.unibo.it/~cianca/wwwpages/seminari/3languages.pdf)
    - **communication**
        - ACL (Agent Communication Language) – developed by FIPA
- platform/software design – software application or MAS environment

4. Implementation

- languages from the **AOP** (Agent Oriented Programming) paradigm (1993 – Yoav Shoham)
    - https://www.infor.uva.es/~cllamas/MAS/AOP-Shoham.pdf
    - Agent0
        - Lisp like syntax
        - An agent is specified in terms of
            - Set of capabilities (things the gent can do)
            - Set of initial beliefs (playing the role of beliefs in a BDI architecture)
            - Set of initial commitments (playing the role of intentions in a BDI architecture)
            - A set of commitment rules
    - Concurrent MetateM
        - based on temporal logic
        - direct execution of logical formulae (https://www.youtube.com/watch?v=ow2vbA9ohC0)
    - AOP is viewed as
        - a specialization of OOP (more particularly the actors variant)
        - concurrent OOP with asynchronous communication
- imperative languages (e.g. Java)
- multiparadigm languages (Python, Scala, etc)
- **MAS frameworks**
    - **agent** abstraction
    - allow **concurrent** execution of multiple agents
        - agents in a MAS operate concurrently

        - **communication** between agents
        - **Examples**
            - JADE (Java Agent DEvelopment framework)
                - one of the most used agent development framework
            - JADEX (extension of JADE for BDI architecture)
            - Jason (Java based platform – plugin for Eclipse)
                - JaCaMo (MAS platform based on Jason),
            - Boris.NET
                - MAS programing platform for .NET
            - SPADE (Python)
            - MaSMT (Java-based)
                - http://www.cs.ubbcluj.ro/~gabis/docs/mas/masmt2-0-complete-6s.pdf
                - http://www.cs.ubbcluj.ro/~gabis/docs/mas/masmt-development-guide2-01.pdf
            - MadKit (Java-based)

5. **Testing**
    - Unit testing
    - Automated testing
    - …

3. Agent design

- when designing agent-based systems, we should consider **PAGE(S)** – Perception, Action, Goal, Environment.
    - conceptually, the environment (**E**) is the set of the possible environment states (**S**). …thus, we will add an additional **S**

Example: **Simple Vacuum Cleaning Agent**

- we have a small robotic agent that will clean up a room
- the room is entirely surrounded by walls
- the robot is equipped with a sensor and a vacuum cleaner that can be used to suck up dirt.
- the robot has a definite orientation (one of north , south, east, or west)
- the robot is capable of sucking up dirt, moving forward and turning right
- the robot is able to sense if it is over any dirt and it can detect if a wall is directly in front of it
- the robot does not know the location of the dirt ahead of time.
- E.g. R starts from (1,1) facing South

```
   1   2   3   4
 +---+---+---+---+
1|R  |   |   |   |
 +---+---+---+---+
2|   |*  |   |   |
 +---+---+---+---+
3|   |   |   |   |
 +---+---+---+---+
4|   |   |   |*  |
 +---+---+---+---+
Location: (1,1)  Facing: SOUTH
```

Conceptual modeling (PAGE)

- **Agent**
    - robot
- **State**
    - a configuration of the map
- **Environment**
    - grid world
    - set of possible states (configurations of the map)
- **Action**
    - Suck up dirt, forward, turn
- **Perception**
    - dirt
    - wall
- **Goal**
    - to search and remove dirt within the room

Possible extensions

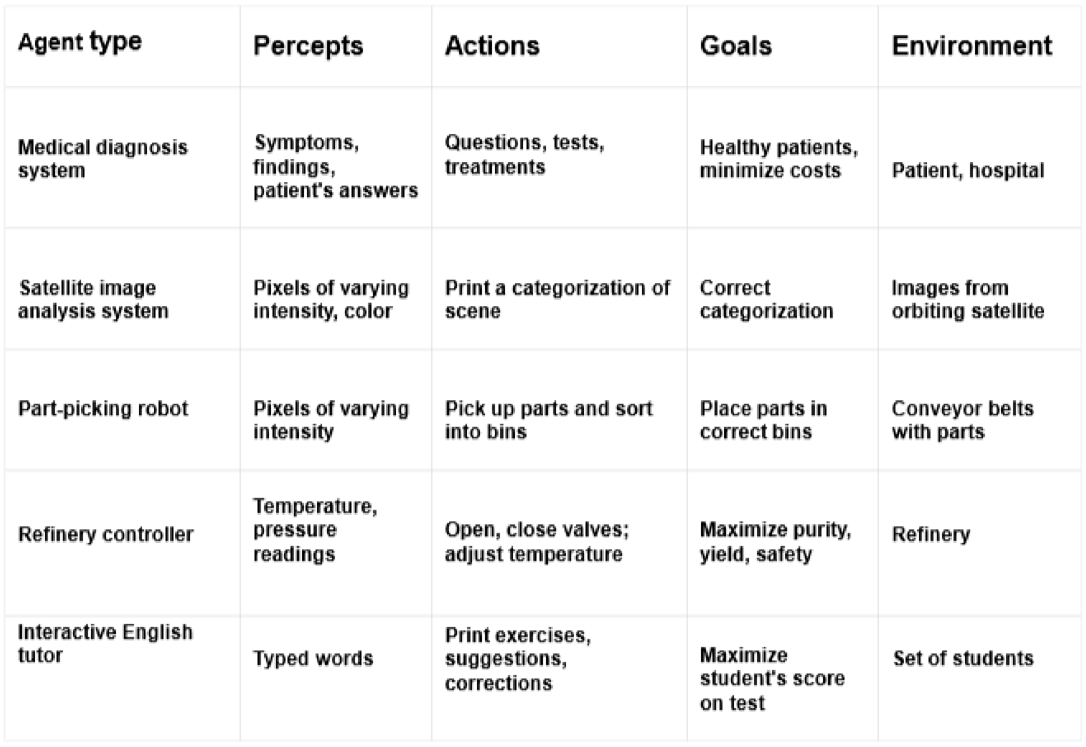
- a more complex environment, with obstacles
    - the agent sensor detects if there is an obstacle in front of it or if it has accidentally bumped into an obstacle
- another action is available for the agent: ShutOff (the robot shuts off the vacuum cleaner)
- the robot has a more complex goal
    - minimize the power
    - return in its initial position after it has removed all dirt
    - **learn** to clean the room (RL)
- multiple robots are cleaning the room (**MAS**)

Note

For implementing the simulation of the Simple Vacuum Cleaning World

- a logic based architecture may be chosen for the agent ([1], Section 1.4.1)
    - logic programming language: Prolog
- a reactive/deliberative architecture

Examples of agents in different types of applications



Agent type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patients, 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 belts 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

#	**Agent**	**P**	**A**	**G**	**E**	**S**
(1)	Simple vacuum cleaning	- dirt - wall	- suck up dirt - go forward - turn right 90°	Clean the room	Grid-like environment	Current configuration of the room
(2)	Autonomous driver	Images	- turn left - turn right …	Drive autonomously on a highway	Highway	Current configuration of the highway
(3)	Backgammon player	Board state	specific moves	Play backgammon	All possible board states	Board state
(4)	Softbot	Web page	file manipulation commands	Collect info on a subject	Internet	Web page

Softbots (Software Robots)

- Etzioni (1993, 1996)
  - determine the e-mail address of a person
  - search papers for a person
  - AI planning techniques
- MARVIN (1998) - retrieve medical information on Internet

4. Environment

Environment

- determines the interaction between the “outside world” and the agent
  - the “outside world” is not necessarily the “real world”
  - it may be a real or virtual environment the agent lives in

Properties of the environment

A. Accessible vs inaccessible.

- An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment’s state.
- Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible.
- The more accessible an environment is, the simpler it is to build agents to operate in it.

  E.g.

  Accessible – (3)

  Inaccessible – (1), (2)

B. Deterministic vs non-deterministic.

- A deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action.
- In a non-deterministic environment, a probability distribution function is required
- In a non-deterministic environment, there is a stochastic process acting on it.
  - The physical world can be regarded as non-deterministic.
  - Non-deterministic environments are more difficult for the agent designer.

  E.g.

  Deterministic – (1)

  Non-deterministic – (2), (3)

C. Episodic vs non-episodic.

- In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different episodes.
  - The agent can decide what action to perform based only on the current episode (no need to reason about the interaction between the current episode and the future ones).
- In a non-episodic (sequential) environment, the current decision could affect all future decisions “*short term actions can have long term consequences*”.
  - the agent has to plan its actions
  - planning
- Episodic environments are simpler from the agent developer’s perspective because the agent can decide what action to perform based only on the current episode — it need not reason about the interactions between this and future episodes.

  E.g.

  Episodic – (1)

  Sequential – (2), (3)

D. Static vs dynamic.

- A static environment is one that can be assumed to remain unchanged except by the performance of actions by the agent.
- A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control. The physical world is a highly dynamic environment.

  E.g.

  Static – (1), (3)
  Dynamic – (2)

E. Discrete vs continuous.

- An environment is discrete if there are a fixed, finite number of actions and percepts in it.
- *How do we **represent** or **abstract** or **model** the world?*

  E.g.

  Discrete – (1), (3)
  Continuous – (2)

Markovian vs non-Markovian.

- *Markov processes* (probabilities and statistics)
  - A process satisfies the Markov property of one can make predictions about the future state of the process based only on the current state (without reference to the history).
  - The conditional probability distribution of future states depends only upon the present state.

- **Markovian agent environment**
  - The environment response at time *t*+1 depends only on the state and action at time *t* $\Rightarrow$ the environment's dynamics can be defined by specifying $Pr(s_{t+1} = s' | s_t, a_t)$.
  - $Pr(s_{t+1} = s' | s_t, a_t) = Pr(s_{t+1} = s' | s_t, a_t, s_{t-1}, a_{t-1}, \dots s_0, a_0)$

Examples

- The *weather evolution* is a Markov process.
- The backgammon game is Markovian.
- A *machine breakdown* is non-Markovian, as it depends on the entire lifecycle of the machine, not only on the present condition.
- An example of a stochastic process that is non-Markovian: *An urn contains two red balls and one green ball. One ball was drawn yesterday, one ball was drawn today and one ball will be drawn tomorrow. All of the draws are without replacement.*
  - Suppose you know that the today's ball was red, but you have no information about yesterday's ball $\Rightarrow$ the chance that the tomorrow's ball will be red is 0.5.
  - If you know that the yesterday's ball was red, then you are guaranteed to get a green ball tomorrow.

- The agent environments can be modeled as being **Markovian**
- The agent may keep track (in its internal memory) of the current state of the environment (considering all the received percepts)
  - and thus it may act as if its environment is a **Markovian** one.

Single agent vs. Multiagent

- image analysis, crossword puzzle, taxi driving, chess
- *An agent operating by itself in an environment*
- *Does the other agent interfere with my performance measure?*
- *Interaction and collaboration among the agents?*
  - competitive, cooperative

PEAS

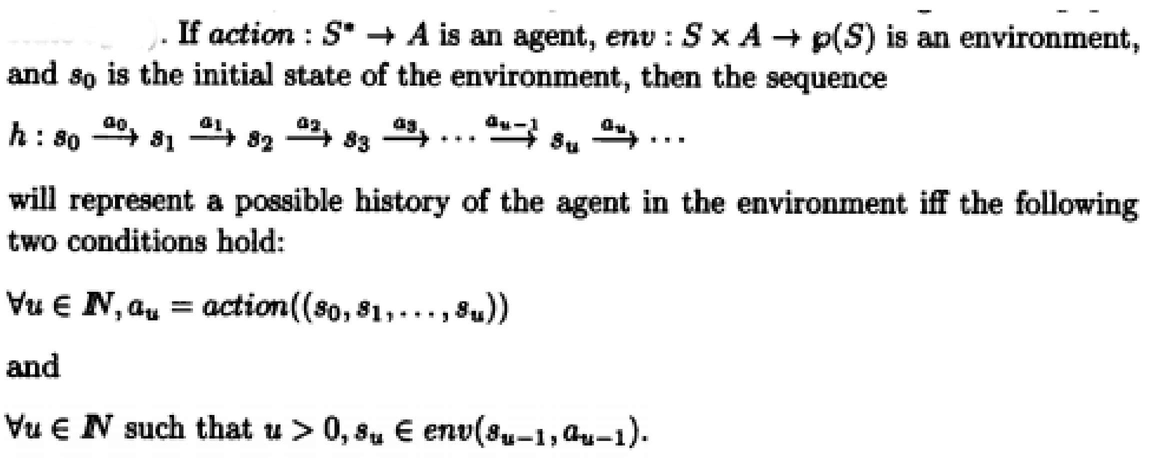
- has to be specified before the design of the agent, besides **PAGE(S)**
- **PEAS**
  - **Performance measure**
  - **Environment**
  - **Actuators**
    - actions performed through actuators
  - **Sensors**
    - perception through sensors

Examples

- **Agent = part-picking robot**
  - **P**: percentage of parts in correct bins
  - **E**: conveyor belt with parts, bins
  - **A**: jointed arm and hand
  - **S**: camera, joint angle sensors
- **Agent = taxi driver**
  - **P**: safe, fast, legal, comfortable trip, maximize profits
  - **E**: roads, other traffic, pedestrians, customers
  - **A**: Steering wheel, accelerator, brake, signal, horn
  - **S**: Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard
- **functions** as *sensors* and *actuators* for software agents

5. Abstract architectures for agents **([1], Section 1.3)**

- formalize the abstract view of agents
- environment states
  - $S = \{s_1, s_2, \ldots\}$
  - at a given moment, the environment is assumed to be in one of these states
- set of actions
  - $A = \{a_1, a_2, \ldots\}$
  - represent the effectoric capability of the agent

- abstractly, an agent can be viewed as a function
  - $action : S^* \rightarrow A$
  - maps sequence of environment states to actions
- the (non-deterministic) behavior of an environment can be modelled as a function
  - $env : S \times A \rightarrow \wp(S)$
  - maps the pair (***s***, ***a***) to a set of environment states ***env***(***s***, ***a***) that could result from performing action ***a*** in state ***s***
  - if all ***env***(***s***, ***a***) are singletons ⇒ the environment is deterministic
- the interaction of the agent and the environment can be represented as a history *h*
  - *h* is a sequence $h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \cdots \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} \cdots$
- 

  . If $action : S^* \rightarrow A$ is an agent, $env : S \times A \rightarrow \wp(S)$ is an environment, and s_0 is the initial state of the environment, then the sequence

  $$h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \cdots \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} \cdots$$

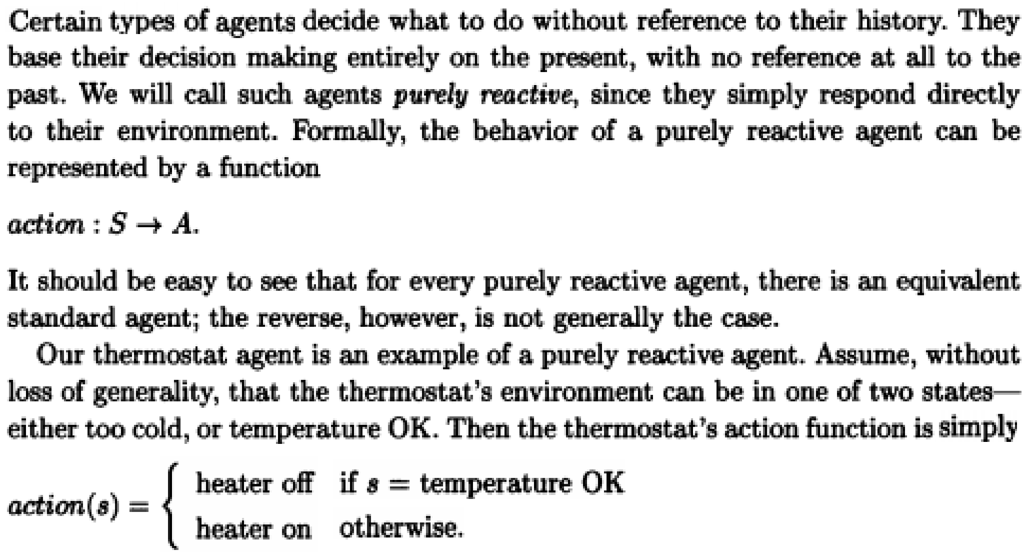
  will represent a possible history of the agent in the environment iff the following two conditions hold:

  $$\forall u \in \mathbb{N}, a_u = action((s_0, s_1, \ldots, s_u))$$

  and

  $$\forall u \in \mathbb{N} \text{ such that } u > 0, s_u \in env(s_{u-1}, a_{u-1}).$$
- **characteristic behavior of an agent** in an environment
  - set of all the histories that satisfy the previous properties

5.1 Purely reactive agents



Certain types of agents decide what to do without reference to their history. They base their decision making entirely on the present, with no reference at all to the past. We will call such agents *purely reactive*, since they simply respond directly to their environment. Formally, the behavior of a purely reactive agent can be represented by a function

$$action : S \rightarrow A.$$

It should be easy to see that for every purely reactive agent, there is an equivalent standard agent; the reverse, however, is not generally the case.

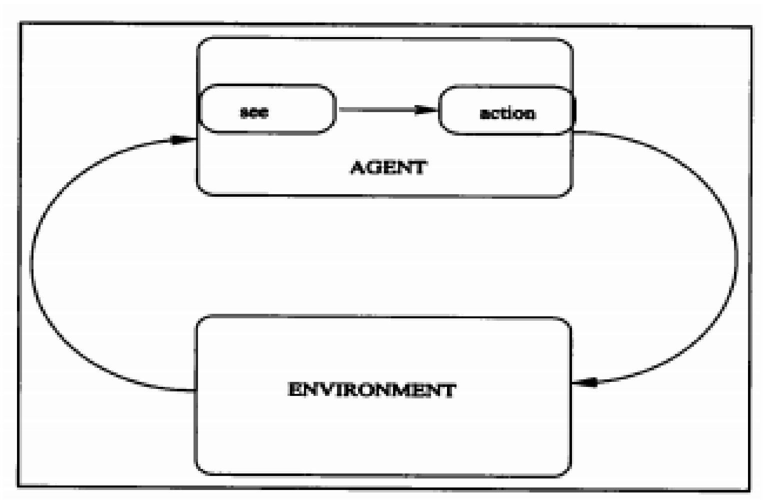
Our thermostat agent is an example of a purely reactive agent. Assume, without loss of generality, that the thermostat's environment can be in one of two states—either too cold, or temperature OK. Then the thermostat's action function is simply

$$action(s) = \begin{cases} \text{heater off} & \text{if } s = \text{temperature OK} \\ \text{heater on} & \text{otherwise.} \end{cases}$$

[1]

5.2 Perception

- the separation of the agent's decision function into **perception** and **action** subsystems.



Perception and action subsystems.

[1]

- if ***P*** is a (non-empty) set of percepts, the ***see*** and ***action*** function are as follows

$see : S \to P$

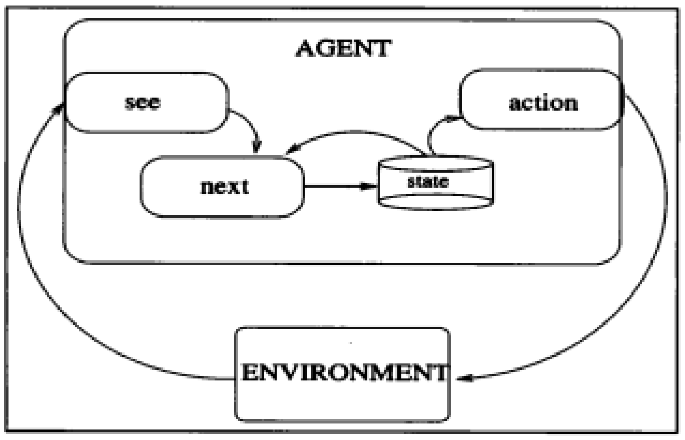
which maps environment states to percepts, and *action* is now a function

$action : P^* \to A$

which maps sequences of percepts to actions.

5.3 Agents with state (state-based agents)

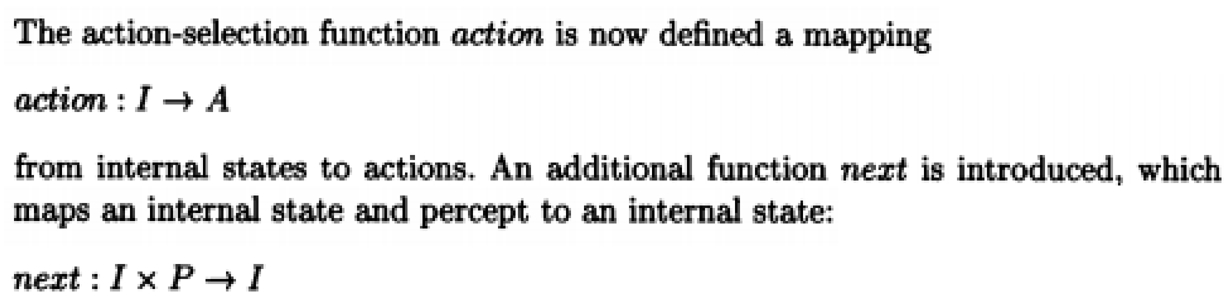
- these agents have an **internal state**



Agents that maintain state.

- ***I*** - the set of all internal states of the agent
- the perception function ***see*** for a state-based agent is unchanged, mapping environment states to percepts:

  $see : S \to P$



The action-selection function *action* is now defined a mapping

$action : I \to A$

from internal states to actions. An additional function *next* is introduced, which maps an internal state and percept to an internal state:

$next : I \times P \to I$

[1]

- the behavior of a state-based agent can be summarized as follows:
  (1) the agent starts in some initial internal state i_0
  (2) it then observes its environment state ***s*** and the percept ***see***(***s***)
  (3) the internal state of the agent is then updated
  $$i_0 \leftarrow next(i_0, see(s))$$
  (4) the action selected by the agent is then
  $$a \leftarrow action(next(i_0, see(s)))$$
  (5) the action ***a*** is performed and the agent enters another cycle - GO TO **(2)**

Bibliography

[1] Weiss, G. (Ed.): Multiagent Systems: *A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999 (available at www.cs.ubbcluj.ro/~gabis/weiss/Weiss.zip) [Ch. 1]
[2] Russell, J.S, Norvig, P., *Artificial Intelligence - A Modern Approach*, Prentice- Hall, Inc., New Jersey, 1995 (available at www.cs.ubbcluj.ro/~gabis/weiss) [Ch. 2, Ch. 6]