# COMS W4701: Artificial Intelligence

## Lecture 1b: Agents and Environments

Tony Dear, Ph.D.

Department of Computer Science

School of Engineering and Applied Sciences

# Today

- Task environments
- Agent programs
- Search problems

# Task Environments

- A rational agent is a *solution* to a **task environment** problem
- **PEAS**: Performance measure, environment, actuators, sensors

- Vacuum cleaner task environment
  - P: Cleanliness, power usage, time taken
  - E: The small grid world
  - A: Wheels to move, filter to clean
  - S: "GPS", cleanliness sensor

# Example: Self-Driving Taxi

- **P**: Correct navigation; minimizing fuel, trip time, cost, traffic violations; maximizing comfort, safety, profits
- **E**: Roads, other traffic, pedestrians, customers, weather
- **A**: Acceleration, steering, braking, communication with others
- **S**: Cameras, GPS, speedometer, accelerometer, odometer, etc.

# Task Environment Properties

- Fully observable vs partially observable vs unobservable
  - Can agent sense all *relevant* information? Is internal state (memory) required?
- Single-agent vs multi-agent
  - Does maximization of performance measure depend on other agents' behaviors?
- Deterministic vs stochastic
  - Can we completely predict the next state of the environment?
- Episodic vs sequential
  - Do current decisions depend on past ones? Do current decisions affect future ones?
- Static vs dynamic
  - Does the environment change while the agent is thinking?
- Discrete vs continuous
  - Is number of states, actions, percepts, time, etc. finite?

# Examples of Environments

| Environment | Partially / Fully Observable | Single- / Multi-Agent | Deterministic / Stochastic | Sequential / Episodic | Dynamic / Static | Continuous / Discrete |
|---|---|---|---|---|---|---|
| **Vacuum cleaner world** | Depends | Single | Deterministic | Sequential | Static | Discrete |
| **Chess** | Fully | Multi (adversarial) | Deterministic | Sequential | Static | Discrete |
| **Self-driving car** | Partially | Multi (cooperative) | Stochastic | Sequential | Dynamic | Continuous |
| **Image classification** | Fully | Single | Deterministic | Episodic | Static | Discrete |

# Agent Design

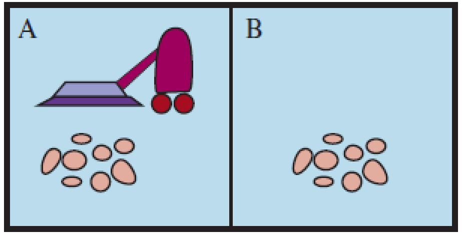
- Understanding the task environment tells us how to design our agents
- The more difficult the task environment, the more complex the agent

- Partially observable env -> agent requires memory / state
- Multi-agent env -> agent requires tracking of other agents

- Stochastic env -> agent must consider multiple scenarios or contingencies
- Sequential env -> agent must consider past and future states
- Dynamic env -> agent must maintain model of the world

# Agent Functions

- An **agent function** maps percept sequences to actions
- Cleaner robot percepts: Current square; is dirty?
- Actions: Move left, move right, clean, do nothing



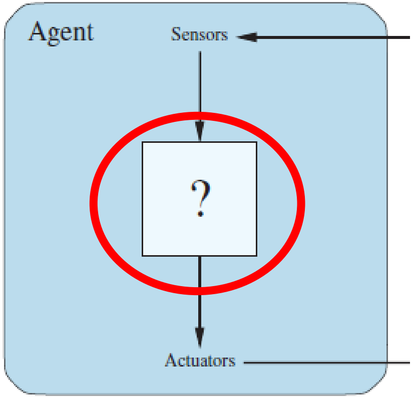

| | |
|---|---|
| [A, IsClean] | Move right |
| [A, IsDirty] | Clean |
| [B, IsClean] | Move left |
| [B, IsDirty] | Clean |
| [[A, IsDirty], [A, IsClean]] | Move right |
| [[B, IsDirty], [A, IsClean]] | Move right |
| [[B, IsClean], [A, IsClean]] | Do nothing |
| …… | …… |

# Agent Programs

- An **agent program** is an *implementation* of an agent function
- Specifies *how* something is computed rather than *what* needs to be computed

- A given agent program is not a universal solution!
- Program usefulness depends on hardware, tractability

- Agent programs also depend on the desired *solutions*
- E.g., *optimal* wrt some utility, approximately optimal, *satisficing* ("good enough"), *probable*

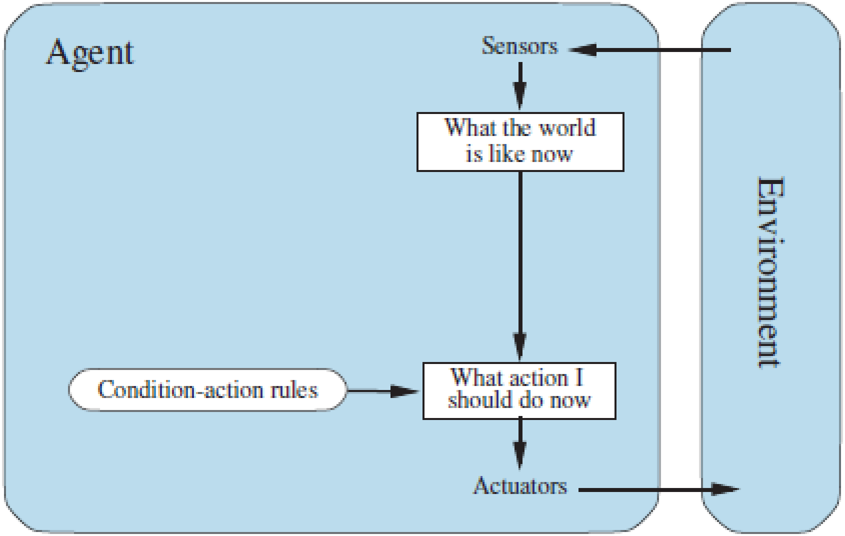


# Simple Reflex Agents

- Simple reflex agent: Use current percept only
- Percepts may be mapped to internal *states*
- Match state to condition-action (if-then) rules

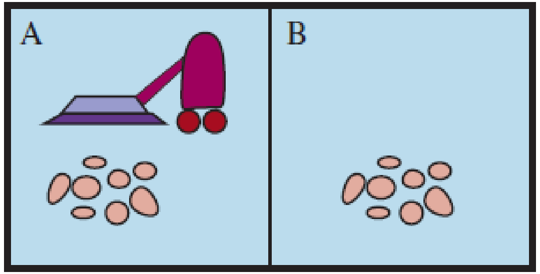
**function** SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  **persistent**: *rules*, a set of condition–action rules

  *state* ← INTERPRET-INPUT(*percept*)
  *rule* ← RULE-MATCH(*state*, *rules*)
  *action* ← *rule*.ACTION
  **return** *action*



# Example: Vacuum Cleaner Robot

- Example: Vacuum cleaner robot as a reflex agent
- Use only the *current* (no past) percept
- What is its resultant behavior?




**function** Reflex-Vacuum-Agent([location, status]) **returns** action

  **if** status **is** dirty **then return** *clean*

  **if** location **is** A **then return** *move right*

  **if** location **is** B **then return** *move left*

  **return** *do nothing*

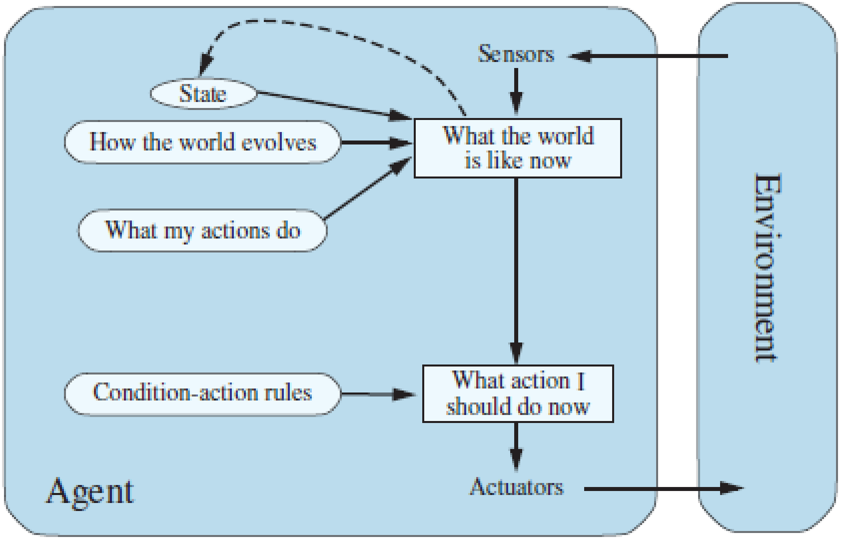
# Model-Based Reflex Agents

- What about partially observable environments?

- Maintain **internal state** and **transition model**
- May also have *sensor model* mapping percepts
- Use all information to *update* the state

**function** MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
  **persistent**: *state*, the agent's current conception of the world state
    *model*, a description of how the next state depends on current state and action
    *rules*, a set of condition–action rules
    *action*, the most recent action, initially none

  *state* ← UPDATE-STATE(*state*, *action*, *percept*, *model*)
  *rule* ← RULE-MATCH(*state*, *rules*)
  *action* ← *rule*.ACTION
  **return** *action*



# Example: Vacuum Cleaner Robot

- We can provide our vacuum cleaner with *state* to *remember* its past

internalState = [*dirty*, *dirty*]

**function** Model-Based-Vacuum-Agent([location, status]) **returns** action

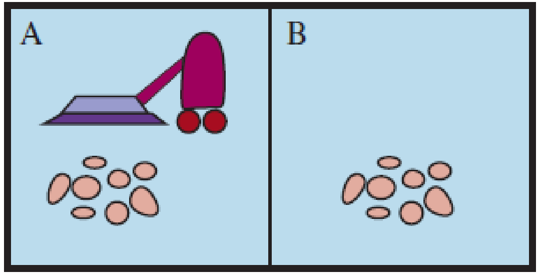
**if** status **is** dirty **then return** *clean*

**else** internalState[location] = *clean*

**if** internalState[B] **is** *dirty* **then return** *move right*

**if** internalState[A] **is** *dirty* **then return** *move left*

**return** *do nothing*



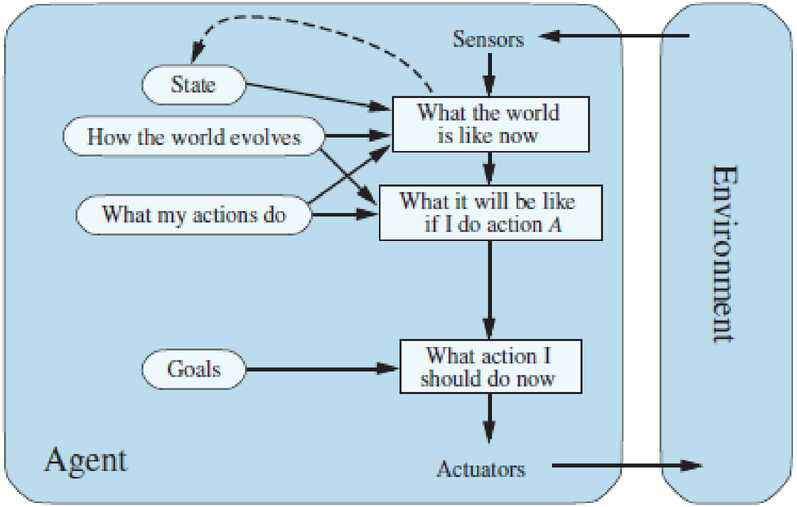
# Goal- and Utility-Based Agents

- Reflex agents are very rigid and predictable

- **Goal-based** agents try to achieve particular states
- Problems often solved using search and planning

- **Utility-based** agents can compare different states
- Utility functions map state to "desirability"
- Internalize the overall performance measure

- Utilities specify tradeoffs for competing goals
- Also useful in face of uncertainty



# Goals and Utilities

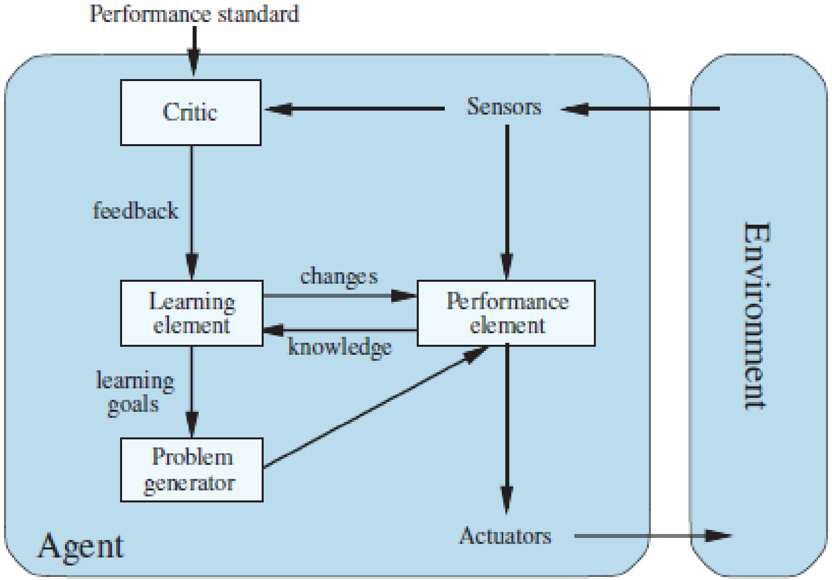
- Designing effective goal and/or utility functions is a hard problem
- May also be framed as objective, cost, or loss functions
- May be simple and concrete, or complex and abstract

- May be assigned explicitly, i.e., rewards in reinforcement learning
- May be implicit, i.e., perform similarly to training data examples

- Often captures tradeoffs between conflicting objectives

# Learning Agents

- **Learning agents** can be used to *create* or *improve* upon initial models in unknown envs
- A *learning element* retrieves knowledge from and then improves the *performance element*
- A *critic* evaluates the learning element according to a *performance standard* (measure)
- A *problem generator* suggests actions that can help gather new information and experiences



# Search Problems

- We start with task environments with the following properties:
- **Fully observable**, **single-agent, deterministic**, **static, discrete**
- We have “perfect” percepts and a known transition model

- *States* describe agent’s possible configurations as well as the goal
- Given problem description, the agent *searches* for a *state sequence* to the goal

- There may be *costs* incurred when going from one state to another
- An **optimal** solution minimizes total cost among all possible state sequences
- A **satisficing** solution may not be optimal, but is “good enough” with certain bounds

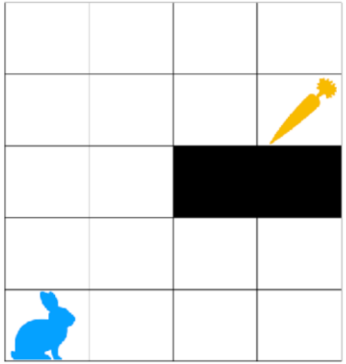
# Search Problems

- State space $S$: Set of descriptions of the agent and environment
- Initial state and one or more goal states

- Goal test $S \rightarrow \{True, False\}$, e.g., $isGoal(s_1) = False$
- Action set for each state, e.g., $Actions(s_1) = \{a_1, a_2, a_3\}$

- Transition model (function) $S \times A \rightarrow S$, e.g., $Result(s_1, a_1) = s_2$
- State-action cost function $S \times A \rightarrow \mathbb{R}$, e.g., $Cost(s_1, a_1) = 10$

# Example: Grid World Path Finding



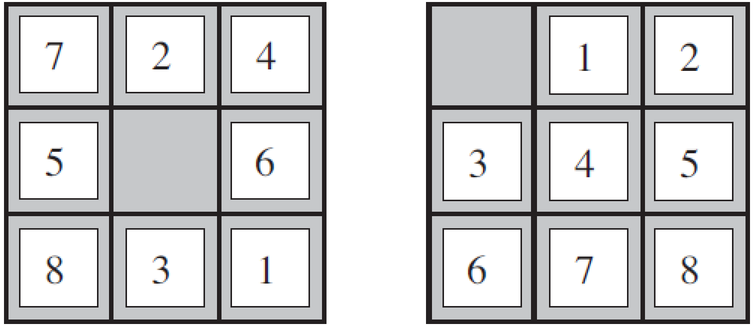
- State space: Current coordinates of the rabbit
  - $S = \{(x, y) \mid x \in \{0,1,2,3\}, y \in \{0,1,2,3,4\}\}$

- Goal test: $isGoal((3,3))$?
- Actions: $Actions\big((x, y)\big) = \{\text{Up}, \text{Down}, \text{Left}, \text{Right}\}$

- Costs: $Cost(s, a) = 1, \forall s, a$

- Transition model: $Result((x, y), \text{Up}) = (x, y + 1)$, $Result((x, y), \text{Down}) = \cdots$
  - Can also account for walls and boundaries, e.g. $Result\big((0,0), Left\big) = (0,0)$

# Example: $n$-puzzle




Start State Goal State

- **State**: Locations of all tiles and blank
- **Action**: 4 possible directions for the blank tile
- **Cost**: Each step taken costs 1
- **Goal test**: Is current state equal to goal state?

# More Search Problems

- Route-finding (e.g., vehicle navigation), robot navigation in the real world
- Touring problems (traveling salesperson)

- Layout and assembly sequencing problems
- Mathematical puzzles and proofs: Infinitely large state spaces!

- Knuth's conjecture (1964): Any positive integer can be obtained using a combination of factorial, floor, and sqrt operations on the input 4
- Ex: $101 = \text{floor}\left(\left(\sqrt{4!}\right)!\right)$

# Summary

- PEAS descriptors define task environments and influence agent design
- Environment properties vary from easy to extremely challenging

- Agent programs may use current percept only, use a model, and/or try to achieve certain goals quantified by utilities
- Agent programs may also be created or improved via learning

- Problem-solving agents solve search problems to find discrete sequences of states and actions between initial and goal states