Week 3 — Intelligent Agents & Environments

This lecture introduces core agent architectures (Simple Reflex, Model-Based, Goal-Based, Utility-Based) and the environments they act in. We'll define each agent type, explain architecture and behavior, show a compact code example, and illustrate with clear real-world photos. Language is simple and educational — content is written so you can read the slide as a standalone learning unit.

Theme colors used for emphasis: #876cd4ff (primary), #D783D8, #FF90A5, #FFB071, #14083A. Images are real-world photos (not clip art) to ground concepts.

What is an Intelligent Agent?

Definition: An intelligent agent is any entity that perceives its environment through sensors and acts upon that environment through actuators to achieve goals. It can be software (chatbot, recommender) or physical (robot, self-driving car).

Core components

- Percepts — raw inputs from sensors (images, text, readings)
- Agent function — maps percepts to actions
- Actuators — ways agent changes the world (motors, network calls)
- Environment — everything outside the agent

Simple view

Agent = Perceive -> Decide -> Act. We design agents by choosing representations, decision rules, and objective measures of success.

Quick example domains: web search engine (software agent), thermostat (embedded agent), delivery drone (robotic agent). Images show real deployments to connect theory to practice.

Simple Reflex Agents — Definition & Architecture

Definition: A Simple Reflex Agent selects actions solely based on the current percept, using condition-action rules (if condition then action). It has no memory of past percepts and no internal model of the world.

Architecture

Perception -> Condition-Action Rules -> Action. The rule set directly maps specific percept patterns to actions (e.g., if obstacle detected then turn right). This architecture is cheap and fast but limited: it fails when the correct action depends on history or unobserved state.

When to use

- Environments that are fully observable and simple
- Cost or latency constraints where quick reflexes matter

Code example (Python pseudo-code)

```
def simple_reflex_agent(percept):
    if percept['front'] == 'obstacle':
        return 'turn_right'
    if percept['goal_seen']:
        return 'move_forward'
    return 'search'
```

Note: This minimal code shows decision by matching current sensing to actions. It illustrates why history or inference is impossible with this agent.

Model-Based Agents — Definition & Architecture

Definition: A Model-Based Agent maintains an internal model of the world (state) that it updates over time. It uses the model plus current percepts to choose actions.

Architecture

- Percepts -> Update Model -> Deliberation/Decision -> Action
- Model stores hidden state, predictions, and beliefs about unobserved facts

Advantages

Handles partially observable environments and temporal dependencies. Enables planning and prediction because the agent reasons about future states.

Code example (Python pseudo-code)

```
class ModelBasedAgent:
    def __init__(self):
        self.model = {}   # internal state

    def update_model(self, percept):
        # incorporate percept into model (e.g., map update)
        self.model.update(percept['observations'])

    def decide(self):
        # simple planner: choose action to reduce distance to goal
        if self.model.get('obstacle_ahead'):
            return 'navigate_around'
        return 'move_forward'
```

Real systems add probabilistic models (belief distributions) and more complex planners.

Goal-Based Agents — Definition & Architecture

Definition: A Goal-Based Agent chooses actions to achieve explicit goals. Unlike reflex agents, it evaluates future consequences and selects actions that lead to goal achievement.

Architecture

- Percepts -> Update Model -> Goal Module -> Search/Planner -> Action
- Goal: a desired world state or condition (e.g., reach location X)

Key ideas

The agent can compare possible action sequences by how well they achieve the goal. This allows flexible behaviour and recovery strategies if the environment changes.

Code example (Python pseudo-code)

```
def goal_based_decision(model, goal):
    # naive breadth-first search for a short plan
    frontier = [[model['state']]]
    while frontier:
        path = frontier.pop(0)
        state = path[-1]
        if satisfies_goal(state, goal):
            return extract_first_action(path)
        for action in available_actions(state):
            frontier.append(path + [apply(action, state)])
    return 'no_op'
```

In practice planners use heuristics (A*, greedy best-first) and integrate with models.

Utility-Based Agents — Definition & Architecture

Definition: A Utility-Based Agent selects actions to maximize an internal utility function — a numeric measure of how desirable a state is. Utility combines multiple preferences (safety, speed, comfort) into a single score.

Architecture

- Percepts -> Update Model -> Evaluate Utilities of Outcomes -> Choose Action
- Utility function assigns numbers; agent picks action with highest expected utility

When useful

When trade-offs are necessary (e.g., risk vs. reward), or when we must compare non-commensurate goals. Utility agents handle stochastic and multi-objective choices.

Code example (Python pseudo-code)

```
def expected_utility(action, model):
    outcomes = simulate(action, model)
    return sum(p * utility(o) for (o,p) in outcomes)

def choose_action(actions, model):
    return max(actions, key=lambda a: expected_utility(a, model))
```

Utility functions can be learned from data or designed by hand; they can include risk-aversion or time-discounting.

What is an Environment in AI?

Definition: The environment is everything external to the agent that the agent interacts with. It provides percepts and receives actions. Designing agents requires carefully modeling the environment's properties.

Percepts

What the agent senses (camera images, sensor readings, messages).

Actuators

How the agent affects the world (motors, network requests, display outputs).

State

True condition of the environment (may be partially observable).

We categorize environments by formal properties (deterministic vs stochastic, episodic vs sequential, static vs dynamic, discrete vs continuous, etc.). These characteristics determine appropriate agent architectures.

Key Environment Properties & Examples

We'll cover core properties and give short examples that show why each matters when selecting agent architecture.

Observable

Fully observable: agent's sensors capture the entire state (e.g., chess). Partially observable: sensors miss hidden variables (e.g., self-driving car with occluded pedestrians).

Deterministic vs Non-deterministic

Deterministic: next state fully determined by current state and action (e.g., classic puzzle games). Non-deterministic/stochastic: actions have probabilistic outcomes (e.g., robot on slippery floor).

3

Episodic vs Sequential

Episodic: each decision is independent (e.g., image classification). Sequential: current decisions affect future ones (e.g., chess, driving).

Static vs Dynamic

Static: environment doesn't change while agent plans (e.g., crossword solver). Dynamic: world changes independent of the agent (e.g., stock market, traffic).

Accessible vs Inaccessible & More Examples

Accessible vs Inaccessible: - Accessible environment: agent can obtain any state information given enough time (e.g., a local database with full records). - Inaccessible: some parts of the environment can never be known by the agent (e.g., opponents' hidden cards in poker).

Practical examples mapped to agent types: - Simple Reflex: suitable for accessible, fully observable, and deterministic tasks with short-term reactions (e.g., thermostat). - Model-Based: needed when partial observability or hidden state exists (e.g., map building robots). - Goal-Based: good for complex tasks requiring planning (e.g., path planning, logistics). - Utility-Based: best when trade-offs and preferences exist (e.g., autonomous vehicles balancing speed vs safety).

Summary, Study Tips & Next Steps

Step 1 — Master Definitions

Memorize concise definitions: Simple Reflex, Model-Based, Goal-Based, Utility-Based, and environment properties. Use flashcards with photos for recall.

Step 2 — Map Architectures to Problems

Practice by listing problems (robot vacuum, chess, recommender) and decide the best agent type. Explain why in one paragraph.

Step 3 — Implement Small Examples

Code the provided pseudo-examples in Python. Extend them with logging and small simulators to see behavior in deterministic and stochastic settings.

Step 4 — Visualize Environments

Draw environment diagrams (state, actions, transitions). Compare episodic vs sequential and note where models or utilities are required.

Extra Resources

- Russell & Norvig — "Artificial Intelligence: A Modern Approach"
- Introductory robotics and reinforcement learning tutorials
- Interactive simulators (OpenAI Gym, simple grid-worlds)

Final tip: Start simple (reflex), add memory/model when needed, convert to goal-driven planning, and use utility when balancing trade-offs. Practice by building small agents in simulated environments to see these ideas come alive.