Artificial Intelligence

CSC411

Lecture Set-02

Intelligent AI Agents

Dr. Ali Asghar Manjotho

Assistant Professor, CSE-MUET

ali.manjotho@faculty.muet.edu.pk
ali.manjotho.ali@gmail.com

22 BSCS

Contents

Contents

- Comparison of AI Agent Types
- When to use each AI Agent Type
- Maze Grid Navigation Agent
  - Simple Reflex Agent
  - Model-Based Agent
  - Goal-Based Agent
  - Utility-Based Agent
- Structure of Intelligent Agent
- Agents VS Expert Systems
- Environment

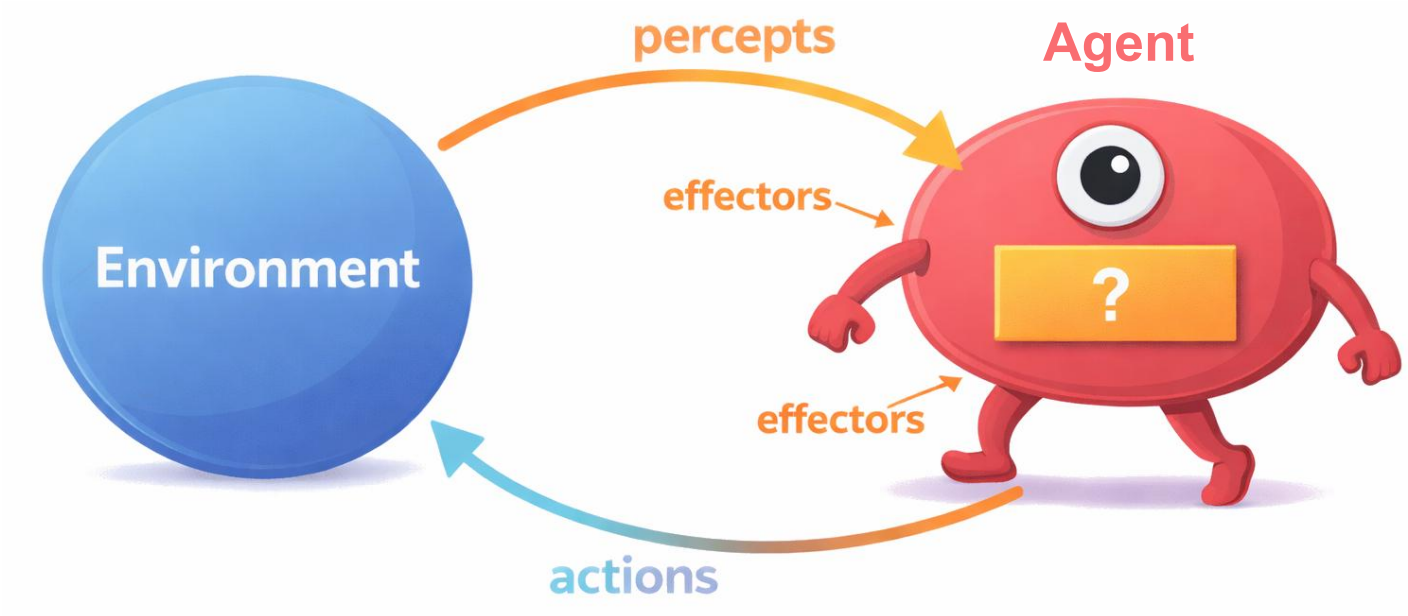
Contents

- Properties of Environment
- Examples of Agent Types and PAGE Descriptors
- Automated Taxi Driver Agent Example

Intelligent AI Agents

- An **agent** is anything that can be viewed as **perceiving** its environment through sensors (real or virtual) and **acting** upon that environment through effectors (real or virtual).
- Examples:
  - **Human Agent**
    - Sensors: Ears, Eyes, Tongue, Nose, and Skin.
    - Effectors: Hands, Legs, Feet and Mouth
  - **Navigating Robot**
    - Sensors: Cameras, Infrared range finders.
    - Effectors: Motors, Mechanical hands and legs, Robotic arm.

Intelligent AI Agents

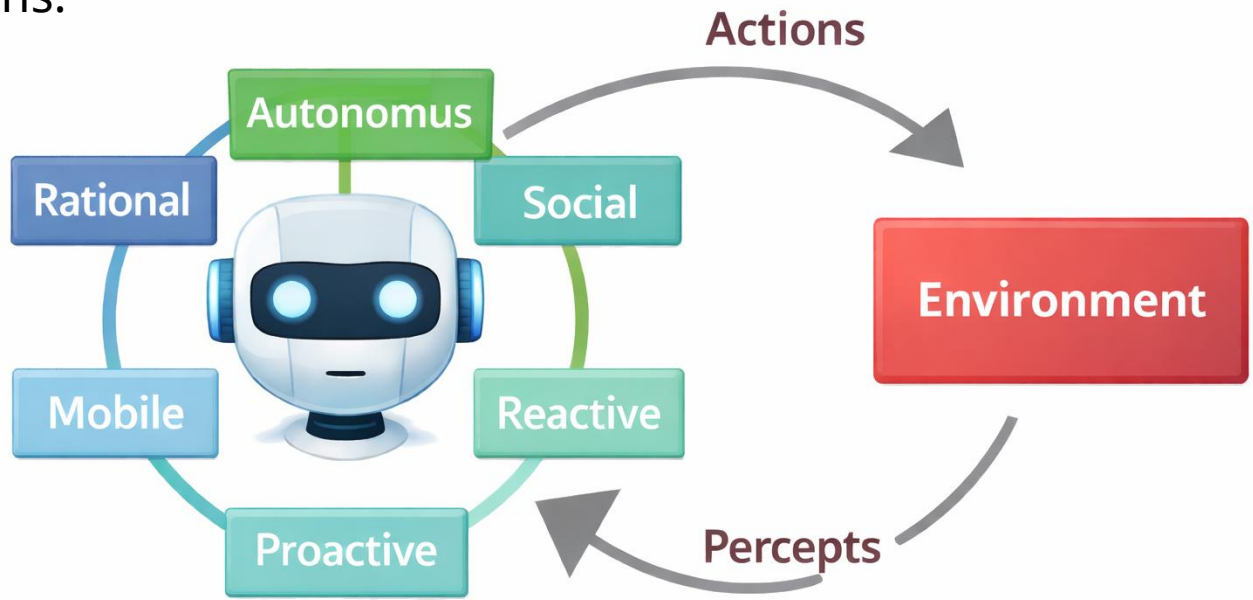


Intelligent AI Agents

- A software system that uses AI to perform tasks, pursue goals, and make decisions autonomously.
- AI agents learn from data and user feedback, adapting over time.
- They act on behalf of users with minimal human input.
- AI agents are software systems that use AI to pursue goals and complete tasks on behalf of users. They show reasoning, planning, and memory and have a level of autonomy to make decisions, learn, and adapt.

Software Agent

- A software agent has encoded bit strings as its percepts and actions.



Software Agent

- **Autonomous:** Operates by itself without requiring a constant user intervention.
- **Social:** Cooperates with other software agents or human users in order to fulfil a series of tasks.
- **Reactive:** Its actions are a direct consequence of what it perceive from the environment.
- **Proactive:** Its actions try to anticipate modification that may affect the environment.
- **Mobile:** Can migrate from one computational node/platform to another.
- **Rational:** Acts rationally using formal logic.

Basic Types of AI Agents

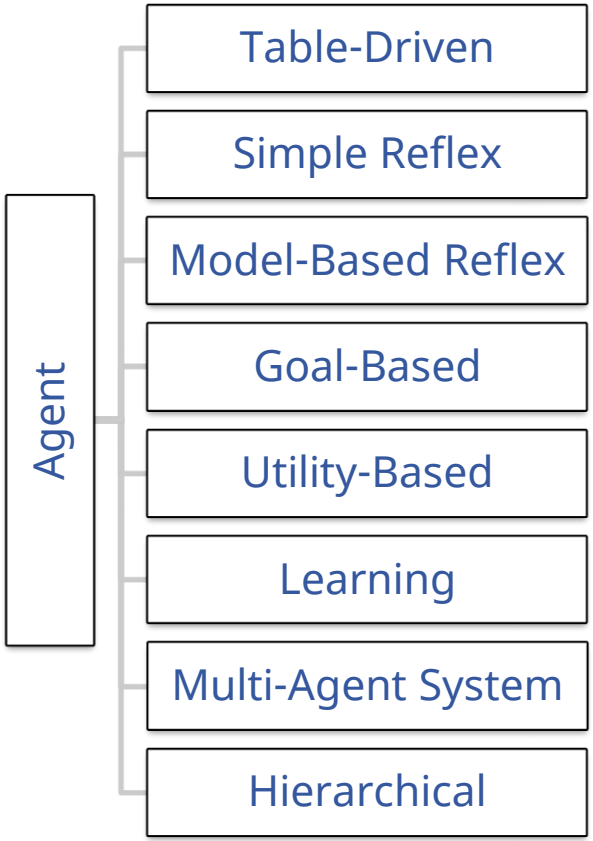


Table-Driven Agents

- They are implemented by a (large) lookup table.
- Table lookup of percept-action pairs mapping from every possible perceived state to the optimal action for that state.

Problems:

- Too big to generate and to store (Chess has about 10^{120} states).
- No knowledge of non-perceptual parts of the current state.
- Not adaptive to changes in the environment; requires entire table to be updated if changes occur.
- Can’t make actions conditional on previous actions/states.

Table-Driven Agents

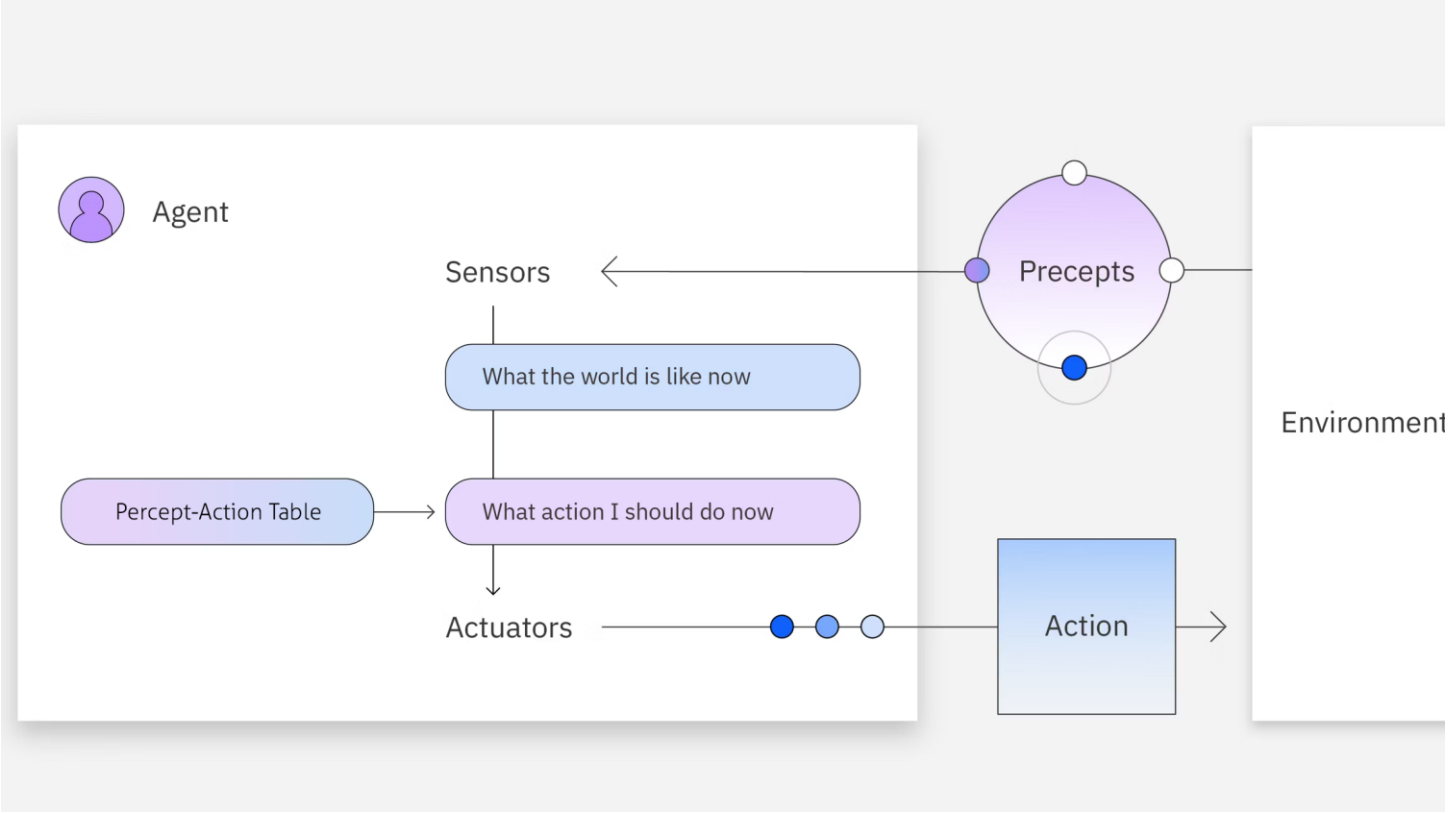


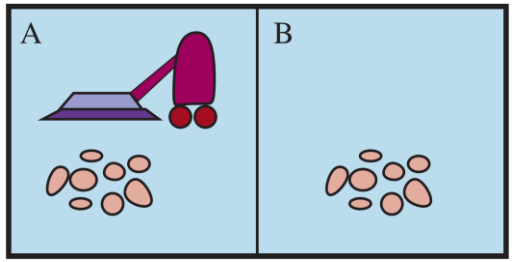
Table-Driven Agent (Two Room Vacuum World)

- A vacuum-cleaner world with just two locations.
- Each location can be **clean** or **dirty**.
- The agent can move left or right and can clean the room/square.

Environment: Two Rooms	**Sensors:** Optical, IR
Agent: Vacuum Cleaner	**Actuators:** Pressure pump, wheels

Percepts:

```
{'A': 'Clean'}
{'A': 'Dirty'}
{'B': 'Clean'}
{'B': 'Dirty'}
```



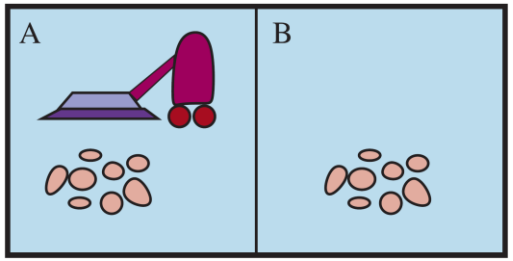
Actions:

```
Left
Right
Suck
NoOp
```

```
Env. Status: {'A': 'Dirty', 'B': 'Dirty', 'agent_loc': 'B'}
```

Table-Driven Agent (Two Room Vacuum World)

This is partial tabulation, in order to create concrete AI agent program , we need full table (table-driven agents are infeasible in real environments).

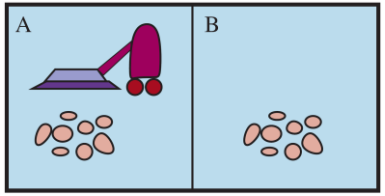



```
Percepts:                Actions:
{'A': 'Clean'}           Left
{'A': 'Dirty'}           Right
{'B': 'Clean'}           Suck
{'B': 'Dirty'}           NoOp
```

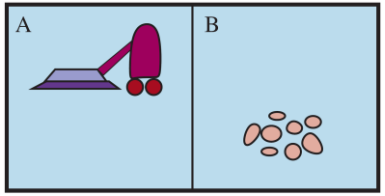
Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
…	…
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
…	…

Partial tabulation of a simple agent function for the vacuum-cleaner world

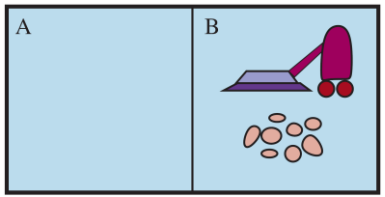
Table-Driven Agents (Two Room Vacuum World)



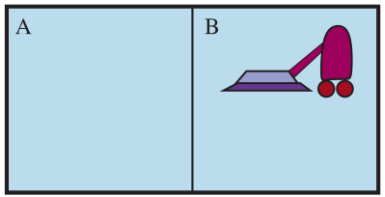

```
Percept Seq.: {'A': 'Dirty'}
Action: Suck
Status: {'A': 'Clean', 'B': 'Dirty', 'agent_loc': 'A'}
```




```
Percept Seq.: {'A': 'Dirty'}{'A': 'Clean'}
Action: Right
Status: {'A': 'Clean', 'B': 'Dirty', 'agent_loc': 'B'}
```




```
Percept Seq.: {'A': 'Dirty'}{'A': 'Clean'}{'B': 'Dirty'}
Action: Suck
Status: {'A': 'Clean', 'B': 'Clean', 'agent_loc': 'B'}
```




```
Percept Seq.: {'A': 'Dirty'}{'A': 'Clean'}{'B': 'Dirty'} '}{'B': 'Clean'}
Action: NoOp
Status: {'A': 'Clean', 'B': 'Clean', 'agent_loc': 'B'}
```

Table-Driven Agents (Two Room Vacuum World)

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Dirty], [A, Clean]	Right
[A, Dirty], [A, Clean] [B, Dirty]	Suck
[A, Dirty], [A, Clean] [B, Dirty] [B, Clean]	NoOp
…	…
[A, Clean], [A, Clean], [A, Clean]	Right
…	…

Simple Reflex Agents

- Simple reflex agents act based solely on **current perceptions** using condition-action rules.
- These agents respond directly to input without considering past experiences or potential future states.
- They operate on basic "`if-then`" logic: if a specific condition is detected, execute a corresponding action.

Key Features:

- No memory of past states
- No model of how the world works
- Purely reactive behavior
- Function best in fully observable environments

Simple Reflex Agents

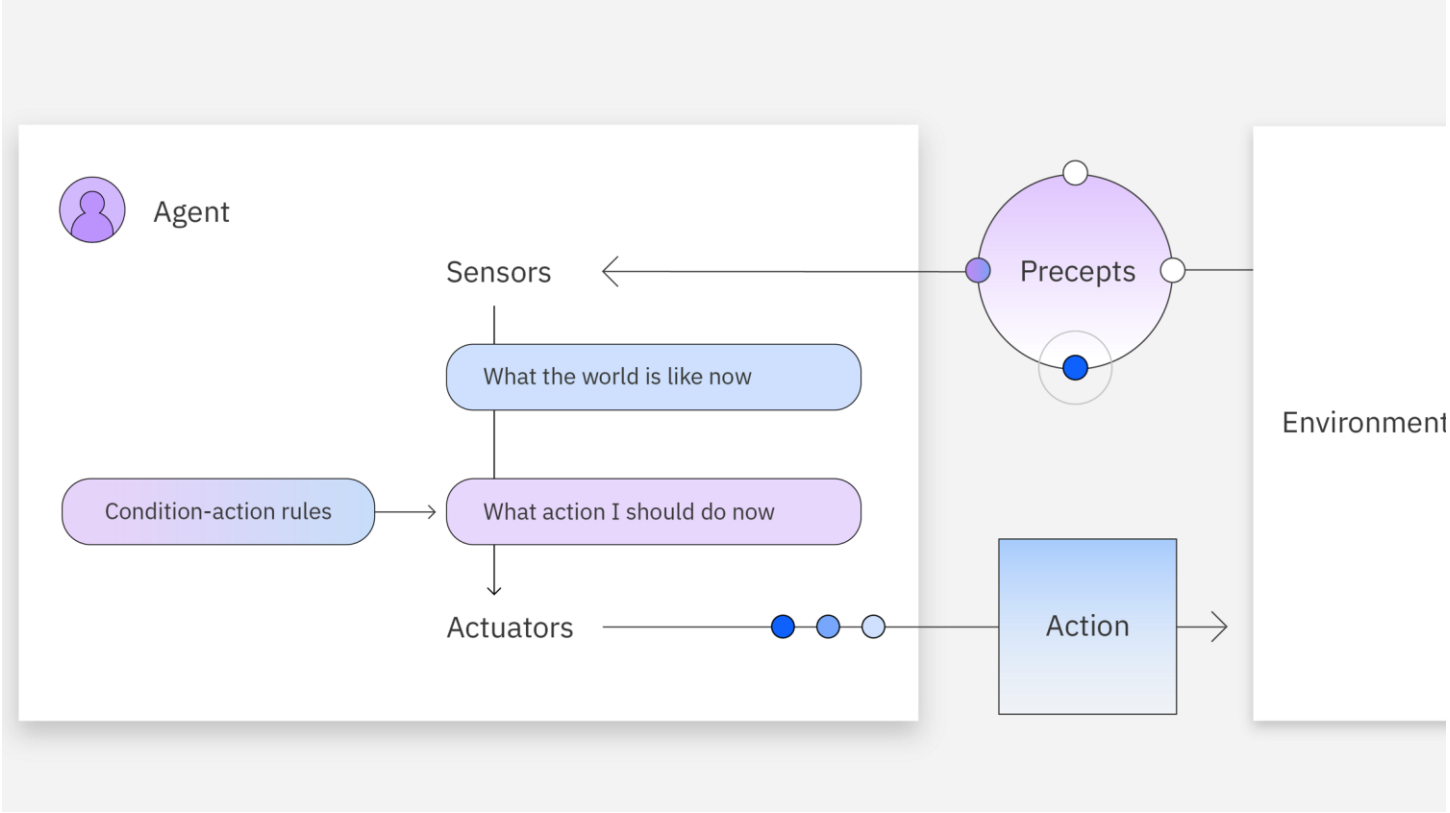
Problems:

- Still usually too big to generate and to store
- Still no knowledge of non-perceptual parts of state
- Still not adaptive to changes in the environment; requires collection of rules to be updated if changes occur
- Still can’t make actions conditional on previous state

Example: Traffic light control systems that change signals based on fixed timing.

When to Use: They are ideal in controlled, well-defined environments.

Simple Reflex Agents



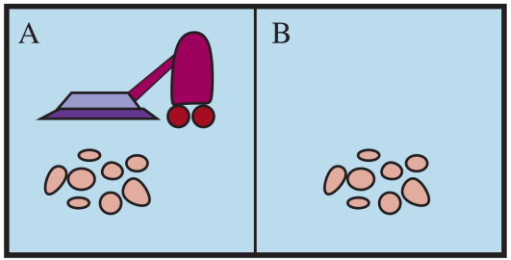
Simple Reflex Agents (Two Room Vacuum World)

- A vacuum-cleaner world with just two locations.
- Each location can be **clean** or **dirty**.
- The agent can move left or right and can clean the room/square.

Environment: Two Rooms	**Sensors:** Optical, IR
Agent: Vacuum Cleaner	**Actuators:** Pressure pump, wheels

Percepts:

```
{'A': 'Clean'}
{'A': 'Dirty'}
{'B': 'Clean'}
{'B': 'Dirty'}
```



Actions:

```
Left
Right
Suck
NoOp
```

Condition-Action Rules:

```
IF Dirty THEN Suck
IF loc=A THEN Right
IF loc=B THEN Left

IF A is Clean AND B is
Clean THEN NoOp
```

```
Status: {'A': 'Dirty', 'B': 'Clean', 'agent_loc': 'A'}
```

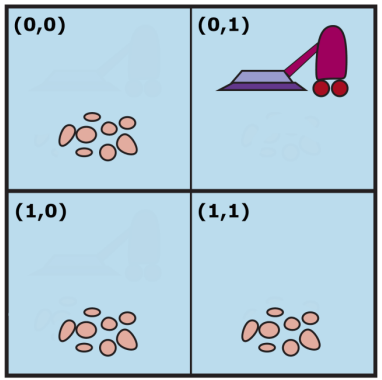
Simple Reflex Agents (2x2 Grid Vacuum World)

- A vacuum-cleaner world with 4 rooms in 2x2 grid.
- Each location can be **clean** or **dirty**.

Environment: 2x2 Room Grid	**Sensors:** Optical, IR
Agent: Vacuum Cleaner	**Actuators:** Pressure pump, wheels

Percepts:

```
{(0,0): 'Clean'}
{(0,0): 'Dirty'}
{(0,1): 'Clean'}
{(0,1): 'Dirty'}
{(1,0): 'Clean'}
{(1,0): 'Dirty'}
{(1,1): 'Clean'}
{(1,1): 'Dirty'}
```

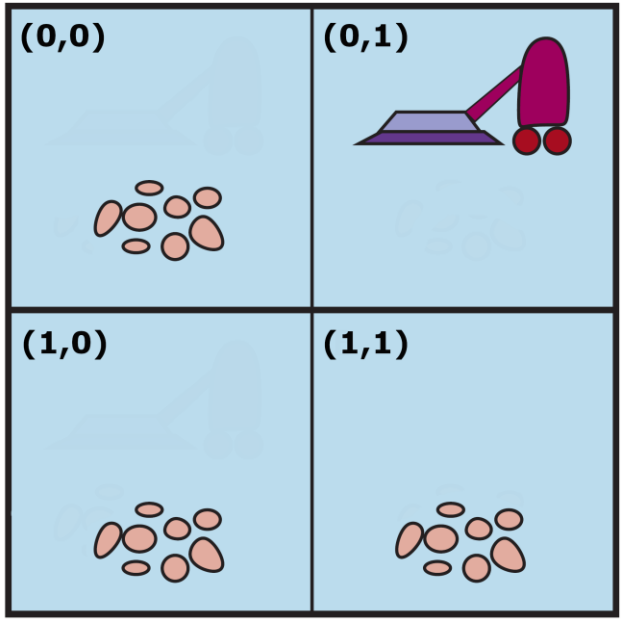


Actions:

Left
Right
Up
Down
Suck
NoOp

Status: `{(0,0): 'Dirty', (0,1): 'Clean', (1,0): 'Dirty', (1,1): 'Dirty', 'agent_loc': (0,1)}`

Simple Reflex Agents (2x2 Grid Vacuum World)

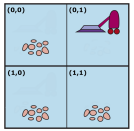


Condition-Action Rules:

```
IF Dirty THEN Suck
IF loc=(0,0) THEN Right
IF loc=(0,1) THEN Down
IF loc=(1,1) THEN Left
IF loc=(1,0) THEN Up

IF (0,0) is Clean AND (0,1) is Clean AND (1,0) is Clean AND (1,1) is
Clean THEN NoOp
```

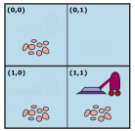
Simple Reflex Agents (2x2 Grid Vacuum World)

Step 1



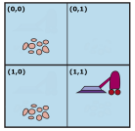
```
Percept: {(0,1): 'Clean'}
Action: Down
Status: {(0,0): 'Dirty', (0,1): 'Clean', (1,0): 'Dirty', (1,1): 'Dirty', 'agent_loc': (1,1)}
```

Step 2



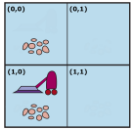
```
Percept: {(1,1): 'Dirty'}
Action: Suck
Status: {(0,0): 'Dirty', (0,1): 'Clean', (1,0): 'Dirty', (1,1): 'Clean', 'agent_loc': (1,1)}
```

Step 3



```
Percept: {(1,1): 'Clean'}
Action: Left
Status: {(0,0): 'Dirty', (0,1): 'Clean', (1,0): 'Dirty', (1,1): 'Clean', 'agent_loc': (1,0)}
```

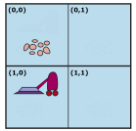
Step 4



```
Percept: {(1,0): 'Dirty'}
Action: Suck
Status: {(0,0): 'Dirty', (0,1): 'Clean', (1,0): 'Clean', (1,1): 'Clean', 'agent_loc': (1,0)}
```

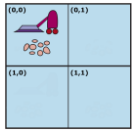
Simple Reflex Agents (2x2 Grid Vacuum World)

Step 5



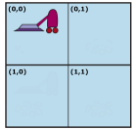
```
Percept: {(1,0): 'Clean'}
Action: Up
Status: {(0,0): 'Dirty', (0,1): 'Clean', (1,0): 'Clean', (1,1): 'Clean', 'agent_loc': (0,0)}
```

Step 6



```
Percept: {(1,1): 'Dirty'}
Action: Suck
Status: {(0,0): 'Dirty', (0,1): 'Clean', (1,0): 'Dirty', (1,1): 'Clean', 'agent_loc': (1,1)}
```

Step 7



```
Percept: {(0,0): 'Clean'}
Action: NoOp
Status: {(0,0): 'Clean', (0,1): 'Clean', (1,0): 'Clean', (1,1): 'Clean', 'agent_loc': (0,0)}
```

Simple Reflex Agents (Two Room Vacuum World)

- We will create the following files in Python:

agent.py	Defines Agent base class
environment.py	Defines Environment base class
vacuum_world.py	Two room vacuum cleaner environment extending Environment
table_driven_vacuum_agent.py	Concrete table-driven agent extending Agent
simple_reflex_vacuum_agent.py	Concrete simple reflex agent extending Agent
simulation.py	Generic episode runner that works for any environment + agent
demo.py	Defines main function that runs the demo

Simple Reflex Agents (Two Room Vacuum World)

agent.py

```python
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import Generic, TypeVar

PerceptT = TypeVar("PerceptT")
ActionT = TypeVar("ActionT")

class Agent(ABC, Generic[PerceptT, ActionT]):

    @abstractmethod
    def act(self, percept: PerceptT) -> ActionT:
```

Simple Reflex Agents (Two Room Vacuum World)

environment.py

```python
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import Generic, TypeVar, Any, Dict

PerceptT = TypeVar("PerceptT")
ActionT = TypeVar("ActionT")


class Environment(ABC, Generic[PerceptT, ActionT]):

    @abstractmethod
    def reset(self) -> None:

    @abstractmethod
    def percept(self) -> PerceptT:

    @abstractmethod
    def step(self, action: ActionT) -> None:

    def is_done(self) -> bool:

    def get_state(self) -> Dict[str, Any]:

    def render(self) -> None:
```

Simple Reflex Agents (Two Room Vacuum World)

simple_reflex_vacuum_agent.py

```python
from __future__ import annotations
from agent import Agent
from vacuum_world import Percept, Action, SUCK, LEFT, RIGHT, NOOP

class SimpleReflexVacuumAgent(Agent[Percept, Action]):
    """Simple reflex agent"""

    def act(self, percept: Percept) -> Action:
        location, status = percept

        if status == "Dirty":
            return SUCK

        if location == "A":
            return RIGHT
        if location == "B":
            return LEFT

        return NOOP
```

Simple Reflex Agents (Two Room Vacuum World)

simulation.py

```python
from __future__ import annotations
from typing import Any, Dict, Generic, List, TypeVar

from agent import Agent
from environment import Environment

PerceptT = TypeVar("PerceptT")
ActionT = TypeVar("ActionT")

def run_episode(
    env: Environment[PerceptT, ActionT],
    agent: Agent[PerceptT, ActionT],
    max_steps: int = 10,
    stop_when_done: bool = True,
) -> List[Dict[str, Any]]:

    agent.reset()
    env.reset()

    logs: List[Dict[str, Any]] = []

    for t in range(1, max_steps + 1):
        percept = env.percept()
        action = agent.act(percept)
        env.step(action)

        logs.append({
            "step": t,
            "percept": percept,
            "action": action,
            "state": env.get_state()
        })

        if stop_when_done and env.is_done():
            break

    return logs
```

Simple Reflex Agents (Two Room Vacuum World)

simulation.py

```
def print_logs(title: str, logs: List[Dict[str, Any]]) -> None:
    print("\n" + "=" * 70)
    print(title)
    print("=" * 70)
    print(f"{'t':>2} | {'percept':>14} | {'action':>6} | state")
    print("-" * 70)

    for row in logs:
        print(f"{row['step']:>2} | {str(row['percept']):>14} |
{str(row['action']):>6} | {row['state']}")
```

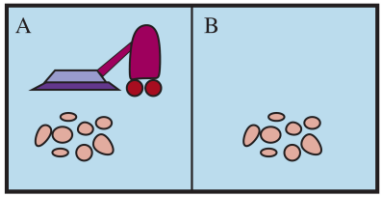
Simple Reflex Agents (Two Room Vacuum World)

demo1.py

```
from vacuum_world import make_random_vacuum_world, VacuumWorld2Rooms
from table_driven_vacuum_agent import TableDrivenVacuumAgent, build_small_demo_table
from simple_reflex_vacuum_agent import SimpleReflexVacuumAgent
from simulation import run_episode, print_logs


def main() -> None:
    env1 = VacuumWorld2Rooms({"A": "Dirty", "B": "Dirty"}, "A")
    env2 = VacuumWorld2Rooms({"A": "Dirty", "B": "Dirty"}, "A")

    table = build_small_demo_table(horizon=6)
    table_agent = TableDrivenVacuumAgent(table=table)

    logs_a = run_episode(env1, table_agent, max_steps=10)
    print_logs("PART A: Table-Driven Vacuum Agent", logs_a)

    reflex_agent = SimpleReflexVacuumAgent()

    logs_b = run_episode(env2, reflex_agent, max_steps=10)
    print_logs("PART B: Simple Reflex Vacuum Agent", logs_b)

if __name__ == "__main__":
    main()
```

Simple Reflex Agents (Two Room Vacuum World)

demo1.py

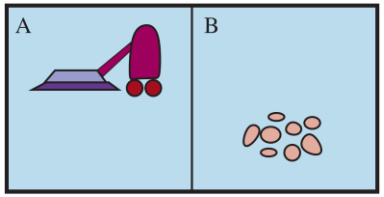



```
env = VacuumWorld2Rooms({"A": "Dirty", "B": "Dirty"}, "A")
reflex_agent = SimpleReflexVacuumAgent()

logs = run_episode(env, reflex_agent, max_steps=10)
print_logs("Simple Reflex Vacuum Agent", logs)
```

```
================================================================================
Simple Reflex Vacuum Agent
================================================================================
 t |        percept | action | state
--------------------------------------------------------------------------------
 1 | ('A', 'Dirty') |   Suck | {'A': 'Clean', 'B': 'Dirty', 'agent_location': 'A'}
 2 | ('A', 'Clean') |  Right | {'A': 'Clean', 'B': 'Dirty', 'agent_location': 'B'}
 3 | ('B', 'Dirty') |   Suck | {'A': 'Clean', 'B': 'Clean', 'agent_location': 'B'}
```

Simple Reflex Agents (Two Room Vacuum World)

demo1.py

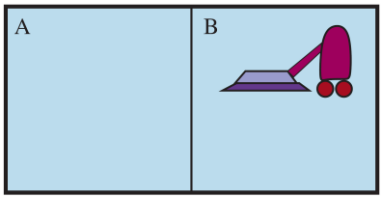



```
env = VacuumWorld2Rooms({"A": "Clean", "B": "Dirty"}, "A")
reflex_agent = SimpleReflexVacuumAgent()

logs = run_episode(env, reflex_agent, max_steps=10)
print_logs("Simple Reflex Vacuum Agent", logs)
```

```
================================================================================
Simple Reflex Vacuum Agent
================================================================================
 t |        percept | action | state
--------------------------------------------------------------------------------
 1 | ('A', 'Clean') |  Right | {'A': 'Clean', 'B': 'Dirty', 'agent_location': 'B'}
 2 | ('B', 'Dirty') |   Suck | {'A': 'Clean', 'B': 'Clean', 'agent_location': 'B'}
```

Simple Reflex Agents (Two Room Vacuum World)

demo1.py

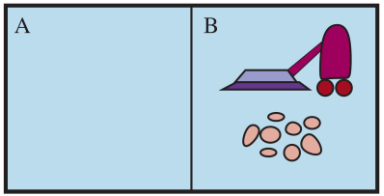



```
env = VacuumWorld2Rooms({"A": "Clean", "B": "Clean"}, "B")
reflex_agent = SimpleReflexVacuumAgent()

logs = run_episode(env, reflex_agent, max_steps=10)
print_logs("Simple Reflex Vacuum Agent", logs)
```

```
======================================================================
Simple Reflex Vacuum Agent
======================================================================
 t |        percept | action | state
----------------------------------------------------------------------
 1 | ('B', 'Clean') |   Left | {'A': 'Clean', 'B': 'Clean', 'agent_location': 'A'}
```

Simple Reflex Agents (Two Room Vacuum World)

demo1.py

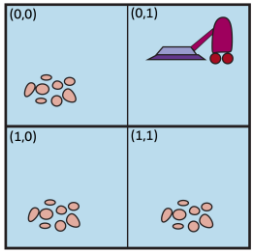



```
env = VacuumWorld2Rooms({"A": "Clean", "B": "Dirty"}, "B")
reflex_agent = SimpleReflexVacuumAgent()

logs = run_episode(env, reflex_agent, max_steps=10)
print_logs("Simple Reflex Vacuum Agent", logs)
```

```
======================================================================
Simple Reflex Vacuum Agent
======================================================================
 t |        percept | action | state
----------------------------------------------------------------------
 1 | ('B', 'Dirty') |   Suck | {'A': 'Clean', 'B': 'Clean', 'agent_location': 'B'}
```

demo2.py

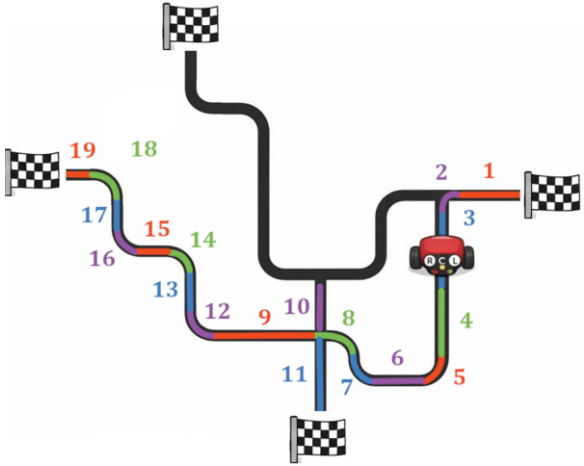
Simple Reflex Agents (2x2 Grid Vacuum World)




```
env = GridVacuumWorld2x2({(0, 0): "Dirty", (0, 1): "Clean", (1, 0): "Dirty", (1, 1): "Dirty"}, (0, 1))

    # Simple reflex
    reflex_agent = SimpleReflexGridVacuumAgent()
    logs = run_episode(env, reflex_agent, max_steps=20)
    print_logs("2x2 GRID: Simple Reflex Agent", logs)

    ======================================================================
    2x2 GRID: Simple Reflex Agent
    ======================================================================
     t |        percept | action | state
    ----------------------------------------------------------------------
     1 | ((0, 1), 'Clean') |   Down | {'(0,0)': 'Dirty', '(0,1)': 'Clean', '(1,0)': 'Dirty', '(1,1)': 'Dirty', 'agent_pos': (1, 1)}
     2 | ((1, 1), 'Dirty') |   Suck | {'(0,0)': 'Dirty', '(0,1)': 'Clean', '(1,0)': 'Dirty', '(1,1)': 'Clean', 'agent_pos': (1, 1)}
     3 | ((1, 1), 'Clean') |   Left | {'(0,0)': 'Dirty', '(0,1)': 'Clean', '(1,0)': 'Dirty', '(1,1)': 'Clean', 'agent_pos': (1, 0)}
     4 | ((1, 0), 'Dirty') |   Suck | {'(0,0)': 'Dirty', '(0,1)': 'Clean', '(1,0)': 'Clean', '(1,1)': 'Clean', 'agent_pos': (1, 0)}
     5 | ((1, 0), 'Clean') |     Up | {'(0,0)': 'Dirty', '(0,1)': 'Clean', '(1,0)': 'Clean', '(1,1)': 'Clean', 'agent_pos': (0, 0)}
     6 | ((0, 0), 'Dirty') |   Suck | {'(0,0)': 'Clean', '(0,1)': 'Clean', '(1,0)': 'Clean', '(1,1)': 'Clean', 'agent_pos': (0, 0)}
```

Simple Reflex Agents (Line Following Robot)

- A line following robot world with **dark lines** on a light floor representing road to follow.
- Robot has three reflective sensors mounted across its front **(L, C, R)** that report whether each sensor is over **Black** or **White**.
- Robot reads sensors ▶ chooses one action ▶ moves a small amount.



Simple Reflex Agents (Line Following Robot)

Environment: 2D Floor with Black Lines	**Sensors:** 3 reflective sensors (L,C,R)
Agent: Line Following Robot	**Actuators:** Tyres, Motors

Percepts (L,C,R) each ∈ {B,W}:

(W, W, W)
(B, W, W)
(W, B, W)
(B, B, W)
(W, W, B)
(B, W, B)
(W, B, B)
(B, B, B)

Actions:

FORWARD
TURN_LEFT
TURN_RIGHT
STOP

Condition-Action Rules:

```
IF C=B THEN FORWARD
ELSE IF L=B THEN TURN_LEFT
ELSE IF R=B THEN TURN_RIGHT

ELSE L=W AND C=W AND R=W THEN
STOP
```

FORWARD

TURN LEFT

TURN RIGHT

Simple Reflex Agents (Line Following Robot)

- Here the status is represented as robot's relation to the line + progress.

Status: (segment, offset, done)

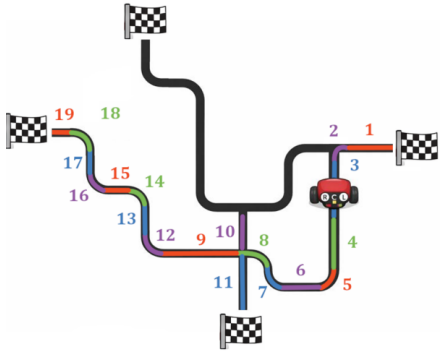
Segment = 1,2,3,...

offset = CENTERED, LINE_LEFT_OF_ROBOT, LINE_RIGHT_OF_ROBOT, CROSSING_OR_MARKER, LOST

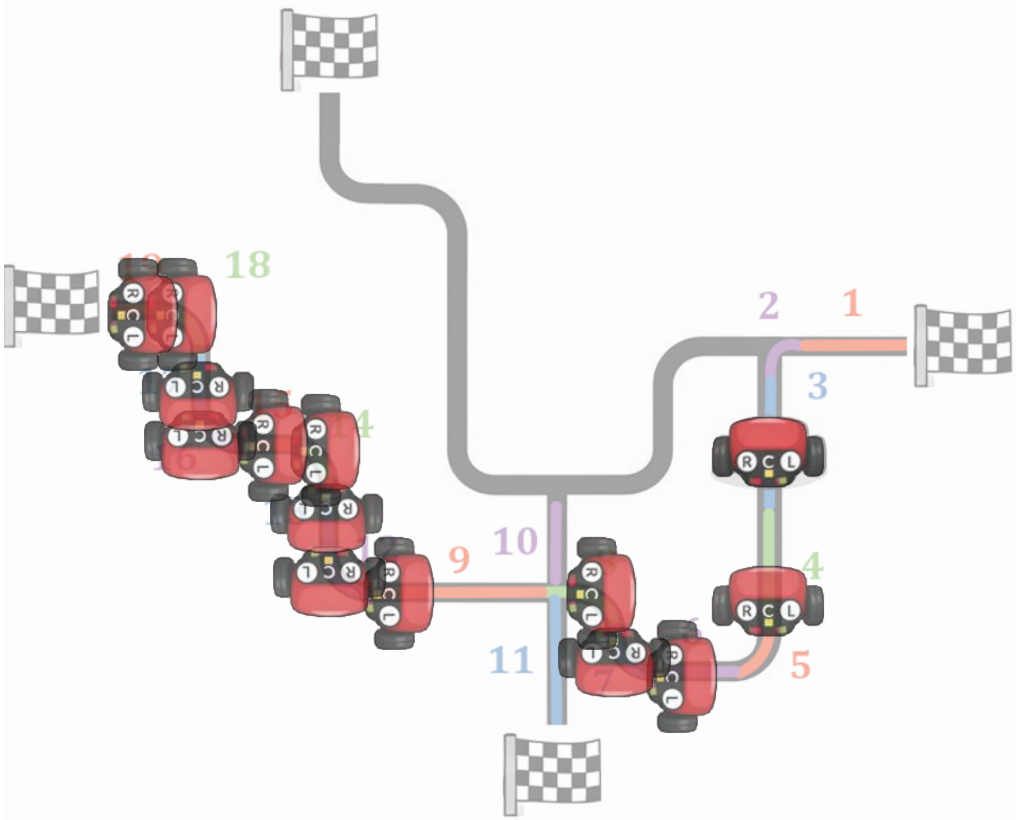
done = True or False (Terminal flag)

Simple Reflex Agents (Line Following Robot)

Step	Status (before)	Percept (L,C,R)	Action	Status (after)
1	(3, CENTERED, False)	(W,B,W)	FORWARD	(4, LINE_RIGHT_OF_ROBOT, False)
2	(4, LINE_RIGHT_OF_ROBOT, False)	(W,W,B)	TURN_RIGHT	(5, CENTERED, False)
3	(5, CENTERED, False)	(W,B,W)	FORWARD	(6, LINE_RIGHT_OF_ROBOT, False)
4	(6, LINE_RIGHT_OF_ROBOT, False)	(W,W,B)	TURN_RIGHT	(7, LINE_LEFT_OF_ROBOT, False)
5	(7, LINE_LEFT_OF_ROBOT, False)	(B,W,W)	TURN_LEFT	(8, CENTERED, False)
6	(8, CENTERED, False)	(B,B,B)	FORWARD	(9, LINE_RIGHT_OF_ROBOT, False)
7	(9, LINE_RIGHT_OF_ROBOT, False)	(W,W,B)	TURN_RIGHT	(12, CENTERED, False)
8	(12, CENTERED, False)	(W,B,W)	FORWARD	(13, LINE_LEFT_OF_ROBOT, False)
9	(13, LINE_LEFT_OF_ROBOT, False)	(B,W,W)	TURN_LEFT	(14, CENTERED, False)
10	(14, CENTERED, False)	(B,B,B)	FORWARD	(15, LINE_RIGHT_OF_ROBOT, False)
11	(15, LINE_RIGHT_OF_ROBOT, False)	(W,W,B)	TURN_RIGHT	(16, CENTERED, False)
12	(16, CENTERED, False)	(W,B,W)	FORWARD	(17, LINE_LEFT_OF_ROBOT, False)
13	(17, LINE_LEFT_OF_ROBOT, False)	(B,W,W)	TURN_LEFT	(18, CENTERED, False)
14	(18, CENTERED, False)	(W,B,W)	STOP	(19, LOST, True)
15	(19, LOST, True)	(W,W,W)	**STOP**	(19, **LOST**, **True**)



Simple Reflex Agents (Line Following Robot)



Model-Based Reflex Agents

- Model-based reflex agents use both their **current perception** and **memory** to maintain an **internal model** of the world.
- As the agent continues to receive new information, the model is updated.
- The agent's actions depend on its model, reflexes, previous precepts and current state.
- These agents, unlike simple reflex agents, can store information in memory and can operate in environments that are partially observable and changing.
- However, they are still limited by their set of rules.

Model-Based Reflex Agents

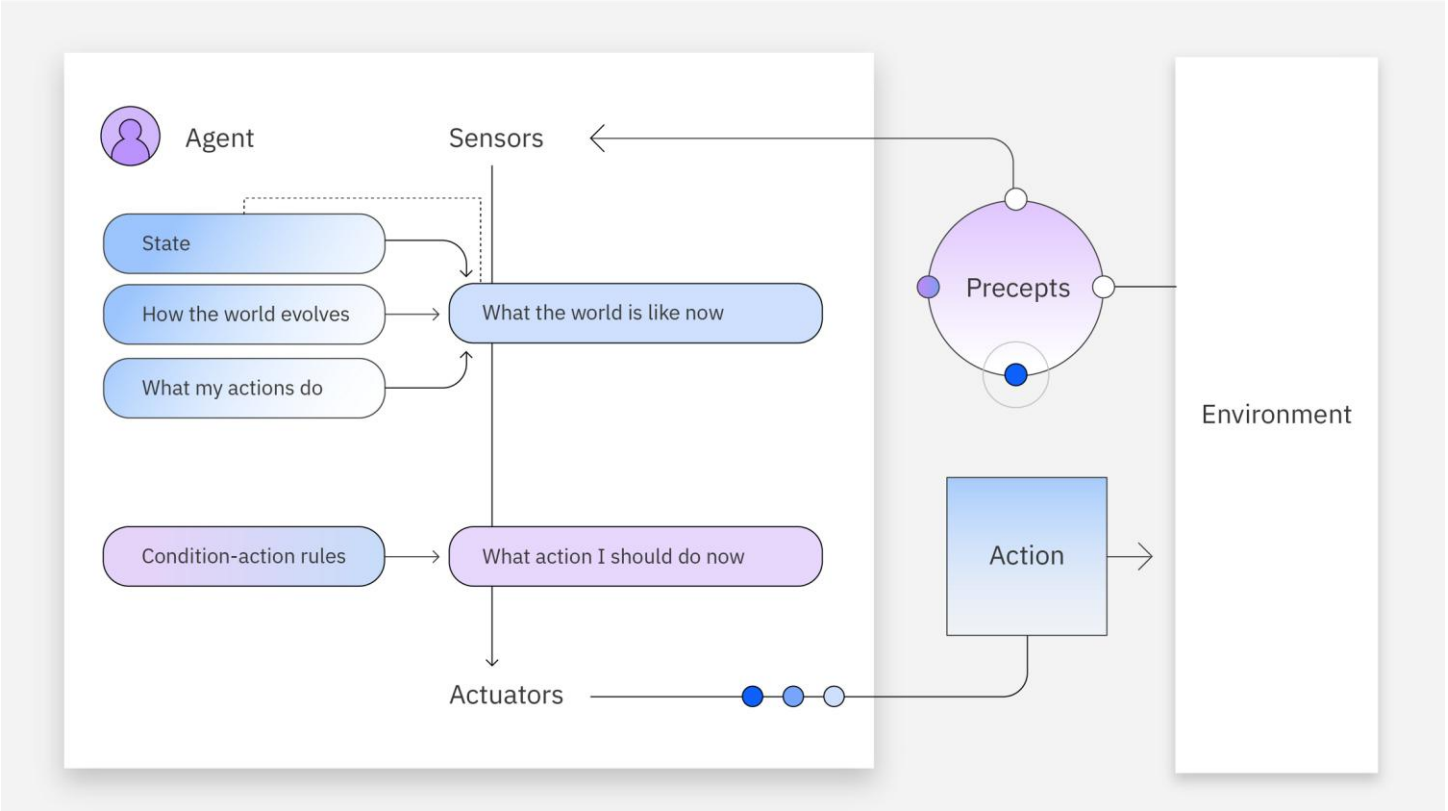
Key Features:

- Track the world's state over time
- Infer unobserved aspects of current states
- Function effectively in partially observable environments
- Still primarily reactive, but with contextual awareness

Example: A robot vacuum cleaner. As it cleans a dirty room, it senses obstacles such as furniture and adjusts around them. The robot also stores a model of the areas that it has already cleaned to not get stuck in a loop of repeated cleaning.

When to Use: They are beneficial in situations where the environment is dynamic and not all elements can be directly observed at once.

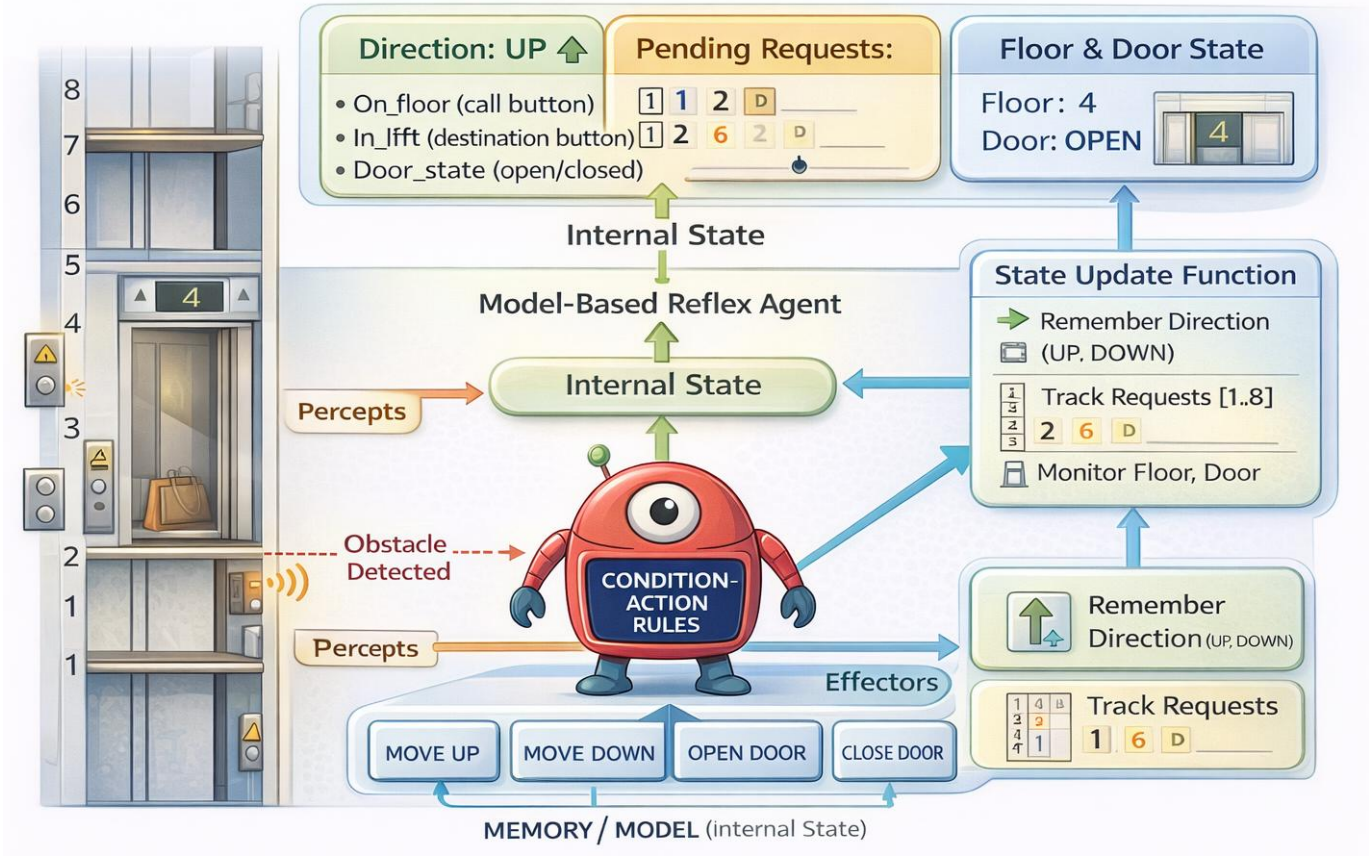
Model-Based Reflex Agents



Model-Based Reflex Agents (Vacuum Cleaner Agent)

- A model-based vacuum agent extends a simple reflex agent by maintaining an internal representation of the environment, allowing it to make decisions based on both current percepts and past information.

Model-Based Reflex Agents (8 Floors Elevator Controller)



Model-Based Reflex Agents (8 Floors Elevator Controller)

- A lift controller with single lift installed in 8-story building.

Environment: Lift and 8 Floor Building	**Sensors:** Electronic Switches
Agent: Lift Controller	**Actuators:** Motors, Pully

Percepts:

```
current_floor ∈ {1..8}
door_state ∈ {OPEN,CLOSED}
obstacle ∈ {True,False}
```

Model

```
inside_req[1..8] ∈ {True,False}
up_req[1..7] ∈ {True,False}
down_req[2..8] ∈ {True,False}
direction ∈ {UP, DOWN, IDLE}
has_above ∈ {True,False}
has_below ∈ {True,False}
stop_here ∈ {True,False}
```

Actions:

```
OPEN_DOOR
CLOSE_DOOR
MOVE_UP
MOVE_DOWN
STOP
```

Model-Based Reflex Agents (8 Floors Elevator Controller)

Condition-Action Rules:

```
IF door_state=OPEN AND obstacle=TRUE     THEN OPEN_DOOR
IF door_state=CLOSED AND stop_here=TRUE THEN OPEN_DOOR
IF door_state=OPEN AND obstacle=FALSE    THEN CLOSE_DOOR

IF door_state=CLOSED AND direction=UP AND has_above=TRUE    THEN MOVE_UP
IF door_state=CLOSED AND direction=DOWN AND has_below=TRUE THEN MOVE_DOWN

IF door_state=CLOSED AND direction=UP AND has_above=FALSE THEN direction=DOWN,
   MOVE_DOWN
IF door_state=CLOSED AND direction=DOWN AND has_below=FALSE THEN direction=UP,
   MOVE_UP

IF direction=IDLE AND has_above=TRUE THEN direction=UP, MOVE_UP
IF direction=IDLE AND has_below=TRUE THEN direction=DOWN, MOVE_DOWN

IF door_state=CLOSED AND has_above=FALSE AND has_below=FALSE THEN
   direction=IDLE, STOP
```

Model-Based Reflex Agents (Smart Room Light Controller)

- A room with a motion sensor and a light.

Environment: Room	**Sensors:** Motion Sensors
Agent: Light Controller	**Actuators:** Light

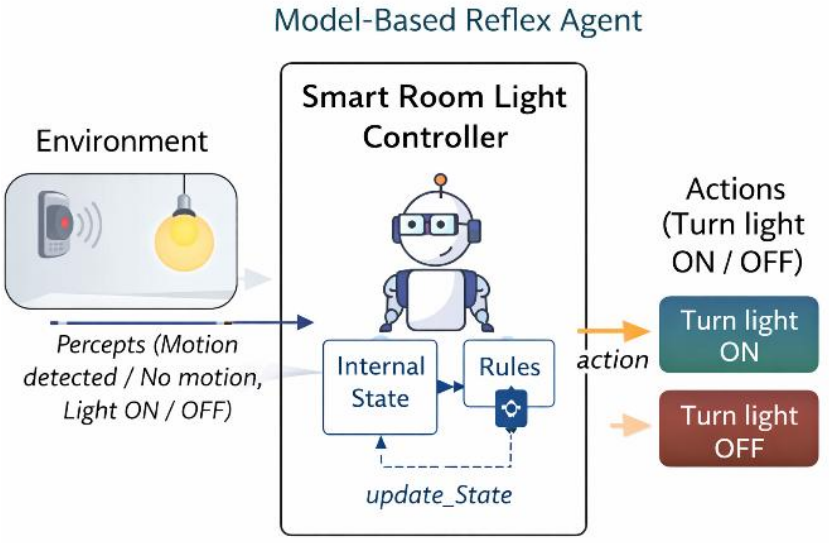
Percepts:

```
motion ∈ {True,False}
light_status ∈ {ON,OFF}
```

Actions:

```
TURN_LIGHT_ON
TURN_LIGHT_OFF
```

Model
`last_motion_status ∈ {True,False}`
`time_since_motion ∈ ℝ`



Goal-Based Agents

- Goal-based agents **plan** their actions with a specific **objective/goal** in mind.
- Unlike reflex agents that respond to immediate stimuli, goal-based agents **evaluate** how different action sequences might lead toward their defined goal, selecting the path that appears most promising.
- Goal-based agents have an **internal model** of the world and also a **goal** or **set of goals**.
- These agents **search** for **action sequences** that reach their goal and **plan** these actions **before acting** on them.

Goal-Based Agents

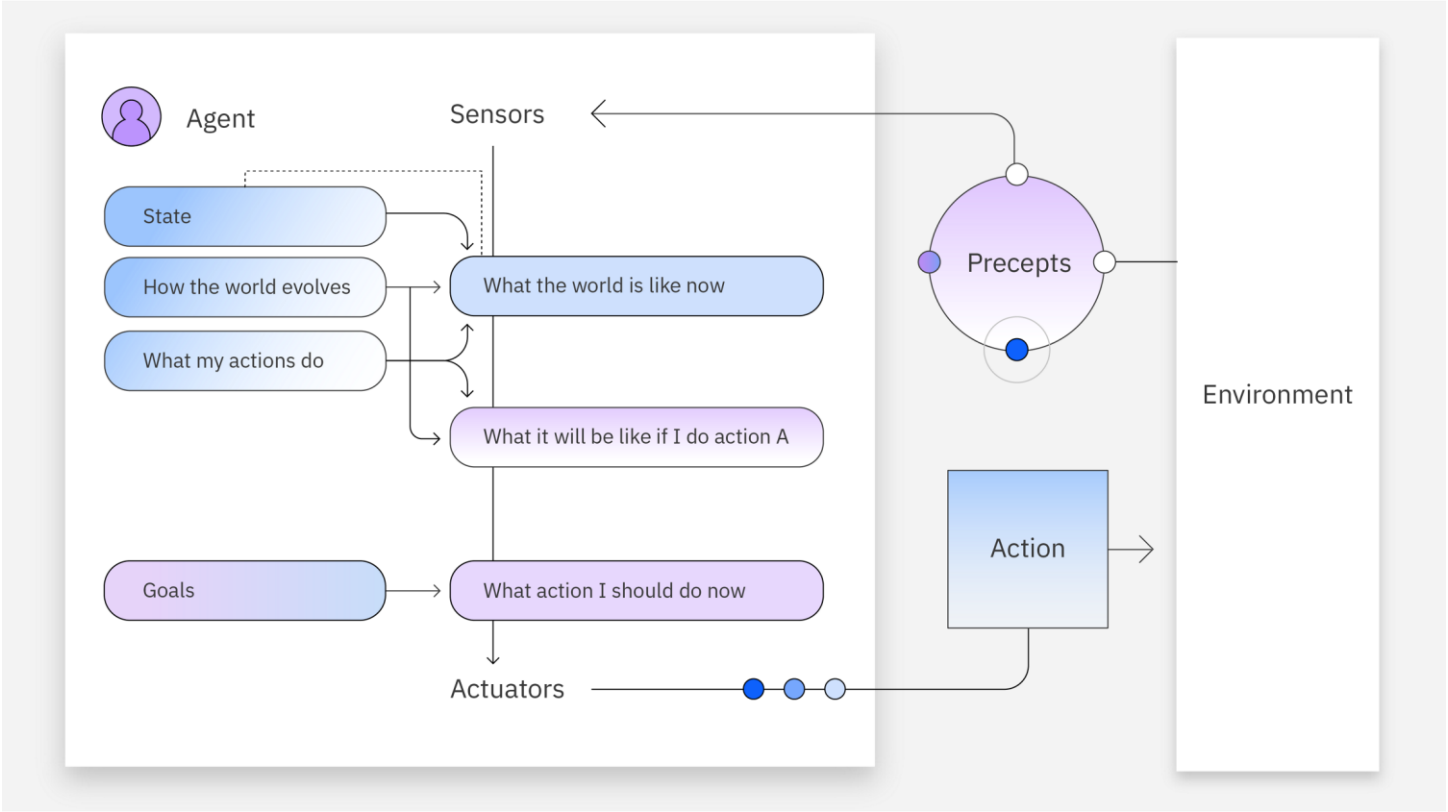
Key Features:

- Employ search and planning mechanisms
- Evaluate actions based on their contribution toward goal achievement
- Consider future states and outcomes
- May explore multiple possible routes to a goal

Example: An autonomous vehicle that recommends the fastest route to your destination. The model considers various routes that reach your destination, or in other words, your goal. In this example, the agent's condition-action rule states that if a quicker route is found, the agent recommends that one instead.

When to Use: They are important in applications that require strategic decision-making and planning.

Goal-Based Agents



Goal-Based Agents (Grid Navigation Robot)

- A robot moves in a grid environment; some cells are blocked by obstacles.

Environment: Navigation Grid	**Sensors:** Sonar/Obstacle Sensors
Agent: Navigation Robot	**Actuators:** Motors, Tyres

Percepts:

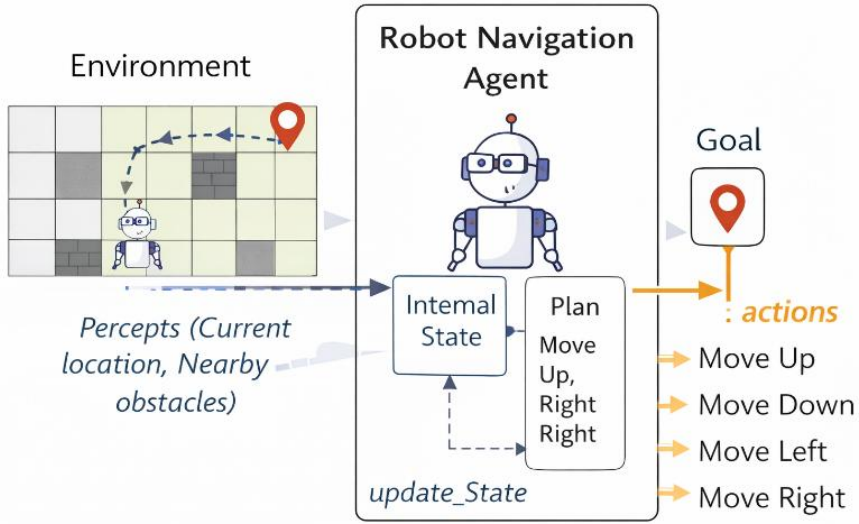
```
loc ∈ (row,col)
obstacle ∈{True,False}
```

Actions:

```
MOVE_UP
MOVE_DOWN
MOVE_LEFT
MOVE_RIGHT
```

Goal:

```
Reach destination cell
```



Utility-Based Agents

- Utility-based agents extend goal-based thinking by evaluating actions based on **how well** they **maximize a utility function**, essentially a measure of "**happiness**" or "**satisfaction**."
- Utility-based agents select the sequence of actions that reach the goal and also **maximize** utility or reward.
- Utility is calculated through a **utility function**.
- This approach allows them to make nuanced trade-offs between competing goals or uncertain outcomes.

Utility-Based Agents

- This function assigns a utility value, a metric measuring the usefulness of an action or how “happy” makes the agent, to each scenario based on a set of fixed criteria.

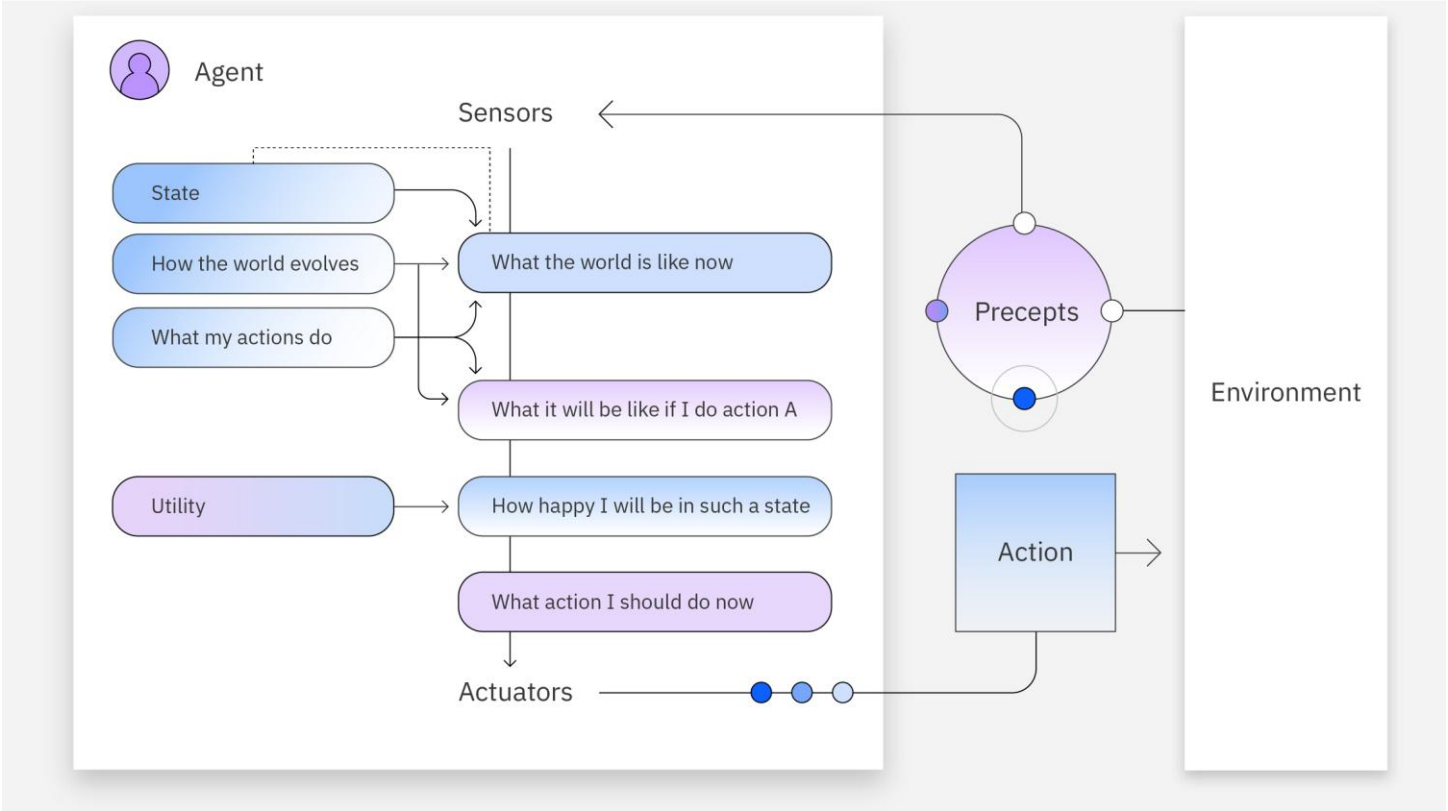
Key Features:

- Balance multiple, sometimes conflicting objectives
- Handle probabilistic and uncertain environments
- Evaluate actions based on expected utility
- Make rational decisions under constraints

Example: An autonomous vehicle that recommends the route to your destination that optimizes fuel efficiency and minimizes the time spent in traffic and the cost of tolls. This agent measures utility through this set of criteria to select the most favorable route.

When to Use: They are ideal for tasks where multiple criteria need to be evaluated simultaneously.

Utility-Based Agents



Utility-Based Agents (Grid Navigation Robot)

- An autonomous taxi operates in a city facilitating passengers.

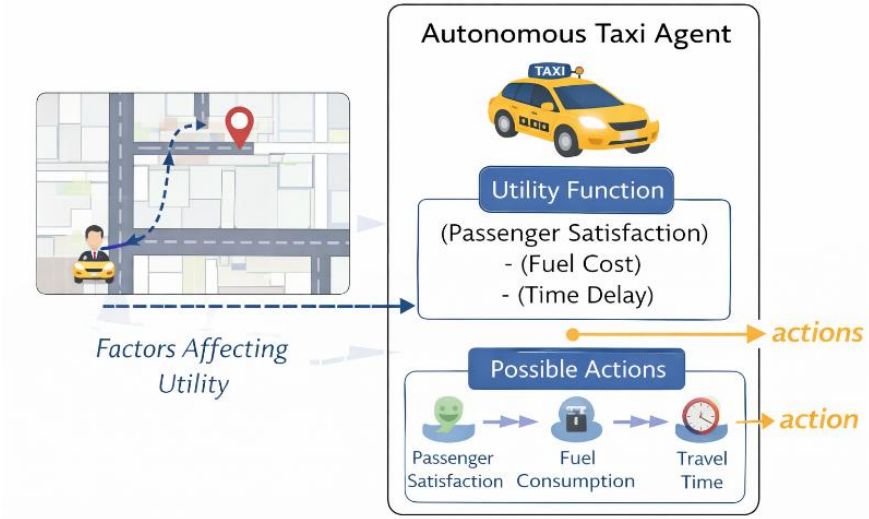
Environment: Road	**Sensors:** Cameras, Lidar, Obstacle Sensors
Agent: Autonomous Driver	**Actuators:** Motors, Tyres

Percepts:
current_loc
obstacles
pedestrians
traffic_lights
traffic_signs
other_traffic

Goal:
Reach destination

Actions:
STEER_LEFT
STEER_RIGHT
BRAKE
ACCELERATE

Utility:
Ride Satisfaction
Fuel Cost
Ride Duration



Summary of Agents till now

Agent	Present	Past	Future	Decision	Learning
Table-Driven	✓	✗	✗	Look-Up Table	✗
Simple Reflex	✓	✗	✗	Condition-Action Rules	✗
Model-Based	✓	✓	✗	Condition-Action Rules	✗
Goal-Based	✓	✓	✓	Goal or Set of Goals	✗
Utility-Based	✓	✓	✓	Utility or Set of Utilities	✗

Learning Agents

- Learning agents **improve** their **performance over time** based on experience.
- They **modify** their **behavior** by observing the consequences of their actions, adjusting their internal models and decision-making approaches to achieve **better outcomes** in **future interactions**.
- Learning agents hold the same capabilities as the other agent types but are unique in their ability to learn.
- New experiences are added to their initial knowledge base, which occurs autonomously.

Learning Agents

- This learning enhances the agent's ability to operate in **unfamiliar environments**.
- Learning agents might be utility or goal-based in their reasoning and are composed of four main elements:
  - **Learning:** This process improves the agent's knowledge by learning from the environment through its precepts and sensors.
  - **Critic:** This component provides feedback to the agent on whether the quality of its responses meets the performance standard.
  - **Performance:** This element is responsible for selecting actions upon learning.
  - **Problem generator:** This module creates various proposals for actions to be taken.

Learning Agents

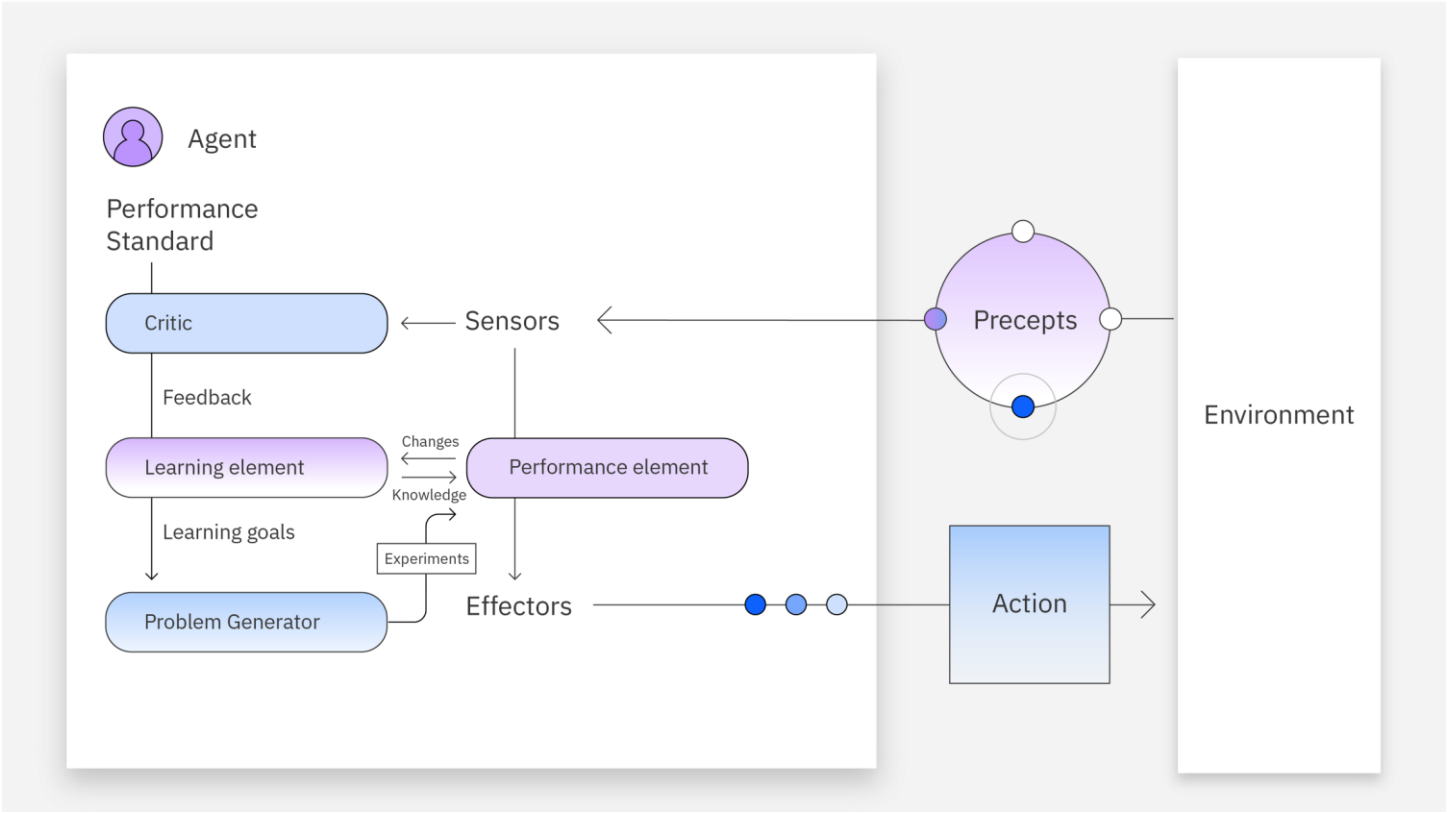
Key Features:

- Adapt to changing environments
- Improve performance with experience
- Contain both a performance element and a learning element
- Generate new knowledge rather than simply applying existing rules

Example: Personalized recommendations on e-commerce sites. These agents track user activity and preferences in their memory. This information is used to recommend certain products and services to the user. The cycle repeats each time new recommendations are made. The user's activity is continuously stored for learning purposes. In doing so, the agent improves its accuracy over time.

When to Use: They are well-suited for dynamic environments that change over time.

Learning Agents



Multi-Agent Systems (MAS)

- Multi-agent systems operate in **environments shared** with other agents, either cooperating or competing to achieve individual or group goals.
- These systems are decentralized, often requiring communication, negotiation or coordination protocols.
- They are well-suited to **distributed problem solving** but can be complex to design due to emergent and unpredictable behaviors.
- Types of multi-agent systems:
  - **Cooperative MAS:** Agents work together toward shared objectives.
  - **Competitive MAS:** Agents pursue individual goals that may conflict.
  - **Mixed MAS:** Agents cooperate in some scenarios and compete in others.

Multi-Agent Systems (MAS)

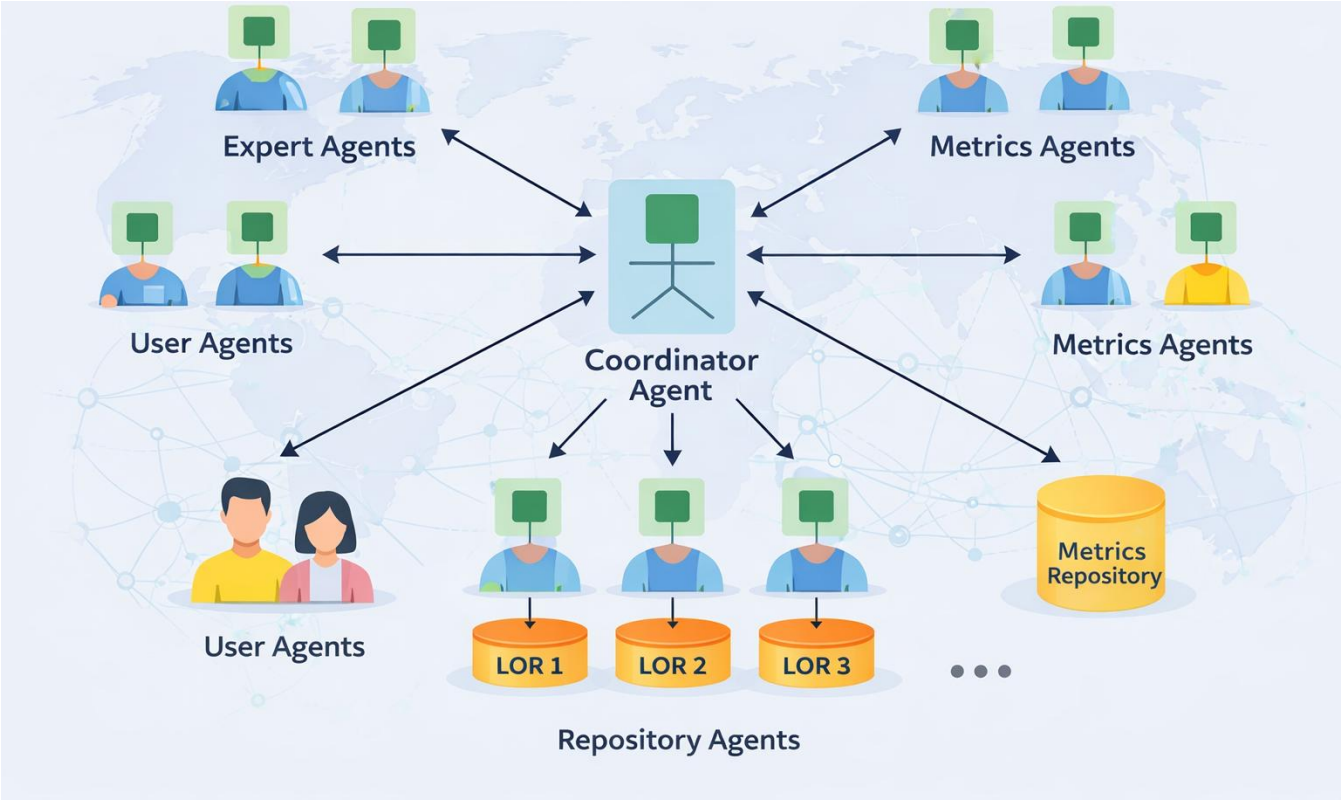
Key Features:

- Each agent acts on its own goals and knowledge.
- Agents communicate, cooperate or compete to achieve individual or shared objectives.
- Agents work together to solve complex problems more efficiently than they could alone.
- No central control, agents make decisions independently.

Example: A warehouse robot might use: (1) Model-based reflexes for navigation (2) Goal-based planning for task sequencing (3) Utility-based decision-making for prioritizing tasks (4) Learning capabilities for route optimization.

When to Use: They are ideal for decentralized environments, where agents need to collaborate or make decisions independently.

Multi-Agent Systems (MAS)



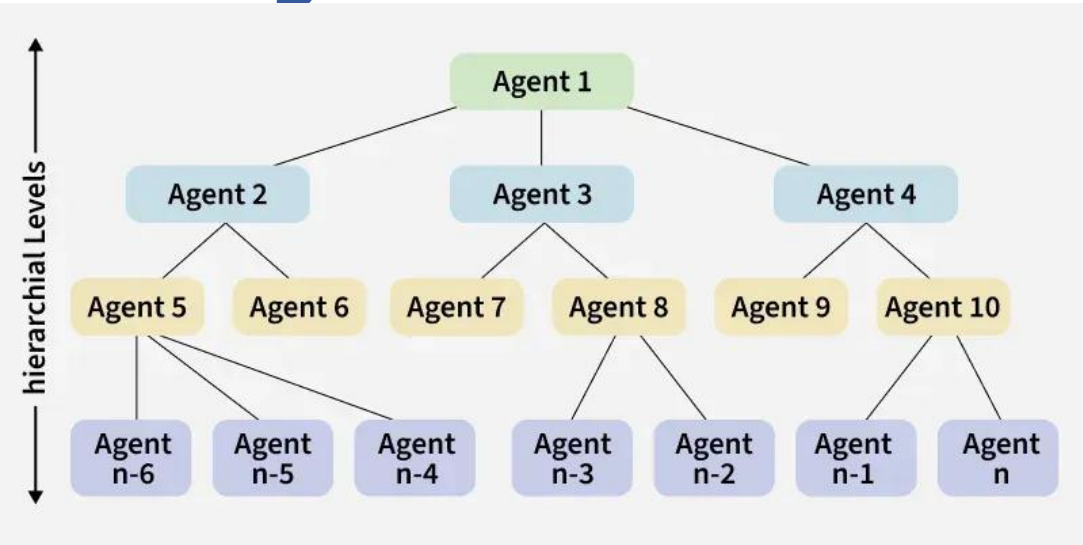
Hierarchical Agents

- Hierarchical agents organize behavior into **multiple layers** such as strategic, tactical and operational.
- Higher levels make abstract decisions that break down into more **specific subgoals** for lower levels to execute.
- This structure improves **scalability**, **reusability** of skills and management of complex tasks, but requires designing effective interfaces between layers.

Key Features:

- Decision-making is divided into different levels for more efficient task handling.
- Complex tasks are broken down into simpler subtasks.
- Higher levels direct lower levels for coordinated action.

Hierarchical Agents



Example: Drone delivery systems in which fleet management is done at top level and individual navigation at lower level.

When to Use: They are useful in scenarios where tasks can be broken into distinct stages such as robotics or industrial automation.

Comparison of AI Agent Types

Agent Type	Main Strength	Limitations	Best For	Example
Simple Reflex Agent	Instant reaction based on fixed rules	No memory or learning; fails in dynamic environments	Fully observable, stable and simple environments	Traffic light timers
Model-Based Reflex Agent	Handles partial observability with internal state	More computational demand; depends on model accuracy	Dynamic or partially observable environments	Robot vacuum cleaners
Goal-Based Agent	Plans ahead to achieve specific objectives	Needs clear goals and planning algorithms	Strategic tasks with defined goals	Logistics route planning
Utility-Based Agent	Balances multiple factors for best outcome	Requires complex utility functions	Multi-criteria decision-making	Financial portfolio management
Learning Agent	Improves over time via experience	Needs data and training time	Dynamic environments with changing conditions	AI chatbots
Multi-Agent System (MAS)	Distributed problem-solving with cooperation or competition	Complex interactions; unpredictable behaviors	Decentralized, multi-entity systems	Smart traffic control
Hierarchical Agent	Breaks complex tasks into levels for efficiency	Requires well-defined interfaces between layers	Large-scale, multi-level operations	Drone delivery management

When to use each AI Agent Type

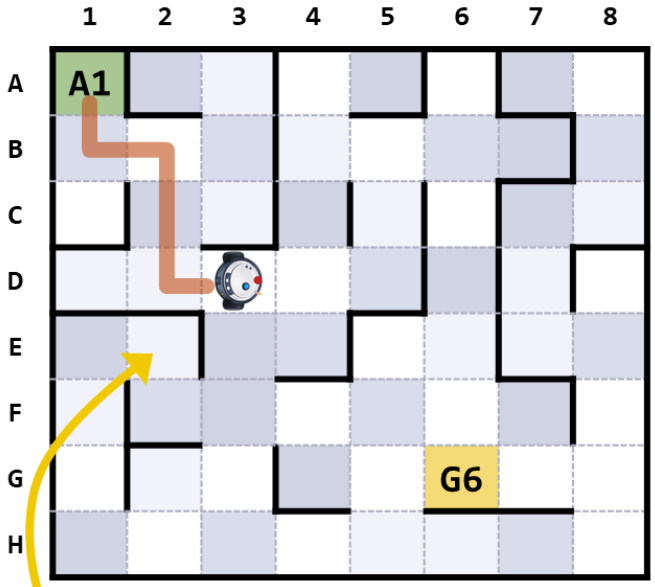
1. **Simple Reflex Agent**
   - Environment is fully observable and predictable
   - Tasks are repetitive with fixed rules
2. **Model-Based Reflex Agent**
   - Some information about the environment is hidden but can be modeled
   - Environment changes but follows predictable patterns
3. **Goal-Based Agent**
   - Tasks require planning multiple steps ahead
   - Clear goals are defined and can be measured
4. **Utility-Based Agent**
   - Need to balance trade-offs like cost, time and risk
   - Multiple objectives must be prioritized

When to use each AI Agent Type

5. **Learning Agent**
   - Environment changes over time and the system must adapt
   - Performance should improve with experience
6. **Multi-Agent System (MAS)**
   - Multiple agents must work together or compete
   - Problem-solving is decentralized and distributed
7. **Hierarchical Agent**
   - Tasks can be split into strategic, tactical and operational levels
   - Large-scale operations require coordination between layers

Maze Grid Navigation Agent

Maze Grid Navigation Agent



Agent Actions

FORWARD
LEFT TURN
RIGHT TURN
UTURN

Percepts

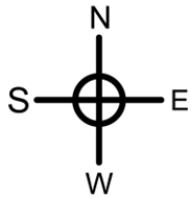
front_wall
left_wall
right_wall

Start Cell

A1

Goal Cell

G6




```
wall["E2"] = {True,
              True,
              False,
              False}
```

Percept
(Left,Center,Right)

Percept = (F,T,T)

Loc = E4

Maze Grid Navigation Agent

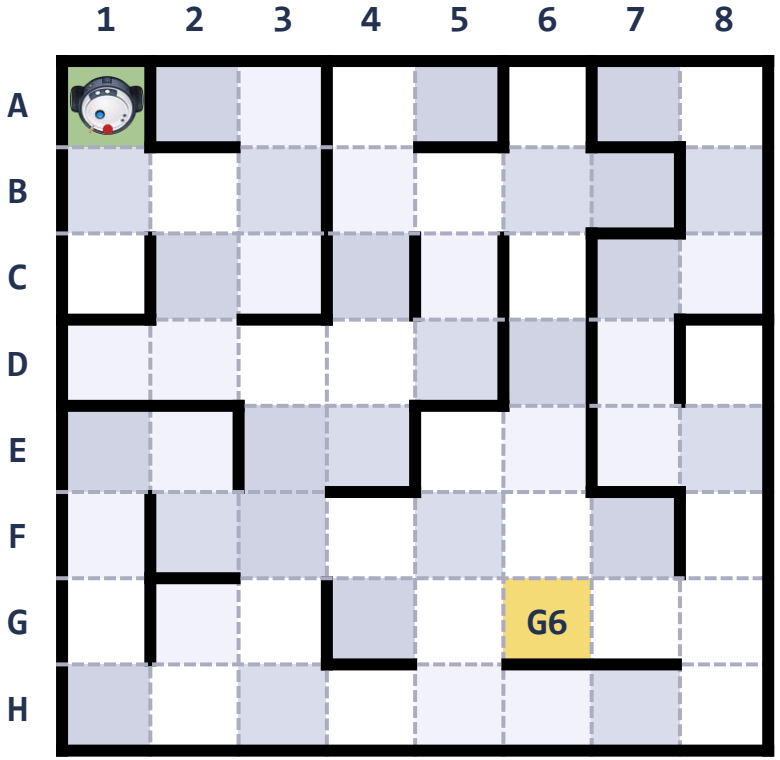
Simple Reflex Agent

Simple Reflex Agent (Example 1)

Start Cell = A1

Goal Cell = G6

Percepts:
{L, C, R}



Actions:
Forward
LeftTurn
RightTurn
UTurn

Simple Reflex Agent (Example 1)

Condition-Action Rules:

```
IF C=False THEN Forward

ELSE IF L=False THEN LeftTurn

ELSE IF R=False THEN RightTurn

ELSE UTurn
```

Simple Reflex Agent (Example 1)

Step	Percept	Action	Loc (Before)	Loc (After)
Start			A1	
1	(T,F,T)	Forward	A1	B1
2	(F,F,T)	Forward	B1	C1
3	(T,T,T)	UTurn	C1	C1
4	(T,F,T)	Forward	C1	B1
5	(T,F,F)	Forward	B1	A1
6	(T,T,T)	UTurn	A1	A1
7	(T,F,T)	Forward	A1	B1
8	(F,F,T)	Forward	B1	C1
9	(T,T,T)	UTurn	C1	C1

A1→B1→C1→B1→A1→B1→C1 ...

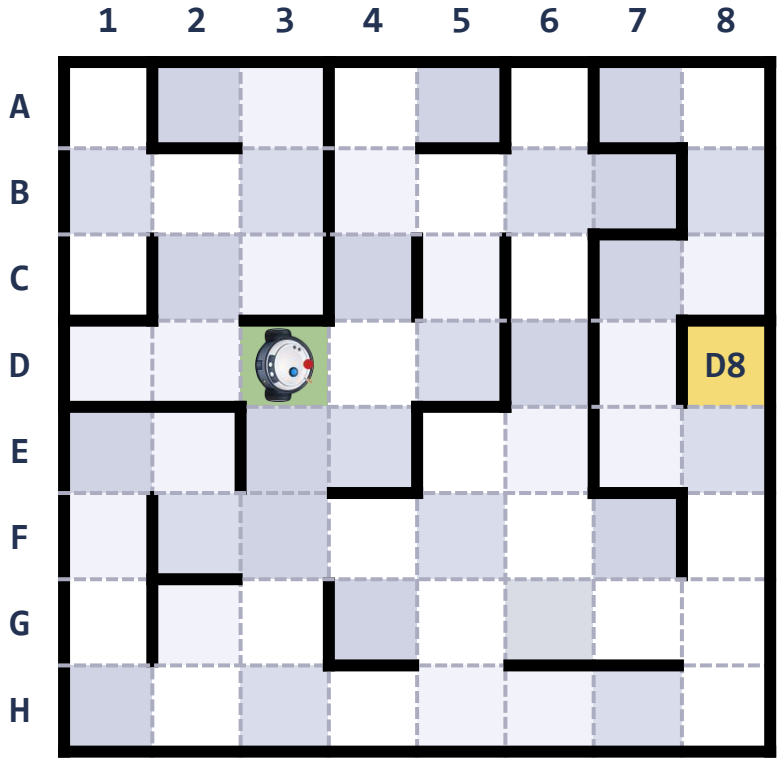
Stuck in Loop

Simple Reflex Agent (Example 2)

Start Cell = D3

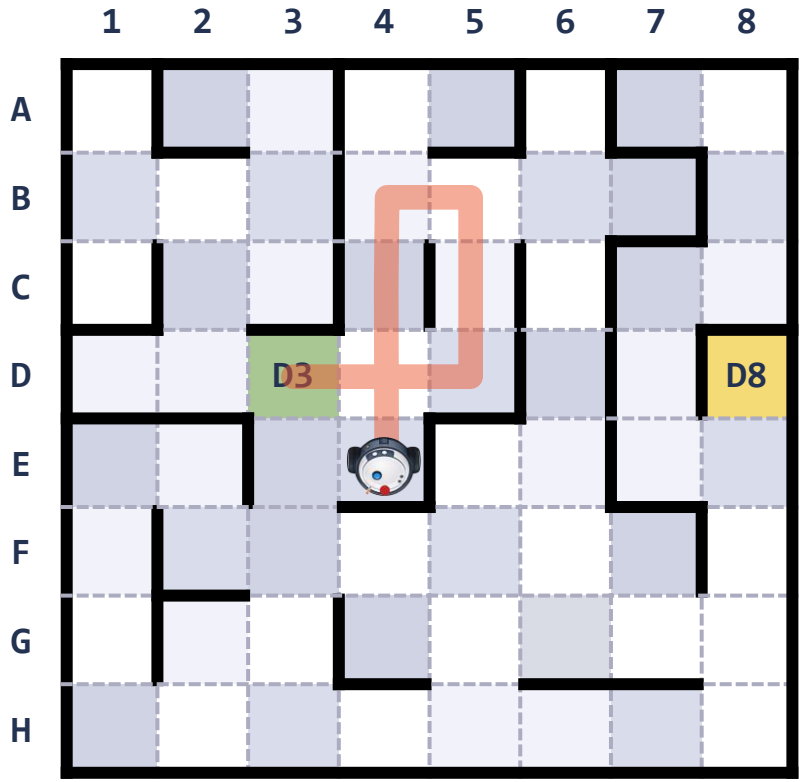
Goal Cell = D8

Percepts:
{L, C, R}



Actions:
Forward
LeftTurn
RightTurn
UTurn

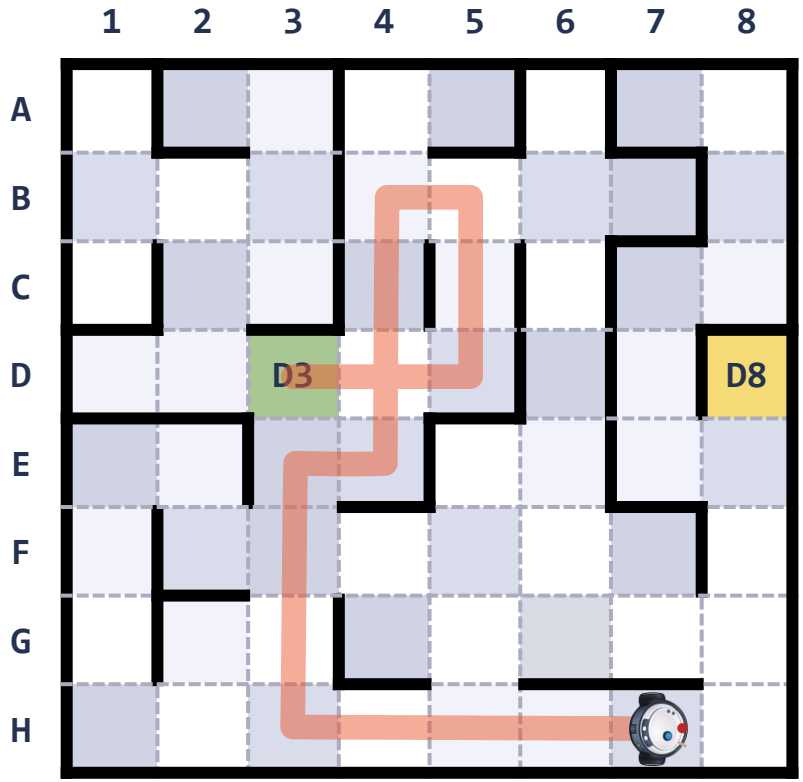
Simple Reflex Agent (Example 2)



Step	Percept	Action	Loc (Before)	Loc (After)
Start			D3	
1	(T,F,F)	Forward	D3	D4
2	(F,F,F)	Forward	D4	D5
3	(F,T,T)	LeftTurn	D5	D5
4	(F,F,T)	Forward	D5	C5
5	(T,F,T)	Forward	C5	B5
6	(F,T,F)	LeftTurn	B5	B5
7	(F,F,T)	Forward	B5	B4
8	(F,T,F)	LeftTurn	B4	B4
9	(F,F,T)	Forward	B4	C4
10	(T,F,T)	Forward	C4	D4
11	(F,F,F)	Forward	D4	E4

D3→D4→D5→C5→B5→B4→C4→D4→E4→

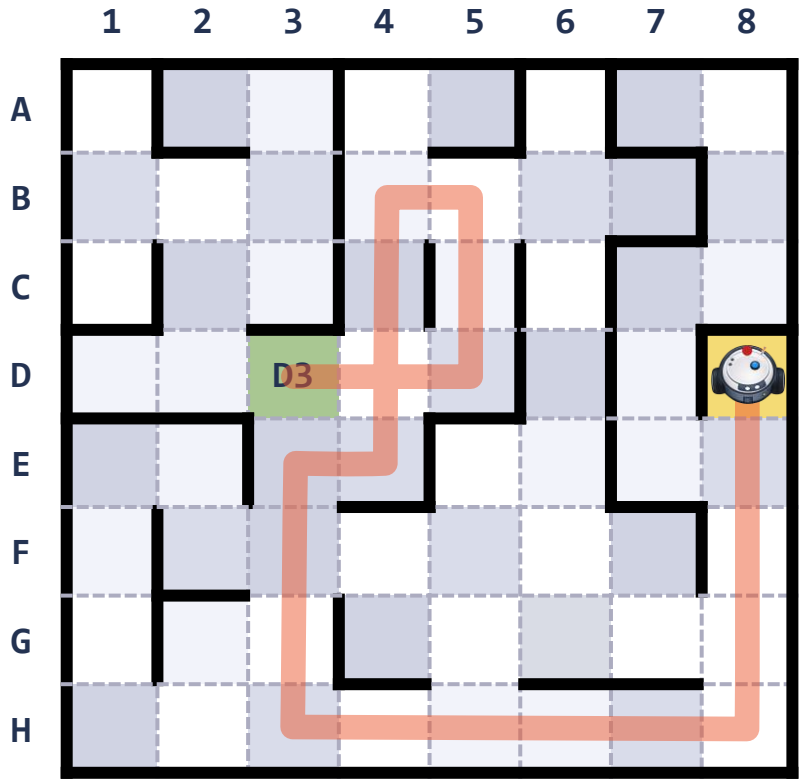
Simple Reflex Agent (Example 2)



Step	Percept	Action	Loc (Before)	Loc (After)
12	(T,T,F)	RightTurn	E4	E4
13	(T,F,F)	Forward	E4	E3
14	(F,T,F)	LeftTurn	E3	E3
15	(F,F,T)	Forward	E3	F3
16	(F,F,F)	Forward	F3	G3
17	(T,F,F)	Forward	G3	H3
18	(F,T,F)	LeftTurn	H3	H3
19	(F,F,T)	Forward	H3	H4
20	(T,F,T)	Forward	H4	H5
21	(F,F,T)	Forward	H5	H6
22	(T,F,T)	Forward	H6	H7

D3→D4→D5→C5→B5→B4→C4→D4→E4→
E3→F3→G3→H3→H4→H5→H6→H7→

Simple Reflex Agent (Example 2)



Step	Percept	Action	Loc (Before)	Loc (After)
23	(T,F,T)	Forward	H7	H8
24	(F,T,T)	LeftTurn	H8	H8
25	(F,F,T)	Forward	H8	G8
26	(F,F,T)	Forward	G8	F8
27	(T,F,T)	Forward	F8	E8
28	(F,F,T)	Forward	E8	D8
Loc = Goal, hence, STOP				

D3→D4→D5→C5→B5→B4→C4→D4→E4→
E3→F3→G3→H3→H4→H5→H6→H7→H8→
G8→F8→E8→D8

Maze Grid Navigation Agent

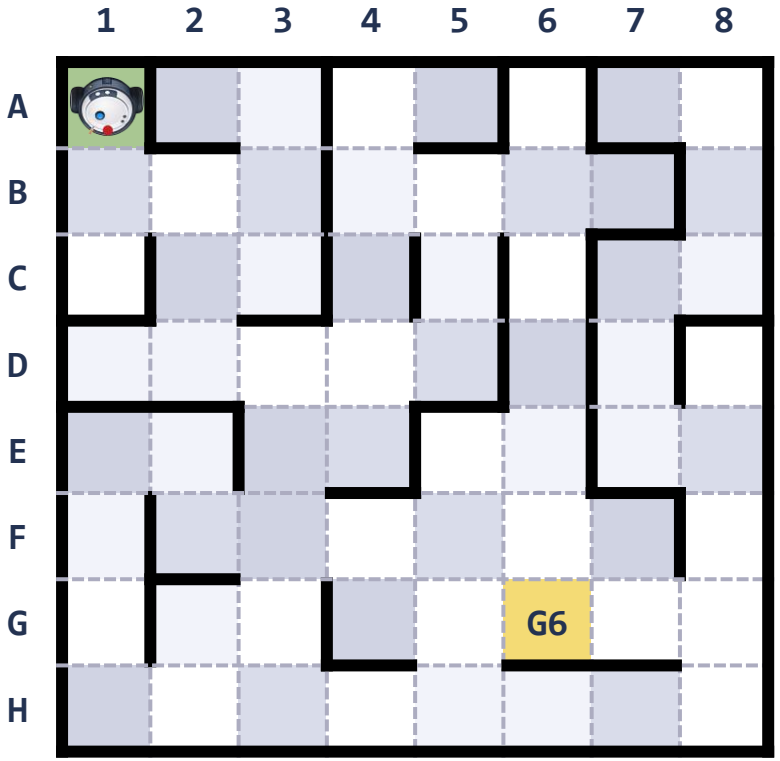
Model-Based Reflex Agent

Model-Based Reflex Agent

Start Cell = A1

Goal Cell = G6

Percepts:
{L, C, R}



Actions:
Forward
LeftTurn
RightTurn
UTurn

Model-Based Reflex Agent

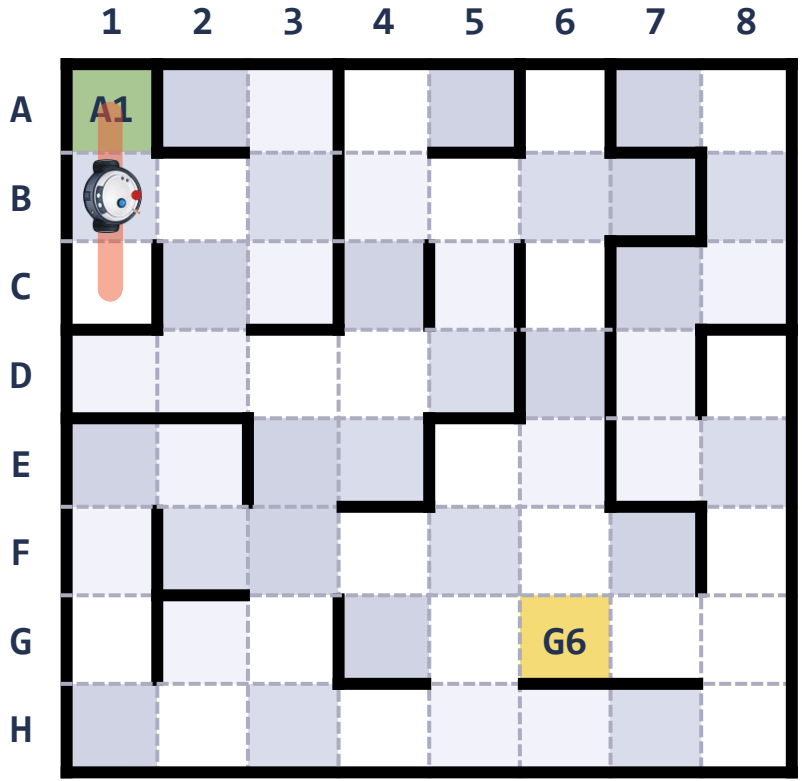
Condition-Action Rules:

```
IF C=False AND Loc+1 not in visited THEN
    Forward AND add Loc+1 in visited
ELSE IF L=False AND Loc+1 not in visited THEN LeftTurn
ELSE IF L=False AND Loc+1 in visited THEN RightTurn
ELSE IF R=False Loc+1 not in visited THEN RightTurn
ELSE IF R=False Loc+1 in visited THEN LeftTurn
ELSE IF L,C,R=False THEN UTurn
ELSE IF Loc+1 in visited THEN UTurn
```

Loc+1 = Next location after performing action

Visited list

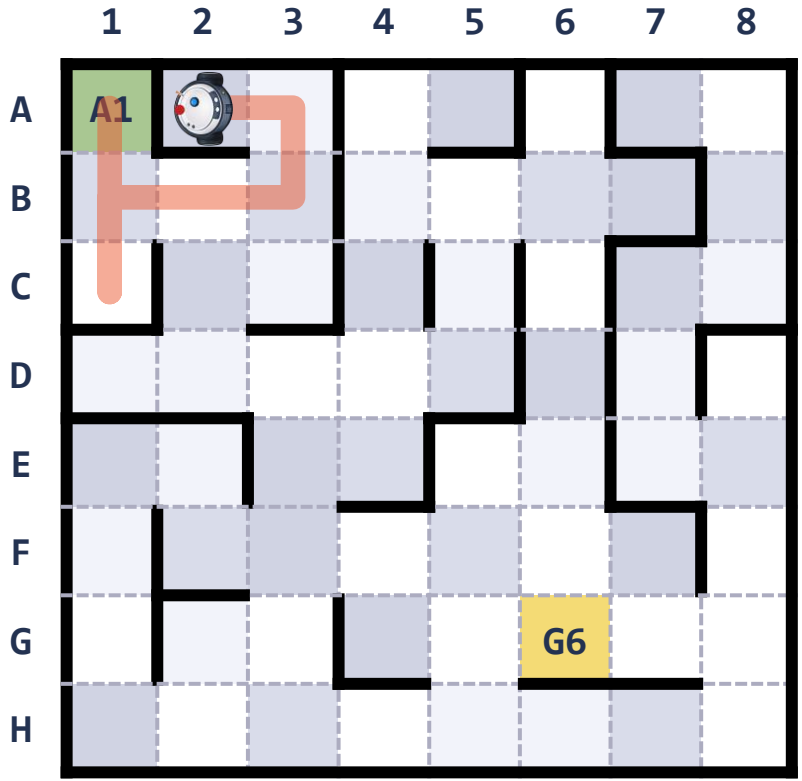
Model-Based Reflex Agent



Step	Percept	Action	Loc (Before)	Loc (After)
Start			A1	
		Visited = [A1]		
1	(T,F,T)	Forward	A1	B1
		[A1,B1]		
2	(F,F,T)	Forward	B1	C1
		[A1,B1,C1]		
3	(T,T,T)	UTurn	C1	C1
		[A1,B1,C1]		
4	(T,F,T)	Forward	C1	B1
		[A1,B1,C1]		
5	(T,F,T)	RightTurn	B1	B1
		[A1,B1,C1]		

A1→B1→C1→B1→

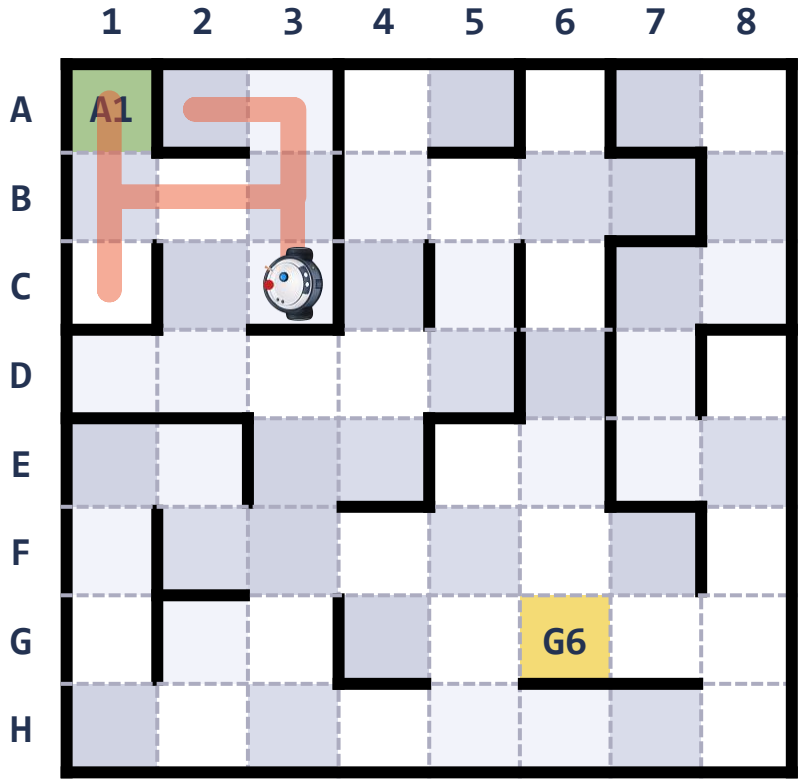
Model-Based Reflex Agent



Step	Percept	Action	Loc (Before)	Loc (After)
6	(F,F,F)	Forward	B1	B2
		[A1,B1,C1,B2]		
7	(T,F,F)	Forward	B2	B3
		[A1,B1,C1,B2,B3]		
8	(F,T,F)	LeftTurn	B3	B3
		[A1,B1,C1,B2,B3]		
9	(F,F,T)	Forward	B3	A3
		[A1,B1,C1,B2,B3,A3]		
10	(F,T,T)	LeftTurn	A3	A3
		[A1,B1,C1,B2,B3,A3]		
11	(F,F,T)	Forward	A3	A2
		[A1,B1,C1,B2,B3,A3,A2]		

A1→B1→C1→B1→B2→B3→A3→A2→

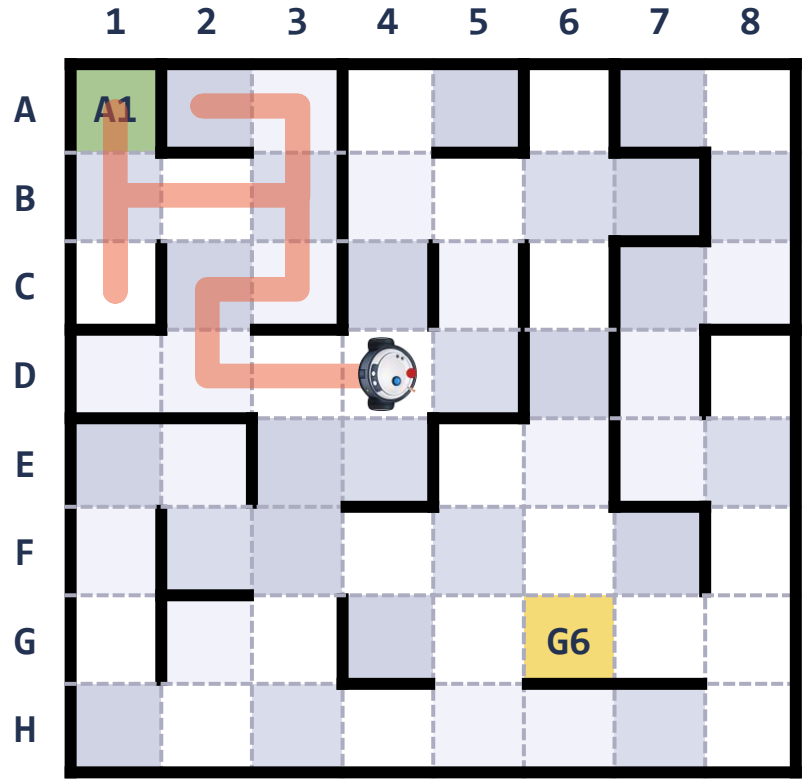
Model-Based Reflex Agent



Step	Percept	Action	Loc (Before)	Loc (After)
12	(T,T,T)	UTurn	A2	A2
		[A1,B1,C1,B2,B3,A3,A2]		
13	(T,F,T)	Forward	A2	A3
		[A1,B1,C1,B2,B3,A3,A2]		
14	(T,T,F)	RightTurn	A3	A3
		[A1,B1,C1,B2,B3,A3,A2]		
15	(T,F,F)	Forward	A3	B3
		[A1,B1,C1,B2,B3,A3,A2]		
16	(T,F,F)	Forward	B3	C3
		[A1,B1,C1,B2,B3,A3,A2,C3]		
17	(T,T,F)	RightTurn	C3	C3
		[A1,B1,C1,B2,B3,A3,A2,C3]		

A1→B1→C1→B1→B2→B3→A3→A2→A3→
B3→C3→

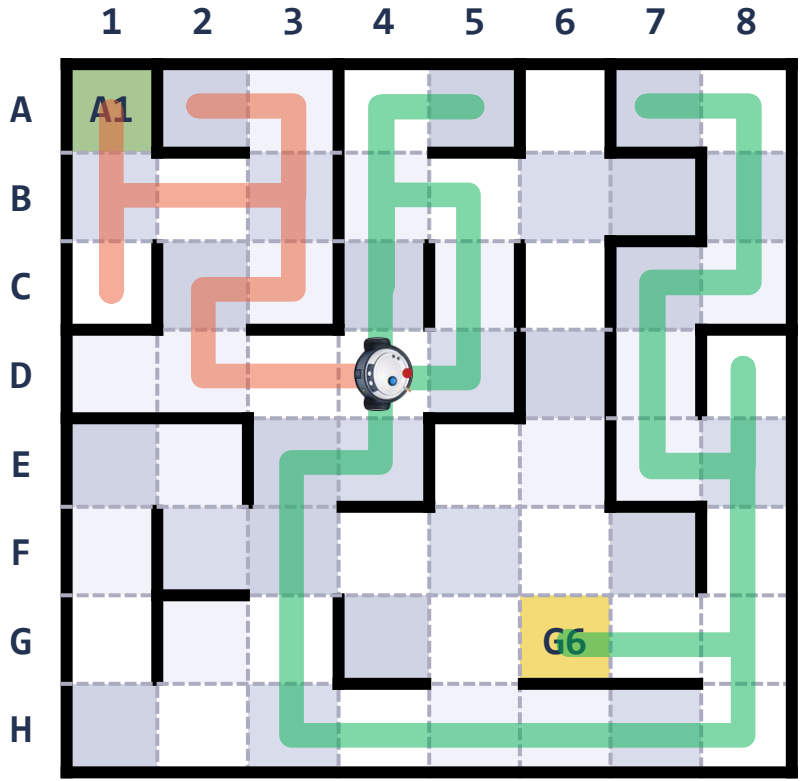
Model-Based Reflex Agent



Step	Percept	Action	Loc (Before)	Loc (After)
18	(T,F,F)	Forward	C3	C2
	[A1,B1,C1,B2,B3,A3,A2,C3,C2]			
19	(F,T,F)	LeftTurn	C2	C2
	[A1,B1,C1,B2,B3,A3,A2,C3,C2]			
20	(F,F,T)	Forward	C2	D2
	[A1,B1,C1,B2,B3,A3,A2,C3,C2,D2]			
21	(F,T,F)	LeftTurn	D2	D2
	[A1,B1,C1,B2,B3,A3,A2,C3,C2,D2]			
22	(F,F,T)	Forward	D2	D3
	[A1,B1,C1,B2,B3,A3,A2,C3,C2,D2,D3]			
23	(T,F,F)	Forward	D3	D4
	[A1,B1,C1,B2,B3,A3,A2,C3,C2,D2,D3,D4]			

A1→B1→C1→B1→B2→B3→A3→A2→A3→
B3→C3→C2→D2→D3→D4→

Model-Based Reflex Agent



A1→B1→C1→B1→B2→B3→A3→A2→A3→
B3→C3→C2→D2→D3→D4→D5→C5→B5→
B4→A4→A5→A4→B4→C4→D4→E4→E3→
F3→G3→H3→H5→H6→H7→H8→G8→F8→
E8→D8→E8→E7→D7→C7→C8→B8→A8→
A7→A8→B8→C8→C7→D7→E7→E8→F8→
G8→G7→G6

Maze Grid Navigation Agent

Goal-Based Agent

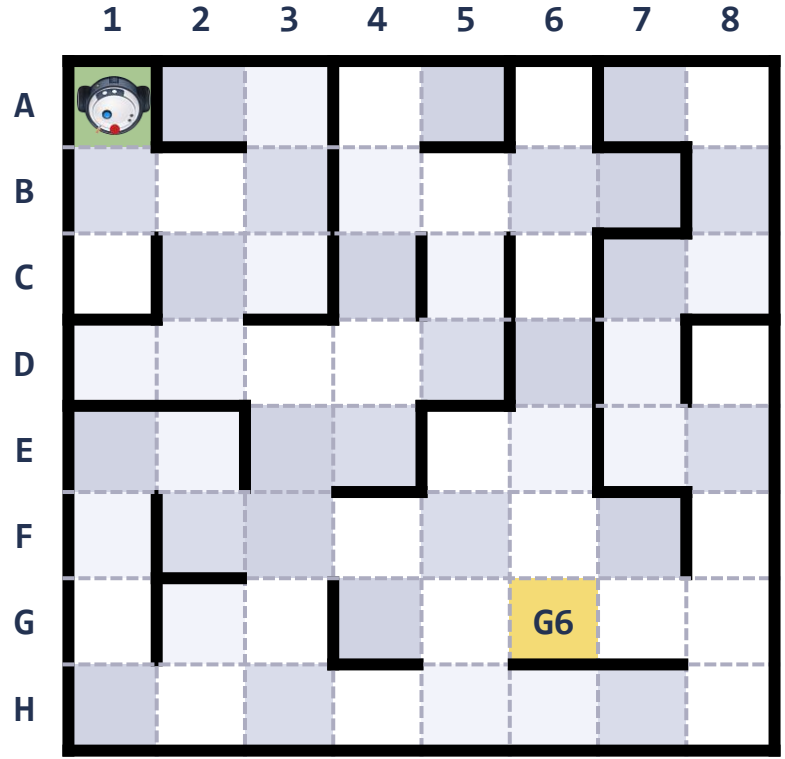
Goal-Based Agent (Example 1)

Start Cell = A1

Goal Cell = G6

Percepts:
{L, C, R}

Actions:
Forward
LeftTurn
RightTurn
UTurn



Agent Goal = Minimize the cost

- Navigate from A1 to G6
- Moving one cell increases cost of +1
- Goal is to choose a path with minimum cost

Goal-Based Agent (Example 1)

Rules:

Step 01: Find all paths $P_1 - P_N$ from Start Cell to Goal Cell

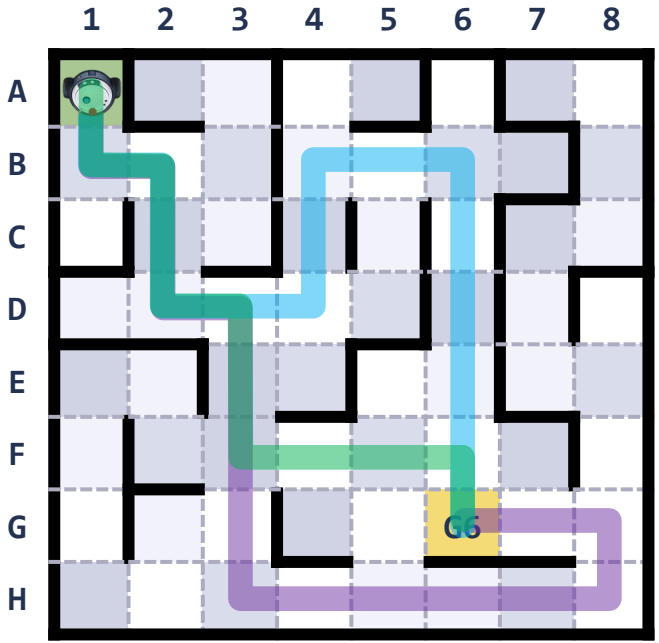
Step 02: Calculate cost $C_1 - C_N$ of each path $P_1 - P_N$

IF FORWARDS THEN C_i = C_i + 1

Step 03: Choose path P_i with minimum cost C_i

Step 04: Perform action sequence to follow path P_i

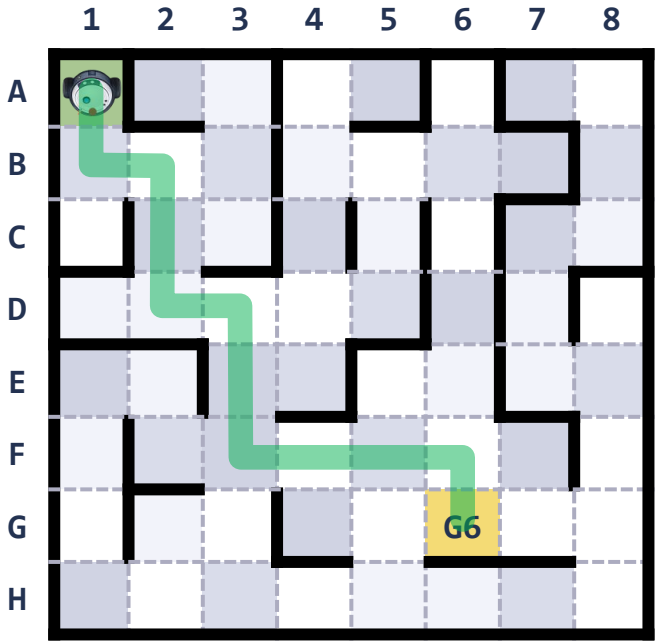
Goal-Based Agent (Example 1)



Path	Cost	Path Navigation
P_1	24	A1→B1→B2→C2→D2→D3→E3→F3→G6→H3→H4→H5→H6→H7→H8→G8→G7→G6
P_2	21	A1→B1→B2→C2→D2→D3→D4→C4→B4→B5→B6→C6→D6→E6→F6→G6
P_3	17	A1→B1→B2→C2→D2→D3→E3→F3→F4→F5→F6→G6
...	...	...

Path P_3 has minimum cost, hence we choose P_3.

Goal-Based Agent (Example 1)



A1→B1→B2→C2→D2→D3→E3→F3→F4→
F5→F6→G6

Action Sequence:

{F,L,F,R,F,F,L,F,R,F,F,L,F,F,F,R,F}

F=Forward, L=LeftTurn, R=RightTurn, U=UTurn

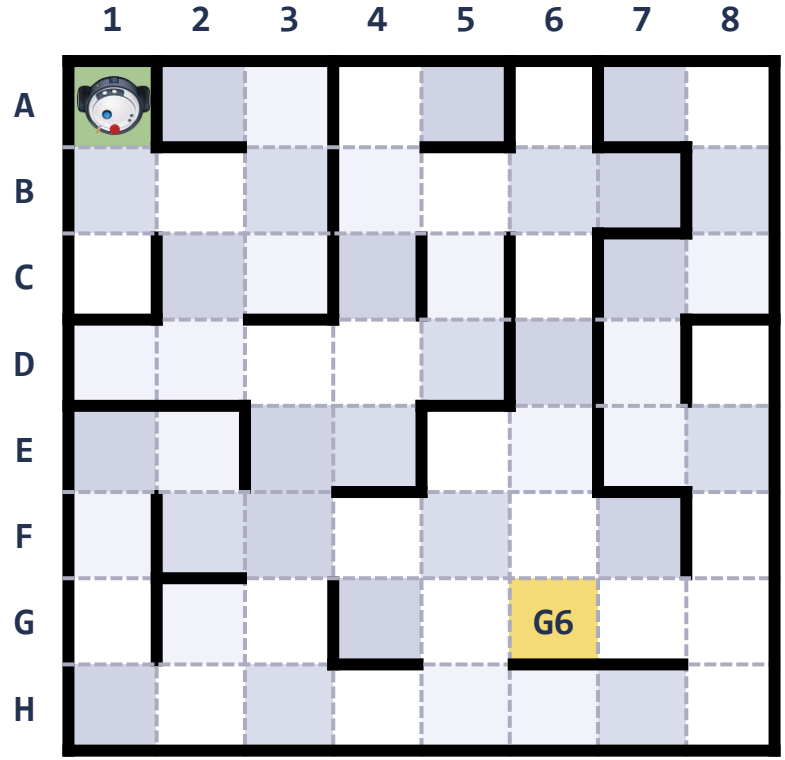
Goal-Based Agent (Example 2)

Start Cell = A1

Goal Cell = G6

Percepts:
{L, C, R}

Actions:
Forward
LeftTurn
RightTurn
UTurn



Agent Goal = Minimize the cost

- Navigate from A1 to G6
- Moving one cell increases cost of +1
- Each Left or Right turn increases cost of +2
- Each UTurn increases cost of +3
- Goal is to choose a path with minimum cost

Goal-Based Agent (Example 2)

Rules:

Step 01: Find all paths $P_1 - P_N$ from Start Cell to Goal Cell

Step 02: Calculate cost $C_1 - C_N$ of each path $P_1 - P_N$

IF FORWARDS THEN C_i = C_i + 1

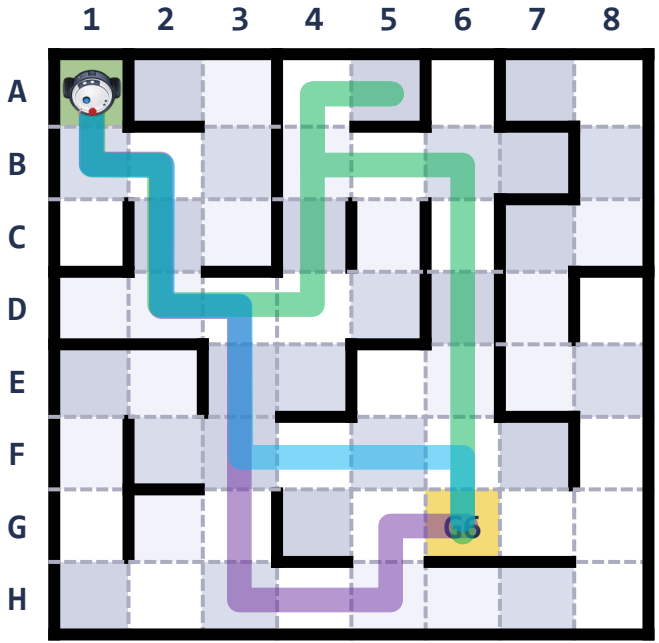
ELSE IF LeftTurn OR RightTurn THEN C_i = C_i + 2

ELSE IF UTurn THEN C_i = C_i + 3

Step 03: Choose path P_i with minimum cost C_i

Step 04: Perform action sequence to follow path P_i

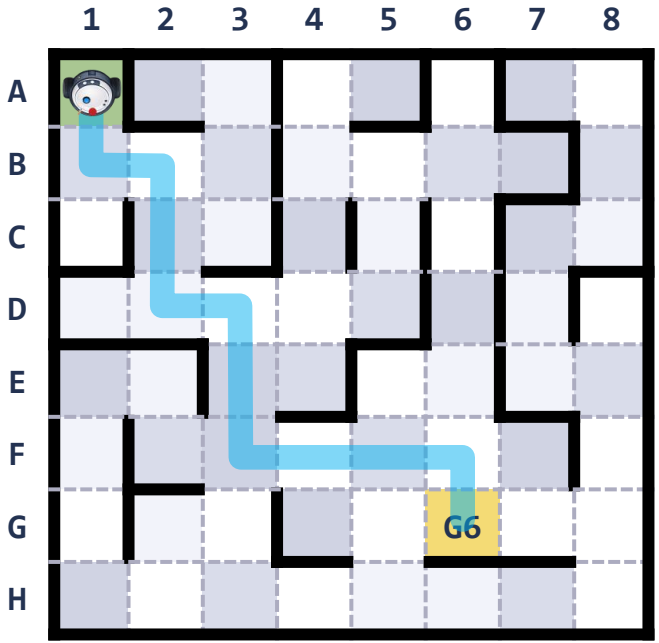
Goal-Based Agent (Example 2)



Path	Cost	Path Navigation
P_1	27	A1→B1→B2→C2→D2→D3→E3→F3→ G6→H3→H4→H5→G5→G6
P_2	23	A1→B1→B2→C2→D2→D3→E3→F3→ F4→F5→F6→G6
P_3	26	A1→B1→B2→C2→D2→D3→D4→C4→ B4→A4→A5→A4→B4→B5→B6→C6→ D6→E6→F6→G6
...	...	...

Path P_2 has minimum cost, hence we choose P_2.

Goal-Based Agent (Example 2)



A1→B1→B2→C2→D2→D3→E3→F3→
F4→F5→F6→G6

Action Sequence:

{F,L,F,R,F,F,L,F,R,F,F,L,F,F,F,R,F}

F=Forward, L=LeftTurn, R=RightTurn, U=UTurn

Maze Grid Navigation Agent

Utility-Based Agent

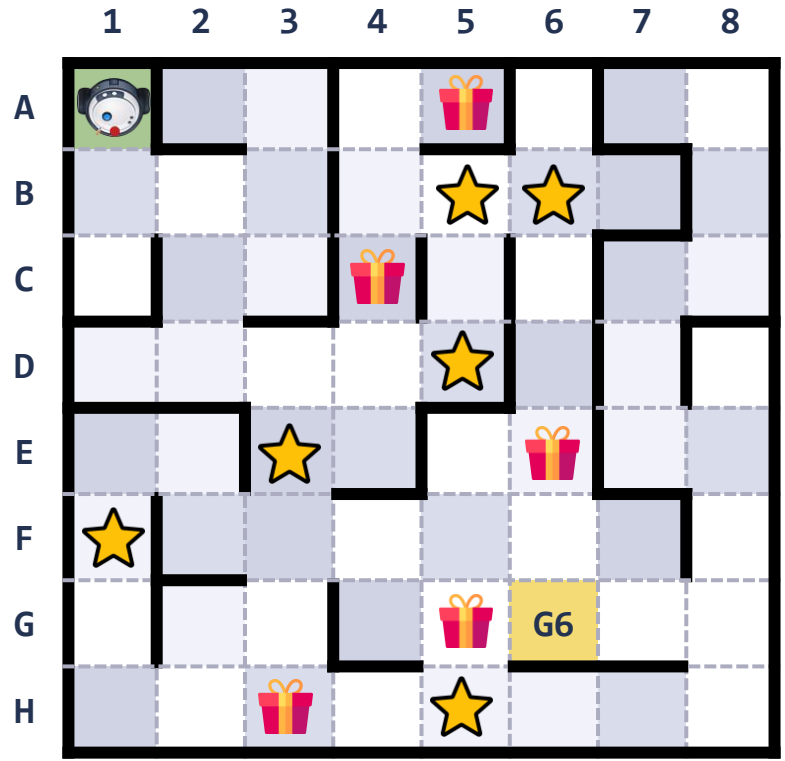
Utility-Based Agent

Start Cell = A1

Goal Cell = G6

Percepts:
{L, C, R}

Actions:
Forward
LeftTurn
RightTurn
UTurn



Agent Goal = Minimize the cost

- Navigate from A1 to G6 with minimum cost
- Moving one cell increases cost of +1

Utility = Maximize Reward

- Collect rewards and maximize them
- : reward of +8
- : reward of +5

Utility-Based Agent

Rules:

Step 01: Find all paths $P_1 - P_N$ from Start Cell to Goal Cell

Step 02: Calculate cost $C_1 - C_N$ of each path $P_1 - P_N$

IF FORWARDS THEN $C_i = C_i + 1$

Step 03: Calculate reward $R_1 - R_N$ of each path $P_1 - P_N$

IF 🎁 THEN $R_i = R_i + 8$

ELSE IF ⭐ THEN $R_i = R_i + 5$

Step 04: Calculate utility $U_1 - U_N$ of each path $P_1 - P_N$

Use Utility Function $U_i = U(C_i, R_i)$

Step 05: Choose path P_i with maximum utility U_i

Step 06: Perform action sequence to follow path P_i

Utility-Based Agent

Utility Function:

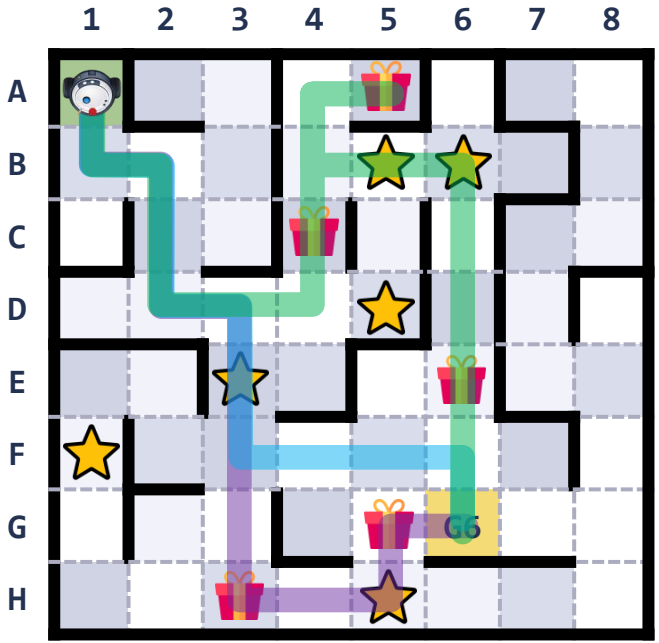
$$utility = U(cost, rewards)$$

$$U_i = U(C_i, R_i)$$

$$U(C_i, R_i) = R_i - \frac{C_i}{2}$$

This is an example utility function; the choice of the function depends on the problem being considered. Hence it may vary depending on the agent and environment.

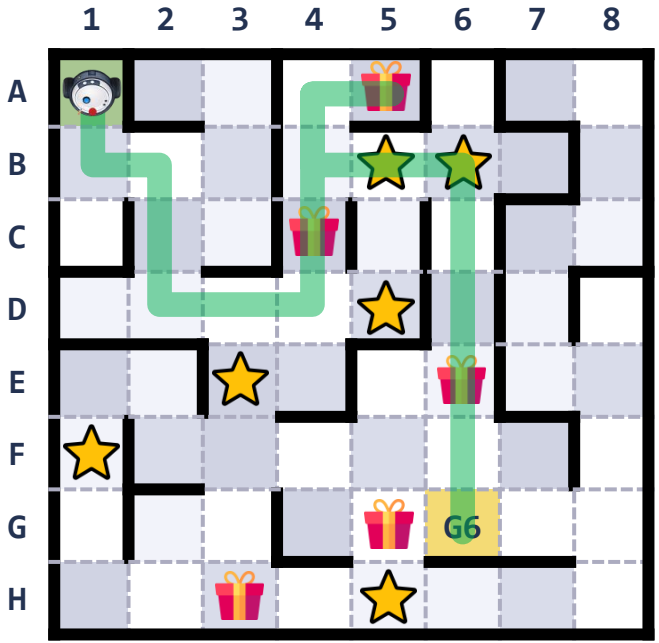
Utility-Based Agent



P_i	C_i	R_i	U_i	Path Navigation
P_1	12	26	20.0	A1→B1→B2→C2→D2→D3→E3→F3→G6→H3→H4→H5→G5→G6
P_2	11	5	-0.5	A1→B1→B2→C2→D2→D3→E3→F3→F4→F5→F6→G6
P_3	19	34	24.5	A1→B1→B2→C2→D2→D3→D4→C4→B4→A4→A5→A4→B4→B5→B6→C6→D6→E6→F6→G6
…	…	…	…	…

Path P_3 has maximum utility, hence we choose P_3.

Utility-Based Agent



A1→B1→B2→C2→D2→D3→D4→C4→
B4→A4→A5→A4→B4→B5→B6→C6→
D6→E6→F6→G6

Action Sequence:

{F,L,F,R,F,F,L,F,F,L,F,F,F,R,F,U,F,L,F,
L,F,F,R,F,F,F,F,F}

F=Forward, L=LeftTurn, R=RightTurn, U=UTurn

Structure of Intelligent Agent

- The job of AI is to design the agent **program**: a function that implements the agent mapping from percepts to actions.
- We assume this program will run on some sort of computing device, which we will call the **architecture**.
- The architecture might be a plain computer, or it might include special purpose hardware for certain tasks.
- The relationship among agents, architectures, and programs can be summed up as follows:

Agent = Architecture + Program

Structure of Intelligent Agent

- Before we design an agent program, we must have a pretty good idea of:
  - Possible percepts and actions.
  - What goals or performance measure the agent is supposed to achieve.
  - What sort of environment it will operate in.

Agents VS Expert Systems

Agents

1. Always exist in environment.
2. Obtain information through sensors.
3. Act on environment.
4. Can cooperate with other agents and humans.

Expert Systems

1. Do not exist in environment.
2. Obtain information from domain expert.
3. Do not act on environment (give feedback/advise)
4. Do not cooperate with other expert systems.

Environment

- An environment is everything **external to an agent** with which the agent interacts.
- It is everything that surrounds the agent that is not the part of agent itself.
- This is where agent lives or operates.
- It provide the agent with something to sense and somewhere to move around it.
- The environment produces **percepts (inputs)** for the agent through sensors and changes **state** based on the **agent's actions** executed via actuators.
- The agent does not control the environment directly; it can only **influence** it through actions.

AGENT + ENVIRONMENT ▶ Agent perceives ▶ Agent acts ▶ Environnent changes

Properties of Environment

1. Fully observable / Partially observable:

Observability

- If agent has access to entire state of the environment through sensor, it is fully observable.
- If part of the environment is observable, it is partially observable.

Fully observable	Agent's sensors provide complete information about the environment state	Chess
Partially observable	Sensors provide incomplete or noisy information	Driving a car

Properties of Environment

2. Deterministic/Stochastic:

Determinism

- An environment is deterministic if the next state of the environment is completely determined by the current state of the environment and the action of the agent.
- In a stochastic environment, there are multiple, unpredictable outcomes.
- In a fully observable, deterministic environment, the agent need not deal with uncertainty.

Deterministic	Next state is completely determined by current state and action	Crossword puzzle
Stochastic	Outcomes involve randomness or uncertainty	Medical diagnosis

Properties of Environment

3. Episodic/Sequential: Temporal Structure

- The environment is episodic if each of the agent's tasks does not rely on past performance or cannot affect future performance.
- If not, then it is sequential.

Episodic	Each action is independent of previous actions	Spam filtering
Sequential	Current decisions affect future states	Game playing

Properties of Environment

4. Static/Dynamic: Dynamics

- A static environment does not change while the agent is thinking.
- The passage of time as an agent deliberates is irrelevant.
- The agent doesn't need to observe the world during deliberation.
- If the environment can change while an agent is deliberating, then we say the environment is dynamic for the agent.

Static	Environment does not change while agent is deliberating	Sudoku puzzle
Dynamic	Environment changes during agent's decision process	Stock trading

Properties of Environment

5. Single-Agent/Multi-Agent: Number of Agents

- In single-agent, there is only one agent working in the environment.
- In multi-agent environment, there are other intelligent agents as well which are either cooperative or competitive agents.

Single-Agent	Only one agent acts in the environment	Maze-solving robot
Multi-Agent	Multiple agents interact (cooperate or compete)	Drone fleet

Properties of Environment

6. Discrete/Continuous: State Space

- Discrete has finite number of possible states.
- Number of states in continuous environment is infinite.

Discrete	Finite number of states, percepts, and actions	Chess
Continuous	Infinite or real-valued states/actions	Robot motion control

Examples of Agent Types and PAGE Descriptors

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient answers	Questions, tests, treatment	Healthy patient, minimize cost	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 belt 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

Automated Taxi Driver Agent Example

- Consider designing an automated taxi driver agent.
- The full driving task is extremely open-ended—there is no limit to the novel combinations of circumstances that can arise.

Agent Type	Percepts	Actions	Goals	Environment
Taxi Driver	Cameras, Speedometer, GPS, Sonar, Microphone	Steer, Accelerate, Brake, Talk to Passenger	Safe, Fast, Legal, Confirmable trip, Maximize profits	Road, Other Traffic, Pedestrians, Customers

Automated Taxi Driver Agent Example

- The taxi will need to know:
  - Where it is?
  - What else is on the road?
  - How fast it is going?

- This information can be obtained from the percepts (sensors).

- The actions available to a taxi driver will be more or less the same ones available to a human driver.

Automated Taxi Driver Agent Example

- Performance measure:
  - Correct destination.
  - Minimizing fuel consumption.
  - Minimizing the trip time and/or cost.
  - Minimizing violations of traffic laws and disturbances to other drivers.
  - Maximizing safety and passenger comfort.
  - Maximizing profits.
- Obviously, some of these goals **conflict**, so there will be **trade-offs** involved.

Automated Taxi Driver Agent Example

- Finally, we would need to decide what kind of driving environment the taxi will face.
- Should it operate on local roads, or also on Highways?
- Will it be in the road where snow is seldom a problem?
- Will it always be driving on the right, or might we want it to be flexible enough to drive on the left in case we want to operate taxis in Britain or Japan?