Agents

Lab 1

Agenda

1. Running example: vacuum-cleaner world
2. Table-driven agent
3. Simple reflex agent
4. Reflex agent with state/memory
5. Homework

Vacuum-cleaner world

Percepts:

Location, status (e.g., [A, dirty])

Actions:

Left, Right, Suck, NoOperation

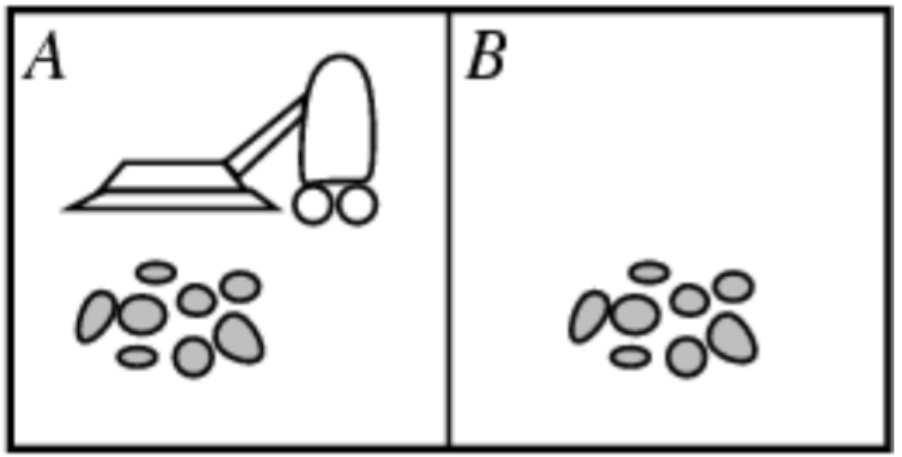


Table-driven agent

Table-driven agent

- Refer to table_driven_agent.py
- **table_definition** contains all possible *percepts* that can occur
- Each step appends current *percept* to list of *percepts*
- **LOOKUP** current *percepts* in *table*

Table-driven agent

function TABLE-DRIVEN-AGENT(*percept)* **returns** an action
static: *percepts,* a sequence, initially empty
table, a table of actions, indexed by percept sequences, initially fully specified

append *percept* to the end of *percepts*
action = LOOKUP*(percepts, table)*
return *action*

```python
def TABLE_DRIVEN_AGENT(percept: Percept) -> Action:  # Determine action based on table and percepts
    total_percepts.append(percept)  # Add percept
    return LOOKUP(total_percepts, table_definition)  # Lookup appropriate action for percepts
```

Exercise 1 - A complicated history

1. Run the module (using run())
2. The percepts should now be: [(‘A’, ’Clean’), (‘A’, ’Dirty’), (‘´B’, ’Clean’)]
   - The table contains all possible percept sequences to match with the percept history
   - Enter:
     print(TABLE_DRIVEN_AGENT((B, 'Clean')), '\t', percepts)
   - Explain the results
3. How many table entries would be required if only the *current(single)* percept was used to select and action rather than the percept history?
4. How many table entries are required for an agent lifetime of T steps?

Reflex vacuum agent

using condition-action rules and if statements

Reflex vacuum agent

- Refer to reflex_vacuum_agent.py
- Only responds to current percept (location and status) ignoring percept history
- Uses *condition-action* rules rather than a table
  - **if** *condition* **then return** *action*
  - **if** *status* = *Dirty* **then return** *Suck*
- **Sensors()** – Function to sense current location and status of environment (i.e., *location* of agent and *status* of square)
- **Actuators(action)** – Function to affect current environment location by some action (i.e., *Suck, Left, Right, NoOp*)

Simple reflex agent

function REFLEX-VACUUM-AGENT(*[location, status]*)
returns an action
if *status* = *Dirty* **then return** *Suck*
else if *location* = *A* **then return** *Right*
else if *location* = *B* **then return** *Left*

```python
def evaluate(self) -> Action:
    """:return: The action that the agent has chosen to take. For printing purposes"""
    state = self.sensor()

    action = self.choose_action(state)

    self.actuator(action)

    return action
```

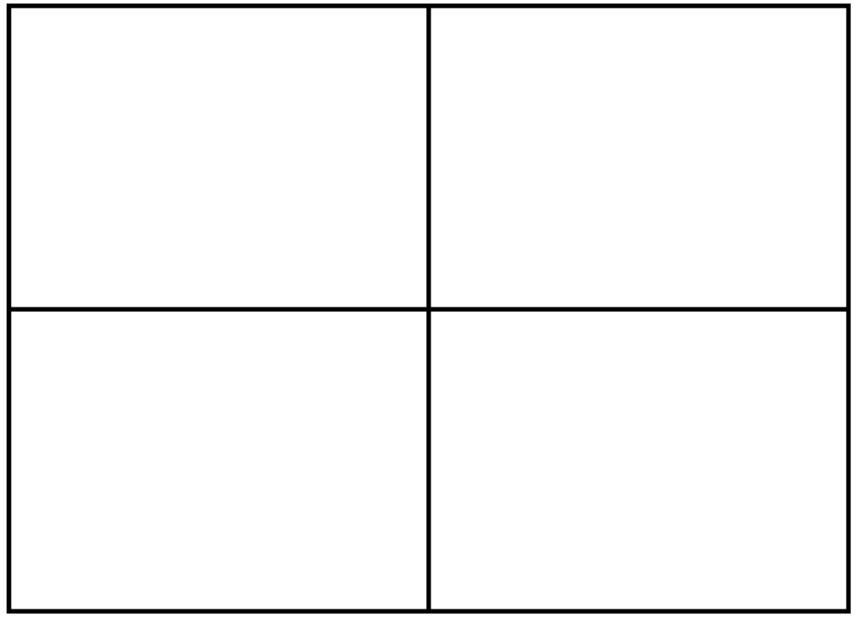
```python
@staticmethod
def choose_action(state: LocationState) -> Action:
    if state[1] == States.DIRTY:
        return Action.SUCK
    if state[0] == Location.A:
        return Action.RIGHT
    if state[0] == Location.B:
        return Action.LEFT
```

Exercise 2 – Bogus actions

1. Run the module
2. Enter *run(10)*
3. Should bogus actions be able to corrupt the environment? Change the REFLEX_VACUUM_AGENT to return bogus action, such as *Left* when it should go *Right* etc. Run the agent. Do the Actuators allow bogus actions?

Exercise 3 – A whole new world

- Extend the REFLEX_VACUUM_AGENT (Exercise 2) program to have 4 locations (4 squares)
  - –The agent should only sense and act on the square where it is located
  - –Allow any starting square
  - –Use run(20) to test and display results
  - –Hint investigate Enums.py



Reflex agent with state/memory

Reflex agent with state

- Reflex agent only responded to current percepts; no history or knowledge
- Model-based reflex agents:
  - Maintain internal state that depends upon percept history
  - Agent has a model of how the world works
  - The model requires two types of information to update:
    - How environment evolves independent of the agent (e.g., Clean square stays clean)
    - How agent’s action affect the environment (e.g., Suck cleans square)

Reflex agent with state

- Refer to reflex_agent_with_state.py
- Model – used to update history
  - History initially empty:
    model = {A: Unknown, B: Unknown}
  - Model only used to change state when A == B == ‘Clean’
    if model[A] == model[B] == ‘Clean’: action= NO_OP

Reflex agent with state

function REFLEX-AGENT-WITH-STATE(*percept*) **returns** an action
static: *state,* a description of the current world state
rules, a sequence, a set of condition-action rules
action, the most recent action, initially none
state = UPDATE-STATE(*state, action, percept*)
rule = RULE-MATCH(*state, rules*)
action = RULE-ACTION*[rule]*
return *action*

```
def act(self, environment: EnvironmentClass) -> Action:
    percept = self.sensors(environment)
    self.state = percept
    self.update_state(percept)
    action = self.match_rule()
    self.actuators(action, environment)
    return action
```

Homework

Homework – Remembering the whole world

- Extend the REFLEX_AGENT_WITH_STATE program to have 4 locations
  - The agent should only sense and act on the square where it is located
  - Allow any starting square
  - Use run(20) to test and display results