THE VACUUM WORLD

In this notebook, we will be discussing the structure of agents through an example of the vacuum agent. The job of AI is to design an agent program that implements the agent function: the mapping from percepts to actions. We assume this program will run on some sort of computing device with physical sensors and actuators: we call this the architecture:

agent = architecture + program

Random Agent Program

A random agent program, as the name suggests, chooses an action at random, without taking into account the percepts.
Here, we will demonstrate a random vacuum agent for a trivial vacuum environment, that is, the two-state environment.

Let's begin by importing all the functions from the agents module:

In [63]:

```
from aima3.agents import *
from aima3.notebook import psource
```

Let us first see how we define the TrivialVacuumEnvironment. Run the next cell to see how abstract class TrivialVacuumEnvironment is defined in agents module:

In [64]:

```
psource(TrivialVacuumEnvironment)
```

```
class TrivialVacuumEnvironment(Environment):

    """This environment has two locations, A and B. Each can be Dirty
    or Clean. The agent perceives its location and the location's
    status. This serves as an example of how to implement a simple
    Environment."""

    def __init__(self):
        super().__init__()
        self.status = {loc_A: random.choice(['Clean', 'Dirty']),
                       loc_B: random.choice(['Clean', 'Dirty'])}

    def thing_classes(self):
        return [Wall, Dirt, ReflexVacuumAgent, RandomVacuumAgent,
                TableDrivenVacuumAgent, ModelBasedVacuumAgent]

    def percept(self, agent):
        """Returns the agent's location, and the location status (Dirty/Clean)."""
        return (agent.location, self.status[agent.location])

    def execute_action(self, agent, action):
        """Change agent's location and/or location's status; track performance.
        Score 10 for each dirt cleaned; -1 for each move."""
        if action == 'Right':
            agent.location = loc_B
            agent.performance -= 1
        elif action == 'Left':
            agent.location = loc_A
            agent.performance -= 1
            agent.performance -= 1
        elif action == 'Suck':
            if self.status[agent.location] == 'Dirty':
                agent.performance += 10
            self.status[agent.location] = 'Clean'

    def default_location(self, thing):
        """Agents start in either location at random."""
        return random.choice([loc_A, loc_B])
```

In [94]:

```
# These are the two locations for the two-state environment
loc_A, loc_B = (0, 0), (1, 0)

# Initialize the two-state environment
trivial_vacuum_env = TrivialVacuumEnvironment()

# Check the initial state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))
```

```
State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Clean'}.
```

Let's create our agent now. This agent will choose any of the actions from 'Right', 'Left', 'Suck' and 'NoOp' (No Operation) randomly.

RANDOM AGENT PROGRAM

In [95]:

```
# Create the random agent
random_agent = Agent(program=RandomAgentProgram(['Right', 'Left', 'Suck', 'NoOp']))
```

We will now add our agent to the environment.

In [96]:

```
# Add agent to the environment
trivial_vacuum_env.add_thing(random_agent)

print("RandomVacuumAgent is located at {}.".format(random_agent.location))
```

```
RandomVacuumAgent is located at (0, 0).
```

Let's run our environment now.

In [100]:

```
# Running the environment
trivial_vacuum_env.step()

# Check the current state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))

print("RandomVacuumAgent is located at {}.".format(random_agent.location))
```

```
State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Clean'}.
RandomVacuumAgent is located at (1, 0).
```

TABLE-DRIVEN AGENT PROGRAM

A table-driven agent program keeps track of the percept sequence and then uses it to index into a table of actions to decide what to do. The table represents explicitly the agent function that the agent program embodies.

In the two-state vacuum world, the table would consist of all the possible states of the agent.

In [101]:

```
table = {((loc_A, 'Clean'),): 'Right',
             ((loc_A, 'Dirty'),): 'Suck',
             ((loc_B, 'Clean'),): 'Left',
             ((loc_B, 'Dirty'),): 'Suck',
             ((loc_A, 'Dirty'), (loc_A, 'Clean')): 'Right',
             ((loc_A, 'Clean'), (loc_B, 'Dirty')): 'Suck',
             ((loc_B, 'Clean'), (loc_A, 'Dirty')): 'Suck',
             ((loc_B, 'Dirty'), (loc_B, 'Clean')): 'Left',
             ((loc_A, 'Dirty'), (loc_A, 'Clean'), (loc_B, 'Dirty')): 'Suck',
             ((loc_B, 'Dirty'), (loc_B, 'Clean'), (loc_A, 'Dirty')): 'Suck'
        }
```

We will now create a table-driven agent program for our two-state environment.

In [102]:

```
# Create a table-driven agent
table_driven_agent = Agent(program=TableDrivenAgentProgram(table=table))
```

Since we are using the same environment, let's remove the previously added random agent from the environment to avoid confusion.

In [103]:

```
trivial_vacuum_env.delete_thing(random_agent)
```

In [104]:

```
# Add the table-driven agent to the environment
trivial_vacuum_env.add_thing(table_driven_agent)

print("TableDrivenVacuumAgent is located at {}.".format(table_driven_agent.location))
```

```
TableDrivenVacuumAgent is located at (1, 0).
```

In [105]:

```
for x in range(3):
    # Run the environment
    trivial_vacuum_env.step()

    # Check the current state of the environment
    print("State of the Environment: {}.".format(trivial_vacuum_env.status))

    print("TableDrivenVacuumAgent is located at {}.".format(table_driven_agent.location))
```

```
State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Clean'}.
TableDrivenVacuumAgent is located at (0, 0).
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Clean'}.
TableDrivenVacuumAgent is located at (0, 0).
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Clean'}.
TableDrivenVacuumAgent is located at (0, 0).
```

SIMPLE REFLEX AGENT PROGRAM

A simple reflex agent program selects actions on the basis of the *current* percept, ignoring the rest of the percept history. These agents work on a **condition-action rule** (also called **situation-action rule**, **production** or **if-then rule**), which tells the agent the action to trigger when a particular situation is encountered.

Let us now create a simple reflex agent for the environment.

In [84]:

```
# Delete the previously added table-driven agent
trivial_vacuum_env.delete_thing(table_driven_agent)
```

```
list.remove(x): x not in list
  in Environment delete_thing
  Thing to be removed: <Agent> at (0, 0)
  from list: [(<Agent>, (1, 0))]
```

To create our agent, we need two functions: INTERPRET-INPUT function, which generates an abstracted description of the current state from the percept and the RULE-MATCH function, which returns the first rule in the set of rules that matches the given state description.

In [88]:

```
loc_A = (0, 0)
loc_B = (1, 0)

"""We change the simpleReflexAgentProgram so that it doesn't make use of the Rule class""
"
def SimpleReflexAgentProgram():
    """This agent takes action based solely on the percept. [Figure 2.10]"""

    def program(percept):
        loc, status = percept
        return ('Suck' if status == 'Dirty'
                else'Right' if loc == loc_A
                            else'Left')
    return program


# Create a simple reflex agent the two-state environment
program = SimpleReflexAgentProgram()
simple_reflex_agent = Agent(program)
```

Now add the agent to the environment:

In [117]:

```
# These are the two locations for the two-state environment
loc_A, loc_B = (0, 0), (1, 0)

# Initialize the two-state environment
trivial_vacuum_env = TrivialVacuumEnvironment()

# Check the initial state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))
```

```
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Dirty'}.
```

In [118]:

```
trivial_vacuum_env.add_thing(simple_reflex_agent)

print("SimpleReflexVacuumAgent is located at {}.".format(simple_reflex_agent.location))
```

```
SimpleReflexVacuumAgent is located at (0, 0).
```

In [119]:

```
for x in range(3):
    # Run the environment
    trivial_vacuum_env.step()

    # Check the current state of the environment
    print("State of the Environment: {}.".format(trivial_vacuum_env.status))

    print("SimpleReflexVacuumAgent is located at {}.".format(simple_reflex_agent.location
))
```

```
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Dirty'}.
SimpleReflexVacuumAgent is located at (1, 0).
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Clean'}.
SimpleReflexVacuumAgent is located at (1, 0).
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Clean'}.
SimpleReflexVacuumAgent is located at (0, 0).
```

MODEL-BASED REFLEX AGENT PROGRAM

A model-based reflex agent maintains some sort of internal state that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state. In addition to this, it also requires a model of the world, that is, knowledge about "how the world works".

We will now create a model-based reflex agent for the environment:

In [178]:

```
# Delete the previously added simple reflex agent
trivial_vacuum_env.delete_thing(simple_reflex_agent)
```

We need another function UPDATE-STATE which will be responsible for creating a new state description.

In [110]:

```
# These are the two locations for the two-state environment
loc_A, loc_B = (0, 0), (1, 0)

# Initialize the two-state environment
trivial_vacuum_env = TrivialVacuumEnvironment()

# Check the initial state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))
```

```
State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Dirty'}.
```

In [111]:

```
# TODO: Implement this function for the two-dimensional environment
def update_state(state, action, percept, model):
    pass

# Create a model-based reflex agent
model_based_reflex_agent = ModelBasedVacuumAgent()

# Add the agent to the environment
trivial_vacuum_env.add_thing(model_based_reflex_agent)

print("ModelBasedVacuumAgent is located at {}.".format(model_based_reflex_agent.location)
)
```

```
ModelBasedVacuumAgent is located at (1, 0).
```

In [112]:

```
for x in range(3):
    # Run the environment
    trivial_vacuum_env.step()

    # Check the current state of the environment
    print("State of the Environment: {}.".format(trivial_vacuum_env.status))
    print("ModelBasedVacuumAgent is located at {}.".format(model_based_reflex_agent.locat
ion))
```

```
ModelBasedVacuumAgent is located at (1, 0).
State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Clean'}.
ModelBasedVacuumAgent is located at (0, 0).
State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Clean'}.
ModelBasedVacuumAgent is located at (0, 0).
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Clean'}.
```

In [ ]: