MDPProblem

You will use the `kuimaze2.MDPProblem` environment in tasks where your goal is to find the optimal strategy for the Markov Decision Process (MDP). It is used in the third compulsory task `08-MDPs` .

Public interface

After creating an instance of the `MDPProblem` class (see Usage [/wiki/courses/be5b33kui/semtasks/kuimaze/20_mdpproblem#usage]), you can use the following methods:

- `env.get_states()` returns a list of all free states (instances of the `kuimaze2.State` class), i.e., without walls.
- `env.is_terminal(state)` returns `True` if the given state is terminal (the agent cannot leave the state). It can be either a "good" terminal state or a "bad" (dangerous) terminal state.
- `env.get_reward(state)` returns a reward for a given state. The reward is "paid out" only when leaving the state, not when reaching it.
- `env.get_actions(state)` returns all actions that are available in the given state. These actions are typically used in combination with the following method.
- `env.get_next_states_and_probs(state, action)` for a given state and requested action, it returns a list of possible resulting states and their probabilities, i.e., a list of pairs `(new_state, probability)` .
- `env.render()` updates the graphical display of the environment. (If you did not turn on the graphical display when creating the environment, it does nothing.) It allows you to "colorize" individual states and q-states, allows you to display texts in them, allows you to display a strategy. An explanation of the individual parameters of this method can be found in the help (`help(env.render)`). See also the use of `env.render()` in the auxiliary method `MDPAgent.render()` in the sample module `example_mdp.py` .

Usage

The environment is typically used as follows:

Import the environment:

```
>>> from kuimaze2 import Map, MDPProblem, State
```

Creating a map to initialize the environment:

```
>>> MAP = """
S.D
..G
"""
>>> map = Map.from_string(MAP)
```

Creating an environment, first deterministic:

```
>>> env1 = MDPProblem(map)
```

If you want to turn on the graphical display of the environment:

```
>>> env1 = MDPProblem(map, graphics=True)
```

If we want to create a non-deterministic environment (and we usually do in the case of MDP), we need to specify with what probability the environment will perform the agent's requested action and with what probabilities it will “slip somewhere else”.

```
>>> env2 = MDPProblem(map, action_probs=dict(forward=0.8, left=0.1, right=0.1, backward=0
```

List of all valid states in the environment:

```
>>> env2.get_states()
[State(r=0, c=0), State(r=0, c=1), State(r=0, c=2), State(r=1, c=0), State(r=1, c=1), Sta
```

Finding out if a state is terminal:

```
>>> env2.is_terminal(State(0, 0)), env2.is_terminal(State(0, 2))
(False, True)
```

What rewards are associated with individual states? Rewards are paid out when leaving the state.

```
>>> env2.get_reward(State(0,0)), env2.get_reward(State(0,2)), env2.get_reward(State(1,2))
(-0.04, 1.0, 1.0)
```

What actions are available in a state? In our environment, all 4 actions will always be available, but if we hit a wall, we will stay in place.

```
>>> actions = env2.get_actions(State(0, 0))
>>> actions
[<Action.UP: 0>, <Action.RIGHT: 1>, <Action.DOWN: 2>, <Action.LEFT: 3>]
```

To which states and with what probability can I get if I perform a certain action in a certain state? In a deterministic environment:

```
>>> env1.get_next_states_and_probs(State(0, 0), actions[0])
[(State(r=0, c=0), 1.0)]
```

In a non-deterministic environment, the result will be different:

```
>>> env2.get_next_states_and_probs(State(0, 0), actions[0])
[(State(r=0, c=0), 0.8), (State(r=0, c=1), 0.1), (State(r=1, c=0), 0.0), (State(r=0, c=0)
```

Note that some resulting states may appear multiple times in the list because they can be reached by different actions.

courses/be5b33kui/semtasks/kuimaze/20_mdpproblem.txt · Last modified: 2024/03/27 08:01 by xposik