

## Course 1: Implementing basic solving algorithms for robot path planning

### Exercise 1: Formalisation

**Example 1 (Grid World)** *An agent is in the bottom left cell of a grid  $n \times p$ . There are some uncrossable walls in the grid, one goal cell and a negative-rewarded well. The agents can move up, right, down and left. But! Things can go wrong — sometimes the effects of the actions are not what we want:*

- *the agent moves to the direction she intended to, with probability 0.7;*
- *the agent moves to one of the three other possible directions with probability 0.1;*
- *if the agent tries to go in a wall or if she slips (condition above, case that happens with probability 0.1), the agent stays where she is.*

*The task is to navigate from the start cell in the bottom left to maximise the expected reward. What would the best sequence of actions be for this problem?*

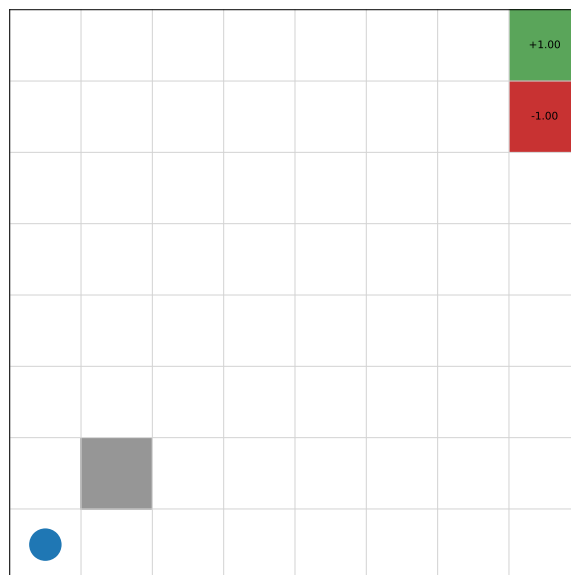


Figure 1: Illustration of the grid world environment

1. Propose a MDP formalism for this problem, when  $n = p = 3$ .
2. An implementation of the Grid World environment is given the github repository. Start implementing your own custom policy, *e.g.*:

```
1 from gridworld import *
2
3 mdp = GridWorld()
4
5 print("states:", mdp.get_states())
6 print("terminal states:", mdp.get_goal_states())
7 print("actions:", mdp.get_actions())
8 print(mdp.get_transitions(mdp.get_initial_state(), mdp.UP))
9
10 def policy_custom(state):
11     return mdp.UP
12
13 while(1):
14     state=mdp.get_initial_state()
15     new_state, _ = mdp.execute(state, policy_custom(state))
16     mdp.initial_state=new_state
17     mdp.visualise()
```

*Remark: the environment defines a “terminal state” (i.e the goal state).*

## Exercise 2: Solving algorithms

1. Implement the dynamic programming algorithm to learn optimal paths
2. TD & MC
  - (a) Q-learning and monte-carlo generate episodes. When does the episode stop?
  - (b) Implement Q-learning and monte-carlo algorithms. Do you observe something?
    - i. tips: use the “defaultdict” library in Python;

```
1 from collections import defaultdict
```

- ii. tips: a pair  $(s, a)$  can be represented by a tuple in Python and given as a key of a defaultdict.

3. (bonus) The environment provides visualisation tools for value function and policy. For dynamic programming, create classes “*value\_function*” and “*policy*”, which should respectively implement “*get\_value(states)*” and “*select\_action(states)*”. Then, call the visualisation tool of the environment with:

```
1 mdp.visualise_value_function(value_function)
2 mdp.visualise_policy(policy)
```

4. (bonus) Visualise the Q-learning & MC q-value function. To do so, create classes “*q\_function*”. Then, call the visualisation tool for the q-function:

```
1 mdp.visualise_q_function(q_function)
```

## Exercise 3: Analysis

Now that the algorithms are running - and hopefully learning something -, an important step is to analyze (i) their learning process and (ii) the resulting policy after reaching the time-limit budget for the learning process.

1. Implement adequate tools for analyzing the learning process.
2. Compare the policies obtained for all three algorithms after different time budgets.