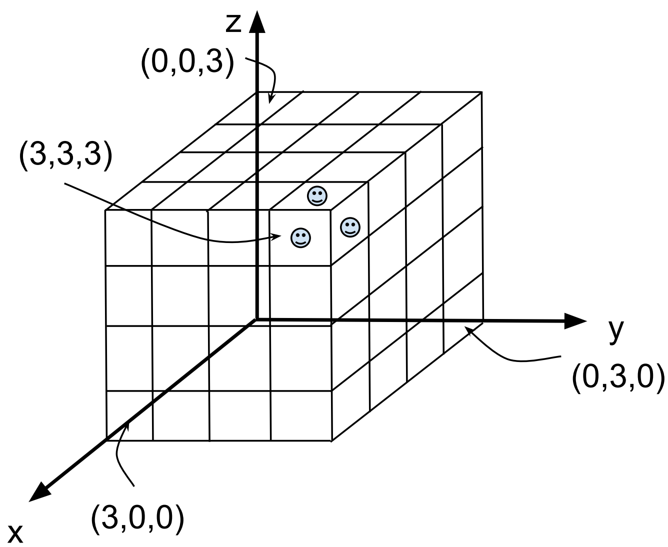
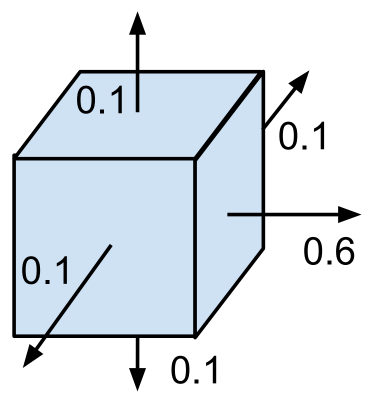
CZ3005 2020 Fall Assignment 2: Reinforcement Learning

1 Project Overview

In this project, you need to implement one reinforcement learning algorithm (e.g., value iteration, policy iteration, Q-learning) for one grid-world-based environment: Treasure Hunting.



(a) 3D grid world. Smile faces represent terminal states which give reward 1.



(b) The illustration of transition, e.g., the intended action is RIGHT

Figure 1: Illustration of treasure hunting in a cube

2 Treasure Hunting in a Cube

The environment is a 3D grid world. The MDP formulation is described as follows:

- State: a 3D coordinate, which indicates the current position where the agent is. The initial state is (0, 0, 0) and there is only one terminal state: (3,3,3).
- Action: The action space is (forward, backward, left, right, up, down). The agent needs to select one of them to navigate in the environment.
- Reward: The agent will receive 1 reward when it arrives at the terminal states, or otherwise receive -0.1 reward.
- Transition: The intended movement happens with probability 0.6. With probability 0.1, the agent ends up in one of the states perpendicular to the intended direction. The happened movement "forward"/"backward" will add/subtract one to the 1st element of state.

”Left”/”right” will add/subtract one to the 2nd element of state. ”Up/down” will add/subtract one to the 3nd element of state. If a collision with a wall happens, the agent stays in the same state.

3 Code Example

We provide the environment code environment.py and examples code test.py. In environment.py, we provide the code: TreasureCube.

In test.py, we provide a random agent. You can modify it to implement your agent. You should install a numpy package additionally to run the code.

```
from collections import defaultdict
import argparse
import random
import numpy as np
from environment import TreasureCube

# you need to implement your agent based on one RL algorithm
class RandomAgent(object):
    def __init__(self):
        self.action_space = ['left','right','forward','backward','up','down'] # in TreasureCube
        self.Q = defaultdict(lambda: np.zeros(len(self.action_space)))

    def take_action(self, state):
        action = random.choice(self.action_space)
        return action

    # implement your train/update function to update self.V or self.Q
    # you should pass arguments to the train function
    def train(self, state, action, next_state, reward):
        pass
```

Besides, in test.py, we implement a test function. You should replace the random agent with your agent in line 3.

```
1 def test_corridor(max_episode, max_step):
2     env = TreasureCorridor(max_step=max_step)
3     agent = RandomAgent()
4 
5     for epsisode_num in range(0, max_episode):
6         state = env.reset()
7         terminate = False
8         t = 0
9         episode_reward = 0
10        while not terminate:
11            action = agent.take_action(state)
12            reward, terminate, next_state = env.step(action)
13            episode_reward += reward
14            # env.render()
15            # print(f'step: {t}, action: {action}, reward: {reward}')
16            t += 1
17            agent.train(state, action, next_state, reward)
18            state = next_state
19        print(f'epsisode: {epsisode_num}, total_steps: {t} episode reward: {episode_reward}')
```

If you use Q-learning, you can use the parameters: discount factor $\gamma = 0.99$, learning rate $\alpha = 0.5$, exploration rate $\epsilon = 0.01$.

You can run the following code to generate output and test your agent.

```
python test.py --max_episode 500 --max_step 500
```

4 Submission and Evaluation

We have provided the environment and examples code. You can modify it and finish your own code based on it.

Submission due: 11:59pm October 18 (Sunday)

Submission files:

- A report of your project, which contains the description of your algorithm, the learning progress (rewards in each episode), the final value table or Q-table, etc
- Your final code implementation

Evaluation criteria (total 100 marks):

- Bug-free: correctly implement the code of your chosen RL algorithm [50 marks]
- Show or plot the learning progress: episode rewards vs. episodes [10 marks]
- Show the final value table or Q-table [15 marks]
- Solution quality of your algorithm [25 marks]