Middle East Technical University | Department of Computer Engineering

CENG462

Artificial Intelligence

Spring 2022-2023

Homework 2

Due: June 20th, 2023

Problem Definition

In this assignment you are going to help the "late-May/early-June" agent find a good policy to make it to the bus stop while avoiding to step on the puddles. You will be given a solution strategy and a **static** environment as an input and you will output the policy your strategy settles on. The solution strategy will either be the **policy iteration** or **Q-learning**.

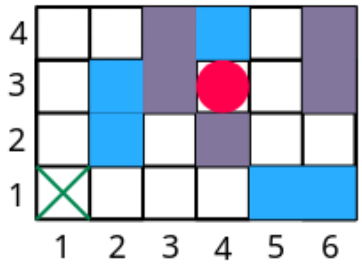

Figure 1: A sample grid, annotated with coordinates. The length of the grid is 6 in the x-dimension, and 4 in the y-dimension. The agent is currently located at $(1, 1)$. The white cells are empty, gray cells are obstacles, blue cells are the puddles, and the red circle marks the bus stop.

Specifications

- You will write a program in Python3. Your program will read the problem from a file. You will output your policy to a file using the convention described below.
- Your implementation of both methods is expected to follow the material covered in the lectures; unless stated otherwise.
- In **policy iteration**;
  - **[IMPORTANT]** The agent's probability of taking a certain action is bound by probabilities. This means that $T(s, a, s')$ is non-trivial. It is given as

$$T(s,a,s') = P(A_a = a') = \begin{cases} 0.8 & \text{if } a' = a \\ 0.09 & \text{if } a' = (a+1) \mod 4 \\ 0.02 & \text{if } a' = (a+2) \mod 4 \\ 0.09 & \text{if } a' = (a+3) \mod 4 \end{cases}$$

    where the random variable A_a stands for the actual action taking place, and actions (Up, Right, Down, Left), referred to by a and a', are enumerated as (0, 1, 2, 3), respectively. For example,

    - if the agent wants to go Up in any state, the chances are it will actually go Up with probability 0.8, go Right/Left with probability 0.09, and go Down with probability 0.02,
    - if the agent wants to go Left in any state, the chances are it will go Left with probability 0.8, go Up/Down with probability 0.09, and go Right with probability 0.02,
    - similarly, for the other two cases.
  - There will be a maximum number of iterations to be taken. You will use it along with the convergence condition $\pi = \pi'$.
  - You can set your initial π_0 to any random policy, I have set it to all zeros (representing the move 'Up') in my implementation.
  - In case of ties, choose the element with lower index as your maximum.
- For **Q-learning**;
  - The learning process will be episodic. An episode will end only when the agent reaches the goal state.
  - In each episode, the agent will start from a random state excluding the locations of the goal, puddles, and obstacles.
  - The agent will retain its Q-memory from earlier episodes.
  - You will use ϵ-greedy approach for action selection mechanism.
- For the given problem,
  - The x-coordinate increases when going right, and the y-coordinate increases when going up. Coordinates of bottom-left is $(1, 1)$.
  - The domain size is not fixed, the length and width of the environment will be specified in the input file.
  - The environment will be static.
  - The agent can move in four directions in the environment: `up`, `right`, `down` and `left`.
  - A state will be represented with its coordinates.
  - If the agent tries to get out of domain boundaries or hits an obstacle, it should stay in the current cell.
  - The environment and the actions are deterministic, so any movement of the agent results in a unique next state.
  - The agent may receive four different rewards from the environment: (i) `regular reward` for walking into an empty cell, (ii) `obstacle reward` for hitting a wall or an obstacle, (iii) `puddle reward` for stepping in a puddle, (iv) `goal reward` for reaching the goal state. The rewards will be specified as **real numbers**.

I/O Structure

Your program will read the input from a file whose path will be given as the first command-line argument. The first line of the input file will specify the method as "P" for policy iteration or "Q" for Q-learning. The rest of file will depend on the requested method, as explained below.

The structure of input which requires policy iteration:

```
P
<number of iterations>
<gamma>                    # discount factor
<epsilon>                  # convergence threshold for Policy-Evaluation
<M> <N>                    # dimensions (M:y-dimension, N:x-dimension)
<j>                        # number of obstacles
<o_1_x> <o_1_y>            # coordinates of obstacle 1
...
<o_j_x> <o_j_y>            # coordinates of obstacle j
<k>                        # number of puddles
```

```
<p_1_x> <p_1_y>              # coordinates of puddle 1
...
<p_k_x> <p_k_y>              # coordinates of puddle k
<g_x> <g_y>                  # coordinates of the goal
<r_d> <r_o> <r_p> <r_g>      # rewards of regular step, hitting an obstacle/wall,
                               penalty for stepping in a puddle, and reaching the goal, ↩
                                   respectively
```

The structure of input which requires Q-learning:

```
Q
<number of episodes>
<alpha>                      # learning rate
<gamma>                      # discount factor
<epsilon>                    # paramater for ϵ-greedy approach
<M> <N>                      # dimensions (M:y-dimension, N:x-dimension)
<j>                          # number of obstacles
<o_1_x> <o_1_y>              # coordinates of obstacle 1
...
<o_j_x> <o_j_y>              # coordinates of obstacle j
<k>                          # number of puddles
<p_1_x> <p_1_y>              # coordinates of puddle 1
...
<p_k_x> <p_k_y>              # coordinates of puddle k
<g_x> <g_y>                  # coordinates of the goal
<r_d> <r_o> <r_p> <r_g>      # rewards of regular step, hitting an obstacle/wall,
                               penalty for stepping in a puddle, and reaching the goal, ↩
                                   respectively
```

As output, your program will create a file, whose path will be given as the second command-line argument. This file should include the policy represented by as following structure, that is, optimal action for each available state except the goal:

```
<s_1_x> <s_1_y> <a_1>        # state 1 and chosen action in this state
                                 (0:'Up', 1:'Right', 2:'Down', 3:'Left')

<s_2_x> <s_2_y> <a_2>        # state 2 and chosen action in this state
...
<s_i_x> <s_i_y> <a_i>        # state i (i = MxN) and chosen action in this state
```

Regulations

1. **Deadline:** The deadline for this homework is strict and no late submissions will be accepted. Submission deadlines are **not** subject to postponement.
2. **Programming Language:** Your code should be written in Python3. Your submission will be tested in inek machines. So make sure that it can be executed on them.
3. **Implementation:** Your code should not import any library other than **numpy** and **queue**.
4. **We have zero tolerance policy for cheating.** People involved in cheating (any kind of code sharing or codes taken from internet included) will be punished according to the university regulations.
5. **Discussion:** You must follow OdtuClass for discussions and possible updates on a daily basis.
6. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. A reasonable timeout will be applied according to the complexity of test cases. This is not about the code efficiency, its only purpose is avoiding infinite loops due to an erroneous code.
7. **Submission:** Submission will be done via OdtuClass. You should upload a **single** python file named `<your_student_id>_hw2.py`, e.g., `e1234567_hw2.py`.