Middle East Technical University | Department of Computer Engineering

CENG 462

Artificial Intelligence

Fall '2024-2025

Homework 3

Due date: 3 January 2025, Friday, 23:55

1 Objectives

This assignment aims to assist you to expand your knowledge on Reinforcement Learning in case of Policy Iteration and SARSA(for state, action, reward, state, action) methods.

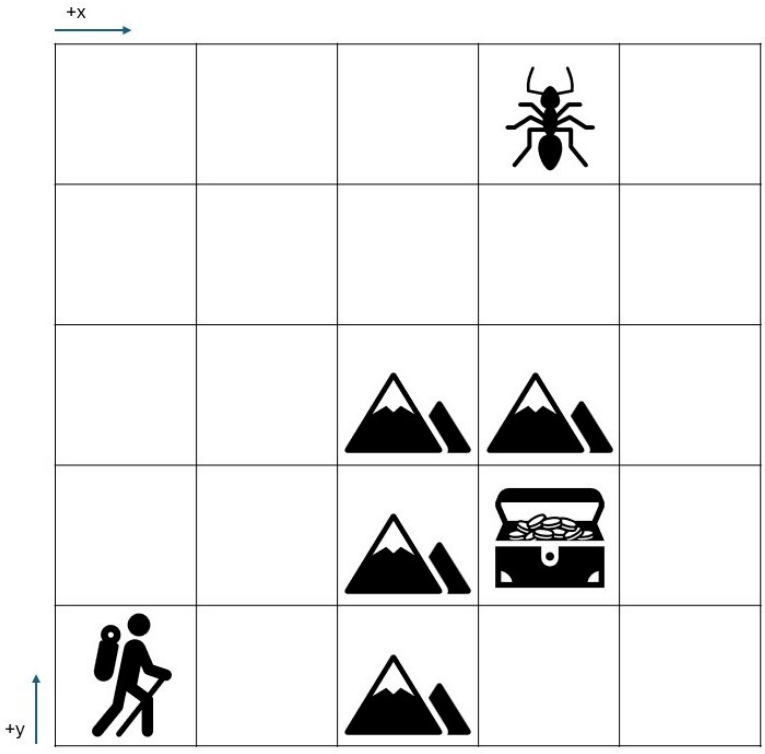

Figure 1: A possible scene from an example problem domain where the adventurer, a pitfall, and the treasure are located at coordinates (1,1), (4,5), and (4,2) respectively, with mountains acting as obstacles scattered throughout the grid.

2 Problem Definition

In this assignment, you will guide alone explorer through a grid-based cavern, where one cell contains an obstacle, another hides a dangerous pitfall, and a precious relic awaits in a distant corner. You will implement either Policy Iteration or SARSA to learn an effective policy. Your goal is to produce a final policy that safely leads the explorer from the start state to the goal while avoiding costly hazards and unnecessary detours.

3 Specifications

You will write a python script which will read the problem configuration from a file and output the policy constructed by the requested method to a file.

In **Policy Iteration**, you alternately perform **Policy Evaluation**, where you compute the value function for a given policy until it converges according to a threshold (θ), and **Policy Improvement**, where you update the policy to be greedy with respect to the computed values, repeating these steps until the old policy and the new policy are equal, indicating that no further improvements are possible.

In **SARSA**:

- 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 goal state and those states which are pitfalls or obstacles.
- The environment will be deterministic, so any movement of the agent should result in only one way, considering the conditions above.
- You will use an ϵ-greedy approach for action selection. At each step, the agent will choose an action with probability ϵ uniformly at random; otherwise, it will choose the action with the highest Q-value.
- For each transition, you will first select an action from the current state using the ϵ-greedy policy, observe the resulting reward and next state, then again select an action from the next state according to the same ϵ-greedy policy. Use both the current and next chosen actions to update the Q-value according to the SARSA update rule.

4 Sample Input/Output

4.1 Policy Iteration Input Explanation

- **Method:** P
- **Convergence Threshold:** θ (For policy evaluation)
- **Discount Factor:** γ
- **Grid Dimensions:** $M \times N$
- **Number of Obstacles:** O
- **Obstacle Coordinates:** $(x_1, y_1), (x_2, y_2), \ldots, (x_O, y_O)$
- **Number of Pitfalls:** P

- **Pitfall Coordinates:** $(x_1, y_1), \ldots, (x_P, y_P)$
- **Goal State:** (x_G, y_G)
- **Rewards:**
  - Default Reward: R_{default}
  - Obstacle Reward: R_{obstacle}
  - Pitfall Reward: R_{pitfall}
  - Goal Reward: R_{goal}

4.1.1 Example Input

```
P
theta
gamma
M N
O
x_1 y_1
x_2 y_2
...
x_O y_O
P
x_1 y_1
x_2 y_2
...
x_P y_P
x_G y_G
R_default R_obstacle R_pitfall R_goal
```

4.2 Sarsa Input explanation

- **Method:** S
- **Number of Episodes:** N_{episodes}
- **Learning Rate:** α
- **Discount Factor:** γ
- **Exploration Rate:** ϵ
- **Grid Dimensions:** $M \times N$
- **Number of Obstacles:** O
- **Obstacle Coordinates:** $(x_1, y_1), (x_2, y_2), \ldots, (x_O, y_O)$
- **Number of Pitfalls:** P
- **Pitfall Coordinates:** $(x_1, y_1), \ldots, (x_P, y_P)$
- **Goal State:** (x_G, y_G)
- **Rewards:**

- Default Reward: R_{default}
- Obstacle Reward: R_{obstacle}
- Pitfall Reward: R_{pitfall}
- Goal Reward: R_{goal}

4.2.1 Example Input

```
S
N_episodes
alpha
gamma
epsilon
M N
O
x_1 y_1
x_2 y_2
...
x_O y_O
P
x_1 y_1
x_2 y_2
...
x_P y_P
x_G y_G
R_default R_obstacle R_pitfall R_goal
```

4.3 Output Explanation

Each line corresponds to a state in the grid:

- **x y:** State coordinates.
- **a:** The action taken at that state, where:
  - **0:** Move North
  - **1:** Move East
  - **2:** Move South
  - **3:** Move West

State (x, y)	**Action** a
(x_1, y_1)	a_1
(x_2, y_2)	a_2
$\vdots$	$\vdots$
(x_M, y_N)	a_{MN}

4.3.1 Example Output

```
x1 y1 a1
x2 y2 a2
x3 y3 a3
...
xM yN aN
```

5 Regulations

1. **Programming Language:** You must code your program in Python 3. Make sure that your code can be executed as below:

   ```
   python3 e1234567_hw3.py input.txt output.txt
   ```

2. **Allowed Libraries:** You are allowed to use **sys**, **collections** and **random** modules from standard python library. You can ask for additional packages.

3. **Late Submission:** No late submissions are allowed.

4. **Cheating: Zero tolerance policy for cheating.** Sharing code or using external sources without proper acknowledgment will result in penalties as per university regulations.

5. **Evaluation:** Your program will be evaluated automatically using a "white-box" technique. A reasonable timeout will be applied according to the complexity of test cases in order to avoid innite loops due to an erroneous code.

6. **Submission:** Submit your assignment via OdtuClass. Upload a **single** Python file named <**your-student-id**>_**hw3.py** (e.g., `e1234567_hw3.py`).