Lab 2: Environmental Simulators

1 Introduction

Environmental simulators play a crucial role in RL by providing a safe and controlled environment where agents can learn and explore without real-world consequences. These simulators allow RL algorithms to iterate through vast numbers of trials, enabling the agent to practice and improve its decision-making process through trial and error. In a simulator, we can model complex environments, introduce variability, and manipulate conditions to better understand how an agent learns. The ability to run experiments at scale and speed that would be impractical in the real world makes simulators indispensable for developing and fine-tuning RL algorithms, especially when real-world testing is costly, dangerous, or slow. In this lab we will practice implementing a relatively simple environment and simulating an agent making random decisions.

1.1 The Gymnasium Standard

Gymnasium, AKA OpenAI Gym, is a popular toolkit used in RL for developing and comparing RL algorithms. Gymnasium provides a relatively simple, standardized interface to a wide range of environments, from simple tasks like balancing a pole on a cart to more complex video games and robotics simulations. Many RL packages are structured to work with Gymnasium. For this reason, we will aim to build environments using a gymnasium-like structure.

In general, a gymnasium environment is specified as a python class that records the current state, specifies the state and action space structure, and has at least two functions: a **reset** function and a **step** function.

- In RL there are two kinds of environments: 1) continuing environments that essentially run forever without stopping and 2) episodic environments that begin, run for a finite number of turns, and then stop. The **reset** function is used to return episodic environments to a starting state so that a new episode may begin.
- The **step** function is typically used to evolve the environment. It takes in an agent action and returns an update to the current state, a generated reward, and an indicator for whether or not the episode has ended.

2 Implement two custom environments

In this lab, you will take the following environments and implement them in python using a gymnasium-like structure.

```python
class Corridor_environment:
    """Example of a custom env in which you have to walk down a corridor.
    You can configure the length of the corridor via the env config."""

    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    def __init__(self, config):
        self.end_pos = config['corridor_length']    #set the length of the corridor
        self.cur_pos = 0    #set the current position to start at 0
        self.action_space = (-1,1)    #set the nature of the action space
        self.state_space = list(range(config['corridor_length']))   #the state space -- just the position in
        #self.reset(seed=8)    #We call this to reset the environmental seed and position

    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    def reset(self, seed=None):
        np.random.seed(seed)    #reset the environmental seed
        self.cur_pos = 0    #reset the position of the agent
        return [self.cur_pos]

    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    def print_state(self):
        print('The current state is:',self.cur_pos)

    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    def step(self, action):
        if action == -1 and self.cur_pos > 0:    #if the action is to take a step back and the position is n
            self.cur_pos -= 1    #move one space back
        elif action == 1:    #if the action is to take a step forward...
            self.cur_pos += 1    #move one step forward
        done = (self.cur_pos >= self.end_pos)    #assess whether the agent has reached the end point
        if done:    #if the agent has reached the end of the corridor...
            reward = np.random.uniform()*2    #assign a relatively large reward
        else:    #otherwise...
            reward = -0.1    #assign a small reward
        return {'state':self.cur_pos
                ,'r':reward
                ,'done':done}
```

Figure 1: An example of a custom environment with a gymnasium-like structure.

Environment 1 – a racetrack. Consider driving a race car around a turn. You want to go as fast as possible, but not so fast as to run off the track. The racetrack is a grid – it begins with a vertical section 10 cells wide and 30 cells high, followed by a right turn into a horizontal section 15 cells wide and 10 cells tall. The starting line is the row of cells at the bottom of the first section. The finish line is the column of cells at the right of the horizontal section. The car begins at any cell on the starting line and each turn it is at one of the grid positions. The velocity is also discrete, a number of grid cells moved horizontally and vertically per time step. The actions are increments to the velocity components. Each may be changed by $+1$, -1, or 0 in each step, for a total of nine (3×3) actions. The vertical velocity component is restricted to be nonnegative. Both velocity components are restricted to be less than 5, and they cannot both be zero except at the starting line. Each episode begins in one of the randomly selected start states with both velocity components zero and ends when the car crosses the finish line. If the car attempts to go through the track boundary (anywhere by the finish line) it crashes and the episode ends.

Environment 2 – a chase. Consider a chase on a square field with each side of length n. In the field is a predator and a player. The player begins in cell position $(0, 0)$. The predator begins in a randomly generated cell position. At position (n, n) is the player's base. Unlike the racetrack example, the player and predator can be at any location in the field, i.e. any real number between 0 and n. The velocity of the player is continuous, a distance moved horizontally and vertically per time step. The actions are increments to the velocity components. Each may be changed by $+1$, -1, or 0 in each step, for a total of nine (3×3) actions. In this case they are stochastic; given a horizontal velocity at time t of $V_H^{(t)}$:

$$V_H^{(t+1)} = \begin{cases} V_H^{(t)} + U([0.5, 1.5]) & \text{if } a_H^{(t)} = 1 \\ V_H^{(t)} & \text{if } a_H^{(t)} = 0 \\ V_H^{(t)} + U([-0.5, -1.5]) & \text{if } a_H^{(t)} = -1. \end{cases}$$

The vertical velocity is the same. Both velocity components are restricted to be no more than 5. Each turn, randomly choose either the predator or the player to move first. The predator moves a distance of no more than 4 directly toward the player. The player moves according to their current velocity (but must stay within the field of play). If the predator lands on the cell occupied by the player then the player is caught and the episode ends. If the player is within a distance of 0.5 of their base without being caught then they escape and the episode ends.

1. Implement these two environments using gymnasium-like python classes. Make sure to include all the relevant elements of said class: the ability to track the current state; the **reset** function; and the **step** function.

2. Simulate 1000 episodes with randomly generated agent actions.

Lab Assignment Report

Write up your analysis in a separate report of no more than 2 pages. Your report should answer the following questions:

1. For each environment, plot a heatmap of the agent's position. How often does it succeed at its task with random actions (crossing the finish line for the track, reaching base for the chase)?

2. Describe how you would seek to reward the agent in each environment. Can you identify any challenges the agent might face in learning given your suggested approach to rewards?

3. Describe how you would conceptualize the state in each environment – what information do you think the agent would need to manage its task?

4. Imagine a version of each environment in which each episode had randomly generated obstacles. What effect do you think this would have on your agent's success rate? Would it complicate learning?

3 Submission Instructions

- Write up the answers to the questions in a short word document; aim for no more than 2 pages of text and include all graphics generated. Add footnotes identifying which sentence addresses which questions. Write in complete sentences organized into paragraphs – your goal is to explain what you've done and what you've learned to your audience (me!). Include the appropriate plots you've generated as mentioned above. Convert this to pdf and submit it. Submit your .ipynb file and an .html of it as well.

- The grading rubric for this assignment will be available in Canvas.

- NO OTHER SUBMISSION TYPES WILL BE ACCEPTED.

- You are welcome to use generative AI as you code up your solution. If you do this you must include all prompts used as an appendix in your written report or provide a shareable link to them. Any uncited use of generative AI (i.e. prompts not provided) will be considered plagiarism. You may not use generative AI to write your report.