```python
import numpy as np
import copy
import math

ACTION_MEANING = {
    0: "UP",
    1: "RIGHT",
    2: "LEFT",
    3: "DOWN",
}

SPACE_MEANING = {
    1: "ROAD",
    0: "BARRIER",
    -1: "GOAL",
}


class MazeEnv:

    def __init__(self, start=[6,3], goals=[[1, 8]]):
        """Deterministic Maze Environment"""

        self.m_size = 10
        self.reward = 10
        self.num_actions = 4
        self.num_states = self.m_size * self.m_size

        self.map = np.ones((self.m_size, self.m_size))
        self.map[3, 4:9] = 0
        self.map[4:8, 4] = 0
        self.map[5, 2:4] = 0

        for goal in goals:
            self.map[goal[0], goal[1]] = -1

        self.start = start
        self.goals = goals
        self.obs = self.start

    def step(self, a):
        """ Perform a action on the environment

            Args:
                a (int): action integer

            Returns:
                obs (list): observation list
                reward (int): reward for such action
                done (int): whether the goal is reached
        """
        done, reward = False, 0.0
        next_obs = copy.copy(self.obs)

        if a == 0:
            next_obs[0] = next_obs[0] - 1
        elif a == 1:
            next_obs[1] = next_obs[1] + 1
        elif a == 2:
            next_obs[1] = next_obs[1] - 1
        elif a == 3:
            next_obs[0] = next_obs[0] + 1
        else:
            raise Exception("Action is Not Valid")

        if self.is_valid_obs(next_obs):
            self.obs = next_obs

        if self.map[self.obs[0], self.obs[1]] == -1:
            reward = self.reward
            done = True

        state = self.get_state_from_coords(self.obs[0], self.obs[1])

        return state, reward, done

    def is_valid_obs(self, obs):
        """ Check whether the observation is valid

            Args:
                obs (list): observation [x, y]

            Returns:
                is_valid (bool)
        """

        if obs[0] >= self.m_size or obs[0] < 0:
            return False

        if obs[1] >= self.m_size or obs[1] < 0:
            return False

        if self.map[obs[0], obs[1]] == 0:
            return False

        return True

    @property
    def _get_obs(self):
        """ Get current observation
        """
        return self.obs

    @property
    def _get_state(self):
        """ Get current observation
        """
        return self.get_state_from_coords(self.obs[0], self.obs[1])

    @property
    def _get_start_state(self):
        """ Get the start state
        """
        return self.get_state_from_coords(self.start[0], self.start[1])

    @property
    def _get_goal_state(self):
        """ Get the start state
        """
        goals = []
        for goal in self.goals:
            goals.append(self.get_state_from_coords(goal[0], goal[1]))
        return goals

    def reset(self):
        """ Reset the observation into starting point
        """
        self.obs = self.start
        state = self.get_state_from_coords(self.obs[0], self.obs[1])
        return state

    def get_state_from_coords(self, row, col):
        state = row * self.m_size + col
        return state

    def get_coords_from_state(self, state):
        row = math.floor(state/self.m_size)
        col = state % self.m_size
        return row, col


class ProbabilisticMazeEnv(MazeEnv):
    """ (Q2.3) Hints: you can refer the implementation in MazeEnv
    """

    def __init__(self, goals=[[2, 8]], p_random=0.05):
        """ Probabilistic Maze Environment

            Args:
                goals (list): list of goals coordinates
                p_random (float): random action rate
        """
        MazeEnv.__init__(self, goals=goals)
        self.p_random=p_random


    def step(self, a):
        """ Perform a action on the environment

            Args:
                a (int): action integer

            Returns:
                obs (list): observation list
                reward (int): reward for such action
                done (int): whether the goal is reached
        """
        done, reward = False, 0.0
        next_obs = copy.copy(self.obs)

        if np.random.uniform(0, 1) <= self.p_random:
            a = np.random.randint(0,4,1)

        if a == 0:
            next_obs[0] = next_obs[0] - 1
        elif a == 1:
            next_obs[1] = next_obs[1] + 1
        elif a == 2:
            next_obs[1] = next_obs[1] - 1
        elif a == 3:
            next_obs[0] = next_obs[0] + 1
        else:
            raise Exception("Action is Not Valid")

        if self.is_valid_obs(next_obs):
            self.obs = next_obs

        if self.map[self.obs[0], self.obs[1]] == -1:
            reward = self.reward
            done = True

        state = self.get_state_from_coords(self.obs[0], self.obs[1])

        return state, reward, done

if __name__=="__main__":
    p = ProbabilisticMazeEnv(p_random=0.5)
    print(p.start)
```