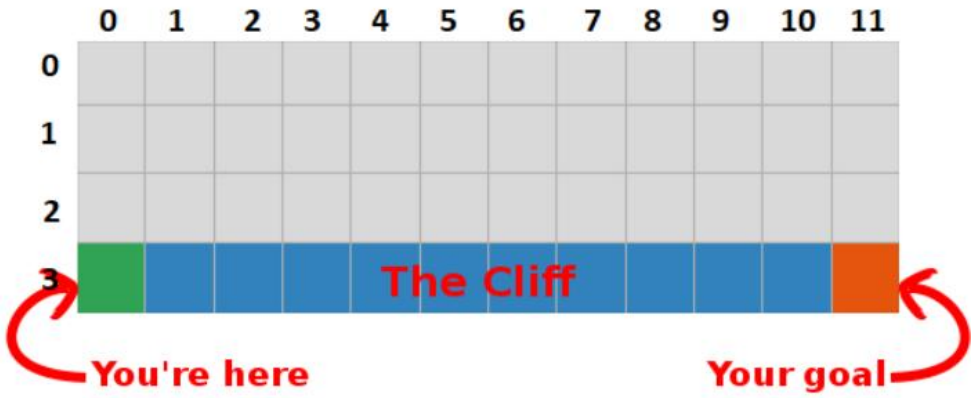
Project1 Part_B

1. Dynamic Programming

Cliff Walk is a classic reinforcement learning environment that requires an agent to start from a starting point, avoid the cliff, and ultimately reach the target location. There is a 4 × 12 grid world, where each grid represents a state. The goal is the state in the lower right corner. The intelligent agent can take four actions in each state: up, down, left, and right. If the intelligent agent touches the boundary wall after taking action, the state will not change, otherwise it will reach the next state accordingly. There is a cliff in the environment, and when the intelligent agent falls into the cliff or reaches the target state, it will end its action and return to the starting point, which means that falling into the cliff or reaching the target state is the termination state. The reward for each step taken by the intelligent agent is -1, and the reward for falling off a cliff is -100.



1.1 Policy Iteration

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $temp \leftarrow v(s)$
         $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \left[r(s, \pi(s), s') + \gamma v(s') \right]$
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $temp \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s, a) \left[r(s, a, s') + \gamma v(s') \right]$
      If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

Q1: Implement the policy iteration algorithm in the Cliff Walk environment in the dynamic_programming.py.

1.2 Value Iteration

Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $temp \leftarrow v(s)$
    $v(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
    $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

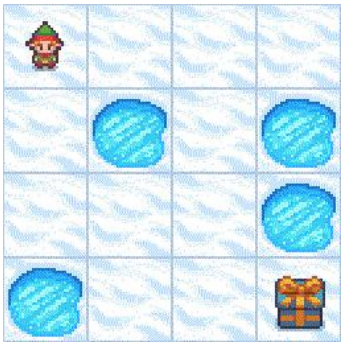
Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v(s')\right]$

Q2: Implement the value iteration algorithm in the Cliff Walk environment in the dynamic_programming.py. Then run the dynamic_programming.py.

Q3. Compare your Policy iteration with the Value iteration to obtain the final policy. Are there any differences between them? Please explain the reason.

Q4. Compare the convergence speed of Policy iteration and Value iteration. Which one can converge faster? Please explain the reason.

2. Model Based Learning



Frozen Lake is an environment in the OpenAI Gym library. The OpenAI Gym library contains many well-known environments, such as Atari and MuJoCo, and supports us to customize our own environment. Frozen Lake is very similar to Cliff Walking, as it is also a grid world where each square represents a state. The starting state of the agent is in the upper left corner, the target state is in the lower right corner, and there are several Holes in the map. In each state, four actions can be taken: up, down, left, and right. Due to the intelligent agent walking on ice, there is a certain probability of sliding to other nearby states each time it walks, and the walking will end prematurely when it reaches the ice cave or target state.
(You can learn basic information about the environment through the gym documentation: *https://www.gymlibrary.dev/environments/toy_text/frozen_lake*)

Q5: In this task, you are provided with a file named All_ episodes.npy, which stores 1000 collected trajectories. Estimate the state transition matrix P and reward function R of the environment by using the 1000 trajectories, then you can directly use ValueIteration in dynamic_programing.py to obtain the optimal policy. Implement your algorithm in the model_based_learning.py, and then run this file. (In this task, you default to not knowing the state transition matrix and reward function of Frozen Lake. So you are not allowed to directly access this information in the environment, such as using env.P)