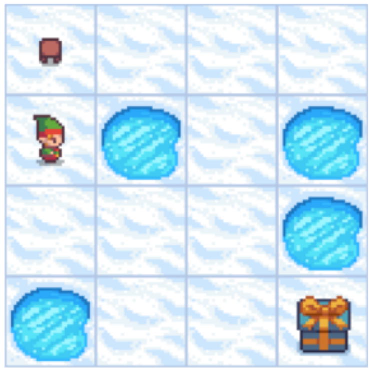
Program 2

[MDP + RL]

Develop a Python program that learns (via Q-learning) to move from the start to the goal on a frozen lake without falling into a hole.



Your agent's only **state observation** is their location, encoded as a single integer (current_row * ncols + current_col, counting from 0). So the start state is 0 and the goal state in a 4x4 lake is 15. You are always certain of your current location.

There are four **actions** encoded as integers:

0: Move left
1: Move down
2: Move right
3: Move up

Due to the icy surface, an action is only successful some of the time. The other results are a slip to the left or right of the intended direction (there is no chance of sliding backward). This probability can be set by you (we assume 0.8 to start)

The simulator provides a reward of 1 for reaching the goal, and a reward of 0 otherwise (this can also be customized, see below). A trajectory ends if you reach the goal or fall into an ice pit (or take over 100 moves in the 4x4 environment without reaching goal or falling into a pit).

Details

FrozenLake is an environment from the Gymnasium fork of OpenAI's AI Gym library for testing reinforcement learning algorithms.
https://gymnasium.farama.org/environments/toy_text/frozen_lake/

Some python sample code is provided that runs the simulation with random actions, printing out the transitions taken during a trajectory as they occur. To run that, you'll need to install gymnasium, and possibly some of the box2d libraries

```
pip install gymnasium
pip install swig
```

By default, the system generates a specific 4x4 map.

```
env = gym.make("FrozenLake-v1", render_mode="human",
               map_name="4x4",
               is_slippery=True,
                success_rate=0.8,
               reward_schedule=(10, -10, 0)
)
```

To generate a random 4x4 map, do

```
env = gym.make("FrozenLake-v1", render_mode="human",
               desc=generate_random_map(size=4),
               is_slippery=True,
                success_rate=0.8,
               reward_schedule=(10, -10, 0)
)
```

You can also turn off the graphics (`render_mode=None`) and generate a larger 8x8 map (either `map_name="8x8"` for a fixed map or `desc=generate_random_map(size=8)` for a random map).

You can also turn off the slippery actions (`is_slippery=False`), so that movement is deterministic, not stochastic.

The following is thanks to Tyler Nauta, who took this class Spring 2025 and fixed up this environment to allow success and reward customization (and had his changes accepted into the main codebase)!!!

`success_rate` dictates the probability p that the agent goes in the intended direction; with probability $(1-p)/2$ it goes 90 degrees to the left or right of that intention (similar to the robot example in class).

`reward_schedule` dictates the payoff for reaching the goal/present, falling into a hole, and just living on the ice.

Submission

Your submission should consist of two files: `qlearn.py` and a written report answering the questions below.

A. [50 points] implement a q-learning agent for this environment. You should be able to handle the 4x4 and 8x8, both deterministic (`is_slippery=False`) and stochastic versions. Your code should work for a random environment, not just the two built-in examples. We are assuming p=¾ and rewards 10 for goal, -10 pit, 0 otherwise.

B. [10 points] What approach did you use for tuning exploration vs. exploitation?

C. [15 points] On average, how long (how many runs) does it take to learn a **deterministic** 4x4 and 8x8 environment (you should report average and standard deviation)?

D. [15 points] On average, how long (how many runs) does it take to learn a **stochastic** 4x4 and 8x8 environment with p=¾ ? (you should report average and standard deviation)?

E. [10 points] How did you determine that you have "learned" a stochastic environment in part D?

F. EXTRA CREDIT [10 points]:
   (i) Compare two different exploration methods (e.g. epsilon-greedy, state counting, Boltzman exploration, etc.)

   (ii) What is the effect of changing the probability of correct action? Does the agent learn more quickly when p is larger?

   (iii) What is the effect of changing the reward structure? Does the agent learn more efficiently with higher or lower rewards? What about a (small) negative living reward?