Final project

Heilmeier Questions in the context of the deterministic Frozen Lake problem explained in simple, non-jargon terms:

1. What are you trying to do?

I'm trying to teach a computer how to find the best path across a frozen lake. The lake is like a grid; the computer starts in the top-left corner and must reach the bottom-right corner without falling through weak ice patches. The goal is to get there in the safest and fastest way possible.

2. How is it done today, and what are the limits of current practice?

Computers use methods like trial and error or simple planning to figure out how to move on this lake. Sometimes, these methods require a lot of time or are ineffective when the situation changes, or the lake is large. Also, if there's any randomness (like slipping), these basic methods can struggle to find a good solution.

3. What is new in your approach, and why do you think it will be successful?

I'm using a more precise version of the lake, where every step is predictable, and the ice doesn't randomly break. This makes it easier for the computer to plan and find the best route. Because there is no chance of slipping, the computer can plan more effectively and complete the task faster and more reliably.

4. Who cares? If you are successful, what difference will it make?

People who design innovative systems, such as robots or automated delivery drones, care deeply. This work could help them design more effective path-planning systems that operate efficiently in predictable environments. It's also valuable for education and teaching the basics of AI decision-making and planning.

5. What are the risks?

The main risk is that my method only works well when the lake is predictable and straightforward. If I later want to handle more complex or random situations, I might need to use a different method.

6. What is the cost?

Since this computer simulation does not require special equipment, the cost is very low, only time and computing resources. It can be performed on a standard laptop.

7. How long will it take?

If everything goes well, setting it up and achieving results could take a few days to a couple of weeks, depending on how complex I want to make the lake and how extensive the testing I want to conduct.

8. What are the mid-term and final "exams" to check for success?

Mid-term: Check if the computer is learning to reach the goal without falling into holes.

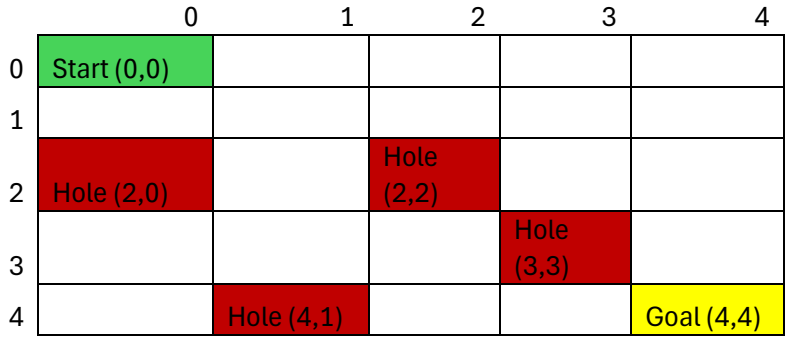
Final exam: See if the computer always chooses the fastest and safest path across any given lake layout, without making mistakes.

The FrozenLake Problem

The "deterministic" FrozenLake is a toy problem from the so called "grid world" category of problems. In this problem the agent lives in a square grid and can move in 4 directions, "up", "down", "left" and "right". The agent always starts in the top-left position and its goal is to reach the bottom right position on the grid (see image below).

Part 1: Create the Frozen Lake

1. Using Python, create a 5x5 grid sized Frozen Lake, with a start state at the top left corner and a goal state at the bottom right corner.
2. Place four holes at the following grid positions in the Frozen Lake. **(2,0)**, **(4,1)**, **(2,2)**, **(3,3)**
3. The reward for reaching the goal state is **+10.0**. The reward for falling into a hole is **-5.0** (because you die!) and the rewards for each transition to a non-terminal state is **-1.0**.
4. The episode ends if the agent falls into a hole or reaches the goal state.
5. The actions are **"up"**, **"down"**, **"left"** and **"right"**.




```
import numpy as np


class FrozenLake:
  def __init__(self):
    self.size = 5
    self.start_pos = (0, 0)
    self.goal_pos = (4, 4)
    self.holes = {(2, 0), (4, 1), (2, 2), (3, 3)}
    self.actions = ['up', 'down', 'left', 'right']
    self.reset()
```

```
def reset(self):
    self.agent_pos = self.start_pos
    return self.agent_pos

def step(self, action):
    x, y = self.agent_pos

    if action == 'up':
        x = max(0, x - 1)
    elif action == 'down':
        x = min(self.size - 1, x + 1)
    elif action == 'left':
        y = max(0, y - 1)
    elif action == 'right':
        y = min(self.size - 1, y + 1)
    else:
        raise ValueError(f"Invalid action: {action}")

    self.agent_pos = (x, y)

    # Check if agent is in a hole
```

```
    if self.agent_pos in self.holes:
        reward = -5.0
        done = True
    # Check if agent reached the goal
    elif self.agent_pos == self.goal_pos:
        reward = +10.0
        done = True
    else:
        reward = -1.0
        done = False

    return self.agent_pos, reward, done

def render(self):
    grid = [['.' for _ in range(self.size)] for _ in range(self.size)]

    for hx, hy in self.holes:
        grid[hx][hy] = 'H'

    gx, gy = self.goal_pos
    grid[gx][gy] = 'G'
```

```
        ax, ay = self.agent_pos
        grid[ax][ay] = 'A'

        for row in grid:
            print(' '.join(row))
        print()

# Example run
env = FrozenLake()
env.render()

state, reward, done = env.step('right')
env.render()
print(f"State: {state}, Reward: {reward}, Done: {done}")
```

Key Features: • 5×5 grid

- Start: `(0, 0)`
- Goal: `(4, 4)` with `+10.0`
- Holes at `(2,0)`, `(4,1)`, `(2,2)`, `(3,3)` with `-5.0`
- Movement: `up`, `down`, `left`, `right`.
- Transition reward: `-1.0`

- Episode ends on hole or goal.
- `render()` method to print grid visually.