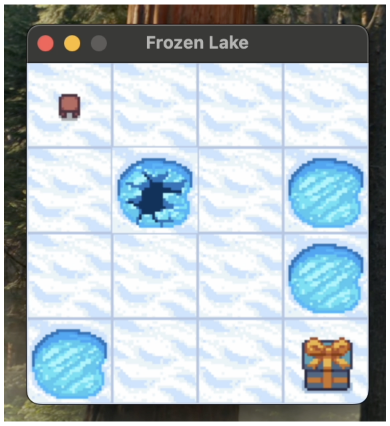
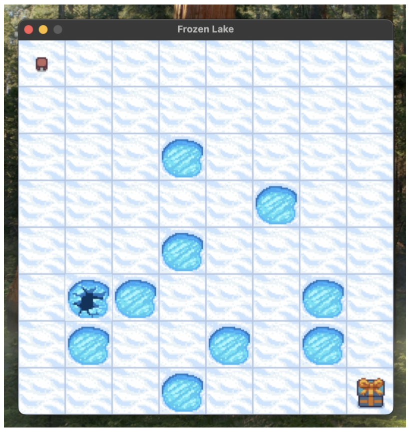
II.a. The main function runs, initializing a 4x4 map and runs a stimulation of the elf's interaction with the environment. The elf falls into a hole.



II.b. To initialize an 8x8 map, I passed the argument for *map_size* while compiling **code_base.py** in the following way: python3 code_base.py -map_size 8x8

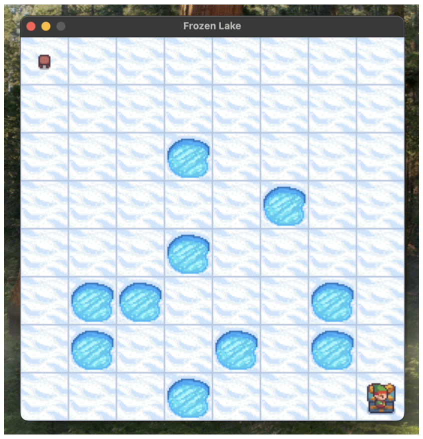


III.a. The initial state is **0**. Its location on the map is **[0,0]**.

III.b. The next state upon taking action 1 is **4**. Its location on the map is **[1,0]**.

IV.a. The agent **didn't** reach the goal and the episode **ended** as it fell into a hole.

IV.b. The agents needs atleast **14** steps to reach the goal.



P.T.O

code_base.py

```
from matplotlib import animation
import matplotlib.pyplot as plt
import gymnasium as gym
from get_args import get_args

def initialize_env(args):
    #################################################
    # Part II Environment Initialization #
    # You can modify map size in get_args.py file#
    env = gym.make(args.env_name, desc=None, map_name=args.map_size,
       render_mode=args.render_mode, is_slippery=False)
    #################################################


    return env

def test_action(args):
    env = initialize_env(args)
    #Run the env
    #################################################
    # Part III Environment Functions#
    state, _ = env.reset()
    print(state)
    for action in range(env.action_space.n):
        state, _ = env.reset()
        next_state, reward, done, _, prob = env.step(action)
        print(f'action:{action}, next state:{next_state},
           reward:{reward}, done:{done}, prob:{prob}')
    #################################################


def test_moves(args):
    env = initialize_env(args)
    total_reward = 0
    num_steps = 0
    #################################################
    # Part IV Environment Test  #
    # You can modify action list in get_args.py file#
    state, _ = env.reset()
    for i in args.actions:
        next_state, reward, done, _, _ = env.step(i)
        print(f'next state:{next_state}, reward:{reward}, done:{done}')
        total_reward = total_reward + reward
        num_steps += 1
    #################################################
    print(f'total reward:{total_reward}, num_steps: {num_steps}')


if __name__ == "__main__":
    # get arguments from get_args.py
    args = get_args()
    print(f'mode: {args.mode}')
    # Part II
    if args.mode == 'initialize_env':
        env = initialize_env(args)
        state, _ = env.reset()
        for t in range(1000):
            action = env.action_space.sample()
            _, _, done, _, _ = env.step(action)
            if done:
                break
        env.close()
    # Part III
    elif args.mode == 'test_action':
        test_action(args)
    # Part IV
    elif args.mode == 'test_moves':
        test_moves(args)
```

---

get_args.py

```
import argparse


def get_args():
    parser = argparse.ArgumentParser()

    # Mode
    parser.add_argument('-mode', type=str, choices=['initialize_env',
       'test_action', 'test_moves'],
                        default='test_moves', # for IV.b
                        help='choose the function to implement')



    # II.b Initializations
    parser.add_argument('-env_name', type=str, default='FrozenLake-v1',
                        help='environment name')
    ###############################
    # Your Code #
    # Please set the parameter 'map_size' to an appropriate value.
    parser.add_argument('-map_size', type=str, choices=['4x4', '8x8'],
                        default='8x8', # for IV.b
                        help='map size')


    #IV.b action list required to create
    ###############################
    # Your Code #
    #Please set the parameter actions's default value to an appropriate
       list
    parser.add_argument('-actions', nargs='+', type=int,
       default=[1,2,1,2,1,2,2,1,1,2,1,1,2,2], help='create an action
       list asked in part IV.b')

    # Render mode
    parser.add_argument(
        "-render_mode",
        "-r",
        type=str,
        help="The render mode for the environment. 'human' opens a
           window to render. 'ansi' does not render anything.",
        choices=["human", "ansi"],
        default="human",
    )

    return parser.parse_args()
```