Dynamic Programming (20 points + 10 bonus points)

In this assignment, we will implement a few dynamic programming algorithms, namely, policy iteration and value iteration and run them on a simple MDP - the Frozen Lake environment.

The sub-routines for these algorithms are present in `vi_and_pi.py` and must be filled out to test your implementation.

The deliverables are located at the end of this notebook and show the point distrbution for each part.

Value iteration is worth 20 points of regular credit and policy iteration is worth 10 points of bonus credit for both sections of this course CS 7643 and CS 4803.

In [1]:

```
%load_ext autoreload
%autoreload 2

import numpy as np
import gym
import time

from IPython.display import clear_output

from lake_envs import *
from vi_and_pi import *

np.set_printoptions(precision=3)

env_d = gym.make("Deterministic-4x4-FrozenLake-v0")
env_s = gym.make("Stochastic-4x4-FrozenLake-v0")
```

```
/home/nicolas/.local/lib/python3.6/site-packages/gym/envs/registration.py:14: PkgResourcesDeprecationWarning: Parameters to load ar
e deprecated.  Call .resolve and .require separately.
  result = entry_point.load(False)
```

Render Mode

The variable `RENDER_ENV` is set `True` by default to allow you to see a rendering of the state of the environment at every time step. However, when you complete this assignment, you must set this to `False` and re-run all blocks of code. This is to prevent excessive amounts of rendered environments from being included in the final PDF.

IMPORTANT: SET `RENDER_ENV` TO FALSE BEFORE SUBMISSION!

In [2]:

```
RENDER_ENV = False
```

Part 1: Value Iteration

For the first part, you will implement the familiar value iteration update from class.

In `vi_and_pi.pi` and complete the `value_iteration` function.

In [3]:

```
##################################################
# Use this space for debugging                   #
# Make sure to delete this code before submission #
##################################################
pass
##################################################
```

Run the cell below to train value iteration and render a single episode of following the policy obtained at the end of value iteration.

You should expect to get an Episode reward of `1.0`.

In [4]:

```
print("\n" + "-"*25 + "\nBeginning Value Iteration\n" + "-"*25)

V_vi, p_vi = value_iteration(env_d.P, env_d.nS, env_d.nA, gamma=0.9, tol=1e-3)
render_single(env_d, p_vi, 100, show_rendering=RENDER_ENV)
```

```
-------------------------
Beginning Value Iteration
-------------------------
Episode reward: 1.000000
```

[BONUS] Part 2: Policy Iteration

This is a bonus question in which you will implement policy iteration. If you do not wish to attempt this bonus quesiton, skip to the next part.

In class, we studied the value iteration update:

$$V_{t+1}(s) \leftarrow \max_a \sum_{s'} p(s' \mid s, a)\Big[r(s, a) + \gamma V_t(s')\Big]$$

This is used to compute the value function V^* corresponding to the optimal policy π^*. We can alternatively compute the value function V^π corresponding to an arbitrary policy π, with a similar update loop:

$$V^\pi_{t+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a)\Big[r(s, a) + \gamma V^\pi_t(s')\Big]$$

On convergence, this will give us V^π, which is the first step of a policy iteration update.

The second step involves policy refinement, which will update the policy to take actions greedily with respect to V^π:

$$\pi_{new} \leftarrow \arg\max_a \left[r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) V^\pi(s')\right]$$

A single update of policy iteration involves the two above steps: (1) policy evaluation (which itself is an inner loop which will converge to V^π and (2) policy refinement. In the first part of assignment, you will implement the functions for policy evaluation, policy improvement (refinement) and policy iteration.

In `vi_and_pi.pi` and complete the `policy_evaluation`, `policy_improvement` and `policy_iteration` functions. Run the blocks below to test your algorithm.

In [5]:

```
####################################################
# Use this space for debugging                     #
# Make sure to delete this code before submission #
####################################################
pass
####################################################
```

In [6]:

```
print("\n" + "-"*25 + "\nBeginning Policy Iteration\n" + "-"*25)

V_pi, p_pi = policy_iteration(env_d.P, env_d.nS, env_d.nA, gamma=0.9, tol=1e-3)
render_single(env_d, p_pi, 100, show_rendering=RENDER_ENV)
```

```
-------------------------
Beginning Policy Iteration
-------------------------
Episode reward: 1.000000
```

Part 3: VI on Stochastic Frozen Lake

Now we will apply our implementation on an MDP where transitions to next states are stochastic. Modify your implementation of value iteration as needed so that policy iteration and value iteration work for stochastic transitions.

In [7]:

```
##################################################
# Use this space for debugging                   #
# Make sure to delete this code before submission #
##################################################
pass
##################################################
```

In [8]:

```
print("\n" + "-"*25 + "\nBeginning Value Iteration\n" + "-"*25)

V_vi, p_vi = value_iteration(env_s.P, env_s.nS, env_s.nA, gamma=0.9, tol=1e-3)
render_single(env_s, p_vi, 100, show_rendering=RENDER_ENV)
```

```
-------------------------
Beginning Value Iteration
-------------------------
Episode reward: 1.000000
```

[BONUS] Part 4: PI on Stochastic Frozen Lake

This is a bonus question to run policy iteration on stochastic frozen lake.

Now we will apply our implementation on an MDP where transitions to next states are stochastic. Modify your implementation of value iteration as needed so that policy iteration and value iteration work for stochastic transitions.

In [9]:

```
##################################################
# Use this space for debugging                   #
# Make sure to delete this code before submission #
##################################################
pass
##################################################
```

In [10]:

```
print("\n" + "-"*25 + "\nBeginning Policy Iteration\n" + "-"*25)

V_pi, p_pi = policy_iteration(env_s.P, env_s.nS, env_s.nA, gamma=0.9, tol=1e-3)
render_single(env_s, p_pi, 100, show_rendering=RENDER_ENV)
```

```
-------------------------
Beginning Policy Iteration
-------------------------
Episode reward: 1.000000
```

Evaluate All Policies

Now, we will first test the value iteration implementation on two kinds of environments - the dererministic FrozenLake and the stochastic FrozenLake. We will also run the same for policy iteration

Deliverable 1 (10 points)

Run value iteration on deterministic FrozenLake. You should get a reward of 1.0 for full credit.

In [11]:

```
print("\nValue Iteration on Deterministic FrozenLake:")
V_vi, p_vi = value_iteration(env_d.P, env_d.nS, env_d.nA, gamma=0.9, tol=1e-3)
evaluate(env_d, p_vi, max_steps=100, max_episodes=2)
```

```
Value Iteration on Deterministic FrozenLake:
> Average reward over 2 episodes:                       1.0
> Percentage of episodes goal reached:                  100%
```

Deliverable 2 (10 points)

Run value iteration on stochastic FrozenLake. Note that this time, running the same policy over multiple episodes will result in different outcomes (final reward) due to stochastic transitions in the environment, and even the optimal policy may not succeed in reaching the goal state 100% of the time.

You should get a reward of 0.7 or higher over 1000 episodes for full credit.

In [12]:

```
print("\nValue Iteration on Stochastic FrozenLake:")
V_vi, p_vi = value_iteration(env_s.P, env_s.nS, env_s.nA, gamma=0.9, tol=1e-3)
evaluate(env_s, p_vi, max_steps=100, max_episodes=1000)
```

```
Value Iteration on Stochastic FrozenLake:
> Average reward over 1000 episodes:                    0.701
> Percentage of episodes goal reached:                  94%
```

Deliverable 3 (5 bonus points)

Run policy iteration on deterministic FrozenLake. You should get a reward of 1.0 for full credit.

In [13]:

```
print("Policy Iteration on Deterministic FrozenLake:")
V_pi, p_pi = policy_iteration(env_d.P, env_d.nS, env_d.nA, gamma=0.9, tol=1e-3)
evaluate(env_d, p_pi, max_steps=100, max_episodes=2)
```

```
Policy Iteration on Deterministic FrozenLake:
> Average reward over 2 episodes:                       1.0
> Percentage of episodes goal reached:                  100%
```

Deliverable 4 (5 bonus points)

Run policy iteration on stochastic FrozenLake.

You should get a reward of 0.7 or higher over 1000 episodes for full credit.

In [14]:

```
print("Policy Iteration on Stochastic FrozenLake:")
V_pi, p_pi = policy_iteration(env_s.P, env_s.nS, env_s.nA, gamma=0.9, tol=1e-3)
evaluate(env_s, p_pi, max_steps=100, max_episodes=1000)
```

```
Policy Iteration on Stochastic FrozenLake:
> Average reward over 1000 episodes:                    0.714
> Percentage of episodes goal reached:                  93%
```

Submission Reminder

PLEASE RE-RUN THE NOTEBOOK WITH `RENDER_ENV` SET TO FALSE BEFORE SUBMISSION!