In [1]:
```
import gym
import time
import numpy as np

from IPython.display import clear_output
```

In [172]:
```
env = gym.make('FrozenLake-v0', is_slippery=True)
env.reset()
env.render()
```

```
SFFF
FHFH
FFFH
HFFG
```

In [173]:
```
print(env.action_space.n)
print(env.observation_space.n)
```

```
4
16
```

In [174]:
```
state_probability = env.env.P

state = 0
action = 0

print("Probability, Next state, Reward, Done for State {} and Action {}: \n {}"
      .format(state, action, state_probability[state][action]))
```

```
Probability, Next state, Reward, Done for State 0 and Action 0:
 [(0.3333333333333333, 0, 0.0, False), (0.3333333333333333, 0, 0.0, False), (0.3333333333333333,
4, 0.0, False)]
```

In [6]:
```
class RandomAgent:
    def __init__(self, env):
        self.action_space = env.action_space
        self.observation_space = env.observation_space

    def choose_action(self):
        return self.action_space.sample()
```

In [189]:
```
class ProAgent:
    def __init__(self, env, discount):
        self.discount = discount
        self.state_probability = env.env.P
        self.num_states = env.observation_space.n
        self.num_actions = env.action_space.n

        self.values = np.zeros(env.observation_space.n)
        self.policy = np.zeros(env.observation_space.n)
        self.best_actions = []

    def value_iteration(self, num_episodes=1000, debug=False):
        for episode in range(num_episodes):
            self.best_actions = []
            if debug:
                print("*" * 50)
                print(f"Episode {episode+1}")
            prev_values = np.copy(self.values)

            for state in range(self.num_states):
                if debug:
                    print("_" * 20)
                    print(f"state {state}")
                Q_value = []

                for action in range(self.num_actions):
                    rewards = []

                    if debug:
                        print(f"action: {action}")

                    for transition_probability, next_state, reward, done in env.env.P[state][action
]:

                        q_sa = transition_probability * (reward + self.discount * prev_values[next_s
tate])

                        rewards.append((q_sa))
                        if debug:
                            print(f"rewards: {rewards}")

                    Q_value.append(np.sum(rewards))
                    if debug:
                        print(f"Q_value: {Q_value}")

                self.values[state] = np.max(Q_value)
                action = np.argmax(Q_value)
                self.best_actions.append(action)

                if debug:
                    print(f"self.values[{state}]: {np.max(Q_value)}")
                    print(f"Best action is {action}")
                if debug:
                    print(f"self.values: {self.values}")
                    print(f"best_actions: {self.best_actions}")

        return self.values


    def choose_action(self, state):
        return self.best_actions[state]
```

In [191]:
```
discount = 0.9

agent = ProAgent(env, discount)

agent.value_iteration()

won = 0
episodes = 100

for episode in range(episodes):
    state = env.reset()

    while True:
        action = agent.choose_action(state)
        state, reward, done, info = env.step(action)
        if done:
            if reward:
                won += 1
            break

print("Winning percentage: {}%".format((won/episodes) * 100))
```