# AI HW 3

Jackson Baker
jab132@uark.edu
011029933

October 2025

# 1 Question I

Q1

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} \left( T(s,a,s') \left( R(s,a,s') + \gamma V_k(s') \right) \right)$$

| | | | |
|---|---|---|---|
| 0.0 | 0.0 | 0.0 | +1.00 |
| 0.0 | | 0.0 | -1.00 |
| 0.0 | 0.0 | 0.0 | 0.0 |

| Actions | | Results |
|---|---|---|
| Up | → | 60% Up, 20% Left, 20% Right |
| Down | → | 60% Down, 20% Left, 20% Right |
| Left | → | 60% Left, 20% Up, 20% Down |
| Right | → | 60% Right, 20% Up, 20% Down |

If action into wall, stay in place

$\gamma = 0.9$

$V_0(S) = 0$

$$V_1(3,3) = \begin{cases} \text{up} : (0.6)\overset{U}{(0+0.9\times 0)} + (0.2)\overset{L}{(0+0.9\times 0)} + (0.2)\overset{R}{(1+0.9\times 0)} = 0.2 \\ \text{down} : (0.6)(0) + (0.2)(1) + (0.2)(0) = 0.2 \\ \text{left} : (0.6)(0) + (0.2)(0) + (0.2)(0) = 0 \\ \text{right} : (0.6)(1) + (0.2)(0) + (0.2)(0) = 0.6 \end{cases} \Bigg\} \; V_1(3,3) = 0.6$$

First Iteration

| | | | |
|---|---|---|---|
| 0.0 | 0.0 | 0.6 ▸ | +1.00 |
| 0.0 | | 0.0 | -1.00 |
| 0.0 | 0.0 | 0.0 | 0.0 |

$$V_2(3,3) : \max_a \text{ is right} : (0.6)(1 + 0.9(0)) + (0.2)(0 + 0.9(0.6)) + (0.2)(0.9(0)) =$$
$$= (0.6)(1.0) + (0.2)(0.9(0.6))$$
$$= 0.6 + 0.2 \times 0.9 \times 0.6 \quad = 0.708$$

$$V_2(2,3) : \max_a \text{ is right} : (0.6)(0 + 0.9\overset{V_1(3,3)}{(0.6)}) + (0.2)(0 + 0.9\overset{V_1(2,3)}{(0)}) + (0.2)(0 + 0.9\overset{V_1(2,3)}{(0)}) = 0.324$$

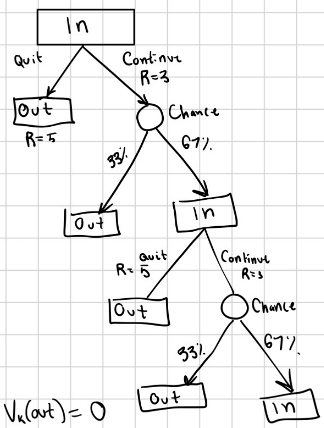
$$V_2(3,2) : \max_a \text{ is up} : (0.6)(0 + 0.9(0.6)) + (0.2(0 + 0.9(0)) + (0.2)(-1 + 0.9(0)) = 0.124$$

Second Iteration

| | | | |
|---|---|---|---|
| 0.0 | 0.324 ▸ | 0.708 ▸ | +1.00 |
| 0.0 | | ▴ 0.124 | -1.00 |
| 0.0 | 0.0 | 0.0 | 0.0 |

## 2 Question II

Q2




States: In , out
Actions : quit , continue

Rewards:
R(in, continue, out) = 3
R(in, continue, in) = 3
R(in, stop, out) = 5

Transition Model:
T(in, quit, out) = 1
T(in, cont., in) = 0.67
T(in, cont, out) = 0.33

$V_k(out) = 0$
$V_0(in) = 0$

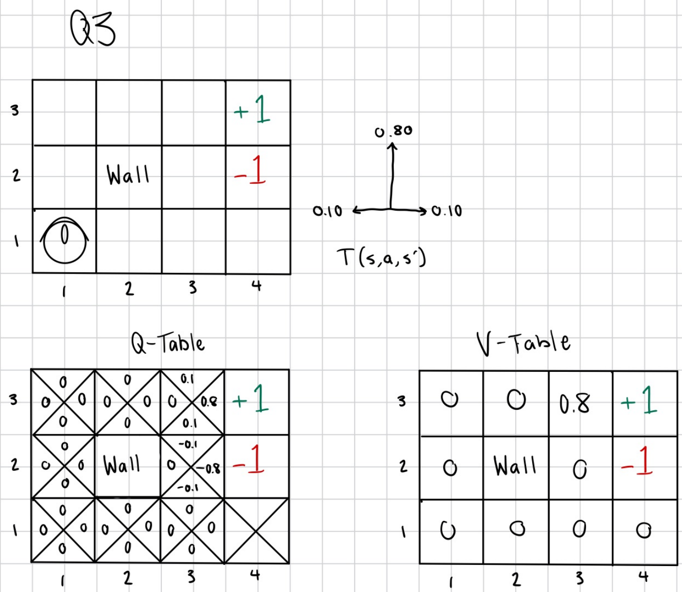
$V_1(in)$ : quit : $1 \cdot (5+0) = 5$
continue : $0.67(3+0) + 0.33(5+0) = 3.66$ } $V_1(in) = 5$

$V_2(in)$ : quit : 5
continue : $0.67(3+5) + 0.33(5+0) = 7.01$ } $V_2(in) = 7.01$

$V_3(in)$ : quit : 5
continue : $0.67(3+7.01) + 0.33(5) = 8.3567$ } $V_3(in) = 8.3567$

The agent may prefer to quit because the immediate reward only given 1 iteration is larger than the reward given to continue. However, as we iterate policies, we see the agent will likely prefer to continue as soon as 2 policy iterations.

# 3 Question III



# 4 Question IV

## 4.1 Python Implementation

```python
import numpy as np


class QLearningAgent:
    def __init__(
        self,
        env,
        episodes=3000,
        alpha=0.9,
        gamma=0.9,
        epsilon=1.0,
        eps_min=0.0,
        eps_decay=0.0001,
        max_steps=200,
        render=False,
        print_every=100
    ):
        self.env = env
        self.episodes = episodes
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.eps_min = eps_min
        self.eps_decay = eps_decay
        self.max_steps = max_steps
        self.render = render
        self.print_every = print_every
        self.q_table = np.zeros((env.observation_space.n, env.action_space.n))

    def train(self, num_episodes=None, render_env=None):
        self.eps_decay = 1 / (num_episodes - (0.1*num_episodes))
        if num_episodes is None:
            num_episodes = self.episodes
        if render_env is None:
            render_env = self.render

        rewards = []
        steps_per_episode = []

        for ep in range(num_episodes):
            result = self.env.reset()
            state = result[0] if isinstance(result, tuple) else result
            done, total_reward, steps = False, 0, 0

            #  initial Q-table and episode for rendering
            if render_env and hasattr(self.env, ’render_mode’) and self.env.render_mode == "human":
                self.env.set_episode(ep + 1)
                self.env.set_q(self.q_table)
                self.env.render()

```

```
            while not done and steps < self.max_steps:
                '''
                ACTIONS:
                0 = LEFT
                1 = DOWN
                2 = RIGHT
                3 = UP
                '''
                if np.random.rand() < self.epsilon:
                    action = self.env.action_space.sample()
                else:
                    action = np.argmax(self.q_table[state,:])

                new_state, reward, terminated, truncated, _ = self.env.step(action)
                # Compute whether the episode has ended
                done = terminated or truncated

                # Calculate the TD target
                if not terminated:
                   best_next_action = np.argmax(self.q_table[new_state, :])
                   td_target = reward + self.gamma * self.q_table[new_state, best_next_action]
                else:
                   td_target = reward

                # Update the Q-table
                td_error = td_target - self.q_table[state, action]
                self.q_table[state, action] += self.alpha * td_error

                #  Q-table
                if render_env and hasattr(self.env, 'render_mode') and self.env.render_mode == "human":
                    self.env.set_q(self.q_table)

                    self.env.render()

                state = new_state
                total_reward += reward
                steps += 1

            if self.epsilon > self.eps_min:
                self.epsilon = max(self.epsilon - self.eps_decay, 0)

            rewards.append(total_reward)
            steps_per_episode.append(steps)

            if (ep + 1) % self.print_every == 0:
                avg_reward = np.mean(rewards[-self.print_every:])
                avg_steps = np.mean(steps_per_episode[-self.print_every:])
                success_rate = np.mean(
                    [1 if r > 0 else 0 for r in rewards[-self.print_every:]])
```

```
                print(f"[Q] Episode {ep+1}/{num_episodes} | "
    f"avg_reward({self.print_every})={avg_reward:.2f} | "
                      f"success_rate={success_rate:.2f} | "
                      f"avg_steps={avg_steps:.1f} | "
                      f"eps={self.epsilon:.3f}")

        return self.q_table, rewards, steps_per_episode

    def get_policy(self):
        return np.argmax(self.q_table, axis=1)
```

## 4.2 Results for 4x4

```
[09:55:16] [~/Github/ai-uark/HW3/Q4] [main ✖] ❱❱❱ poetry run python main.py --mode q --map 4x4 --episodes 2000
pygame 2.6.1 (SDL 2.28.4, Python 3.13.5)
Hello from the pygame community. https://www.pygame.org/contribute.html

 Starting FrozenLake 4x4 — Deterministic (non-slippery) Environment

  Running Q-learning...
[Q] Episode 100/2000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=7.3 | eps=0.944
[Q] Episode 200/2000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=8.2 | eps=0.889
[Q] Episode 300/2000 | avg_reward(100)=0.06 | success_rate=0.06 | avg_steps=7.6 | eps=0.833
[Q] Episode 400/2000 | avg_reward(100)=0.12 | success_rate=0.12 | avg_steps=7.9 | eps=0.778
[Q] Episode 500/2000 | avg_reward(100)=0.17 | success_rate=0.17 | avg_steps=7.7 | eps=0.722
[Q] Episode 600/2000 | avg_reward(100)=0.13 | success_rate=0.13 | avg_steps=7.4 | eps=0.667
[Q] Episode 700/2000 | avg_reward(100)=0.26 | success_rate=0.26 | avg_steps=6.9 | eps=0.611
[Q] Episode 800/2000 | avg_reward(100)=0.35 | success_rate=0.35 | avg_steps=6.9 | eps=0.556
[Q] Episode 900/2000 | avg_reward(100)=0.34 | success_rate=0.34 | avg_steps=6.4 | eps=0.500
[Q] Episode 1000/2000 | avg_reward(100)=0.42 | success_rate=0.42 | avg_steps=6.8 | eps=0.444
[Q] Episode 1100/2000 | avg_reward(100)=0.45 | success_rate=0.45 | avg_steps=6.2 | eps=0.389
[Q] Episode 1200/2000 | avg_reward(100)=0.57 | success_rate=0.57 | avg_steps=6.5 | eps=0.333
[Q] Episode 1300/2000 | avg_reward(100)=0.63 | success_rate=0.63 | avg_steps=6.8 | eps=0.278
[Q] Episode 1400/2000 | avg_reward(100)=0.80 | success_rate=0.80 | avg_steps=7.2 | eps=0.222
[Q] Episode 1500/2000 | avg_reward(100)=0.77 | success_rate=0.77 | avg_steps=6.5 | eps=0.167
[Q] Episode 1600/2000 | avg_reward(100)=0.86 | success_rate=0.86 | avg_steps=6.2 | eps=0.111
[Q] Episode 1700/2000 | avg_reward(100)=0.94 | success_rate=0.94 | avg_steps=6.2 | eps=0.056
[Q] Episode 1800/2000 | avg_reward(100)=0.99 | success_rate=0.99 | avg_steps=6.0 | eps=0.000
[Q] Episode 1900/2000 | avg_reward(100)=1.00 | success_rate=1.00 | avg_steps=6.0 | eps=0.000
[Q] Episode 2000/2000 | avg_reward(100)=1.00 | success_rate=1.00 | avg_steps=6.0 | eps=0.000
Success rate (last 100 greedy eval): 1.00
 Congratulations! You've successfully solved FrozenLake!
```

### 4.3 Results for 8x8

```
[09:56:40] [~/Github/ai-uark/HW3/Q4] [main ×] ))) poetry run python main.py --mode q --map 8x8 --episodes 4000 --slippery
pygame 2.6.1 (SDL 2.28.4, Python 3.13.5)
Hello from the pygame community. https://www.pygame.org/contribute.html

 Starting FrozenLake 8x8 — Stochastic (slippery) Environment

  Running Q-learning...
[Q] Episode 100/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=32.9 | eps=0.972
[Q] Episode 200/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=33.8 | eps=0.944
[Q] Episode 300/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=36.9 | eps=0.917
[Q] Episode 400/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=37.8 | eps=0.889
[Q] Episode 500/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=34.3 | eps=0.861
[Q] Episode 600/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=37.3 | eps=0.833
[Q] Episode 700/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=36.0 | eps=0.806
[Q] Episode 800/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=30.5 | eps=0.778
[Q] Episode 900/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=32.2 | eps=0.750
[Q] Episode 1000/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=38.2 | eps=0.722
[Q] Episode 1100/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=34.2 | eps=0.694
[Q] Episode 1200/4000 | avg_reward(100)=0.02 | success_rate=0.02 | avg_steps=36.9 | eps=0.667
[Q] Episode 1300/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=32.1 | eps=0.639
[Q] Episode 1400/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=34.4 | eps=0.611
[Q] Episode 1500/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=36.6 | eps=0.583
[Q] Episode 1600/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=36.1 | eps=0.556
[Q] Episode 1700/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=35.5 | eps=0.528
[Q] Episode 1800/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=38.5 | eps=0.500
[Q] Episode 1900/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=32.8 | eps=0.472
[Q] Episode 2000/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=36.2 | eps=0.444
[Q] Episode 2100/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=33.1 | eps=0.417
[Q] Episode 2200/4000 | avg_reward(100)=0.02 | success_rate=0.02 | avg_steps=38.0 | eps=0.389
[Q] Episode 2300/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=39.9 | eps=0.361
[Q] Episode 2400/4000 | avg_reward(100)=0.00 | success_rate=0.00 | avg_steps=45.1 | eps=0.333
[Q] Episode 2500/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=39.1 | eps=0.306
[Q] Episode 2600/4000 | avg_reward(100)=0.03 | success_rate=0.03 | avg_steps=46.2 | eps=0.278
[Q] Episode 2700/4000 | avg_reward(100)=0.01 | success_rate=0.01 | avg_steps=44.8 | eps=0.250
[Q] Episode 2800/4000 | avg_reward(100)=0.05 | success_rate=0.05 | avg_steps=57.9 | eps=0.222
[Q] Episode 2900/4000 | avg_reward(100)=0.02 | success_rate=0.02 | avg_steps=51.4 | eps=0.194
[Q] Episode 3000/4000 | avg_reward(100)=0.04 | success_rate=0.04 | avg_steps=47.8 | eps=0.167
[Q] Episode 3100/4000 | avg_reward(100)=0.07 | success_rate=0.07 | avg_steps=66.0 | eps=0.139
[Q] Episode 3200/4000 | avg_reward(100)=0.07 | success_rate=0.07 | avg_steps=54.3 | eps=0.111
[Q] Episode 3300/4000 | avg_reward(100)=0.11 | success_rate=0.11 | avg_steps=64.4 | eps=0.083
[Q] Episode 3400/4000 | avg_reward(100)=0.10 | success_rate=0.10 | avg_steps=68.1 | eps=0.056
[Q] Episode 3500/4000 | avg_reward(100)=0.23 | success_rate=0.23 | avg_steps=83.7 | eps=0.028
[Q] Episode 3600/4000 | avg_reward(100)=0.30 | success_rate=0.30 | avg_steps=94.0 | eps=0.000
[Q] Episode 3700/4000 | avg_reward(100)=0.55 | success_rate=0.55 | avg_steps=105.3 | eps=0.000
[Q] Episode 3800/4000 | avg_reward(100)=0.37 | success_rate=0.37 | avg_steps=92.5 | eps=0.000
[Q] Episode 3900/4000 | avg_reward(100)=0.56 | success_rate=0.56 | avg_steps=107.7 | eps=0.000
[Q] Episode 4000/4000 | avg_reward(100)=0.51 | success_rate=0.51 | avg_steps=95.6 | eps=0.000
Success rate (last 100 greedy eval): 0.74
 Congratulations! You've successfully solved FrozenLake!
```

## 4.4 Commentary on results

In the 4x4 deterministic configuration, learning is rapid and achieves a 100 percent success rate. Starting at 0 percent success at episode 100, the agent initially takes random exploratory moves until it discovers the reward. As epsilon decays, random moves decrease while exploitation of learned knowledge increases. By episode 1800, epsilon reaches 0, meaning no further exploration occurs—only optimal moves are executed. This results in a 100 percent success rate because the deterministic environment allows the agent to reliably repeat a working solution every time. The 8x8 configuration with slippery mode presents a different challenge. The possibility of slipping necessitated modifications to the default hyperparameters: alpha (learning rate), gamma (discount factor), and epsilon (exploration rate). The most impactful change was modifying epsilon decay to a linear function with an x-intercept at 10 percent of the total episode count. This approach relies on achieving early successes; as shown in the results, when the agent finds a reward within the first 1000 episodes, it establishes a foundation of successful moves to build upon. With initially high epsilon enabling sufficient exploration, the agent reliably achieves over 50 percent success rate by the final episodes.

Regarding optimality, the 4x4 deterministic policy is excellent, achieving 100 percent success in exactly 6 steps—the shortest possible path. However, the 8x8 stochastic configuration presents greater challenges. Achieving any success above 0 percent heavily depends on discovering rewards early during high-exploration phases. With linear epsilon decay, success rates spike dramatically near the end as the agent increasingly exploits the best moves learned during early random exploration. Despite this improvement, I do not believe the policy is optimal for the 8x8 stochastic configuration, likely due to insufficient training episodes. More episodes could allow the agent to discover safer paths where probability favors success and hole-avoidance is more likely. While no path may guarantee 100 percent success in a stochastic environment, I believe better solutions exist beyond the observed 51-74 percent success rate.

The fundamental difference between deterministic and stochastic learning is convergence reliability. In deterministic problems, convergence is guarantee. Once the agent learns an optimal solution, executing the same action sequence produces identical outcomes and rewards every time. In stochastic problems, convergence is not guaranteed. A path that succeeds in one episode may fail in another due to random state transitions, making it inherently impossible to achieve perfect reliability regardless of policy quality. This uncertainty requires different learning strategies, including potentially extended exploration and acceptance of suboptimal success rates.