HOMEWORK 2 - REINFORCEMENT LEARNING

Temporal Difference (TD) Learning

Discussions

1. The environment here is stochastic, i.e., the outcomes of the actions performed by the agent are not deterministic. Specifically, when the agent selects a direction to move, it moves in the intended direction with probability of 0.75. There is a 0.05 chance to move in the opposite direction and a 0.10 chance to move in both perpendicular directions. This random action is used to give a realistic simulation where an agent has incomplete control over its move, e.g., when there is environmental noise or uncertainty. The reward function was designed to nudge the agent to optimal states. Typically, moving to a goal state is rewarded positively, while other transitions receive smaller or zero rewards. The reward system influences the behavior of the agent by encouraging exploration to rewarding terminal states, while the stochasticity forces the agent to consider paths that are robust to uncertainty.

2. Utilities are random at first, without any pattern. Over the course of training episodes, the heatmap evolves. States close to the reward become high–value and those close to traps become low–value, as shown in the example figure Figure 1.

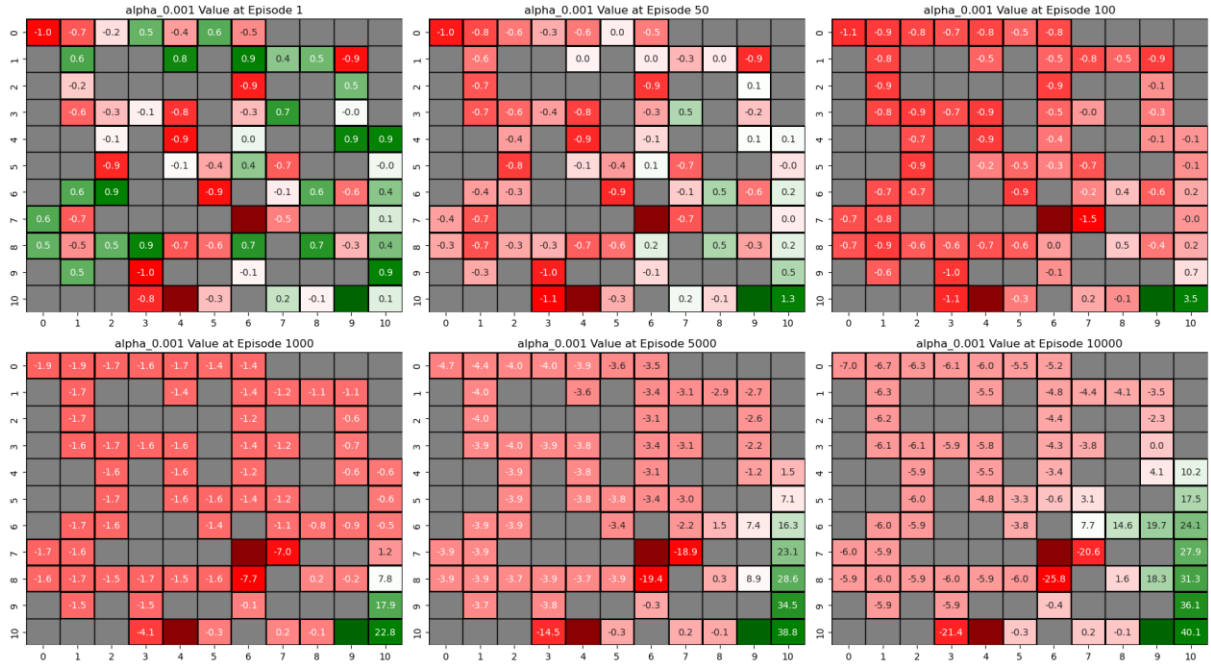


Figure 1: utility value function when alfa = 0.001

3. The utility value function did converge for alfa=1 at episode 2000 and for epsilon=0 immediately. For other values we observe oscillations around some values. However, by looking at the heatmaps (Figure 1) it can be said that utility value function converged between episode 1000 and 5000.

4. The TD(0) learning procedure was highly sensitive to the learning rate (α) and discount factor (γ). When α was placed too low, say at 0.001, learning was very slow (Figure 1) and utility values passed incrementally from episode to episode to converge. Conversely, high α values such as 1.0 caused faster learning (Figure 2) but unstable and even oscillating learning with more substantial changes in utility values. Similarly, small values of γ such as 0.10 led the agent to focus more on short-term rewards (Figure 3) and disallowed the agent to pass information regarding long-term returns. Higher values of γ (such as 0.95) encouraged the agent to look ahead better (Figure 4) in anticipation of future rewards and to generate more informed utility estimates. These are evidenced in both the plots of convergence and learning curves, where certain settings of parameters allowed for faster convergence and improved performance compared to others.

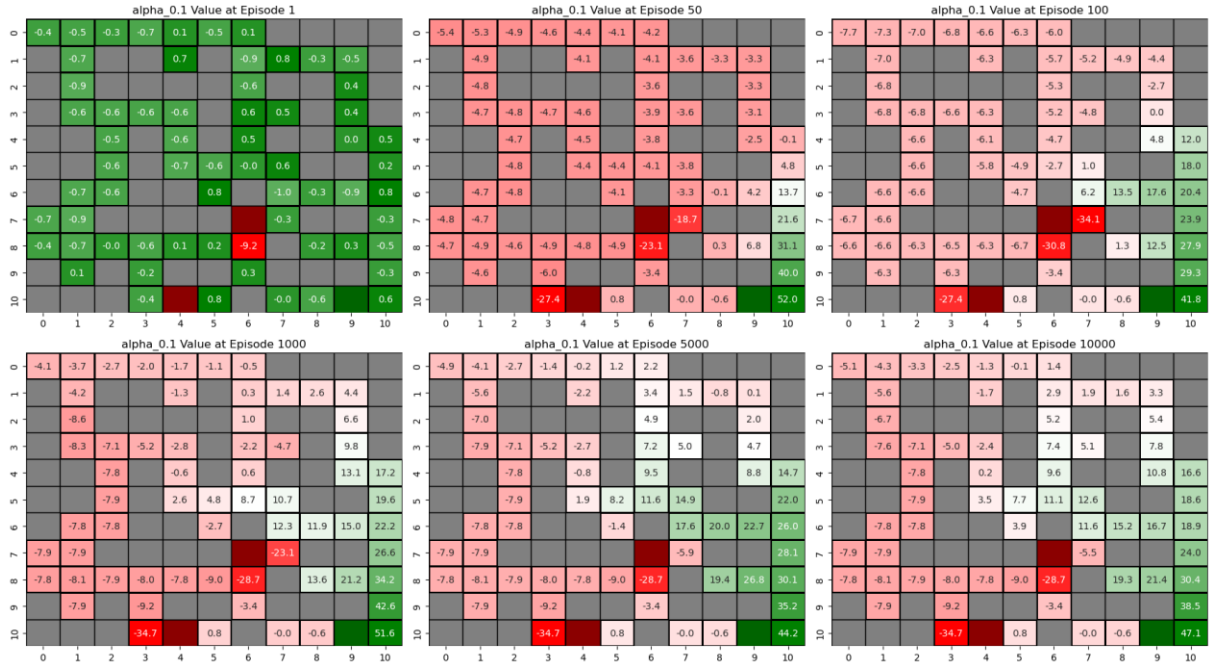


Figure 2: utility value function when alfa = 1

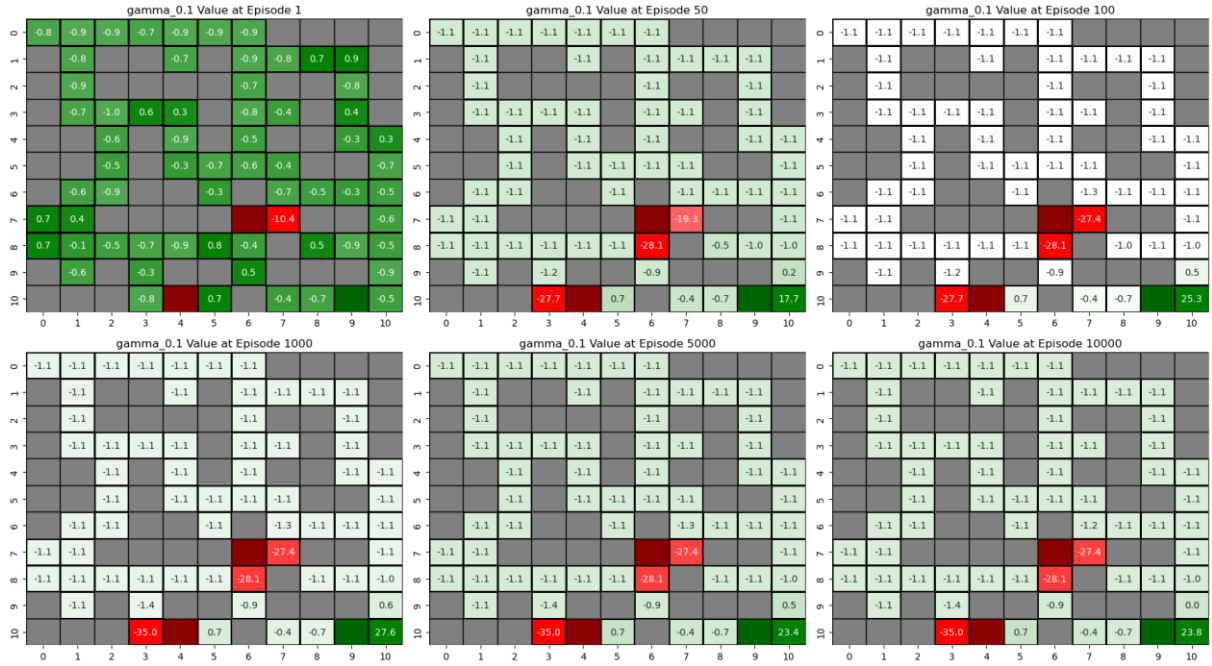


Figure 3: utility value function when gamma = 0.1

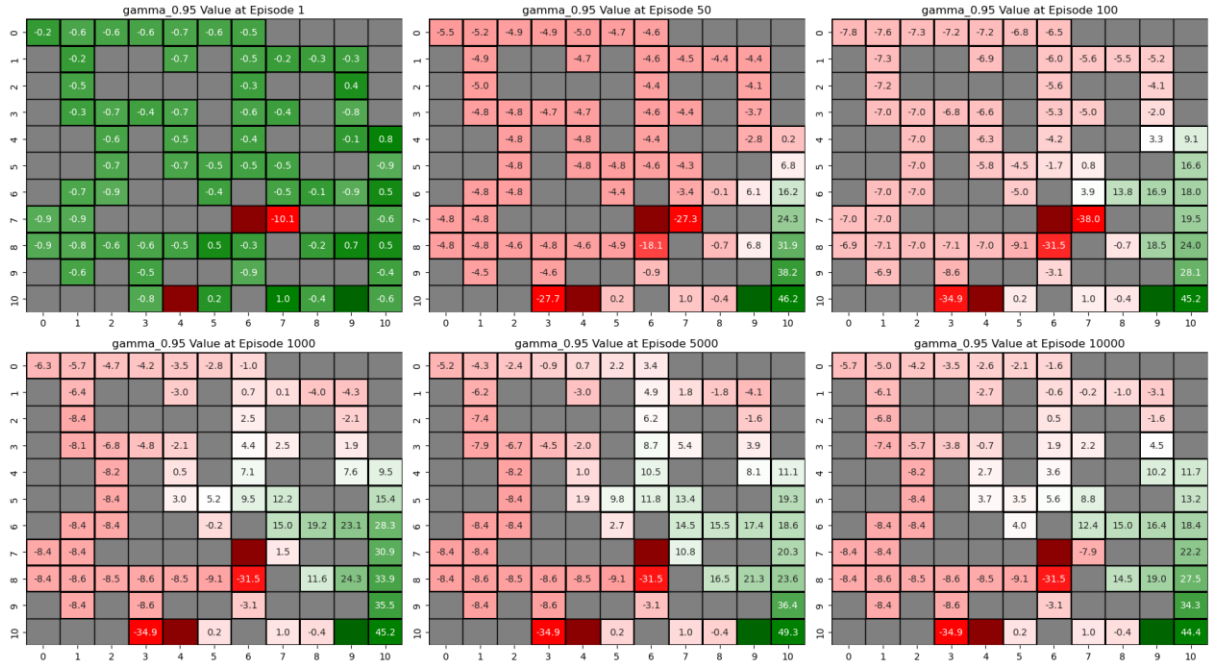


Figure 4: utility value function when gamma = 0.95

5. In implementing TD(0) learning, one of the most significant issues was how to deal with stochastic transitions correctly. Using the correct probability distributions for results of action also required careful monitoring of the transitions so that the transitions represented the desired model. Numerical convergence and stability was also problematic, especially while performing learning rate and discount factor fine-tuning. Debugging of policy and utility updates also required visualizations including heatmaps and convergence plots in order to assure that the algorithm behaved as demanded. These problems were overcome through iterative testing, visualization, and cross-validation of the environment dynamics.

6. The exploration strategy, specifically the ε-greedy strategy, was crucial in the learning process. When ε was 0 (pure exploitation), the agent learned quickly to a suboptimal policy as it did not explore alternative actions. When ε was increased to intermediate values such as 0.2 or 0.5, the agent achieved a finer balance between exploitation and exploration with more powerful utility estimates and policies. However, increasing ε to too high of a value (e.g., 0.8 or 1.0) caused an excessive amount of exploration that impeded convergence as well as general performance. This is evident in the learning curves, where values of ε in the middle produced higher and faster average rewards over time compared to very extreme values.

7. Adjustments in the maze layout or problem setup might make it easier or more difficult for the agent to learn. Adding more obstacles or narrower passageways would make navigation more challenging, requiring the agent to use more sophisticated strategies. Adding a number of goals with different values for rewards might test the agent to prioritize between conflicting goals. Similarly, dynamic elements such as mobile barriers or time-changing rewards might increase the difficulty and test the resilience of the agent. Conversely, maze simplification or rewarding sparsity might facilitate learning being more easy or faster to achieve.

8. The TD(0) learning algorithm can be enhanced by merging it with other reinforcement learning techniques. For instance, incorporating eligibility traces (TD(λ)) would allow the agent to learn multiple states simultaneously, speeding up learning. By merging function approximation methods, e.g., linear approximators or neural networks, the algorithm can be made scalable to bigger or continuous state spaces. In addition, more sophisticated exploration strategies, e.g., decaying ε or optimistic initialization, may enhance exploration efficiency. Lastly, ensemble methods or model-based methods may make the learned policies more robust and reliable.

APPENDIX

```
import numpy as np
import matplotlib.pyplot as plt
import json
import os
from utils import plot_value_function, plot_policy, plot_learning_curves



class MazeEnvironment:
    def __init__(self):
        # Define the maze layout, rewards, action space (up, down, left, right)
        self.maze = np.array([
            [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
            [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1],
            [1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1],
            [1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1],
            [1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0],
            [1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0],
            [1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0],
            [0, 0, 1, 1, 1, 1, 2, 0, 1, 1, 0],
            [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
            [1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0],
            [1, 1, 1, 0, 2, 0, 1, 0, 0, 3, 0]
        ])
        self.start_pos = (0,0) # Start position of the agent
        self.current_pos = self.start_pos
        self.state_penalty = -1
        self.trap_penalty = -100
```

```
        self.goal_reward = 100
        self.actions = {0: (-1, 0), 1: (1, 0), 2: (0, -1), 3: (0, 1)}

    def reset(self):
        self.current_pos = self.start_pos
        return self.current_pos

    def step(self, action):
        move = self.actions[action]
        next_pos = (self.current_pos[0] + move[0], self.current_pos[1] + move[1])
        # Check if next_pos is out of bounds or hits wall (1)
        if (0 <= next_pos[0] < self.maze.shape[0] and
            0 <= next_pos[1] < self.maze.shape[1] and
            self.maze[next_pos] != 1):
            self.current_pos = next_pos
        # Determine reward
        cell_value = self.maze[self.current_pos]
        if cell_value == 0:
            reward = self.state_penalty
        elif cell_value == 2:
            reward = self.trap_penalty
        elif cell_value == 3:
            reward = self.goal_reward
        else:
            reward = self.state_penalty
        return self.current_pos, reward


class MazeTD0(MazeEnvironment): # Inherited from MazeEnvironment
    def __init__(self, maze, alpha=0.1, gamma=0.95, epsilon=0.2, episodes=10000):
```

```
        super().__init__()
        self.maze = maze
        self.alpha = alpha #Learning Rate
        self.gamma = gamma #Discount factor
        self.epsilon = epsilon #Exploration Rate
        self.episodes = episodes
        self.utility = np.random.uniform(low=-1, high=1, size=self.maze.shape)  # FILL HERE, Encourage exploration


    def choose_action(self, state):
        #Explore and Exploit: Choose the best action based on current utility values
        #Discourage invalid moves
        if np.random.rand() < self.epsilon:
            # Exploration: choose random valid action
            valid_actions = []
            for action, move in self.actions.items():
                new_x = state[0] + move[0]
                new_y = state[1] + move[1]
                if 0 <= new_x < self.maze.shape[0] and 0 <= new_y < self.maze.shape[1]:
                    if self.maze[new_x, new_y] != 1:  # not obstacle
                        valid_actions.append(action)
            return np.random.choice(valid_actions)
        else:
            # Exploitation: choose best action
            best_action = None
            best_utility = -np.inf
            for action, move in self.actions.items():
                new_x = state[0] + move[0]
                new_y = state[1] + move[1]
                if 0 <= new_x < self.maze.shape[0] and 0 <= new_y < self.maze.shape[1]:
```

```
            if self.maze[new_x, new_y] != 1:
                util = self.utility[new_x, new_y]
                if util > best_utility:
                    best_utility = util
                    best_action = action
        return best_action

    def update_utility_value(self, current_state, reward, new_state):
        current_value = self.utility[current_state]  # FILL HERE
        new_value = self.utility[new_state]          # FILL HERE
        # TD(0) update formula
        self.utility[current_state] = current_value + self.alpha * (reward + self.gamma * new_value - current_value)  # FILL HERE

    def run_episodes(self):
        for ep in range(self.episodes):
            state = self.reset()
            done = False
            while not done:
                action = self.choose_action(state)
                next_state, reward = self.step(action)
                self.update_utility_value(state, reward, next_state)
                state = next_state
                if self.maze[state] in [2, 3]:  # Trap or Goal = terminal states
                    done = True
        return self.utility
```

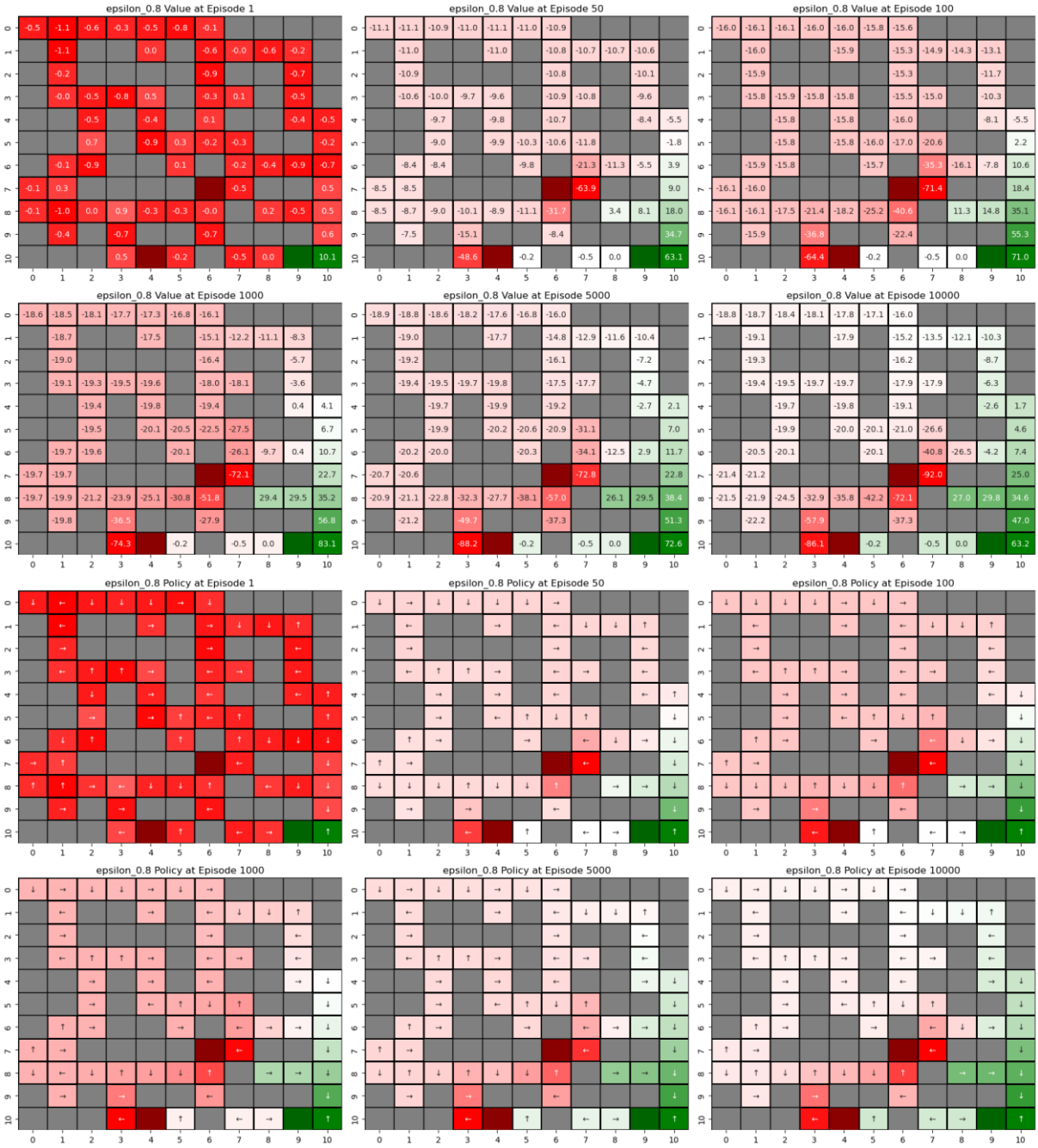
```
# Assuming the Maze and MazeTD0 classes are already defined

# Set up the parameter combinations from Table 1
alpha_values = [0.001, 0.01, 0.1, 0.5, 1.0]
gamma_values = [0.10, 0.25, 0.50, 0.75, 0.95]
epsilon_values = [0, 0.2, 0.5, 0.8, 1.0]



def run_experiments():
    os.makedirs("results", exist_ok=True)
    json_paths = []
    snapshot_eps = [1, 50, 100, 1000, 5000, 10000]

    # Basit moving average
    def moving_average(data, window=100):
        c = np.cumsum(np.insert(data, 0, 0))
        ma = (c[window:] - c[:-window]) / window
        return np.concatenate([np.full(window-1, ma[0]), ma])

    DEFAULT_ALPHA, DEFAULT_GAMMA, DEFAULT_EPSILON = 0.1, 0.95, 0.2
    sweeps = [
        ("alpha", alpha_values, DEFAULT_GAMMA, DEFAULT_EPSILON),
        ("gamma", DEFAULT_ALPHA, gamma_values, DEFAULT_EPSILON),
        ("epsilon", DEFAULT_ALPHA, DEFAULT_GAMMA, epsilon_values),
    ]

    for name, a_vals, g_vals, e_vals in sweeps:
        alphas   = a_vals   if isinstance(a_vals, list) else [a_vals]
        gammas   = g_vals   if isinstance(g_vals, list) else [g_vals]
```

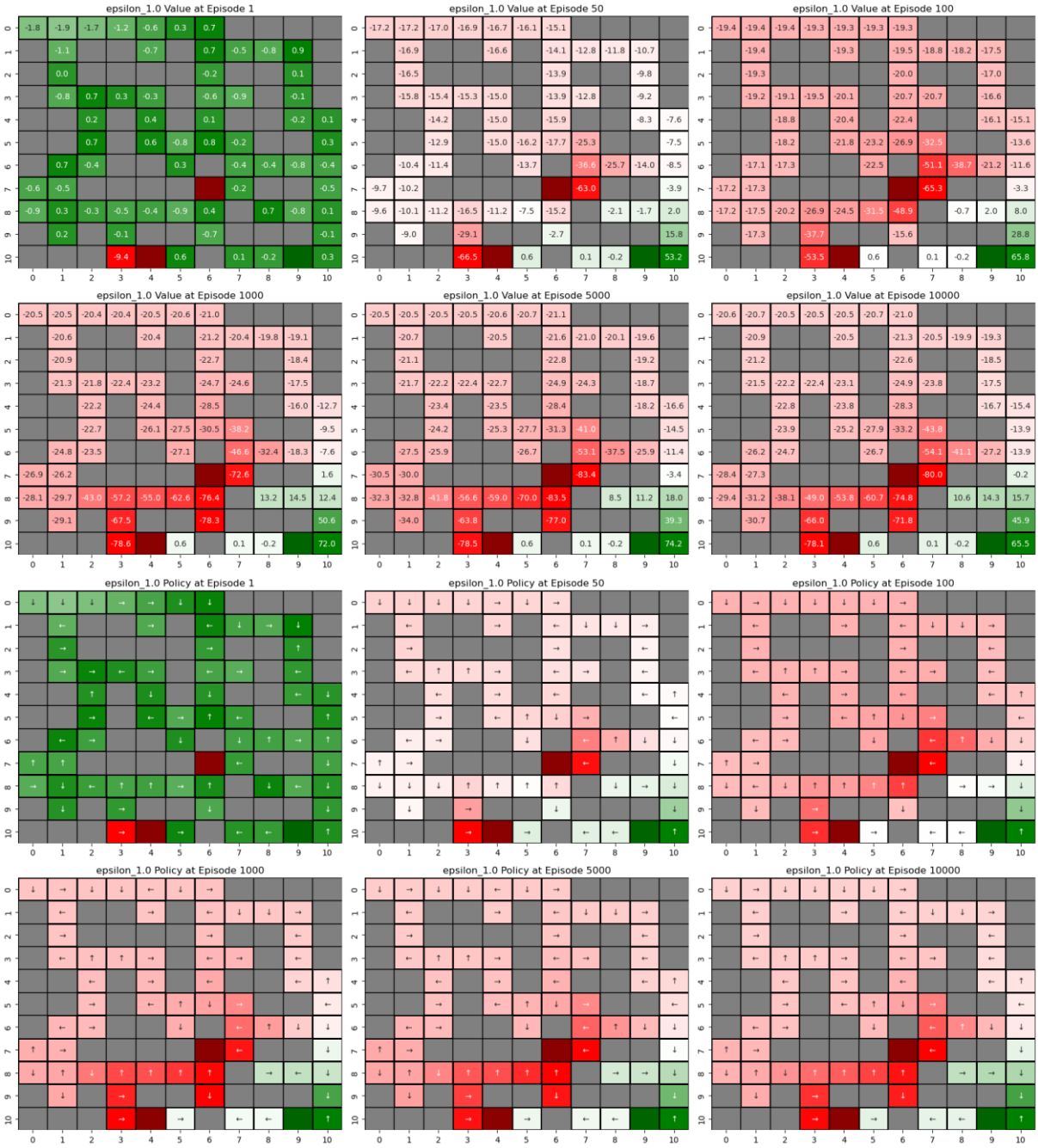
```
epsilons = e_vals   if isinstance(e_vals, list) else [e_vals]


for alpha in alphas:
   for gamma in gammas:
      for epsilon in epsilons:
         label = f"{name}_{alpha if name=='alpha' else gamma if name=='gamma' else epsilon}"
         print(f"\nRunning {label}: α={alpha}, γ={gamma}, ε={epsilon}")


         env   = MazeEnvironment()
         maze  = env.maze
         agent = MazeTD0(maze, alpha=alpha, gamma=gamma, epsilon=epsilon, episodes=10000)


         deltas = []
         episode_rewards = []


         for ep in range(1, agent.episodes + 1):
            prev_util = agent.utility.copy()


            s = agent.reset()
            done = False
            total_r = 0
            step_count = 0
            MAX_STEPS = 500  # episode başına adım limiti


            while not done and step_count < MAX_STEPS:
               a = agent.choose_action(s)
               ns, r = agent.step(a)
               agent.update_utility_value(s, r, ns)
               s = ns
               total_r += r
```

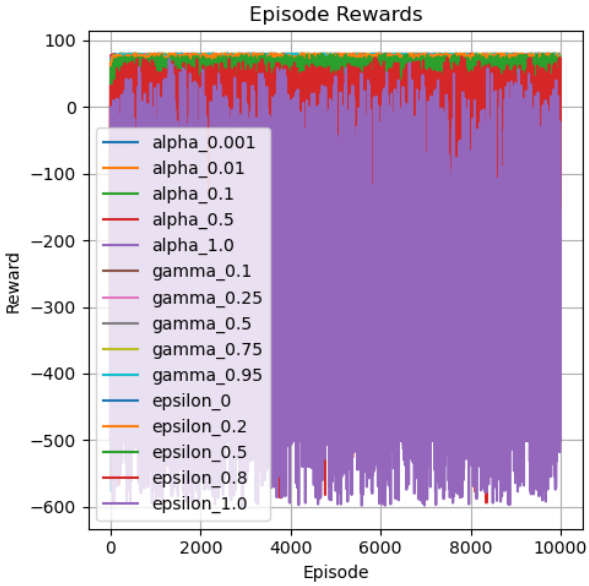
```
            step_count += 1

            if maze[s] in [2, 3]:
                done = True

        episode_rewards.append(total_r)
        deltas.append(np.nansum(np.abs(agent.utility - prev_util)))

        if ep in snapshot_eps:
            U = agent.utility.copy()
            plot_value_function(U, maze)
            plot_policy(U, maze)

    # Convergence plot
    plt.figure()
    plt.plot(deltas)
    plt.title(f"Convergence ({label})")
    plt.xlabel("Episode")
    plt.ylabel("Sum(|ΔU|)")
    plt.grid()
    plt.savefig(f"results/convergence_{label}.png")
    plt.show()

    # JSON kaydet
    avg_scores = moving_average(episode_rewards, 100).tolist()
    data = {
        "episode_rewards": episode_rewards,
        "average_scores": avg_scores
    }
    fout = f"results/{label}.json"
```

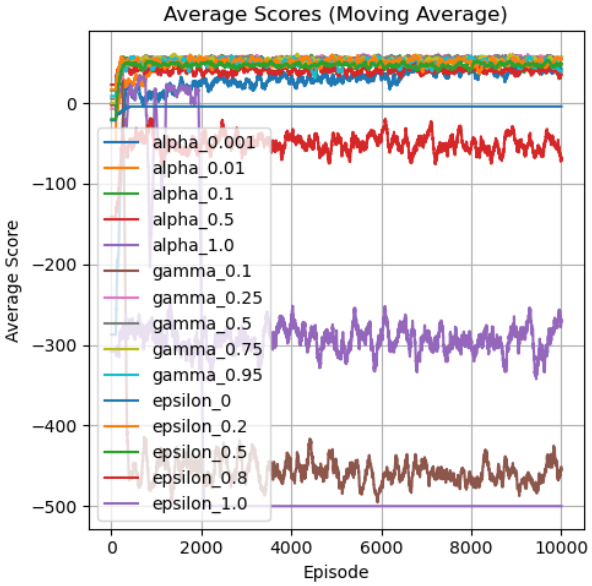
```
            with open(fout, "w") as f:
                json.dump(data, f)
            json_paths.append(fout)

    # Tüm learning-curve’ları tek grafikte ve results klasörüne kaydet
    plot_learning_curves(json_paths, output_file="results/learning_curves.png")

run_experiments()
```

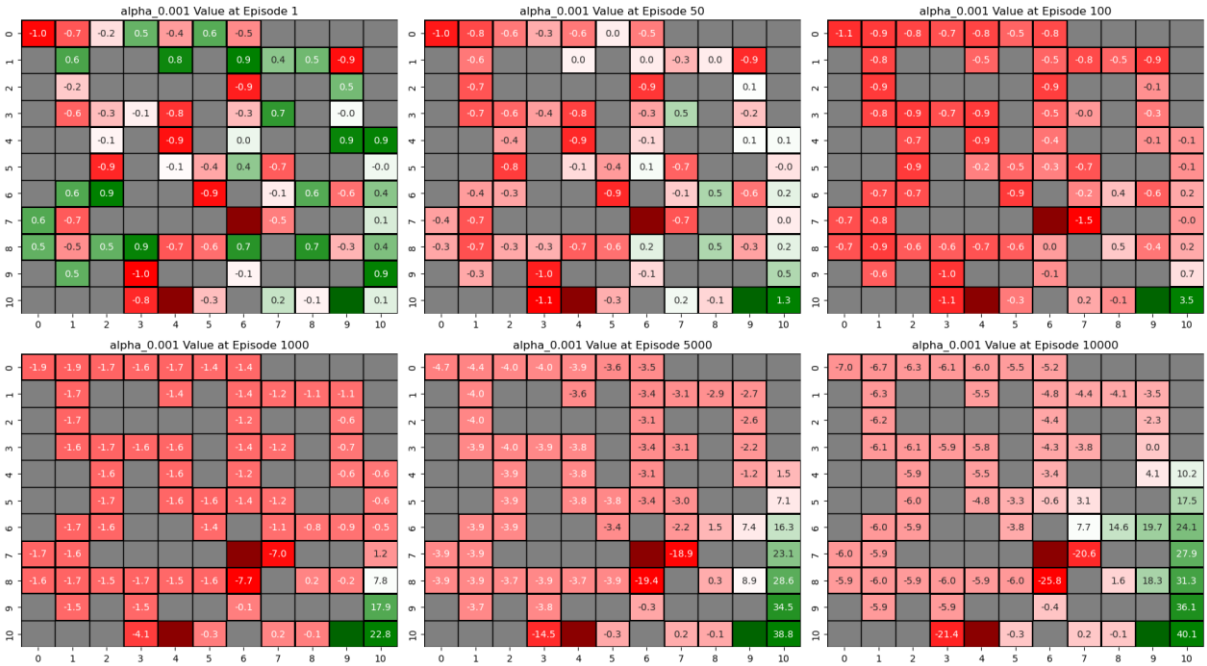
Utility Value Function and Policy Heatmaps:

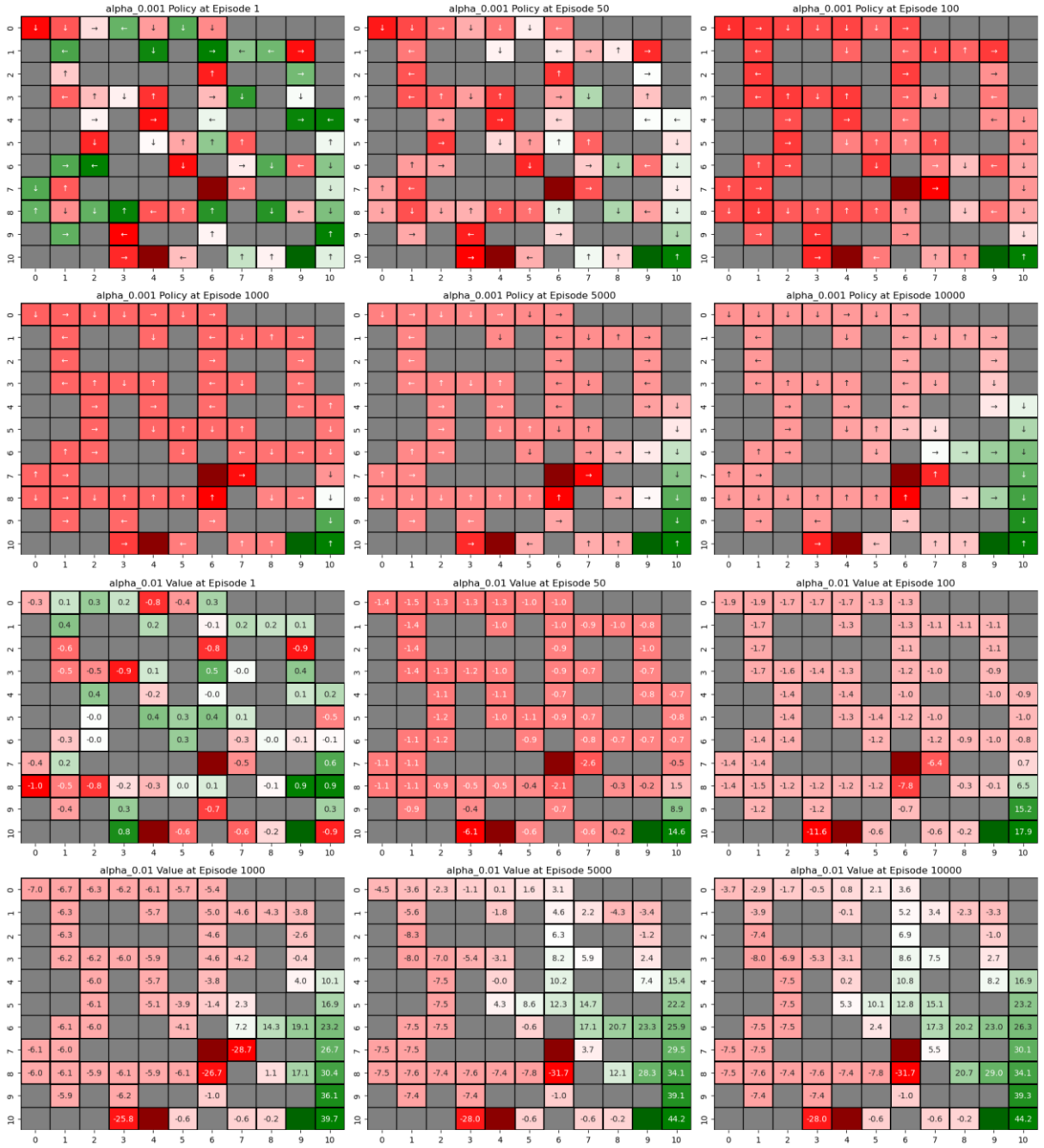

epsilon_0.8 Value at Episode 1
epsilon_0.8 Value at Episode 50
epsilon_0.8 Value at Episode 100
epsilon_0.8 Value at Episode 1000
epsilon_0.8 Value at Episode 5000
epsilon_0.8 Value at Episode 10000
epsilon_0.8 Policy at Episode 1
epsilon_0.8 Policy at Episode 50
epsilon_0.8 Policy at Episode 100
epsilon_0.8 Policy at Episode 1000
epsilon_0.8 Policy at Episode 5000
epsilon_0.8 Policy at Episode 10000

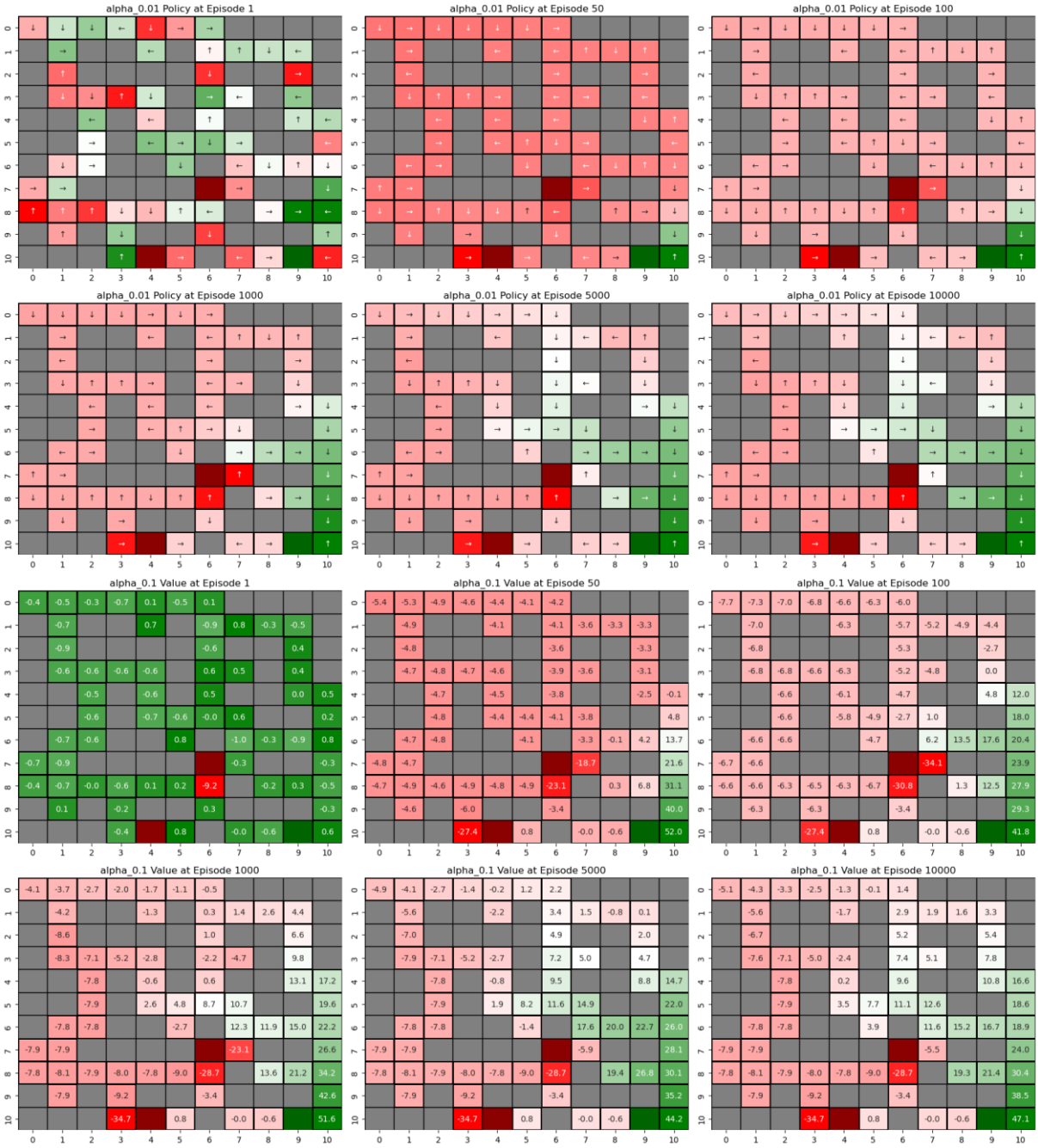

epsilon_1.0 Value at Episode 1
epsilon_1.0 Value at Episode 50
epsilon_1.0 Value at Episode 100
epsilon_1.0 Value at Episode 1000
epsilon_1.0 Value at Episode 5000
epsilon_1.0 Value at Episode 10000
epsilon_1.0 Policy at Episode 1
epsilon_1.0 Policy at Episode 50
epsilon_1.0 Policy at Episode 100
epsilon_1.0 Policy at Episode 1000
epsilon_1.0 Policy at Episode 5000
epsilon_1.0 Policy at Episode 10000

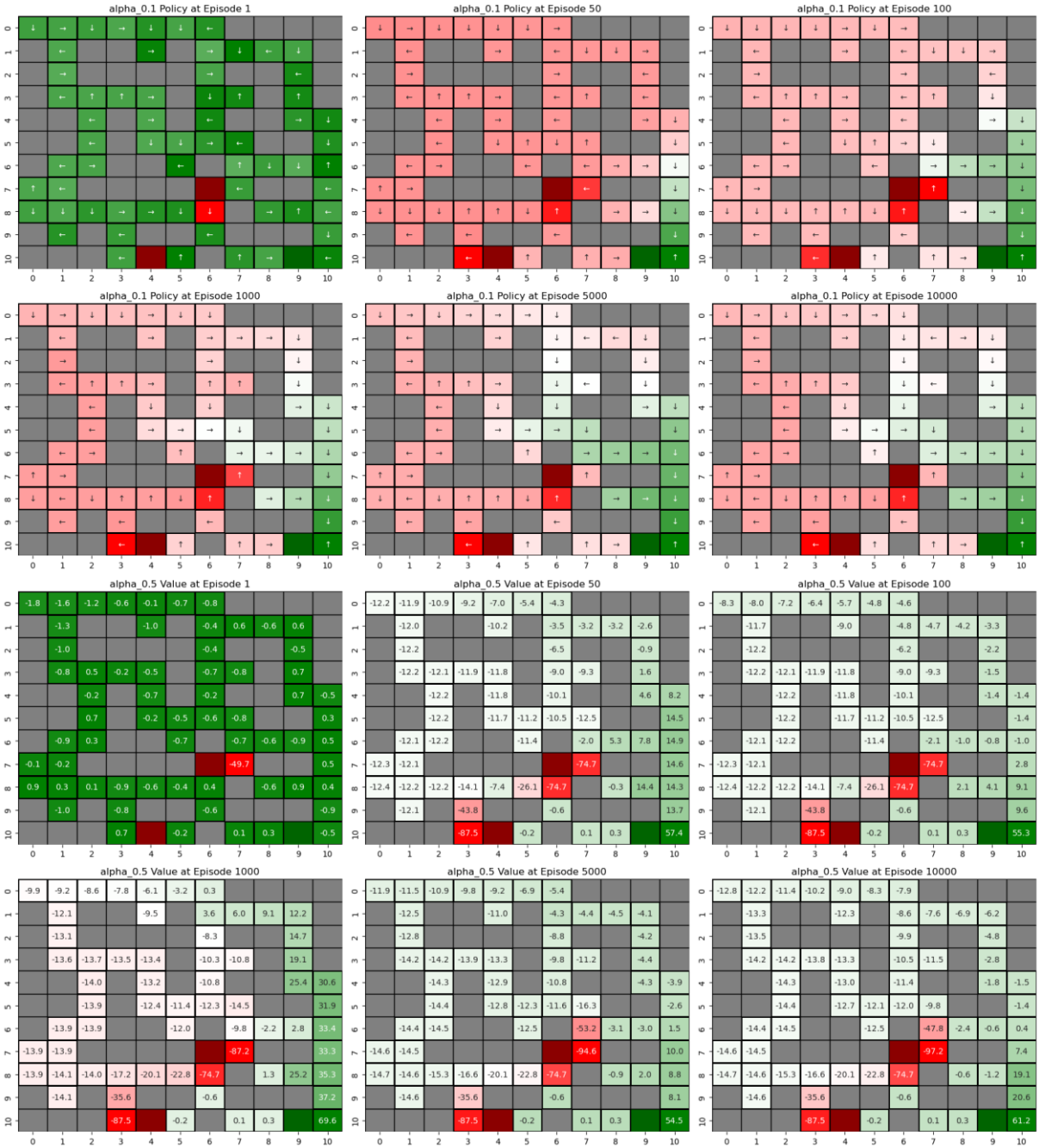

Episode Rewards
Reward
Episode
alpha_0.001
alpha_0.01
alpha_0.1
alpha_0.5
alpha_1.0
gamma_0.1
gamma_0.25
gamma_0.5
gamma_0.75
gamma_0.95
epsilon_0
epsilon_0.2
epsilon_0.5
epsilon_0.8
epsilon_1.0

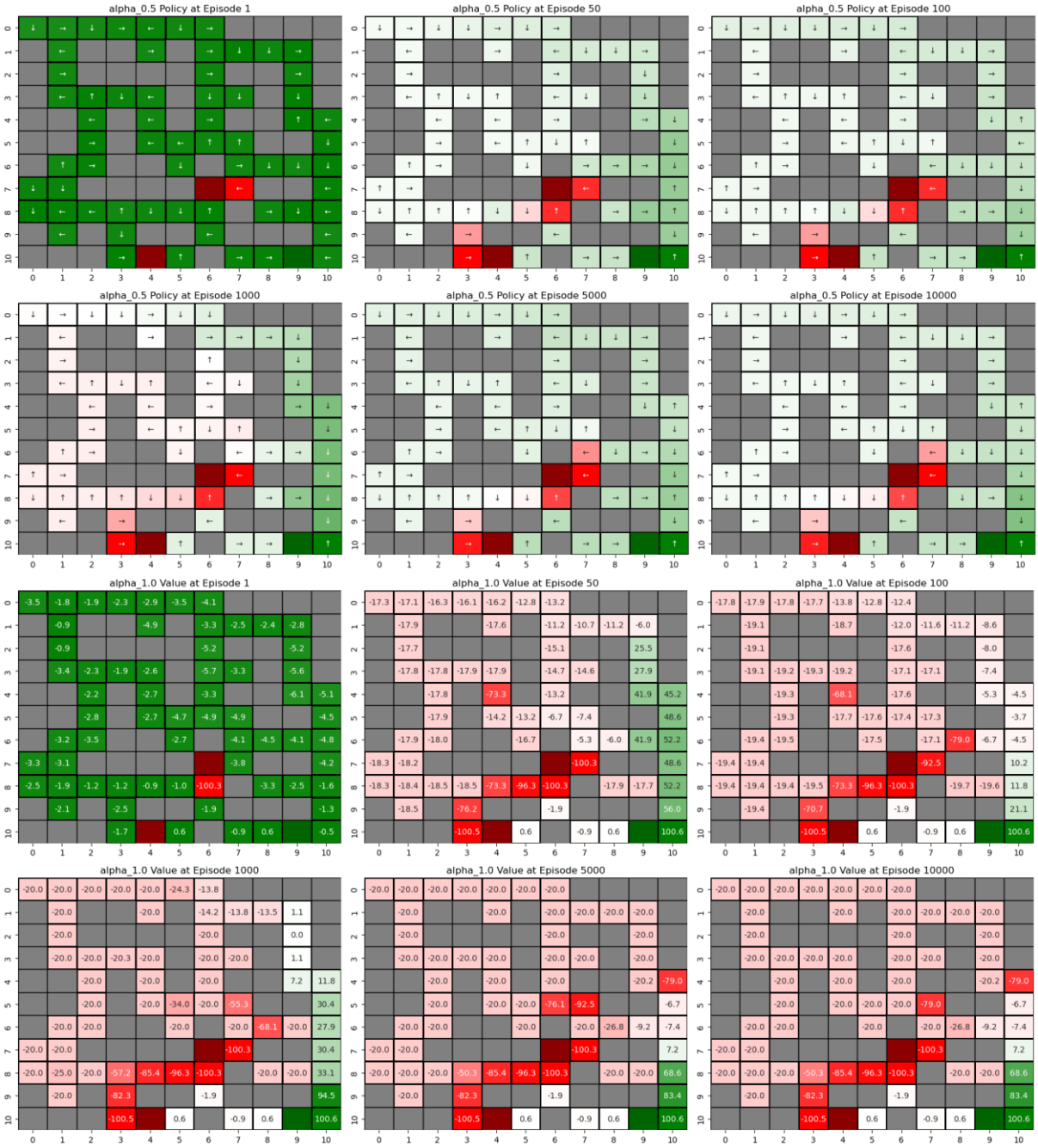

Average Scores (Moving Average)
Average Score
Episode
alpha_0.001
alpha_0.01
alpha_0.1
alpha_0.5
alpha_1.0
gamma_0.1
gamma_0.25
gamma_0.5
gamma_0.75
gamma_0.95
epsilon_0
epsilon_0.2
epsilon_0.5
epsilon_0.8
epsilon_1.0

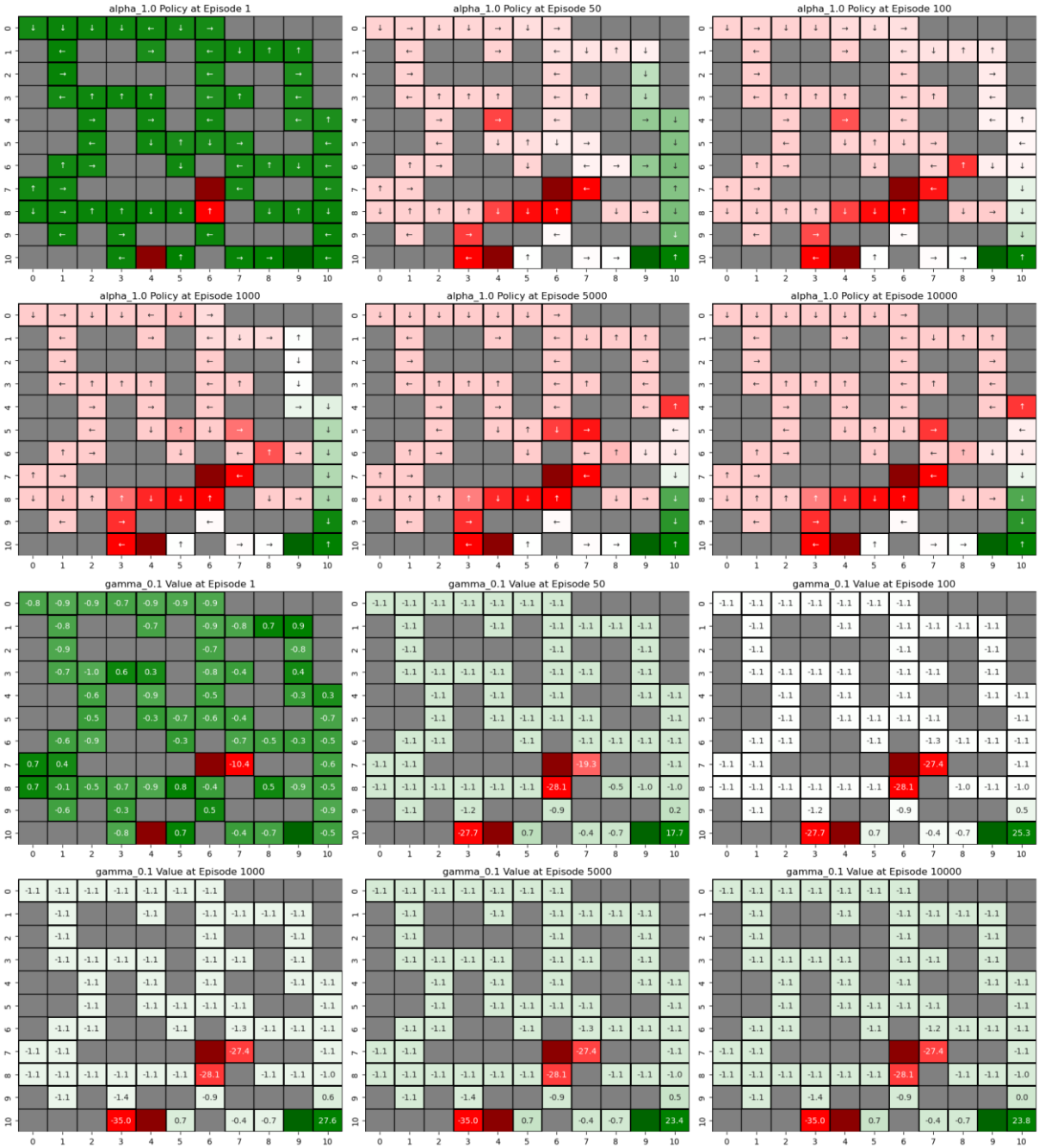

alpha_0.001 Value at Episode 1
alpha_0.001 Value at Episode 50
alpha_0.001 Value at Episode 100
alpha_0.001 Value at Episode 1000
alpha_0.001 Value at Episode 5000
alpha_0.001 Value at Episode 10000

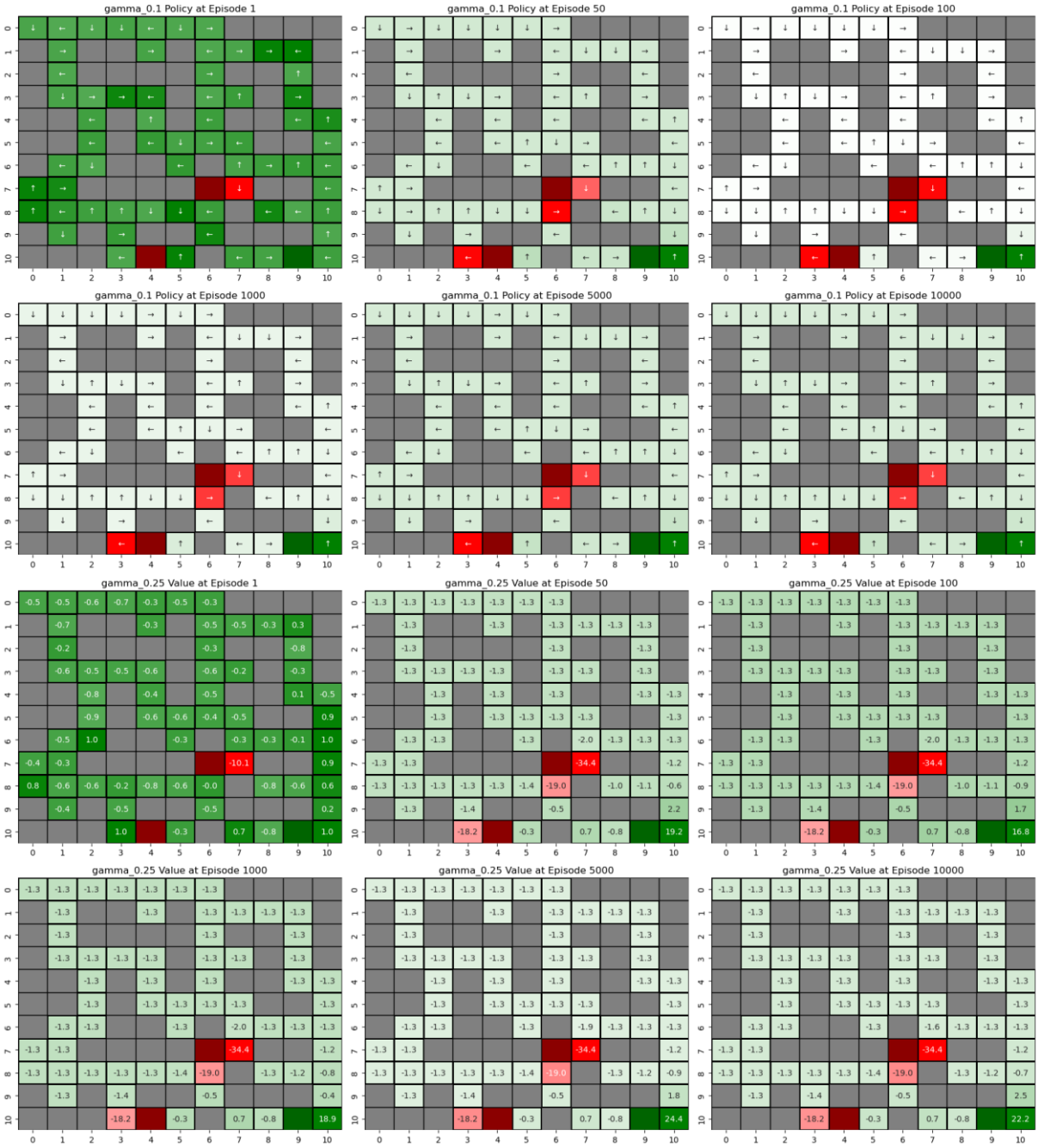

alpha_0.001 Policy at Episode 1
alpha_0.001 Policy at Episode 50
alpha_0.001 Policy at Episode 100
alpha_0.001 Policy at Episode 1000
alpha_0.001 Policy at Episode 5000
alpha_0.001 Policy at Episode 10000
alpha_0.01 Value at Episode 1
alpha_0.01 Value at Episode 50
alpha_0.01 Value at Episode 100
alpha_0.01 Value at Episode 1000
alpha_0.01 Value at Episode 5000
alpha_0.01 Value at Episode 10000

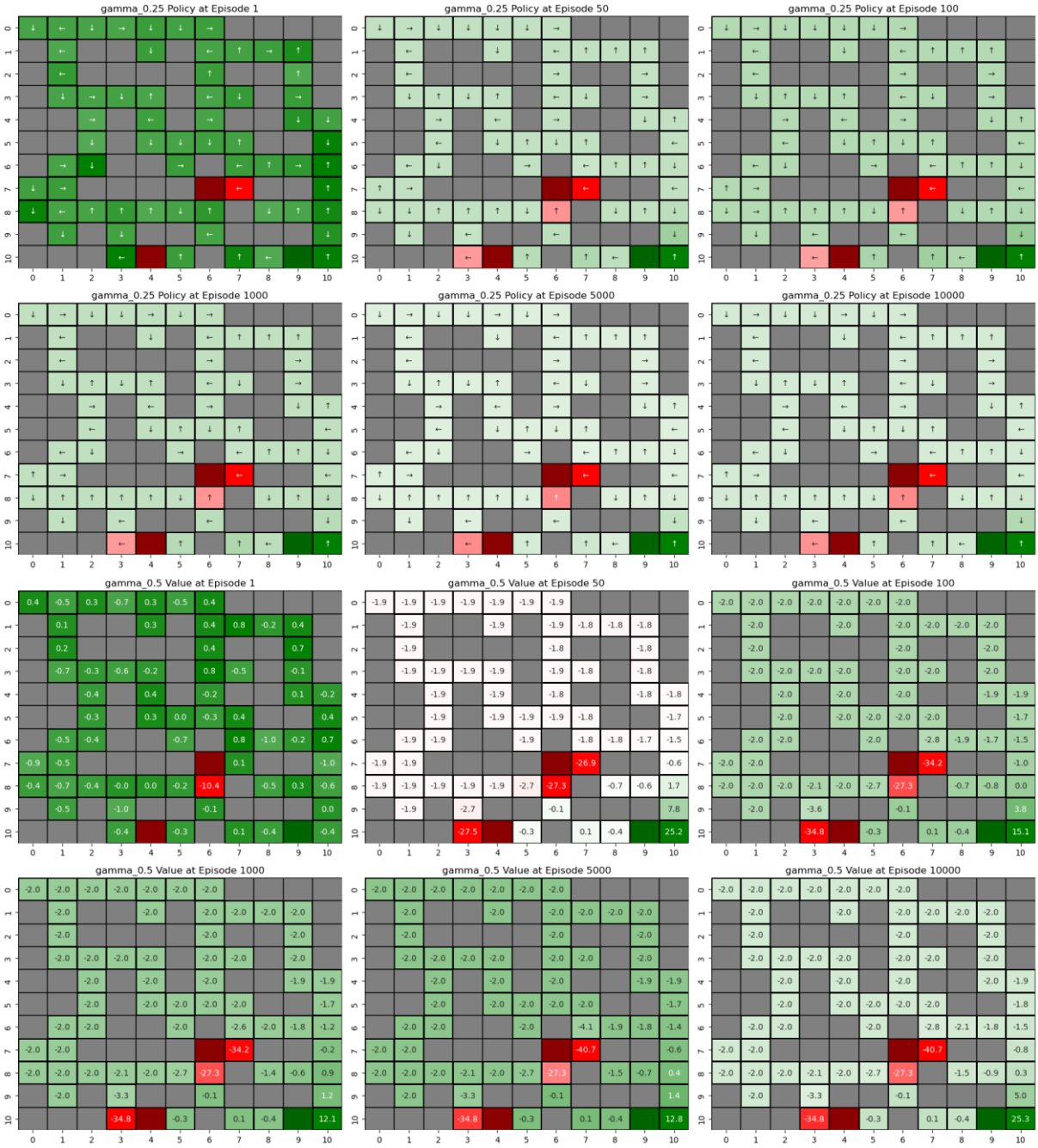

alpha_0.01 Policy at Episode 1
alpha_0.01 Policy at Episode 50
alpha_0.01 Policy at Episode 100
alpha_0.01 Policy at Episode 1000
alpha_0.01 Policy at Episode 5000
alpha_0.01 Policy at Episode 10000
alpha_0.1 Value at Episode 1
alpha_0.1 Value at Episode 50
alpha_0.1 Value at Episode 100
alpha_0.1 Value at Episode 1000
alpha_0.1 Value at Episode 5000
alpha_0.1 Value at Episode 10000

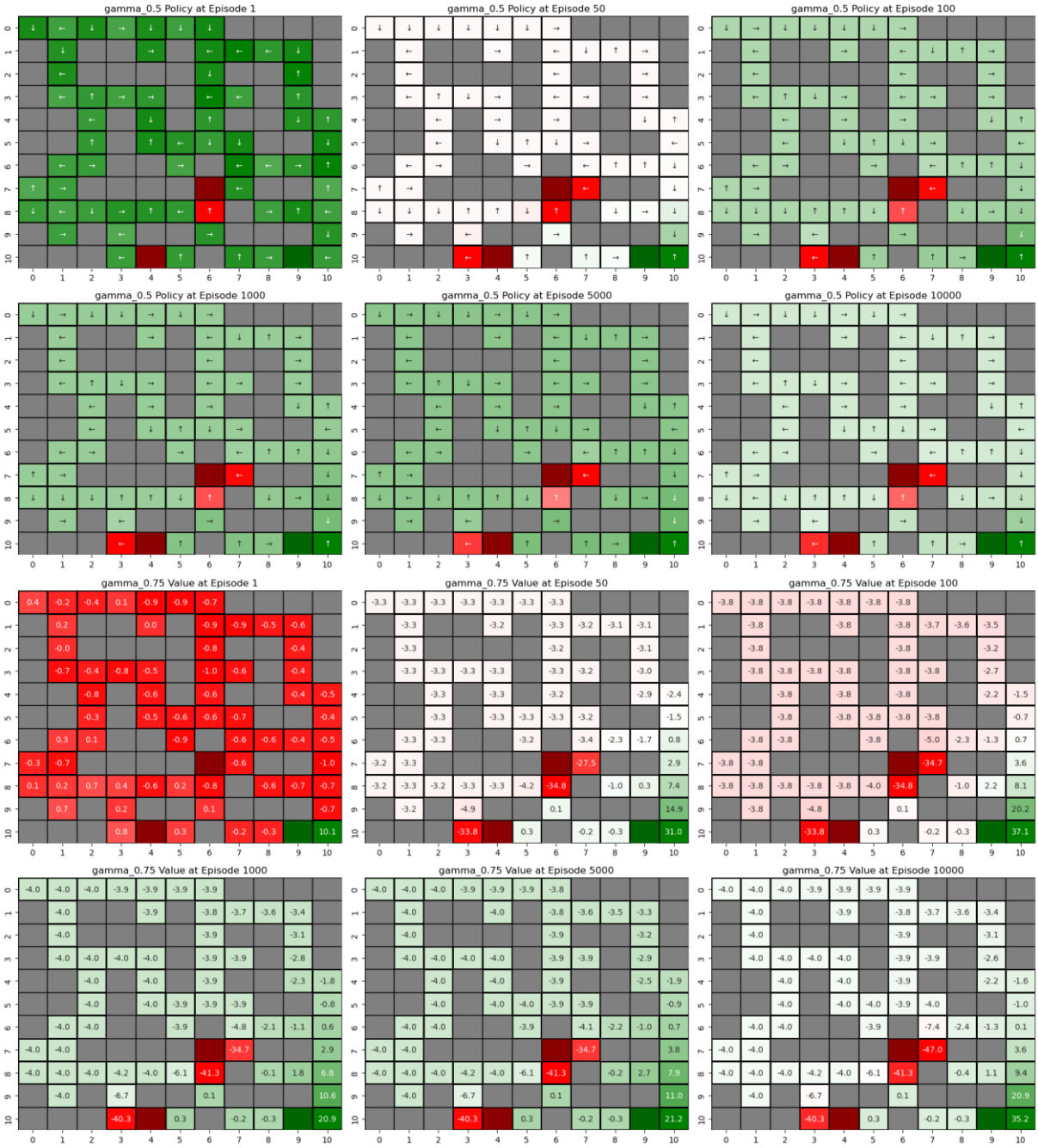

alpha_0.1 Policy at Episode 1
alpha_0.1 Policy at Episode 50
alpha_0.1 Policy at Episode 100
alpha_0.1 Policy at Episode 1000
alpha_0.1 Policy at Episode 5000
alpha_0.1 Policy at Episode 10000
alpha_0.5 Value at Episode 1
alpha_0.5 Value at Episode 50
alpha_0.5 Value at Episode 100
alpha_0.5 Value at Episode 1000
alpha_0.5 Value at Episode 5000
alpha_0.5 Value at Episode 10000

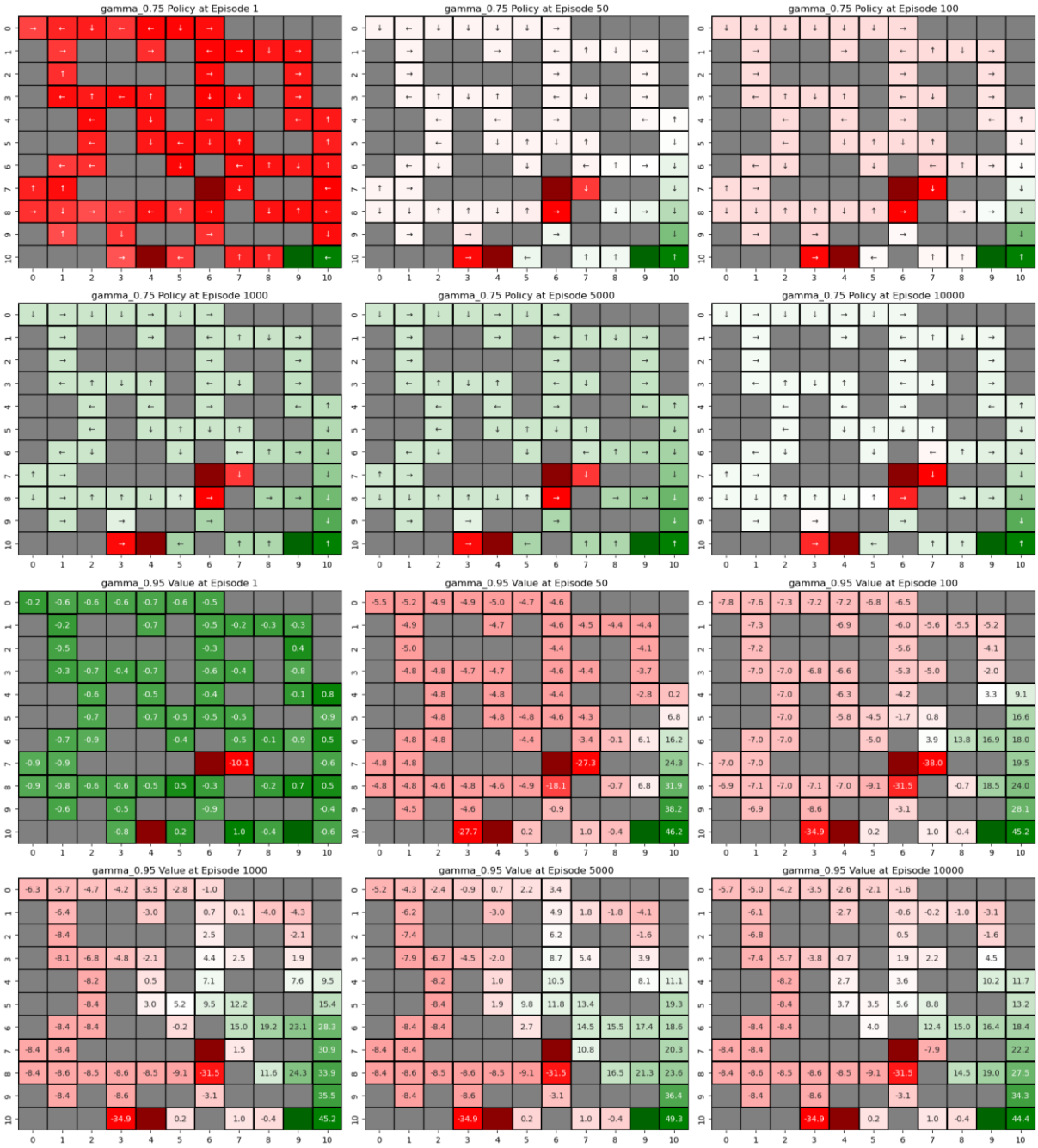

alpha_0.5 Policy at Episode 1
alpha_0.5 Policy at Episode 50
alpha_0.5 Policy at Episode 100
alpha_0.5 Policy at Episode 1000
alpha_0.5 Policy at Episode 5000
alpha_0.5 Policy at Episode 10000
alpha_1.0 Value at Episode 1
alpha_1.0 Value at Episode 50
alpha_1.0 Value at Episode 100
alpha_1.0 Value at Episode 1000
alpha_1.0 Value at Episode 5000
alpha_1.0 Value at Episode 10000

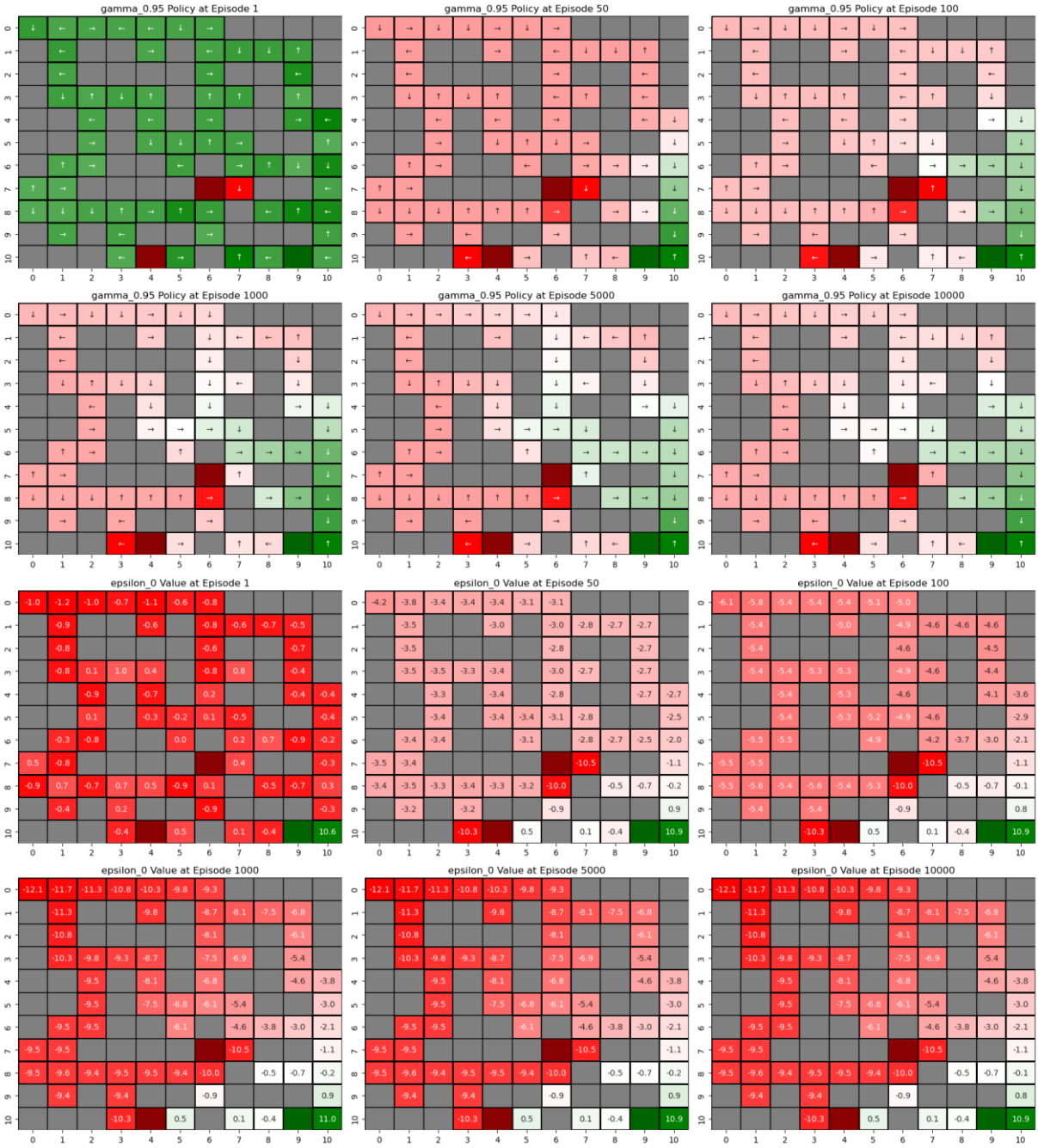

alpha_1.0 Policy at Episode 1
alpha_1.0 Policy at Episode 50
alpha_1.0 Policy at Episode 100
alpha_1.0 Policy at Episode 1000
alpha_1.0 Policy at Episode 5000
alpha_1.0 Policy at Episode 10000
gamma_0.1 Value at Episode 1
gamma_0.1 Value at Episode 50
gamma_0.1 Value at Episode 100
gamma_0.1 Value at Episode 1000
gamma_0.1 Value at Episode 5000
gamma_0.1 Value at Episode 10000

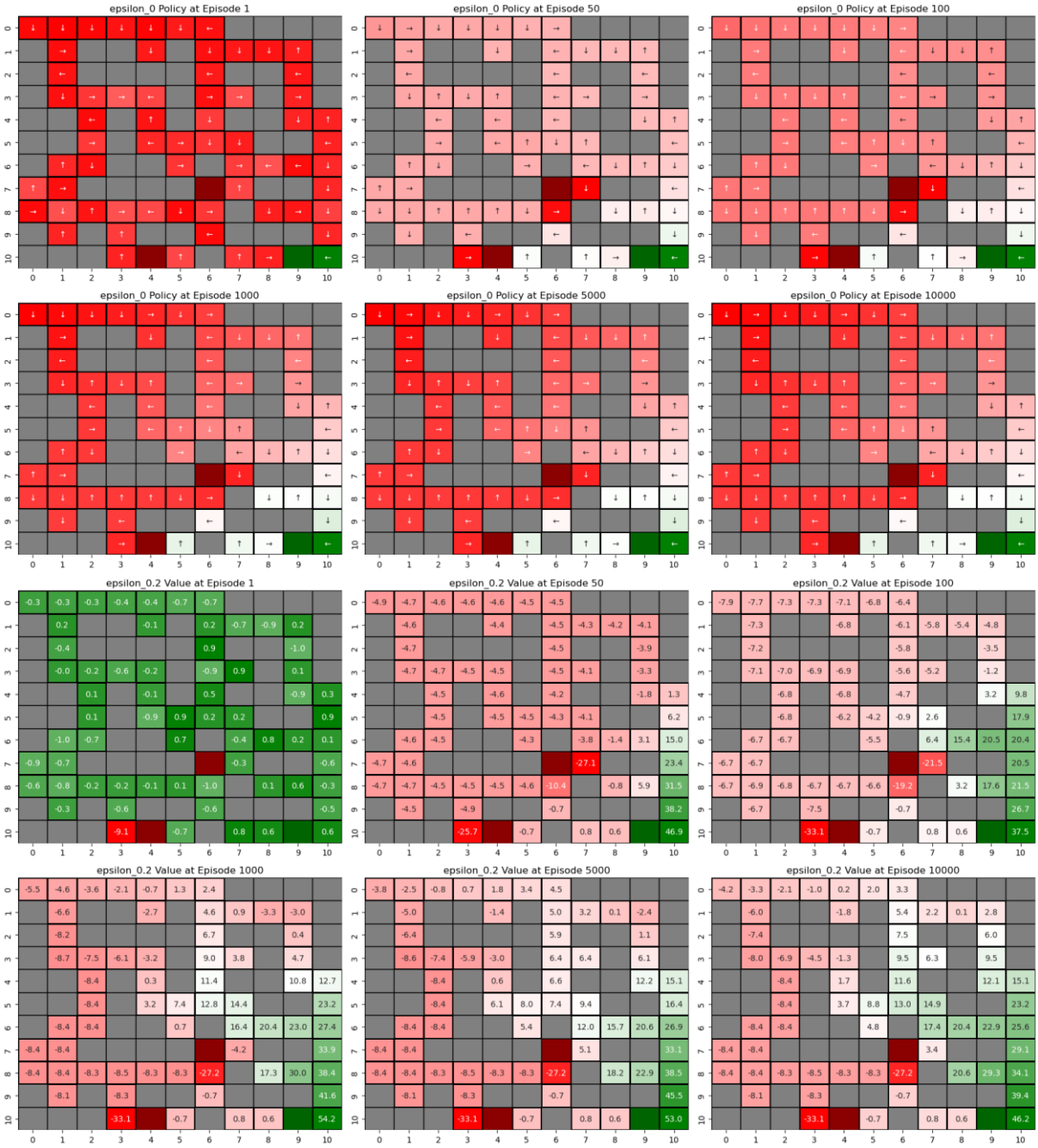

gamma_0.1 Policy at Episode 1
gamma_0.1 Policy at Episode 50
gamma_0.1 Policy at Episode 100
gamma_0.1 Policy at Episode 1000
gamma_0.1 Policy at Episode 5000
gamma_0.1 Policy at Episode 10000
gamma_0.25 Value at Episode 1
gamma_0.25 Value at Episode 50
gamma_0.25 Value at Episode 100
gamma_0.25 Value at Episode 1000
gamma_0.25 Value at Episode 5000
gamma_0.25 Value at Episode 10000

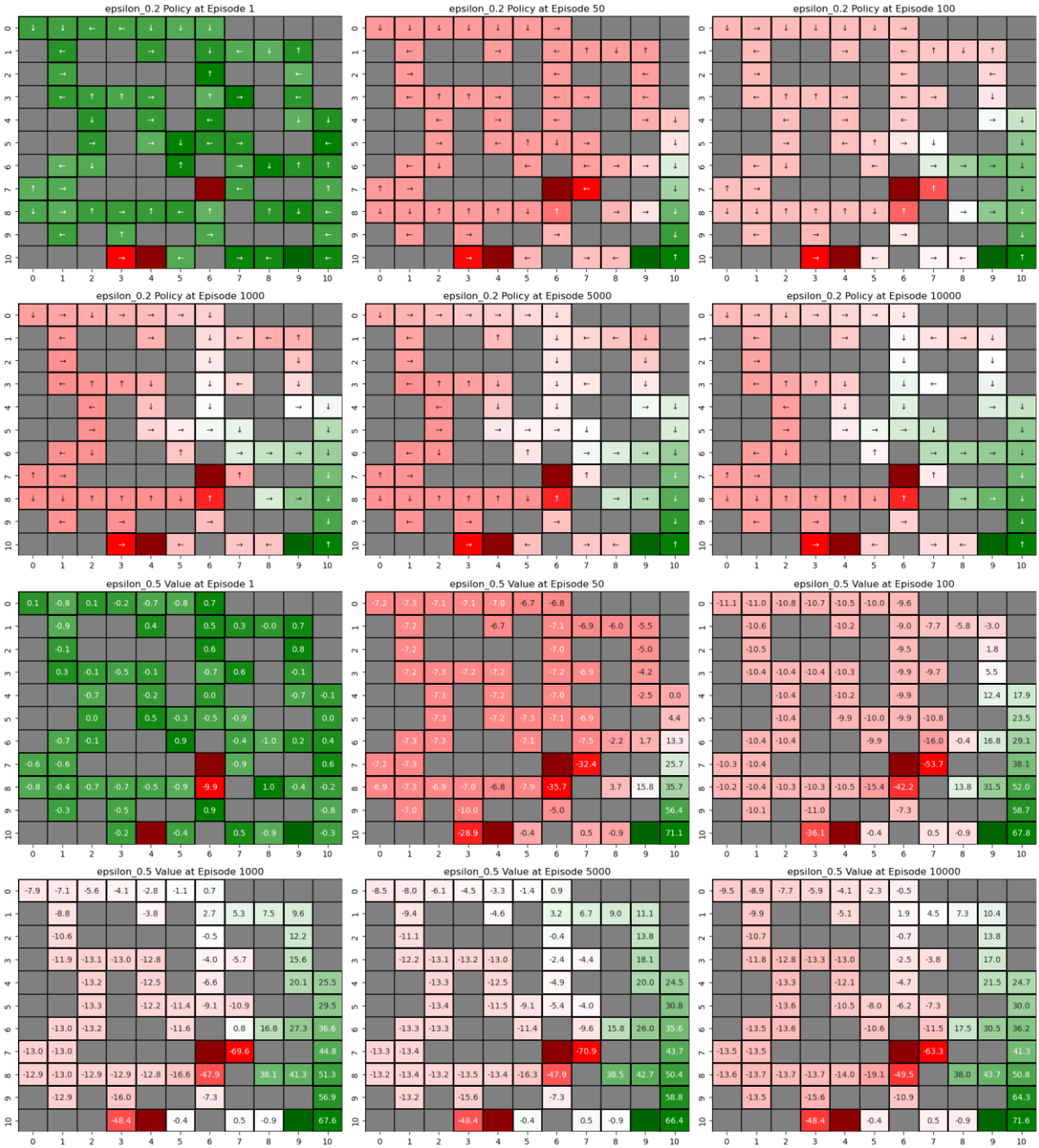

gamma_0.25 Policy at Episode 1
gamma_0.25 Policy at Episode 50
gamma_0.25 Policy at Episode 100
gamma_0.25 Policy at Episode 1000
gamma_0.25 Policy at Episode 5000
gamma_0.25 Policy at Episode 10000
gamma_0.5 Value at Episode 1
gamma_0.5 Value at Episode 50
gamma_0.5 Value at Episode 100
gamma_0.5 Value at Episode 1000
gamma_0.5 Value at Episode 5000
gamma_0.5 Value at Episode 10000

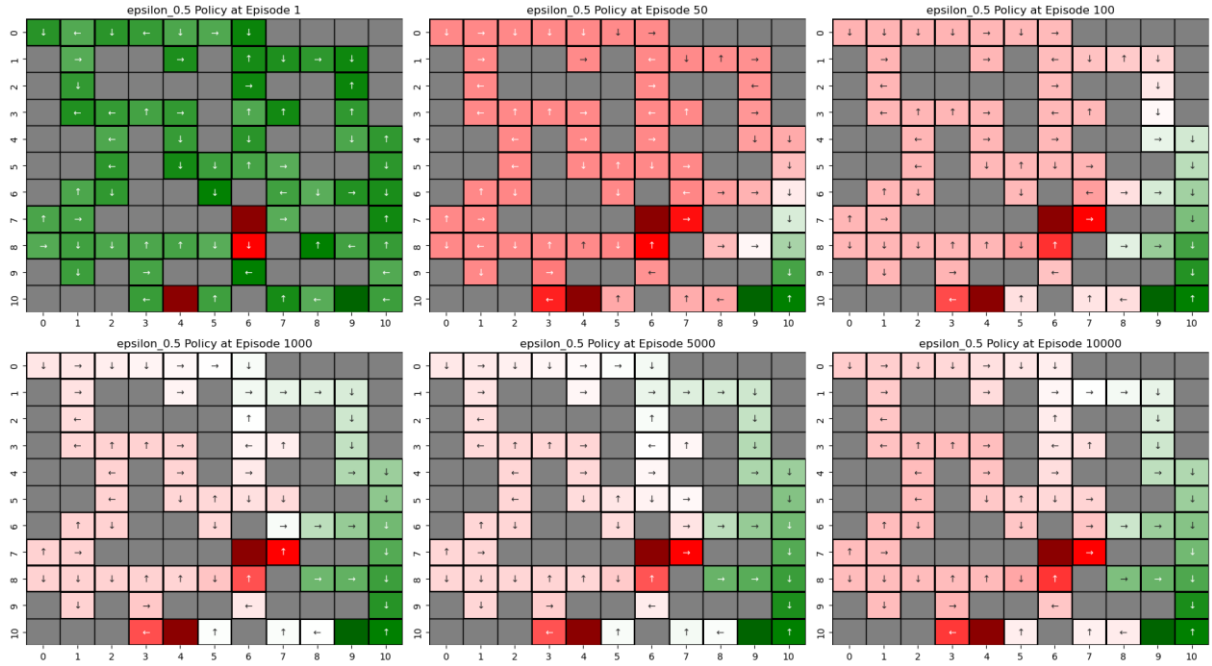

gamma_0.5 Policy at Episode 1
gamma_0.5 Policy at Episode 50
gamma_0.5 Policy at Episode 100
gamma_0.5 Policy at Episode 1000
gamma_0.5 Policy at Episode 5000
gamma_0.5 Policy at Episode 10000
gamma_0.75 Value at Episode 1
gamma_0.75 Value at Episode 50
gamma_0.75 Value at Episode 100
gamma_0.75 Value at Episode 1000
gamma_0.75 Value at Episode 5000
gamma_0.75 Value at Episode 10000


gamma_0.75 Policy at Episode 1
gamma_0.75 Policy at Episode 50
gamma_0.75 Policy at Episode 100
gamma_0.75 Policy at Episode 1000
gamma_0.75 Policy at Episode 5000
gamma_0.75 Policy at Episode 10000
gamma_0.95 Value at Episode 1
gamma_0.95 Value at Episode 50
gamma_0.95 Value at Episode 100
gamma_0.95 Value at Episode 1000
gamma_0.95 Value at Episode 5000
gamma_0.95 Value at Episode 10000

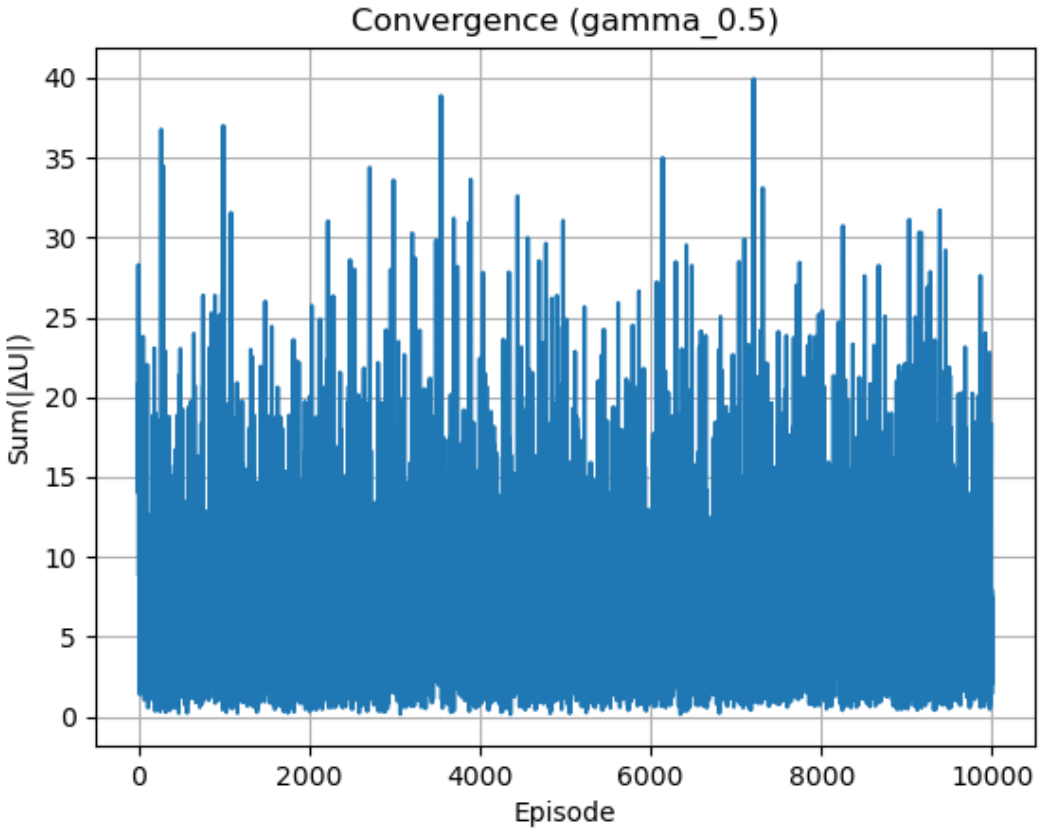

gamma_0.95 Policy at Episode 1
gamma_0.95 Policy at Episode 50
gamma_0.95 Policy at Episode 100
gamma_0.95 Policy at Episode 1000
gamma_0.95 Policy at Episode 5000
gamma_0.95 Policy at Episode 10000
epsilon_0 Value at Episode 1
epsilon_0 Value at Episode 50
epsilon_0 Value at Episode 100
epsilon_0 Value at Episode 1000
epsilon_0 Value at Episode 5000
epsilon_0 Value at Episode 10000

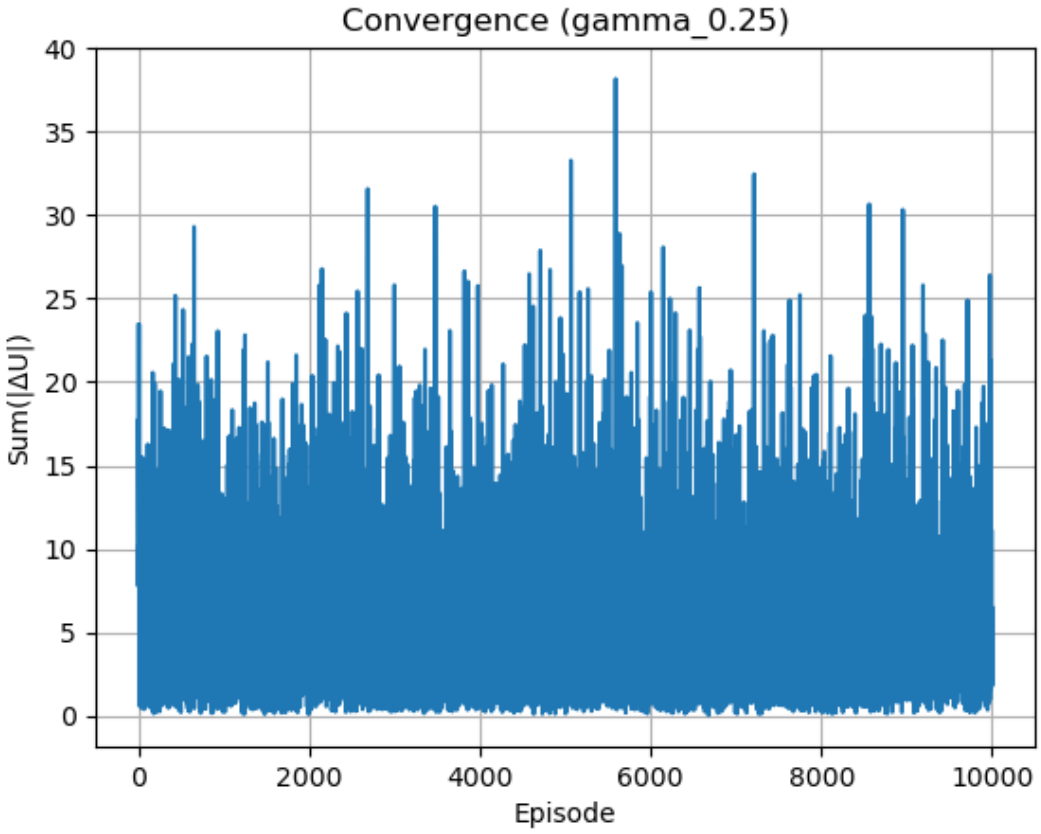

epsilon_0 Policy at Episode 1
epsilon_0 Policy at Episode 50
epsilon_0 Policy at Episode 100
epsilon_0 Policy at Episode 1000
epsilon_0 Policy at Episode 5000
epsilon_0 Policy at Episode 10000
epsilon_0.2 Value at Episode 1
epsilon_0.2 Value at Episode 50
epsilon_0.2 Value at Episode 100
epsilon_0.2 Value at Episode 1000
epsilon_0.2 Value at Episode 5000
epsilon_0.2 Value at Episode 10000

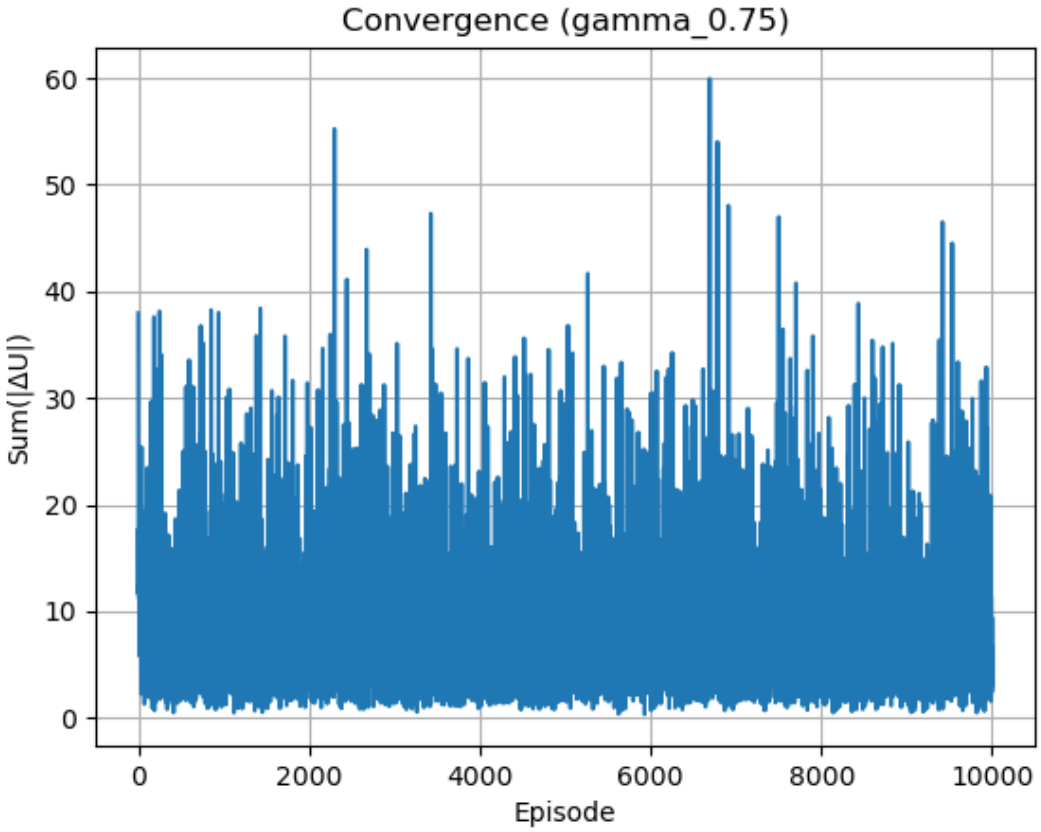

epsilon_0.2 Policy at Episode 1
epsilon_0.2 Policy at Episode 50
epsilon_0.2 Policy at Episode 100
epsilon_0.2 Policy at Episode 1000
epsilon_0.2 Policy at Episode 5000
epsilon_0.2 Policy at Episode 10000
epsilon_0.5 Value at Episode 1
epsilon_0.5 Value at Episode 50
epsilon_0.5 Value at Episode 100
epsilon_0.5 Value at Episode 1000
epsilon_0.5 Value at Episode 5000
epsilon_0.5 Value at Episode 10000

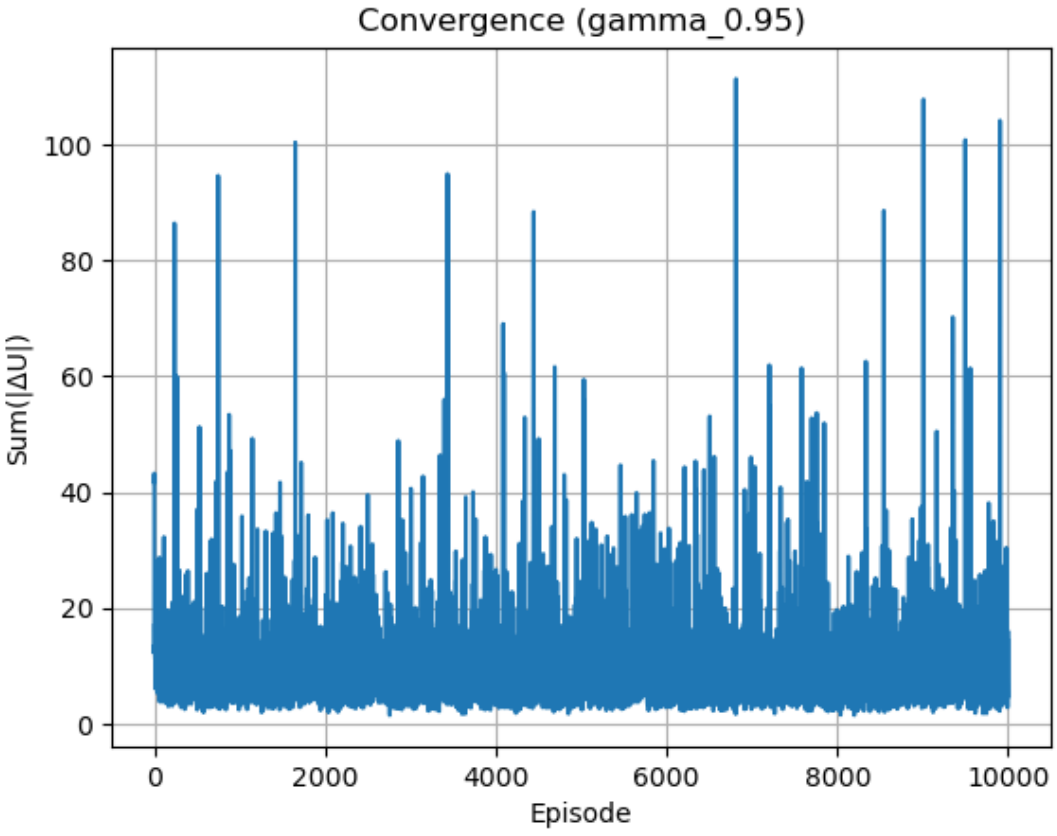


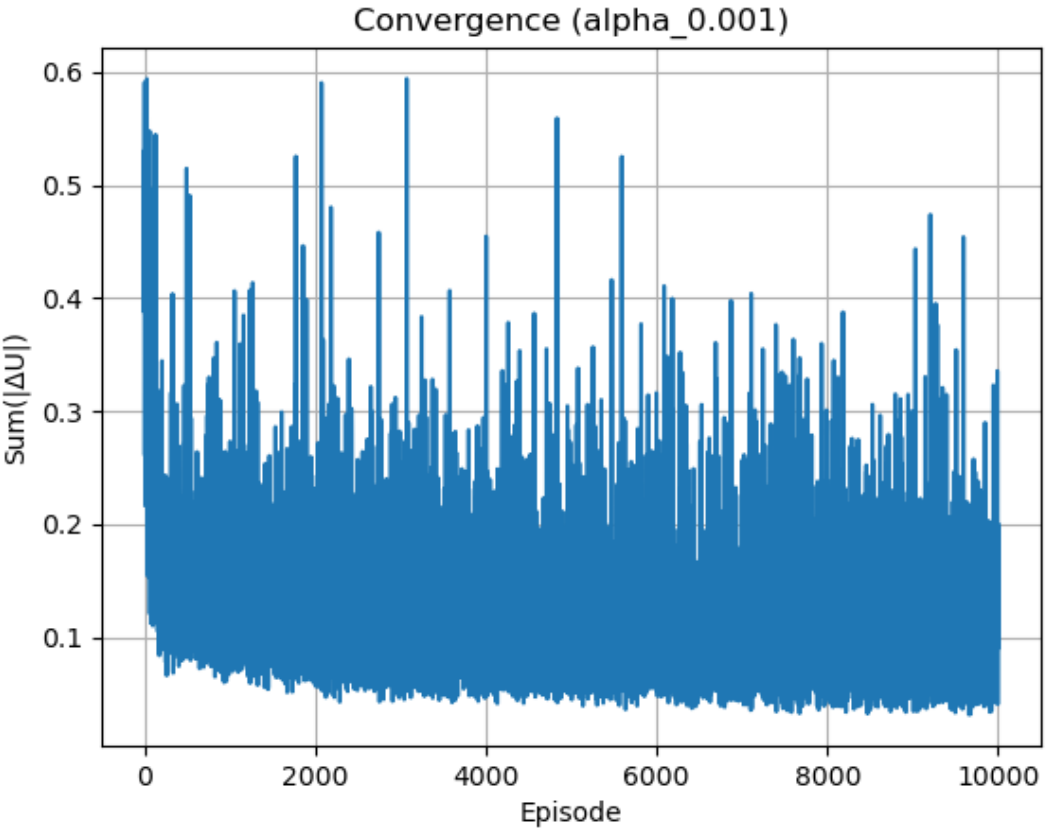
Convergence Plots:

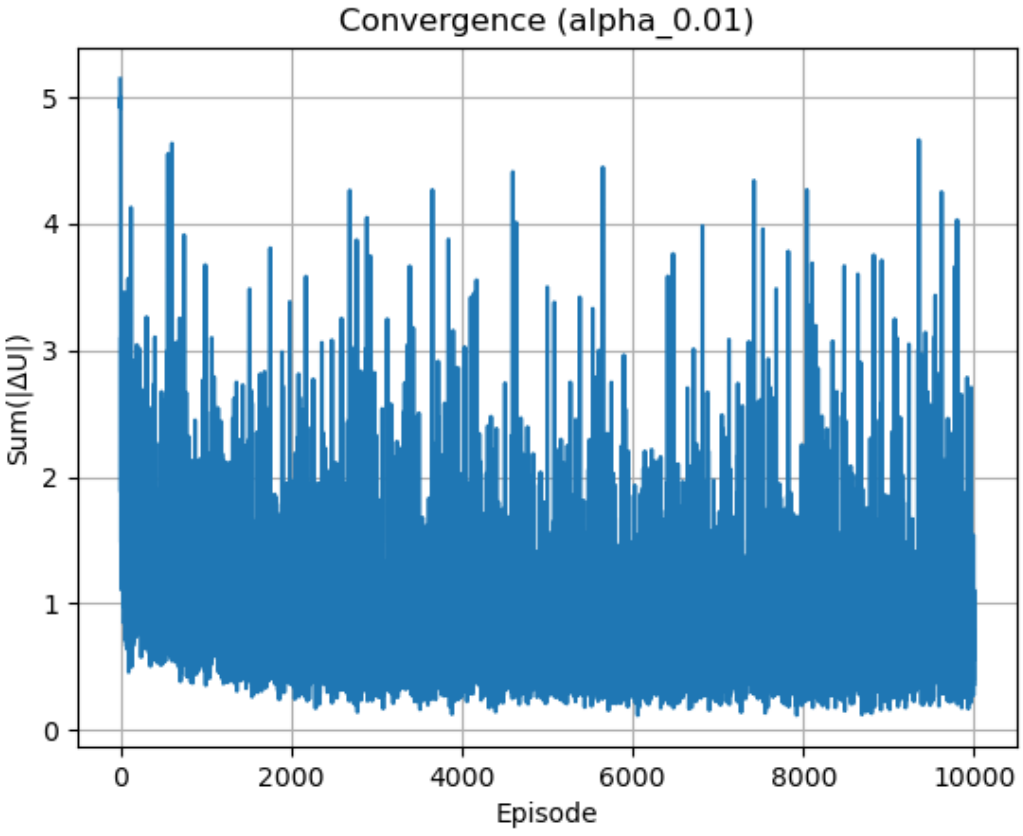

Convergence (gamma_0.5)
Sum(|ΔU|)
Episode
0
5
10
15
20
25
30
35
40
0
2000
4000
6000
8000
10000

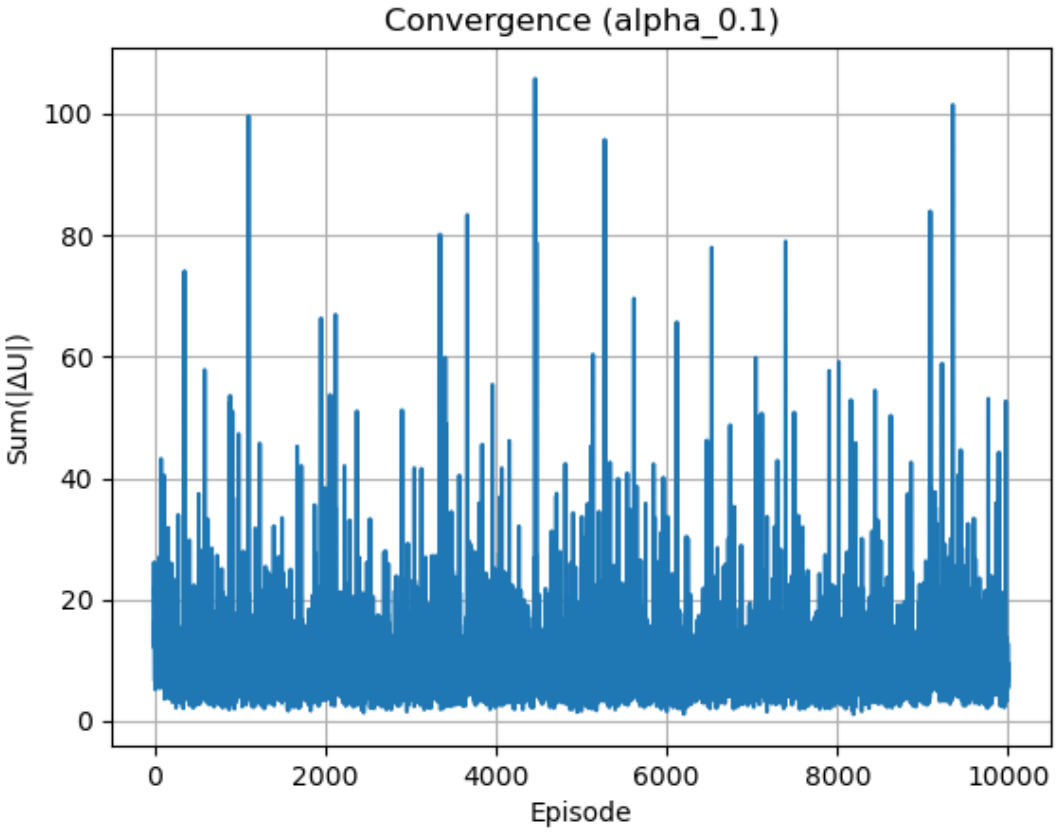

Convergence (gamma_0.25)
Sum(|ΔU|)
Episode
0
5
10
15
20
25
30
35
40
0
2000
4000
6000
8000
10000

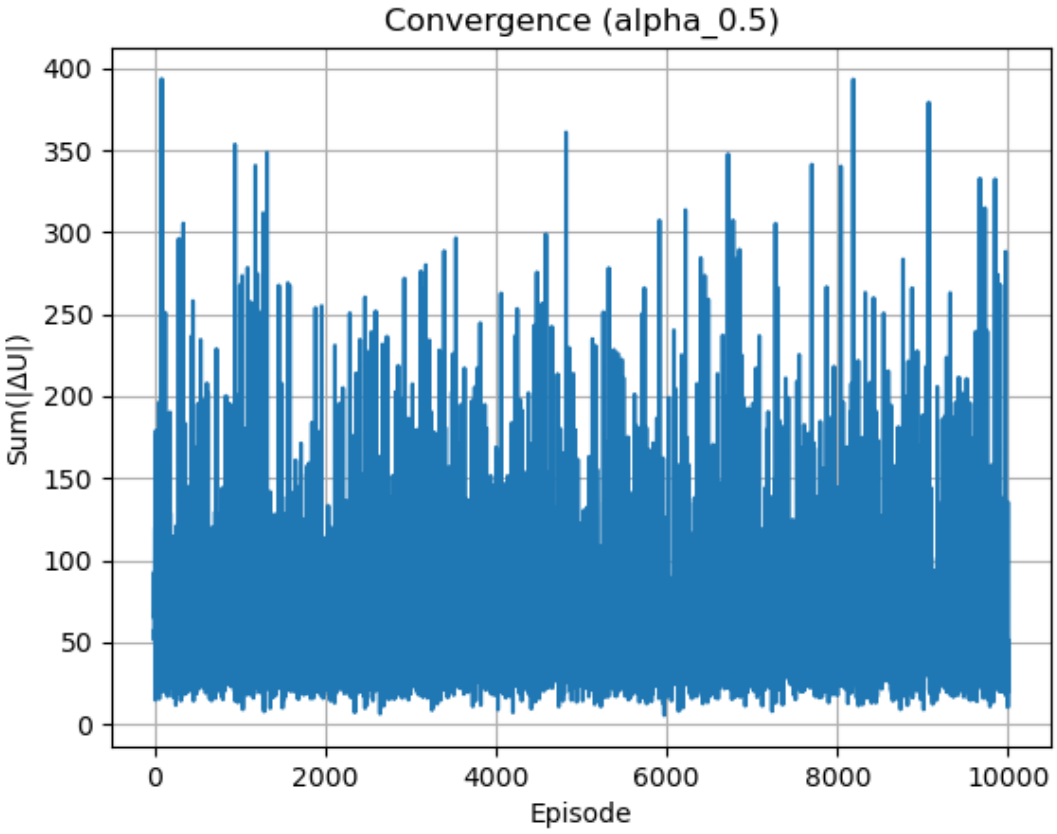

Convergence (gamma_0.75)
Sum(|ΔU|)
Episode
0
10
20
30
40
50
60
0
2000
4000
6000
8000
10000

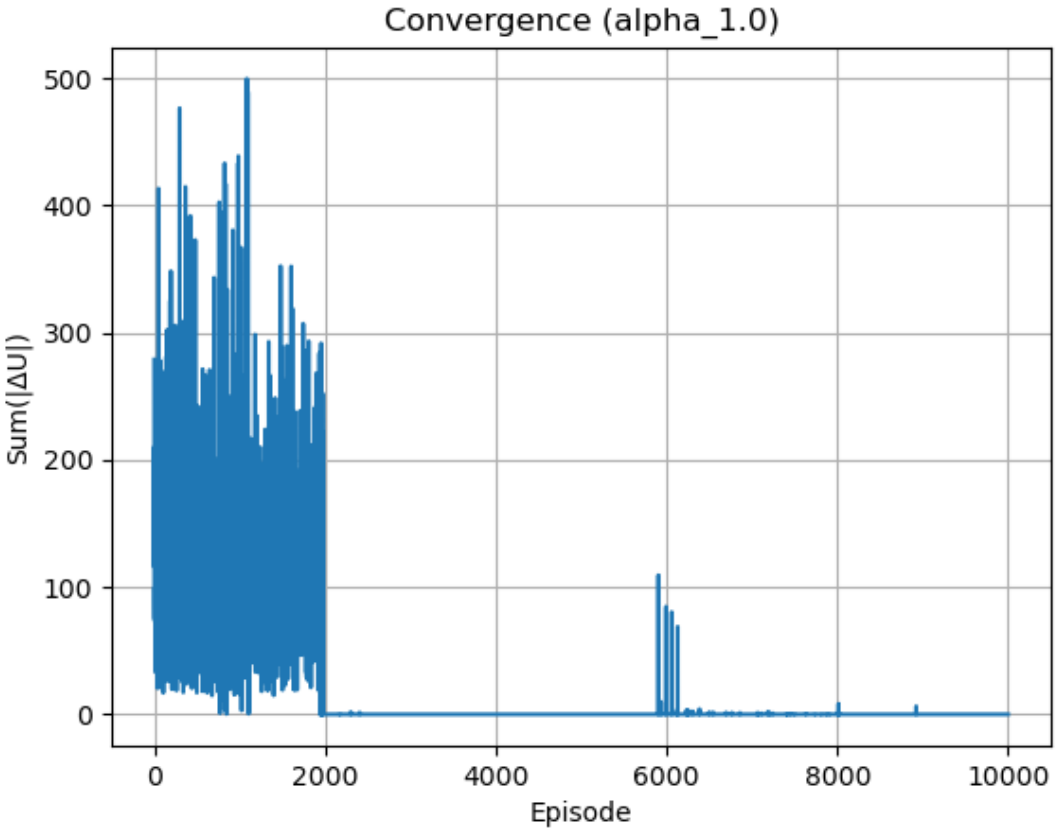

Convergence (gamma_0.95)
Sum(|ΔU|)
Episode
0
20
40
60
80
100
0
2000
4000
6000
8000
10000

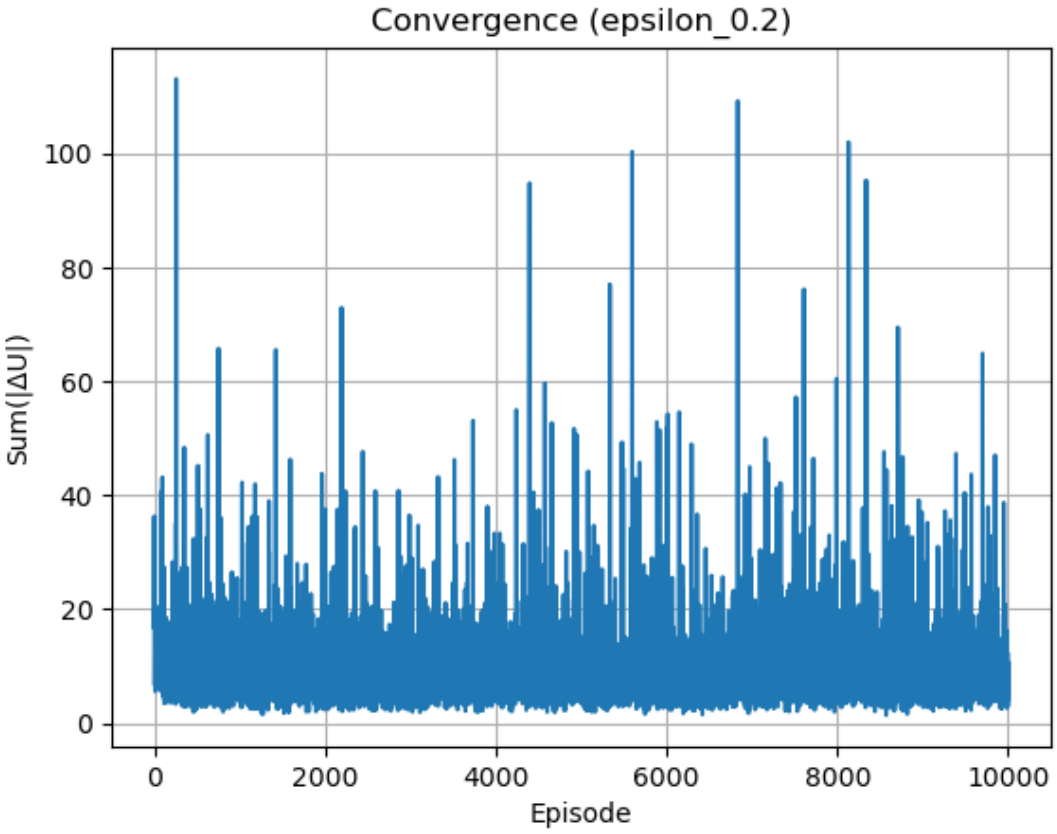

Convergence (alpha_0.001)
Sum(|ΔU|)
0.6
0.5
0.4
0.3
0.2
0.1
0
2000
4000
6000
8000
10000
Episode

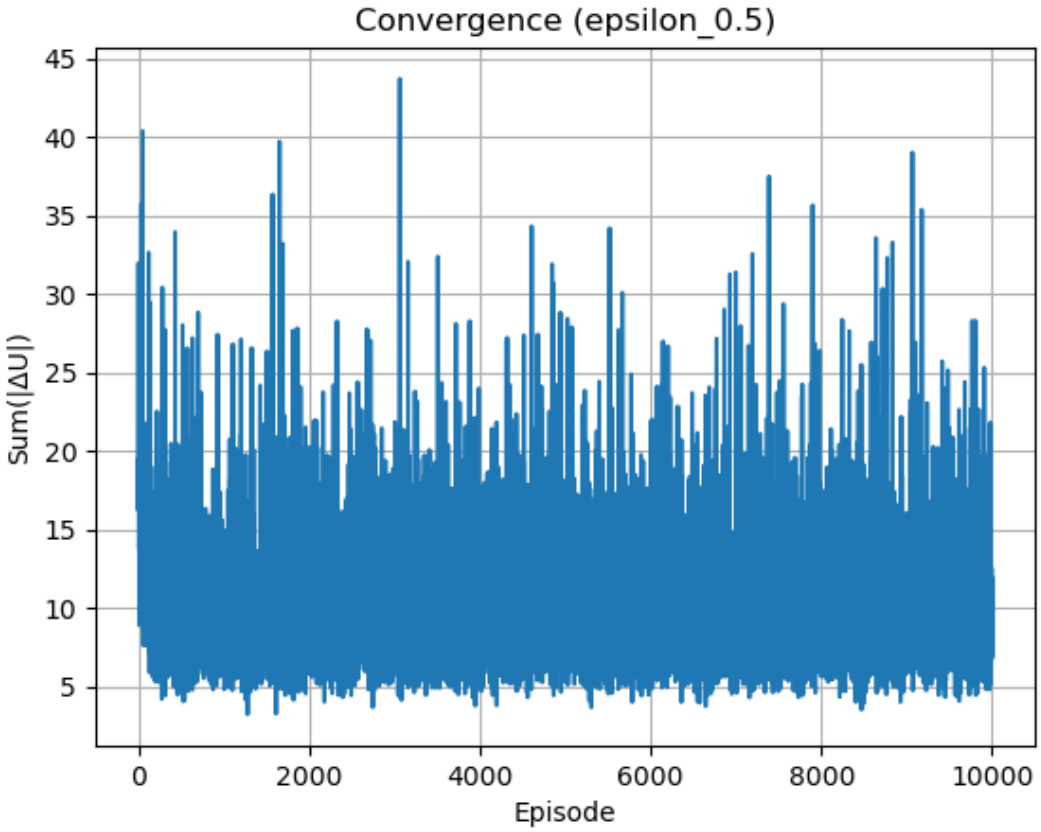

Convergence (alpha_0.01)
Sum(|ΔU|)
5
4
3
2
1
0
0
2000
4000
6000
8000
10000
Episode

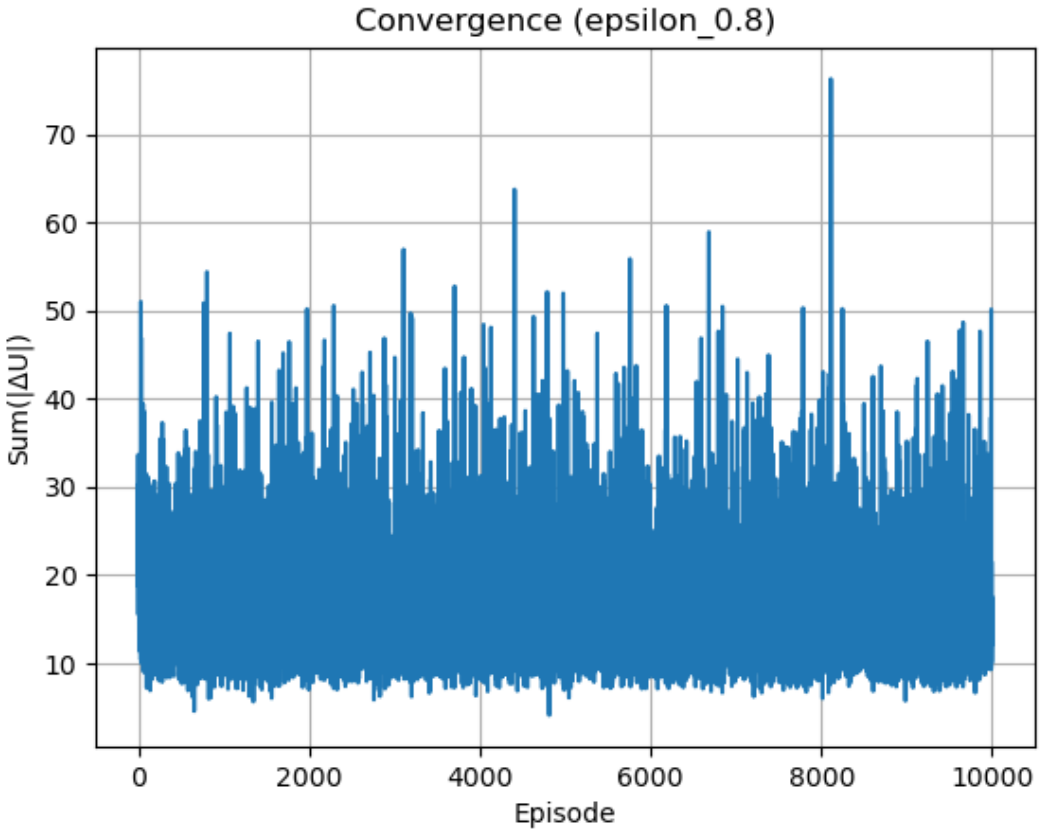

Convergence (alpha_0.1)
Sum(|ΔU|)
0
20
40
60
80
100
0
2000
4000
6000
8000
10000
Episode

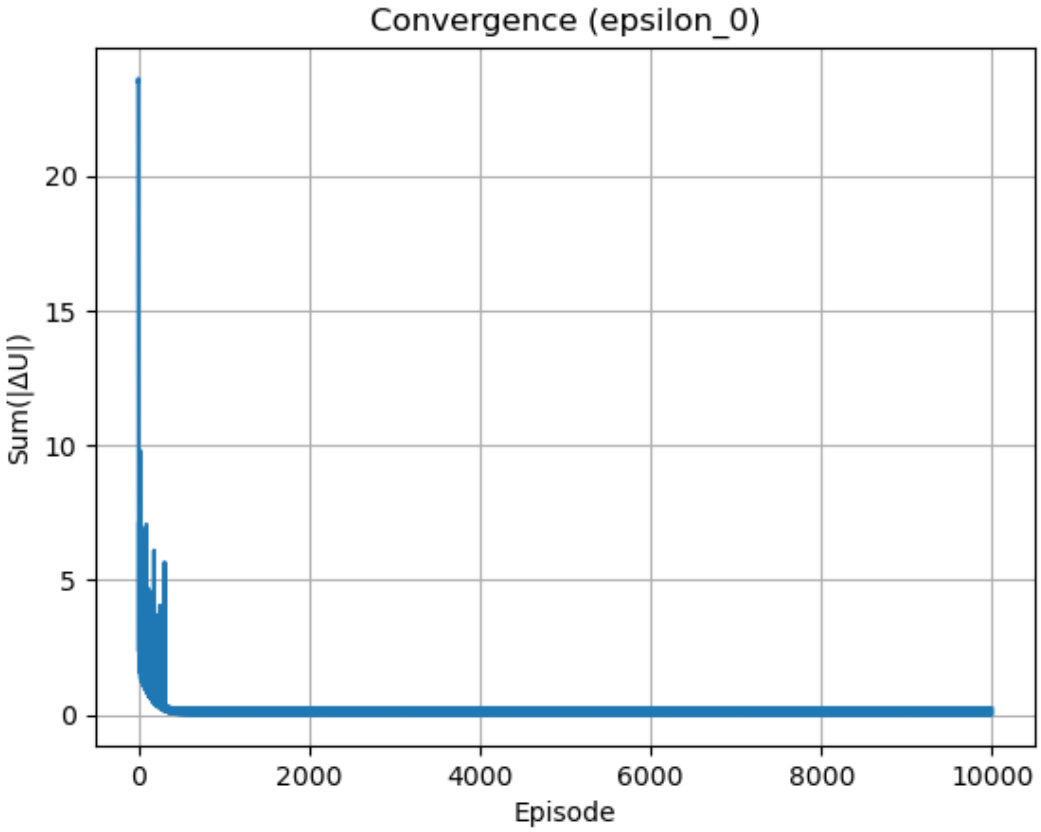

Convergence (alpha_0.5)
Sum(|ΔU|)
0
50
100
150
200
250
300
350
400
0
2000
4000
6000
8000
10000
Episode

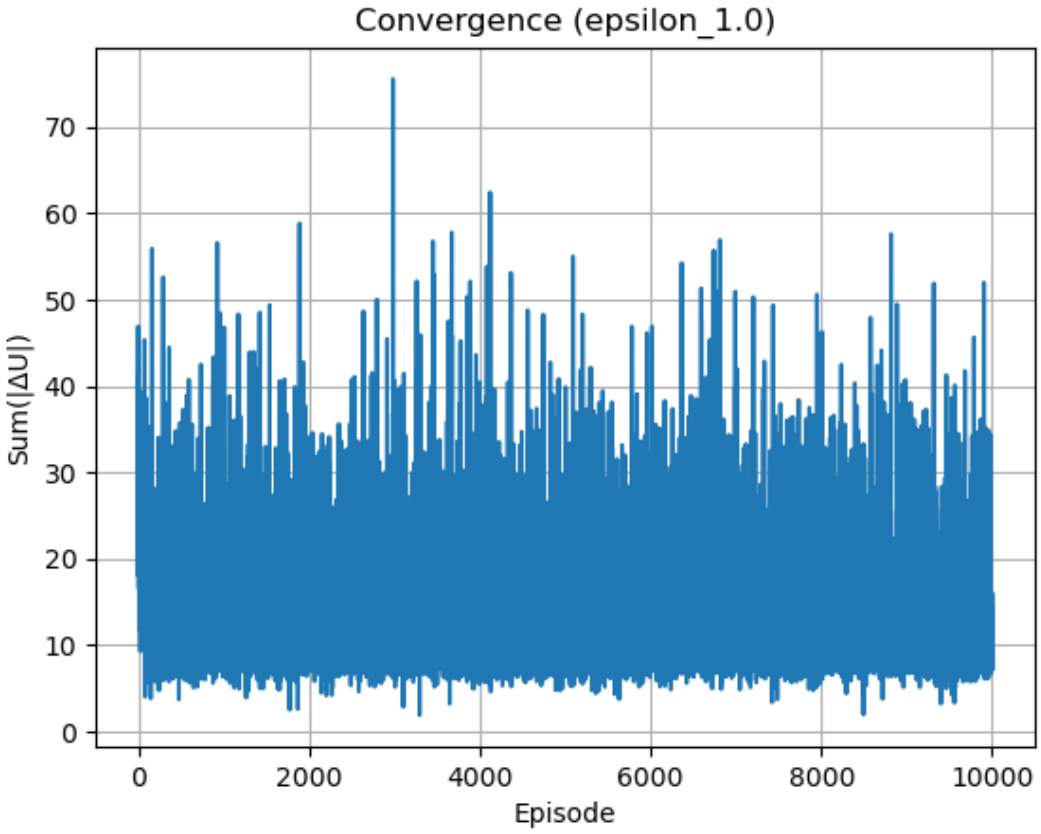

Convergence (alpha_1.0)
Sum(|ΔU|)
500
400
300
200
100
0
0
2000
4000
6000
8000
10000
Episode

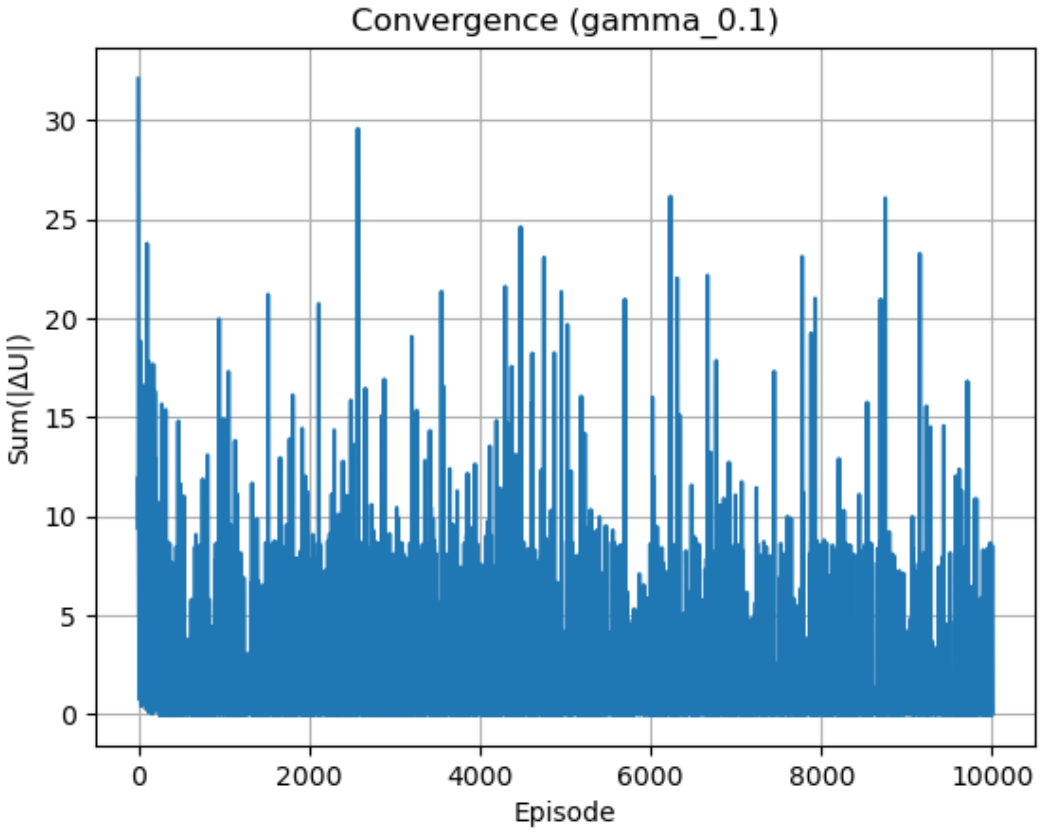

Convergence (epsilon_0.2)
Sum(|ΔU|)
100
80
60
40
20
0
0
2000
4000
6000
8000
10000
Episode


Convergence (epsilon_0.5)
Sum(|ΔU|)
45
40
35
30
25
20
15
10
5
0
2000
4000
6000
8000
10000
Episode


Convergence (epsilon_0.8)
Sum(|ΔU|)
70
60
50
40
30
20
10
0
2000
4000
6000
8000
10000
Episode


Convergence (epsilon_0)
Sum(|ΔU|)
0
5
10
15
20
0
2000
4000
6000
8000
10000
Episode


Convergence (epsilon_1.0)
Sum(|ΔU|)
0
10
20
30
40
50
60
70
0
2000
4000
6000
8000
10000
Episode


Convergence (gamma_0.1)
Sum(|ΔU|)
Episode
0
5
10
15
20
25
30
0
2000
4000
6000
8000
10000