Assignment 3: Reinforcement Learning Taxi Game

Student ID: 114522602, Name: Robby Arifandri

1. *(10%)Try to improve the reward settings or add new reward rules for the game, then explain why you design it this way and what effects these modifications bring to the agent's learning and behavior.*

Table 1. Experiment result on Reward Values

Algorithm	Reward Values	Result
Policy Gradient Agent, 50000 episodes	REWARD_STEP = -1 REWARD_DELIVERY = 20 REWARD_ILLEGAL = -10	avg reward: -149.89 avg steps: 148.53 success rate: 27.7% evaluation score: 0.2613 (26.13%)
Policy Gradient Agent, 50000 episodes	REWARD_STEP = -0.5 REWARD_DELIVERY = 10 REWARD_ILLEGAL = -1	avg rewards: -69.86 avg steps: 142.57 success rate: 27.7% evaluation score: 0.2919 (29.19%)
Policy Gradient Agent, 50000 episodes	REWARD_STEP = -0.5 REWARD_DELIVERY = 10 REWARD_ILLEGAL = -5	avg reward: -71.47 avg steps: 148.76 success rate: 27.7% evaluation score: 0.2603 (26.03%)

Initially, we used PolicyGradientAgent and 50000 episodes from the baseline code, and only changing the reward values resulted in getting evaluation scores of maximum 29.19%. Since evaluation score is calculated from 80% steps score + 20% success rate, low evaluation score came from high number of unsuccessful attempt (agent made a lot of illegal actions) and the high number of steps the agent has taken each episode. Therefore, addition of new reward rules is necessary to enforce the agent taking shorter distance and less unnecessary actions.

The additional rule introduced are as following:

```
# --- CUSTOM REWARD RULES ---
self._valid_move_penalty = -0.5
self._wall_penalty = -2
self._successful_pickup_bonus = 5
self._wrong_pickup_penalty = -5
self._successful_dropoff_reward = 20
self._wrong_dropoff_penalty = -10
```

```
self._delivery_bonus = 30 # This will be added on top of
successful_dropoff_reward
self._timeout_penalty = -15
self._closer_reward = 1 # Reward for getting closer to the target
self._farther_penalty = -1.5 # Penalty for getting farther from the
target
```

The new rules encourage goal-oriented behaviour by giving intermediate positive rewards for progress (closer or further with passenger or destination), balances exploration and penalty (lower step penalty), discourages rule violation (wall violation, invalid dropoff).
The agent gets intermediate positive rewards for progress (approaching passenger or destination), which acts as reward shaping. It helps the agent discover useful sub-goals instead of wandering randomly.

Beside using the same Policy Gradient Agent, Q-Learning also explored with both 50000 training episodes to get the result below.

Table 2. Comparison of Policy Gradient and Q-Learning

Algorithm	Reward Values	Results

Policy Gradient Agent, 50000 episodes	self._valid_move_penalty = -0.5
	self._wall_penalty = -2
	self._successful_pickup_bonus = 5
	self._wrong_pickup_penalty = -5
	self._successful_dropoff_reward = 20
	self._wrong_dropoff_penalty = -10
	self._delivery_bonus = 30 # This will be added on top of successful_dropoff_reward
	self._timeout_penalty = -15
	self._closer_reward = 1
		avg reward: -41.29
		avg steps: 142.24
		success rate: 31.1%
		evaluation score: 0.2933 (29.33%)

Q-Learning, 50000 episodes	self._valid_move_penalty = -0.5 self._wall_penalty = -2 self._successful_pickup_bonus = 5 self._wrong_pickup_penalty = -5 self._successful_dropoff_reward = 20 self._wrong_dropoff_penalty = -10 self._delivery_bonus = 30 # This will be added on top of successful_dropoff_reward	avg reward: 10.75 avg steps: 65.65 success rate: 85.9% evaluation score: 0.7092 (70.92%)

	`self._timeout_penalty = -15` `self._closer_reward = 1`	

2. *(15%)Try to modify and compare at least three sets of hyperparameters (episodes, discount factor, learning rate, etc.) and explain what you observed.*

Using the Q Learning algorithm with 10000 episodes, several hyperparameters are explored using grid search.

Table 3. Result of hyperparameter tuning.

#	Learning Rate	Discount Factor	Epsilon Decay Rate	Epsilon Min	Avg Reward	Avg Steps	Success Rate (%)	Evaluation Score	Notes
1	0.1	0.99	0.9999	0.01	-49.99	141.42	32	0.2983	Baseline
2	0.1	0.99	0.999	0.01	15.18	66.95	86	0.7042	Highest
3	0.01	0.99	0.999	0.05	-28.99	168.29	17	0.1608	Worse
4	0.01	0.99	0.999	0.01	-31.59	177.53	12	0.1139	Worse
5	0.1	0.95	0.999	0.01	0.23	99.63	61	0.5235	Better

Note: No. Episodes: 10000

From the grid search, there are several hyperparameters that are better than the baseline. The first one, used epsilon decay 0.999 which is lower than baseline, showed using less randomness of the agent might get good results, balancing between exploration and exploitation. Using a lower learning rate 0.01 showed too slow learning, making agents stuck with poor evaluation scores. Discount factor, which is related the future reward importance, less than baseline (0.95) also showed a better evaluation score than baseline which showed we may get better results when the agent is more into direct reward.

3. *(15%)Please try other RL methods then specifically compare (data, graphs, etc.) the differences between the method you implemented and the policy gradient method, and explain their respective differences. What are the advantages and disadvantages of .*

Besides the Policy Gradient, Actor-Critic and Q-Learning algorithms are also explored in this assignment. Policy Gradient, a policy-based algorithm, works by directly optimizing policy parameters θ by following the gradient of expected return. Q-Learning on the other hand, is a value-based algorithm, which works by learning action-value function Q(s,a) on each step. Actor-Critic, which is the combination of policy and value-based algorithm, works by calculating gradient of combining policy and value function. The actor improves policy using

feedback from critic. In theory, Q-Learning has advantages in discrete, small environments like Taxi-v3 but not scalable to large or continuous state space. Policy Gradient doesn't need Q table like Q-learning since it works by optimizing policy directly. Policy Gradient also works best in continuous or high-dimensional action spaces but require many samples to converge. Actor-Critic combines benefit of value estimation and policy optimization. However, Actor-Critic if poorly trained, the actor will receive bad gradients and create overfitting.

Policy Gradient Experiment

```
======================================================================
training episodes: 50000
======================================================================
episode 1000/50000 | avg reward: -303.90 | success rate: 7.5% | learning rate: 0.009900
episode 2000/50000 | avg reward: -274.55 | success rate: 14.0% | learning rate: 0.009802
episode 3000/50000 | avg reward: -222.94 | success rate: 19.4% | learning rate: 0.009704
episode 4000/50000 | avg reward: -206.81 | success rate: 28.4% | learning rate: 0.009608
episode 5000/50000 | avg reward: -193.92 | success rate: 38.8% | learning rate: 0.009512
episode 6000/50000 | avg reward: -162.82 | success rate: 45.0% | learning rate: 0.009418
episode 7000/50000 | avg reward: -166.06 | success rate: 52.2% | learning rate: 0.009324
episode 8000/50000 | avg reward: -139.72 | success rate: 51.8% | learning rate: 0.009231
episode 9000/50000 | avg reward: -133.67 | success rate: 60.3% | learning rate: 0.009139
episode 10000/50000 | avg reward: -118.88 | success rate: 60.6% | learning rate:
0.009048
episode 11000/50000 | avg reward: -132.91 | success rate: 61.6% | learning rate:
0.008958
episode 12000/50000 | avg reward: -126.17 | success rate: 64.9% | learning rate:
0.008869
episode 13000/50000 | avg reward: -99.69 | success rate: 64.0% | learning rate: 0.008781
episode 14000/50000 | avg reward: -109.41 | success rate: 64.6% | learning rate:
0.008694
episode 15000/50000 | avg reward: -101.58 | success rate: 66.3% | learning rate:
0.008607
episode 16000/50000 | avg reward: -105.83 | success rate: 68.8% | learning rate:
0.008521
episode 17000/50000 | avg reward: -116.55 | success rate: 63.6% | learning rate:
0.008437
episode 18000/50000 | avg reward: -131.46 | success rate: 64.1% | learning rate:
0.008353
episode 19000/50000 | avg reward: -99.52 | success rate: 64.7% | learning rate: 0.008270
episode 20000/50000 | avg reward: -108.04 | success rate: 66.1% | learning rate:
0.008187
episode 21000/50000 | avg reward: -119.36 | success rate: 65.2% | learning rate:
0.008106
episode 22000/50000 | avg reward: -137.49 | success rate: 63.7% | learning rate:
0.008025
episode 23000/50000 | avg reward: -116.09 | success rate: 65.1% | learning rate:
0.007945
episode 24000/50000 | avg reward: -97.50 | success rate: 64.6% | learning rate: 0.007866
episode 25000/50000 | avg reward: -106.89 | success rate: 64.5% | learning rate:
0.007788
episode 26000/50000 | avg reward: -106.43 | success rate: 67.2% | learning rate:
0.007711
episode 27000/50000 | avg reward: -114.14 | success rate: 67.6% | learning rate:
0.007634
episode 28000/50000 | avg reward: -120.42 | success rate: 65.0% | learning rate:
0.007558
episode 29000/50000 | avg reward: -90.59 | success rate: 66.7% | learning rate: 0.007483
episode 30000/50000 | avg reward: -112.20 | success rate: 65.9% | learning rate:
0.007408
```

```
episode 31000/50000 | avg reward: -133.65 | success rate: 66.4% | learning rate:
0.007334
episode 32000/50000 | avg reward: -120.84 | success rate: 63.8% | learning rate:
0.007261
episode 33000/50000 | avg reward: -100.61 | success rate: 67.7% | learning rate:
0.007189
episode 34000/50000 | avg reward: -125.81 | success rate: 63.3% | learning rate:
0.007118
episode 35000/50000 | avg reward: -106.75 | success rate: 67.4% | learning rate:
0.007047
episode 36000/50000 | avg reward: -117.33 | success rate: 68.5% | learning rate:
0.006977
episode 37000/50000 | avg reward: -105.22 | success rate: 69.3% | learning rate:
0.006907
episode 38000/50000 | avg reward: -99.83 | success rate: 66.4% | learning rate: 0.006839
episode 39000/50000 | avg reward: -98.97 | success rate: 65.5% | learning rate: 0.006771
episode 40000/50000 | avg reward: -100.92 | success rate: 67.8% | learning rate:
0.006703
episode 41000/50000 | avg reward: -121.78 | success rate: 67.1% | learning rate:
0.006636
episode 42000/50000 | avg reward: -118.33 | success rate: 67.2% | learning rate:
0.006570
episode 43000/50000 | avg reward: -86.53 | success rate: 68.9% | learning rate: 0.006505
episode 44000/50000 | avg reward: -127.62 | success rate: 66.3% | learning rate:
0.006440
episode 45000/50000 | avg reward: -117.25 | success rate: 69.3% | learning rate:
0.006376
episode 46000/50000 | avg reward: -106.56 | success rate: 67.3% | learning rate:
0.006313
episode 47000/50000 | avg reward: -108.44 | success rate: 66.7% | learning rate:
0.006250
episode 48000/50000 | avg reward: -100.84 | success rate: 69.4% | learning rate:
0.006188
episode 49000/50000 | avg reward: -111.22 | success rate: 67.9% | learning rate:
0.006126
episode 50000/50000 | avg reward: -107.36 | success rate: 67.7% | learning rate:
0.006065
======================================================================
Training completed!

model saved to policy_gradient_optimized.npy

load model

test result (seed 420000-420999):
======================================================================
   avg reward: -41.29
   avg steps: 142.24
   success rate: 31.1%
   evaluation score: 0.2933 (29.33%)
======================================================================
{'avg_reward': np.float64(-41.287),
 'avg_steps': np.float64(142.236),
 'success_rate': 0.311,
 'evaluation_score': np.float64(0.29325600000000007)}
```

Actor Critic Experiment

```
episode 47000/50000 | avg reward: -52.12 | success rate: 99.8% | learning rate: 0.006250
episode 48000/50000 | avg reward: -64.12 | success rate: 100.0% | learning rate:
0.006188
```

```
episode 49000/50000 | avg reward: -57.27 | success rate: 99.9% | learning rate: 0.006126
episode 50000/50000 | avg reward: -59.19 | success rate: 99.8% | learning rate: 0.006065
======================================================================
Training completed!
model saved to actor_critic_agent.npy

load model

test result (seed 420000-420999):
======================================================================
   avg reward: -136.65
   avg steps: 143.08
   success rate: 30.6%
   evaluation score: 0.2889 (28.89%)
======================================================================
```

Result: Training success rate increase to 99.8% but when test it still drops to just 30%

Q-Learning Experiment (main)

```
======================================================================
training episodes: 50000
======================================================================
episode 1000/50000 | avg reward: -247.43 | success rate: 14.1% | epsilon: 0.9048
episode 2000/50000 | avg reward: -162.37 | success rate: 42.7% | epsilon: 0.8187
episode 3000/50000 | avg reward: -72.72 | success rate: 69.8% | epsilon: 0.7408
episode 4000/50000 | avg reward: -49.21 | success rate: 86.9% | epsilon: 0.6703
episode 5000/50000 | avg reward: -10.46 | success rate: 93.0% | epsilon: 0.6065
episode 6000/50000 | avg reward: 6.15 | success rate: 98.0% | epsilon: 0.5488
episode 7000/50000 | avg reward: 14.74 | success rate: 99.2% | epsilon: 0.4966
episode 8000/50000 | avg reward: 12.81 | success rate: 98.0% | epsilon: 0.4493
episode 9000/50000 | avg reward: 24.77 | success rate: 98.0% | epsilon: 0.4066
episode 10000/50000 | avg reward: 28.08 | success rate: 99.2% | epsilon: 0.3679
episode 11000/50000 | avg reward: 31.37 | success rate: 99.1% | epsilon: 0.3329
episode 12000/50000 | avg reward: 32.73 | success rate: 99.2% | epsilon: 0.3012
episode 13000/50000 | avg reward: 38.85 | success rate: 98.9% | epsilon: 0.2725
episode 14000/50000 | avg reward: 40.40 | success rate: 99.2% | epsilon: 0.2466
episode 15000/50000 | avg reward: 37.81 | success rate: 99.2% | epsilon: 0.2231
episode 16000/50000 | avg reward: 39.98 | success rate: 98.8% | epsilon: 0.2019
episode 17000/50000 | avg reward: 41.22 | success rate: 99.3% | epsilon: 0.1827
episode 18000/50000 | avg reward: 40.41 | success rate: 98.8% | epsilon: 0.1653
episode 19000/50000 | avg reward: 44.98 | success rate: 99.1% | epsilon: 0.1496
episode 20000/50000 | avg reward: 41.71 | success rate: 98.1% | epsilon: 0.1353
episode 21000/50000 | avg reward: 40.30 | success rate: 98.4% | epsilon: 0.1224
episode 22000/50000 | avg reward: 42.49 | success rate: 97.9% | epsilon: 0.1108
episode 23000/50000 | avg reward: 48.82 | success rate: 98.6% | epsilon: 0.1002
episode 24000/50000 | avg reward: 44.56 | success rate: 98.8% | epsilon: 0.0907
episode 25000/50000 | avg reward: 44.00 | success rate: 98.6% | epsilon: 0.0821
episode 26000/50000 | avg reward: 49.62 | success rate: 98.3% | epsilon: 0.0743
episode 27000/50000 | avg reward: 45.34 | success rate: 98.7% | epsilon: 0.0672
episode 28000/50000 | avg reward: 44.23 | success rate: 98.4% | epsilon: 0.0608
episode 29000/50000 | avg reward: 48.63 | success rate: 99.0% | epsilon: 0.0550
```

```
episode 30000/50000 | avg reward: 46.40 | success rate: 98.7% | epsilon: 0.0498
episode 31000/50000 | avg reward: 47.05 | success rate: 98.9% | epsilon: 0.0450
episode 32000/50000 | avg reward: 48.27 | success rate: 99.3% | epsilon: 0.0408
episode 33000/50000 | avg reward: 48.78 | success rate: 99.0% | epsilon: 0.0369
episode 34000/50000 | avg reward: 47.79 | success rate: 98.9% | epsilon: 0.0334
episode 35000/50000 | avg reward: 51.02 | success rate: 97.8% | epsilon: 0.0302
episode 36000/50000 | avg reward: 44.77 | success rate: 98.7% | epsilon: 0.0273
episode 37000/50000 | avg reward: 48.97 | success rate: 98.0% | epsilon: 0.0247
episode 38000/50000 | avg reward: 50.38 | success rate: 98.2% | epsilon: 0.0224
episode 39000/50000 | avg reward: 48.29 | success rate: 98.6% | epsilon: 0.0202
episode 40000/50000 | avg reward: 50.46 | success rate: 98.7% | epsilon: 0.0183
episode 41000/50000 | avg reward: 50.42 | success rate: 98.8% | epsilon: 0.0166
episode 42000/50000 | avg reward: 46.87 | success rate: 98.0% | epsilon: 0.0150
episode 43000/50000 | avg reward: 50.52 | success rate: 98.8% | epsilon: 0.0136
episode 44000/50000 | avg reward: 48.09 | success rate: 98.5% | epsilon: 0.0123
episode 45000/50000 | avg reward: 49.42 | success rate: 98.7% | epsilon: 0.0111
episode 46000/50000 | avg reward: 53.41 | success rate: 98.9% | epsilon: 0.0100
episode 47000/50000 | avg reward: 50.08 | success rate: 98.5% | epsilon: 0.0100
episode 48000/50000 | avg reward: 49.97 | success rate: 98.5% | epsilon: 0.0100
episode 49000/50000 | avg reward: 49.47 | success rate: 98.8% | epsilon: 0.0100
episode 50000/50000 | avg reward: 50.80 | success rate: 99.1% | epsilon: 0.0100
======================================================================
Training completed!

model saved to q_learning_agent.npy

test result (seed 420000-420999):
======================================================================
   avg reward: 33.21
   avg steps: 64.09
   success rate: 83.9%
   evaluation score: 0.7114 (71.14%)
======================================================================
```

Result: Q-Learning showed the best evaluation score amongst other models.

We also noticed that another method to increase the performance is through using action_mask which limit actions taken by the agent based on the rule. In one of the experiment (attached on ./notebook/Assignment3_actionmask_exp.ipynb) showed that the performance reached as follow

```
Episode 45000/50000 | avg reward (last 100): -1.10 | avg steps (last 100): 22.1
Episode 45500/50000 | avg reward (last 100): 0.45 | avg steps (last 100): 20.6
Episode 46000/50000 | avg reward (last 100): 0.65 | avg steps (last 100): 20.4
Episode 46500/50000 | avg reward (last 100): 1.73 | avg steps (last 100): 19.3
Episode 47000/50000 | avg reward (last 100): 1.43 | avg steps (last 100): 19.6
Episode 47500/50000 | avg reward (last 100): 1.75 | avg steps (last 100): 19.2
Episode 48000/50000 | avg reward (last 100): 1.33 | avg steps (last 100): 19.7
Episode 48500/50000 | avg reward (last 100): 1.19 | avg steps (last 100): 19.8
Episode 49000/50000 | avg reward (last 100): 1.26 | avg steps (last 100): 19.7
Episode 49500/50000 | avg reward (last 100): 0.51 | avg steps (last 100): 20.5
Episode 50000/50000 | avg reward (last 100): 0.49 | avg steps (last 100): 20.5
Saved Q-table to q_learning_baseline.npy
```

```
Evaluating learned Q-table...

Q-learning test result:
============================================================
  avg reward: 0.63
  avg steps: 20.33
  success rate: 99.8%
  evaluation score: 0.9183 (91.83%)
============================================================

Done. Metrics:
{'avg_reward': np.float64(0.63), 'avg_steps': np.float64(20.328),
'success_rate': 0.998, 'evaluation_score': np.float64(0.9182880000000001)}
```

However, we didn't include in this experiment to emphasize the reinforcement learning without any intervention.

Algorithm	Avg Reward (Test)	Avg Steps	Success Rate (%)	Evaluation Score
Policy Gradient	-41.29	142.24	31.1	0.2933
Actor-Critic	-136.65	143.08	30.6	0.2889
Q-Learning	**33.21**	**64.09**	**83.9**	**0.7114**

Q-Learning achieved the highest performance across all metrics — high average reward, fewer steps per episode, and a strong success rate (83.9%). Policy Gradient and Actor-Critic both struggled to generalize in test conditions, reflected by low success rates and negative average rewards. Despite Actor-Critic's promising training performance, it suffered during evaluation, suggesting overfitting or unstable critic updates.

These experiments showed the advantage of using Q-Learning in simple discrete problems like in Taxi V3. It also can handle stochasticity emitted from the custom rainy slip that might occur in the problem.

Attachments

Table 4. Grid-search hyperparameter tuning for Q-Learning

#	Learning Rate	Discount Factor	Epsilon Decay Rate	Epsilon Min	Avg Reward	Avg Steps	Success Rate (%)	Evaluation Score
1	0.01	0.9	0.999	0.01	-37.46	190.65	5	0.0474
2	0.01	0.9	0.999	0.05	-33.7	168.17	17	0.1613
3	0.01	0.9	0.9999	0.01	-61.54	154.16	25	0.2334
4	0.01	0.9	0.9999	0.05	-62.8	155.34	24	0.2266
5	0.01	0.9	0.99999	0.01	-57.21	177.39	12	0.1144
6	0.01	0.9	0.99999	0.05	-55.3	173.64	14	0.1334
7	0.01	0.95	0.999	0.01	-42.73	188.74	6	0.057
8	0.01	0.95	0.999	0.05	-34.34	170.11	16	0.1516
9	0.01	0.95	0.9999	0.01	-48.54	137.47	34	0.3181
10	0.01	0.95	0.9999	0.05	-54.45	147.1	30	0.2716
11	0.01	0.95	0.99999	0.01	-56.11	169.93	16	0.1523
12	0.01	0.95	0.99999	0.05	-57.83	167.91	17	0.1624
13	0.01	0.99	0.999	0.01	-31.59	177.53	12	0.1139
14	0.01	0.99	0.999	0.05	-28.99	168.29	17	0.1608
15	0.01	0.99	0.9999	0.01	-53.99	144.33	30	0.2827
16	0.01	0.99	0.9999	0.05	-42.95	149.94	27	0.2542
17	0.01	0.99	0.99999	0.01	-65.9	173.54	14	0.1338
18	0.01	0.99	0.99999	0.05	-59.6	164.18	19	0.1813
19	0.1	0.9	0.999	0.01	-1.05	131.26	39	0.353
20	0.1	0.9	0.999	0.05	-4.39	122.07	45	0.4017
21	0.1	0.9	0.9999	0.01	-47.23	136.2	35	0.3252
22	0.1	0.9	0.9999	0.05	-44.05	139.27	33	0.3089
23	0.1	0.9	0.99999	0.01	-65.41	162.62	20	0.1895
24	0.1	0.9	0.99999	0.05	-58.27	149.64	27	0.2554
25	0.1	0.95	0.999	0.01	0.23	99.63	61	0.5235

26	0.1	0.95	0.999	0.05	-9.55	91.53	63	0.5599
27	0.1	0.95	0.9999	0.01	-54.92	156.02	24	0.2239
28	0.1	0.95	0.9999	0.05	-62.01	153.78	25	0.2349
29	0.1	0.95	0.99999	0.01	-57.23	149.65	27	0.2554
30	0.1	0.95	0.99999	0.05	-58.77	155.47	24	0.2261
31	0.1	0.99	0.999	0.01	15.18	66.95	86	0.7042
32	0.1	0.99	0.999	0.05	8.12	107.19	55	0.4812
33	0.1	0.99	0.9999	0.01	-49.99	141.42	32	0.2983
34	0.1	0.99	0.9999	0.05	-56.08	147.06	29	0.2698
35	0.1	0.99	0.99999	0.01	-55.54	166.54	18	0.1698
36	0.1	0.99	0.99999	0.05	-57.59	166.32	18	0.1707
37	0.2	0.9	0.999	0.01	-3.65	131.54	39	0.3518
38	0.2	0.9	0.999	0.05	-9.7	141.96	33	0.2982
39	0.2	0.9	0.9999	0.01	-36.27	133.9	36	0.3364
40	0.2	0.9	0.9999	0.05	-37.73	137.97	34	0.3161
41	0.2	0.9	0.99999	0.01	-66.79	171.86	15	0.1426
42	0.2	0.9	0.99999	0.05	-52.77	155.48	24	0.2261
43	0.2	0.95	0.999	0.01	-5.21	103.78	58	0.5009
44	0.2	0.95	0.999	0.05	-15.65	108.44	54	0.4742
45	0.2	0.95	0.9999	0.01	-37.94	146.83	29	0.2707
46	0.2	0.95	0.9999	0.05	-55.52	154.44	27	0.2362
47	0.2	0.95	0.99999	0.01	-67.09	173.88	14	0.1325
48	0.2	0.95	0.99999	0.05	-59.76	166.56	18	0.1698
49	0.2	0.99	0.999	0.01	8.09	75.3	79	0.6568
50	0.2	0.99	0.999	0.05	-15.71	113.02	53	0.4539

Actor Critic Agent

```
class ActorCriticAgent:

    def __init__(self, n_states, n_actions,
                 learning_rate=0.01,
                 value_lr=0.1,
                 lr_decay=0.9999,
                 lr_min=0.0001,
                 discount_factor=0.99,
                 entropy_coef=0.01):
        self.n_states = n_states
        self.n_actions = n_actions
        self.lr = learning_rate
        self.lr_init = learning_rate
        self.value_lr = value_lr
        self.lr_decay = lr_decay
        self.lr_min = lr_min
        self.gamma = discount_factor
        self.entropy_coef = entropy_coef

        # policy parameters (Actor)
        self.theta = np.zeros((n_states, n_actions))

        # Value function (Critic)
        self.V = np.zeros(n_states)

        # statistics information
        self.episode_count = 0

    def get_policy(self, state):
        """calculate action probability distribution (softmax)"""
        theta_state = self.theta[state] - np.max(self.theta[state])
        exp_theta = np.exp(theta_state)
        return exp_theta / np.sum(exp_theta)

    def choose_action(self, state):
        """sample action according to policy"""
        policy = self.get_policy(state)
        return np.random.choice(self.n_actions, p=policy)
```

```
    def update(self, episode_history):
        """
        update policy using Advantage and Entropy
        """
        if len(episode_history) == 0:
            return

        # Actor-Critic Update
        advantages = []
        for t in range(len(episode_history)):
            state, action, reward = episode_history[t]
            next_state = episode_history[t+1][0] if t + 1 <
len(episode_history) else state # Assume next state is current state if
episode ends
            terminated = t + 1 == len(episode_history) # Check if current step
is the last step

            # Calculate TD Target
            td_target = reward + self.gamma * self.V[next_state] * (1 -
terminated)

            # Calculate TD Error
            td_error = td_target - self.V[state]

            # Update Value function (Critic)
            self.V[state] += self.value_lr * td_error

            # Calculate Advantage (TD Error)
            advantages.append(td_error)

        # Standardize Advantage
        advantages = np.array(advantages)
        if len(advantages) > 1:
            advantages = (advantages - np.mean(advantages)) /
(np.std(advantages) + 1e-9)


        # Update policy parameters (Actor)
        for t, (state, action, reward) in enumerate(episode_history):
            policy = self.get_policy(state)
```

```
            # Policy gradient
            grad = np.zeros(self.n_actions)
            grad[action] = 1.0
            grad -= policy

            # Entropy gradient
            entropy_grad = -np.log(policy + 1e-9) - 1

            # combine update
            total_grad = (advantages[t] * grad +
                          self.entropy_coef * entropy_grad)

            self.theta[state] += self.lr * total_grad

        # Decay learning rate
        self.decay_learning_rate()
        self.episode_count += 1

    def decay_learning_rate(self):
        """gradually decrease learning rate"""
        self.lr = max(self.lr_min, self.lr * self.lr_decay)

    @property
    def Q(self):
        """compatible test function"""
        return self.theta
```

Q Learning Agent

```
class QLearningAgent:
    def __init__(self, n_states, n_actions, learning_rate=0.1,
discount_factor=0.99, epsilon=1.0, epsilon_decay_rate=0.999,
epsilon_min=0.01):
        self.n_states = n_states
        self.n_actions = n_actions
        self.lr = learning_rate
        self.gamma = discount_factor
        self.epsilon = epsilon
        self.epsilon_decay_rate = epsilon_decay_rate
        self.epsilon_min = epsilon_min
        self.q_table = np.zeros((n_states, n_actions))
```

```
    def choose_action(self, state, env):
        """Choose action using epsilon-greedy policy"""
        if np.random.rand() < self.epsilon:
            # Explore: choose a random action
            return np.random.choice(self.n_actions)
        else:
            # Exploit: choose the action with the highest Q-value
            return np.argmax(self.q_table[state])

    def update(self, state, action, reward, next_state):
        """Update Q-table using the Q-learning update rule"""
        best_next_action = np.argmax(self.q_table[next_state])
        td_target = reward + self.gamma *
self.q_table[next_state][best_next_action]
        td_error = td_target - self.q_table[state][action]
        self.q_table[state][action] += self.lr * td_error

    def decay_epsilon(self):
        """Decay epsilon"""
        self.epsilon = max(self.epsilon_min, self.epsilon *
self.epsilon_decay_rate
```

Action Mask snippet

```
def train_q_learning(env, n_episodes=2000, alpha=0.1, gamma=0.99, epsilon=0.1,
max_steps=200, seed_start=0, verbose=True):
    """Train a tabular Q-learning agent using action masks from the env.
    Returns Q-table (n_states x n_actions) and training stats.
    """
    n_states = env.observation_space.n
    n_actions = env.action_space.n
    Q = np.zeros((n_states, n_actions), dtype=float)

    episode_rewards = []
    steps_per_episode = []
    successes = 0

    for ep in range(n_episodes):
        state, info = env.reset(seed=seed_start + ep)
        total_reward = 0.0
```

```
        for t in range(max_steps):
            mask = info.get('action_mask', np.ones(n_actions, dtype=int))
            if np.random.random() < epsilon:
                action = masked_random_choice(mask)
            else:
                action = masked_argmax(Q[state], mask)

            next_state, reward, terminated, truncated, info = env.step(action)
            done = terminated or truncated

            next_mask = info.get('action_mask', np.ones(n_actions, dtype=int))
            if next_mask.sum() == 0:
                next_max = np.max(Q[next_state])
            else:
                next_max = np.max(np.where(next_mask, Q[next_state], -np.inf))
                if np.isneginf(next_max):
                    next_max = np.max(Q[next_state])

            td_target = reward + gamma * next_max * (0 if done else 1)
            td_error = td_target - Q[state, action]
            Q[state, action] += alpha * td_error

            total_reward += reward
            state = next_state

            if done:
                if terminated and reward == env.reward_delivery:
                    successes += 1
                break

        episode_rewards.append(total_reward)
        steps_per_episode.append(t + 1)

        if verbose and (ep + 1) % 500 == 0:
            print(f"Episode {ep+1}/{n_episodes} | avg reward (last 100):
{np.mean(episode_rewards[-100:]):.2f} | avg steps (last 100):
{np.mean(steps_per_episode[-100:]):.1f}")

    stats = {
        'episode_rewards': episode_rewards,
        'steps_per_episode': steps_per_episode,
        'successes': successes,
```

```
    }
    return Q, stats


def test_q_learning(Q, env, n_episodes=200, seed_start=0, max_steps=200,
verbose=True):
    """Evaluate a Q-table deterministically using masked argmax. Returns
metrics dict."""
    n_actions = env.action_space.n
    rewards = []
    steps = []
    successes = 0

    for ep in range(n_episodes):
        state, info = env.reset(seed=seed_start + ep)
        ep_reward = 0.0
        for t in range(max_steps):
            mask = info.get('action_mask', np.ones(n_actions, dtype=int))
            action = masked_argmax(Q[state], mask)
            next_state, reward, terminated, truncated, info = env.step(action)
            ep_reward += reward
            if terminated or truncated:
                if terminated and reward == env.reward_delivery:
                    successes += 1
                break
            state = next_state
        rewards.append(ep_reward)
        steps.append(t + 1)

    avg_reward = np.mean(rewards)
    avg_steps = np.mean(steps)
    success_rate = successes / n_episodes
    normalized_steps = avg_steps / max_steps
    step_score = 1 - normalized_steps
    evaluation_score = success_rate * 0.2 + step_score * 0.8

    metrics = {
        'avg_reward': avg_reward,
        'avg_steps': avg_steps,
        'success_rate': success_rate,
        'evaluation_score': evaluation_score,
    }
```

```
    if verbose:
        print("\nQ-learning test result:")
        print("="*60)
        print(f"  avg reward: {avg_reward:.2f}")
        print(f"  avg steps: {avg_steps:.2f}")
        print(f"  success rate: {success_rate:.1%}")
        print(f"  evaluation score: {evaluation_score:.4f}
({evaluation_score*100:.2f}%)")
        print("="*60)
    return metrics
```

[redacted]