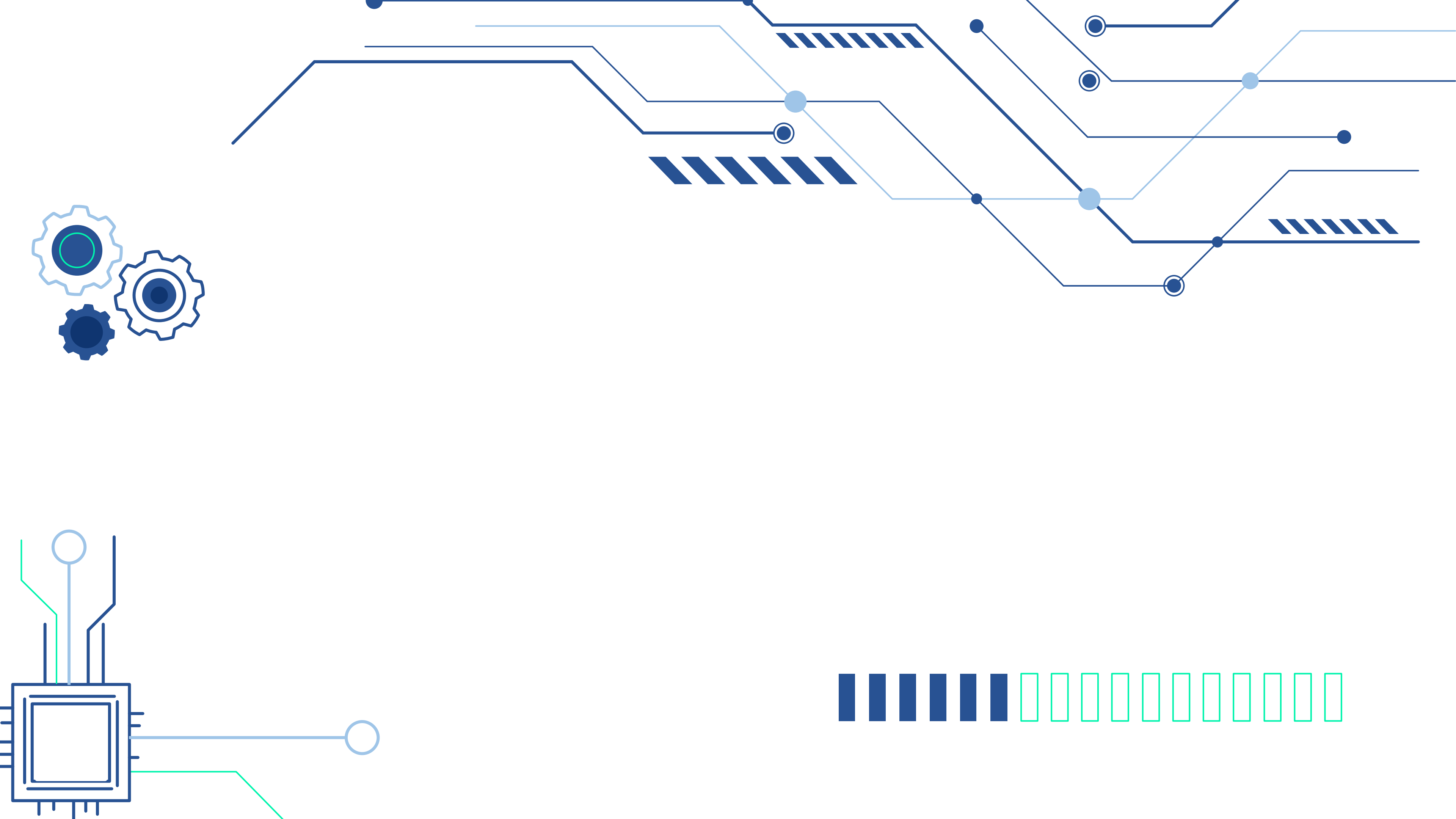


REINFORCEMENT LEARNING

By: Masoud Pourghavam

Professor: Dr. M. S. Panahi

July 2023

TABLE OF CONTENTS

CONTENTS

QUESTIONS

- **Off-policy & on-policy**
  - Differences
  - Applications
- **Model-free**
  - Challenges
  - Solutions
- **Reward**
  - Issue
  - Suggestions

QUESTIONS

Village

- MDP
  - Action spaces
    - Participate
    - Abstain
  - States
    - Healthy
    - Sick
  - State transitions
    - No state trans.
    - Disease trans. prob.
  - Rewards
    - R(s, *, *) = 0
    - R(h, p, s) = -100
    - R(h, p, h) = 50
    - R(h, a, h) = -10
- Algorithms
- Value-iteration

QUESTIONS

Algorithms

- Q-Learning
- Policy Gradient Methods
- Deep Q-Network (DQN)

- N_h (Number of healthy people) = 8
- N_s (Number of sick people) = 2
- a (Transmission factor) = 2
- γ (Discount factor) = 0.9

2. For each state s in S:

For each action a in A:

Calculate the expected value of taking action a in state s:

- If s = Sick:

  V'(Sick) = R(Sick, *, *) + γ * 0 (No reward for being sick)

- If s = Healthy and a = Participate:

  V'(Healthy) = max { R(Healthy, Participate, Sick) + γ * V(Sick), R(Healthy, Participate, Healthy) + γ * V(Healthy) }

- If s = Healthy and a = Abstain:

  V'(Healthy) = R(Healthy, Abstain, Healthy) + γ * V(Healthy)

Update the value function for state s:

V(s) = max { V'(Healthy), V'(Sick) }

QUESTIONS

Value-Iteration

- **Initialization**: V(sick, healthy) = (0, 0)
- **Iteration 1**: V(sick, healthy) = (50, 50)
- **Iteration 2**: V(sick, healthy) = (95, 95)
- **Iteration 3**: V(sick, healthy) = (135.5, 135.5)

CODES EXPLANATION

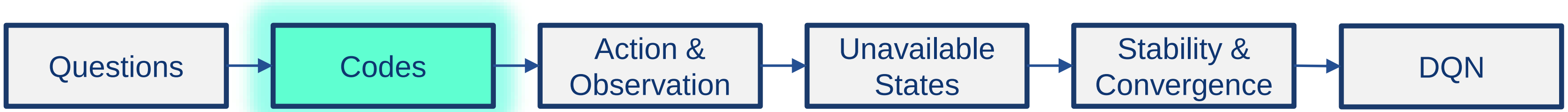
```
class RLAgent:
    def __init__(self, env):
        self.env = env
        self.state_size = env.observation_space.n
        self.action_size = env.action_space.n

    def act(self, state):
        return np.random.choice(self.action_size)

    def learn(self, episode):
        pass
```

CODES EXPLANATION

```
    def learn(self, episode):
        states, actions, rewards = zip(*episode)
        G = 0
        for t in reversed(range(len(episode))):
            state = states[t]
            action = actions[t]
            reward = rewards[t]
            G = self.gamma * G + reward
            self.state_action_counts[state][action] += 1
            alpha = 1 / self.state_action_counts[state][action]
            self.Q[state][action] = (1 - alpha) * self.Q[state][action]
+ alpha * G
```

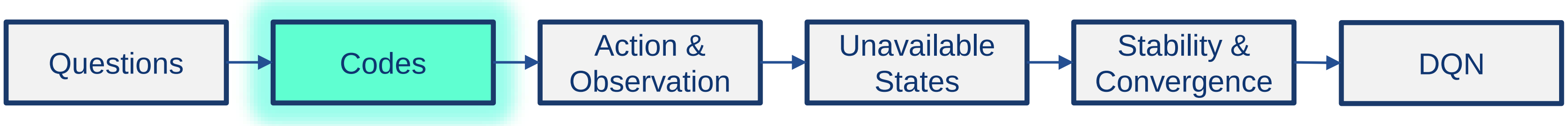


```
    def train(self, episodes=2000, max_steps=500, num_runs=10):
        self.episodes = episodes
        total_penalties = np.zeros(num_runs)
        total_rewards = np.zeros(num_runs)
        average_rewards = np.zeros((num_runs, episodes))

        min_abs_avg_reward = float('inf')
        min_abs_avg_reward_episode = None

        for run in range(num_runs):
            for episode_num in range(episodes):
                env.seed(seed=44 + run)
                state = self.env.reset()
                episode = []
                penalties, rewards = 0, 0
                steps = 0

                for step in range(max_steps):
                    action = self.act(state)
                    next_state, reward, done, info =
self.env.step(action)
                    if reward == -10:
                        penalties += 1
                    rewards += reward
                    episode.append((state, action, reward))
                    state = next_state
                    steps += 1
```



ACTION & OBSERVATION SPACES

Action Space → **Discrete(6)**

Observation Space → **Discrete(500)**

Questions → Codes → Action & Observation → Unavailable States → Stability & Convergence → DQN

ACTION & OBSERVATION SPACES

```
State Grid:
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Actions

- **Move south: 0**
- **Move north: 1**
- **Move east: 2**
- **Move west: 3**
- **Pickup passenger: 4**
- **Drop off passenger: 5**

Questions → Codes → **Action & Observation** → Unavailable States → Stability & Convergence → DQN

ACTION & OBSERVATION SPACES

```
State Grid:
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Passenger Locations

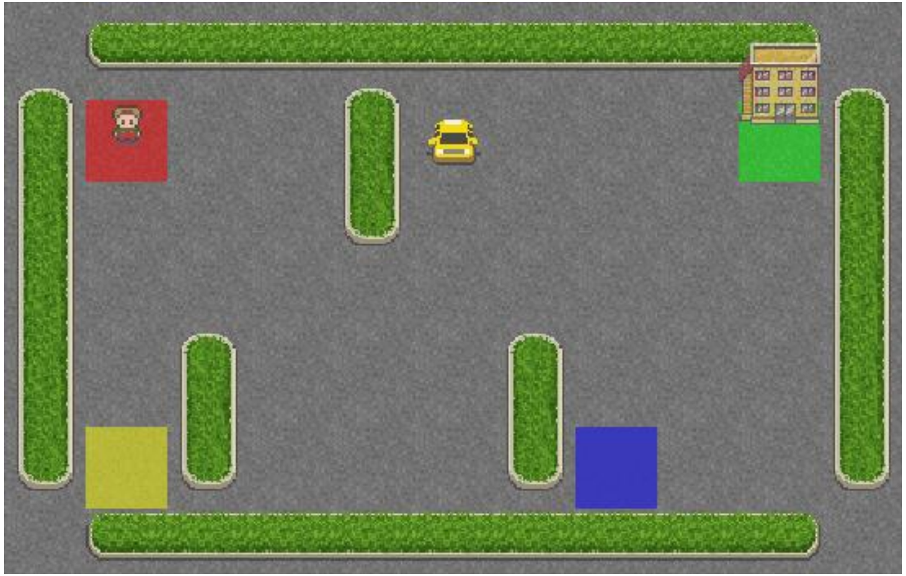
- Red: 0
- Green: 1
- Yellow: 2
- Blue: 3
- In Taxi: 4

Passenger Locations

- Red: 0
- Green: 1
- Yellow: 2
- Blue: 3

Questions → Codes → Action & Observation → Unavailable States → Stability & Convergence → DQN

UNAVAILABLE STATES



- **Wall constraints**
- **Passenger constraints**
- **Destination constraints**
- **Illegal actions**

Questions → Codes → Action & Observation → **Unavailable States** → Stability & Convergence → DQN

STABILITY AND CONVERGENCE

- Constant learning rate and discount factor = 0.999
- Learning rate decay and discount factor = 0.999
- Constant learning rate and discount factor = 0.9
- Learning rate decay and discount factor = 0.9

Table 2. Parameters of the 1st case for monte carlo

Parameter	Value
Learning rate (alpha)	0.1
Learning rate decay (alpha decay)	1
Discount factor (gamma)	0.999
Epsilon	1.0
Epsilon decay	0.99
Episodes	2000
Max Steps	500
Number of runs	10

STABILITY AND CONVERGENCE

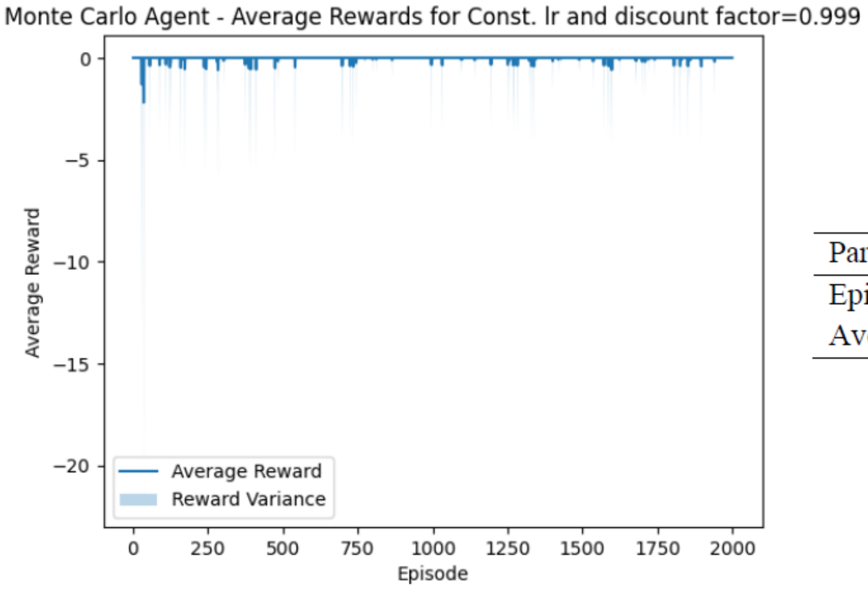


Figure 5. Average rewards in terms of episodes for Monte Carlo - Constant learning rate and discount factor = 0.999

Table 3. Monte Carlo algorithm results for 1st case

Parameter	Value
Episode with min. abs. avg. reward	1002
Average penalties per episode	0.0735

Questions → Codes → Action & Observation → Unavailable States → **Stability & Convergence** → DQN

STABILITY AND CONVERGENCE

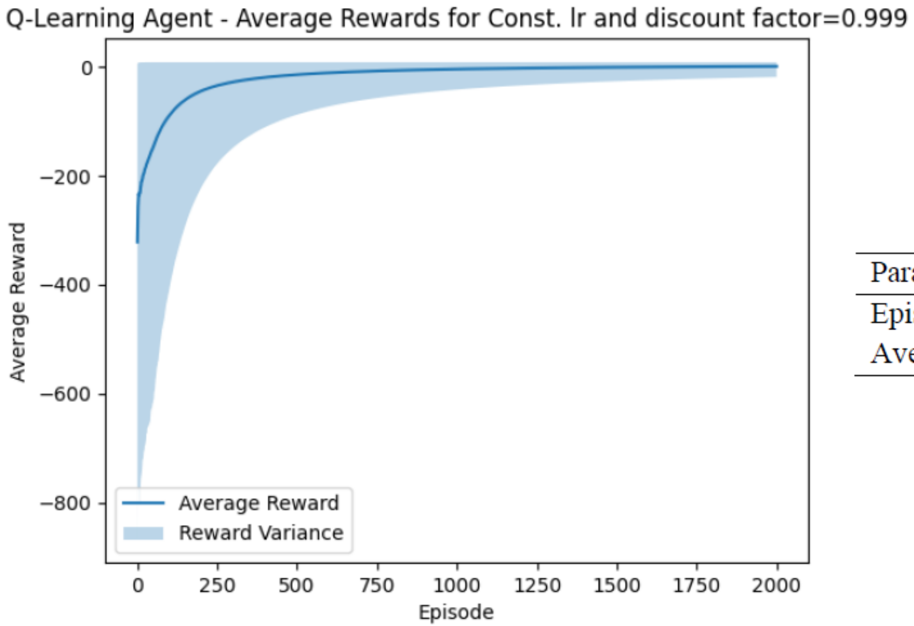


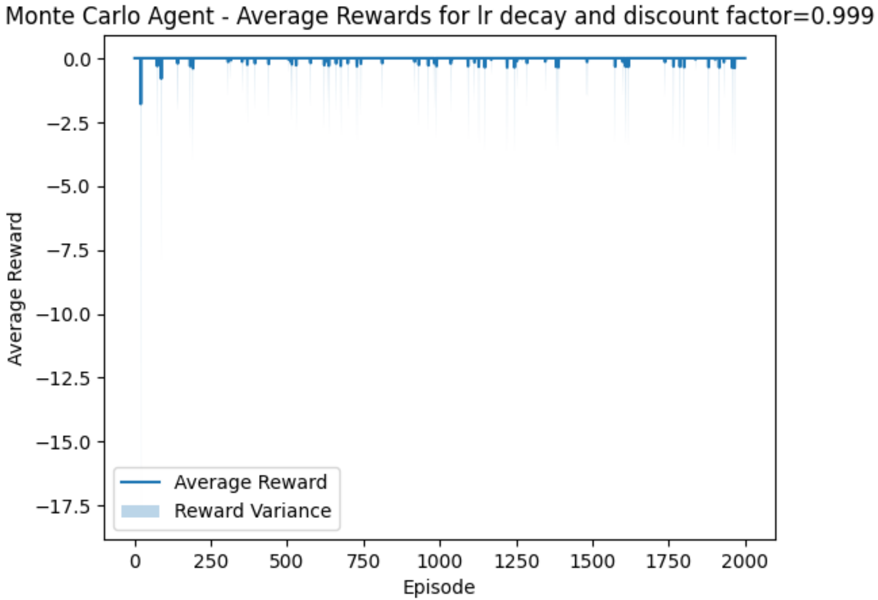
Figure 6. Average rewards in terms of episodes for Q-Learning - Constant learning rate and discount factor = 0.999

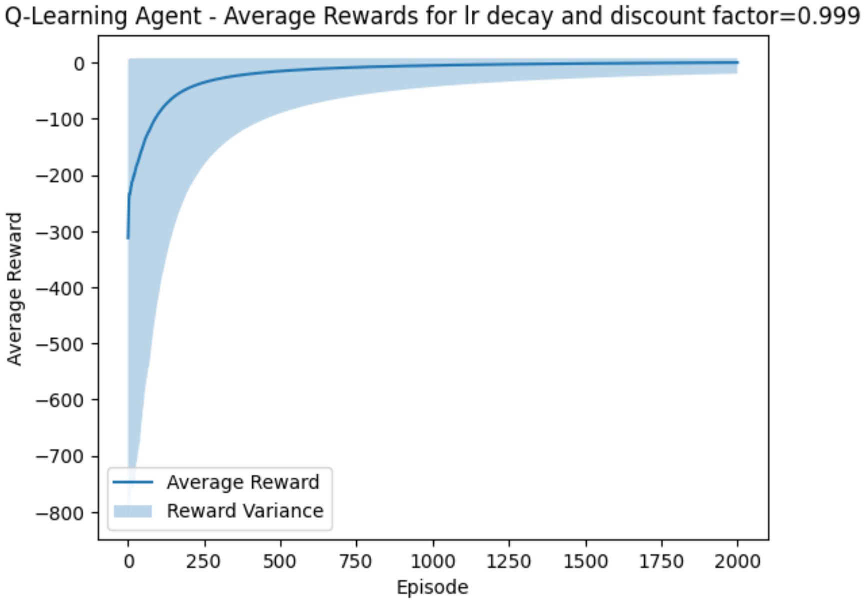
Table 5. Q-Learning algorithm results for 1st case

Parameter	Value
Episode with min. abs. avg. reward	271
Average penalties per episode	0.016

Questions → Codes → Action & Observation → Unavailable States → Stability & Convergence → DQN

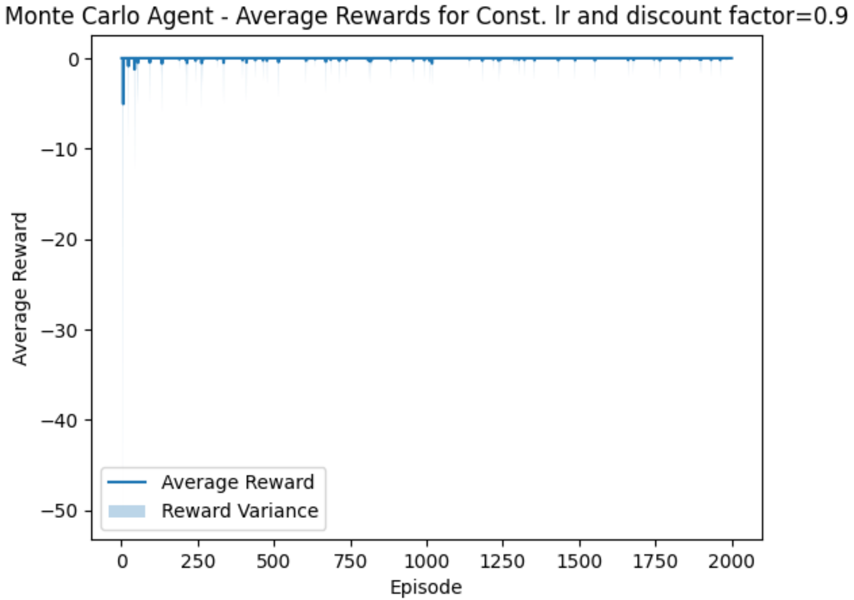
STABILITY AND CONVERGENCE

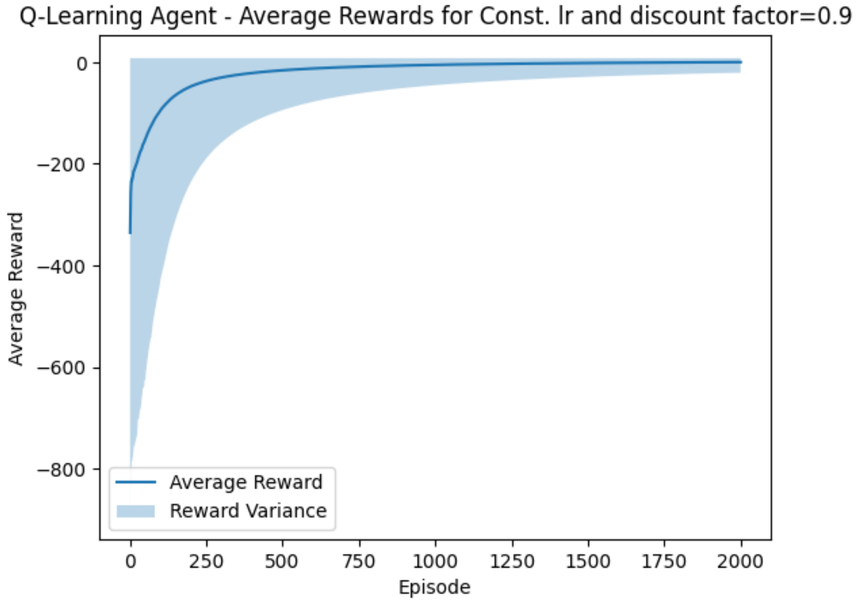




Questions → Codes → Action & Observation → Unavailable States → **Stability & Convergence** → DQN

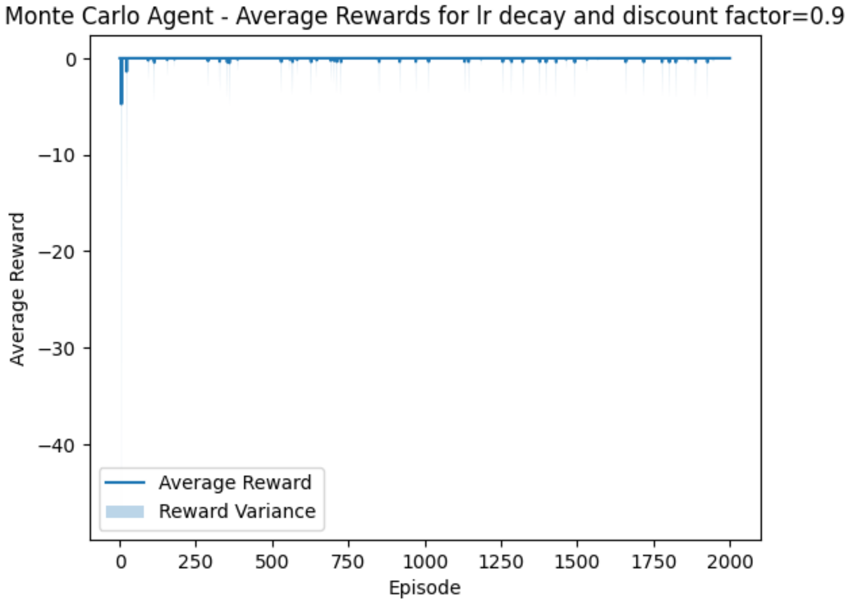
STABILITY AND CONVERGENCE

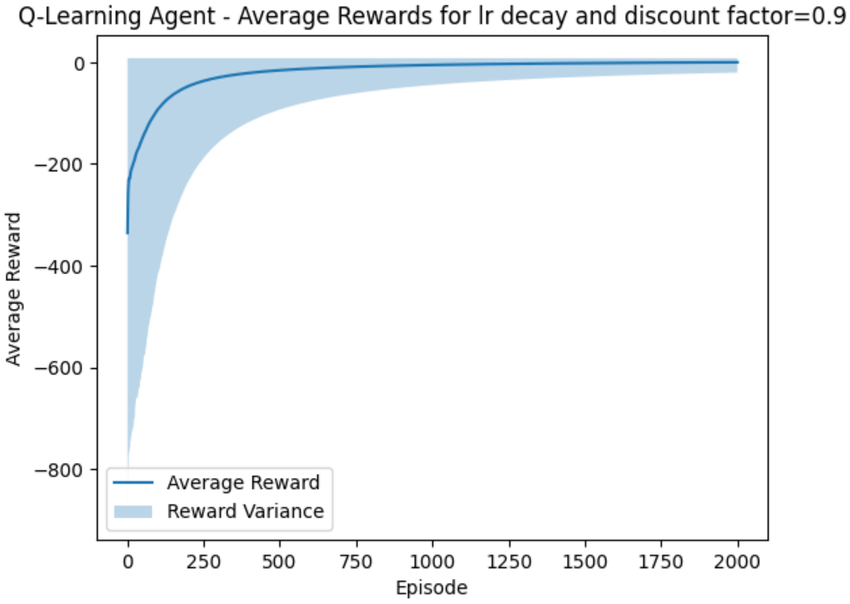




Questions → Codes → Action & Observation → Unavailable States → **Stability & Convergence** → DQN

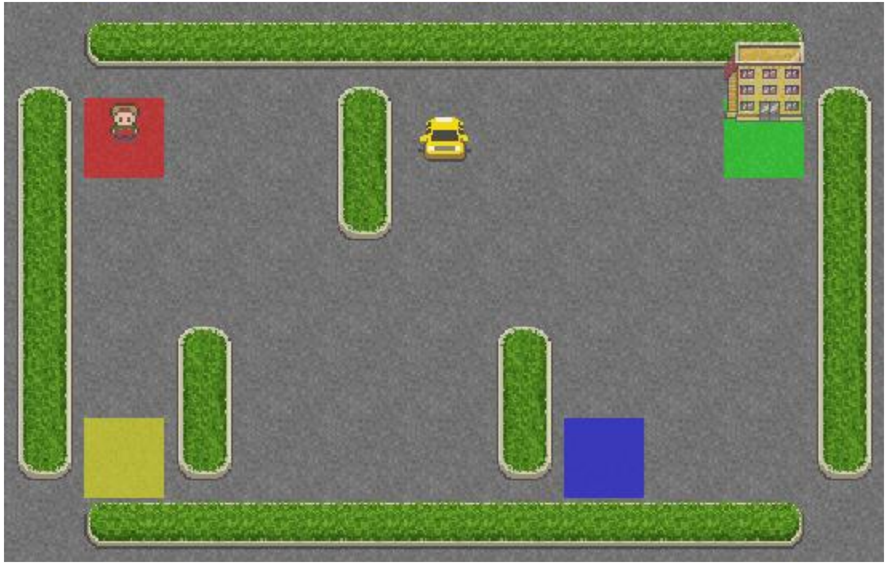
STABILITY AND CONVERGENCE





Questions → Codes → Action & Observation → Unavailable States → **Stability & Convergence** → DQN

RENDERING



Questions → Codes → Action & Observation → Unavailable States → **Stability & Convergence** → DQN

DEEP Q-LEARNING

Table 18. Parameters of the deep q-learning algorithm

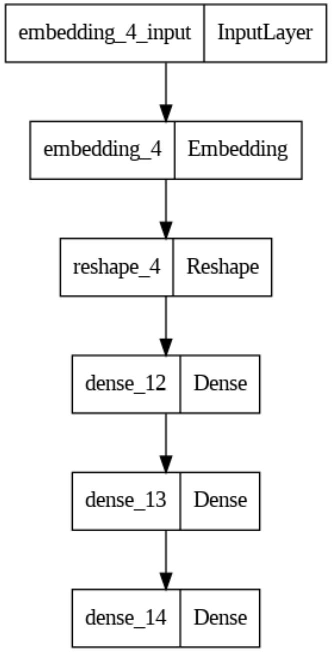
Parameter	Value
Learning rate (alpha)	0.1
Maxlen	1.0
Discount factor (gamma)	0.6
Epsilon	1.0
Epsilon decay	0.99
Episodes	1000
Epsilon min	0.01
Batch size	32
Time steps per episode	2
Loss function	MSE
Hidden layers activation	Relu
Last layer activation	Linear
Optimizer	Adam

Questions → Codes → Action & Observation → Unavailable States → Stability & Convergence → DQN

DEEP Q-LEARNING

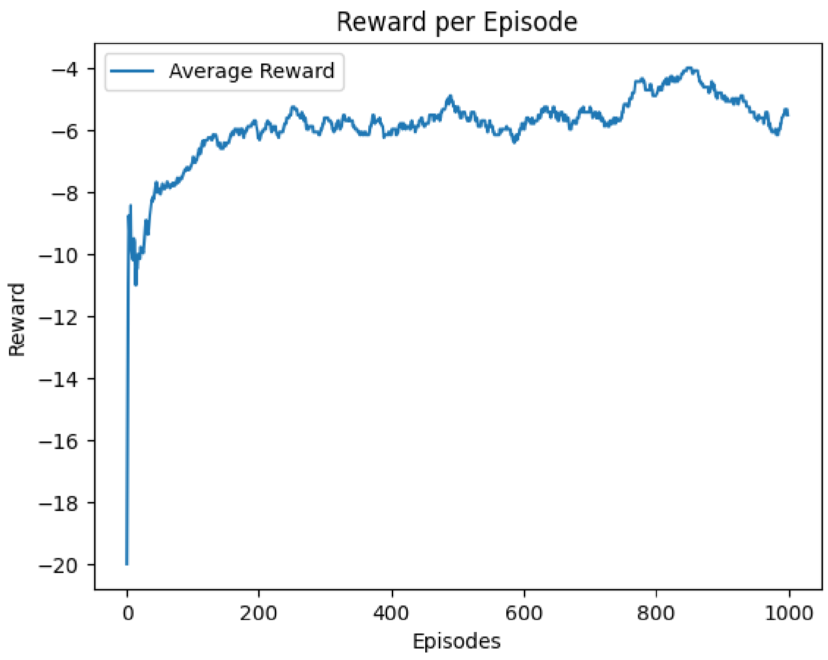
```
    def _build_compile_model(self):     # Build and compile the Q-
Network
        model = Sequential()
        model.add(Embedding(self._state_size, 10, input_length=1))
        model.add(Reshape((10,)))
        model.add(Dense(65, activation='relu'))
        model.add(Dense(65, activation='relu'))
        model.add(Dense(self._action_size, activation='linear'))

        model.compile(loss='mse', optimizer=self._optimizer)
        return model
```



Questions → Codes → Action & Observation → Unavailable States → Stability & Convergence → **DQN**

DEEP Q-LEARNING



Questions → Codes → Action & Observation → Unavailable States → Stability & Convergence → DQN

THANK YOU!