МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-наукового інституту атомної і теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ЗВІТ

ДО ВИКОНАННЯ ПРАКТИЧНОГО ЗАВДАННЯ №4

з дисципліни

«МЕТОДОЛОГІЯ РОЗРОБКИ ІНТЕЛЕКТУАЛЬНИХ КОМП'ЮТЕРНИХ ПРОГРАМ »

Тема: «Інтелектуальні агенти. Алгоритм Q-навчання»

Варіант 16(6)

КИЇВ 2025

Тема: Інтелектуальні агенти. Алгоритм Q-навчання

Мета: Ознайомитися з поняттям інтелектуального агента та одним з методів його навчання – Q-learning.

Завдання: Побудувати математичну модель інтелектуального агента та його зовнішнього середовища. Проілюструвати роботу алгоритму Q-навчання розрахунками без програмної реалізації. Для цього показати 2 спроби агента досягти цільової мети. Перший раз, обираючи маршрут цілком довільним чином (так як пам'ять агента порожня), другий раз – враховуючи здобуту пам'ять агента. Розрахунки повинні містити обчислення винагород за дії агента, значення матриці Q та зображення графа зовнішнього середовища з позначенням шляху агента і змінених значень ваг ребер. Відповідно до свого варіанта та побудованої в 1 завданні математичної моделі реалізувати програмне забезпечення для ілюстрації роботи алгоритму Q-навчання.

Опис дії алгоритму

Завдання 1

Відповідно до нашого варіанту розмір поля буде 5x5. Множину станів нашого агента можна описати у вигляді формули нижче:

$$S = \{(x, y) \mid x, y = [0, 4]\}$$

Цільовим станом(координатою) нашого агента є клітинка з індексами (0, 0)

Пронумеруємо наші клітинки від 0 до 24:

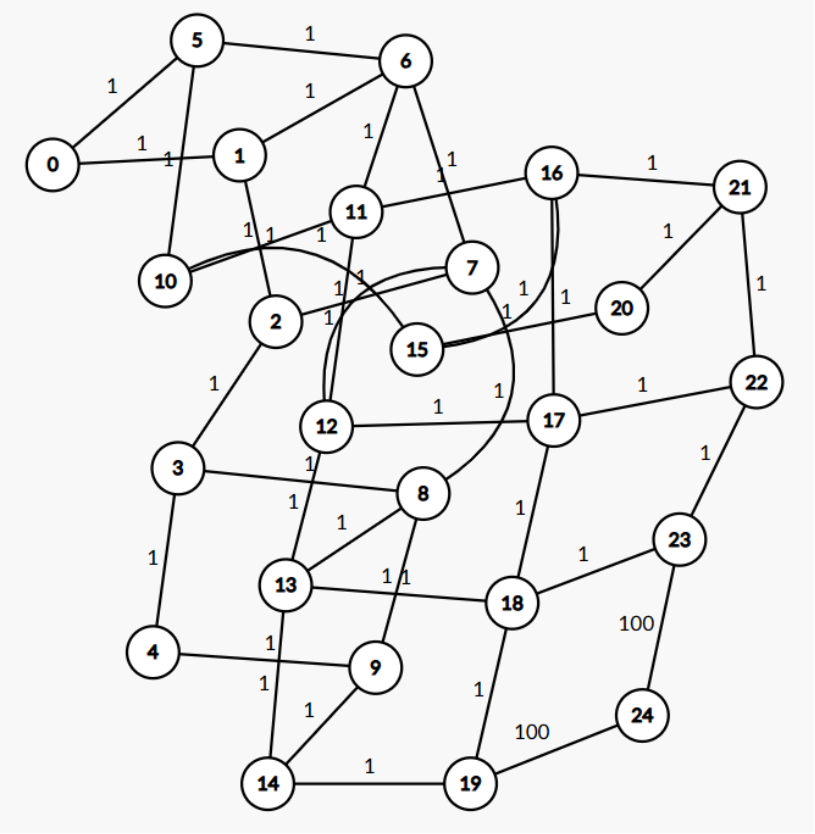
0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Відповідно до нашої таблички побудуємо матрицю суміжності:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	100	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	100	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	100	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1
12	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1

13	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	-1
14	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1
15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1
16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1	-1
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1	-1
18	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1	-1	0	-1
19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	-1	-1	-1	-1	0
20	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1
21	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1	-1
22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0	-1
23	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1	0
24	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	0	-1

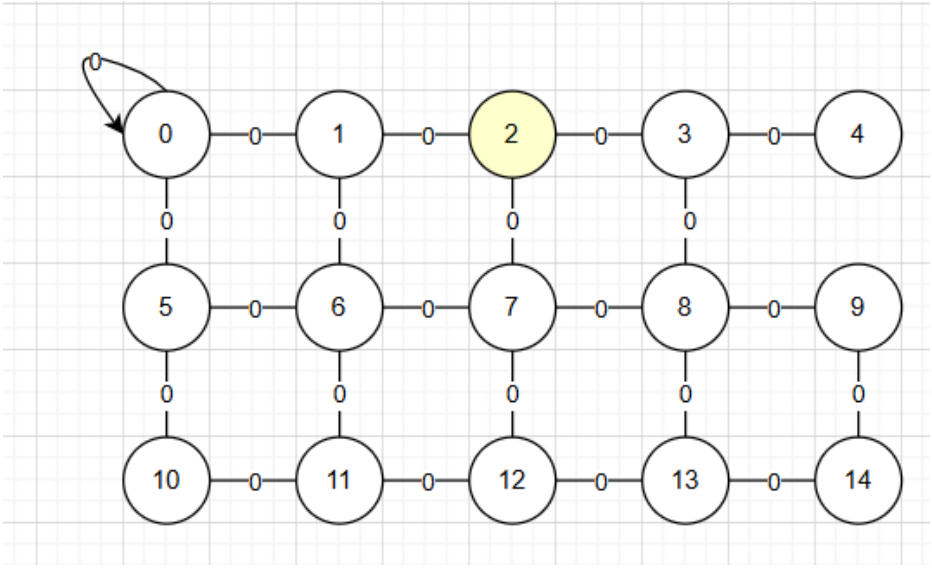
Даний граф показує кількість станів агента та переходи між ними:



Моделювання роботи алгоритму

Щоб зберегти чіткість візуалізацій та показати процес навчання максимально наглядно, початкове положення агента вибирається максимально близько до цільової клітинки. Таким чином, на графі будуть зображені лише ті стани та переходи між ними, які зустрічаються на всіх кроках проходження маршруту.

Нехай агент опинився в клітинці під номером 2(0, 2)



З цієї клітинки агент може перейти до клітинок 1, 3 або 7. Оскільки на початку невідомо, яка дія наблизить його до цілі, він вибирає наступний крок випадковим чином. Припустимо, він обрав перехід до клітинки 1 (дія зі стану 2 у стан 1). У класичному алгоритмі Q-навчання винагорода за виконану дію оновлюється за наступною формулою:

$$Q^{new}(s_t, a_t) = Q(s_t, a_t) + \alpha * (r_t + \gamma * maxQ(s_{t+1}, a) - Q(s_t, a_t))$$

Візьмемо α = 0.1 (швидкість навчання) та γ = 0.8 (коефіцієнт дисконтованої винагороди, який визначає, наскільки важлива майбутня винагорода порівняно з поточною). Обчислимо оновлене значення за формулою:

$$Q[2, 1] = 0 + 0.1 * (0 + 0.8 * 0 - 0) = 0$$

Поки агент не досягне цілі, під час першого проходу матриця Q фактично не оновлюється (окрім останнього кроку, коли ціль досягнута). З клітинки 1 доступні три можливі дії: перейти в клітинки (1, 0), (1, 2) та (1, 6). Оскільки перехід до цільової клітинки 0 має найбільшу вагу, інтелектуальний агент обирає саме його. Після цього переходу розрахуємо оновлене значення в матриці Q:

$$Q[1, 0] = 0 + 0.1 * (100 + 0.8 * 0 - 0) = 10$$

І врешті оновимо матрицю за дію перебування на клітинці 0:

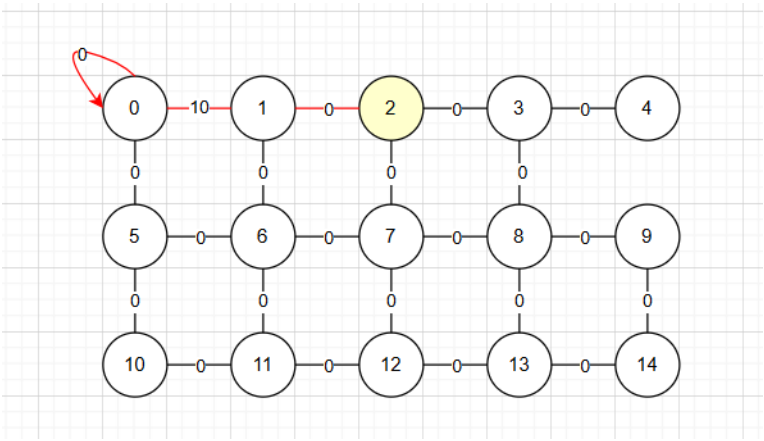
$$Q[0, 0] = 0 + 0.1 (100 + 0.8 * 0 - 0) = 10$$

Фрагмент оновленої матриці Q (початковий варіант був з повним заповненням 0):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

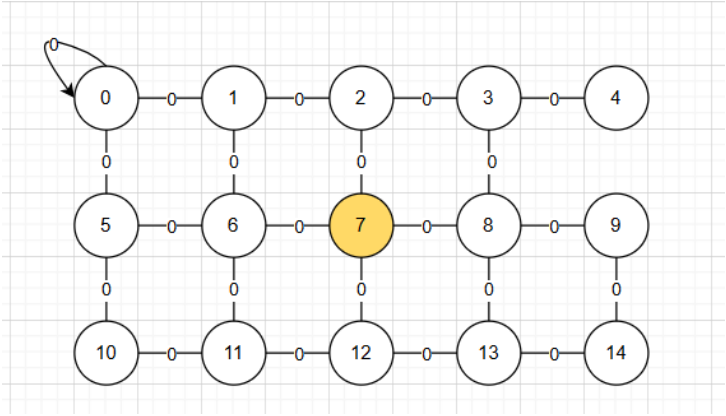
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Кінцевий варіант з виділеними пройденими станами, переходами, і змінами значень матриці Q:



Другий прохід:

Випадково вибрано стартове положення на клітинці 7.



Перехід звідти можливий на клітинки 2, 6, 8, 12. Серед значень рядка 7 матриці Q маємо усі 0, тому обираємо дію випадково. Скажімо, агент переходить на 6 клітинку.

$$Q[7, 6] = 0 + 0.1 * (0 + 0.8 * 0 - 0) = 0$$

Знаходячись у клітинці 6, агент має чотири можливі варіанти руху: перейти у клітинки 1, 5, 7 або 11. Як і раніше, вибір наступної клітинки здійснюється випадковим чином. Припустимо, що агент обирає перейти до клітинки 1:

$$Q[6, 1] = 0 + 0.1 * (0 + 0.8 * 10 * 0 - 0) = 0.8$$

А от уже з клітинки 1 ІА цілеспрямовано переходить до 0 клітинки, бо саме такий перехід має найбільше значення за матрицею Q:

$$Q[1,0] = 10 + 0.1 * (100 + 0.8 * 10 - 10) = 19.8$$

І наостанок оновимо елемент (0, 0) матриці Q - перебування у фінальному стані:

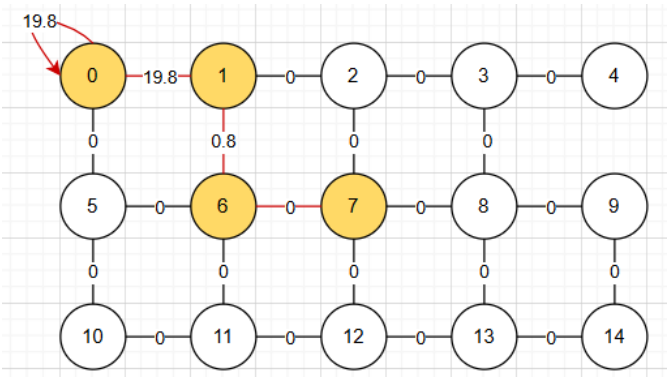
$$Q[0, 0] = 10 + 0.1 * (100 + 0.8 * 10 - 10) = 19.8$$

Оновлений варіант частини матриці Q

Матриця Q матиме наступний вигляд:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	19.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	19.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Другий прохід буде виглядати наступним чином:



Лістинг програми

```
import numpy as np

import matplotlib.pyplot as plt
```

```
class QLearningAgent:
    def __init__(self, dim=5, goal=0, start=24, alfa=0.1, gamma=0.8, epsilon=0.2, episodes=500):
        self.dim = dim           # Розмір сітки (dim x dim)
        self.goal = goal         # Цільова клітинка (індекс)
        self.start = start       # Початкова клітинка (індекс)
        self.alfa = alfa         # Швидкість навчання (learning rate)
        self.gamma = gamma       # Коефіцієнт дисконтної винагороди
        self.epsilon = epsilon   # Імовірність вибору випадкової дії (exploration)
        self.episodes = episodes # Кількість епізодів тренування

        # Побудова матриці винагород (R) та ініціалізація Q-матриці
        self.R = self._build_R()
        self.Q = np.zeros_like(self.R, dtype=float)  # Матриця оцінок Q, спочатку нулі

    def _state2coord(self, s):
        return divmod(s, self.dim)

    def _coord2state(self, i, j):
        return i * self.dim + j

    def _available_actions(self, state):
        # Повертає список допустимих переходів (дій) із заданого стану
        i, j = self._state2coord(state)
        actions = []
        # Перевіряємо 4 напрямки: вгору, вниз, вліво, вправо
        for di, dj in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            ni, nj = i + di, j + dj
            # Якщо координати в межах сітки, додаємо дію
            if 0 <= ni < self.dim and 0 <= nj < self.dim:
                actions.append(self._coord2state(ni, nj))
        return actions

    def _build_R(self):
```

```
    # Створення матриці винагород R
    size = self.dim * self.dim
    R = np.full((size, size), -1)  # Спочатку всі переходи -1 (негативна винагорода)
    for s in range(size):
        for a in self._available_actions(s):
            R[s, a] = -1            # Дозволені переходи отримують -1 (рух)
    # Встановлюємо позитивну винагороду за перехід у цільову клітинку
    for a in self._available_actions(self.goal):
        R[a, self.goal] = 100
    return R

def train(self):
    for _ in range(self.episodes):
        # Випадковий початковий стан
        state = np.random.randint(0, self.dim * self.dim)
        while state == self.goal:
            state = np.random.randint(0, self.dim * self.dim)

        # Поки не досягнемо цілі, робимо кроки
        while state != self.goal:
            actions = self._available_actions(state)  # Доступні дії

            # Вибір дії: випадкова з ймовірністю epsilon (exploration)
            if np.random.rand() < self.epsilon:
                next_state = np.random.choice(actions)
            else:
                # Вибір кращої дії згідно з Q (exploitation)
                qs = [self.Q[state, a] for a in actions]
                max_q = max(qs)
                best = [a for a, q in zip(actions, qs) if q == max_q]
                next_state = np.random.choice(best)

            # Отримуємо винагороду за перехід
```

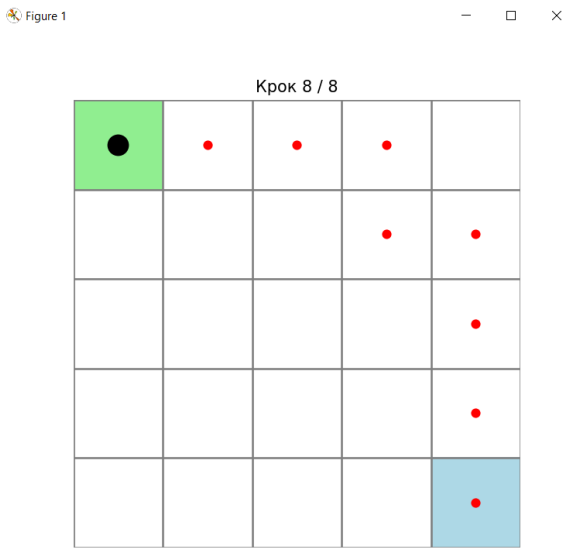
```
            reward = self.R[state, next_state]
            old_q = self.Q[state, next_state]

            # Оновлення Q-значення за формулою
            self.Q[state, next_state] = old_q + self.alfa * (
                reward + self.gamma * np.max(self.Q[next_state]) - old_q)

            # Переходимо до наступного стану
            state = next_state

    def get_optimal_path(self):
        # Відновлення оптимального шляху із Q-матриці
        state = self.start
        path = [state]
        visited = set()

        while state != self.goal:
            actions = self._available_actions(state)
            qs = [self.Q[state, a] for a in actions]
            max_q = max(qs)
            best = [a for a, q in zip(actions, qs) if q == max_q]
            next_state = np.random.choice(best)

            path.append(next_state)

            if next_state in visited:
                break
            visited.add(next_state)

            state = next_state

            if len(path) > 100:
                break
```

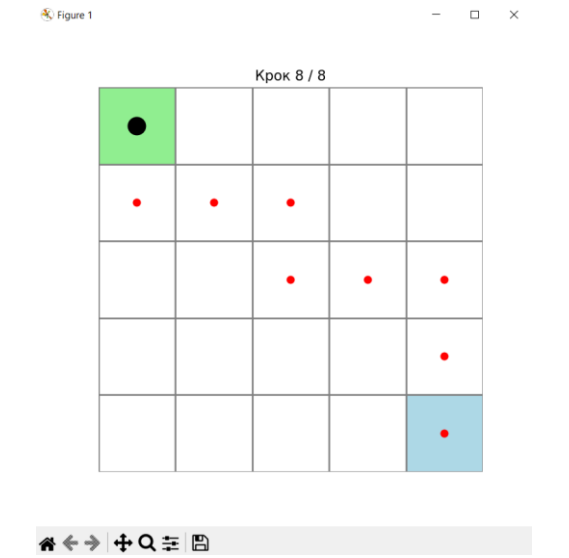
```
    return path

def visualize_path(self, path):
    fig, ax = plt.subplots(figsize=(6, 6))

    for idx in range(len(path)):
        ax.clear()

        for i in range(self.dim + 1):
            ax.plot([-0.5, self.dim - 0.5], [i - 0.5, i - 0.5], color='gray')
            ax.plot([i - 0.5, i - 0.5], [-0.5, self.dim - 0.5], color='gray')

        ax.set_xlim(-0.5, self.dim - 0.5)
        ax.set_ylim(self.dim - 0.5, -0.5)
        ax.set_aspect('equal')
        ax.axis('off')
        gi, gj = self._state2coord(self.goal)
        ax.add_patch(plt.Rectangle((gj - 0.5, gi - 0.5), 1, 1, color='lightgreen'))

        si, sj = self._state2coord(self.start)
        ax.add_patch(plt.Rectangle((sj - 0.5, si - 0.5), 1, 1, color='lightblue'))

        for s in path[:idx + 1]:
            r, c = self._state2coord(s)
            ax.plot(c, r, 'o', color='red')

        ci, cj = self._state2coord(path[idx])
        ax.plot(cj, ci, 'o', color='black', markersize=15)

        ax.set_title(f"Крок {idx} / {len(path) - 1}")

        plt.pause(0.4)
```

```
    plt.ioff()
    plt.show()


agent = QLearningAgent()
agent.train()
optimal_path = agent.get_optimal_path()
agent.visualize_path(optimal_path)
```

Результат тестування

Під час вибору маршруту інтелектуальний агент використовує елемент випадковості, що додає різноманітності процесу, проте загалом для досягнення цілі потрібно не більше 8 кроків.





Висновок

У цій роботі було реалізовано алгоритм Q-навчання для навігації агента у двовимірній сітці розміром 5x5. Метою агента було навчитися знаходити оптимальний шлях від початкової клітинки до цільової, використовуючи винагороди за правильні дії та покарання за неправильні. В ході тренування агент поступово покращував свою стратегію завдяки оновленню матриці Q, що відображає очікувану цінність переходів між станами.

Завдяки використанню балансу між дослідженням (exploration) і використанням набутих знань (exploitation), агент зміг ефективно знаходити найкоротший шлях до мети. Візуалізація процесу навчання та руху агента допомогла наочно продемонструвати, як агент приймає рішення на кожному кроці.

Отже, реалізований підхід показав ефективність у задачі навчання послідовності дій у просторовому середовищі з обмеженою кількістю станів. Подальше вдосконалення може включати масштабування до більших сіток та ускладнених сценаріїв з різноманітними нагородами.