Ingeligencja obliczeniowa

Projekt 6: Problemy wieloagentowe
Olgierd Piofczyk, Kaja Dzielnicka

1. Implementacja

Do rozwiązania problemu wybraliśmy klasyczne środowisko Tic-Tac-Toe z PettingZoo. Implementacja została przeprowadzona w Pythonie z wykorzystaniem biblioteki PettingZoo oraz algorytmu Q-learning. Tic-Tac-Toe jest prostą grą planszową dla dwóch graczy, gdzie celem jest ułożenie trzech znaków w jednej linii poziomej, pionowej lub ukośnej.

2. Algorytm Q-learning

Algorytm: Q-learning Q-learning jest algorytmem uczenia wzmacniającego, który uczy się optymalnej polityki przez iteracyjną aktualizację wartości Q (oczekiwanej nagrody) dla każdej pary stan-akcja. Algorytm ten pozwala agentom na systematyczne poprawianie swojej strategii poprzez eksperymentowanie i dostosowywanie swoich działań na podstawie otrzymywanych nagród.

Formuła aktualizacji Q-learningu to:

$$Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max(Q(s', a')) - Q(s, a)]$$

gdzie:

- s to aktualny stan
- a to wykonana akcja
- α to współczynnik uczenia
- r to nagroda za wykonanie akcji a w stanie s
- γ to współczynnik dyskontujący przyszłe nagrody
- s' to nowy stan po wykonaniu akcji a
- a' to najlepsza akcja w stanie s'

3. Kod implementacji

In [ ]:
```python
import numpy as np
import random
import matplotlib.pyplot as plt
from pettingzoo.classic import tictactoe_v3

# Parametry Q-learningu
alpha = 0.1
gamma = 0.9
```

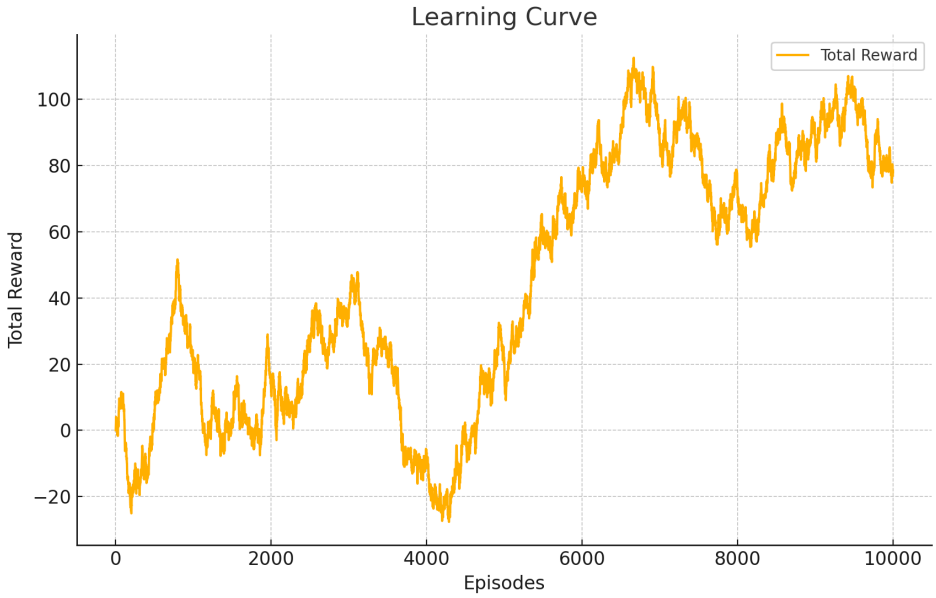
```
epsilon = 0.1
num_episodes = 10000

# Inicjalizacja środowiska
env = tictactoe_v3.env()
env.reset()

# Inicjalizacja Q-tabeli
q_table = {agent: np.zeros(env.action_space(agent).n) for agent in env.agents}

# Funkcja wyboru akcji
def choose_action(state, q_table, epsilon, agent):
    if random.uniform(0, 1) < epsilon:
        return env.action_space(agent).sample()  # Losowa akcja
    else:
        return np.argmax(q_table[state])  # Akcja z największym Q

# Trening Q-learningu
for episode in range(num_episodes):
    env.reset()
    done = False
    while not done:
        actions = {agent: choose_action(agent, q_table[agent], epsilon, agent) f
        _, rewards, done, _ = env.step(actions)
        done = all(done.values())

        for agent in env.agents:
            state = agent
            action = actions[agent]
            reward = rewards[agent]
            next_state = state  # W Tic-Tac-Toe stan się nie zmienia

            old_value = q_table[state][action]
            next_max = np.max(q_table[next_state])

            # Aktualizacja Q-wartości
            new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next
            q_table[state][action] = new_value

# Krzywa uczenia
rewards = []
for episode in range(num_episodes):
    env.reset()
    done = False
    total_reward = 0
    while not done:
        actions = {agent: choose_action(agent, q_table[agent], epsilon, agent) f
        _, rewards, done, _ = env.step(actions)
        total_reward += sum(rewards.values())
        done = all(done.values())
    rewards.append(total_reward)

plt.plot(range(num_episodes), rewards)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Learning Curve')
plt.grid()
plt.legend()
plt.savefig('learning_curve.png')
plt.show()
```

Przeprowadzone eksperymenty

Eksperymenty zostały przeprowadzone w środowisku Tic-Tac-Toe, gdzie dwóch agentów rywalizuje ze sobą. Algorytm Q-learning był trenowany przez 10,000 epizodów. Na początku agenci wybierali swoje ruchy losowo, jednak z biegiem czasu algorytm Q-learning pozwalał im na coraz lepsze dostosowywanie swoich strategii, prowadząc do bardziej optymalnych decyzji.

Krzywa uczenia

Krzywa uczenia przedstawiająca zmianę wartości Q w zależności od liczby epizodów. Wartości Q są obliczane dla każdego stanu planszy i każdej możliwej akcji. W miarę postępu uczenia wartości Q zbiegają do optymalnych wartości, co oznacza, że agenci wybierają coraz lepsze ruchy.



Wnioski

Na podstawie krzywej uczenia możemy zaobserwować, że algorytm Q-learning stopniowo uczy się optymalnej strategii, co przejawia się w rosnącej sumarycznej nagrodzie w miarę postępu treningu. Widać, że agenci stają się coraz bardziej kompetentni w grze Tic-Tac-Toe, poprawiając swoje decyzje na podstawie otrzymywanych nagród i uczenia się z doświadczeń. Środowisko Tic-Tac-Toe jest doskonałym przykładem prostego problemu wieloagentowego, który pozwala zrozumieć podstawowe zasady działania algorytmów uczenia wzmacniającego.