Acest joc reprezinta un agent care invata sa joace jocul Frozen Lake folosind algoritmul Q-Learning.

"FrozenLake" este un mediu clasic de învățare prin întărire oferit de biblioteca Gym. Este un joc simplu de tip grid-world conceput pentru a preda și testa algoritmii de învățare prin întărire.

1. Grid World: FrozenLake este reprezentat ca o lume grilă, care este o grilă sau o hartă 2D. Fiecare celulă din grilă reprezintă o stare, iar agentul se poate deplasa de la o stare la alta prin efectuarea de acțiuni.

2. Obiectiv: Obiectivul agentului este de a naviga prin lacul înghețat de la poziția de pornire până la poziția de obiectiv, evitând în același timp găurile din gheață. Obiectivul este situat, de obicei, la unul dintre colțurile rețelei.

3. Stări: Grila este formată din stări discrete. În cazul lacului înghețat 4x4 standard, există 16 stări (4 rânduri și 4 coloane). Cu toate acestea, există variante de FrozenLake cu grile mai mari, cum ar fi 8x8.

4. Acțiuni: Agentul poate întreprinde patru acțiuni posibile în fiecare stare:

- Mutarea în sus
- Mutarea în jos
- Deplasare spre stânga
- Deplasare la dreapta

5. Recompense: Agentul primește recompense în funcție de acțiunile sale și de tranzițiile de stare rezultate. Recompensele sunt definite de obicei după cum urmează:

- Recompensă pozitivă (de exemplu, +1) atunci când atinge starea de obiectiv.
- Recompensă negativă (de exemplu, -1) atunci când se cade într-o gaură (o gaură este reprezentată ca un obstacol).

O recompensă negativă mică (de exemplu, -0,01) pentru fiecare pas de timp pentru a încuraja agentul să ajungă rapid la obiectiv.

6. Întâmplare: În FrozenLake standard, există un element de aleatorism. Atunci când un agent întreprinde o acțiune, există o probabilitate ca acesta să alunece și să se deplaseze într-o direcție diferită de cea intenționată. Acest lucru adaugă o provocare suplimentară mediului.

7. Sfârșitul episodului: Fiecare episod se termină atunci când agentul fie ajunge la obiectiv, fie cade într-o gaură. Agentul este apoi resetat la poziția de pornire pentru următorul episod.

8. Explorare vs. Exploatare: Pentru a rezolva mediul FrozenLake, agentul trebuie să echilibreze explorarea și exploatarea. Acesta poate fie să ia acțiunea care crede că va duce la cea mai mare recompensă cumulativă (exploatare), fie să ia o acțiune aleatorie pentru a explora și a descoperi noi strategii (explorare).

Algoritmul Q-learning este folosit de agentul din jocul Frozen Lake pentru a învăța și a lua decizii. Principalele idei includ:

Inițializarea tabelului Q cu valori mici pentru toate combinațiile (stare, acțiune).

Agentul alege între explorare (alegere aleatoare a acțiunilor) și exploatare (alegere a celei mai bune acțiuni pe baza valorilor Q).

Tabelul Q este actualizat folosind regula Q-learning, care se bazează pe recompensa obținută și valorile Q curente.

Epsilon se reduce gradual pentru a reduce explorarea și a permite exploatarea cunoștințelor.

Procesul se repetă pentru mai multe episoade, permițând agentului să învețe o strategie optimă.

Algoritmul Q-learning converge către o strategie optimă atunci când este antrenat suficient de mult și cu parametri adecvați.

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

s reprezintă starea curentă în care se află agentul.

a reprezintă acțiunea pe care agentul a ales-o în starea s.

α este rata de învățare (learning rate) care controlează cât de mult sunt actualizate valorile Q la fiecare pas.

r este recompensa obținută pentru acțiunea a din starea s.

Câteva funcții ale aplicației:
1.

```
def choose_action(self, state):
    if self.rng.random() < self.epsilon:
        return self.env.action_space.sample()
    return np.argmax(self.q_table[state, :])
```

Aceasta este metoda de alegere a acțiunilor pe baza stării curente. În cazul în care un număr aleatoriu este mai mic decât epsilon, se efectuează o acțiune aleatorie; în caz contrar, se alege cea mai bună acțiune pe baza tabelului Q.

2.

```
def update_q_table(self, state, action, reward, next_state):
    self.q_table[state, action] += self.learning_rate * (
            reward + self.discount_factor * np.max(self.q_table[next_state, :]) -
self.q_table[state, action]
    )
```

Această funcție face update la q-table după fiecare acțiune

3.

```
def train_agent(episodes, render=False):
    env = gym.make('FrozenLake-v1', map_name="8x8", is_slippery=True, render_mode='human'
if render else None)
    agent = FrozenLakeAgent(env)

    rewards_per_episode = np.zeros(episodes)

    for episode in range(episodes):
        state = env.reset()[0]
        terminated, truncated = False, False

        while not terminated and not truncated:
            action = agent.choose_action(state)
            next_state, reward, terminated, truncated, _ = env.step(action)
            agent.update_q_table(state, action, reward, next_state)
            state = next_state
            rewards_per_episode[episode] = reward

        agent.update_epsilon()

    env.close()
    save_q_table(agent.q_table, "frozen_lake8x8.pkl")

    return rewards_per_episode
```

Aceasta este responsabilă pentru antrenarea agentului. Mai întâi se creează mediul jocului, o grilă de 8x8 și condiția că aceasta este alunecoasă. Apoi inițializăm o instanță a agentului cu mediul creat. Inițializăm o matrice pentru a stoca recompensa primită în fiecare episod. Acum iterăm prin fiecare episod, resetăm mediul pentru fiecare episod și luăm starea inițială, inițializăm flags pentru a verifica dacă episodul s-a terminat. Dacă episodul nu s-a terminat continua pana se termina.

4.

```
env.close()
save_q_table(agent.q_table, filename: "frozen_lake8x8.pkl")
```

Inchidem mediul creat.

Salveaza Q-tabel-ul intr-un fisier.

5.

```
    return rewards_per_episode


1 usage
def plot_training_metrics(rewards_per_episode):
```

Returneaza vectorul de recompense pentru fiecare episod.

6.

```
def plot_training_metrics(rewards_per_episode):
    moving_avg_rewards = np.convolve(rewards_per_episode, np.ones(100) / 100, mode='valid')
```

-este o functie care afiseaza grafice pentru recompensele obtinute in timpul antrenamentului. Aceasta calculeaza, de asemenea, media mobila a recompenselor pentru a vizualiza progresul antrenamentului.

7.

```
if __name__ == '__main__':
    rewards = train_agent( episodes: 1000, render=True)
    plot_training_metrics(rewards)
```

-este punctul de intrare al programului. Aici, se apeleaza functia train_agent pentru a incepe antrenamentul cu un numar specificat de episoade (in acest caz, 1000) si se afiseaza apoi graficele de antrenament cu functia plot_training_metrics.