Sprawozdanie ćwiczenia nr 6 – Łukasz Szydlik

Cel i opis eksperymentów

Celem ćwiczenia jest zaimplementowanie algorytmu Q-learning, a następnie użycie go do wytrenowania agenta rozwiązującego problem Cliff Walking oraz stworzyć wizualizację wyuczonej polityki.

Treść zadania pod adresem:

https://gymnasium.farama.org/environments/toy_text/cliff_walking/

Instrukcja programu

Instalacja

1. Należy pobrać repozytorium

2. Wejść w folder lab6

3. Upewnić się, że posiadamy wymagane biblioteki:

‘pip install -r requirements.txt’

4. Zainstalować rozszerzenie Jupyter (VS Code)

Uruchomienie programu

W celu uruchomienia QLearning.ipynb wymagane jest korzystanie z notatnika Jupyter

W Jupyter ustawiamy kernel na nasze środowisko python. Następnie po naciśnięciu ”Run All” utworzą się wykresy, które zostaną wyświetlone w notatniku oraz zapisane do folderu plots.

Graph1: Łączna liczba występowania danego stanu oraz akcji w ”n_runs” treningach.

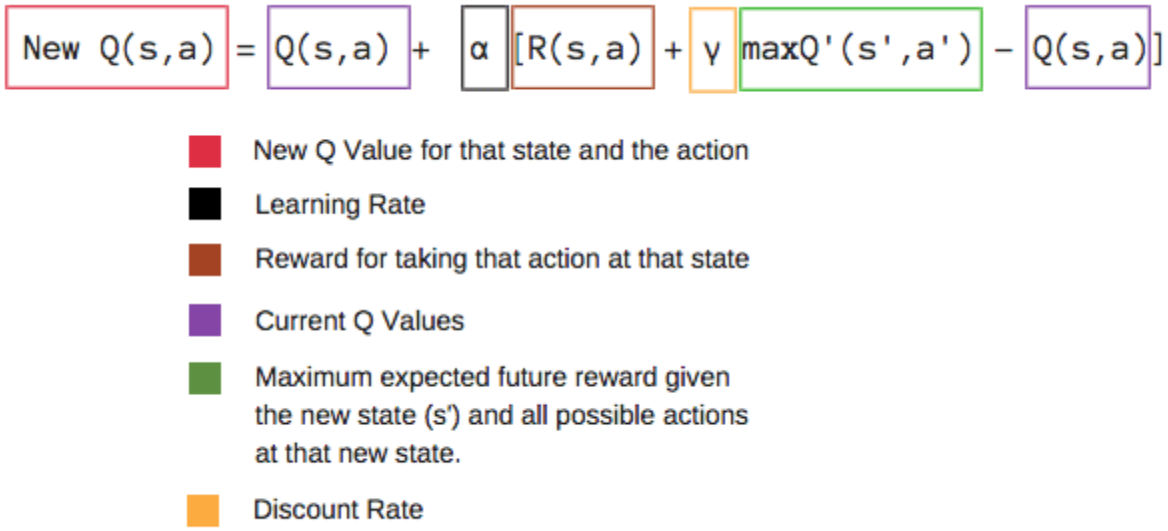
Graph2: Ostatnia pozycja agenta oraz wyuczona Q-tabela prezentująca najlepsze akcje.

Graph_n0: Średnia nagroda oraz średnia ilość kroków w danym epizodzie po ”n_runs” treningach.

Graph_nX: Średnia nagroda oraz średnia ilość kroków w danym epizodzie po ”n_runs” treningach, gdzie X to ”start_episode”, czyli epizod od którego chcemy obserwować wykres.

W pliku config.json znajudują się parametry, które możemy dostosować.

Wstęp



Podstawowymi parametry algorytmu Q-learning są:

Liczba epizodów, czyli ile razy agent podejmie próbę rozwiązania zadania od początku.

Za mała wartość: Agent może nie mieć wystarczająco dużo czasu, aby nauczyć się skutecznej polityki. Proces uczenia zostanie przerwany zbyt wcześnie.

Za duża wartość: Czas treningu wydłuża się. W praktyce, po osiągnięciu pewnego poziomu, dodatkowe epizody mogą nie przynosić korzyści.

Współczynnik uczenia (alfa, α), który określa, w jakim stopniu nowe informacje zastępują stare wartości Q.

Duża wartość (np. 0.8):

Szybsze dostosowanie się do nowych informacji.

Może prowadzić do niestabilności, gdy wartości Q zmieniają się zbyt szybko.

Mała wartość (np. 0.1):

Stabilniejsze uczenie się, ale wolniejsze.

Agent potrzebuje więcej epizodów, aby zbliżyć się do optymalnej polityki.

Współczynnik dyskontowania (gamma, γ), który określa, jak bardzo agent dba o przyszłe nagrody.

Duża wartość (np. 0.95–1.0):

Agent bardziej dba o długoterminowe nagrody.

W skrajnych przypadkach może ignorować natychmiastowe nagrody.

Mała wartość (np. 0.1–0.5):

Agent koncentruje się na nagrodach krótkoterminowych.

Może prowadzić do suboptymalnych strategii w środowiskach wymagających dalekowzroczności.

Epsilon - parametr kontrolujący eksplorację w strategii ϵ-greedy.

W ϵ-greedy agent wybiera:

Z prawdopodobieństwem ϵ: losową akcję (eksploracja).

Z prawdopodobieństwem 1−ϵ: najlepszą znaną akcję (eksploatacja).

Wpływ:

Duża wartość (np. 0.5):

Więcej eksploracji. Agent odkrywa nowe stany i akcje.

Może prowadzić do dłuższego czasu uczenia się, ponieważ agent rzadziej wykorzystuje aktualną wiedzę.

Mała wartość (np. 0.01):

Większa eksploatacja znanych wartości Q.

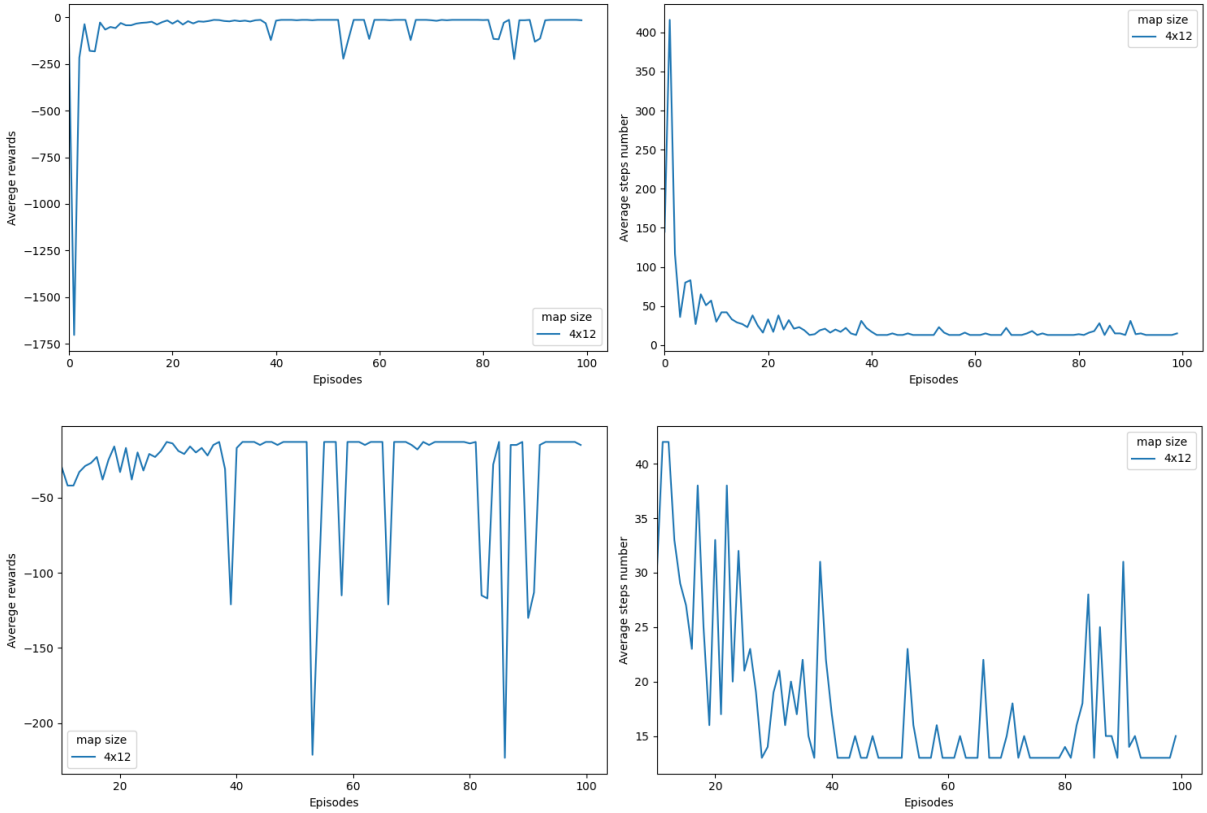
Może prowadzić do utknięcia w lokalnym minimum, jeśli agent nie eksploruje wystarczająco środowiska.

Wyniki ćwiczenia

Na następnych stronach przedstawiono eksperymenty. Wykorzystano tylko jedną próbę, ze względu na to, że przy wielu próbach otrzymamy średnią nagród i kroków i niestabilność będzie trudna do wykrycia.

Eksperyment 1: parametry wybrane wedle uznania

```
"total_episodes": 100,  "learning_rate": 0.8,
"gamma": 0.95,          "epsilon": 0.05,
```

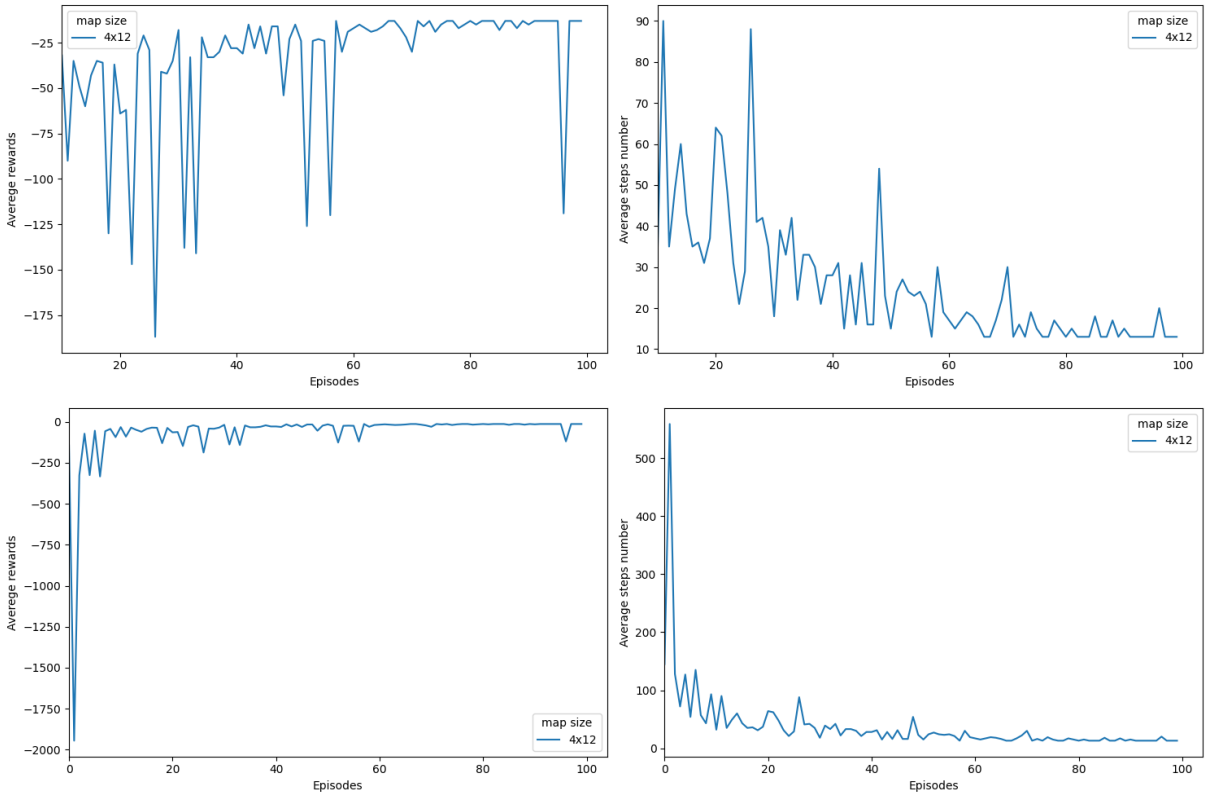


Obserwacje:

Algorytm dość szybko nauczył się dobrej strategii, już po 10 epizodach wartość kosztu wyniosła poniżej 50. Jednak był bardzo niestabilny.

Eksperyment 2: zmniejszony współczynnik uczenia

```
"total_episodes": 100,  "learning_rate": 0.4,
"gamma": 0.95,          "epsilon": 0.05,
```

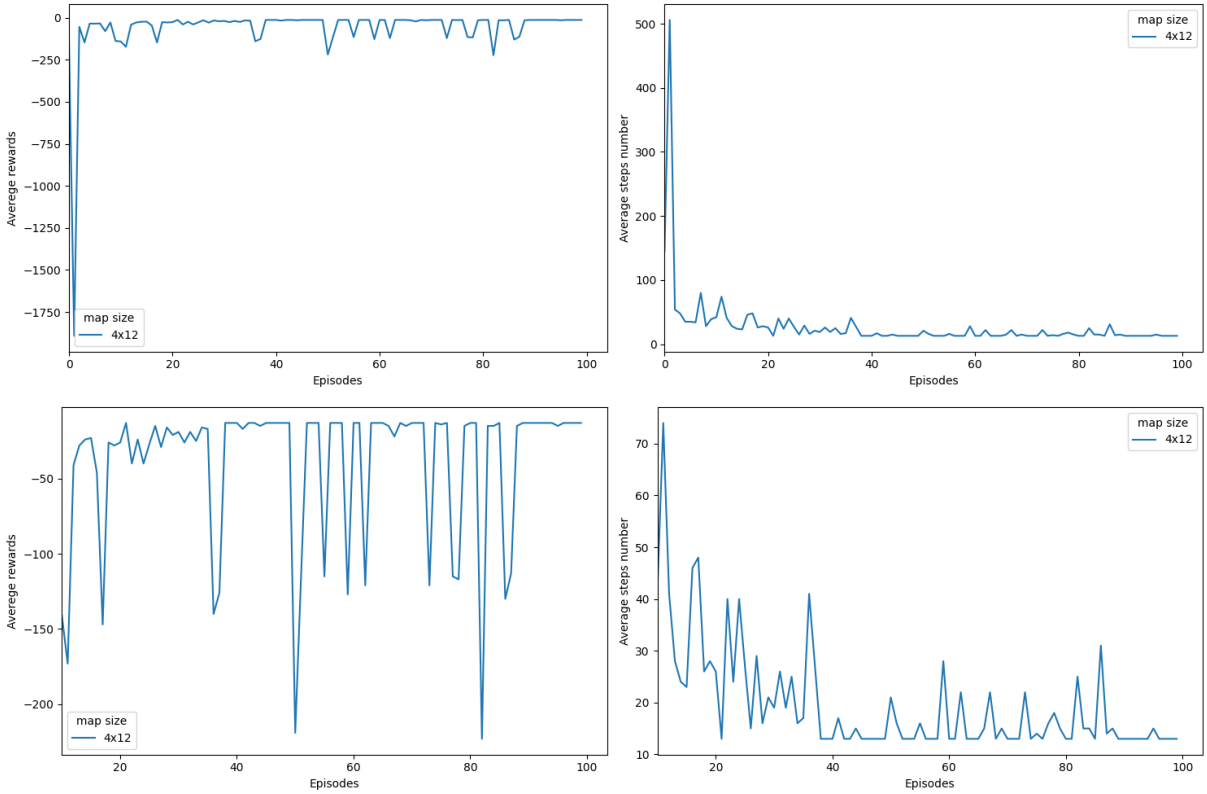


Obserwacje:

Tak samo niestabilny. Późno osiąga optymalną wartość (po około 70 epizodach)

Eksperyment 3: zmniejszono współczynnik gamma (nagrody krótkoterminowe)

```
"total_episodes": 100,  "learning_rate": 0.8,
"gamma": 0.7,           "epsilon": 0.05,
```

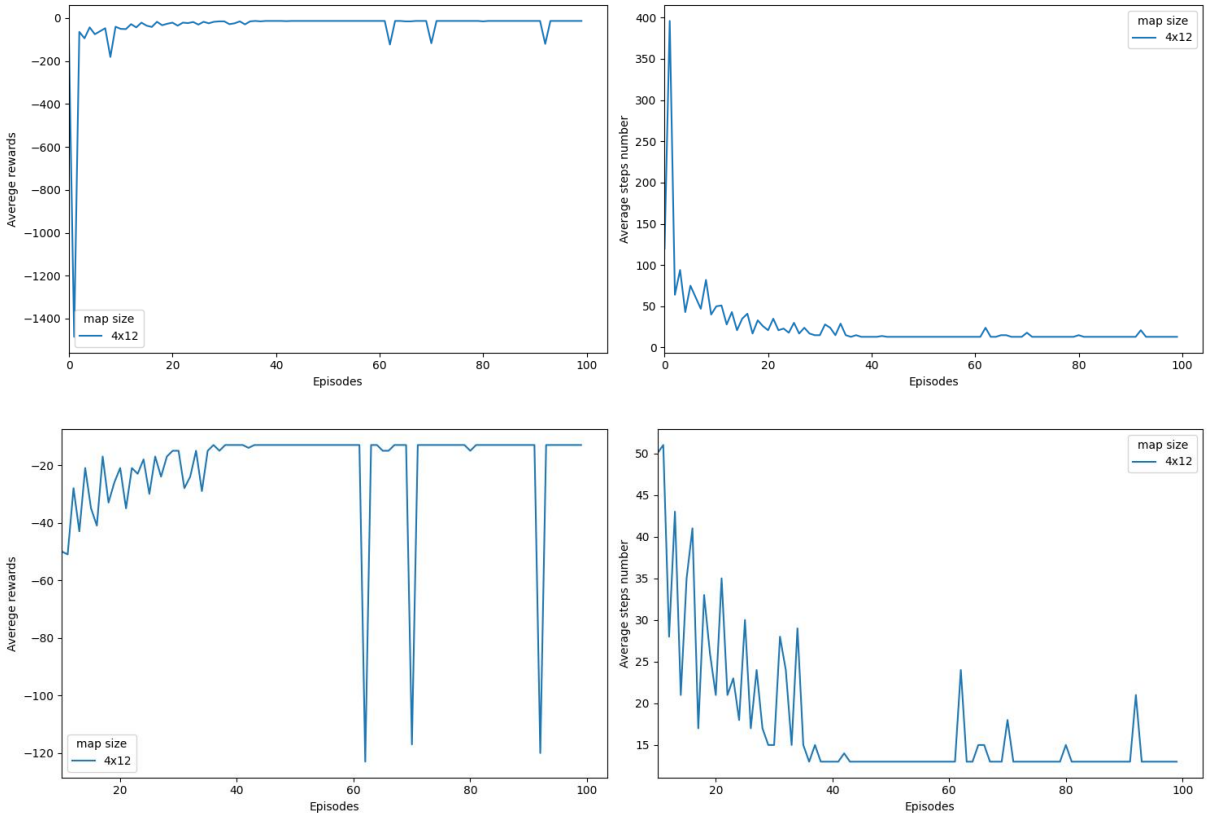


Obserwacje:

Również niestabilny, ale szybciej osiągnął najlepszy rezultat (przed 40 epizodem).

Eksperyment 4: próba, zmniejszono epsilon (zmniejszona eksploracja)

```
"total_episodes": 100,  "learning_rate": 0.8,
"gamma": 0.7,           "epsilon": 0.01,
```

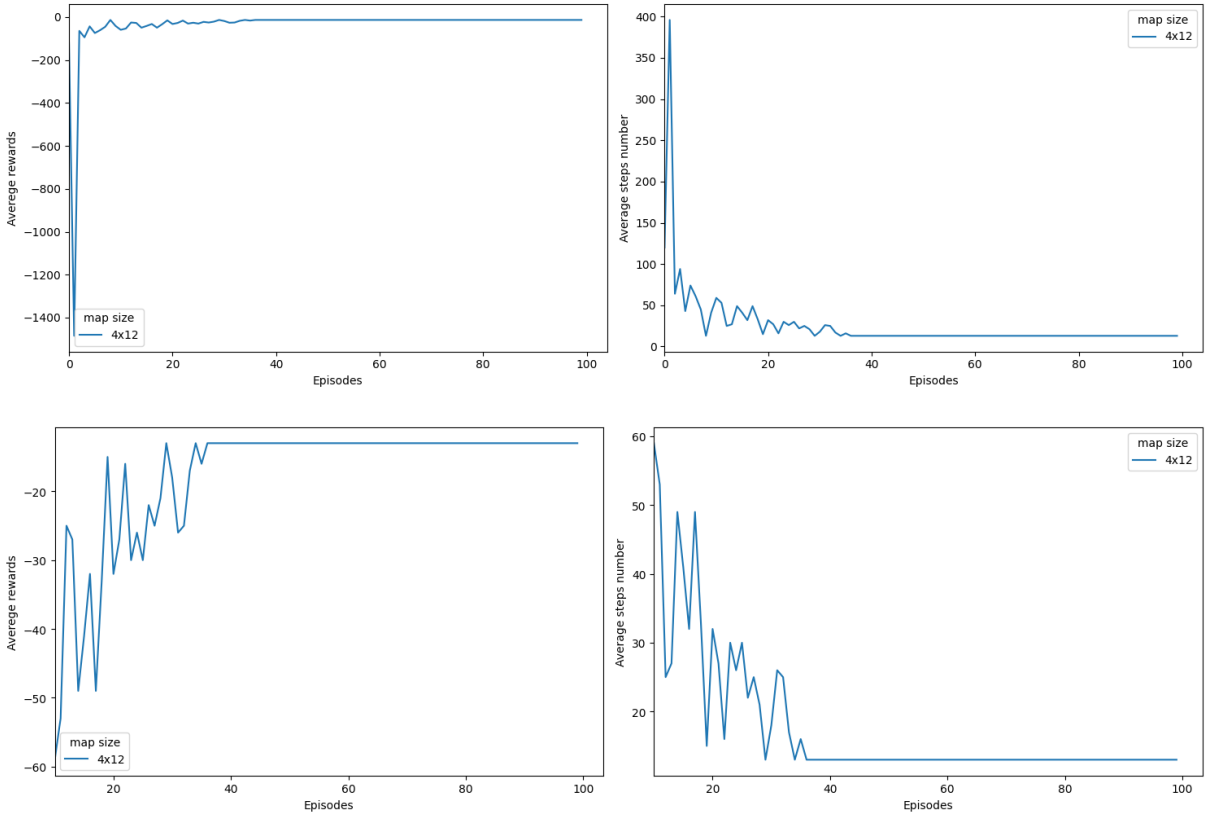


Obserwacje:

Stabilność algorytmu się polepszyła. Optymalne wyniki są bardzo szybko osiągane (już przed 40 epizodem)

Eksperyment 5: zerowy epsilon (maksymalna eksploatacja)

```
"total_episodes": 100,  "learning_rate": 0.8,
"gamma": 0.7,           "epsilon": 0,
```

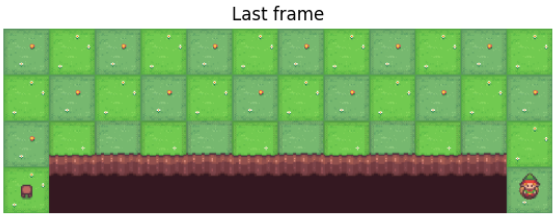


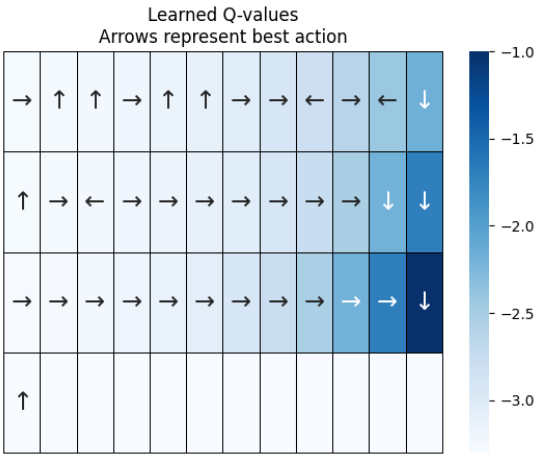
Obserwacje:

Stabilny. Równie szybko dąży do dobrego wyniku.

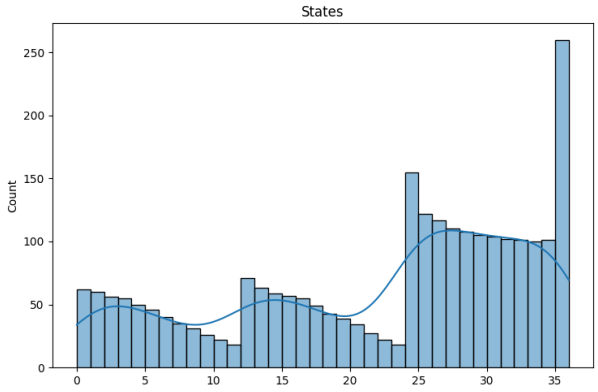
Dla parametrów z eksperymentu 5

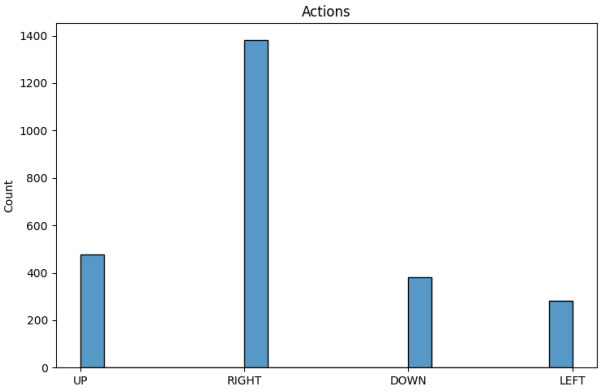
Najlepsza polityka





Ilość odwiedzin poszczególnych stanów oraz akcji





Wnioski

1) Eksploatacja (niski epsilon, bliski zeru) ma kluczową rolę w przypadku stabilności algorytmu, szczególnie przy łatwych zadaniach. Jednakże w przypadku trudniejszych zadań może on spowodować utknięcie w ekstremum lokalnym. Dobrym rozwiązaniem jest zmniejszanie epsilon z każdym epizodem. Początkowo skupi się on na eksploracji a w późniejszych etapach na eksploatacji.
2) Zmniejszenie gamma (większy wpływ nagrody krótkoterminowej) w prostym zadaniu może spowodować szybsze dotarcie do najlepszego rezultatu.
3) W prostym zadaniu warto stosować duży współczynnik uczenia.
4) Dobranie odpowiedniej liczby epizodów jest ważne. Gdy nadamy zbyt dużą wartość może ona nie przynieść korzyści, a wydłuży czas potrzebny do treningu agenta. W naszym zadaniu najlepszą wartość jesteśmy w stanie uzyskać już przed 40 epizodem.
5) Ze względu na niską eksplorację (epsilon) polityka nie jest najlepsza, jeśli rozważalibyśmy start z dowolnego miejsca.