Grupa 2 Środa 17:05	Algorytmy sztucznej inteligencji w przemyśle 4.0
Dawid Jordan 255330 Natalia Stępień 254978	Temat: Uczenie ze wzmocnieniem.

Sprawozdanie

1. Krótki wstęp teoretyczny.

Uczenie ze wzmocnieniem (ang. *reinforcement learning*) to jedna z technik uczenia maszynowego, polegająca na szkoleniu agenta poprzez interakcje z otoczeniem i uzyskiwane sygnały zwrotne w postaci nagrody lub kary za wykonane działania. Docelowo agent ma dążyć do maksymalizacji sumy nagród. W tym przypadku zamiast zestawu danych uczących przygotuje się środowisko, z którego model zbiera dane automatycznie. W większości algorytmów zasada działania sprowadza się do utworzenia **polityki** określającej strategię podejmowania przez agenta decyzji, opierając się o stan bieżący. Jest to funkcja przyjmująca na wejściu obserwację i zwracająca akcję, natomiast **środowisko** w RL definiuje się jako zadanie bądź symulację, z którym agent wchodzi w interakcję (np. gry komputerowe). Każde środowisko posiada określone cechy:

- Zmienną stanu środowiska (ang. *state*)
- Krok (ang. *step*) – funkcję aktualizującą stan środowiska na podstawie podanej akcji i zwracająca nagrodę i obserwację
- Epizod (ang. *episode*) – zbiór kroków, po których wykonaniu resetuje się stan środowiska
- Nagrodę (ang. *reward*)– zmienną określającą korzystność wykonanego kroku
- Obserwację (ang. *observation*)– skalar, wektor albo macierz, którą zwraca krok i która opisuje aktualny stan środowiska

Q-learning jest rodzajem algorytmu uczenia ze wzmocnieniem, który nie wymaga modelu środowiska i którego zadaniem jest nauczenie agenta przypisywania wartości do każdej możliwej do podjęcia akcji z zachowaniem warunku znajdowania się agenta w określonym stanie. Za każdym skończeniem procesu decyzyjnego Markova znajduje optymalną politykę maksymalizującą oczekiwaną wartość całkowitej nagrody zebranej w wszystkich kolejnych krokach. Q w nazwie algorytmu odnosi się do jakości (ang. *quality*), a więc oczekiwanej nagrody

za działanie podjęte w danym stanie. Jest to również nazwa funkcji obliczającej jakość kombinacji stan-akcja (Q(s,a)) opisanej wzorem Bellmana:

$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma a' \max Q(s',a') - Q(s,a))$, gdzie:

- Q(s,a) - bieżąca wartość Q podjęcia akcji (a) w danym stanie (s).
- α - szybkość uczenia się, określająca stopień zastępowania starych informacji nowymi.
- r - nagroda otrzymana po wykonaniu akcji (a) w danym stanie (s).
- γ - współczynnik dyskonta, który reprezentuje znaczenie przyszłych nagród.
- s' - kolejny stan po podjęciu akcji (a) w stanie (s).
- maxQ(s', a') - maksymalna wartość Q dla następnego stanu (s') we wszystkich możliwych akcjach (a').

Algorytm Q-learning pośrednio uczy się też polityki, zazwyczaj zachłannej, gdzie agent wybiera akcję, która ma przynieść najwyższą skumulowaną nagrodę, w oparciu o poznane wartości Q (przechowywane w tablicy Q-table). Ważna jest tu równowaga pomiędzy eksploracją a eksploatacją. W trakcie trwania procesu uczenia się agent musi eksplorować otoczenie, żeby zdecydować, które działania przynoszą najwyższe nagrody przy jednoczesnym wykorzystaniu zdobytej już wiedzy w celu maksymalizacji nagrody. Jedną ze strategii jest polityka zachłanności epsilon, gdzie agent wybiera losową akcję z prawdopodobieństwem (epsilon) i najbardziej znaną akcję z prawdopodobieństwem (1 – epsilon).

Algorytm DQN (ang. *Deep Q-learning*) to połączenie Q-learningu z głębokimi sieciami neuronowymi w celu przybliżenia wartości Q, co z kolei umożliwia algorytmowi obsługę wielowymiarowych przestrzeni stanów. Wzór Bellmana opisujący DQN:

$Q*(s,a) = E[r + \gamma a' \max Q*(s',a')]$, gdzie:

- s – stan
- a – akcja
- r – nagroda
- γ – współczynnik dyskontowy

Używa się dwóch sieci neuronowych: głównej i docelowej. Główna sieć neuronowa przyjmuje jako wejście reprezentację stanu (s), a na wyjściu zwraca wartość Q(s,a) dla każdej możliwej akcji (a), przy czym wagi sieci aktualizuje się w taki sposób, żeby minimalizować błąd między wartością oczekiwaną a docelową. Sieć docelowa rozwiązuje problem niestabilności procesu uczenia. Jest to kopia głównej sieci, przy czym parametry są aktualizowane co kilka kroków treningowych. Dostarcza ona wartości docelowe używane do uczenia sieci głównej. Z kolei problem korelacji danych rozwiązuje zastosowanie buforu powtórek, przechowującego dane (s,a,r,s',done) z poprzednich kroków. Następnie próbki są losowo wybierane z bufora podczas treningu. Wagi aktualizuje się na podstawie gradientu za pomocą któregoś z algorytmów optymalizacji, np. Adam, RMSProp.

Taxi-v3 to środowisko deterministyczne dostarczane przez bibliotekę Gymnasium, gdzie steruje się akcjami taksówki w przestrzeni będącej siatką o wymiarach 5x5, na której znajdują się 4 potencjalne lokalizacje pasażerów i miejsc ich dostarczenia oznaczone literami: R (Red), G (Green), Y (Yellow), i B (Blue). Problem polega na takim dobraniu trasy i podjętych akcji, aby taksówka była w stanie jak najszybciej odebrać pasażera i dowieźć go w miejsce docelowe. Stan środowiska zwracany jest jako (x, y, lokalizacja pasażera, lokalizacja celu), gdzie x, y to współrzędne położenia taksówki.

Możliwych do podjęcia jest 6 akcji:

- 0: Przesuń się na północ
- 1: Przesuń się na południe
- 2: Przesuń się na wschód
- 3: Przesuń się na zachód
- 4: Odbierz pasażera (pick up)
- 5: Odstaw pasażera (drop off)

Przy czym należy pamiętać o dwóch ograniczeniach, tj.:

- Brak możliwości przejścia taksówki przez ściany
- Za próbę zabrania lub odstawienia pasażera w złym miejscu przewidziana jest kara.

Dodatkowo za poszczególne akcje przyznaje się nagrody:

- +20: Za poprawne dostarczenie pasażera do celu.
- -1: Za każdy krok (motywacja do szybkiego ukończenia zadania).
- -10: Za próbę zabrania lub odstawienia pasażera w niewłaściwej lokalizacji.

Epizod kończy się w momencie dostarczenia pasażera do celu.

2. Cel badań

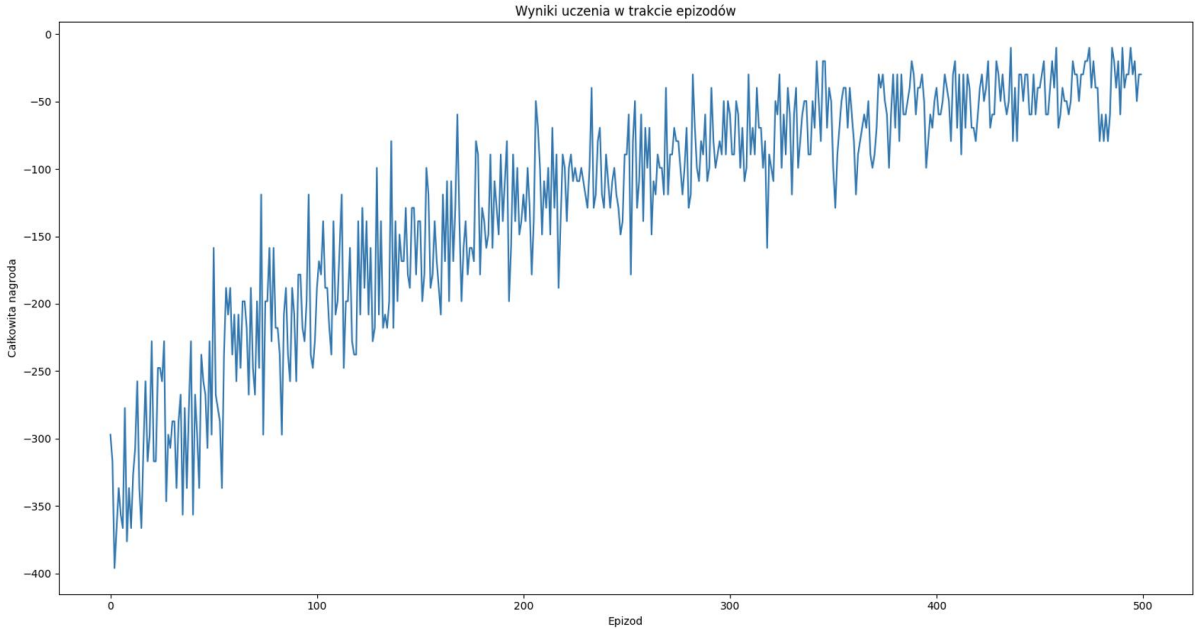
Cel badań dotyczył zaimplementowania i porównania działania dwóch różnych algorytmów w celu rozwiązania problemu Taxi-v3 z biblioteki Gymnasum, a także przeprowadzenia badań wpływu zmiany poszczególnych parametrów na jakość wyników.

Jako że problem Taxi-v3 jest problemem deterministycznym, do rozwiązania go użyto gotowego algorytmu DQN dostępnego w bibliotece Stable Baselines 3 kompatybilnego z biblioteką Gymnasium oraz algorytmu Q-learning napisanego samodzielnie. Na podstawie algorytmu DQN sprawdzona została poprawność implementacji środowiska Gymnasium, a także posłużył on później do porównania jakości drugiego z algorytmów. Z kolei algorytm Q-learning został zaimplementowany w celu przebadania wpływu zmiany wartości parametrów na jakość uzyskiwanych wyników i efektów uczenia.

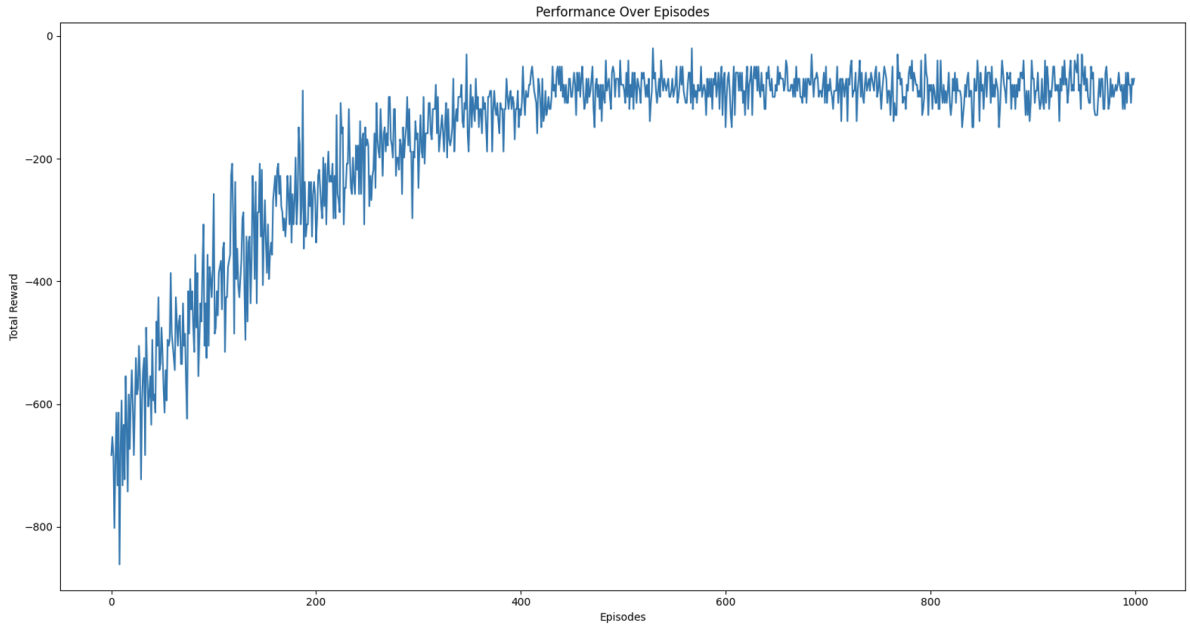
3. Wyniki badań

a. DQN

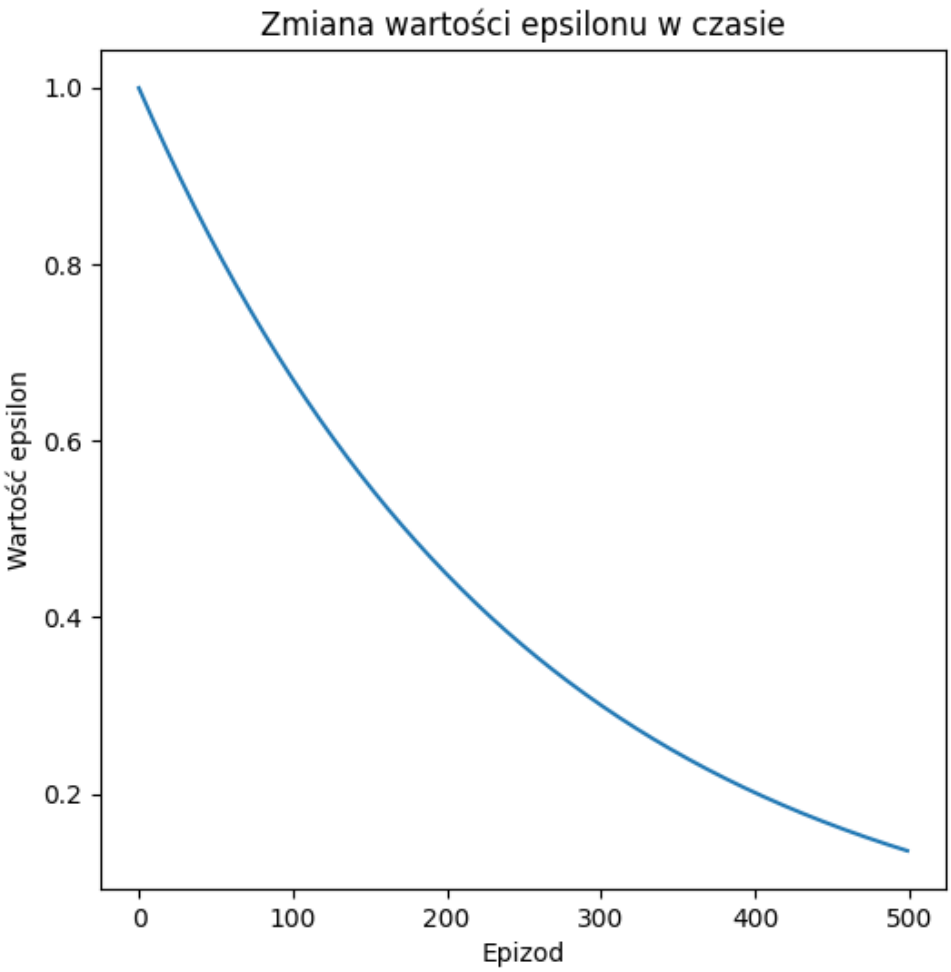
Poniżej przedstawiono wyniki uzyskane przy użyciu algorytmu DQN dla próbek 500 epizodów oraz 1000 epizodów. Wartość kluczowa Epsilon przyjęto jako 1 wraz z współczynnikiem spadku wynoszącym 0,996. Za wartość minimalną Epsilon przyjęto 0.1, aby zapewnić szansę przy końcowych epizodach treningu na eksplorację losowych akcji przez agenta. Średnia nagroda przez pierwsze 50 epizodów wyniosła -303.04 a przez ostatnie 50 epizodów -38.31. Odchylenie standardowe dla ostatnich 50 epizodów wyniosło 18.99.



Rys.1 Wartości nagród za dany epizod, sieć trenowana przez 500 epizodów.



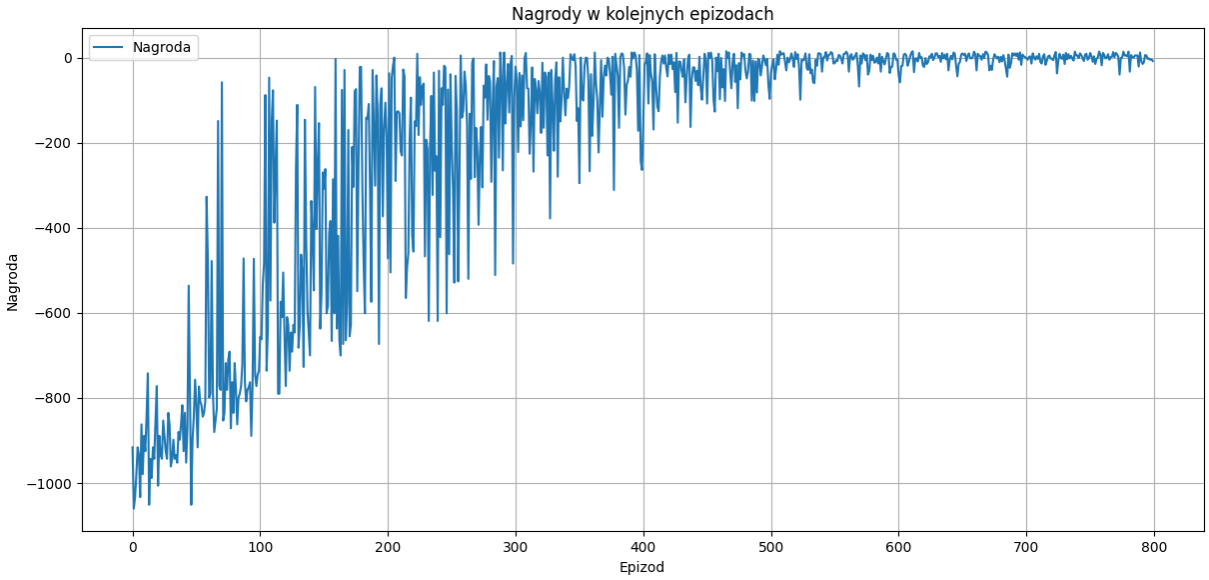
Rys. 2 Wartości nagród za dany epizod, sieć trenowana przez 1000 epizodów.



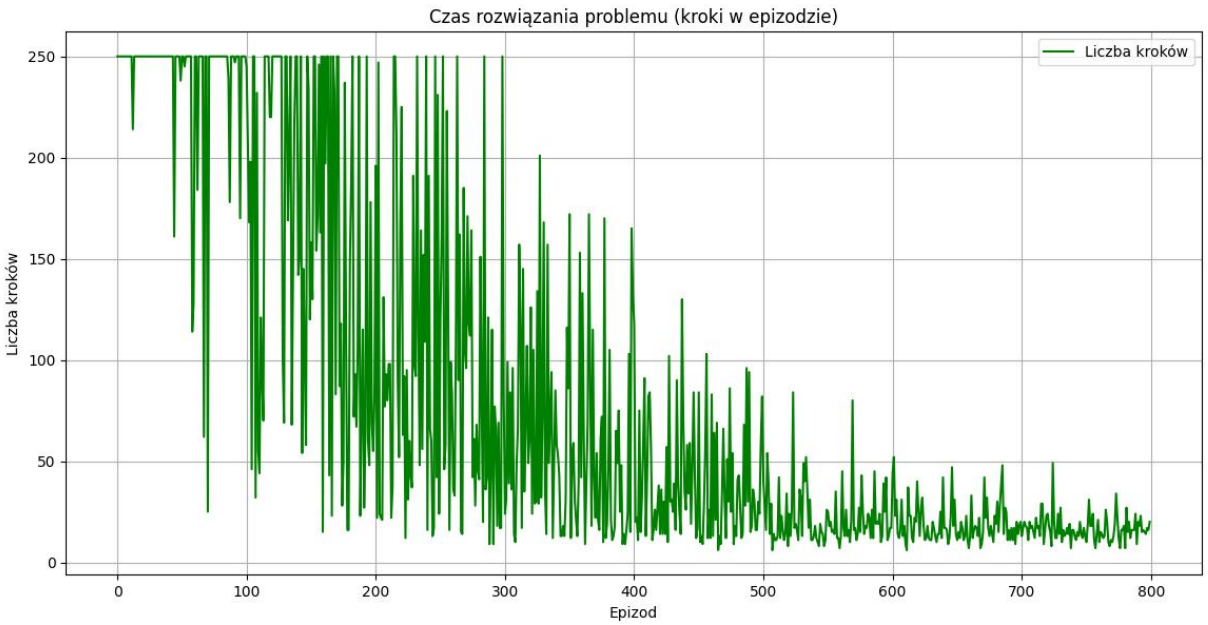
Rys. 3 Zmiana wartości Epsilon w trakcie uczenia sieci.

b. Q-learning

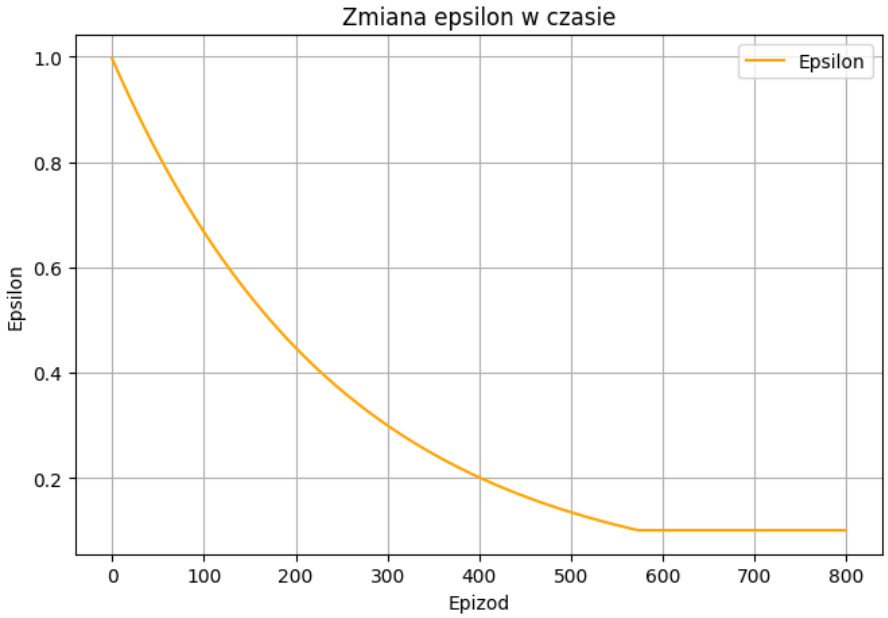
Drugim rozwiązaniem było podejście z wykorzystaniem algorytmu Q-learning. Wartość kluczowa Epsilon przyjęto tak jak w podpunkcie a, ilość epizodów przyjęto jako 800 a ilość dozwolonych kroków wynosiła 250. Średnia nagroda podczas pierwszych 100 epizodów wyniosła -750.27 a przez ostatnie 100 epizodów -1.25. Odchylenie standardowe dla ostatnich 50 epizodów wyniosło 0.54, nagroda uzyskana podczas testu wyniosła 5 w liczbie 16 kroków. Poniżej przedstawiono wykresy



Rys. 4 Wartość nagród w kolejnych epizodach działania algorytmu Q-learning.

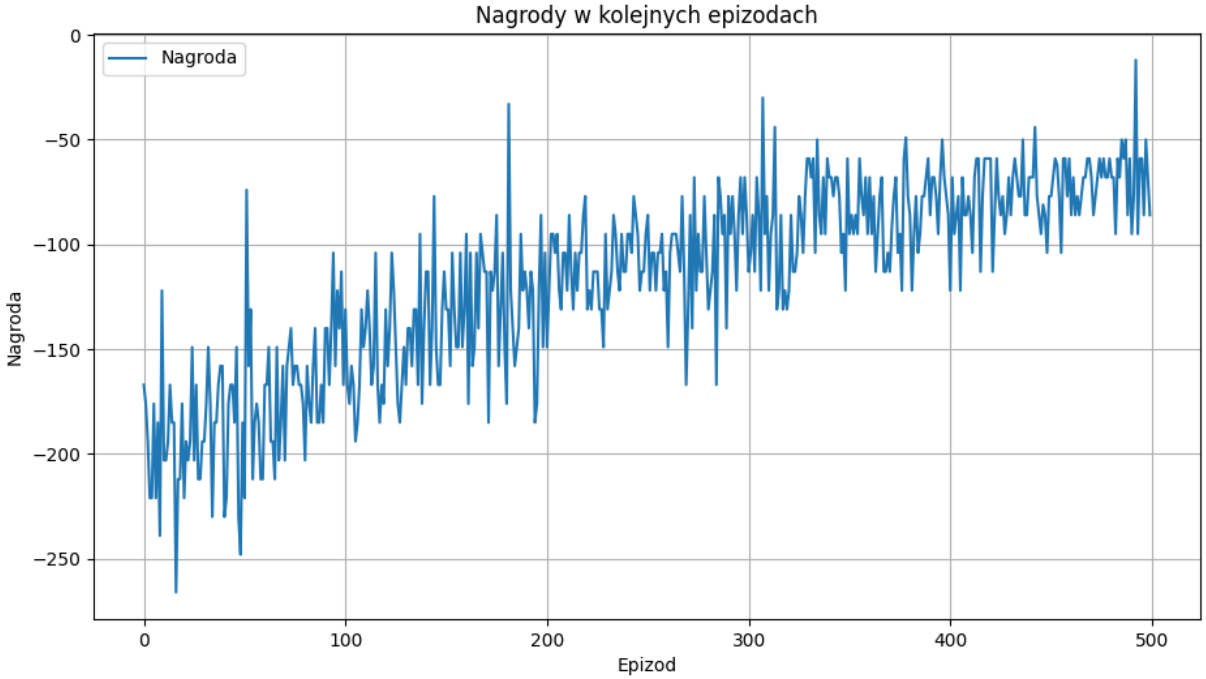


Rys. 5 Liczba kroków w kolejnych epizodach (Q-learning).

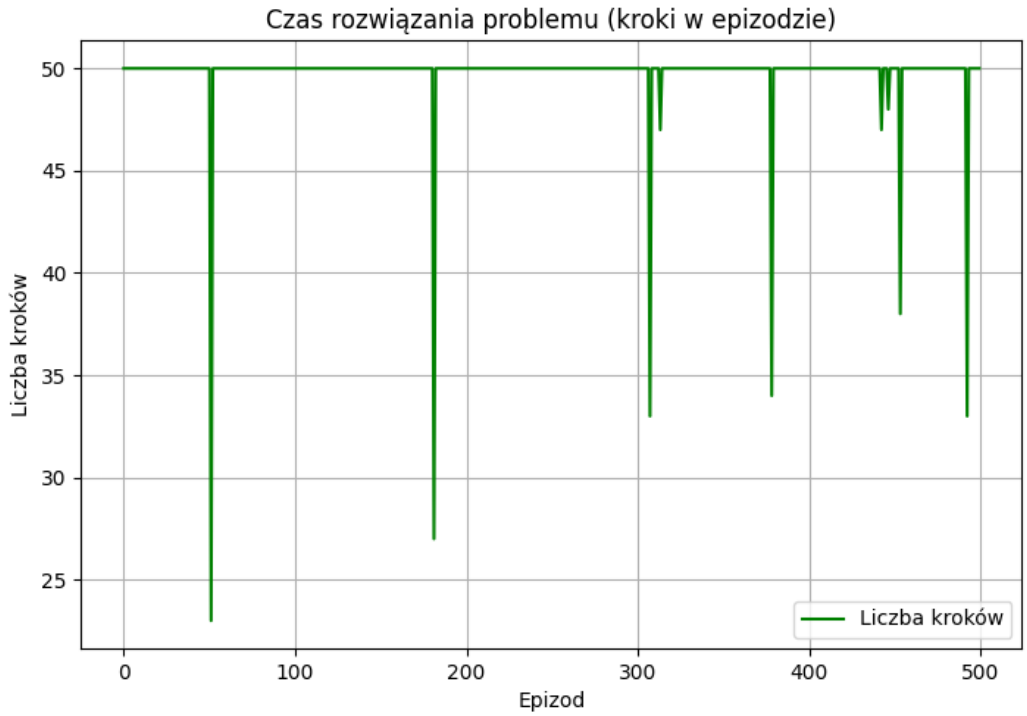


Rys. 6 Zmiana wartości Epsilon w trakcie działania algorytmu Q-learning.

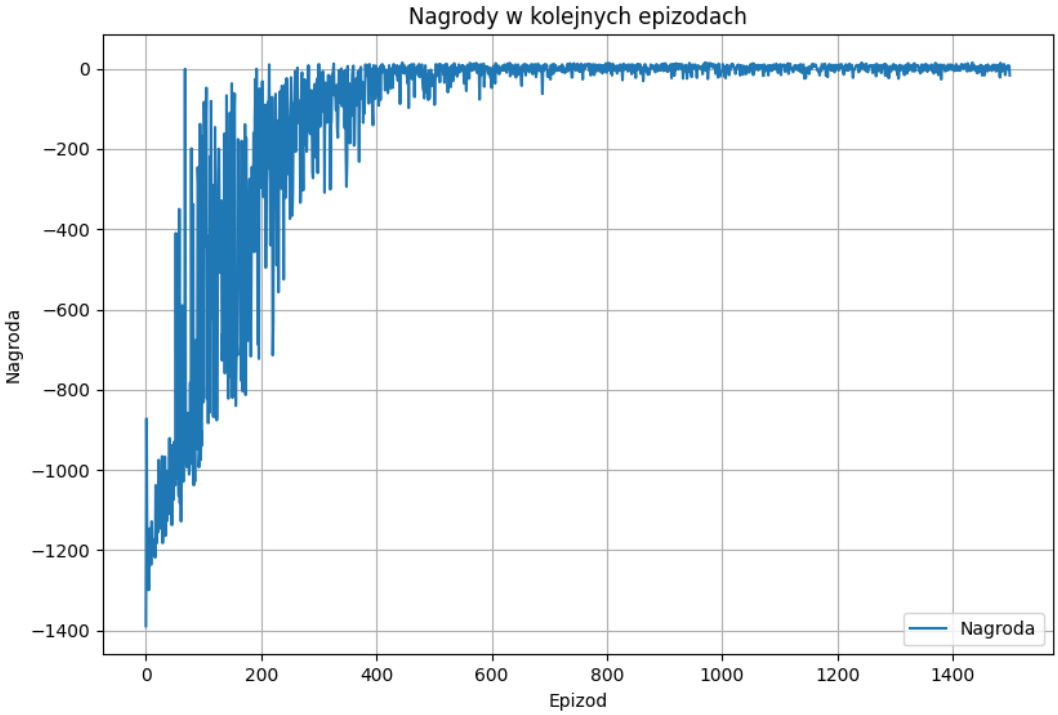
Sprawdzono również wpływ zmiany parametrów otoczenia takich jak maksymalna ilość kroków w epizodzie oraz ilość epizodów, czego wyniki przedstawiono poniżej:



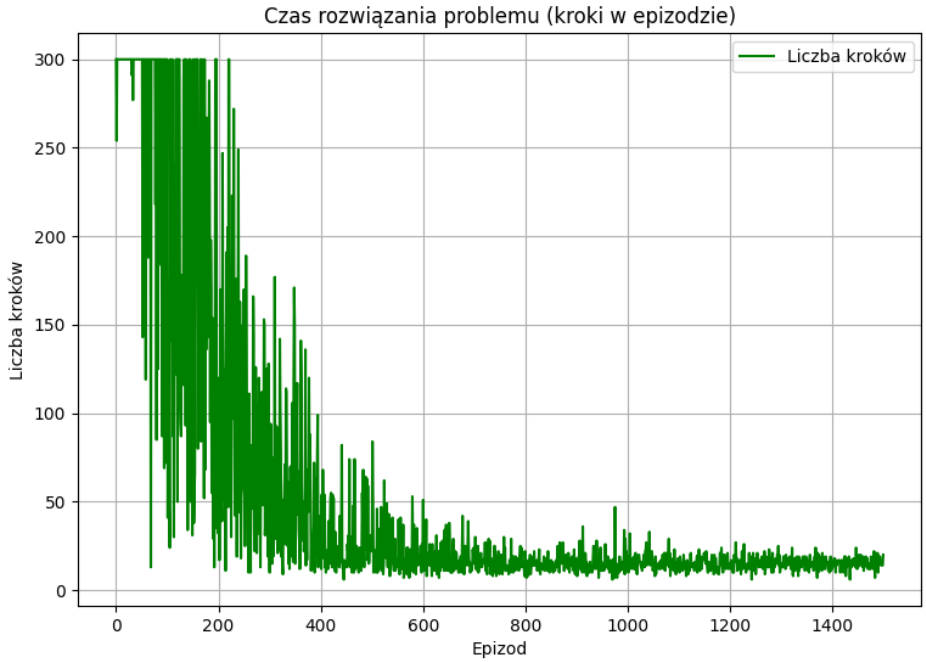
Rys. 7 Wartość nagród w kolejnych epizodach działania algorytmu Q-learning (maks. Liczba kroków = 50, l. epizodów = 500).



Rys. 8 Liczba kroków w kolejnych epizodach (Q-learning) (maks. Liczba kroków = 50, l. epizodów = 500).

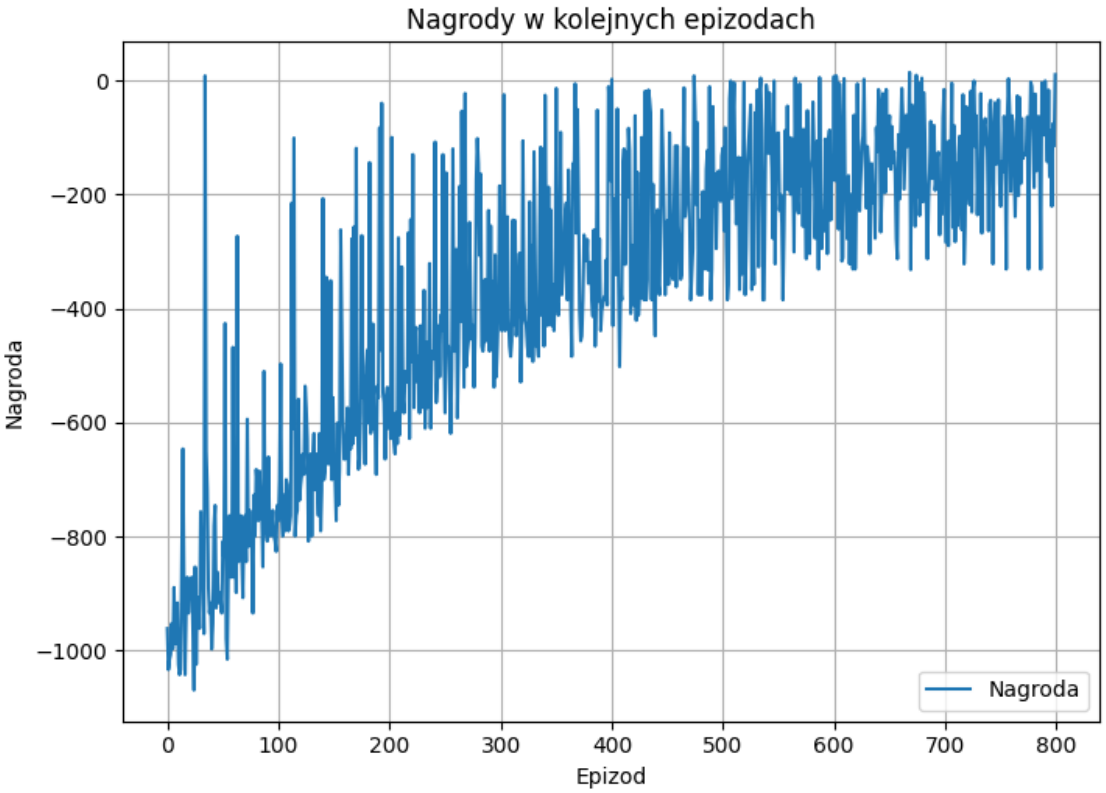


Rys. 9 Wartość nagród w kolejnych epizodach działania algorytmu Q-learning (maks. Liczba kroków = 300, l. epizodów = 1500).

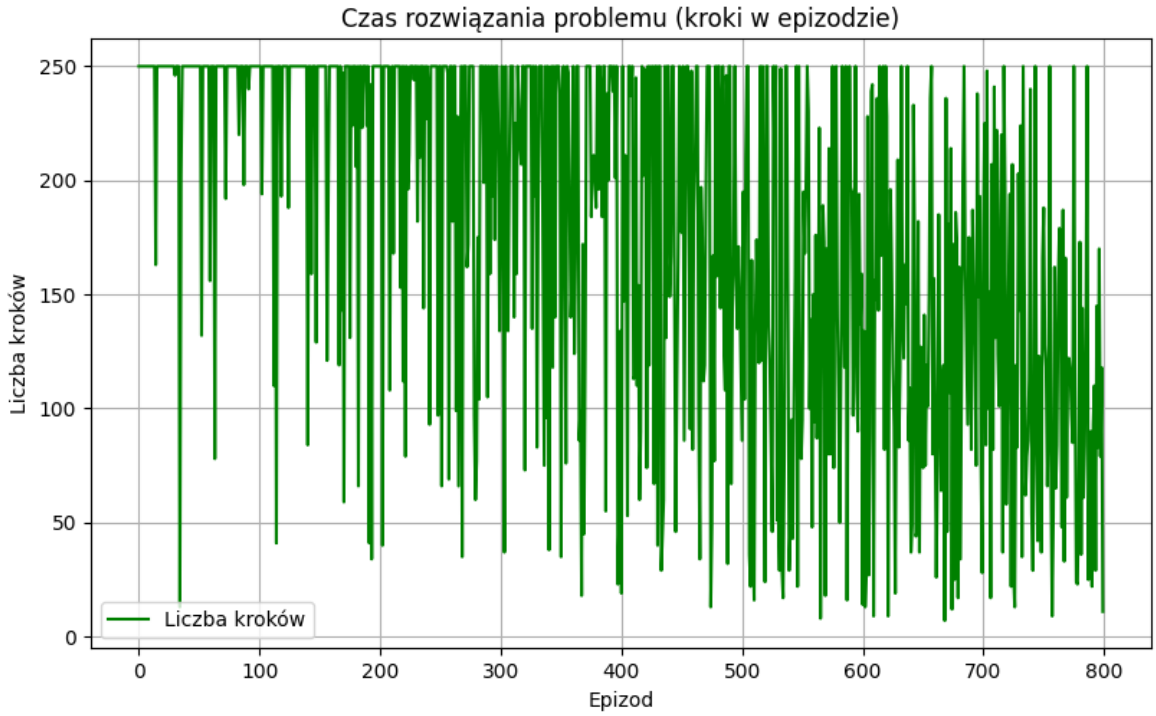


Rys. 10 Liczba kroków w kolejnych epizodach (Q-learning) (maks. Liczba kroków = 300, l. epizodów = 1500).

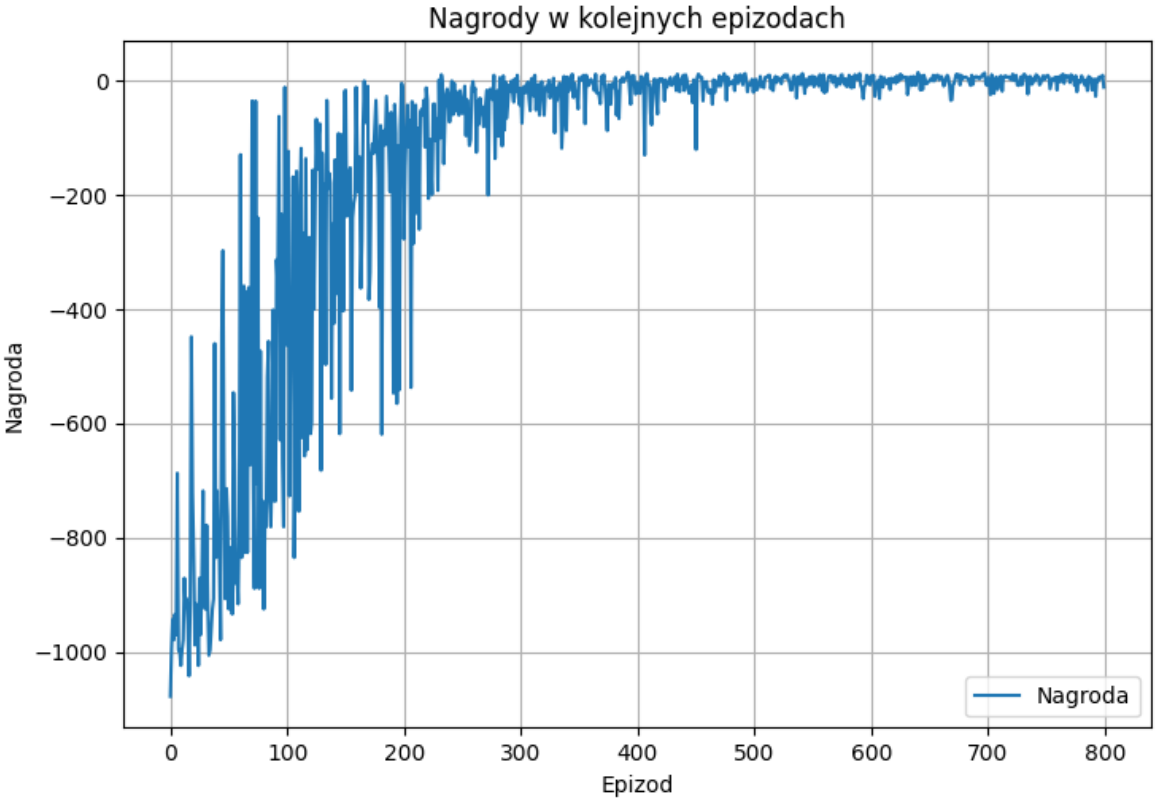
W kolejnym kroku badano wpływ hiperparametrów takich jak współczynnik uczenia α oraz współczynnik dyskontowania γ, czego wyniki przedstawiono poniżej:



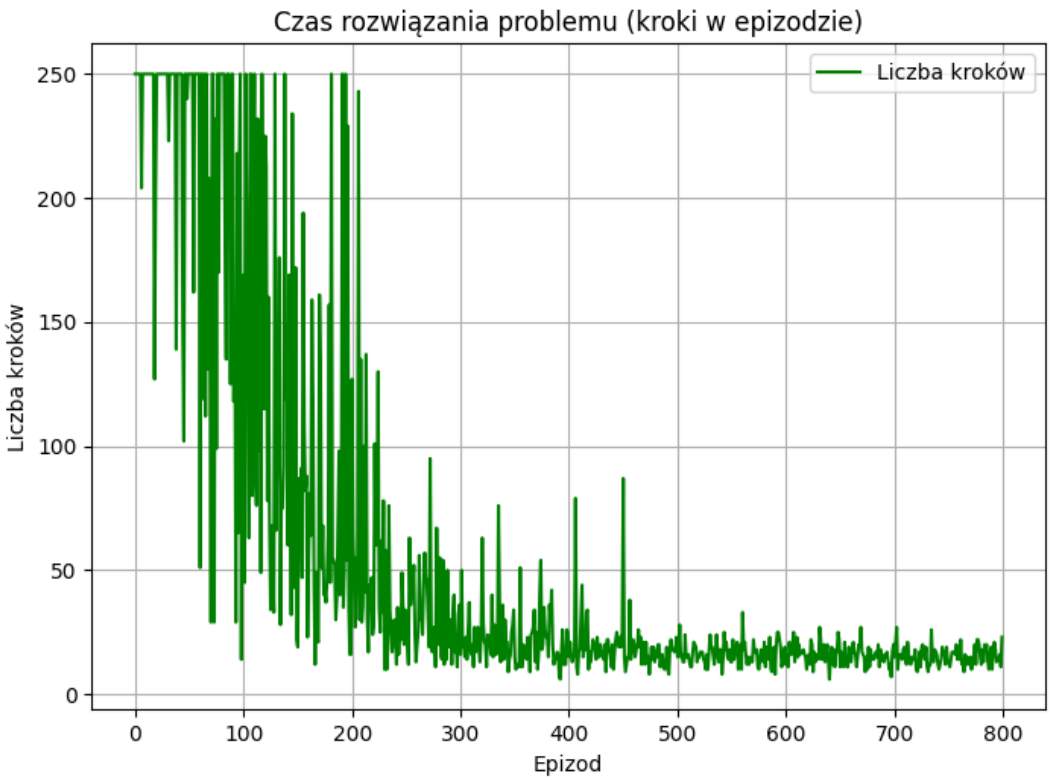
Rys. 11 Wartość nagród w kolejnych epizodach działania algorytmu Q-learning (α = 0.05, γ = 0.7).



Rys. 12 Liczba kroków w kolejnych epizodach (Q-learning) (α = 0.05, γ = 0.7).

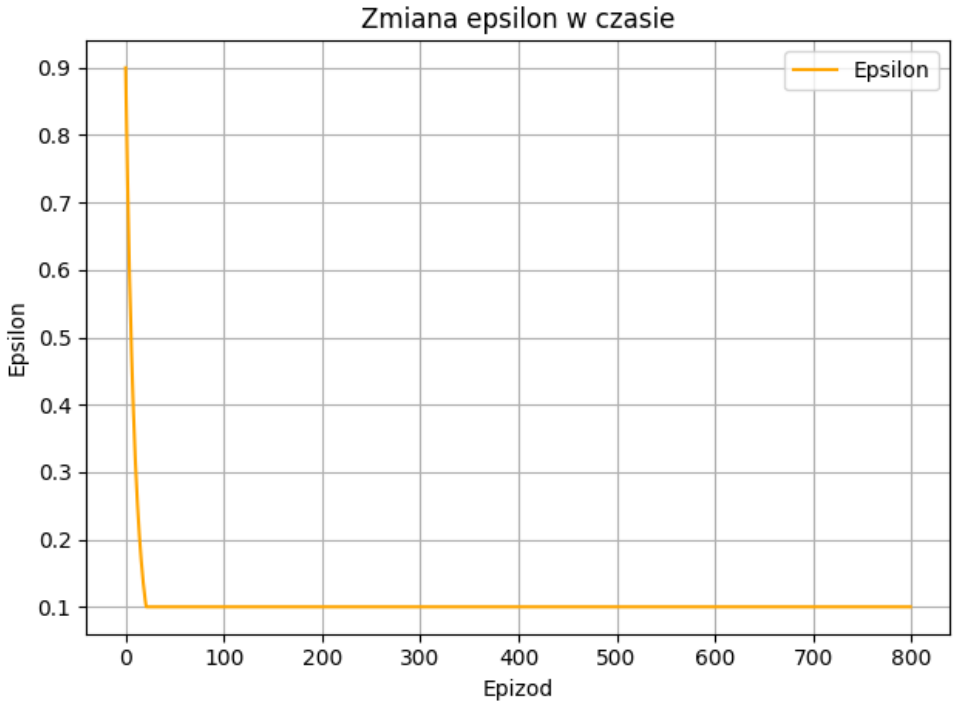


Rys. 13 Wartość nagród w kolejnych epizodach działania algorytmu Q-learning (α = 0.5, γ = 0.99).

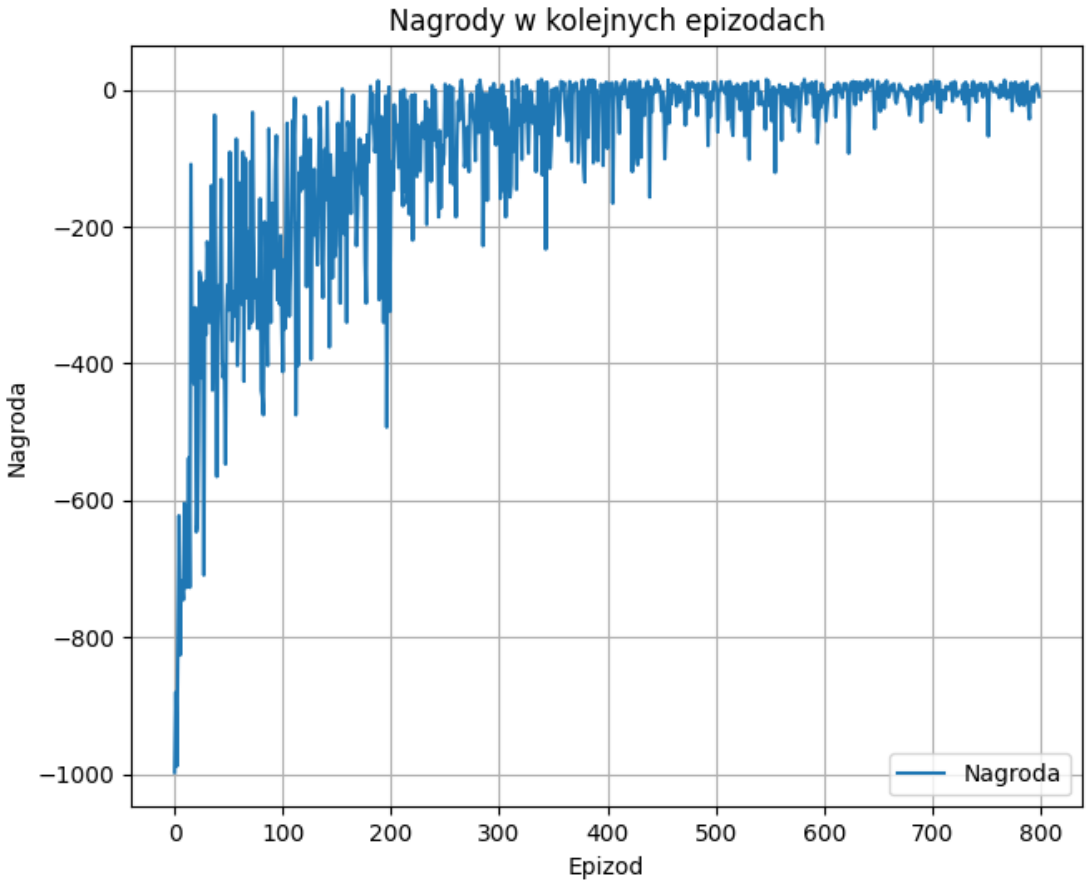


Rys. 14 Liczba kroków w kolejnych epizodach (Q-learning) (α = 0.5, γ = 0.99).

W kolejnym kroku sprawdzono również wpływ krytycznej zmiennej jaką jest epsilon a w zasadzie prędkość jej spadku:

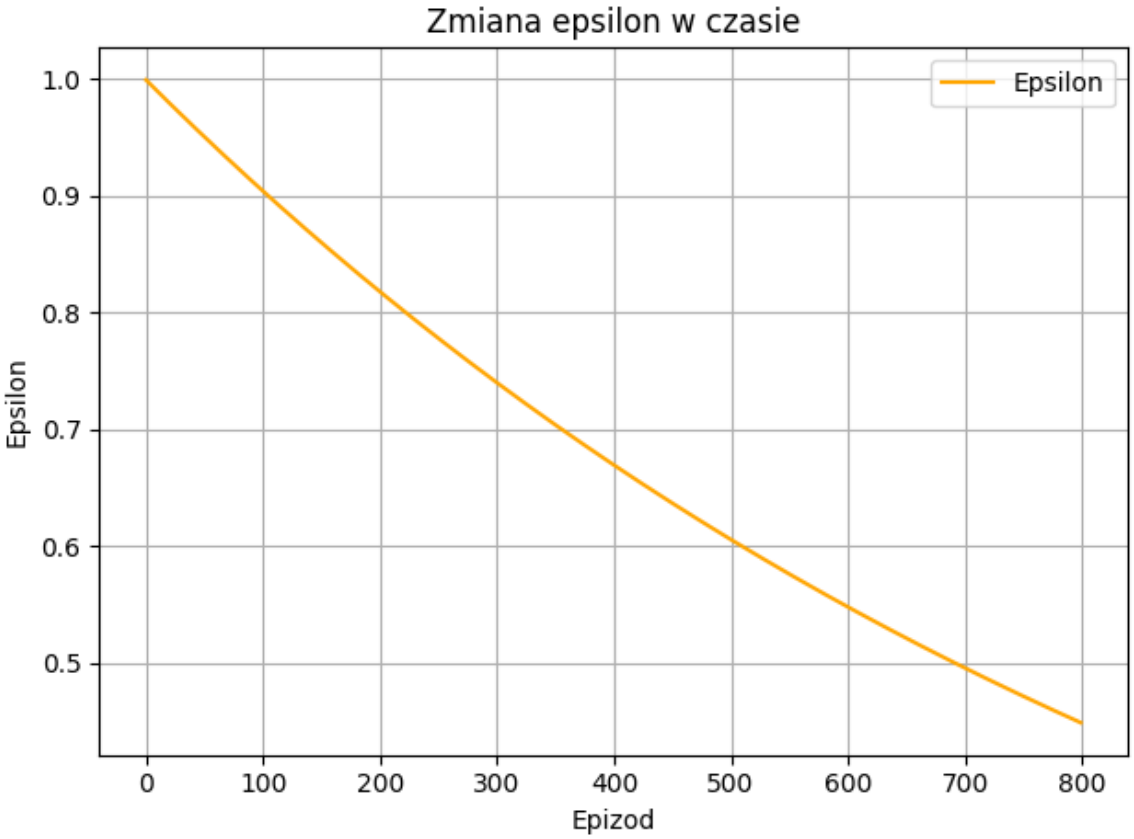


Rys. 15 Zmiana wartości Epsilon w trakcie działania algorytmu Q-learning (wsp. Spadku epsilon = 0.9).

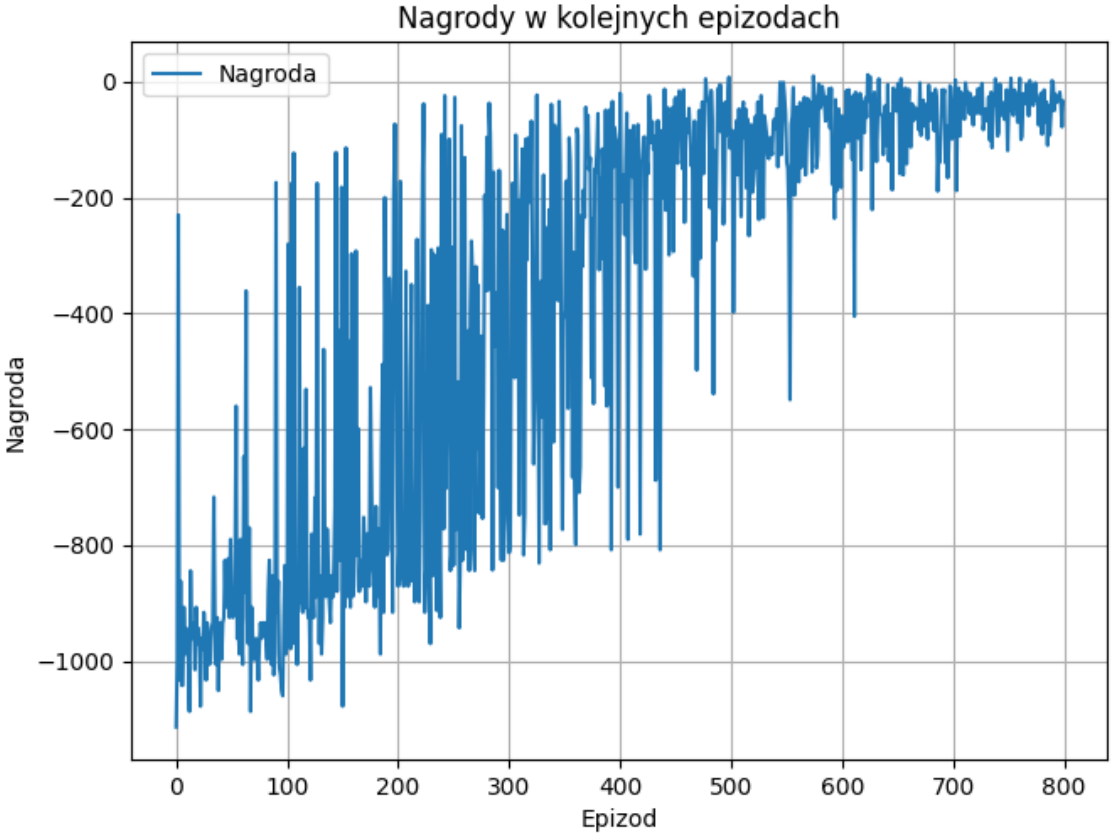


Rys. 16 Wartość nagród w kolejnych epizodach działania algorytmu Q-learning (wsp. Spadku epsilon = 0.9).

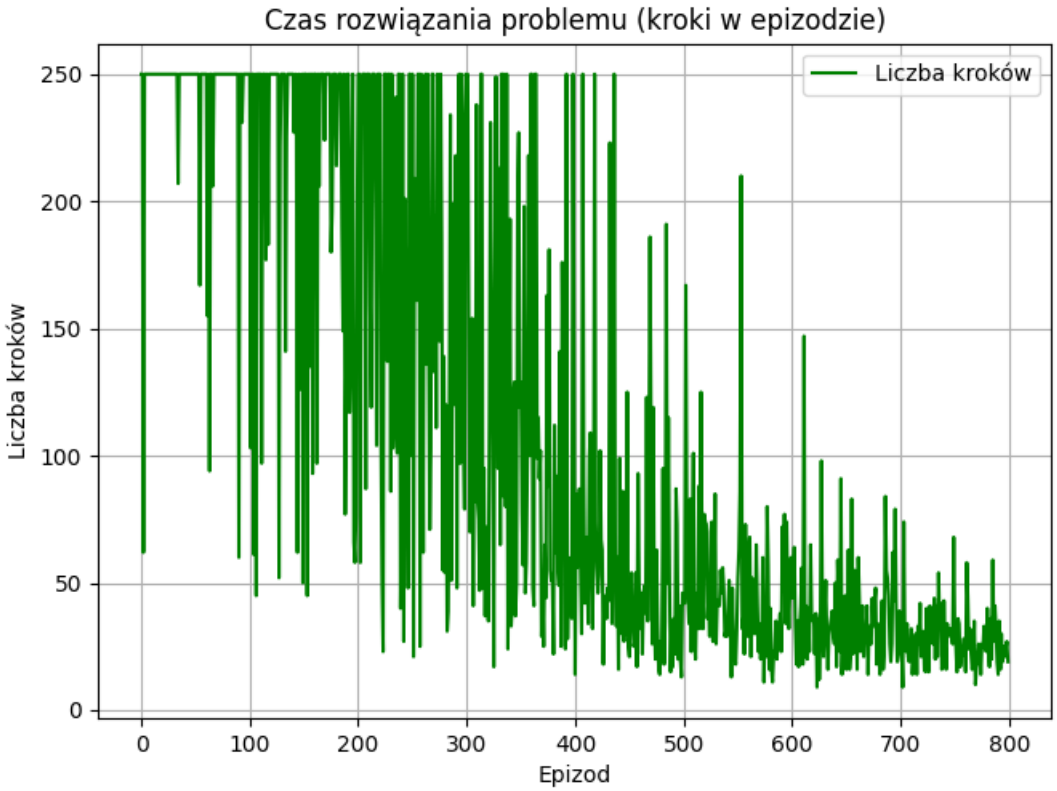
Rys. 17 Liczba kroków w kolejnych epizodach (Q-learning) (wsp. Spadku epsilon = 0.9).



Rys. 18 Zmiana wartości Epsilon w trakcie działania algorytmu Q-learning (wsp. Spadku epsilon = 0.999).

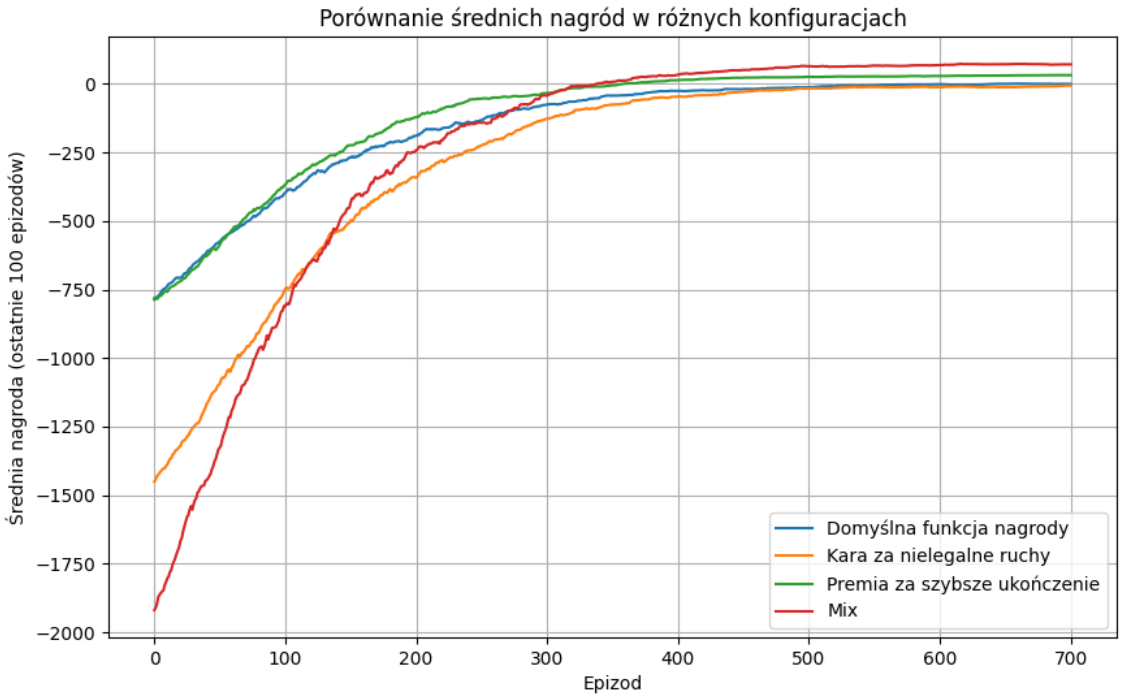


Rys. 19 Wartość nagród w kolejnych epizodach działania algorytmu Q-learning (wsp. Spadku epsilon = 0.999).



Rys. 20 Liczba kroków w kolejnych epizodach (Q-learning) (wsp. Spadku epsilon = 0.999).

Dodatkowo, aby całkowicie przebadać środowisko Taxi-v3 i algorytm Q-learning zaimplementowano zmiany systemu nagród pod kątem pesymistycznym (zwiększając kary za nielegalne ruchy), pod kątem optymistycznym (zwiększając nagrodę za szybsze ukończenie zadania) o raz mix, czyli połączenie obu poprzednich podejść. Na poniższym wykresie przedstawiono wyniki w postaci średniej nagrody z ostatnich 100 epizodów działania algorytmu:



Rys. 21 Porównanie średnich nagród podczas różnych systemów kar i nagród.

4. Wnioski

Implementacja algorytmu DQN z biblioteki Standard Baselines 3 miała na celu przećwiczenie implementacji środowiska dla problemu Taxi-v3 z biblioteki Gymnasium. Charakterystyki w tym przypadku wskazują na zbyt powolny spadek wartości epsilon co powoduje występowanie niestabilności w uczeniu. Lepsze dostrojenie hiperparametrów mogłoby pozwolić na otrzymanie lepszych wyników.

Wyniki otrzymane za pomocą implementacji algorytmu Q-learning pokazują, że praktycznie każda zmiana wartości parametru modelu wpływa na jakość otrzymanych wyników. Skrócenie maksymalnej liczby kroków w epizodzie (rys. 7 i 8) skutkowało spadkiem uzyskanych nagród, co potwierdza, że agent potrzebował więcej czasu na optymalizację ścieżek. Zwiększenie liczby epizodów (rys. 9 i 10) poprawiło wyniki, co wskazuje na lepszą adaptację agenta do środowiska. Wyższy współczynnik uczenia $\alpha = 0.5$ oraz dyskontowania $\gamma = 0.99$ skutkował lepszymi nagrodami końcowymi (rys. 13) w porównaniu do niższych wartości tych współczynników (Rys. 11; $\alpha = 0.05$, $\gamma = 0.7$). Badania nad parametrami α i γ wskazują ich kluczowe znaczenie dla szybkości i stabilności procesu uczenia. Większy współczynnik spadku epsilon wynoszący 0.9, pozwolił na dłuższą eksploatację, co skutkowało lepszymi wynikami końcowymi pokazanymi na rys. 16. Przy mniejszym współczynniku wynoszącym 0.999, wyniki były bardziej stabilne, ale mogło to ograniczać eksplorację nowych działań.

Zwiększenie kar za nielegalne ruchy spowodowało bardziej zachowawczą strategię agenta, a zwiększenie nagród za szybsze ukończenie zadania zachęcało do agresywnego optymalizowania działań. Kombinacja obu podejść (system „mix”) wydaje się najbardziej efektywna, pozwalając na lepsze wyniki w różnych warunkach.