Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Sprawozdanie

Wprowadzenie do sztucznej inteligencji

Ćwiczenie nr. 6

Mikołaj Bańkowski

Numer albumu 310408

prowadzący
Grzegorz Rypeść

Warszawa 2023

1. Proszę napisać w raporcie ile stanów może mieć środowisko.

ANALIZA STANU GRY

Rozmiar planszy:

- Plansza ma wymiary: 300×300 pikseli
- Każdy blok ma rozmiar: 30×30 pikseli
- Liczba bloków w pionie i poziomie wynosi: 10×10=100 bloków

Elementy na planszy:

- Każdy blok może zawierać jedno z następujących: puste miejsce, część ciała węża, jedzenie, ścianę (jeśli jest na brzegu planszy).
- Stan gry opisuje, co znajduje się w każdym bloku na planszy.

Stan węża:

- Wąż składa się z głowy i ciała.
- Głowa węża może być skierowana w jednym z czterech kierunków: w górę, w dół, w lewo, w prawo.
- Długość węża może się zmieniać w zależności od liczby zjedzonych jedzeń.

Jedzenie:

- Jedzenie może pojawić się na dowolnym bloku, który nie jest zajęty przez węża ani ścianę.

OBLICZANIE LICZBY STANÓW

Stan węża:

- Pozycja głowy: 100 możliwych pozycji.
- Kierunek głowy: 4 możliwe kierunki.
- Pozycje ciała: liczba możliwych konfiguracji ciała zależy od długości węża i każda pozycja ciała może być na jednym z 100 bloków, ale musi być połączona zgodnie z regułami gry.

Pozycja jedzenia:

- Jedzenie może pojawić się na każdym bloku, który nie jest zajęty przez węża ani ścianę. Ponieważ pozycje jedzenia są zależne od aktualnej pozycji węża, możliwe konfiguracje zmniejszają się wraz z rosnącą długością węża.

Aby uprościć obliczenia, możemy przyjąć następujące założenia:

- Każdy blok może być w jednym z kilku stanów: pusty, część węża, jedzenie.
- Głowa węża może mieć 4 różne kierunki.

W przybliżeniu możemy określić liczbę stanów w ten sposób:

- Liczba bloków: 100
- Każdy blok może mieć jeden z 3 stanów (pusty, jedzenie, część węża).
- Dodatkowo uwzględniamy kierunki głowy węża: 4 kierunki.

Zatem liczba stanów S może być szacowana jako:

$$S \approx 3^{100} \cdot 4$$

To szacowanie daje ogromną liczbę stanów, co pokazuje złożoność problemu.

REDUKCJA LICZBY STANÓW

Jeżeli agent używa 21 dyskretnych atrybutów a każdy atrybut może przyjąć wartość 0 lub 1, to liczba możliwych obserwacji wynosi:

$$2^{21} = 2097152$$

Dla każdej obserwacji agent podejmuje jedną z 4 możliwych akcji, więc liczba kombinacji obserwacja-akcja wynosi:

$$S = \ 2097152 \cdot 4$$

PODSUMOWANIE:

Złożoność środowiska gry węża jest ogromna z powodu liczby możliwych konfiguracji pozycji węża, kierunków i położenia jedzenia. Choć dokładne obliczenie liczby stanów jest trudne do określenia z powodu dynamicznej natury gry, szacunkowe obliczenia pokazują, że mamy do czynienia z bardzo dużą przestrzenią stanów.

W praktyce jednak, liczba stanów, którą agent będzie eksplorował, będzie ograniczona przez efektywną reprezentację stanu, jak np. poprzez dyskretne atrybuty opisujące najbliższe otoczenie węża, co znacząco redukuje przestrzeń stanów do zarządzanego rozmiaru.

Redukcja liczby stanów znacząco zmniejsza złożoność problemu w porównaniu do pełnego stanu gry, jednocześnie pozwalając agentowi efektywnie uczyć się i podejmować decyzje.

2. Atrybuty stanów

Położenia jedzenia względem głowy węża

```
is_up = int(gs["food"][1] < gs["snake_body"][-1][1])
is_right = int(gs["food"][0] > gs["snake_body"][-1][0])
is_down = int(gs["food"][1] > gs["snake_body"][-1][1])
is_left = int(gs["food"][0] < gs["snake_body"][-1][0])
```

Czy w pobliżu głowy węża znajduje się jego ciało

```
# Define relative coordinates of each adjacent to head field
relative_coords = [(-30, -30), (0, -30), (30, -30), (30, 0),
                   (30, 30), (0, 30), (-30, 30), (-30, 0)]
head_x, head_y = gs["snake_body"][-1]

tail = [0] * 8
# Check for each adjacent field if there is a tail
for i, (dx, dy) in enumerate(relative_coords):
    if (head_x + dx, head_y + dy) in gs["snake_body"][:-1]:
        tail[i] = 1
```

Czy w pobliżu głowy węża znajduje się jego ciało

```
wall = [0] * 8
bounds_x, bounds_y = gs["bounds"]
for i, (dx, dy) in enumerate(relative_coords):
    if (head_x + dx < 0) or (head_x + dx >= bounds_x) or \
            (head_y + dy < 0) or (head_y + dy >= bounds_y):
        wall[i] = 1
```

Kierunek ruchu węża

```
gs["snake_direction"].value
return (is_up, is_right, is_down, is_left, *tail, *wall, gs["snake_direction"].value)
```

3. Implementacja klasy QLearningAgent

W celu polepszenia wyników osiąganych przez algorytm QLearning implementację klasy QLearningAgent rozszerzamy o informację na temat położenia ścian i ogona węża względem jego głowy. W tym celu do kod metody game_state_to_observation modyfikujemy następująco:

```
# Define relative coordinates of each adjacent to head field
relative_coords = [(-30, -30), (0, -30), (30, -30), (30, 0),
                   (30, 30), (0, 30), (-30, 30), (-30, 0)]
head_x, head_y = gs["snake_body"][-1]

tail = [0] * 8
# Check for each adjacent field if there is a tail
for i, (dx, dy) in enumerate(relative_coords):
    if (head_x + dx, head_y + dy) in gs["snake_body"][:-1]:
        tail[i] = 1

wall = [0] * 8
bounds_x, bounds_y = gs["bounds"]
for i, (dx, dy) in enumerate(relative_coords):
    if (head_x + dx < 0) or (head_x + dx >= bounds_x) or \
            (head_y + dy < 0) or (head_y + dy >= bounds_y):
        wall[i] = 1

return (is_up, is_right, is_down, is_left, *tail, *wall, gs["snake_direction"].value)
```

Wówczas długość zwracanego wektora zwiększa się o 16 i zawiera dodatkowe binarne wartości informujące o obecności ściany lub ogona na każdym z 8 pól sąsiadujących z polem na którym znajduje się głowa węża. Zmianie ulega również rozmiar Q-tabeli:

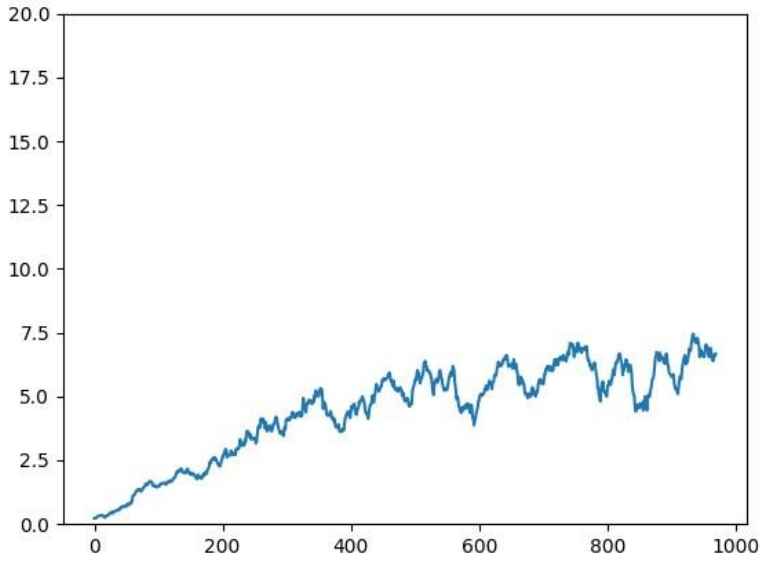
```
class QLearningAgent:
    mik00laj *
    def __init__(self,
                 block_size,
                 bounds,
                 epsilon=0.1,
                 discount=0.9,
                 is_training=True,
                 load_qfunction_path=None,
                 lr=0.01,
                 qtable_size=(2, 2, 2, 2, *(2 for i in range(16)), 4, 4)
                 ):
```

Następnie wykonane modyfikacje testujemy dla różnych wartości ε i γ (dyskonty), wykresy średniej wartości nagrody za epizod podczas treningu prezentują się następująco:

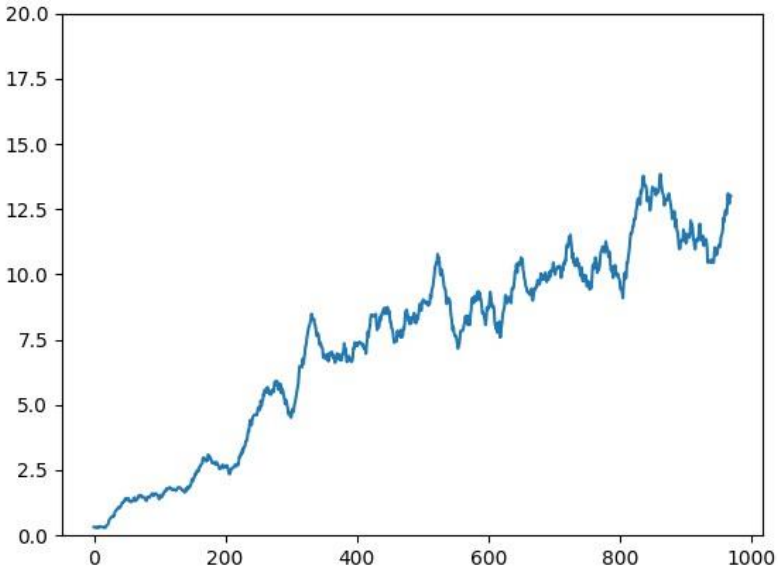
Episodes 1000

TRENOWANIE

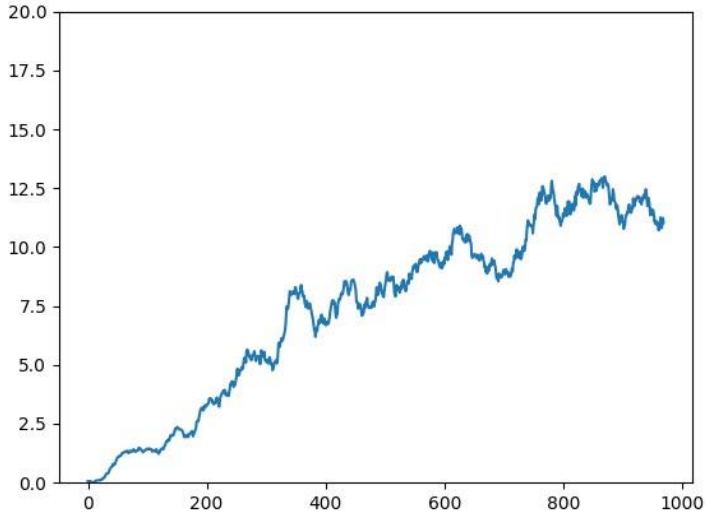
eps: 0.1 discount: 0.99 learning_rate=0.01



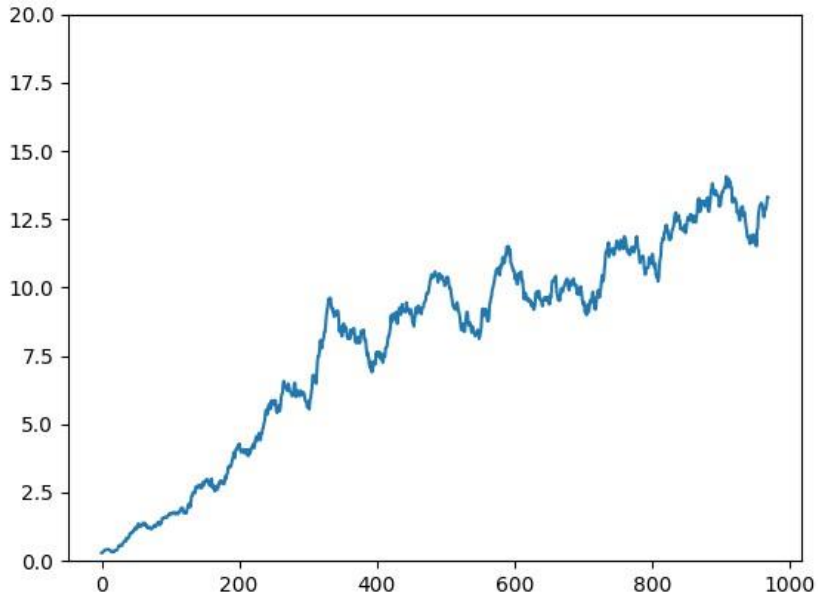
eps: 0.01 discount: 0.99 learning_rate=0.01



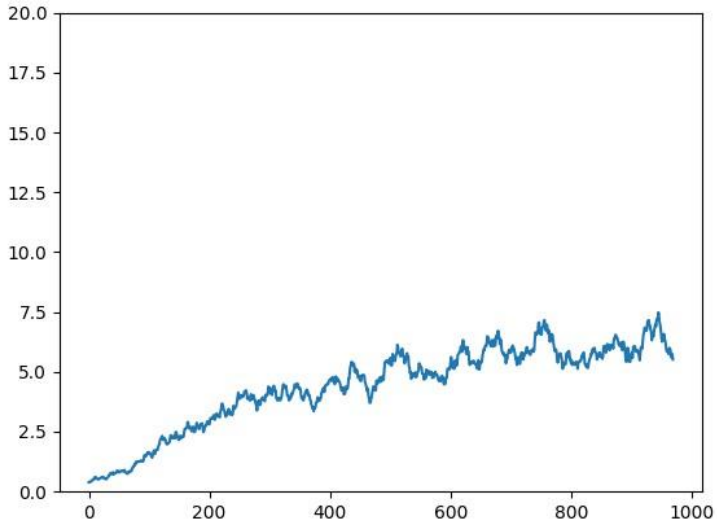
eps: 0.001 discount: 0.99 learning_rate=0.01



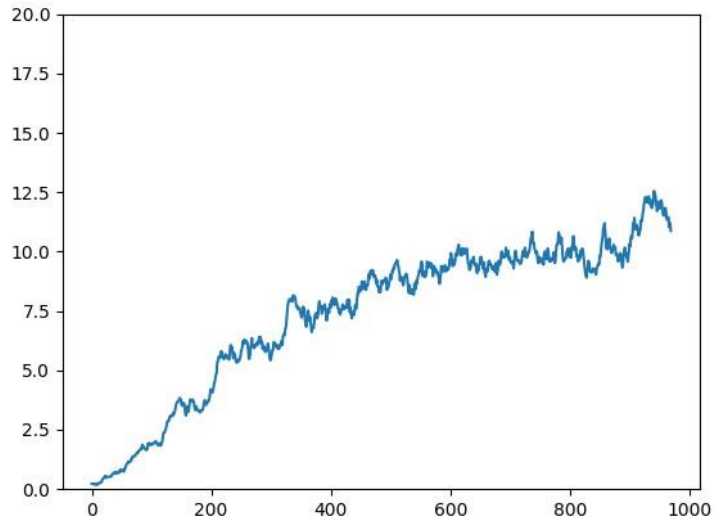
eps: 0.0001 discount: 0.99 learning_rate=0.01



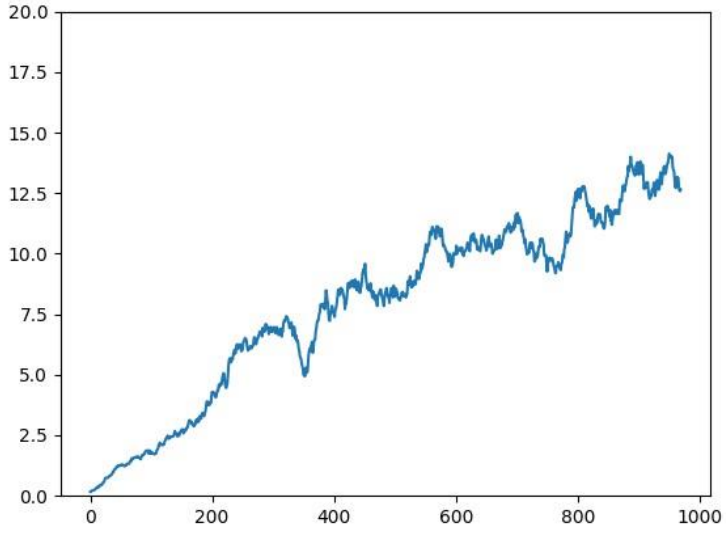
eps: 0.1 discount: 0.9 learning_rate=0.01



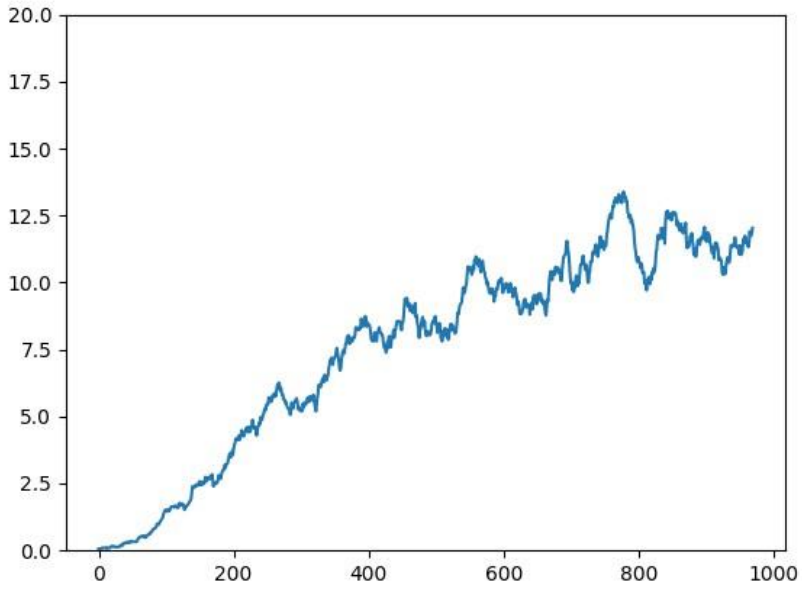
eps: 0.01 discount: 0.9 learning_rate=0.01



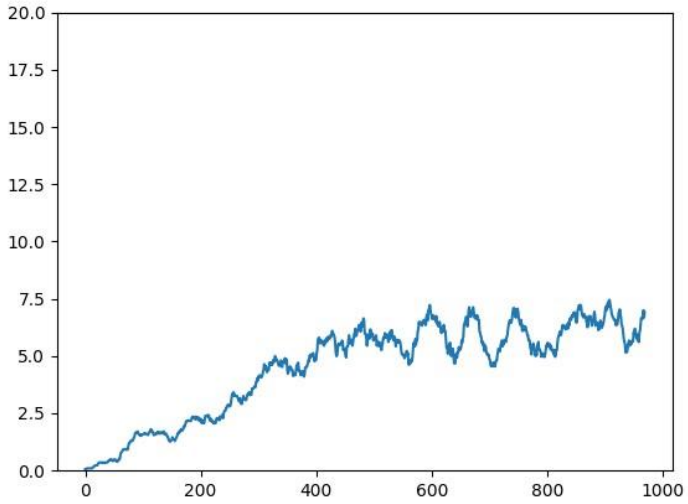
eps: 0.001 discount: 0.9 learning_rate=0.01



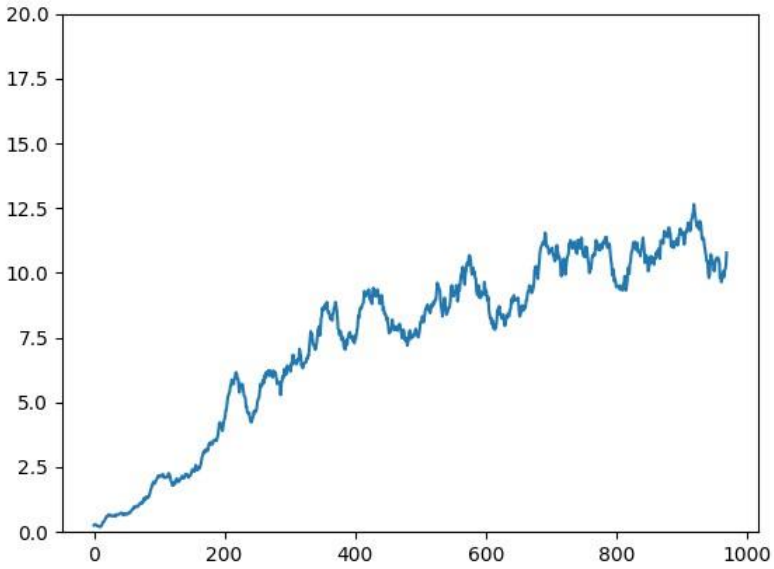
eps: 0.0001 discount: 0.9 learning_rate=0.01



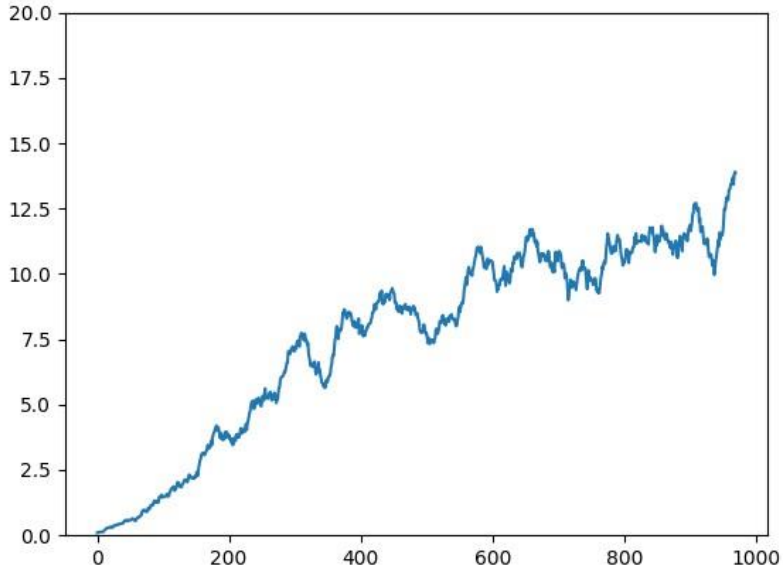
eps: 0.1 discount: 0.8 learning_rate=0.01



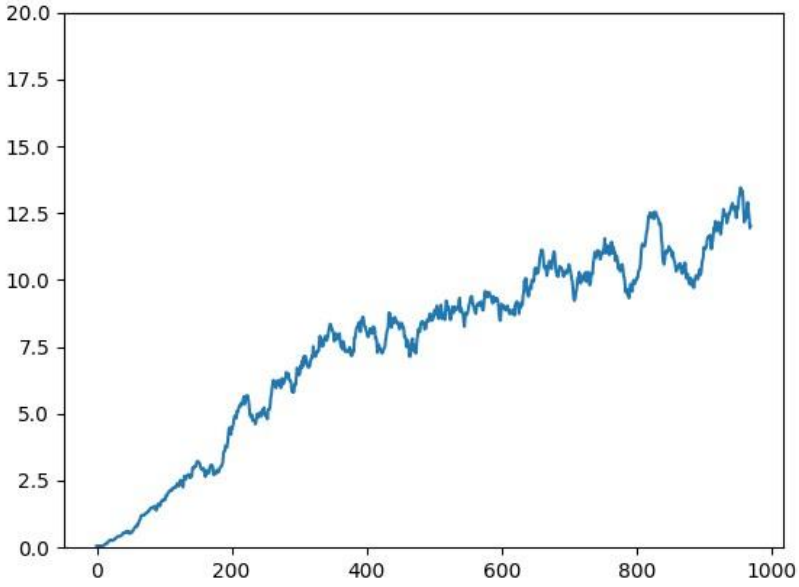
eps: 0.01 discount: 0.8 learning_rate=0.01



eps: 0.001 discount: 0.8 learning_rate=0.01

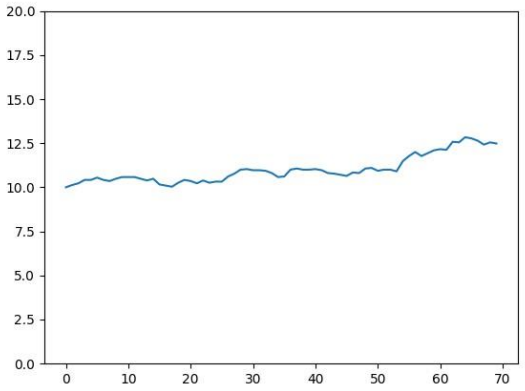


eps: 0.0001 discount: 0.8 learning_rate=0.01

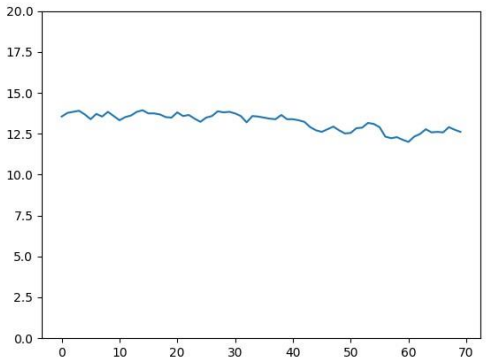


TESTOWANIE

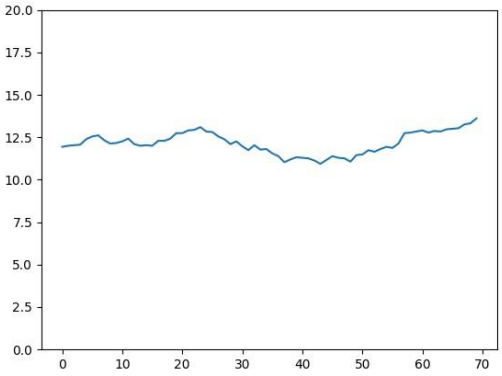
eps: 0 discount: 0.99 learning_rate=0.01



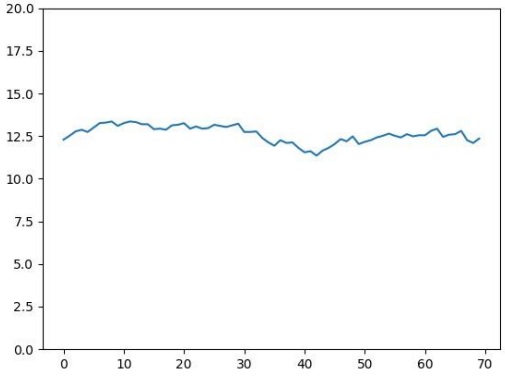
1



2



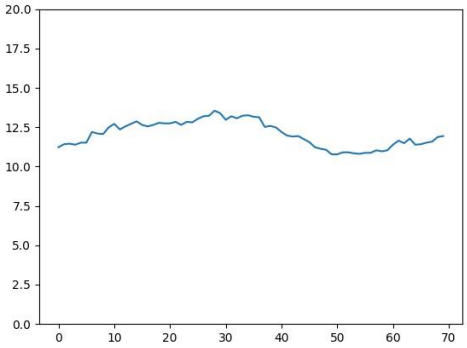
3



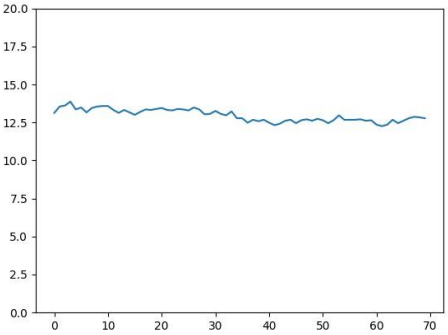
4

Lp.	Model	Średni wynik
1	q_function_eps0.1_discount0.99_lr0.01_train	10.18
2	q_function_eps0.01_discount0.99_lr0.01_train	12.75
3	q_function_eps0.001_discount0.99_lr0.01_train	11.74
4	q_function_eps0.0001_discount0.99_lr0.01_train	12.90

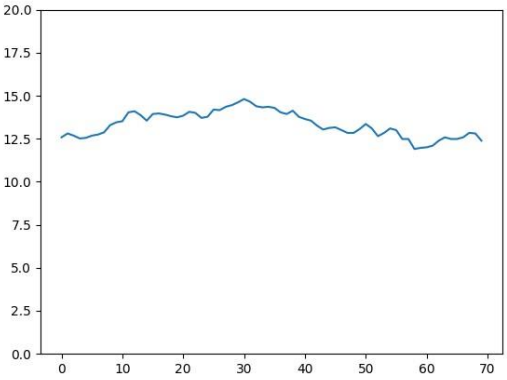
1



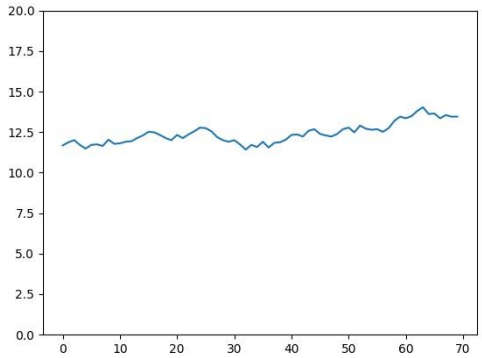
2



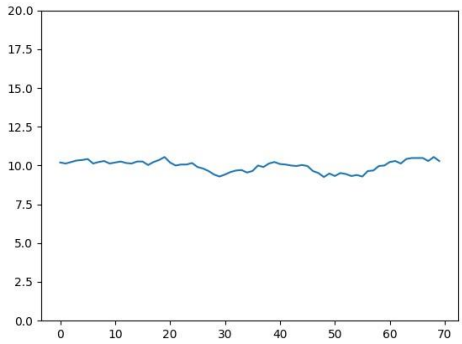
3



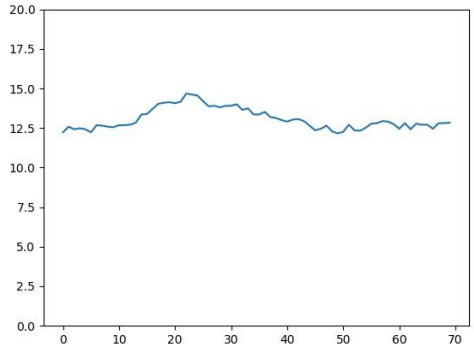
4



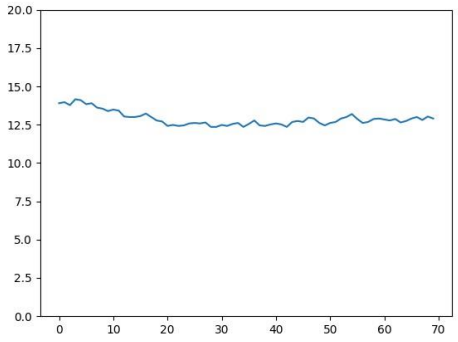
Lp.	Model	Średni wynik
1	q_function_eps0.1_discount0.9_lr0.01_train	10.61
2	q_function_eps0.01_discount0.9_lr0.01_train	11.4
3	q_function_eps0.001_discount0.9_lr0.01_train	12.91
4	q_function_eps0.0001_discount0.9_lr0.01_train	12.78



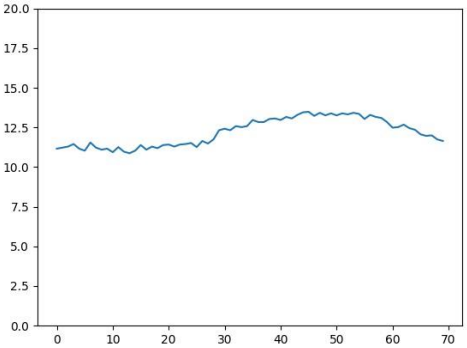
1



2



3



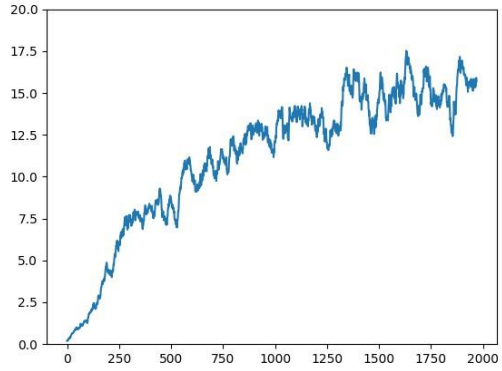
4

Lp.	Model	Średni wynik
1	q_function_eps0.1_discount0.8_lr0.01_train	10.50
2	q_function_eps0.01_discount0.8_lr0.01_train	12.16
3	q_function_eps0.001_discount0.8_lr0.01_train	12.72
4	q_function_eps0.0001_discount0.8_lr0.01_train	11.63

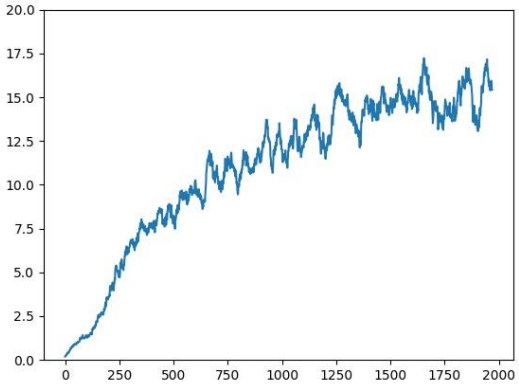
Episodes 2000

TRENOWANIE

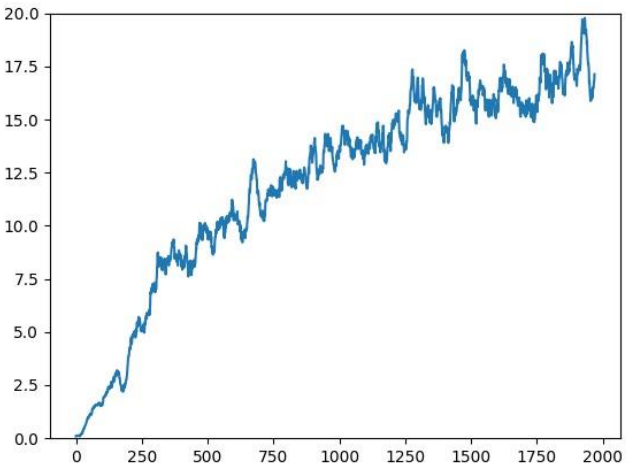
eps: 0.001 discount: 0.99 learning_rate=0.01



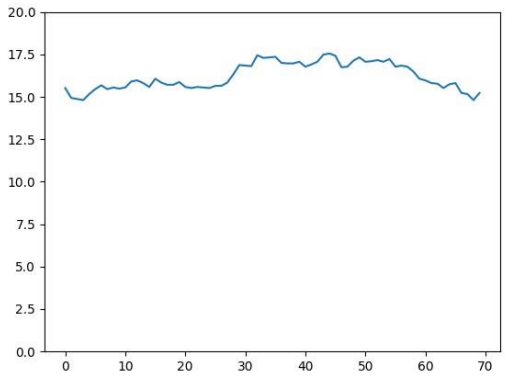
eps: 0.001 discount: 0.9 learning_rate=0.01



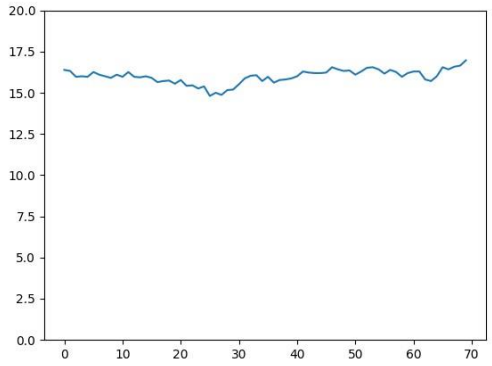
eps: 0.001 discount: 0.8 learning_rate=0.01



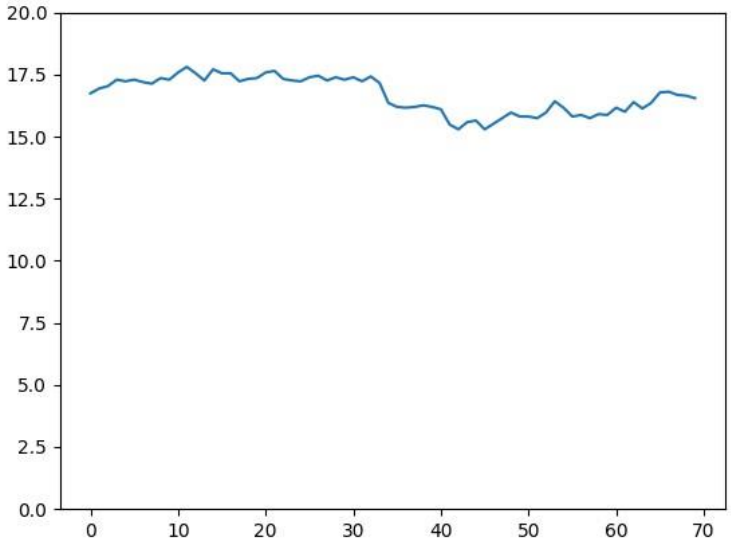
TESTOWANIE



1



2



3

Lp.	Model	Średni wynik
1	q_function_eps0.001_discount0.99_lr0.01_train	16.19
2	q_function_eps0.001_discount0.90_lr0.01_train	15.84
3	q_function_eps0.001_discount0.80_lr0.01_train	17.29

Czym jest probem balansu między eksploracją, a eksploatacją (ang. exploration-exploatation trade-off)?

Odnosi się do sposobu, w jaki agent decyduje, czy eksplorować nowe stany i akcje, czy też eksploatować już znane informacje w celu maksymalizacji swojej funkcji wartości. W algorytmie Q-learning agent uczy się podejmować decyzje w dynamicznym środowisku poprzez eksplorację różnych akcji i obserwowanie ich skutków lub eksploatację najlepszych znanych akcji, bazując na aktualnie oszacowanych wartościach Q (funkcji wartości). Trade-off eksploracji i eksploatacji w Q-learningu jest zazwyczaj reprezentowany przez parametr zwany "epsilon-greedy". Ten parametr kontroluje, w jakim stopniu agent będzie eksplorował nowe akcje (przez losowy wybór) w porównaniu do eksploatowania najlepszej znanej akcji (poprzez wybór akcji o najwyższej wartości Q).