WSI – Ćwiczenie 6.

Uczenie ze wzmocnieniem (Q-learning)

Dawid Bartosiak 318361

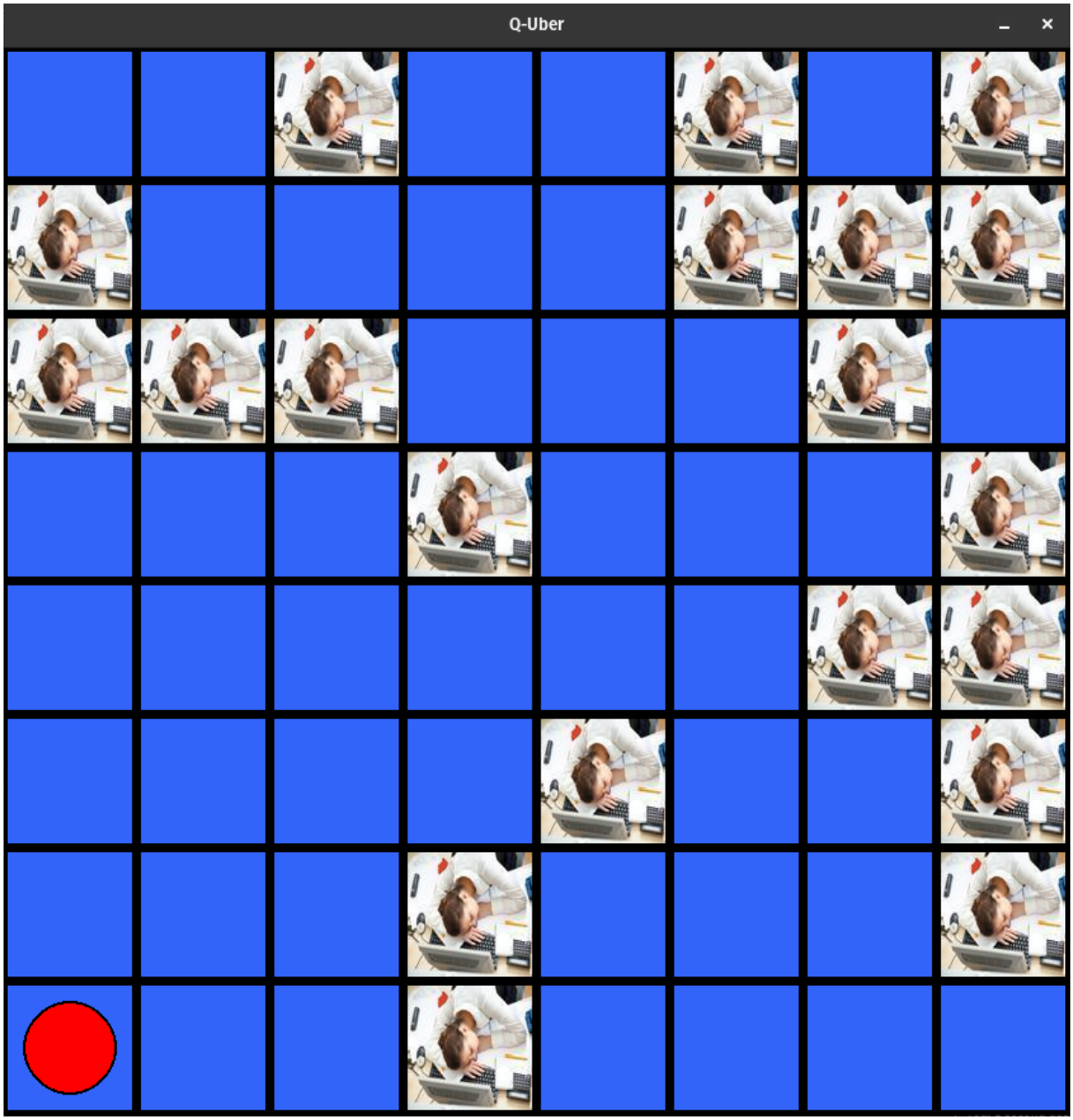
---

1. Treść zadania – *Q-Uber*

Elon Piżmo konstruuje autonomiczne samochody do swojego najnowszego biznesu. Dysponujemy planszą NxN (domyślnie N=8) reprezentującą pole do testów jazdy. Na planszy jako przeszkody stoją jego bezpłatni stażyści. Mamy dwa autonomiczne samochody: Random-car, który kierunek wybiera rzucając kością (błądzi losowo po planszy) oraz Q-uber, który uczy się przechodzić ten labirynt (używa naszego algorytmu). Samochody zaczynają w tym samym zadanym punkcie planszy i wygrywają, jeśli dotrą do punktu końcowego, którym jest inny punkt planszy. Istnieje co najmniej jedna ścieżka do startu do końca. Elon oszczędzał na module do liczenia pierwiastków, dlatego samochody poruszają się przy użyciu metryki Manhattan (góra, dół, lewo, prawo). Jeżeli samochód natrafi na stażystę to kończy bieg i przegrywa. Analogicznie jak wejdzie na punkt końca to wygrywa i również nie kontynuuje dalej swojej trasy. Celem agenta jest minimalizacja pokonywanej trasy.

2. Labirynt

Plansza, po której poruszają się samochody, to lista zawierająca N list o długości N, przy czym N to długość boku planszy. Wszystkie elementy inicjowane są wartością 0, oznaczającą pole, po którym samochody mogą się dowolnie poruszać. Następnie losowane są miejsca dla stażystów, które oznacza się poprzez zamianę zera na jedynkę, przy czym miejsca te muszą być unikalne i z założenia nie mogą znajdować się na punktach: startowym i końcowym. Poprawność wygenerowanego labiryntu jest sprawdzana za pomocą funkcji *dfs*, która znajduje pola na planszy, do których można dotrzeć z punktu początkowego. Jeżeli na liście odwiedzonych punktów znajduje się punkt końcowy, labirynt jest poprawny. Labirynt można zapisać jak i odczytać z plików tekstowych.



Rysunek 1. Przykładowy wygenerowany labirynt.

3. Taksówka Elona

Plansza reprezentuje środowisko, w którym taksówka musi osiągnąć określony punkt końcowy, rozpoczynając od punktu początkowego umieszczonego na przeciwnym krańcu planszy. Wyróżnia się kilka kluczowych elementów tej symulacji:

1. **Punkty początkowy i końcowy:** Taksówka rozpoczyna swoją trasę w ustalonym punkcie początkowym i porusza się w kierunku ustalonego punktu końcowego, gdzie znajduje się jej cel.
2. **Zakończenie jazdy:** Jazda taksówki kończy się w dwóch przypadkach: gdy taksówka uderzy w stażystę lub gdy dotrze do punktu końcowego.
3. **Wybór akcji:** Taksówka może wykonywać cztery rodzaje ruchów, które są numerowane od 0 do 3, reprezentujące ruch w lewo, w górę, w prawo i w dół. Wybór ruchu zmienia współrzędne taksówki na planszy.
4. **Sprawdzanie poprawności ruchu i nagrody:** Po wykonaniu ruchu, sprawdzana jest jego poprawność. Jeśli taksówka uderzy w ścianę, wraca na poprzednie współrzędne i otrzymuje karę -10 punktów, ale nie kończy jazdy. Uderzenie w stażystę kończy jazdę z tą samą karą punktową. Dotarcie do celu przynosi nagrodę 20 punktów.
5. **Minimalizacja liczby ruchów:** Aby zachęcić do jak najkrótszej trasy, za każdy ruch taksówki, który nie kończy się uderzeniem w ścianę lub stażystę, otrzymuje ona karę -1 punktu. To zachęta do minimalizowania liczby ruchów potrzebnych do dotarcia do celu.

3.1. Q-learning

Każda pozycja taksówki na planszy jest reprezentowana jako unikalny stan. Numer stanu wylicza się poprzez wzór, który uwzględnia współrzędne taksówki (x, y), mnożąc współrzędną x przez liczbę rzędów planszy i dodając do tego wartość współrzędnej y. Kluczowy aspekt uczenia ze wzmocnieniem realizowany jest w metodzie o nazwie drive. Metoda ta odpowiada za symulację całej trasy taksówki, od momentu rozpoczęcia jazdy aż do zakończenia. Przebieg trasy jest determinowany przez parametry takie jak epsilon, beta, gamma oraz przez tablicę q-learningu, którą przekazuje się jako parametr. Tablica ta jest kluczowa, gdyż jej poprawne wykształcenie wymaga wielu prób i błędów wykonanych przez taksówki. Podczas symulacji, taksówka podejmuje decyzje o swoich ruchach. Wybór ten zależy od wartości parametru epsilon, który określa prawdopodobieństwo podjęcia ruchu losowego w stosunku do ruchu opartego na danych z tablicy q-learningu. Jeśli wylosowana liczba z przedziału od 0 do 1 jest mniejsza niż epsilon, ruch taksówki jest losowy. W przeciwnym wypadku, wybiera się akcję, która ma najwyższą wartość w odpowiednim rzędzie tablicy. Po wykonaniu ruchu, taksówka otrzymuje pewną nagrodę, a tablica q-learningu jest aktualizowana za pomocą funkcji update_q_table. Ta aktualizacja polega na modyfikacji wartości w tablicy odpowiadającej poprzedniemu stanowi taksówki i wykonanej akcji. Wzór tej operacji wygląda następująco

*q_table[state, action] += beta * (reward + gamma * next_state_max - q_table[state, action])*

4. Random taxi

Samochód poruszający się po labiryncie w losowy sposób ma nikłe szanse na jego przejście, zwłaszcza zaczynając za każdym razem od zera, to znaczy nie posiadając żadnej wiedzy na temat labiryntu. Aby przejść labirynt, którego używałem do testów taksówki Elona, musiałby wykonać bezbłędnie ponad 15 ruchów pod rząd co jest bardzo mało prawdopodobne, zatem taksówka losowa będzie także się uczyć - nie może wybrać pola zajętego przez stażystę lub wyjechać poza obszar labiryntu, co daje nam większą szansę otrzymania jakkolwiek sensownych wyników. Po jeździe zakończonej sukcesem do listy wykonanych akcji dodawana jest na końcu akcja nr „4”, ułatwiająca potem testowanie taksówek.

5. Wizualizacja

Wykorzystałem bibliotekę pygame do wizualizacji trasy pokonywanej przez samochód. Żeby włączyć odpowiednią trasę, należy ręcznie wkleić uzyskaną listę ruchów po czym uruchomić program.

Rysunek 2. Plansza przed wyruszeniem taksówki z zaznaczonym punktem końcowym

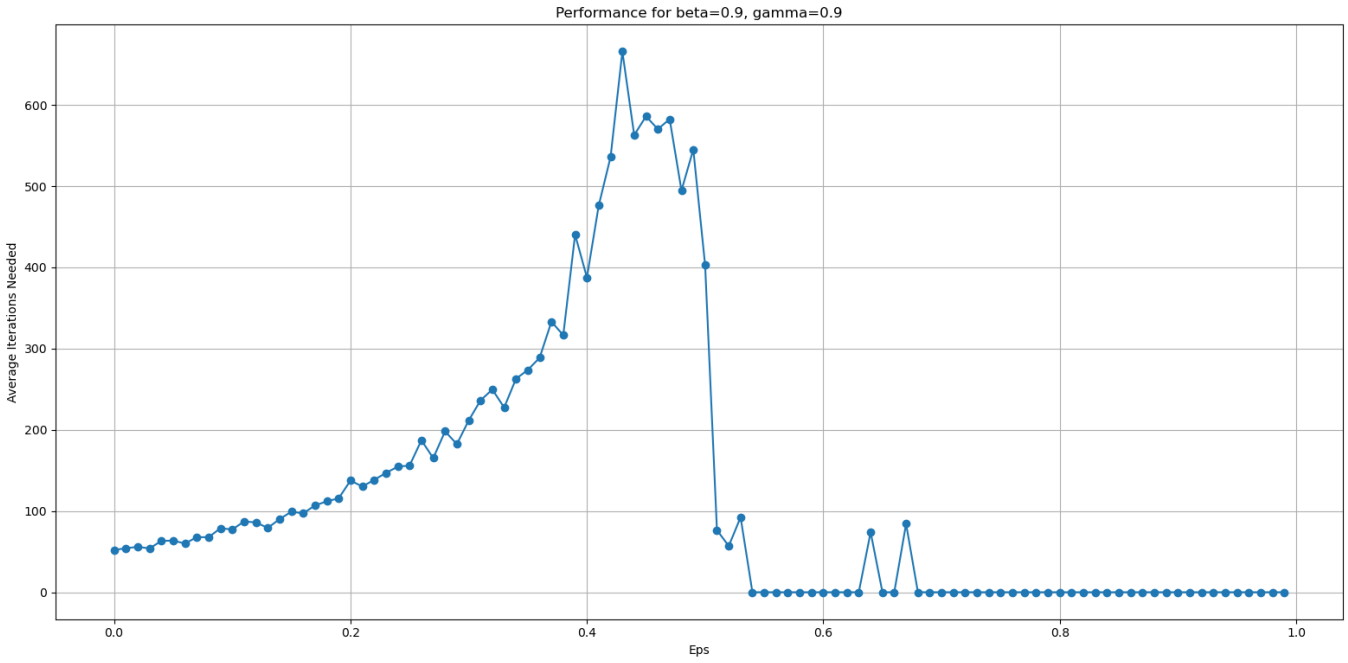
Rysunek 3. Plansza po dojechaniu taksówki do celu.

6. Testowanie algorytmu

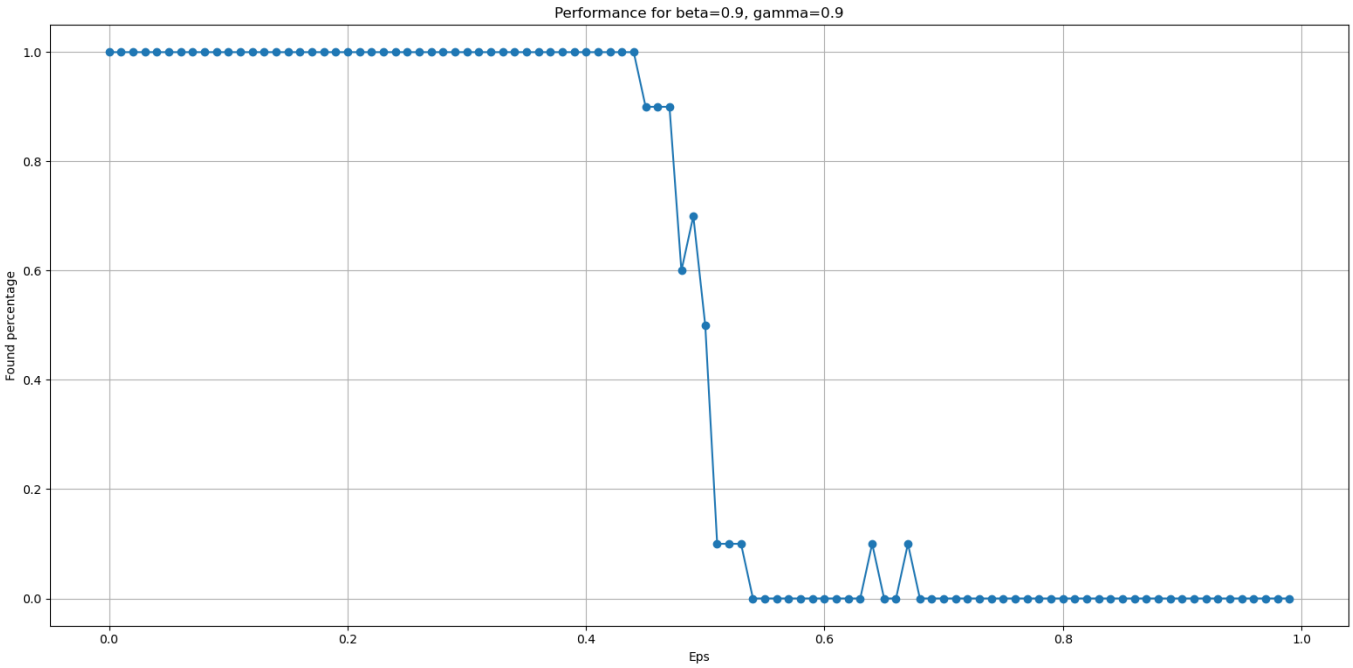
Jakość działania algorytmu q-learningu zależała od parametrów beta, gamma oraz epsilon. Zdecydowałem przeprowadzić serię prób dla różnych wartości tych parametrów. Wynikiem jednej próby była liczba iteracji potrzebnych do ukształtowania wystarczająco poprawnej tablicy, żeby na jej podstawie taksówka przejechała do punktu docelowego najkrótszą trasą oraz czy takie ukształtowanie się udało. Wyniki w największym stopniu zależały od wartości epsilon. Im był większy, tym gorsze były osiągi algorytmu. Oznacza to, że program działa lepiej, korzystając z kształtowanej tabeli, niż wybierając ruchy losowo. Na drugim miejscu pod względem wagi plasuje się współczynnik beta. W jego przypadku osiągi poprawiały się proporcjonalnie do jego wzrostu, ponieważ algorytm mógł uczyć się szybciej. Najmniejszy wspływ na wynik miała wartość dyskonta (gamma). Tutaj jedynie dla

bardzo małych wartości algorytm nie działał poprawnie. Dodawane liczby nie różniły się znacząco, biorąc również ich skalę, co przełożyło się na taki wynik testów.

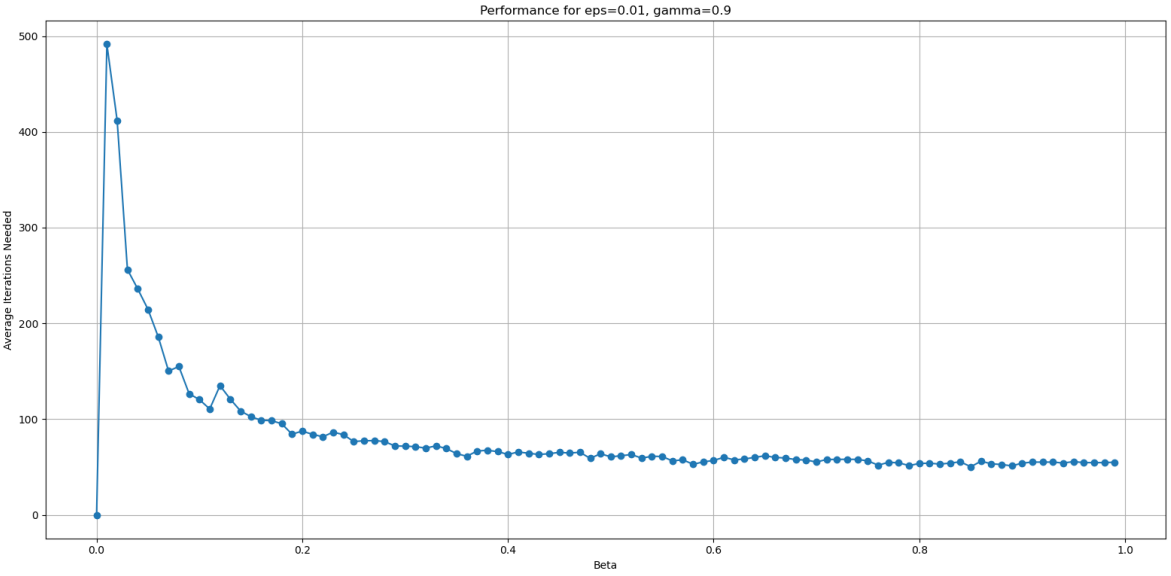
Wyniki testów zaprezentowane są na podstawie 10000 iteracji q-learningu uśredniając 10 przypadków (czyli każda wartość jest powtórzona 100000 razy).



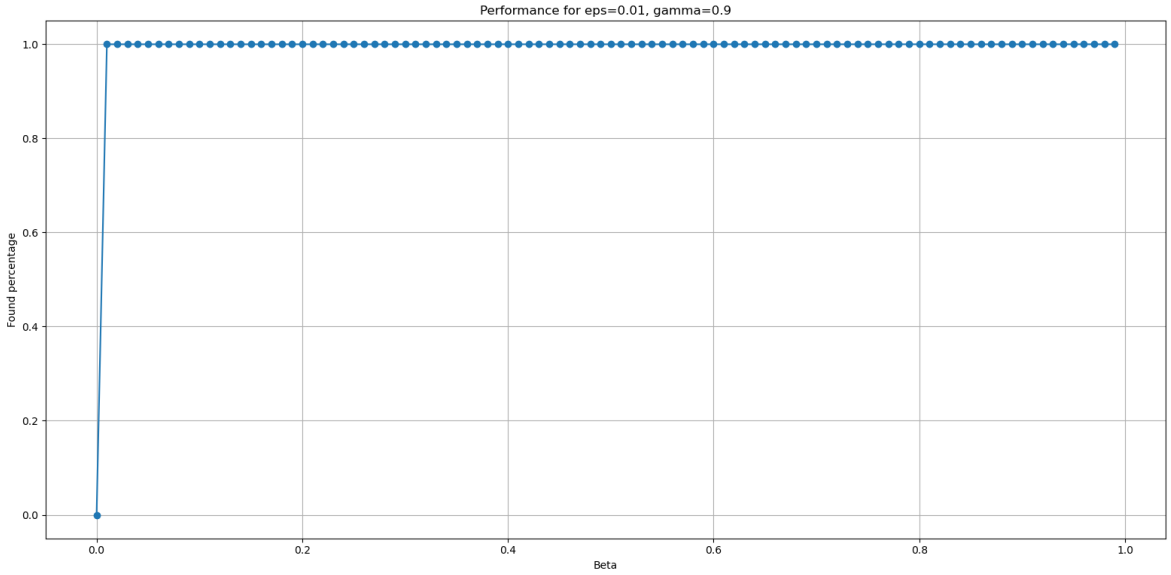
Rysunek 4. Średnia ilość iteracji w zależności od Epsilona.



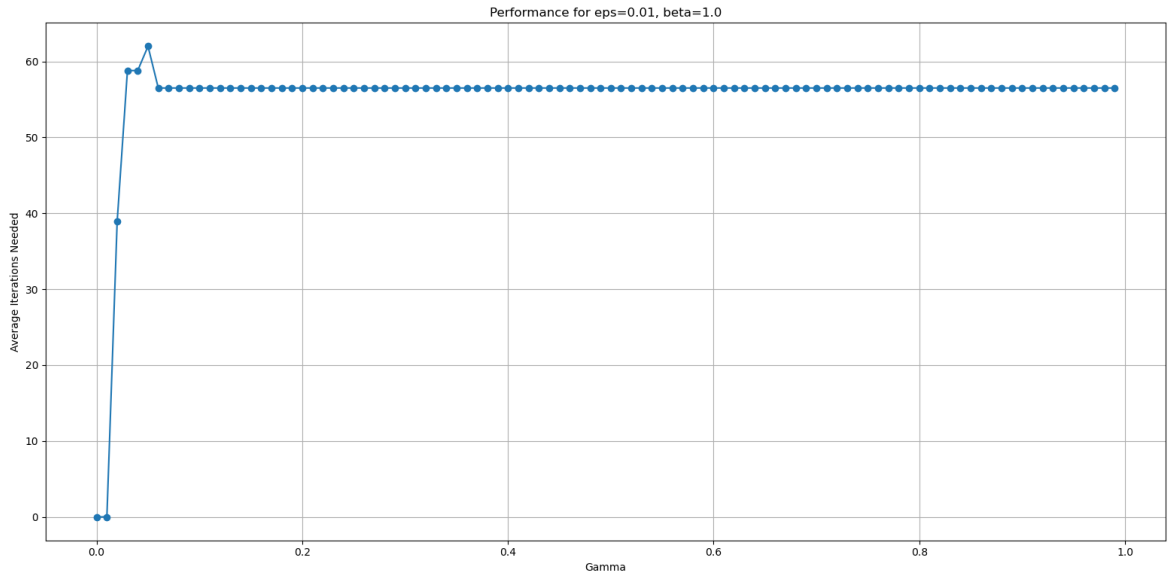
Rysunek 5. Średnia ilość znalezionych optymalnych dróg w zależności od Epsilona.



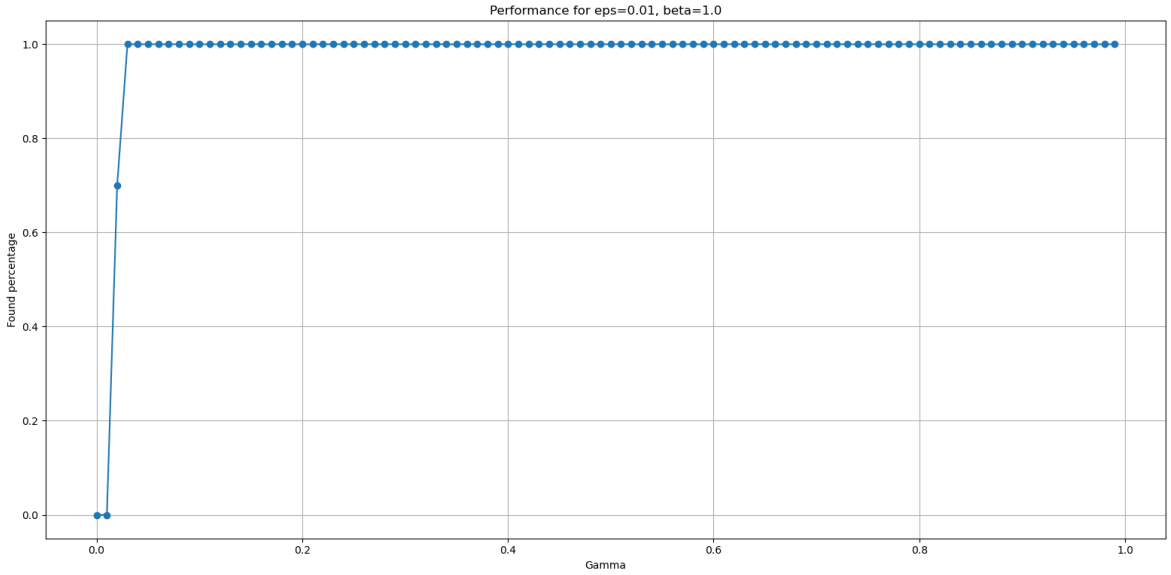
Rysunek 6. Średnia ilość iteracji w zależności od Bety.



Rysunek 7. Średnia ilość znalezionych optymalnych dróg w zależności od Bety.



Rysunek 8. Średnia ilość iteracji w zależności od Gamma.



Rysunek 9. Średnia ilość znalezionych optymalnych dróg w zależności od Gamma.

Taksówka poruszająca się losowo po planszy, ucząc się błędnych ruchów zgodnie z wcześniejszym opisem, potrzebowała średnio 49 prób, żeby znaleźć pierwszą poprawną trasę. Sprawdzając kiedy zostanie wygenerowana trasa optymalna wyszedł mi wynik następujący:
[3, 3, 3, 2, 2, 3, 3, 2, 3, 3, 0, 0, 0]
Length of best found route: 13
Iterations needed to find random route: 74831
Iterations needed to find q-uber route: 49
[3, 3, 3, 2, 2, 3, 2, 3, 3, 3, 0, 0, 0]
Co udowadnia, że q-learning jest znacznie lepszy, przede wszystkim wydajniejszy, niż algorytm losowy. Szczególnie widoczne jest to, że dla 49 iteracji algorytm q-learningowy ma wynik optymalny a losowy – dozwolony.