WSI LAB 6
Uczenie ze wzmocnieniem – Q-Learning

Będkowski Patryk, 310603

I. WSTĘP

Niniejszy raport przedstawia ćwiczenie polegające na implementacji algorytmu *Q-Learning* i jego wykorzystanie w wytworzeniu polityki decyzyjnej dla agenta minimalizującego ścieżkę w losowo generowanym labiryncie.

II. URUCHOMIENIE

Do uruchomienia skryptu potrzebne są następujące narzędzia:

- Python w wersji 3.8.8
- Moduły:
    - tabulate 0.8.9
    - pandas 1.2.4
    - numpy 1.20.1

Aby uruchomić skrypt należy użyć polecenia:

```
python main.py
```

III. GENEROWANY LABIRYNT

Do generowania labiryntu, po którym poruszać się będzie agent, wykorzystany został moduł *gym-maze*[1]. Na potrzeby tego zadania został on rozszerzony o dodatkowe funkcjonalności. Moduł ten pozwala generować losową planszę o ustalonej wielkości oraz określać miejsce początkowe i docelowe agenta.

IV. ZAŁOŻENIA IMPLEMENTACYJNE

W symulacji ustanowiono następujące założenia:

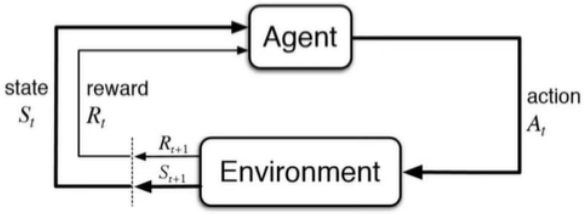
- agent (samochód) nie może wyjechać poza ustalone krawędzi mapy,
- w momencie, gdy agent natrafi na przeszkodę (stażystę), rozbija się i kończy bieg (reset środowiska)
- istnieje co najmniej jedna ścieżka od punktu początkowego do docelowego

V. ROZWIĄZANIE

1. Opis zaimplementowanego algorytmu Q-Learning

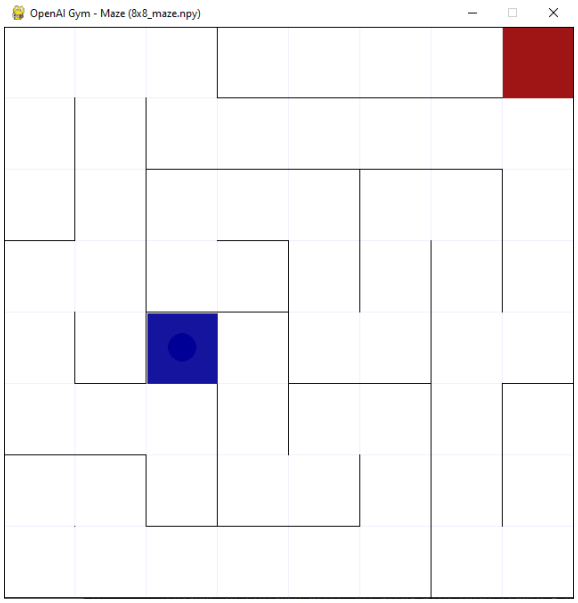
1.1. Opis algorytmu

W symulacji wykorzystano algorytm Q-Learning w celu minimalizacji długości ścieżki osiągnięcia żądanego celu. Aby to zrobić, agent znajdujący się w danym stanie w środowisku, podejmuje akcje prowadzące do nowych stanów zyskując tym samym nagrodę. Jego celem jest natomiast maksymalizacja nagród.



(grafika 1.1) Uproszczony system uczenia

Na akcje składają się dozwolone kierunki poruszania się samochodu, tj. góra, dół, prawo, lewo. Natomiast stany opisane w tabeli *q-table* przedstawiają 64 pozycje (w przypadku rozmiaru planszy 8x8), w których może znaleźć się agent. Poniżej zostało przedstawione przykładowe środowisko. Niebieskie i czerwone pola wskazują kolejno punkt startowy i docelowy samochodu.



(grafika 1.2) Plansza o rozmiarze 8x8

Nagroda, którą agent otrzymuje po wykonaniu jednej akcji została przedstawiona jako ujemna liczba zdefiniowana według wzoru:

$$-\frac{0.1}{ilość\ stanów}$$

W przypadku planszy o rozmiarach 8x8, nagroda będzie wynosić:

$$-0{,}0015625$$

Wartości te dodawane są do skumulowanej sumy nagród agenta, która początkowo posiada wartość 1. W przypadku gdy agent wykona ruch prowadzący do osiągnięcia stanu docelowego, otrzymywana przez niego nagroda wynosi 1.

1.2. Parametry algorytmu

Do parametrów zaimplementowanego algorytmu Q-Learning należą:

- *learning_rate* – wielkość aktualizowanych wartości w tabeli *q-table* w kolejnych iteracjach
- *discount_factor* – determinuje, jak dużą wagę chcemy nadać nagrodom przyszłych stanów
- *exploration_rate* – prawdopodobieństwo wybrania losowej akcji przez algorytm. Wartość tego parametru determinuje strategię działania algorytmu. Początkowo, gdy agent znajduje się w nieznanym środowisku chcemy aby faworyzował potrzebę eksploracji w celu znalezienia nowych ścieżek. Po pewnym czasie chcemy rozwijać potrzebę eksploatacji środowiska, aby znaleźć najlepsze ścieżki. Wprowadzam zatem trzy dodatkowe parametry określające wartość współczynnika eksploracji dla danej iteracji:
  - *exploration_rate_min* - minimalna wartość współczynnika eksploracji
  - *exploration_rate_max* - maksymalna wartość współczynnika eksploracji
  - *exploration_rate_decay* – współczynnik określające wielkość zmiany parametru eksploracji

1.3. Sposób zmiany parametru modelu *exploration_rate*

Aktualizacja tego parametru odbywa się poprzez wzór w każdym epizodzie:

$$new\_exploration\_rate = exploration\_rate\_\min + (exploration\_rate\_\max - exploration\_rate\_min) * e^{exploration\_rate\_decay * episode\_number}$$

Czynnik $e^{exploration\_rate\_decay * episode\_number}$ gwarantuje, że wraz ze wzrostem numerów kolejnych epizodów, maleje nam współczynnik eksploracji.

1.4. Warunki stopu algorytmu

W symulacji zaimplementowane zostały następujące warunki przerwania wykonywania kolejnych iteracji:

- agent dotrze do stanu docelowego
- ilość wykonanych akcji agenta przekroczy maksymalną wartość kroków zdefiniowaną przez wzór:

$$ilość\ stanów * 10$$

- agent wykona niedozwolony ruch (uderzy w ścianę/stażystę bądź w krawędzie labiryntu)
- maksymalna ilość epizodów wynosząca 5000
- program tworzy strukturę danych nazwaną ***last_steps*** będącą kolejką LIFO, która trzyma informację o ***N*** ostatnich krokach w epizodach potrzebnych na zakończenie epizodu. Domyślna ilość przechowywanych kroków to 20. Następnie obliczane są dwie wartości ***k*** i ***l*** według wzoru:

$$k = std(last\_steps) * \min(last\_steps)$$

$$l = 1.2 * \min(last\_steps)$$

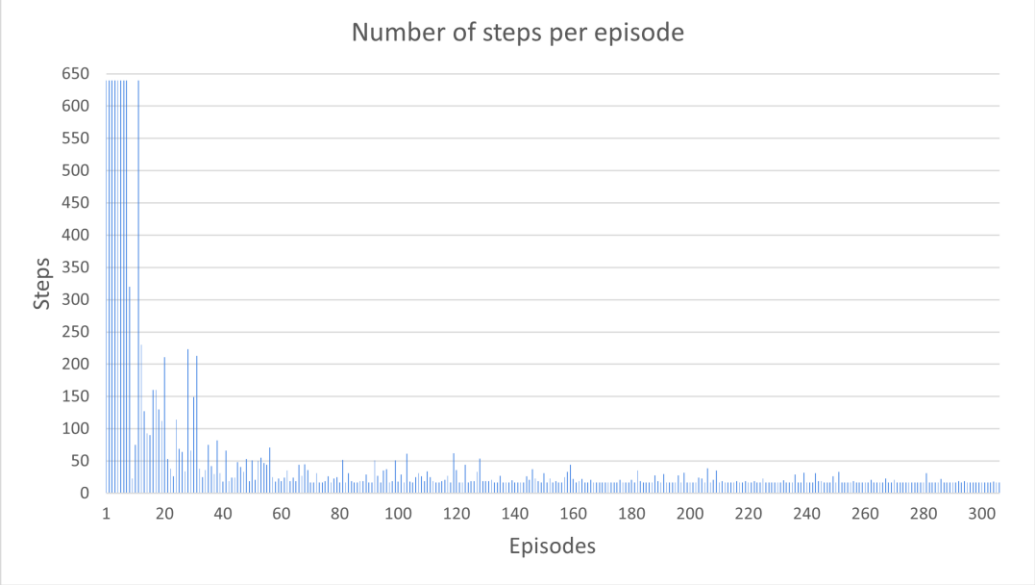
Kolejno następuje porównanie tych dwóch wartości. Przerwanie wykonywania algorytmu następuje gdy $k < l$. Bada się to po to, żeby sprawdzić czy wartości ***N*** ostatnich kroków nie odbiegają od drastycznie siebie. Można zatem wnioskować, iż model znalazł optymalną drogę i nie potrzebne są kolejne iteracje.

2. Test parametrów algorytmu Q-Learning

Testy wykonywano na jednakowej planszy wygenerowanej do pliku o nazwie *8x8_maze.npy* i przedstawionej na grafice *1.2.* Testowania wpływu zmiany parametrów na wyniki będzie odbywać się poprzez zdefiniowanie jednego bazowego modelu[1] i zmianę jego pojedynczych parametrów w kolejnych podpunktach.

learning_rate	0.1
discount_factor	0.99
exploration_rate_min	0.01
exploration_rate_max	0.1
exploration_rate_decay	0.005

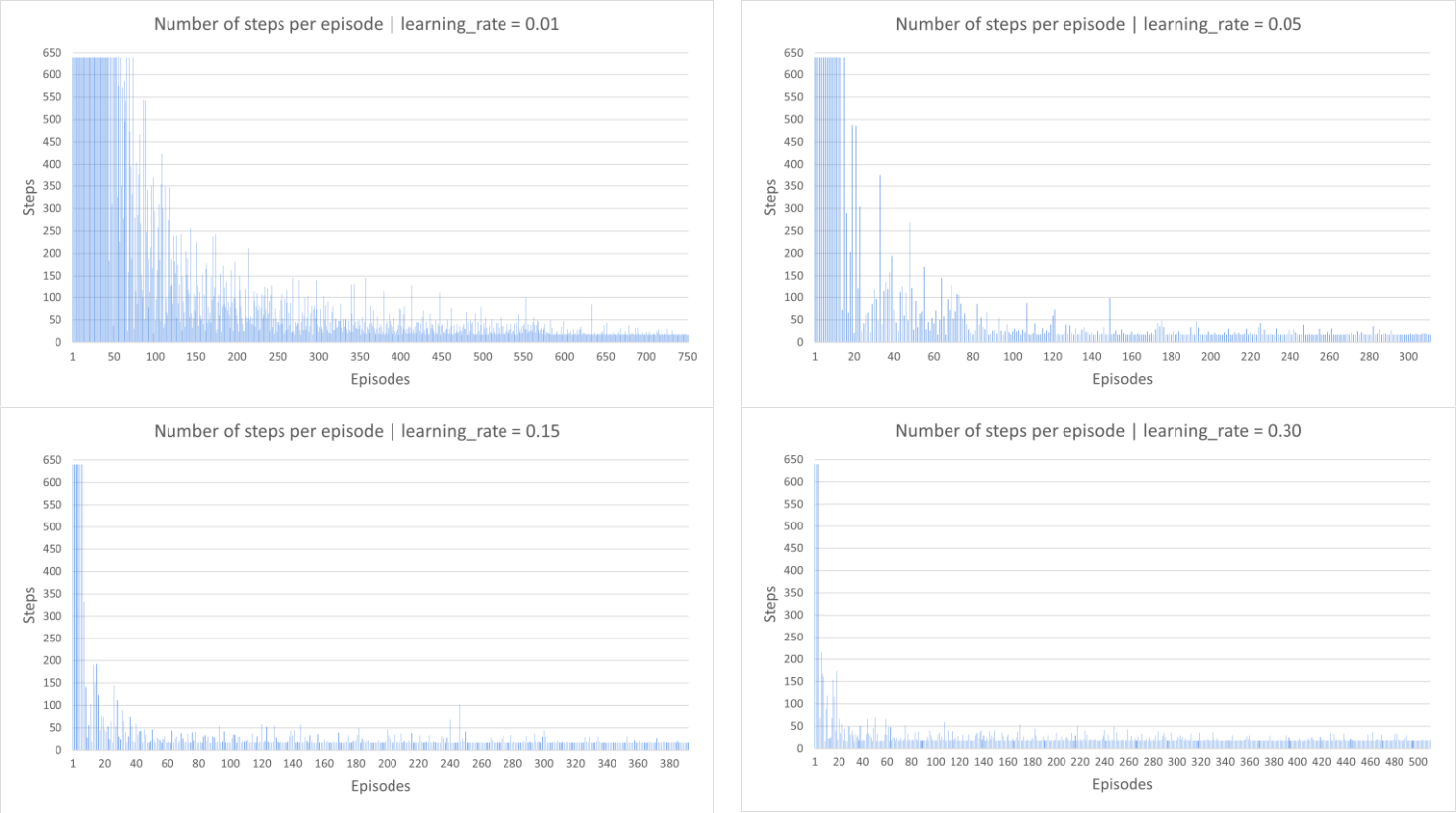
(grafika 2.1) Parametry bazowego modelu



(grafika 2.2) Ilość kroków w kolejnych epizodach dla modelu [1]

2.1. Zmiana parametru *learning_rate*

Sprawdzono jak zmiana parametru *learning_rate* wpływa na ilość kroków w kolejnych epizodach:

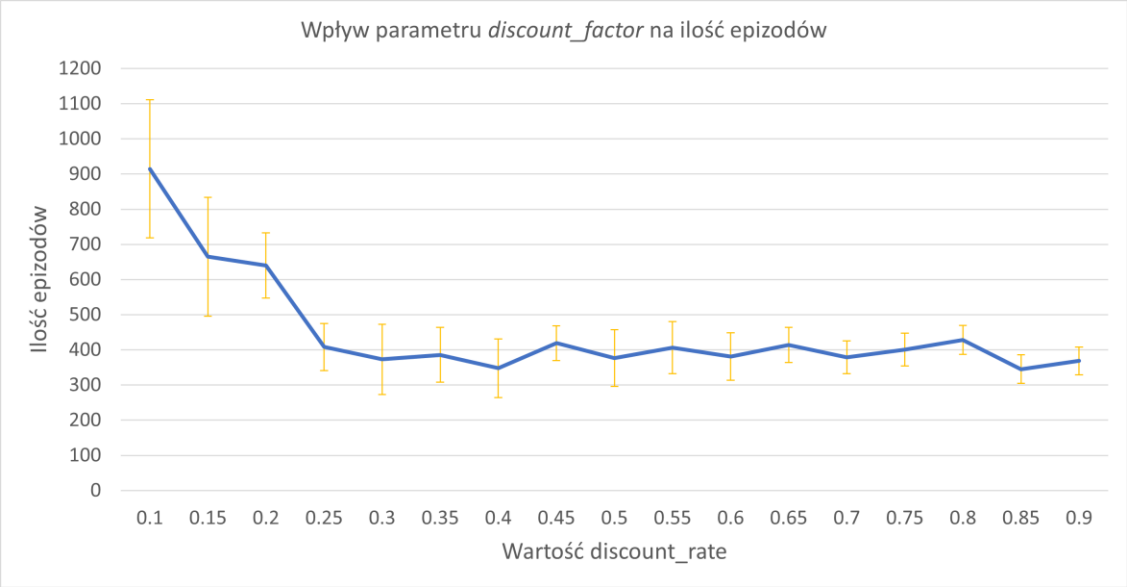


(grafika 2.3) Ilość kroków w kolejnych epizodach dla różnych wartości *learning_rate* modelu [1]

Z powyższej analizy wynika, że zbyt małe wartości parametru *learning_rate* powodują wolniejszą naukę algorytmu, przez co potrzebuje on więcej epizodów przeznaczonych na aktualizację wartości *q-table.* Wraz ze wzrostem wartości tego parametru następuje redukcja ilości potrzebnych epizodów, ale do pewnego momentu. Zbyt duża wartość tego współczynnika powoduje, że mimo tego iż algorytm w miarę szybko redukuję ilość kroków w kolejnych epizodach, to potrzebuje on więcej czasu aby zostało uruchomione kryterium stopu warunkujące ilość kroków 20 ostatnich epizodów.

2.2. Zmiana parametru *discount_factor*

Zbadano jak zmiana parametru *discount_rate* wpływa ma ilość epizodów, po których następuje przerwanie kolejnych iteracji dla modelu bazowego [1] o pozostałych parametrach zdefiniowanych powyżej. Dla każdej wartości tego parametru, test został wykonany 20 razy. Zaznaczono wartości średnie i odchylenie standardowe

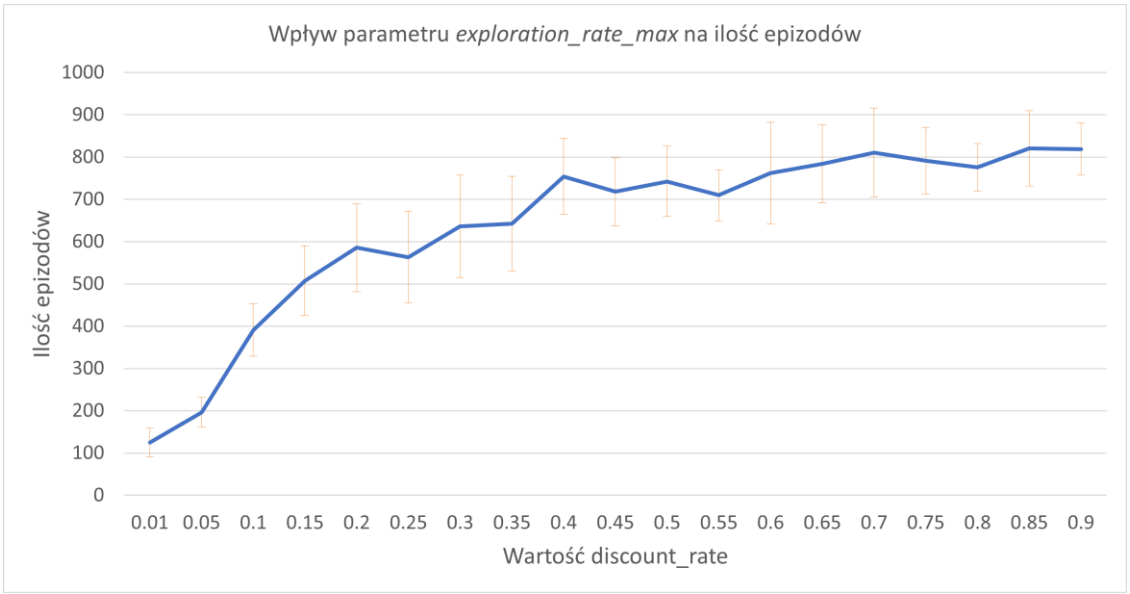


(grafika 2.4) Zmiana parametru *discount_factor* przy parametrach modelu [1]

Z powyższego wykresu wynika, że zbyt niskie wartości tego parametru powodują, że algorytm potrzebuje więcej iteracji aby wypracować optymalne rozwiązanie problemu. Warto zauważyć, że wraz ze wzrostem tego parametru, maleje odchylenie standardowe ilości epizodów w poszczególnych próbach.

2.3. Zmiana parametru *exploration_rate_max*

Dla każdej wartości tego parametru, test został wykonany 20 razy. zaznaczono wartości średnie i odchylenie standardowe.



(grafika 2.5) Zmiana parametru *exploration_rate_max* przy parametrach modelu [1]

Widzimy, że wraz ze wzrostem wartości tego parametru rośnie liczba epizodów, w których działa algorytm. Zatem możemy stwierdzić, że niepożądany jest model, który przez długi czas poznawania środowiska faworyzuje eksplorację dla naszego problemu.

VI. PODSUMOWANIE

Zaimplementowana przeze mnie algorytm Q-Learning został przebadany pod różnymi aspektami wpływającymi na jakoś uczenia się agenta. Widzimy, że na wyniki zbudowanego modelu ma wpływ wiele czynników. W załącznik przesyłam przykładowe korki samochodu dla różnych epizodów w formie graficznej. Obrazują one wzrost jakości agenta w znajdowaniu punktu docelowego oraz potwierdzają, że rozważany agent nauczył się najkrótszej drogi do punktu docelowego.

[1] https://github.com/MattChanTK/gym-maze