WSI – LAB 6 – Q Learning

Łukasz Szarejko

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 (reprezentują dziury o ujemnej punktacji). 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 również nie kontynuuje dalej swojej trasy. Celem agenta jest minimalizacja pokonywanej trasy.

Wymagania

Python 3.9.7
Matplotlib 3.4.3
Numpy 1.21.3
Pygame 2.0.3

Generowanie labiryntu

Labirynt o rozmiarze (N+2)x(N+2) jest generowany losowo, w taki sposób, że krawędzie to stażyści. Następnie sprawdzany algorytmem DFS, czy istnieje ścieżka od miejsca startowego do mety. Jeśli tak, jest on brany do dalszej pracy. Jest on przedstawiony za pomocą dwuwymiarowej tablicy numpy. Punkt z wartością 200 to meta, -1000 to stażyści, 1 to start a pozostałe pola mają wartość -1. Wartości zostały dobrane w taki sposób, aby mogły zostać użyte do nagradzania w dalszej fazie programu.

Akcja

Akcją nazywamy wybranie kierunku, w którym uber podąży.

Polityka

Określa ona prawdopodobieństwo wyboru danej akcji. Jest liczona na zasadzie zachłannej, czyli uber wybiera ruch najlepszy z prawdopodobieństwem epsilon.

Stany

Poszczególnymi stanami, są plansze z naniesionymi botami przedstawione przez dwuwymiarowe tabice numpy.

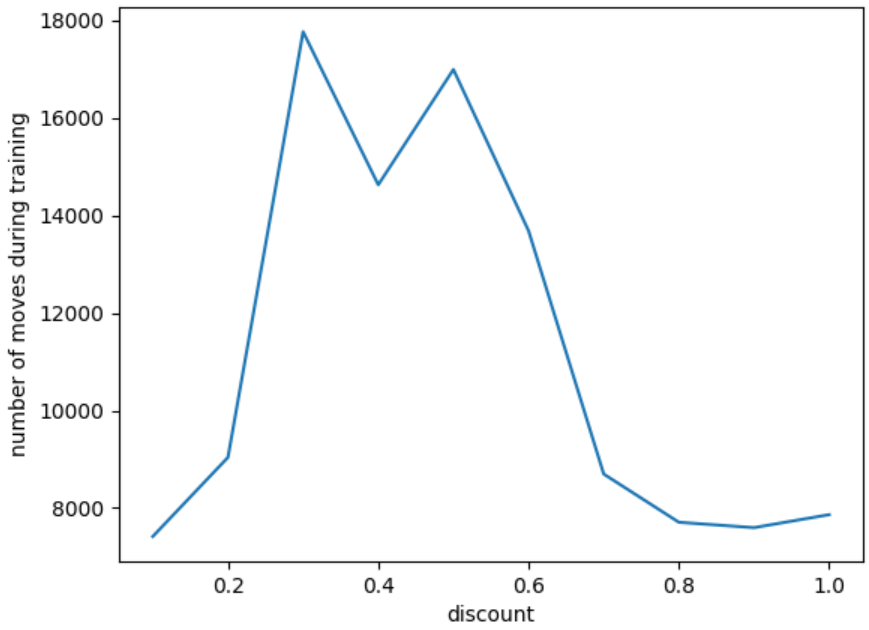
Nagroda

Tak jak zostało to wspomniane powyżej, nagrody są już umieszczone na planszy. Uber wchodząc na dane pole poznaje nagrodę jaką dostaje. System działa w taki sposób, aby karać bota za długość czasu jaki będzie pozostawał w labiryncie, więc będzie on usilnie dążyć do mety jak najkrótszą drogą.

Dobieranie parametrów

Wszystkie parametry dobierane są na analizie 25 prób dla każdej testowanej wartości. Ruchy są liczone w trakcie trwania nauki ubera. Nauka jest wykonywana 100 razy zaczynając od losowych, ale właściwych punktów na mapie. Każdorazowo uber ma maksymalnie 1000 ruchów podczas jednej iteracji nauki. Iteracja kończy się, gdy uber trafi do celu, czyli maksymalna ilość kroków podczas całej nauki wynosi 100 tysięcy (zakładając, że uber nigdy nie trafi do celu). Dobieranie parametrów jest przeprowadzane na na planszy 8x8 z 30% szansą na stażystę. W celu lepszej konfiguracji parametrów, ustaliłem seed'a, aby przy 25 próbach generowana była zawsze taka sama plansza.
Początkowe wartości learning rate to 1 i epsilonu to 0.7.

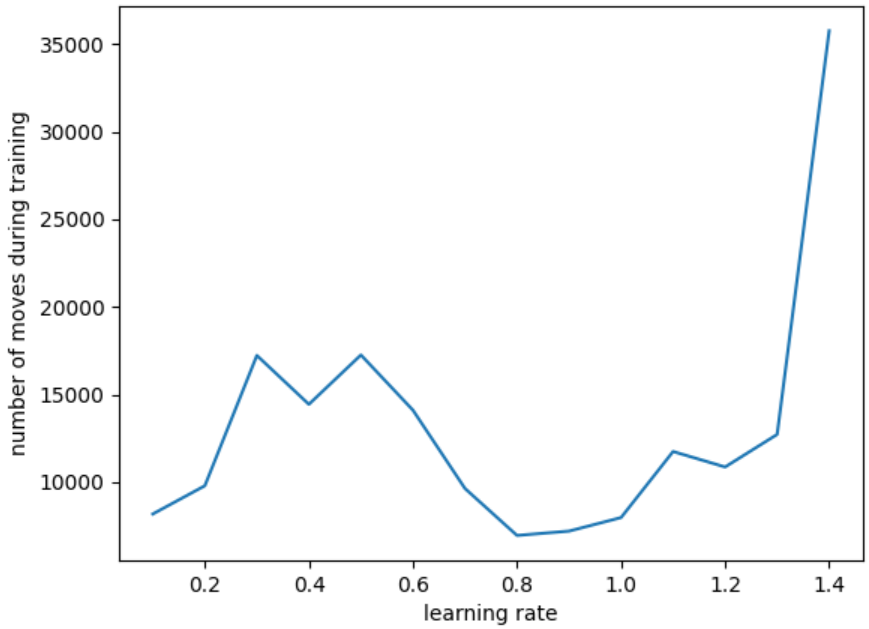
Discount



Rys. 1. Zależności szybkości nauki modelu od wartości parametru discount

Jak widzimy na powyższym wykresie, model uczy się najszybciej dla discount równego 0.9, dlatego do dalszych badań zostanie wzięta właśnie taka wielkość.

Learning rate

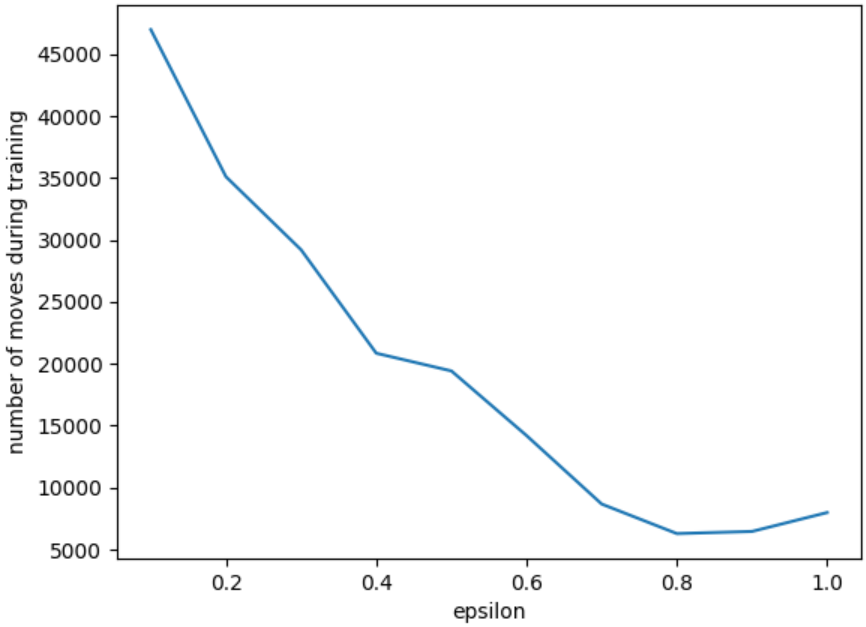


Rys.2. Zależność szybkości nauki od wartość parametru learning rate

Jak możemy zauważyć na powyższym wykresie, model osiąga bardzo dobrą sprawność dla wartości od 0.8 do 1.0, dlatego do dalszych badań wezmę wartość 0.8.

Epsilon

Określa on, jaki procent ruchów ubera ma zostać wykonywane na podstawie wiedzy. Pozostała część jest szansą na ruch losowy, w celach eksploracji środowiska. Oznacza to, że dla epsilonu równego 1, szansa na ruch losowy nie istnieje.

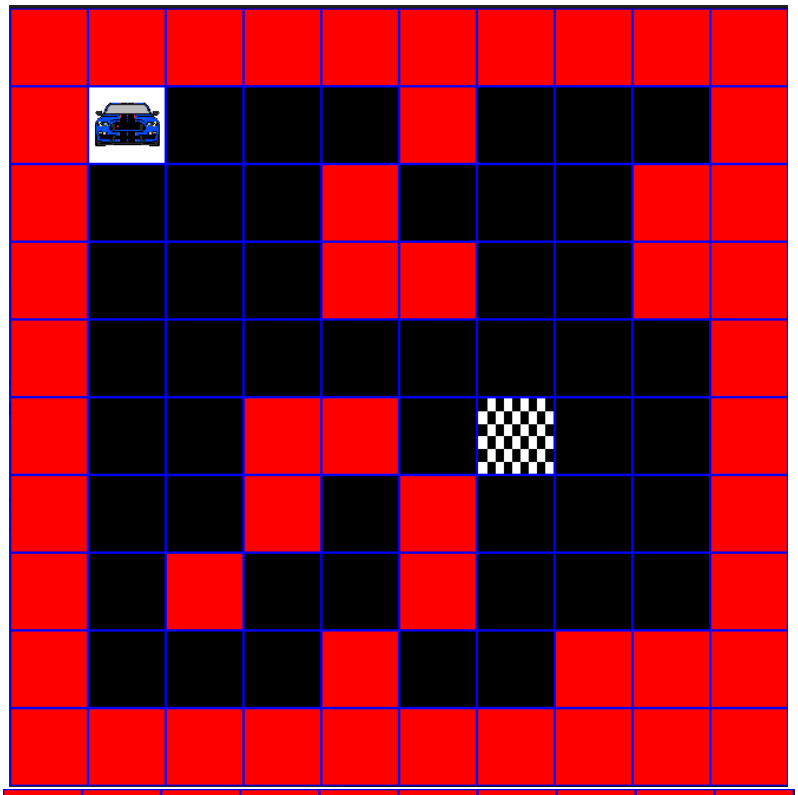


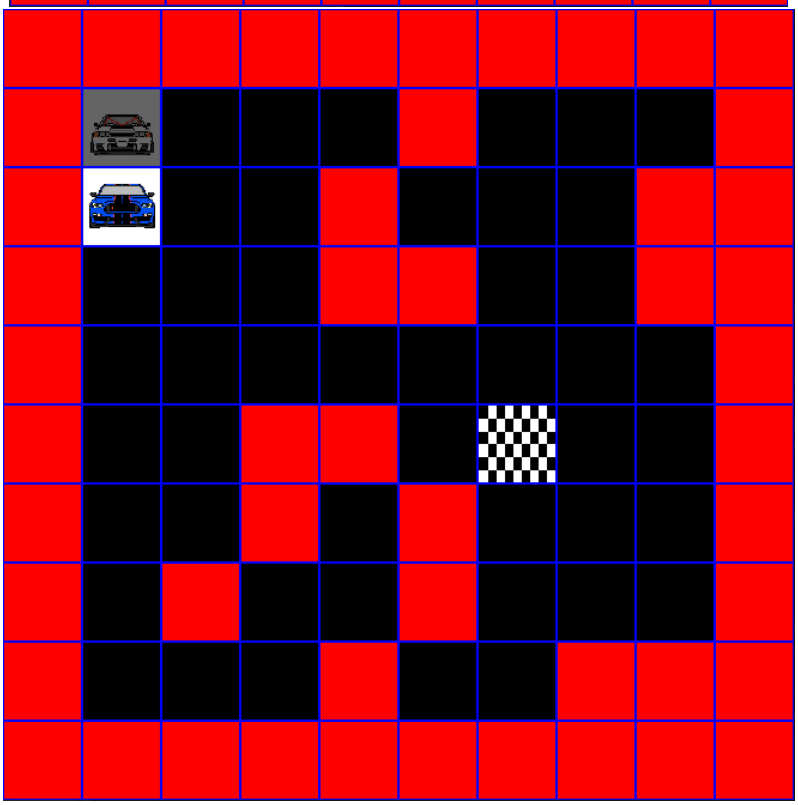
Rys. 3. Zależność szybkości nauki od wartości parametru epsilon

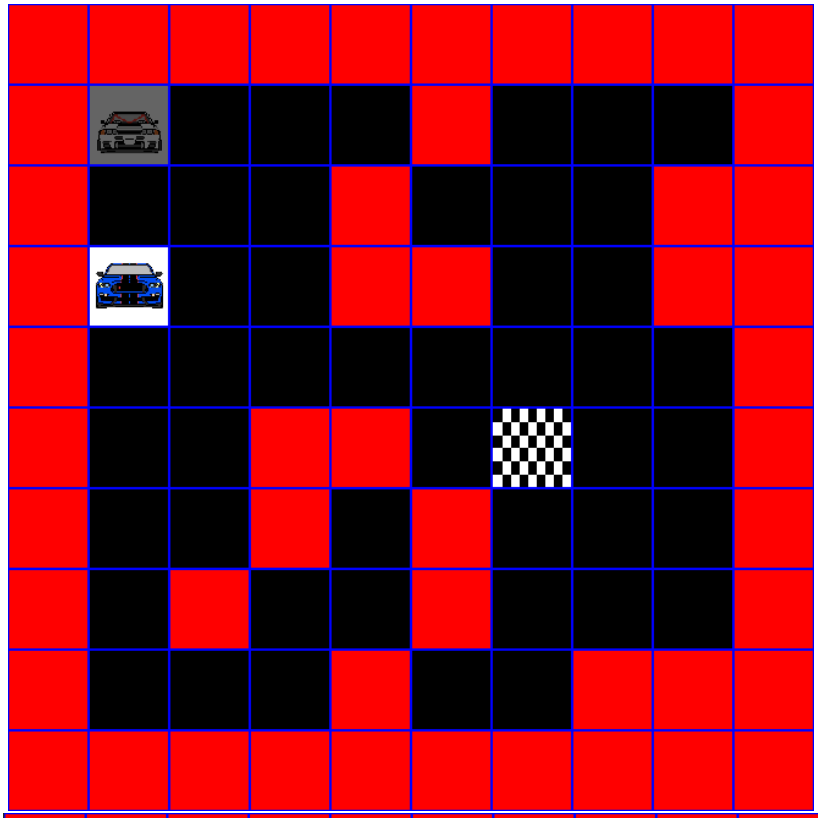
Jak widzimy na powyższym wykresie, najlpiej spisuje się model z epsilonem równym 0.8. Wywnioskować możemy to, że model potrzebuje około 20% ruchów losowych w celach eksploracyjnych.

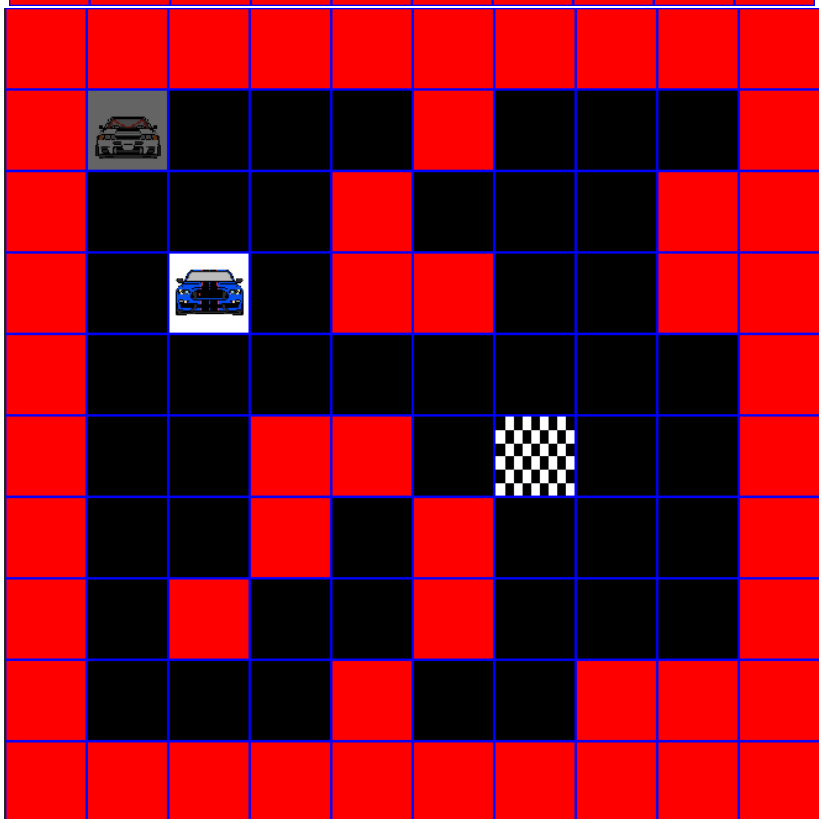
Przykładowe działanie wraz z najkrótszą trasą

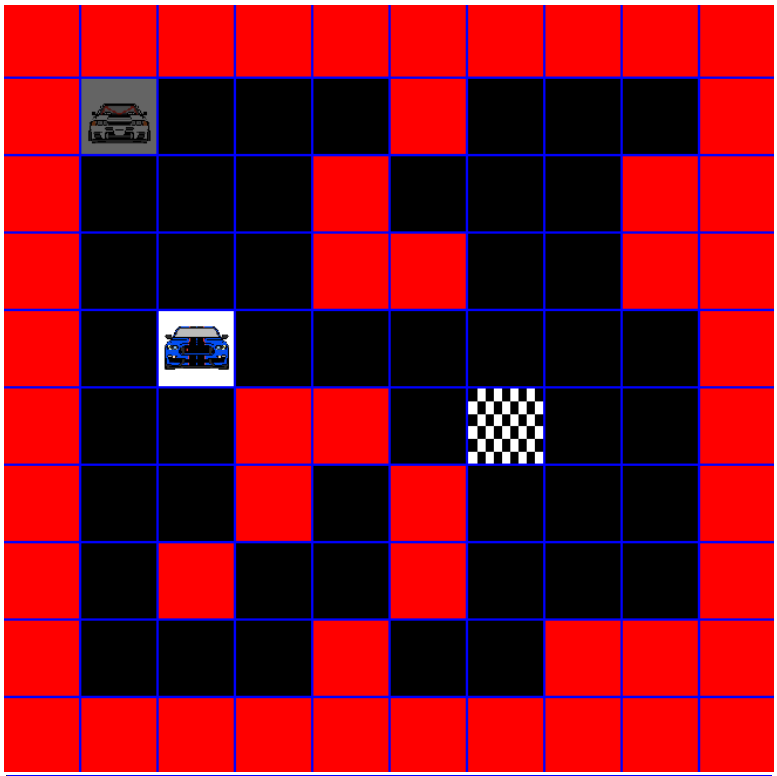
Dla dobranych parametrów: discount=0.9, learning rate=0.8, epsilon=0.8, prześledźmy ruchy nauczonego modelu, aby sprawdzić, czy rzeczywiście podąża on do celu najkrótszą możliwą ścieżką. Na zrzutach, uber jest przedstawiony przy użyciu obrazka niebieskiego samochodu (Mustang), a gracz randomowy białego (Nissana). Testy zostały przeprowadzone na dokładnie tej samej mapie.

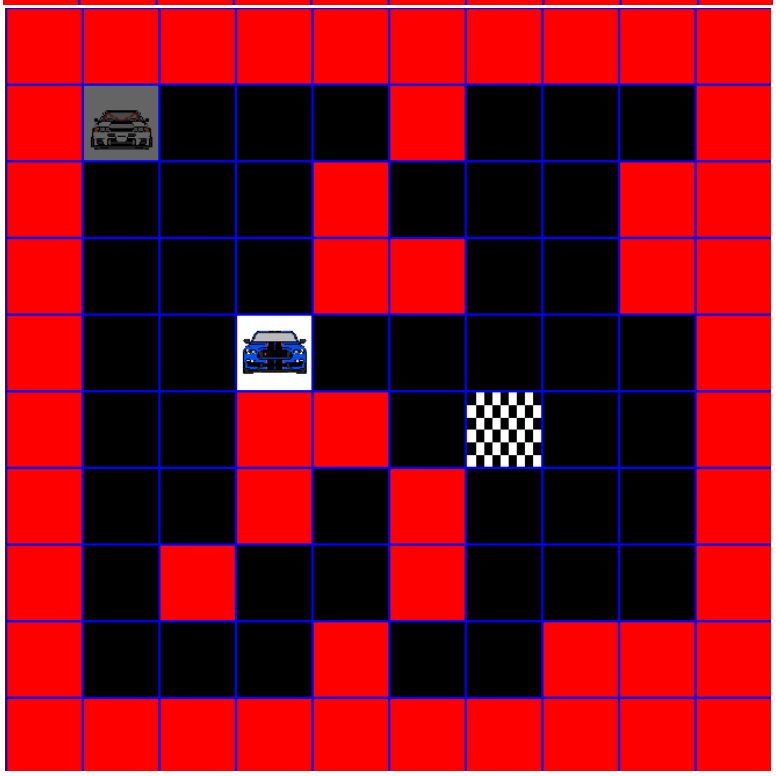


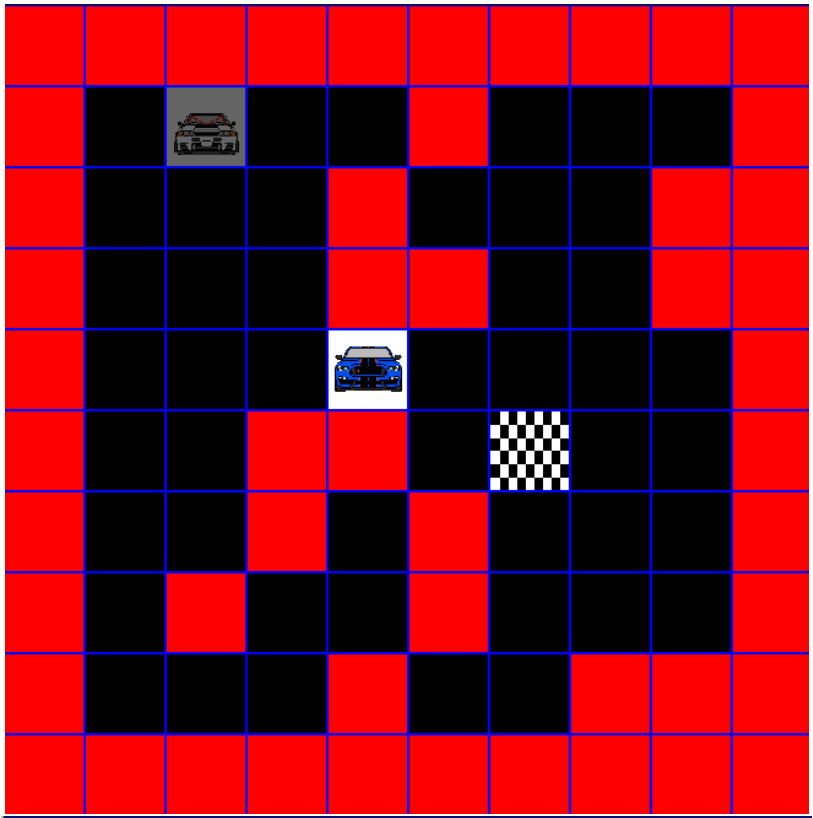


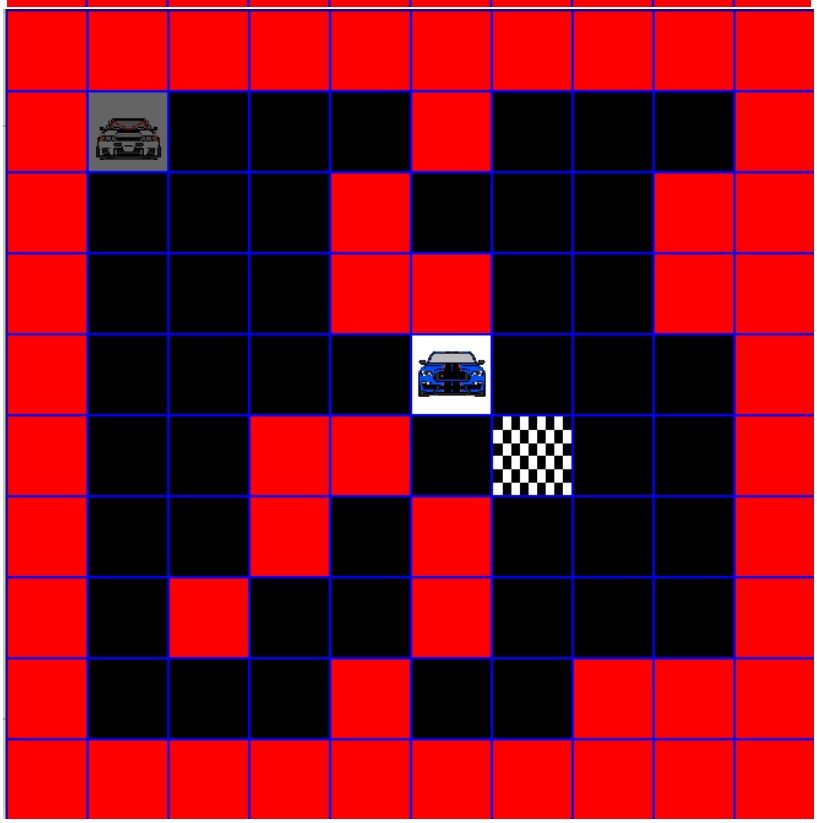


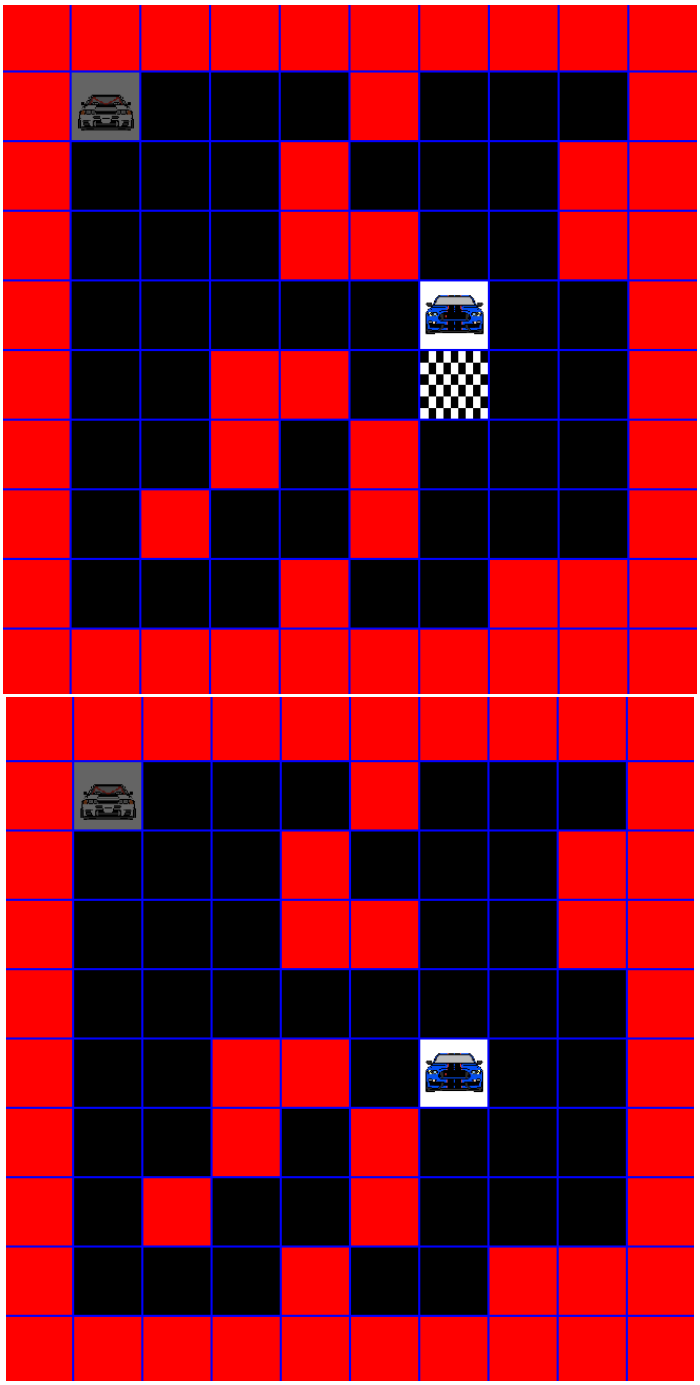












Jak widzimy na powyższych zrzutach, agent qlearningowy dotarł do celu najkrótszą drogą, natomiast agent randomowy w tym samym czasie nawet nie wyszedł z punktu startowego. Zrzuty, na których agent randomowy jest w tym samym miejscu oznaczają, że wbił się on w stażystę i został cofnięty do punktu początkowego.
Zdarzyć się może, że agent qlearningowy podczas drogi do celu też popełni błąd, gdyż może się tak zdarzyć, że nie miał odpowiedniej ilości iteracji do nauki, lub punkty, z których startował wylosowały się niedaleko siebie, więc nie miał możliwości nauki całej planszy.