

**Uniwersytet Jagielloński w Krakowie**  
Wydział Fizyki, Astronomii i Informatyki Stosowanej

**Piotr Kucharski**

Nr albumu: 1124564

# **Uczenie agentów sterowania pojazdami kosmicznymi**

Praca licencjacka  
na kierunku Informatyka Stosowana

Praca wykonana pod kierunkiem  
prof. dr. hab. Piotra Białasa  
Zakład Technologii Gier

Kraków 2020

## **Oświadczenie autora pracy**

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

.....

Kraków, dnia

.....

Podpis autora pracy

## **Oświadczenie kierującego pracy**

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

.....

Kraków, dnia

.....

Podpis kierującego pracę

# Spis treści

<b>1 Abstrakt</b>	<b>3</b>
<b>2 Wstęp</b>	<b>3</b>
<b>3 Środowisko</b>	<b>3</b>
3.1 Akcje . . . . .	4
3.2 Stan . . . . .	4
3.3 Graficzna reprezentacja środowiska . . . . .	4
3.4 Funkcja aktualizacji środowiska . . . . .	5
3.5 Scenariusze i modelowanie funkcji nagrody . . . . .	5
<b>4 Definicja uczenia ze wzmacnieniem</b>	<b>6</b>
<b>5 Głębokie uczenie funkcji wartości akcji <math>Q</math></b>	<b>7</b>
5.1 Problemy głębokiego uczenia ze wzmacnieniem . . . . .	8
5.2 Sztuczna sieć neuronowa . . . . .	9
5.3 Algorytm . . . . .	10
<b>6 System do przeprowadzania eksperymentów</b>	<b>10</b>
6.1 Tworzenie eksperymentów . . . . .	10
6.2 Uruchamianie obliczeń . . . . .	11
6.3 Synchronizacja wyników . . . . .	11
<b>7 Przeprowadzone eksperymenty i wyniki</b>	<b>11</b>
7.1 Pierwszy scenariusz . . . . .	12
7.2 Drugi scenariusz . . . . .	13
7.3 Trzeci scenariusz . . . . .	14
<b>8 Wyniki testów</b>	<b>15</b>
8.1 Pierwszy scenariusz . . . . .	16
8.2 Drugi scenariusz . . . . .	17
8.3 Trzeci scenariusz . . . . .	18
<b>9 Analiza trzech agentów z pierwszego środowiska</b>	<b>19</b>

# 1 Abstrakt

W niniejszej pracy prezentuję wyniki eksperymentów mających na celu zbadanie wydajności i niezawodności prostego algorytmu uczenia ze wzmacnieniem DQN[6]. Zadaniem algorytmu jest uczenie sztucznej sieci neuronowej sterowania pojazdami w symulowanej przestrzeni kosmicznej. Z przeprowadzonych eksperymentów wynika, że algorytm jest skuteczny w znajdywaniu optymalnych strategii w prostych środowiskach. W skomplikowanych środowiskach algorytm rzadziej znajduje dobre strategie, a parametry muszą być precyzyjnie dobierane.

## 2 Wstęp

Uczenie nadzorowane, najczęściej wykorzystywany rodzaj uczenia maszynowego, polega na uczeniu funkcji na podstawie zbioru danych. Zbiór ten zawiera dane wejściowe oraz oczekiwany wynik. Po predykcji wyniku na podstawie danych wejściowych znana jest dokładna wartość popełnionego błędu. Uczenie ze wzmacnieniem polega na uczeniu funkcji na podstawie interakcji ze środowiskiem. Dane wejściowe są obserwowane podczas symulacji, a nagroda(błąd) poznawana jest dopiero po zakończeniu symulacji. Sposób interakcji uczonej funkcji ze środowiskiem wpływa na rozkład obserwowanych danych wejściowych. Mimo takich ograniczeń Algorytm Atari DQN[6] był w stanie nauczyć sztuczną sieć neuronową strategii podejmowania dobrych decyzji w siedmiu grach na platformie Atari. Nauka kolejnych gier Atari nie wymagała modyfikacji w algorytmie. Moim celem jest sprawdzenie wydajności algorytmu w trzech środowiskach symulowanej przestrzeni kosmicznej.

## 3 Środowisko

Agent podlegający uczeniu steruje pojazdami poruszającymi się po dwuwymiarowej, ciągłej przestrzeni. Pojazdy przemieszczają się zgodnie z zasadami dynamiki Newtona, bez oporów wpływających na ich aktualną prędkość. Środowisko jest kontrolowane z poziomu skryptu w Pythonie. Interfejs komunikacji środowiska z agentem składa się trzech funkcji: funkcji aktualizacji stanu środowiska, funkcja pozwalająca przywrócić środowisko do stanu początkowego oraz funkcji zwracającej informację o tym, czy środowisko jest nadal aktywne. Funkcja aktualizacji środowiska przyjmuje na wejściu akcje i zwraca następny stan, nagrodę oraz informacje o tym, czy dany pojazd został zresetowany. Funkcja przywracająca środowisko do stanu początkowego zwraca tylko stan.

### 3.1 Akcje

Przy każdej aktualizacji środowiska zadaniem agenta jest kontrolowanie ruchu pojazdu wybierając jedną z możliwych akcji. Wybrana przez agenta akcja odpowiada z jaką mocą będą działać dwa silniki. Pierwszy silnik nadaje przyspieszenie w kierunku przodu pojazdu, zaś drugi przyspieszenie obrotowe. Pierwszy silnik posiada dwie możliwe akcje: wyłączony albo włączony [0, 1]. Drugi silnik posiada trzy możliwe akcje: lewo, wyłączony, prawo [-1, 0, 1]. Przestrzeń akcji powstaje z iloczynu kartezjańskiego zbiorów możliwych akcji obu silników. Z przestrzeni akcji usuwana jest akcja neutralna kiedy oba silniki są wyłączone, gdyż w początkowym etapie uczenia ta akcja była faworyzowana, co prowadziło do stagnacji procesu uczenia.

### 3.2 Stan

Na podstawie stanu otrzymanego ze środowiska agent podejmuje decyzję którą akcję wykonać w danym kroku. Stan obserwowany przez agenta w składa się z trzech liczb rzeczywistych odpowiadających kolejno szybkości pojazdu, kątowi pomiędzy kierunkiem statku a wektorem prędkości oraz prędkością kątowej.

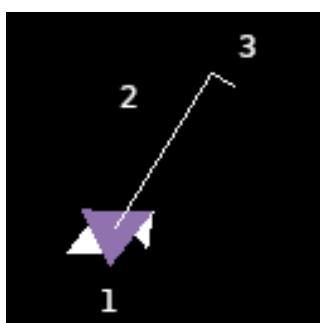
- 1)  $|Velocity|$
  - 2)  $\angle ShipDirection - \angle Velocity$
  - 3)  $AngularVelocity$
- (1)

Dodatkowo, w scenariuszach w których celem jest dotarcie pojazdem do wyznaczonego punktu kontrolnego, stan zawiera dwie dodatkowe liczby rzeczywiste odpowiadające odległości pojazdu od wyznaczonego punktu docelowego oraz kątowi pomiędzy kierunkiem statku a wektorem odległości do tego punktu.

- 4)  $|CheckpointPosition - ShipPosition|$
  - 5)  $\angle ShipDirection - \angle (CheckpointPosition - ShipPosition)$
- (2)

### 3.3 Graficzna reprezentacja środowiska

Rysunek 1: Pojazd.



Pojazd (1) jest reprezentowany przez trzy trójkąty. Kolorowy trójkąt reprezentuje kadłub pojazdu, a jego najostrzejszy wierzchołek wskazuje przód. Wielkość dwóch białych trójkątów odpowiada mocy silników. Trójkąt połączony z krótszą krawędzią pojazdu reprezentuje główny silnik, a trójkąt połączony z najostrzejszym wierzchołkiem pojazdu reprezentuje silnik rotujący. Odcinki (2) prędkości oraz (3) prędkość obrotowa są graficzną reprezentacją stanu. Do renderowania środowiska używam biblioteki SFML[4].

### 3.4 Funkcja aktualizacji środowiska

Po wybraniu przez agenta akcji następuje aktualizacja stanu środowiska. Podczas kroku aktualizacji środowiska w pierwszej kolejności obliczane jest przyspieszenie działające na pojazd, które zależy od wybranej przez agenta mocy głównego silnika.

$$\begin{cases} Acceleration_x = firstAction * \cos(ShipDirection) * mainEnginePower / mass \\ Acceleration_y = firstAction * \sin(ShipDirection) * mainEnginePower / mass \end{cases} \quad (3)$$

$$Velocity = Velocity + Acceleration * timeStep$$

$$ShipPosition = ShipPosition + Velocity * timeStep$$

Po aktualizacji położenia pojazdu obliczana jest rotacja pojazdu.

$$AngularAcceleration = secondAction * rotationEnginePower / (0.5 * mass * size)$$

$$AngularVelocity = AngularVelocity + AngularAcceleration * timeStep \quad (4)$$

$$ShipDirection = ShipDirection + AngularVelocity * timeStep$$

Środowisko aktualizowane jest ze stałym krokiem czasowym wynoszącym dziesiątą część sekundy. Parametry pojazdu *mainEnginePower*, *mass*, *rotationEnginePower* oraz *size* są stałe. Po zaktualizowaniu położenia oraz rotacji pojazdu obliczana jest nagroda oraz sprawdzane są warunki kończące symulację pojazdu. Funkcja zwraca krotkę z adresami pamięci, pod którymi znajdują się wektory zawierające następny stan, nagrodę oraz informację czy pojazd zakończył symulację w tym kroku. Korzystam z zaimplementowanego w bibliotece BOOST[1] sposobu komunikacji pomiędzy Pythonem a C++.

### 3.5 Scenariusze i modelowanie funkcji nagrody

Na potrzeby eksperymentów zaimplementowałem trzy scenariusze różniące się pomiędzy sobą poziomem trudności oraz sposobem obliczania nagrody.

Celem pierwszego, najprostszego scenariusza jest nauczenie agenta zmniejszania szybkości rozpoczętego pojazdu. Pojazd rozpoczyna symulację posiadając losową rotację, losową prędkość obrotową z zakresu  $[-2, 2] \text{ rad/s}$  oraz wektor prędkości, którego elementy losowane są z zakresu  $[5, 10] \text{ m/s}$ . Po każdej aktualizacji środowiska agent otrzymuje nagrodę odpowiadającą odwrotnej zmianie szybkości w danym kroku  $r_t = |Velocity_{t-1}| - |Velocity_t|$ . Pojazd ulega zniszczeniu i jest resetowany gdy jego szybkość przekroczy  $20 \text{ m/s}$  lub gry szybkość obrotowa przekroczy  $2 \text{ rad/s}$ . Agent otrzymuje karę za wybranie akcji prowadzącej do zniszczenia pojazdu. Gdy szybkość pojazdu zmala się poniżej  $2 \text{ m/s}$  agent otrzymuje nagrodę za zaliczenie scenariusza i ocalenie pojazdu. Otrzymywana przez agenta nagroda/kara po osiągnięciu stanu terminalnego we wszystkich scenariuszach wynosi  $+1/-1$ .

Celem drugiego i trzeciego scenariusza jest nauczenie agenta dolatywania pojazdem do wyznaczonego losowo punktu kontrolnego. Podobnie jak w pierwszym scenariuszu pojazd na początku posiada losową rotację, prędkość obrotową  $[-1, 1] \text{ rad/s}$ , oraz elementy wektora prędkość z zakresu  $[0, 10] \text{ m/s}$ . Pozycja pojazdu ustawiana jest na środek układu współrzędnych, a współrzędne punktu kontrolnego losowane są z zakresu  $[-700, 700] \text{ m}$ . Gdy pojazd zbliży się do punktu kontrolnego na mniej niż  $25 \text{ m}$  pojazd oraz punkt kontrolny są losowane na nowo, a agent otrzymuje nagrodę. Gdy pojazd oddali się od punktu kontrolnego na więcej niż  $1400 \text{ m}$  lub jego szybkość obrotowa przekroczy  $2 \text{ rad/s}$  agent otrzymuje karę, pojazd ulega zniszczeniu a punkt kontrolny pozostaje w tym samym miejscu. W drugim scenariuszu nagroda otrzymywana przez agenta po każdej aktualizacji odpowiada odwrotnej zmianie odległości pojazdu od punktu kontrolnego  $r_t = d_{t-1} - d_t$ , gdzie  $d_t$  odpowiada odległości euklidesowej pojazdu od punktu kontrolnego. W trzecim scenariuszu nagroda odpowiada zmianie szybkości w kierunku punktu kontrolnego  $r_t = d_{t-2} - 2 * d_{t-1} + d_t$ .

## 4 Definicja uczenia ze wzmacnieniem

Uczenie ze wzmacnieniem polega na optymalizacji strategii wybierania akcji na podstawie doświadczeń generowanych podczas interakcji ze środowiskiem. Proces decyzyjny Markowa jest matematycznym sformułowaniem procesu interakcji agenta ze środowiskiem, którego celem jest osiągnięcie najwyższej możliwej nagrody. Składa się z krotki  $(S, A, P, R)$ , gdzie  $S$  to zbiór stanów środowiska w jakich może się znaleźć symulowany pojazd,  $A$  to zbiór akcji,  $P(s'|s, a)$  to rozkład prawdopodobieństwa przejścia pomiędzy stanami w zależności od wybranej akcji,  $R(s, a)$  to funkcja nagrody otrzymywanej przez agenta natychmiast po wykonaniu akcji w zadanym stanie. Proces interakcji ze środowiskiem tworzy serię doświadczeń  $s_1, a_1, r_1, s_2, a_2, r_2 \dots s_T, a_T, r_T$ , gdzie  $T$  oznacza numer kroku terminalnego, w którym kończy się epizod i symulacja pojazdu jest resetowana. Parametr zniżki  $0 < \gamma < 1$  zmniejszający nagrody z przyszłych kroków zmusza agenta do podejmowania dobrych decyzji wcześniej. Gdyby zniżki przyszłych nagród nie było agent mógłby odwlekać wybranie dobrej akcji w nieskończoność. Obniżona nagroda definiowana jest następującym równaniem.

$$R_t = \sum_{i=t}^T \gamma^{(i-t)} r_i \quad (5)$$

Maksymalna obniżona nagroda możliwa do uzyskania z danego stanu zdefiniowana jest jako optymalna funkcja wartości.

$$V^*(s) = \max_a [R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s')] \quad (6)$$

Maksymalna oczekiwana nagroda do uzyskania z danego stanu po podjęciu określonej akcji to optymalna funkcja wartości akcji  $Q^*$ .

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a') \quad (7)$$

$$V^*(s) = \max_a Q^*(s, a)$$

Strategię  $\pi$  zdefiniowana jest jako funkcja rozkładu prawdopodobieństwa akcji na zadanym stanie. Jeśli, tak jak w przypadku tej pracy, strategia jest deterministyczna, to zwraca ona wartość 1 dla wybranej akcji i 0 dla wszystkich pozostałych.

$$\pi(a|s) = \mathbb{P}[a_t = a|s_t = s] \quad (8)$$

Optymalna strategia  $\pi^*$  w każdym stanie ze zbioru  $S$  wybierze taką akcję, która da maksymalną obniżoną nagrodę która agent jest w stanie uzyskać z danego stanu. Jeśli optymalna wartość  $Q^*(s', a')$  w następnym kroku jest znana dla wszystkich możliwych akcji  $a'$  to optymalną strategią podejmowania akcji jest wybieranie  $a'$  maksymalizując oczekiwana obniżoną nagrodę.

$$\pi^*(s) = \arg \max_a Q^*(s, a) \quad (9)$$

$$V_\pi(s) = \sum_a \pi(a|s) [R(s, a) + \gamma \sum_{s'} P(s'|s, a) V_\pi(s')] \quad (10)$$

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') Q_\pi(s', a') \quad (11)$$

$$V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s, a)$$

Funkcję (11) można rozwiązać iterując równanie Temporal Difference[8]. Parametr  $\alpha$  to wielkość kroku optymalizacyjnego. Taka iteracja funkcji wartości akcji zbiega do optymalnej funkcji  $Q_i \rightarrow Q^*$  dla  $i \rightarrow \infty$ [8].

$$Q_{i+1}(s, a) = Q_t(s, a) + \alpha [R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q_t(s', a') - Q_t(s, a)] \quad (12)$$

Takie podejście jest niepraktyczne w przypadku tej pracy, gdyż przestrzeń stanów jest za duża, algorytm uczący nie ma dostępu do funkcji  $P(s'|s, a)$  oraz epizody mogą być w skrajnych przypadkach bardzo długie.

## 5 Głębokie uczenie funkcji wartości akcji $Q$

Bez dostępu do funkcji przejścia  $P(s'|s, a)$  proces optymalizacji bazuje na seriach doświadczeń generowanych podczas interakcji ze środowiskiem. Wszystkie doświadczenia, razem z

następnym zaobserwowanym stanem, są na zapisywane w pamięci  $M$ . Funkcja wartości jak i funkcja wartości akcji opiera się na założeniu, że kolejne akcje w serii były wybierane zgodnie ze strategią  $\pi$ .

$$V_\pi(s) = \mathbb{E}[R_t + \gamma V_\pi(s_{t+1})|s_t = s, a_t \sim \pi(s_t)] \quad (13)$$

$$Q_\pi(s, a) = \mathbb{E}[r_t + \gamma Q_\pi(s_{t+1}, a_{t+1})|s_t = s, a_t = a, \pi] \quad (14)$$

Celem znalezienia optymalnej strategii korzystam ze sztucznej sieci neuronowej z parametrami  $\theta$ , której zadaniem jest aproksymacja optymalnej funkcji wartości akcji  $Q(s, a : \theta) \approx Q^*(s, a)$ . Sieć neuronową można uczyć minimalizując sekwencje funkcji koszta  $L_i(\theta_i)$ . Funkcja koszta odpowiada kwadratowi błędu z równania (12) z tą różnicą, że stan  $s$ , akcja  $a$ , nagroda  $r$  oraz stan po aktualizacji środowiska  $s'$  są losowane z pamięci doświadczeń  $M$ .

$$L_i(\theta_i) = \mathbb{E}_{s, a, r, s' \sim M} [(r + \gamma \max_{a'} Q(s', a' : \theta_{i-1}) - Q(s, a : \theta_i))^2] \quad (15)$$

Wagi sieci neuronowej optymalizuje się za pomocą stochastycznego spadku wzdłuż gradientu.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a, r, s' \sim M} [(r + \gamma \max_{a'} Q(s', a' : \theta_{i-1}) - Q(s, a : \theta_i)) \nabla_{\theta_i} Q(s, a : \theta_i)] \quad (16)$$

Wagi  $\theta_{t-1}$  są podczas treningu zamrożone. Korzystanie z wag z poprzedniej aktualizacji mogłoby prowadzić do katastroficznego zapominania, dlatego wagi  $\theta_{t-1}$  zastępowane są wagami  $\theta_f$  które co stały liczbę kroków przyjmują aktualną wartość  $\theta_t$ .

## 5.1 Problemy głębokiego uczenia ze wzmacnieniem

Kolejne stany środowiska są bardzo do siebie podobne, co w procesie uczenia prowadziły do zbiegania do lokalnego minimum. Celem uniknięcia tej sytuacji pamięć  $M$  ma dużą pojemność  $n$ , a partie do uczenia o wielkości  $b < n$  są wybierane z pamięci z jednakowym prawdopodobieństwem  $(S_b, A_b, R_b, S'_b) \sim M$ . Prowadzi to do zwiększenia niezależności stanów, na podstawie których obliczany jest krok uczący. Drugim sposobem na przeciwdziałanie podobieństwa stanów jest jednoczesne symulowanie wielu niezależnych pojazdów, które są inicjalizowane w sposób losowy. Prowadzi to do lepszej eksploracji przestrzeni stanów oraz pozwala na lepszą ewaluację aktualnej strategii podczas treningu. Wyniki przeprowadzonych eksperymentów wskazują na znaczącą poprawę wyników już przy pięciu symulowanych jednocześnie pojazdach.

Gdyby agent od początku chciwie podejmował akcje na podstawie losowo zainicjalizowanej funkcji wartości akcji  $a_t = \max_a Q(s_t, a : \theta)$  mógłby, przez niedostateczną eksplorację, pozostać w lokalnym minimum. Eksplorację wymusza się na agencie poprzez wybieranie z prawdopodobieństwem  $\epsilon$  losowej akcji  $a_t \sim A$ , a z prawdopodobieństwem  $1 - \epsilon$  akcji chciwej  $a_t = \max_a Q(s_t, a : \theta)$ . Ten sposób podejmowania akcji nazywa się  $\epsilon$ -greedy[8]. Wartość  $\epsilon$  jest

warunkowana początkową wartością  $\epsilon_{start}$ , końcową wartością  $\epsilon_{end}$  oraz liczbą kroków  $\epsilon_{decay}$  po których  $\epsilon$  ma osiągnąć wartość końcową. W trakcie treningu  $\epsilon$  zmniejsza się z każdym krokiem zgodnie z równaniem  $\epsilon_t = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end})e^{(-t/\epsilon_{decay})}$ .

Następnym problemem w głębokim uczeniu ze wzmacnieniem są eksplodujące gradienty. Główną przyczyną eksplozji jest duże odchylenie standardowe obserwowanych nagród. Drugą przyczyną jest sposób aktualizacji wag  $\theta_t$ , który zależąc od wag  $\theta_f$  może spowodować sprzężenie zwrotne i w wyniku eksplozję. Żeby przeciwdziałać eksplodującym gradientom korzystam z funkcji koszta *smooth\_L1\_loss*[3] celem zmniejszenia wpływu skrajnych pomiarów na całkowy gradient danego kroku uczącego.

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & dla |x| < 1 \\ |x| - 0.5 & dla |x| \geq 1 \end{cases} \quad (17)$$

Podobnie jak w Atari DQN[6] korzystam z optymalizatora *RMSprop* zaproponowanego przez G. Hintona[9] a po raz pierwszy opublikowanego w pracy na temat optymalizacji rekurencyjnych sieci neuronowych służących do generowania tekstu[5]. Optymizator przechowuje średnią kroczącą kwadratów gradientów dla każdej wagi  $w \in \theta$ , a podczas aktualizacji wagi dzieli gradient przez pierwiastek kwadratowy tej średniej.

$$\mathbb{E}[g^2]_t = \beta \mathbb{E}[g^2]_{t-1} + (1 - \beta) \left( \frac{\delta L(x)}{\delta w} \right)^2 \quad (18)$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\mathbb{E}[g^2]_t}} \frac{\delta L(x)}{\delta w} \quad (19)$$

## 5.2 Sztuczna sieć neuronowa

Funkcja aproksymująca  $Q^*$  jest złożona z wielu warstw funkcji liniowych zawierających nieliniowe aktywacje pomiędzy kolejnymi warstwami. Pierwsza funkcja liniowa przyjmuje na wejściu stan otrzymany ze środowiska, przekształca na wektor o wymiarze  $w$  a następnie oblicza nieliniową funkcję aktywacji *ReLU*[2] na wektorze wyjściowym. Kolejne  $d$  funkcji liniowych o tej samej szerokości  $w$  przekształca wektor w ten sam sposób co pierwsza funkcja. Ostatnia funkcja liniowa przekształca wektor o wymiarze  $w$  na wektor o wymiarze odpowiadającym rozmiarowi przestrzeni akcji  $A$ , a wartości reprezentują  $Q_\pi(*|s)$ . W tej pracy korzystam z algorytmów uczenia maszynowego zaimplementowanych w bibliotece PyTorch[7].

### 5.3 Algorytm

---

**Algorithm 1:** Głębokie uczenie Q-network z wykorzystaniem pamięci

---

Inicjalizacja parametrów  $\theta$  funkcji wartości akcji  $Q$ ,  $\theta_f = \theta$   
Inicjalizacja środowiska oraz otrzymanie pierwszego stanu  $s_1$   
**for**  $t=0; t < T; t++$  **do**  
    Z prawdopodobieństwem  $\epsilon$  wybierz losową akcję  $a_t \sim A$ ,  
    w innym przypadku wybierz  $a_t = \max_a Q(s_t, a : \theta)$   
    Zaktualizuj stan środowiska wykonując akcję  $a_t$  i otrzymując  $(r_t, s_{t+1})$   
    Zapisz w pamięci element  $(s_t, a_t, r_t, s_{t+1})$   
    **if**  $t > b$  **then**  
         $(S_b, A_b, R_b, S'_b) \sim M$   
        Zaktualizuj wagi  $\theta$  zgodnie z  $(R_b + \gamma \max_{a'} Q(S'_b, a' : \theta_f) - Q(S_b, A_b : \theta))^2$   
    **end**  
    **if**  $(t \% \text{częstotliwość aktualizacji } \theta_f) == 0$  **then**  
         $\theta_f = \theta$   
    **end**  
**end**

---

## 6 System do przeprowadzania eksperymentów

Chcąc przetestować w jaki sposób algorytm uczenia DQN reaguje na różne parametry potrzebowałem systemu przechowującego wyniki przeprowadzonych eksperymentów, kolejkę eksperymentów do wykonania oraz proces, który wykonuje kolejne eksperymenty. Dzięki uprzejmości Naukowego Koła Robotyki i Sztucznej Inteligencji na Uniwersytecie Jagiellońskim miałem dostęp do dwóch maszyn obliczeniowych, co wymagało napisania dodatkowych narzędzi odpowiedzialnych za synchronizację kolejki eksperymentów i wyników.

### 6.1 Tworzenie eksperymentów

Szablon eksperymentów tworzony jest po wskazaniu plików tekstowych w formacie JSON, które zawierają domyślne parametry potrzebne do uruchomienia danego eksperymentu. Szablon można uruchomić jako eksperyment lub można go wykorzystać jako bazę do stworzenia wielu eksperymentów, nazywanych serią, różniących się pomiędzy sobą parametrami. Serię eksperymentów tworzy się podając folder, w którym znajduje się szablon, oraz listy wartości parametrów. Eksperymenty tworzone są z elementów iloczynu wszystkich list wartości parametrów, a następnie zapisywane są w folderze kolejki.

## 6.2 Uruchamianie obliczeń

Po stworzeniu serii można lokalnie uruchomić skrypt *runq.sh* wykonujący lokalnie eksperymenty z kolejki. W celu wysłania części eksperymentów na zdalne maszyny trzeba uruchomić skrypt *split\_training.sh* który dzieli eksperymenty z kolejki i wysyła je przez ssh do maszyn obliczeniowych. Po uruchomieniu skryptu eksperymenty są losowo dzielone na równe części dla każdej z maszyn zapisanych w pliku *remote\_config.py*. Po podzieleniu eksperymenty są kompresowane a następnie przesyłane na maszynę. Po przesłaniu pliku eksperymenty są rozpakowywane i umieszczone w folderze kolejki. Za pomocą skryptu *work.py* można uruchomić skrypt *runq.sh* na wszystkich zdalnych maszynach. Podczas obliczeń można uruchomić skrypt *status.py* celem sprawdzenia ilości pozostałych eksperymentów na maszynach oraz utylizacji procesorów kart graficznych.

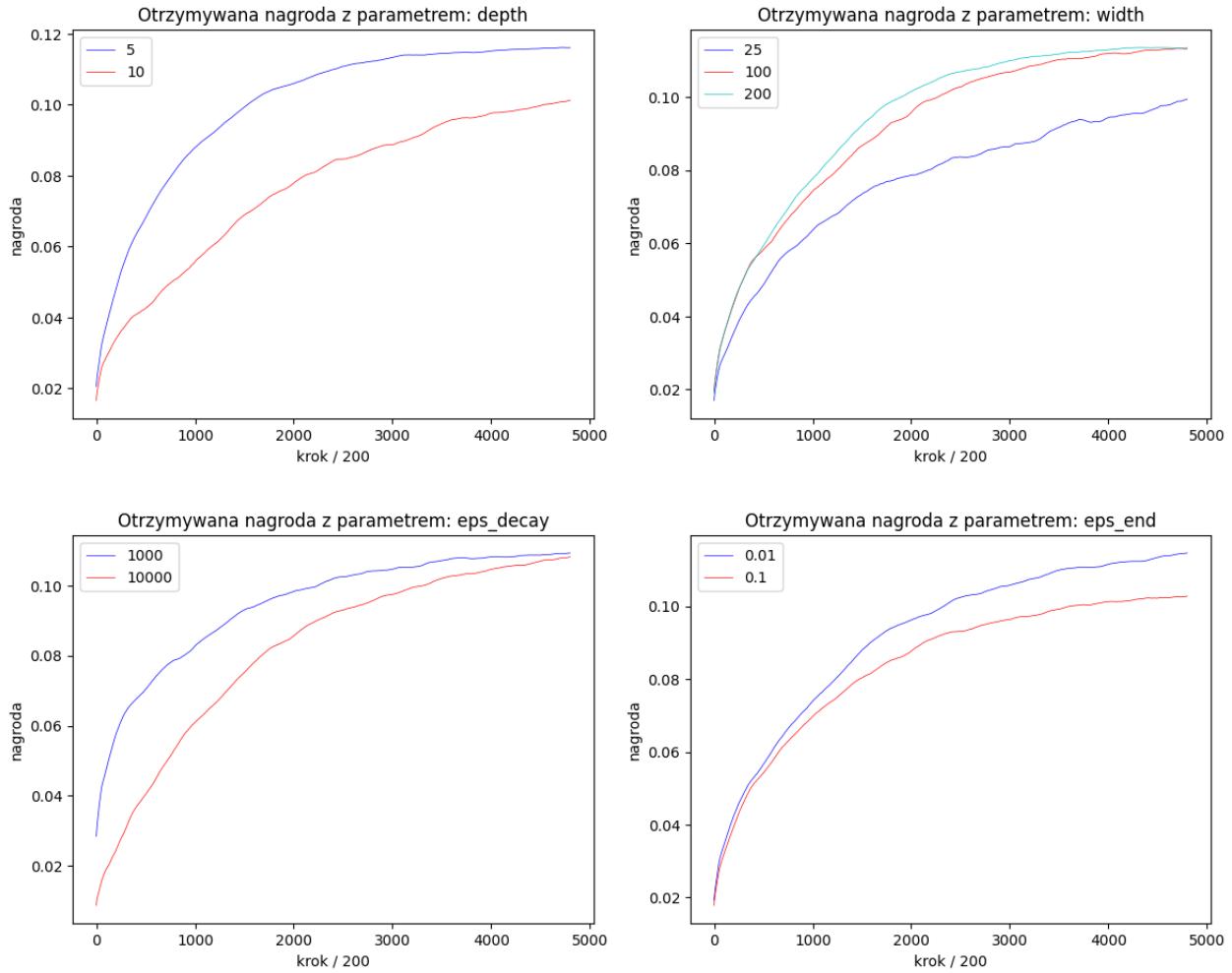
## 6.3 Synchronizacja wyników

Żeby pobrać z maszyn wyniki eksperymentów które zostały zakończone trzeba uruchomić skrypt *pull.py*. Skrypt ten wysyła na maszynę listę plików które znajdują się lokalnie. Maszyna po porównaniu lokalnych plików z otrzymaną listą kompresuje te pliki, które nie znajdowały się na liście. Skrypt następnie pobiera skompresowane wyniki i rozpakowuje je lokalnie.

# 7 Przeprowadzone eksperymenty i wyniki

Chcąc dokładnie przetestować możliwości algorytmu DQN zdecydowałem się na zastosowanie metodologii przeszukiwania siatki, co pozwoliło mi sprawdzić wpływ wszystkich parametrów na wyniki uczenia. W każdym z trzech scenariuszy sprawdzałem inne kombinacje parametrów. Wadą takiego podejścia okazał się być czas wymagany do zakończenia wszystkich eksperymentów. Długość trwania wszystkich eksperymentów wynosi milion kroków aktualizacji środowiska. Na każdym wykresie przedstawiony jest wpływ pojedynczego parametru na uczenie. Wyniki wszystkich eksperymentów są dzielone na grupy na podstawie wartości badanego parametru. Wykres przedstawia różnymi kolorami uśrednioną nagrodę otrzymaną w każdym kroku każdej z grup. Na wykresach pozioma oś odpowiada liczbie kroków aktualizacji podzielonych przez dwieście, gdyż średnia nagroda otrzymywana przez agenta w czasie treningu była zapisywana co 200 kroków. Pionowa oś odpowiada wartości nagrody zdobytej w danym kroku.

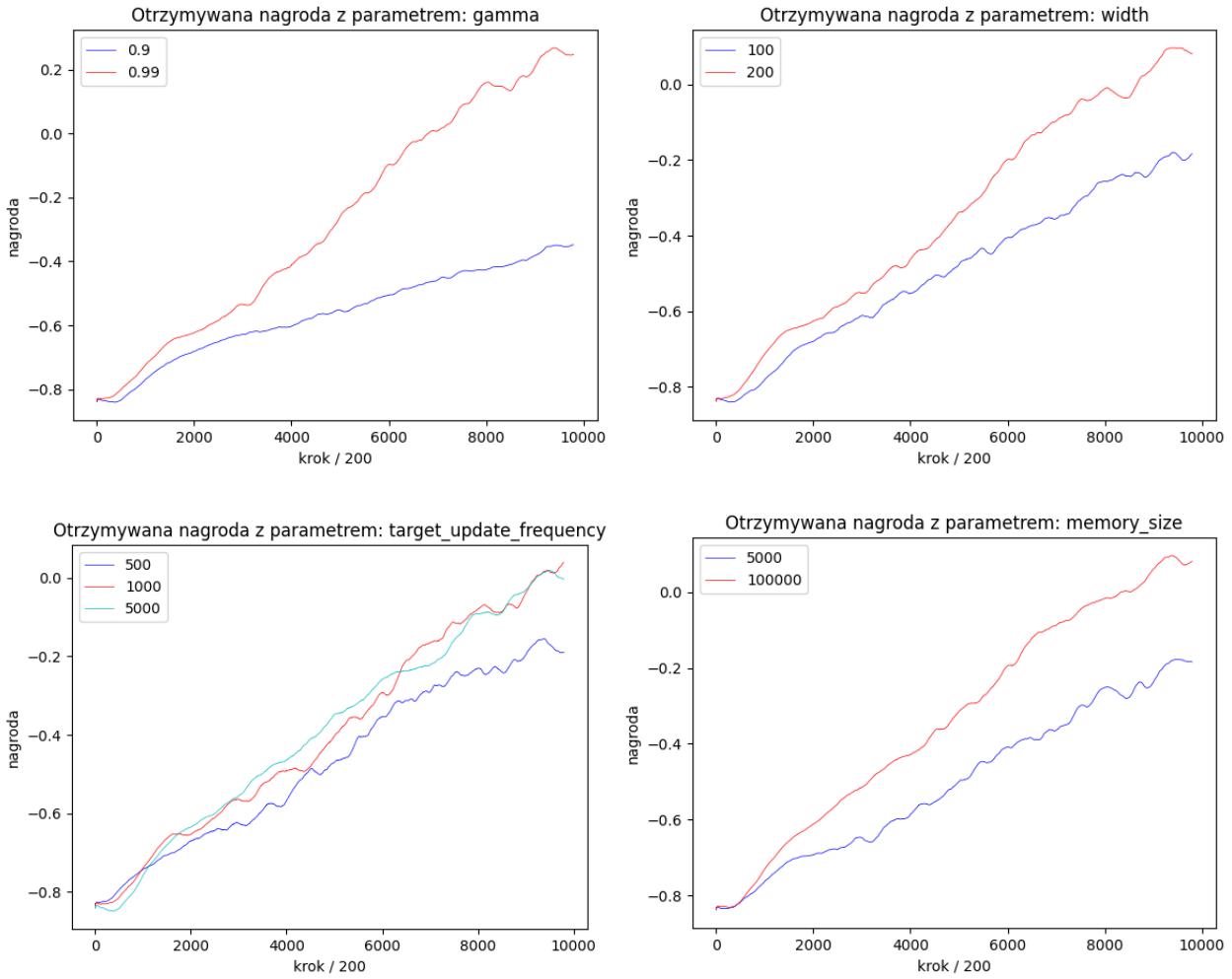
## 7.1 Pierwszy scenariusz



Rysunek 2: Wpływ parametrów na zdobywaną nagrodę w pierwszym środowisku.

Pierwszy scenariusz polegający na zatrzymywaniu pojazdu okazał jest najłatwiejszym środowiskiem do rozwiązania, co można wnioskować obserwując niewielkie postępy w końcowym etapie uczenia po początkowym okresie dynamicznego wzrostu. Sieć o mniejszej ilości szerszych warstw spisywała się lepiej od bardziej rozbudowanych sieci neuronowych. Parametry kontrolujące sposób eksploracji wpływają na wynik treningu zgodnie z oczekiwaniemi. Większy (*eps\_end*) powoduje częstsze wybieranie losowej akcji i co za tym idzie zniża nagrodę funkcji dobrze dopasowanej. Mniejszy (*eps\_decay*) skraca okres eksploracji pozwalając na lepsze dopasowanie funkcji w przypadku prostych środowisk. W trudniejszych środowiskach zwiększoną stabilność i dłuższa eksploracja zwiększoną (*eps\_decay*) może być pożądana.

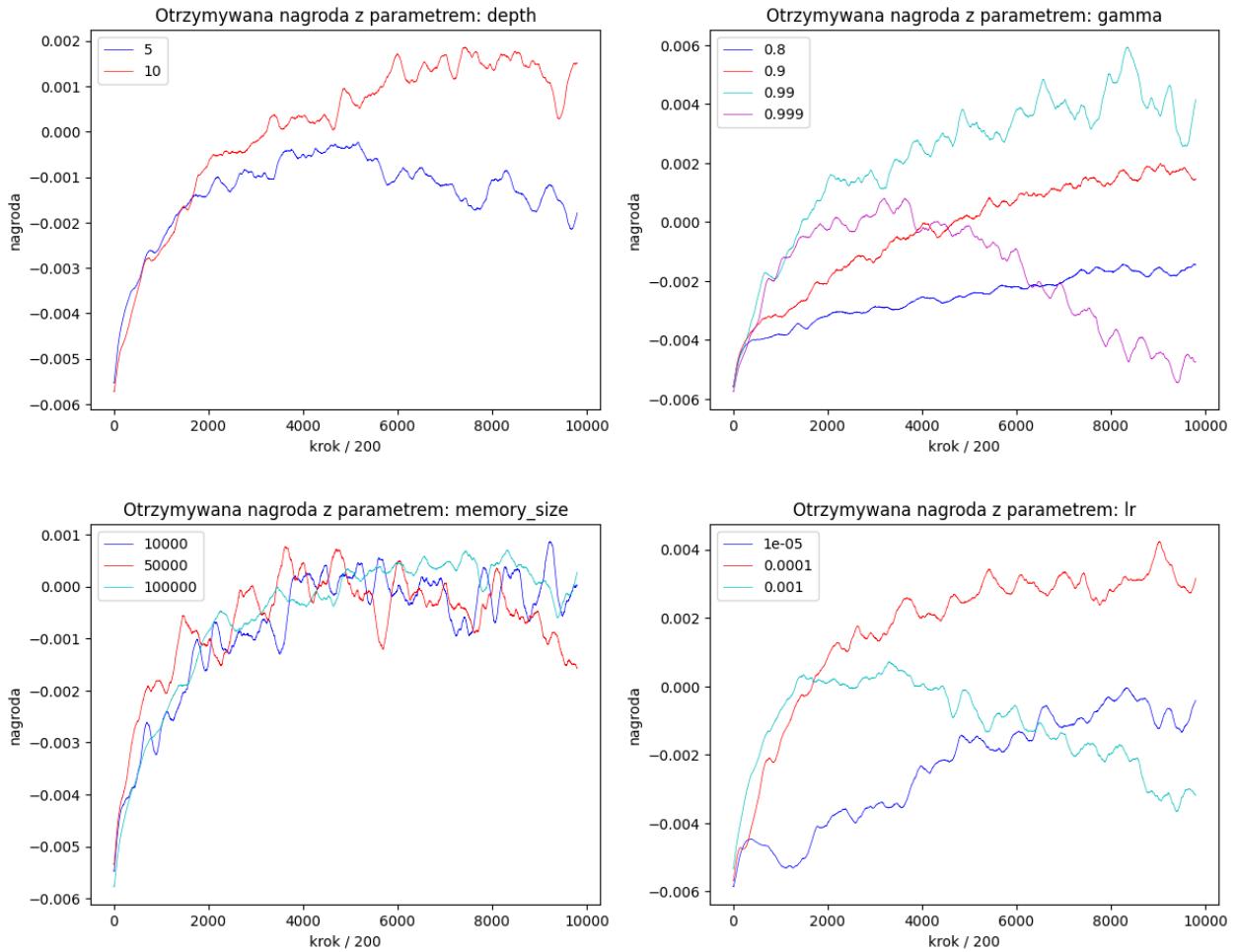
## 7.2 Drugi scenariusz



Rysunek 3: Wpływ parametrów na zdobywaną nagrodę w drugim środowisku.

W drugim scenariuszu widać nietypowy, liniowy przyrost zdobywanej przez agenta nagrody. Wynika to ze sposobu obliczania nagrody który odpowiada zmianie odległości. Zwiększenie pamięci poprawiło osiągane wyniki oraz stabilność treningu. Zwiększenie parametru ( $\gamma$ ) kontrolującego planowanie również poprawiło wyniki. Pojawia się teżauważalny wpływ częstotliwości aktualizacji wag sieci na podstawie której obliczana jest maksymalna wartość akcji z następnego kroku.

### 7.3 Trzeci scenariusz



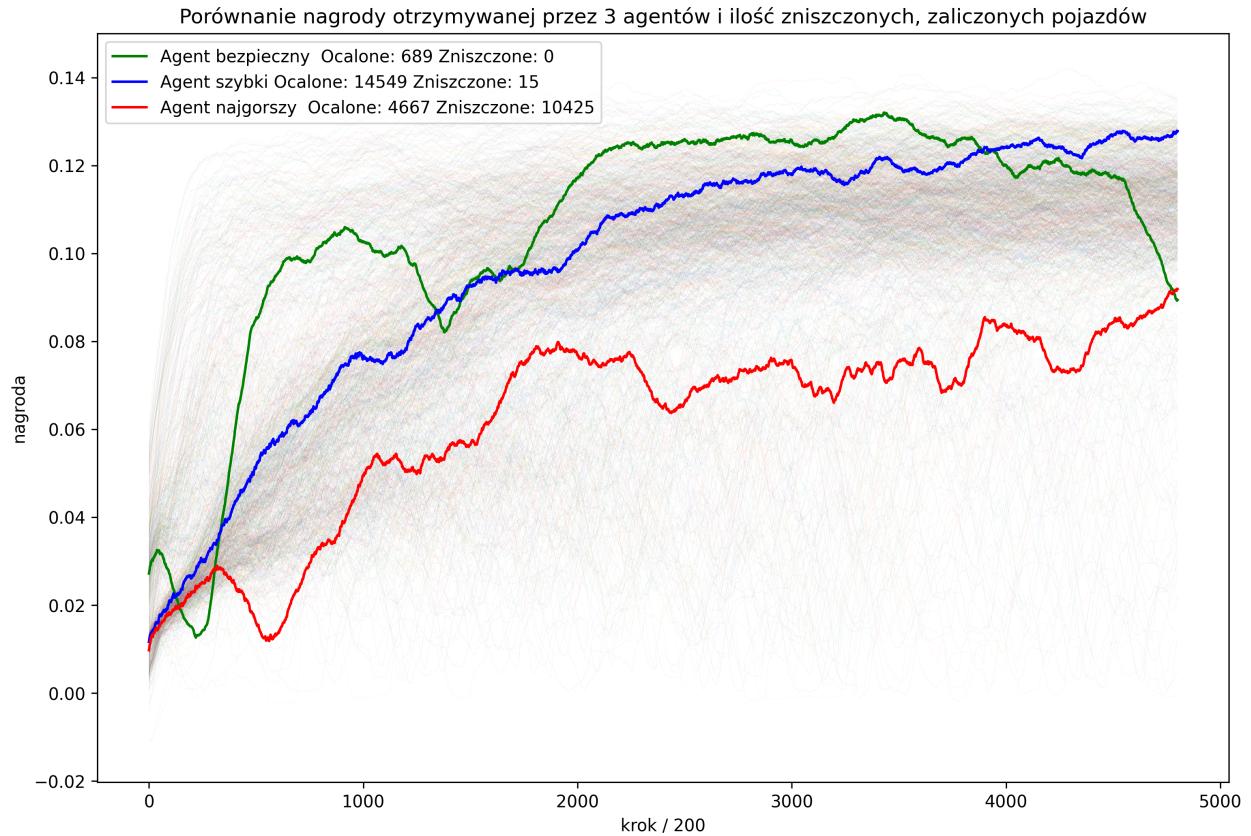
Rysunek 4: Wpływ parametrów na zdobywaną nagrodę w trzecim środowisku.

W przeciwieństwie do pierwszego scenariusza trzeci scenariusz jest bardziej skomplikowany. Pogłębienie sieci neuronowej przyniosło lepsze rezultaty. Parametr *gamma* nie powinien przyjmować wartości bliskich jeden, gdyż dochodzi do tragicznego zapominania. Mniejsza wartość tego parametru nie wpływa tak negatywnie na wydajność nauki jak za duża wartość. Krok uczący o wielkości 0.0001 (*lr*) okazał się być najlepszy i został zastosowany również w pozostałych środowiskach.

## 8 Wyniki testów

Na koniec treningu zapisywane były wagi sieci neuronowej agenta. Testowanie polegało na zliczaniu pojazdów, które zakończyły symulację z nagrodą (Ocalone) bądź karą (Zniszczone) w przeciągu 100000 kroków aktualizacji środowiska symulującego 100 pojazdów jednocześnie. Na wykresach zaznaczam kolorem czerwonym najgorszego agenta który zniszczył najwięcej statków, kolorem niebieskim najszybszego agenta który ocalił największą ilość statków oraz kolorem zielonym najbezpieczniejszego agenta który miał najmniejszą ilość zniszczonych pojazdów. Wyniki otrzymywanej podczas treningu nagrody przez trzech wybranych agentów rysuję na tle wszystkich innych trenowanych agentów żeby zarysować wynik trzech w odniesieniu do całej populacji. Parametry różniące tych trzech agentów są wypisane w tabelce pod wykresem.

## 8.1 Pierwszy scenariusz

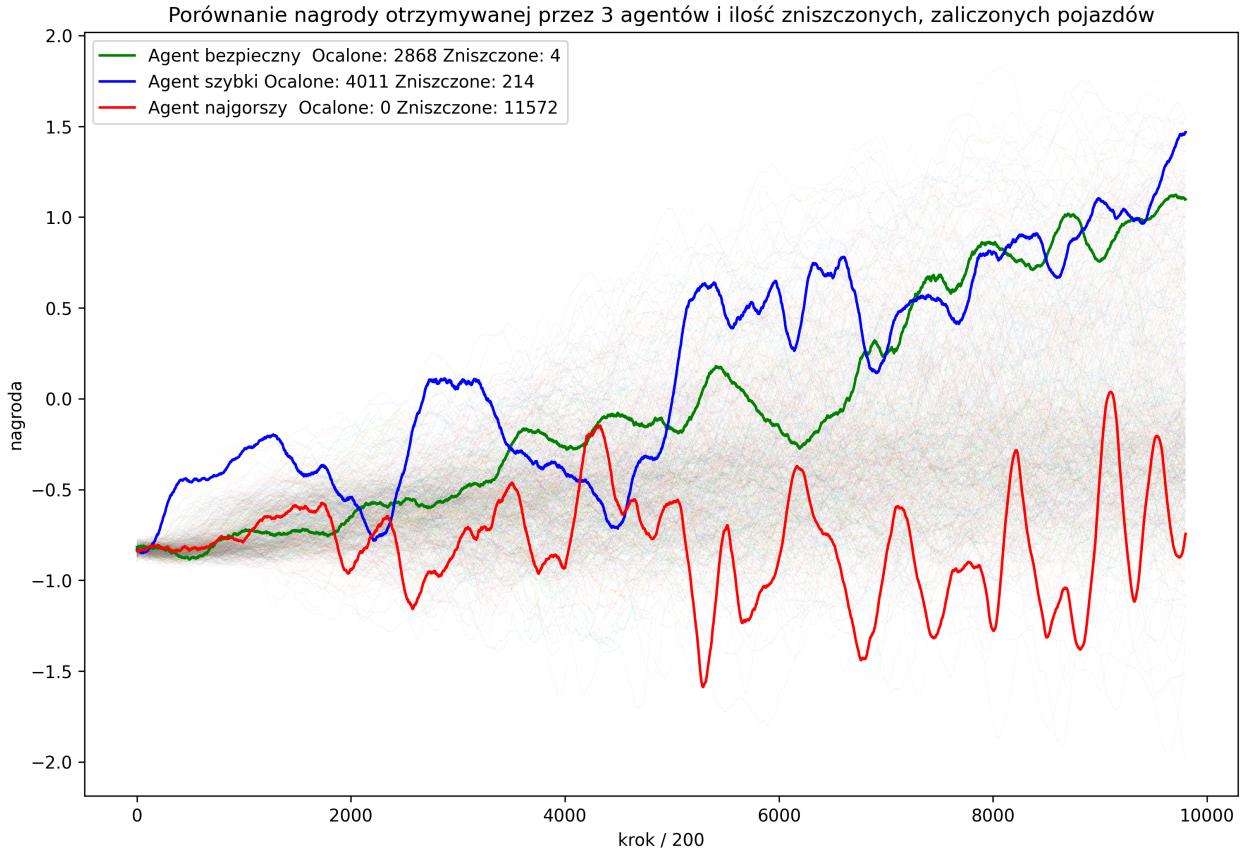


Rysunek 5: Wyniki testu w pierwszym środowisku.

Agent	living penalty	depth	width	eps end	eps decay	batch size
Najgorszy	1.0	10	25	0.1	10000	128
Szybki	0.01	5	200	0.01	10000	32
Bezpieczny	0.25	5	25	0.01	1000	32

Bezpieczny agent zapomniał optymalnej strategii. Można by również podejrzewać, że parametr  $living\_penalty$  zadecydował o ostatecznym zachowaniu agenta, jednak by to stwierdzić potrzebne są dużo szersze badania.

## 8.2 Drugi scenariusz

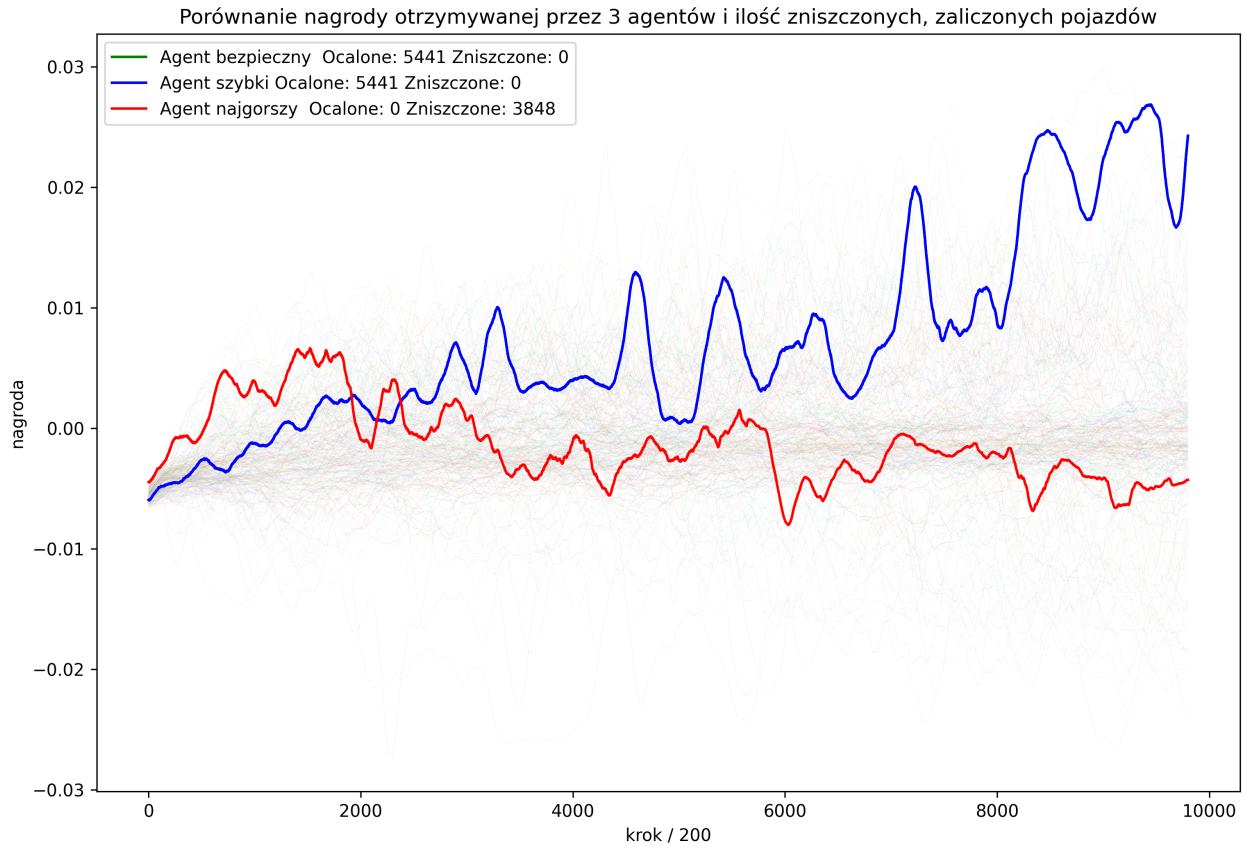


Rysunek 6: Wyniki testu w drugim środowisku.

Agent	depth	width	eps end	memory size	target update	batch size	living penalty
Najgorszy	10	100	0.01	5000	500	128	0
Szybki	10	200	0.1	100000	1000	256	0.1
Bezpieczny	20	200	0.1	100000	5000	128	0.05

W przeciwieństwie do wyników pierwszego scenariusza nic żaden parametr nie wskazuje wyraźnie na przyczynę tak złego dopasowania najgorszego agenta. Gdyby była to kwestia funkcji nagrody która uniemożliwia naukę strategii to pozostały agenci też mieliby problem z nauką.

### 8.3 Trzeci scenariusz



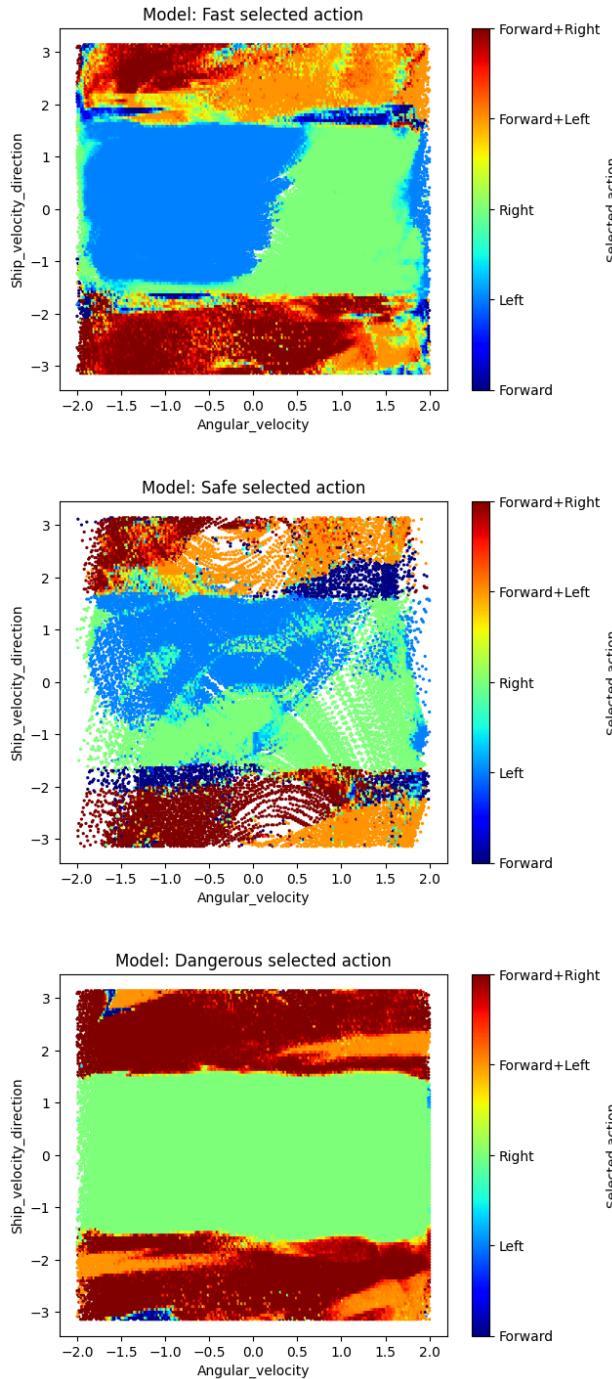
Rysunek 7: Wyniki testu w trzecim środowisku.

Agent	eps end	eps decay	memory size	lr
Najgorszy	0.1	100000	10000	0.001
Bezpieczny	0.01	1000000	100000	0.0001

Agent który na samym początku uczenia zapowiadał się na najlepszego w okolicy kroku 2000 zmienił trajektorię nauki kończącą się najgorszym wynikiem. Porównując wyniki tego testu z wynikami poszczególnych parametrów podczas treningu powodem tak tragicznego zapomnienia mógł być parametr *lr*.

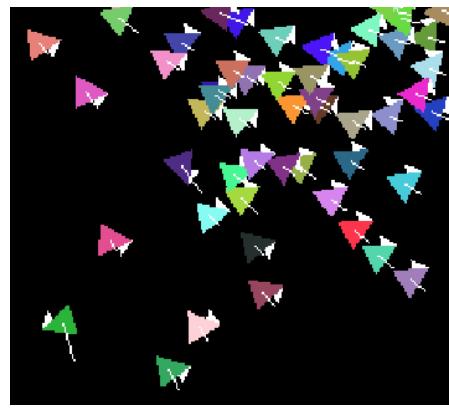
## 9 Analiza trzech agentów z pierwszego środowiska

Rysunek 8: Akcje wybrane podczas testów.



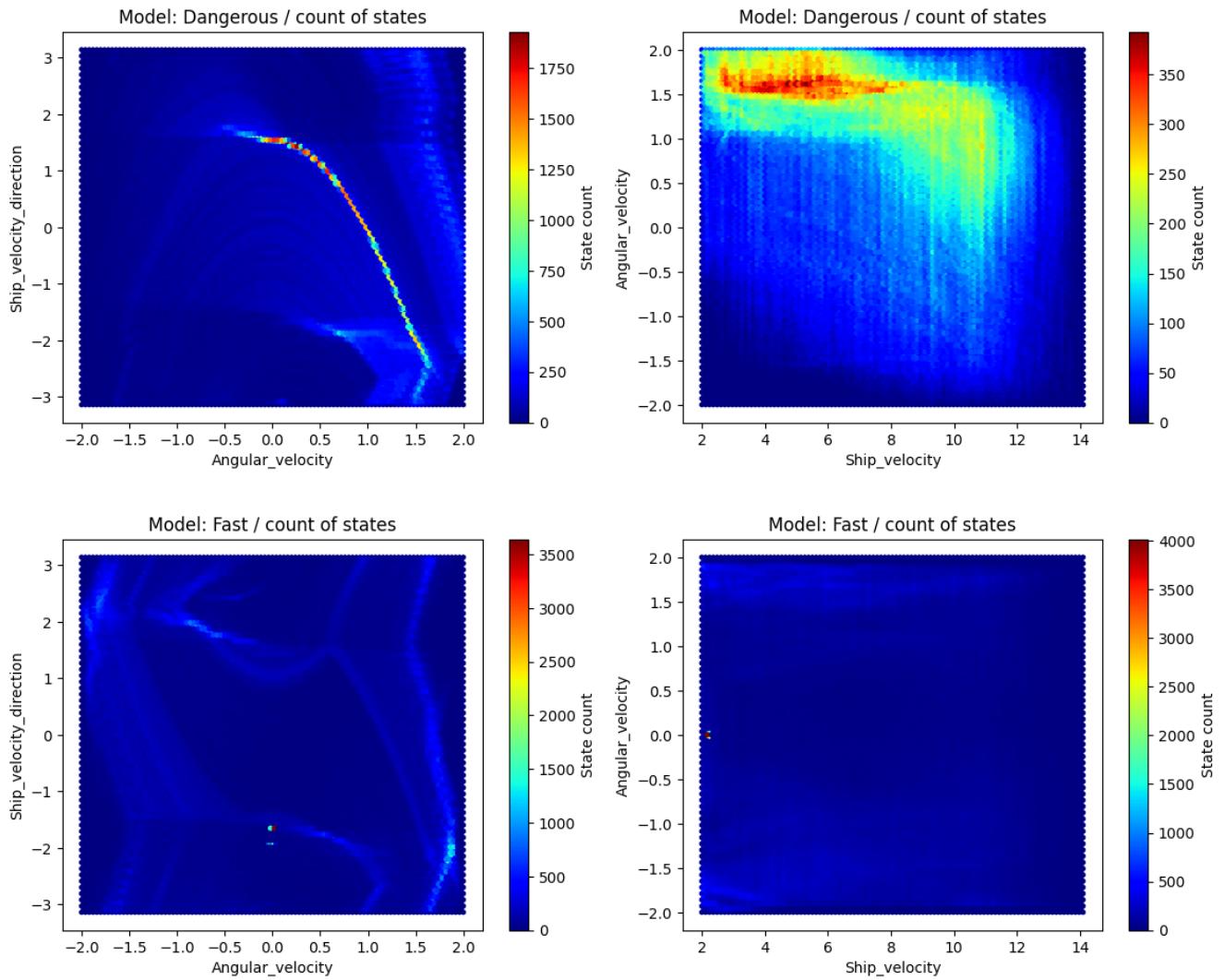
Na rysunku 8 znajdują się trzy wykresy przedstawiające wybrane przez agentów akcje podczas testowania. Stan przedstawiony jest na osiach. Oś X odpowiada prędkości obrotowej a oś Y kątowi pomiędzy kierunkiem statku a wektorem prędkości oraz prędkości kątowej. Statek zwalnia najefektywniej w momencie kiedy pojazd zwrócony jest w przeciwną stronę co prędkość, czyli wartość na osi Y jest bliska  $\pi \text{ rad}$  bądź  $-\pi \text{ rad}$ . Dobrą akcją jest również uruchomienie głównego silnika kiedy kąt wynosi ponad  $\pi/2 \text{ rad}$  bądź  $-\pi/2 \text{ rad}$ .

Agent bezpieczny ma na wykresie dużo białych plam odpowiadających braku obserwacji tego stanu. Wynika to z tego, że pomimo takiej samej ilości kroków uczących i obserwacji statek wykonywał sukcesywnie negującą się akcję uruchamiając prawy i lewy silnik utrzymując stałą prędkość i rotację pojazdu jak widać na załączonym poniżej rysunku 9. Agent jest najbezpieczniejszy bo wszystkie pojazdy odlatują w nieskończoną przestrzeń.

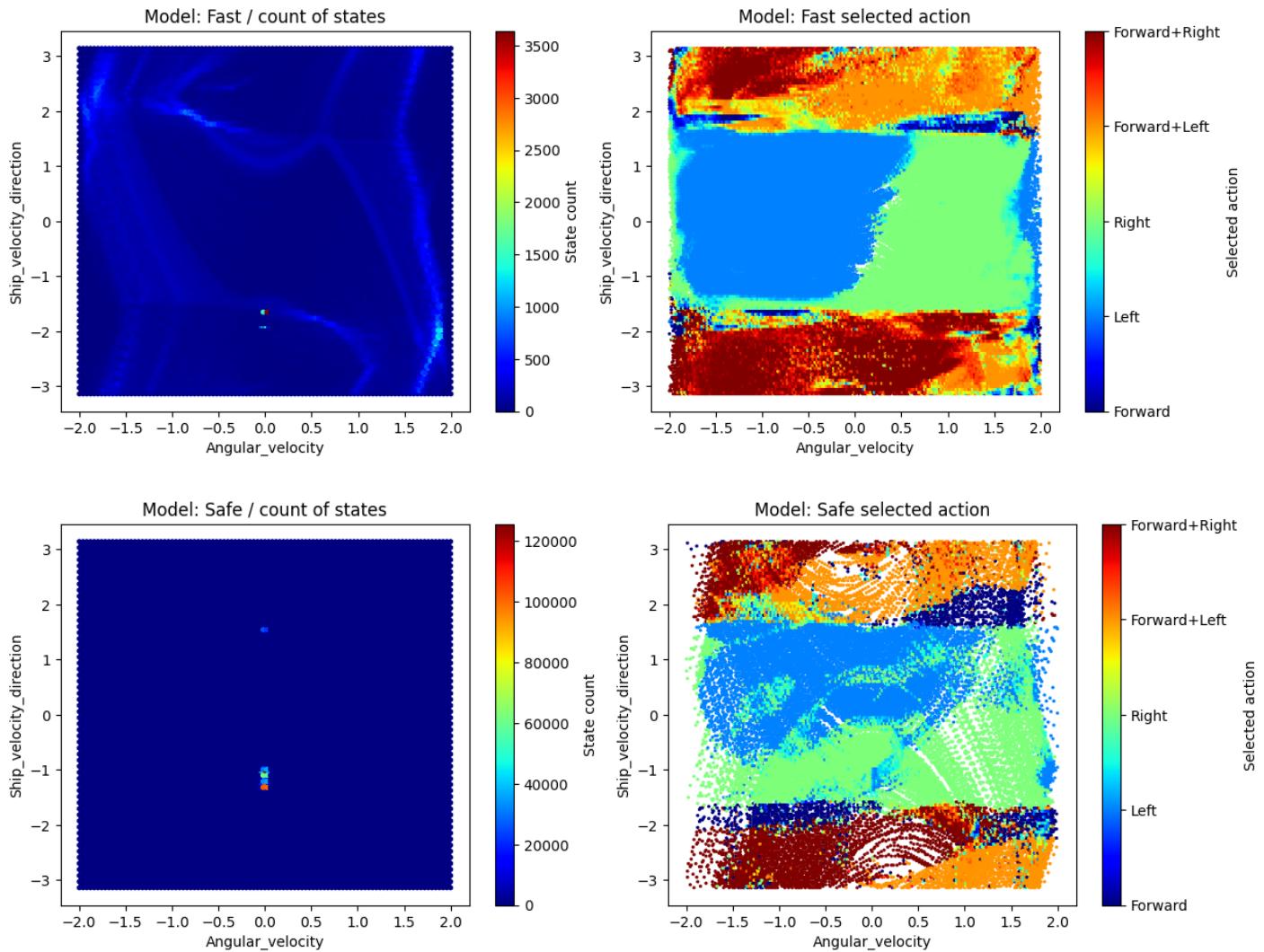


Rysunek 9: Cała populacja pojazdów zablokowana w cyklu akcji przez strategię bezpiecznego agenta

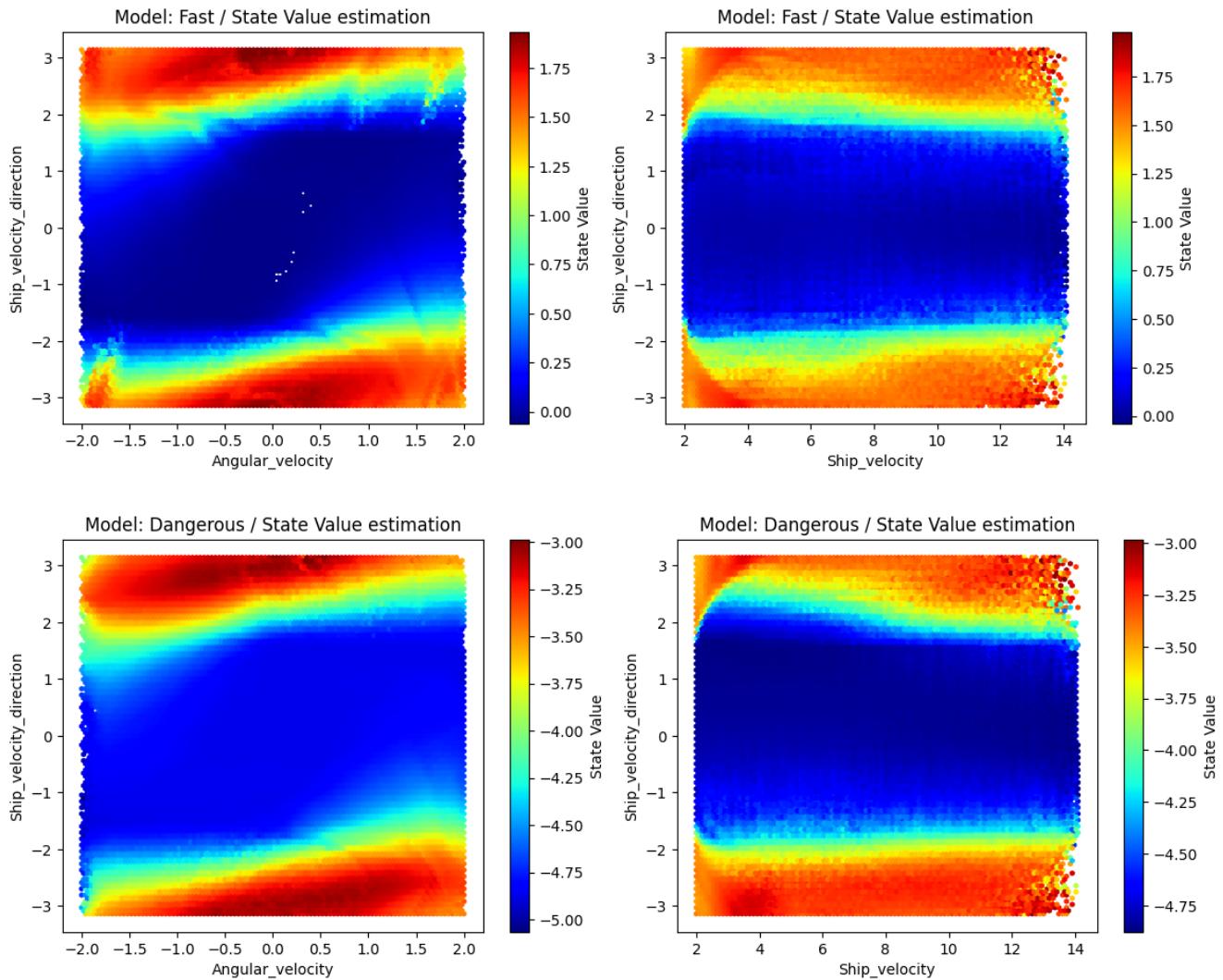
Największą różnicę widać w środkowej części wykresów gdzie agent zwrócony jest przodem do kierunku lotu, to jest nie może jeszcze hamować. Agent niebezpieczny w ogóle nie korzysta z akcji skręcania w lewo. Wszystkie pojazdy przez większość stanów skręcają w prawo, co powoduje znaczącą zmianę dystrybucji obserwowanych stanów co wpływa na dalszą naukę agenta.



Rysunek 10: różnicę w dystrybucji stanów obserwowanych podczas testów przez agenta niebezpiecznego i szybkiego. Chcę sprawdzić czym jest ten zielono czerwony punkt w dolnej części lewego dolnego wykresu



Rysunek 11: W podobnych miejscach jest czerwona plamka powodująca zapętlenie pojazdu w nieskończonym cyklu prawo/lewo. Chciałbym czerwonym markerem zaznaczyć dokładne miejsca w strategiach które powodują takie zachowanie



Rysunek 12: Wartości stanów wg szybkiego i niebezpiecznego są kształtem do siebie podobne, chociaż agent niebezpieczny przez living penalty przewiduje najlepszą oczekiwana nagrodę -3

## Literatura

- [1] boost c++ libraries.
- [2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.
- [3] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.
- [4] Laurent Gomila. Simple and fast multimedia library sfml.
- [5] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.
- [6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
- [7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.
- [9] Geoffrey Hinton with Nitish Srivastava Kevin Swersky. Neural networks for machine learning.

## Spis rysunków

1	Pojazd . . . . .	4
2	Wpływ parametrów na zdobywaną nagrodę w pierwszym środowisku. . . . .	12
3	Wpływ parametrów na zdobywaną nagrodę w drugim środowisku. . . . .	13
4	Wpływ parametrów na zdobywaną nagrodę w trzecim środowisku. . . . .	14
5	Wyniki testu w pierwszym środowisku. . . . .	16
6	Wyniki testu w drugim środowisku. . . . .	17
7	Wyniki testu w trzecim środowisku. . . . .	18

8	Akcje wybrane podczas testów. . . . .	19
9	Cała populacja pojazdów zablokowana w cyklu akcji przez strategię bezpiecznego agenta . . . . .	20
10	różnica w dystrybucji stanów obserwowanych podczas testów przez agenta niebezpiecznego i szybkiego. Chcę sprawdzić czym jest ten zielono czerwony punkt w dolnej części lewego dolnego wykresu . . . . .	21
11	W podobnych miejscach jest czerwona plamka powodująca zapętlenie pojazdu w nieskończonym cyklu prawo/lewo. Chciałbym czerwonym markerem zaznaczyć dokładne miejsca w strategiach które powodują takie zachowanie . . . . .	22
12	Wartości stanów wg szybkiego i niebezpiecznego są kształtem do siebie podobne, chociaż agent niebezpieczny przez living penalty przewiduje najlepszą oczekiwana nagrodę -3 . . . . .	23

## Spis tabelic