Implementacja AI do gry w Trackmanię z wykorzystaniem uczenia ze wzmocnieniem

Raport projektowy

Wojciech Klusek i Aleksander Kuś

25 stycznia 2024

Streszczenie

Niniejszy raport prezentuje projekt implementacji sztucznej inteligencji (AI) do gry w popularną grę wyścigową - Trackmanie 2020, opierając się na metodach uczenia ze wzmocnieniem. Projekt wykorzystuje środowisko `tmrl` i testuje dwa algorytmy: DDPG (Deep Deterministic Policy Gradient) oraz SAC (Soft Actor-Critic). Głównym celem jest osiągnięcie przez AI jak najlepszego czasu przejazdu na testowym torze. Wyniki projektu wskazują, że algorytm SAC osiąga znacznie lepsze rezultaty niż DDPG.

Spis treści

1 Trackmania

TrackMania 2020 [4] to odsłona popularnej serii gier wyścigowych, która zyskała uznanie dzięki swojej dynamicznej rozgrywce. Gra charakteryzuje się połączeniem wyścigów z elementami platformowymi, co sprawia, że jest dużym wyzwaniem zręcznościowym. Gracze mają możliwość ścigania się na rozmaitych torach, które często zawierają skomplikowane pętle, skoki i inne ekstremalne elementy trasy.

Gra jest znana z szybkiego tempa i wymaga od graczy precyzyjnego sterowania oraz umiejętności szybkiego podejmowania decyzji. To sprawia, że TrackMania 2020 jest idealnym środowiskiem do implementacji i testowania algorytmów AI, szczególnie w dziedzinie uczenia ze wzmocnieniem, gdzie algorytmy muszą nauczyć się optymalnych strategii poruszania się po torze, aby osiągnąć jak najlepszy czas przejazdu.

Rysunek 1: TrackMania 2020.

2 Środowisko tmrl

`tmrl`[3] to rozproszony framework dla robotyki, zaprojektowany, do treningu AI z wykorzystaniem uczenia ze wzmocnieniem w aplikacjach czasu rzeczywistego. `tmrl` jest dostarczany z potokiem autonomicznej jazdy dla gry wideo TrackMania 2020.

W środowisku `tmrl` sztuczna inteligencja nie ma wiedzy na temat jazdy samochodem, ani o tym, czym jest droga. AI umieszczona jest w punkcie startowym toru, a jej celem jest nauczenie się jak najszybszego ukończenia toru poprzez eksplorację otoczenia. Samochód przekazuje obserwacje takie jak obrazy, prędkość czy też bieg, do sieci neuronowej, która musi wygenerować najlepsze możliwe sterowanie na podstawie tych obserwacji. Oznacza to, że AI musi w pewien sposób zrozumieć swoje otoczenie. Aby osiągnąć to zrozumienie, eksploruje świat przez kilka godzin (a nawet dni), stopniowo zdobywając wiedzę o tym, jak efektywnie działać.

Środowisko `tmrl` umożliwia:

- Trenowanie polityk przy użyciu algorytmów głębokiego uczenia ze wzmocnie-

niem.

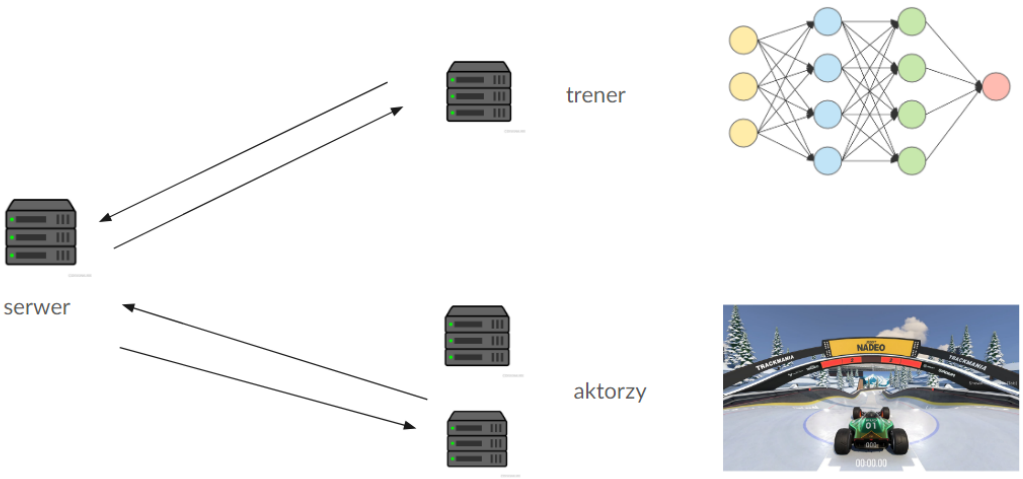
- Sterowanie przy użyciu wirtualnego gamepada z wykorzystaniem surowych zrzutów ekranu lub promieni "LIDAR", przechwyconych w czasie rzeczywistym.

2.1 Architektura środowiska

`tmrl` charakteryzuje się architekturą klient-serwer. Próbki treningowe zbierane są przez kilku agentów, co zwykle oznacza kilka komputerów. Każdy z tych agentów przechowuje zebrane próbki w lokalnym buforze i okresowo wysyła ten bufor do centralnego serwera. Okresowo każdy agent otrzymuje nowe wagi od serwera i aktualizuje swoją sieć.

Centralny serwer może być zlokalizowany w lokalnej sieci lub na innym komputerze. Zbiera próbki od wszystkich podłączonych agentów i przechowuje je w lokalnym buforze. Ten bufor jest następnie przesyłany do trenera. Centralny serwer otrzymuje zaktualizowane wagi polityki od trenera i rozsyła je do wszystkich podłączonych agentów.

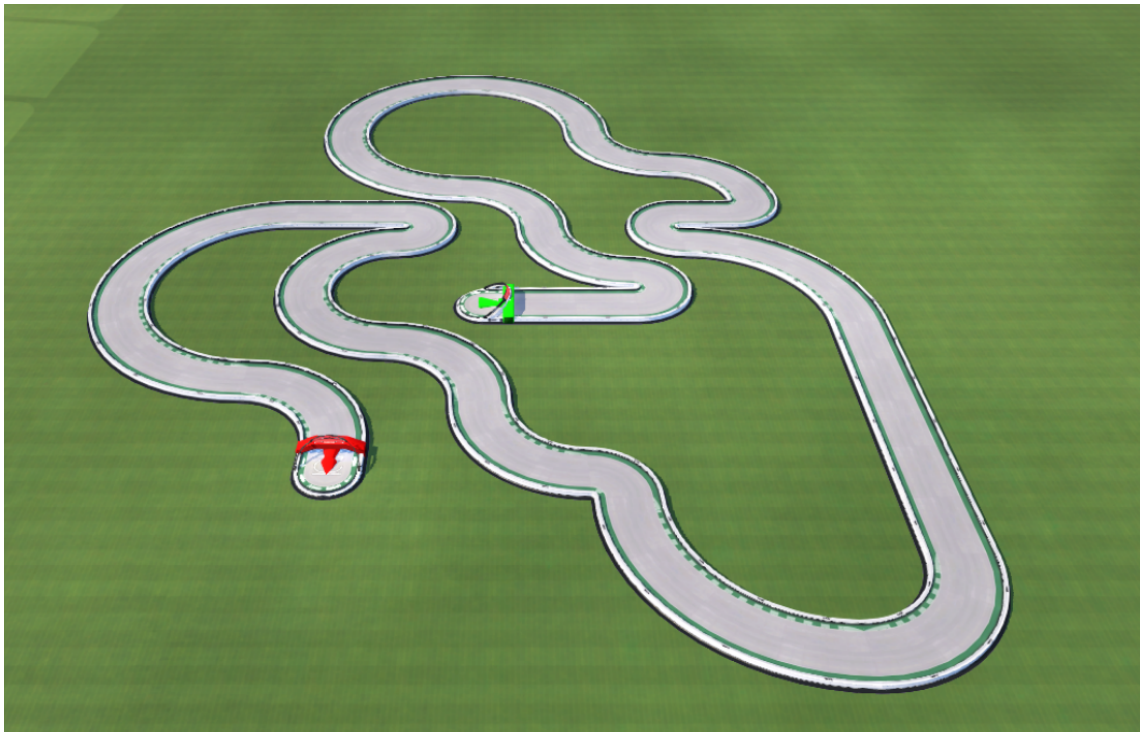
Trener może znajdować się się na komputerze niebędącym agentem w lokalnej sieci lub na innej maszynie. Istnieje również możliwość uruchomienia wszystkiego na jednym komputerze. Trener okresowo otrzymuje próbki zebrane przez serwer i dodaje je do lokalnego bufora. Okresowo wysyła nowe wagi sieci do centralnego serwera.



Rysunek 2: Architektura środowiska.

3 Mapa testowa

W ramach projektu wykorzystana została mapa testowa dostarczona wraz z `tmrl`. Prezentowana trasa, przedstawiona na poniższym rysunku, charakteryzuje się prostotą konstrukcji bez użycia skoków czy innych elementów zwiększających prędkość, co stawia w centrum umiejętność precyzyjnego manewrowania.



Rysunek 3: Mapa testowa.

Trasa wyróżnia się skomplikowanymi zakrętami, wymagającymi inteligentnego sterowania i zaplanowania trajektorii jazdy. Brak elementów takich jak skoki czy boosty sprawia, że AI musi polegać wyłącznie na zdolnościach nawigacyjnych i podejmowaniu decyzji o odpowiednim momencie przyspieszenia, hamowania oraz kierowaniu pojazdem. Przejechanie tej mapy bez błędów stanowi wyzwanie dla sztucznej inteligencji i jest testem jej zdolności do uczenia się oraz adaptacji do nowych warunków.

4 Architektura sieci neuronowych

Poniższa architekura sieci neuronowych została wykorzystana do implementacji algorytmów DDPG oraz SAC.

4.1 Konwolucyjna sieć neuronowa do przetwarzania obrazów

Podstawą przetwarzania danych wejściowych w formie obrazów w naszym zadaniu jest konwolucyjna sieć neuronowa, jest ona odpowiedzialna za ekstrakcję cech z sekwencji klatek wideo dostarczanych przez środowisko gry Trackmania 2020. Architektura została zainspirowana pracą "Optimizing the Neural Architecture of Reinforcement Learning Agents" [6], w której to ta architektura okazała się najlepsza dla gry w Freeway.

Każda z warstw konwolucyjnych ma za zadanie sukcesywne wydobywanie coraz bardziej złożonych cech z obrazu, począwszy od krawędzi i tekstur, a kończąc na

Layer	input	output	kernel	stride
1	4	32	8	4
2	32	64	4	2
3	64	64	3	1

Tabela 1: Konfiguracja warstw sieci neuronowej.

specyficznych dla danego zadania charakterystykach.

4.2 Wielowarstwowy perceptron do generowania akcji

Wyjście z konwolucyjnej sieci neuronowej jest następnie przekazywane do wielowarstwowego perceptronu (MLP), którego zadaniem jest generowanie ostatecznych akcji sterujących pojazdem w grze. MLP składa się z dwóch warstw ukrytych, każda zawierająca 512 neuronów.

Warstwy te służą do dalszego przetwarzania i integracji cech wysokopoziomowych, co pozwala na efektywne mapowanie skomplikowanych danych wejściowych na zestaw akcji. Całość architektury jest optymalizowana za pomocą algorytmów uczenia ze wzmocnieniem, co umożliwia adaptację i naukę efektywnych strategii działania w dynamicznie zmieniającym się środowisku gry.

5 DDPG

Algorytm DDPG [1] jest metodą uczenia ze wzmocnieniem, która łączy idee z DPG (Deterministic Policy Gradient) i uczenia głębokiego. Jest ona szczególnie skuteczna w przestrzeniach ciągłych działań i została zaprojektowana do maksymalizacji funkcji wartości Q.

DDPG jest algorytmem typu *actor-critic*, który wykorzystuje dwie główne sieci: sieć agenta $\mu(s)$ do aproksymacji optymalnej polityki i sieć krytyka $Q(s, a)$ do oceny optymalnej wartości funkcji.

5.1 Opis algorytmu

Pierwszym krokiem jest inicjalizacja parametrów polityki i funkcji Q oraz stworzenie pustego bufora doświadczeń. Następnie, w każdej iteracji, agent obserwuje aktualny stan środowiska, wybiera i wykonuje akcję, po czym zapisuje wynik w buforze doświadczeń. Gdy nadejdzie czas na aktualizację, algorytm wybiera losowy zestaw doświadczeń z bufora i używa ich do obliczenia celu dla funkcji Q oraz do aktualizacji parametrów polityki i funkcji Q za pomocą gradientu wstępnego i gradientu prostego.

Proces aktualizacji obejmuje także aktualizacje sieci docelowych, które są wolniejszymi wersjami głównych sieci polityki i Q, aby zapewnić stabilność nauki. Algorytm kontynuuje proces uczenia do momentu osiągnięcia konwergencji, co oznacza, że polityka agenta i funkcja wartości Q stabilizują się i nie ulegają już dalszym znaczącym zmianom.

Sieć wykorzystywana jako polityka aktora μ została szczegółowo opisana w sekcji: Architektura sieci neuronowych. Sieć krytyka została rozszerzona o dodatkową warstwę z jednym neuronem wyjściowym, który zwraca wartość funkcji Q, co pozwala ocenić jakość i przewidywaną nagrodę dla danej akcji w danym stanie.

Algorithm 1 DDPG

Inicjuj początkowe parametry polityki θ, parametry funkcji Q ϕ, pusty bufor doświadczeń D
Ustaw parametry docelowe równe parametrom głównym: $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
    Obserwuj stan s i wybierz akcję $a = \text{clip}(\mu(s|\theta) + \epsilon, a_{\min}, a_{\max})$, gdzie $\epsilon \sim \mathcal{N}$
    Wykonaj akcję a w środowisku
    Obserwuj nowy stan s', nagrodę r, i sygnał zakończenia d wskazujący czy s' jest stanem końcowym
    Zapisz (s, a, r, s', d) w buforze doświadczeń D
    Jeśli s' jest stanem końcowym, zresetuj stan środowiska
    Jeśli nadszedł czas na aktualizację, to:
    for liczba aktualizacji **do**
        Losowo wybierz próbkę tranzycji $B = \{(s, a, r, s', d)\}$ z D
        Oblicz cele:
            $y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$
        Aktualizuj funkcję Q używając:
            $\nabla_\phi \frac{1}{|B|} \sum (Q_\phi(s, a) - y(r, s', d))^2$
        Aktualizuj politykę używając:
            $\nabla_\theta \frac{1}{|B|} \sum Q_\phi(s, \mu_\theta(s))$
        Aktualizuj sieci targetowe z:
            $\phi_{\text{targ}} \leftarrow \rho\phi + (1 - \rho)\phi_{\text{targ}}$
            $\theta_{\text{targ}} \leftarrow \rho\theta + (1 - \rho)\theta_{\text{targ}}$
    end for
until Konwergencja

5.2 Wyniki

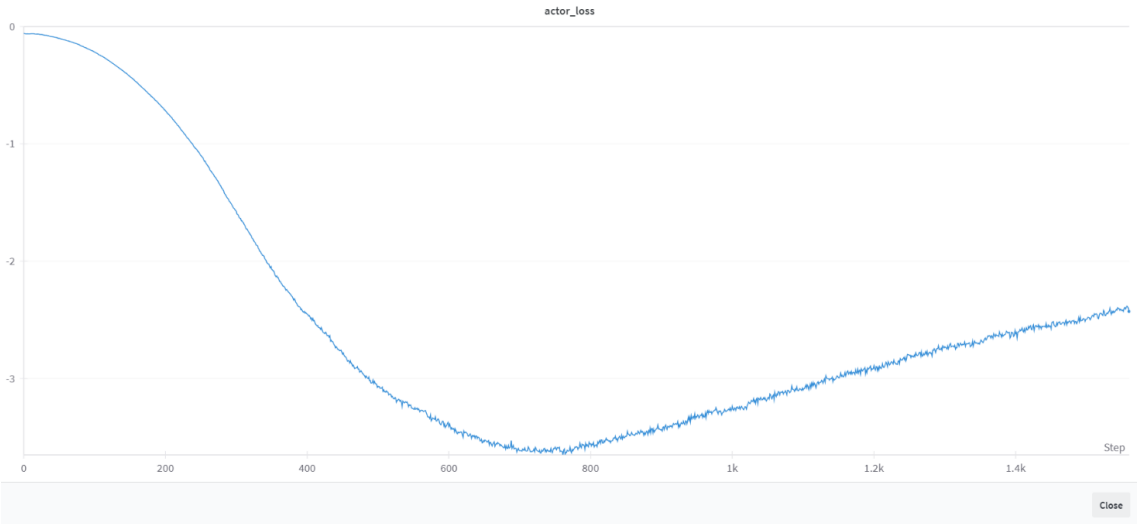
W trakcie przeprowadzonych eksperymentów z algorytmem DDPG, nie zaobserwowano zadowalających wyników. Algorytm szybko zaprzestał eksploracji przestrzeni akcji, wykazując tendencję do powtarzania tej samej, nieoptymalnej sekwencji działań. W konsekwencji, pojazd sterowany przez AI systematycznie zmierzał w prawą stronę toru, kierując się w stronę światła, aby następnie kontynuować jazdę wzdłuż ściany.

Modyfikacja parametru losowości ruchów, mająca na celu stopniowe zmniejszanie stopnia eksploracji w kolejnych epokach, nie przyniosła oczekiwanych efektów. W dalszym ciągu algorytm nie był w stanie nauczyć się efektywnego manewrowania pojazdem i osiągnąć lepszych wyników na torze wyścigowym.

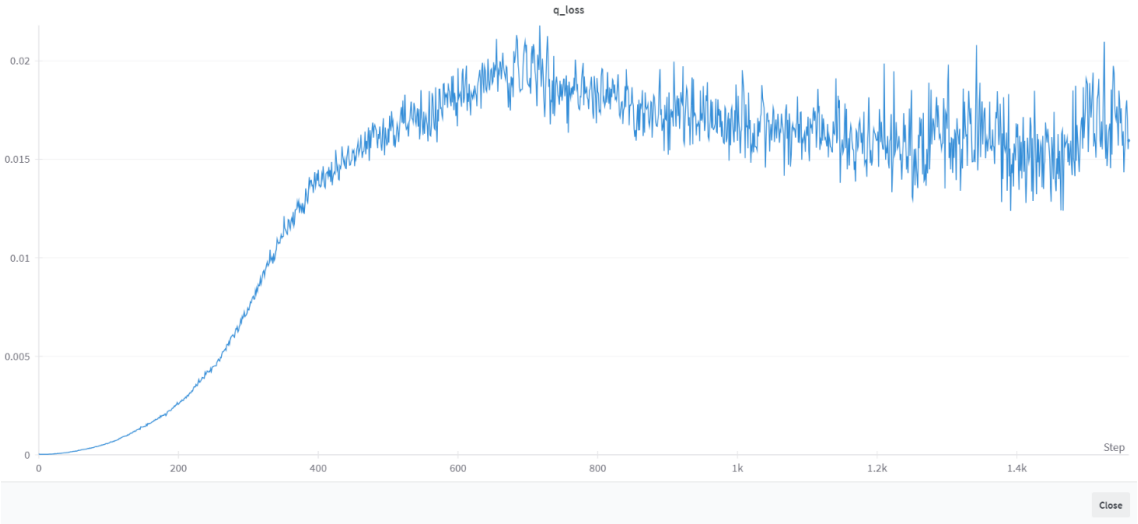
Rysunek 4: Aktor wykorzystujący algorytm DDPG.

Poniższe wykresy pokazują wartości funkcji straty aktora i krytyka, a także otrzymywane nagrodach w zależności od liczby iteracji. Funkcja straty aktora, wyrażająca aktualną wartość Q dla podjętej akcji, powinna być maksymalizowana – w praktyce oznacza to minimalizację negatywnej wartości funkcji Q. Funkcja straty krytyka odzwierciedla błąd średniokwadratowy pomiędzy przewidywaną wartością Q a wartością docelową, co jest wskaźnikiem dokładności oceny stanu przez krytyka.

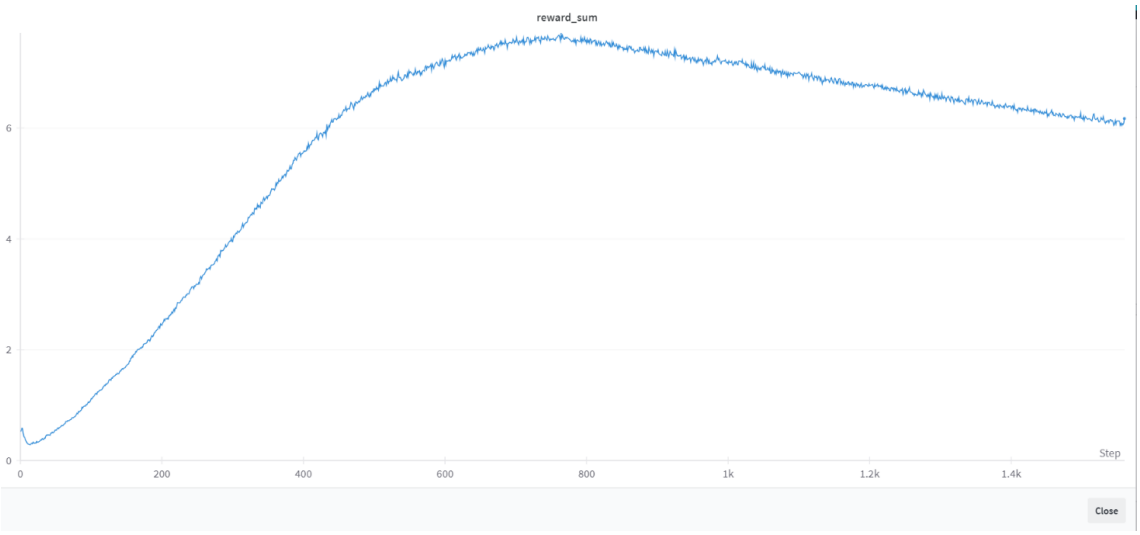
Z obserwacji wynika, że w okolicach 800 iteracji nastąpił nieoczekiwany wzrost `actor_loss` i równoczesny spadek otrzymywanych nagród. Korespondowało to z obserwowanym zachowaniem agenta w środowisku symulacyjnym, gdzie zaczął on konsekwentnie zbaczać w prawą stronę toru. Takie działanie, skutkujące kontaktem z murami toru, sugeruje problemy z procesem eksploracji.



Rysunek 5: Funkcja straty aktora DDPG.



Rysunek 6: Funkcja straty krytyka DDPG.



Rysunek 7: Nagroda DDPG.

6 SAC

Algorytm Soft Actor-Critic (SAC) [2] jest zaawansowaną metodą uczenia ze wzmocnieniem, która stosuje podejście aktor-krytyk z dodatkiem entropii do polityki, promując tym samym eksplorację. Algorytm ten został również szczegółowo opisany przez autorów `tmrl` [5].

6.1 Opis algorytmu

Inicjalizacja algorytmu rozpoczyna się od ustalenia parametrów polityki i funkcji wartości Q. W trakcie każdej iteracji algorytmu, agent obserwuje bieżący stan środowiska i podejmuje akcję zgodnie z polityką, która jest funkcją prawdopodobieństwa

zależną od stanu i parametrów polityki. Akcje te są następnie realizowane w środowisku, a wynikające z nich obserwacje, nagrody i ewentualne sygnały zakończenia epizodu są rejestrowane w buforze.

Kiedy nadchodzi czas na aktualizację, SAC wykorzystuje próbki z bufora do aktualizacji parametrów funkcji wartości Q oraz polityki. W przeciwieństwie do tradycyjnych metod, które dążą jedynie do maksymalizacji nagród, SAC dąży do maksymalizacji sumy nagród i entropii polityki, co zachęca do eksploracji i zapobiega przedwczesnej konwergencji do suboptymalnych polityk.

Aktualizacja sieci docelowej w SAC odbywa się poprzez śledzenie zwolnionych wersji parametrów sieci głównych, co ma na celu stabilizację procesu uczenia. Algorytm kontynuowany jest aż do osiągnięcia konwergencji, co oznacza stabilizację w ocenie wartości stanów przez funkcje Q oraz w generowaniu akcji przez politykę.

Polityka aktora μ jest opisana w sekcji: Architektura sieci neuronowych, natomiast funkcja Q krytyka zawiera dodatkową warstwę z jednym neuronem wyjściowym, który zwraca szacowaną wartość Q dla danego stanu i akcji.

Algorithm 2 Soft Actor-Critic

Zainicjuj początkowe parametry polityki θ, parametry funkcji Q ϕ_1, ϕ_2, pusty bufor doświadczeń $\mathcal{D}$
Ustaw parametry sieci docelowych równymi parametrom głównych sieci: $\phi_{\text{targ},1} \leftarrow \phi_1, \phi_{\text{targ},2} \leftarrow \phi_2$
repeat
    Obserwuj stan s i wybierz akcję $a \sim \pi_\theta(\cdot|s)$
    Wykonaj akcję a w środowisku
    Obserwuj następny stan s', nagrodę r oraz sygnał zakończenia d wskazujący, czy s' jest stanem końcowym
    Zapisz (s, a, r, s', d) w buforze doświadczeń $\mathcal{D}$
    Jeśli s' jest stanem końcowym, zresetuj stan środowiska
    Jeżeli nadszedł czas na aktualizację, wykonaj następujące kroki:
    for j w zakresie (ilość aktualizacji) **do**
        Losowo wybierz paczkę tranzycji $B = \{(s, a, r, s', d)\}$ z $\mathcal{D}$
        Oblicz cele dla funkcji Q:
            $y(r, s', d) = r + \gamma(1 - d)(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \tilde{\pi}_\theta(\tilde{a}'|s'))$
        Zaktualizuj funkcje Q:
            $\nabla_{\phi_i} \frac{1}{|B|} \sum (Q_{\phi_i}(s, a) - y(r, s', d))^2$ dla $i = 1, 2$
        Zaktualizuj politykę:
            $\nabla_\theta \frac{1}{|B|} \sum (\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}(s)) - \alpha \log \pi_\theta(\tilde{a}(s)|s))$
        Zaktualizuj sieci docelowe:
            $\phi_{\text{targ},i} \leftarrow \rho\phi_i + (1 - \rho)\phi_{\text{targ},i}$ dla $i = 1, 2$
    end for
until konwergencja

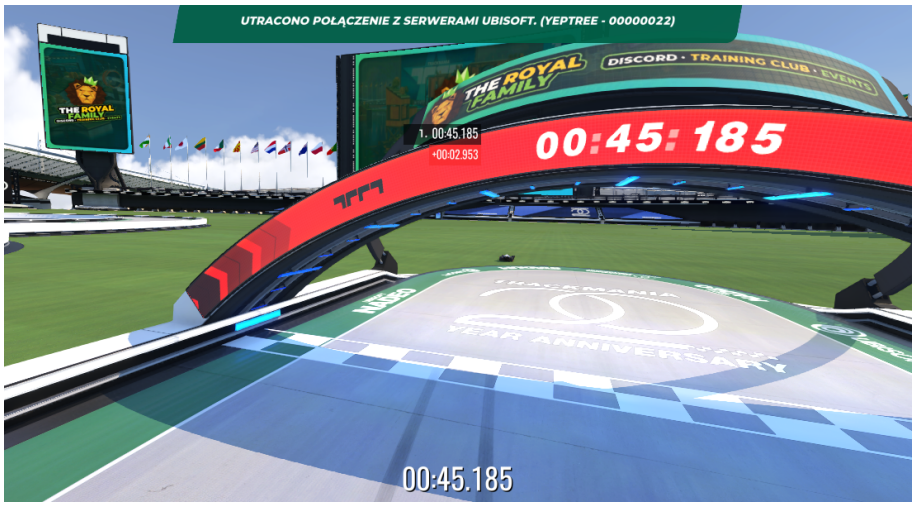
6.2 Wyniki

W ramach przeprowadzonych testów, algorytm SAC wykazał bardzo dobre wyniki. W przeciwieństwie do algorytmu DDPG, SAC wykazywał ciągłą zdolność do nauki i

adaptacji. Agent był w stanie efektywnie nauczyć się toru, demonstrując umiejętność skręcania i przyspieszania, a także unikania zderzeń z elementami otoczenia. Nie odnotowano przypadków, w których algorytm utknąłby w jakiejkolwiek części mapy.

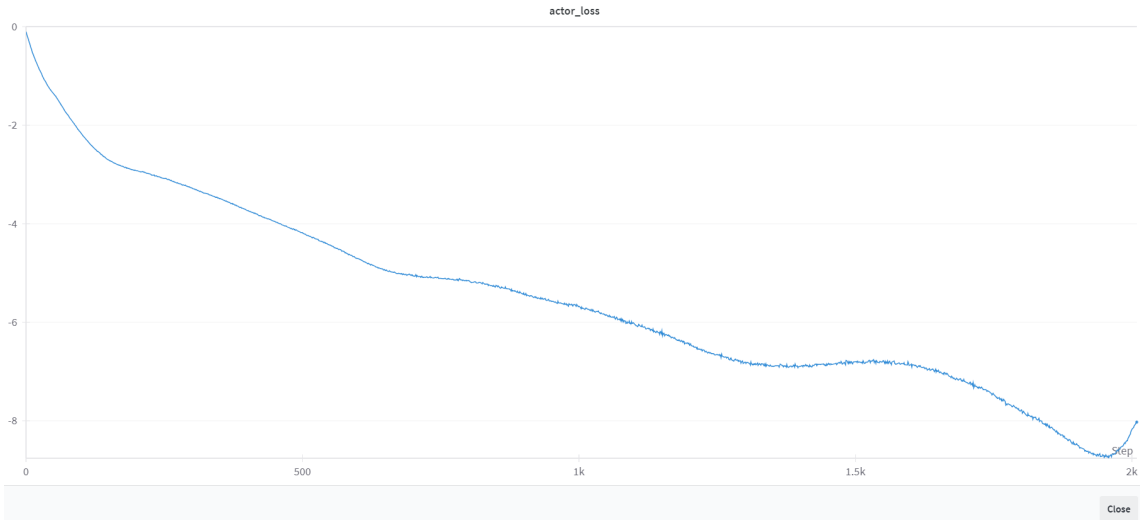
Znaczącym osiągnięciem SAC było ukończenie toru w czasie 45 sekund, co jest wynikiem porównywalnym do czasów osiąganych przez ludzkiego gracza, który poświęciłby znaczną ilość czasu na doskonalenie swoich umiejętności na danym torze. Taki wynik podkreśla potencjał algorytmu SAC w kontekście nauki złożonych zadań wymagających precyzji i adaptacji do dynamicznie zmieniających się warunków.

Pomimo osiągnięcia satysfakcjonujących wyników, istnieje przestrzeń na dalsze ulepszenia. Potencjalną ścieżką rozwoju może być modyfikacja funkcji celu, która mogłaby jeszcze bardziej zbliżyć wyniki doświadczonych graczy. Taka zmiana mogłaby pozwolić na jeszcze szybsze i bardziej efektywne przejazdy.

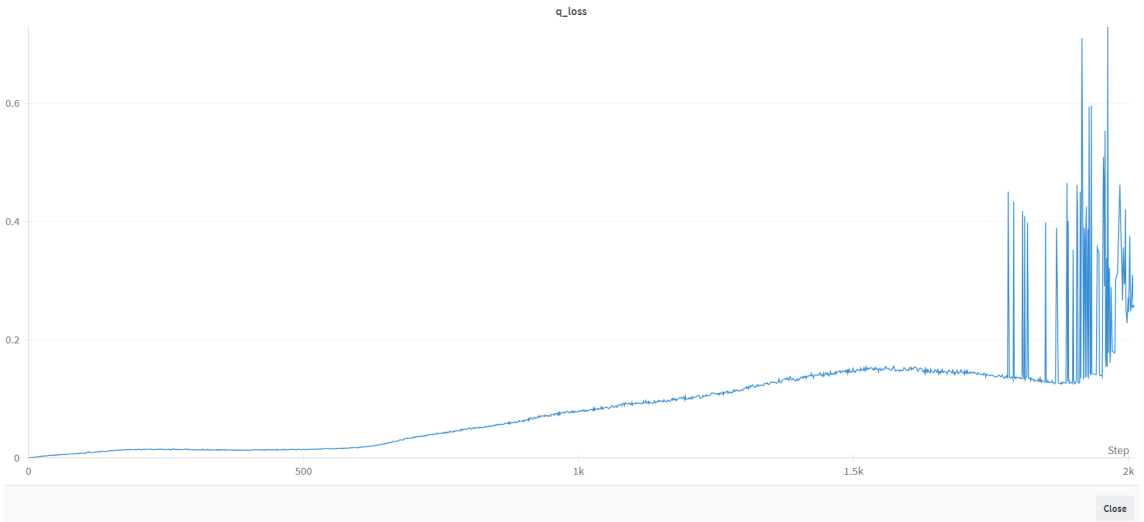


Rysunek 8: Aktor wykorzystujący algorytm SAC.

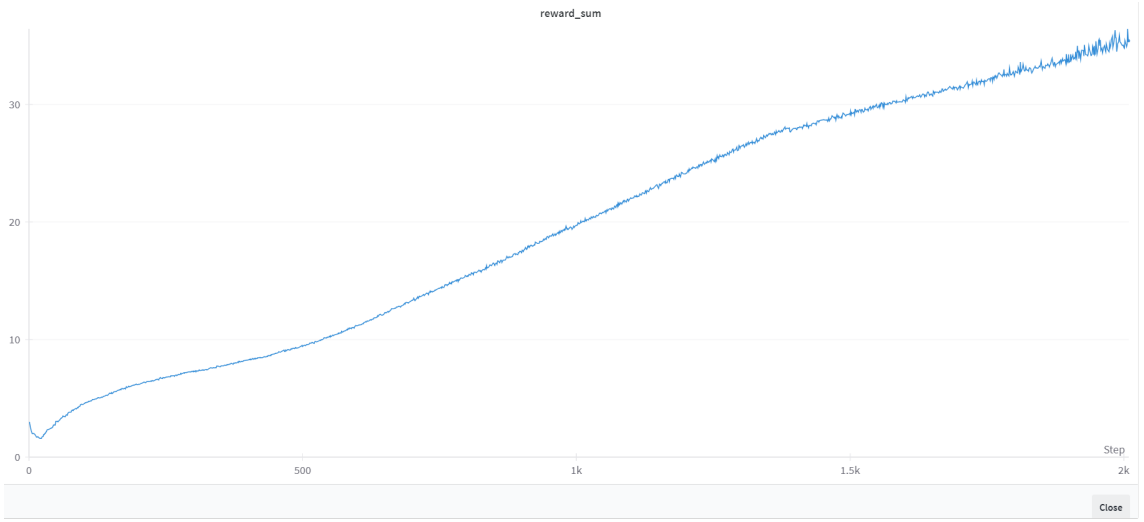
Analizując wykresy dotyczące procesu uczenia algorytmu Soft Actor-Critic (SAC), możemy zauważyć zachowanie wartości funkcji straty dla aktora oraz krytyka, a także trend w otrzymywanych nagrodach w zależności od liczby iteracji. Funkcja straty dla aktora, która jest odzwierciedleniem średniej różnicy wartości Q dla wybranej akcji oraz logarytmowi prawdopodobieństwa wybrania danej akcji dla stanu, powinna być minimalizowana. Z kolei funkcja straty dla krytyka odnosi się do błędu średniokwadratowego między estymowaną a docelową wartością Q.



Rysunek 9: Funkcja straty aktora SAC.



Rysunek 10: Funkcja straty krytyka SAC.



Rysunek 11: Nagroda SAC.

7 Wyzwania w trakcie implementacji

Podczas implementacji algorytmu Soft Actor-Critic (SAC) napotkaliśmy na szereg wyzwań, które znacząco wpłynęły na proces rozwoju projektu. Jednym z głównych problemów była kwestia odpowiedniego doboru parametrów algorytmu. Znalezienie optymalnych ustawień wymagało wielokrotnych prób i modyfikacji, co w połączeniu z długim czasem treningu stanowiło znaczącą przeszkodę.

Dodatkowo, ze względu na konieczność wykorzystania GPU w celu przyspieszenia obliczeń, napotkaliśmy trudności związane z debuggowaniem.

Wreszcie, każda wprowadzona zmiana wymagała przeprowadzenia długotrwałych sesji treningowych w celu oceny wpływu modyfikacji na zachowanie agenta. Takie podejście znacząco wydłużało czas potrzebny na iteracyjne ulepszanie algorytmu.

8 Wnioski

Podsumowując przeprowadzone prace nad implementacją algorytmów uczenia ze wzmocnieniem w grze Trackmania 2020, można wyciągnąć następujące wnioski:

Pierwszym i kluczowym wnioskiem jest wyższość algorytmu Soft Actor-Critic (SAC) nad algorytmem Deep Deterministic Policy Gradient (DDPG) w kontekście zadanego problemu. SAC wykazał się lepszą zdolnością do adaptacji, eksploracji i osiągnięcia wysokiej wydajności w dynamicznym środowisku gry. Agent SAC był w stanie nauczyć się toru, skutecznie skręcać i przyspieszać, co pozwoliło mu na osiągnięcie czasów przejazdu porównywalnych do doświadczonych graczy ludzkich.

Drugim wnioskiem jest stwierdzenie, że długi czas treningu i trudności związane z debuggowaniem na GPU stanowią istotne przeszkody w szybkim iteracyjnym rozwoju algorytmów. Wymaga to nie tylko cierpliwości i systematyczności w pracy nad projektem, ale także dostępu do odpowiednich zasobów obliczeniowych.

Wreszcie, projekt potwierdził, że uczenie ze wzmocnieniem może być skutecznie stosowane do złożonych zadań wymagających precyzyjnej kontroli i szybkiego podejmowania decyzji. Wyniki osiągnięte przez algorytm SAC otwierają perspektywy na jego zastosowanie nie tylko w grach, ale także w bardziej zaawansowanych aplikacjach robotyki i automatyzacji.

Literatura

[1] DDPG. https://spinningup.openai.com/en/latest/algorithms/ddpg.html.

[2] SAC. https://spinningup.openai.com/en/latest/algorithms/sac.html.

[3] tmrl. https://github.com/trackmania-rl/tmrl.

[4] Trackmania 2020. https://www.ubisoft.com/pl-pl/game/trackmania/trackmania.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[6] N. Mazyavkina, Samir Moustafa, Ilya Trofimov, and Evgeny Burnaev. *Optimizing the Neural Architecture of Reinforcement Learning Agents*, pages 591–606. 07 2021.