Super Mario Bros Raport

Grabias Wojtek, Kita Miłosz, Safiejko Natalia, Wnęk Kacper

Czerwiec 2024

1 Streszczenie

Celem niniejszego projektu było stworzenie agenta, który z wykorzystaniem technik uczenia ze wzmocnieniem (Reinforcement Learning) nauczyłby się przechodzić pierwszy poziom klasycznej gry "Super Mario Bros" (świat 1-1) w jak najkrótszym czasie. Proces obejmował tworzenie środowiska gry, projektowanie i implementację agenta, a także testowanie i optymalizację modelu.
Do zbudowania środowiska gry wykorzystano bibliotekę 'gym-super-mario-bros' [1], umożliwiającą interakcję agenta z wirtualnym światem gry. Projektowanie agenta polegało na opracowaniu modelu opartego na głębokiej sieci neuronowej, zdolnego do podejmowania decyzji na podstawie aktualnego stanu gry. Kluczową rolę w procesie uczenia odegrał algorytm Double Deep Q-Learning, który przy użyciu dwóch sieci neuronowych - głównej i docelowej - stabilizował proces uczenia się, minimalizując problem przeszacowania wartości Q.
Implementacja mechanizmu "epsilon-greedy" zapewniła balans między eksploracją nowych strategii a eksploatacją znanych, skutecznych metod. W trakcie uczenia modelu stosowano także mechanizm pamięci doświadczeń (experience replay), który umożliwiał przechowywanie i losowe wybieranie minipartii doświadczeń do treningu, co zwiększało efektywność procesu uczenia.
System nagród i kar został zaprojektowany tak, aby motywować agenta do szybkiego przechodzenia poziomu. Nagrody przyznawano za ruchy w prawo, unikanie i przeskakiwanie przeszkód oraz ukończenie poziomu, natomiast kary za stanie w miejscu oraz powolny postęp. Do testowania i optymalizacji modelu przeprowadzono serie eksperymentów, które pozwoliły na dostosowanie parametrów w celu uzyskania optymalnych wyników.
Aby przyspieszyć trenowanie modelu, obraz z gry został odpowiednio przetworzony: zmniejszono jego rozdzielczość, przekonwertowano do skali szarości, a także ograniczono dostępne ruchy agenta do prostych działań. W późniejszych etapach usunięto górną część obrazu, zawierającą informacje o pozostałym czasie, oraz dolną, nieistotną dla rozgrywki.
Podczas trenowania modelu kluczowe było dostosowanie wartości parametrów takich jak epsilon (decydujący o stopniu eksploracji), epsilon-decay (szybkość zmniejszania się epsilon), learning rate (szybkość uczenia się), oraz gamma (uwzględnianie przyszłych nagród). Wyniki eksperymentów wykazały, że najlepiej trenować model na tle bez chmur (v1), co minimalizowało wpływ zbędnych informacji na decyzje podejmowane przez agenta.
Ostatecznie, agent nauczył się przechodzenia poziomu 1-1 w sposób skuteczny, minimalizując czas potrzebny na jego ukończenie. Wyniki projektu pokazują, że współczesne techniki uczenia maszynowego mogą być z powodzeniem zastosowane w kontekście gier wideo, demonstrując możliwości optymalizacji i automatyzacji procesów decyzyjnych.

Spis treści

2 Wstęp

Celem niniejszego projektu jest stworzenie agenta, który przy użyciu technik uczenia ze wzmocnieniem (Reinforcement Learning) nauczy się przechodzić pierwszy poziom klasycznej gry "Super Mario Bros" (świat 1-1) w jak najkrótszym czasie. Całość technicznej implementacji poruszanych w raporcie rozważań zamieszczona jest w [2].

2.1 Cele i Zakres Pracy

W ramach pracy planowane jest:

1. Stworzenie środowiska gry:
   - Implementacja środowiska "Super Mario Bros" za pomocą biblioteki gym-super-mario-bros, umożliwiającego interakcję agenta ze światem gry w sposób kontrolowany.
2. Projektowanie i implementacja agenta:
   - Opracowanie modelu agenta opartego na głębokiej sieci neuronowej, który będzie podejmował decyzje na podstawie aktualnego stanu gry.
   - Implementacja dwóch sieci neuronowych (głównej i pomocniczej), które będą współpracować w celu stabilizacji procesu uczenia się.
3. Mechanizmy uczenia się:
   - Zastosowanie algorytmu Double Deep Q-Learning, który umożliwia agentowi uczenie się optymalnych akcji poprzez minimalizowanie funkcji straty pomiędzy przewidywanymi a rzeczywistymi wartościami Q, przy użyciu dwóch sieci neuronowych.
   - Wprowadzenie mechanizmu "epsilon-greedy" do balansowania eksploracji nowych akcji z eksploatacją znanych, skutecznych strategii.
4. Pamięć doświadczeń:
   - Implementacja pamięci doświadczeń (experience replay), która umożliwia agentowi przechowywanie i losowe wybieranie mini-partii doświadczeń do treningu, co poprawia stabilność i efektywność procesu uczenia się.
5. System nagród i kar:
   - Opracowanie systemu nagród, który motywuje agenta do podejmowania działań prowadzących do szybkiego przejścia poziomu. Nagrody będą przyznawane za ruchy w prawo, unikanie przeszkód, przeskakiwanie przeszkód oraz ukończenie poziomu.
   - Implementacja mechanizmu kar za stanie w miejscu oraz powolny postęp agenta w przechodzeniu poziomu.
6. Testowanie i optymalizacja:
   - Przeprowadzenie serii eksperymentów mających na celu ocenę efektywności różnych strategii uczenia się oraz dostosowanie parametrów modelu w celu osiągnięcia jak najlepszych wyników.

2.2 Realizacja

Projekt opiera się na paradygmacie uczenia ze wzmocnieniem, gdzie agent (Mario) eksploruje środowisko gry, podejmuje decyzje, otrzymuje nagrody lub kary, a następnie na tej podstawie aktualizuje swoje strategie. Cały proces uczenia się jest iteracyjny i opiera się na analizie zebranych doświadczeń oraz ciągłym doskonaleniu strategii decyzyjnych. W szczególności, używany jest algorytm Double Deep Q-Learning, który wykorzystuje dwie sieci neuronowe do stabilizacji procesu uczenia się, co pozwala na bardziej precyzyjne oszacowanie wartości Q i redukcję problemu z wysokimi wartościami Q.

2.2.1 Double Deep Q-Learning

Double Deep Q-Learning to ulepszona wersja algorytmu Deep Q-Learning, która redukuje problem przeszacowania wartości Q poprzez użycie dwóch sieci neuronowych: głównej (Q) i docelowej (Q'). Obie posiadają identyczną architekturę opartą o CNN (Convolutional Neural Network). CNN przetwarza obrazy, ucząc się istotnych cech. Akcje są wybierane przy użyciu polityki ϵ-greedy. Różnica pomiędzy sieciami to sposób aktualizacji wag - na początku uczenia obie sieci posiadają identyczne parametry, jednak z biegiem trenowania sieć główna (Q) aktualizowana jest na podstawie wartości obliczonych przez sieć docelową (Q'). W regularnych odstępach czasu parametry sieci głównej są kopiowane do sieci docelowej, co stabilizuje proces uczenia. [4]
ϵ-greedy to strategia wyboru akcji, która balansuje między eksploracją (szukaniem nowych rozwiązań) a eksploatacją (wykorzystaniem już znanych dobrych akcji):
Eksploracja (z prawdopodobieństwem ϵ) Z pewnym niewielkim prawdopodobieństwem (ϵ), agent wybiera losową akcję. Pozwala to na odkrywanie nowych stanów i strategii, które mogłyby być korzystniejsze niż te już znane.
Eksploatacja (z prawdopodobieństwem 1-ϵ) Z większym prawdopodobieństwem (1-ϵ), agent wybiera akcję, która ma najwyższą wartość Q w danym stanie. Wykorzystuje to dotychczas zdobytą wiedzę, aby maksymalizować natychmiastową nagrodę. Na przykład Jeśli $\epsilon = 0.1$, agent wybiera losową akcję w 10 % przypadków (eksploracja) i najlepszą znaną akcję w 90 % przypadków (eksploatacja). Z czasem wartość ϵ jest stopniowo zmniejszana, aby agent coraz bardziej polegał na swojej wiedzy, jednocześnie zachowując możliwość odkrywania nowych strategii.
Funkcja nagrody [1] zakłada, że celem gry jest poruszanie się jak najdalej w prawo (zwiększanie wartości x agenta) jak najszybciej, bez umierania. Aby modelować tę grę, nagroda składa się z trzech zmiennych:

1. v - różnica wartości x agenta między stanami, poruszanie się w prawo: v > 0
2. c - różnica w czasie gry między klatkami, upływ czasu: c<0
3. d - kara za umieranie w stanie, zachęca agenta do unikania śmierci, d=0 (żywy), d=-15 (martwy)

Całkowita nagroda:

$$r = v + c + d$$

2.2.2 Przygotowanie środowiska

W celu przyspieszenia trenowania modelu obraz z gry został odpowiednio przygotowany. Aby zredukować złożoność danych wejściowych przekazywanych do sieci zmniejszono jego rozdzielczość poprzez przeskalowanie oraz przekonwertowano do skali szarości(usunięcie informacji o kolorze). Aby uchwycić dynamikę ruchu nałożono na siebie cztery kolejne klatki widoku z gry. Pojedyncza klatka może nie zawierać wystarczającej informacji o kierunku ruchu lub prędkości obiektów. Dzięki nakładaniu kilku kolejnych klatek agent otrzymuje bardziej dokładną informację, co umożliwia lepsze podejmowanie decyzji i przewidywanie przyszłych stanów. Oprócz tego ograniczono dostępne ruchy do tych niezłożonych:

- Brak ruchu
- Prawo
- Prawo + A
- Prawo + B
- Prawo + A + B
- A
- Lewo

Dodatkowo w późniejszych etapach uczenia modelu pozbyto się górnej części obrazu, gdzie były informacje o pozostałym czasie, ponieważ zauważono zależność pomiędzy ruchami wykonywanymi przez Mario, a liczbami pokazywanymi na ekranie. Obraz został także ucięty od dołu, ponieważ nie zawierał informacji pomocnych w rozgrywce, a sprawiał, że dane wejściowe były bardziej złożone.

2.3 Metodologia

Przedstawione powyżej pojęcie zaimplementowaliśmy przy użyciu języka Python wraz z szeregiem zewnętrznych bibliotek. Szczegółowa techniczna realizacja projektu znajduje się w [2].
W ramach projektu podjęto próbę wytrenowania modelu agenta przejdzie dany poziom oraz zrobi to w jak najkrótszym czasie, w tym celu podjęto próbę dobrania parametrów uczenia w sposób umożliwiający realizację celu. Ze względu na liczbę możliwości kombinacji wszystkich zmiennych, ustaliliśmy niektóre z parametrów jako stałe:

- Wszystkie sesje trenowania modelu trwały 6000 epok,
- Proces uczenia zaczynano z $\epsilon = 1$ oraz gamma $= 0.9$,
- Learning rate ustalono na 0.0003.

Zmianom natomiast podlegały parametry:

- epsilon-decay,
- macierzowa interpretacja środowiska,
- funkcja nagrody.

Proces doboru hiperparametrów rozpoczęto dobierając arbitralnie zestaw zmiennych oparty na intuicji autorów. Na koniec wytrenowania modelu z danym zestawem, analizie poddawano zarówno zachowanie agenta w ramach heurystycznego dobierania ruchów, jak i dobierania ruchów na podstawie aktualnie wytrenowanego modelu. Analizowanymi parametrami były wartości funkcji straty, maksymalna odległość agenta od punktu początkowego, suma zdobytych nagród, liczba podjętych przez agenta kroków, czas trwania rozgrywki oraz liczba obiektywnie nieoptymalnych ruchów (ruch w lewo oraz brak ruchu). Przy analizie wyżej wymienionych metryk, sprawdzano każde kolejne 100 epok uczenia, następnie uśredniając ich wartości (Przykładem może być wartość funkcji straty; dla pierwszych 1000 epok wyróżniamy 10 wartości, każda z nich będąca średnią dla kolejno: średnia wartość dla pierwszych stu epok, średnia wartość dla epok pomiędzy 101, a 200 itd.).
W ramach oceny aktualnie wytrenowanego modelu na przestrzeni uczenia (ruchy bez losowości) agregacja nie miała miejsca - model ewaluowano tylko raz na 100 epok.
Po uzyskaniu wyniku modelu dla danego zestawu hiperparametrów, ulegał on ewaluacji pod względem zarówno metryk, jak i empirycznemu sprawdzeniu jakości modelu (sprawdzając wizualnie, jak dany model przechodzi ustalony poziom na różnych etapach uczenia). Na podstawie takiej oceny ustalano, czy dany parametr wpłynął pozytywnie na proces uczenia.
W ramach każdego etapu z opisywanego procesu sprawdzano zmianę jakości etapu uczenia dla każdego z trzech parametrów. Każdy etap składał się więc z wytrenowania trzech modeli, każdy z tylko jednym zmienionym parametrem względem etapu-rodzica. Jeżeli zmiana wpłynęła na trening pozytywnie, następny etap przyjmował parametr i trenował sieci z zaktualizowanym zestawem. Jeżeli zmiana wpłynęła negatywnie na jakość uczenia, parametr nie przechodził do następnego etapu i trenowanie zachodziło ze starą wartością parametru. Projekt objął trzy takie etapy.

2.4 Spodziewane rezultaty

Oczekujemy, że w wyniku przeprowadzonych prac agent nauczy się skutecznie przechodzić poziom 1-1 w grze "Super Mario Bros", minimalizując czas potrzebny na jego ukończenie. Projekt nie tylko demonstruje możliwości współczesnych technik uczenia maszynowego w kontekście gier wideo, ale również przyczynia się do lepszego zrozumienia mechanizmów optymalizacji i automatyzacji procesów decyzyjnych.

3 Krótki opis gry

Super Mario Bros to klasyczna gra platformowa wydana przez Nintendo w 1985 roku, która stała się ikoną i jednym z najbardziej rozpoznawalnych tytułów w historii gier wideo. Gra została zaprojektowana przez Shigeru Miyamoto i Takashi Tezuka, a jej głównym bohaterem jest Mario, włoski hydraulik, który musi uratować księżniczkę Peach porwaną przez złego Bowsera.

3.1 Mechanika gry

Gra *Super Mario Bros* składa się z ośmiu światów, z których każdy zawiera cztery poziomy. Gracz kontroluje Mario, poruszając się w prawo na przewijanym ekranie, skacząc po platformach, unikając lub eliminując wrogów, zbierając monety i przedmioty wzmacniające (*power-ups*), takie jak grzyby, które powiększają Mario, oraz kwiaty ognia, które dają mu możliwość strzelania kulami ognia. Celem każdego poziomu jest dotarcie do flagi na końcu, co pozwala przejść do kolejnego etapu.

3.2 Elementy gry

Gra Super Mario Bros jest grą złożoną, w której skład wchodzą liczne elementy oraz postacie.

3.2.1 Postacie

- **Mario**: Główny bohater gry, kontrolowany przez gracza.
- **Luigi**: Brat Mario, dostępny jako postać dla drugiego gracza w trybie multiplayer.
- **Księżniczka Peach**: Porwana przez Bowsera, jest celem misji ratunkowej Mario.
- **Bowser**: Główny antagonista gry, którego Mario musi pokonać, aby uratować księżniczkę Peach.

3.2.2 Wrogowie

- **Goomba**: Najbardziej podstawowy przeciwnik, który może być zniszczony poprzez skok na jego głowę.
- **Koopa Troopa**: Żółw, który po skoku na niego chowa się w skorupie i może być użyty jako broń.
- **Piranha Plant**: Roślina, która wychodzi z rur, próbując ugryźć Mario.
- **Lakitu**: Latający wróg rzucający kolczaste stwory na Mario.
- **Bowser**: Końcowy przeciwnik na końcu większości zamków.

3.2.3 Przedmioty

- **Monety**: Zbierane przez Mario, za każde 100 monet gracz dostaje dodatkowe życie.
- **Grzyb**: Powiększa Mario, dając mu dodatkowe umiejętności i jeden dodatkowy hit punkt.
- **Kwiat Ognia**: Daje Mario możliwość strzelania kulami ognia.
- **Gwiazda**: Czyni Mario niewrażliwym na wrogów przez krótki czas.

3.3 Poziomy i Światy

Każdy świat w grze składa się z czterech poziomów, które mogą obejmować różne środowiska, takie jak naziemne krainy, podziemne tunele, podwodne etapy oraz zamki pełne pułapek i lawy. Poziom kończy się, gdy Mario dotrze do flagi na końcu etapu lub pokona Bowsera w zamku.

- **Świat 1-1**: Pierwszy poziom gry, jeden z najbardziej ikonicznych poziomów w historii gier wideo. Stanowi on wprowadzenie do podstawowych mechanik gry.

3.4 Zakończenie

Gra kończy się, gdy Mario pokona Bowsera w ostatnim zamku i uratuje księżniczkę Peach. Gracz jest nagradzany sceną końcową, gdzie księżniczka dziękuje Mario za jego odwagę i pomoc.
Super Mario Bros jest nie tylko klasycznym tytułem, który ustanowił standardy dla przyszłych gier platformowych, ale także grą, która wciąż inspiruje i przyciąga graczy na całym świecie, zarówno weteranów, jak i nowych entuzjastów. W kontekście tego projektu, poziom 1-1 stanowi doskonały przykład do nauki i testowania technik uczenia ze wzmocnieniem, ze względu na swoją prostotę, rozpoznawalność i znaczenie historyczne.

4 Główne wyniki

Korzystając z wcześniej ustalonej metodologii, wytrenowaliśmy szereg agentów, analizując wpływ poszczególnych parametrów na końcowe rezultaty.

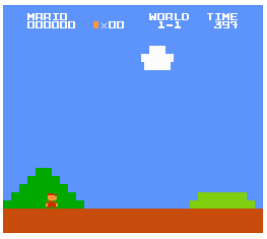
4.1 Opis parametrów

Aby uzyskać najlepsze wyniki, zmieniane były następujące parametry:

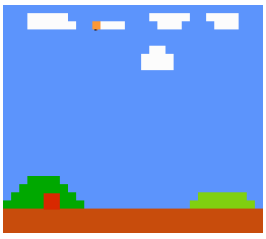
- epsilon - ϵ od którego zaczynano trenowanie, (jego znaczenie opisano w rozdziale 2.2.1),
- epsilon-decay - parametr decydujący o szybkości zmniejszania się ϵ. Im większy tym agent częściej wykonywał ruchy wynikające z trenowania, a rzadziej losowe,
- liczba epok - liczba podejść w danej sesji uczenia modelu (jedna epoka to jedno podejście do gry pamiętając o tym, że Mario ma 3 życia),
- tło - decydowało o tym jak szczegółowy obraz gry jest przekazywany do sieci,
- learning rate - parametr kontrolujący, jak bardzo aktualizowane są wagi sieci neuronowej w procesie uczenia. Im większa wartość learning rate, tym większe są kroki aktualizacji wag, co może prowadzić do szybszego, ale bardziej chaotycznego procesu uczenia. Z kolei zbyt mała wartość learning rate może prowadzić do bardzo powolnego uczenia się lub utknięcia w minimach lokalnych,
- gamma - parametr określający, jak bardzo agent ma uwzględniać przyszłe nagrody w procesie podejmowania decyzji. Jest to wartość między 0 a 1, gdzie 0 oznacza brak względu, a 1 oznacza pełne uwzględnienie przyszłych nagród. Wartość gamma wpływa na to, jak bardzo agent kieruje się przyszłymi korzyściami w swoich działaniach. Wysoka wartość gamma skupia się bardziej na długoterminowych korzyściach, podczas gdy niska wartość gamma bardziej koncentruje się na krótkoterminowych korzyściach.

Rysunek 1: tło v0

Rysunek 2: tło v1



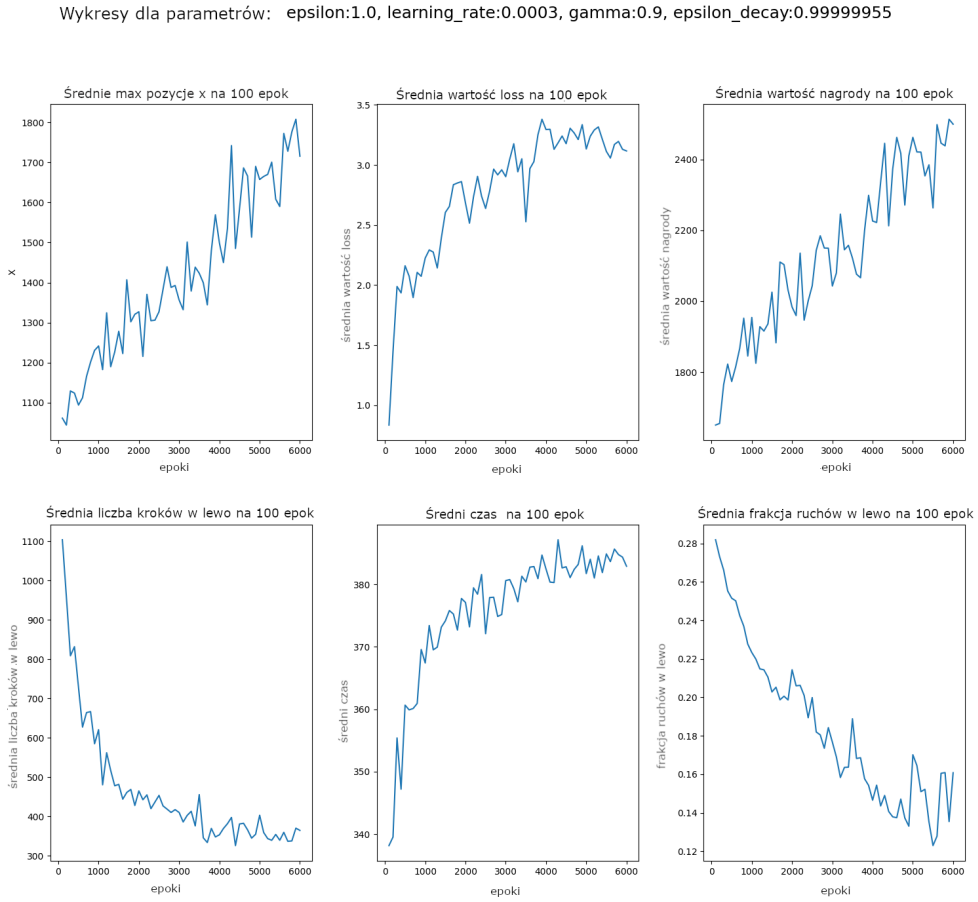
Rysunek 3: tło v2



Rysunek 4: tło v3

4.2 Wyniki

Po wstępnych próbach i obserwacjach zadecydowaliśmy, że najlepiej jest trenować model na tle v1 (omówione w 4.1) z uwagi na brak chmur, które mogły mieć wpływ na decyzje podejmowane przez agenta. Największym wyzwaniem okazało się wybranie odpowiedniej wartości parametru epsilon-decay. Ponieważ ϵ był zmniejszany po każdym wykonanym ruchu (step), a nie po epoce, trudno było oszacować to tak, aby przy ostatnich iteracjach treningu ϵ był odpowiedni. Wraz z kolejnymi iteracjami agent wykonywał coraz mniej ruchów (co można zauważyć na wykresie 5 "Average Number of Steps"), zatem wartość nie mogła być zbyt mała, aby końcowy ϵ nie był zbyt duży- wtedy decyzje dokonywane przez agenta byłyby zbyt losowe. Z drugiej strony, jeśli epsilon-decay byłby za mały spowodowałoby to z kolei zwiększenie liczby wykonywania losowych ruchów (zwiększenie liczby kroków, czyli szybsze zmniejszanie ϵ) przez co końcowy ϵ mógłby być za mały- co mogłoby skutkować sytuacjami, gdy Mario utykał w miejscu, bo z niskim prawdopodobieństwem wykonywał losowe ruchy, na przykład skok.
Przetestowaliśmy zatem kilka wartości epsilon-decay:

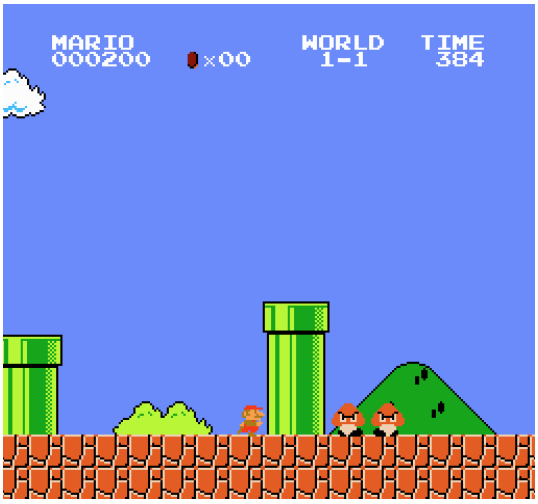


Rysunek 5: Wyniki trenowania modelu z epsilon = 1.0, learning rate = 0.0003, gamma = 0.9, epsilon-decay = 0.99999955 na kolejnych epokach.

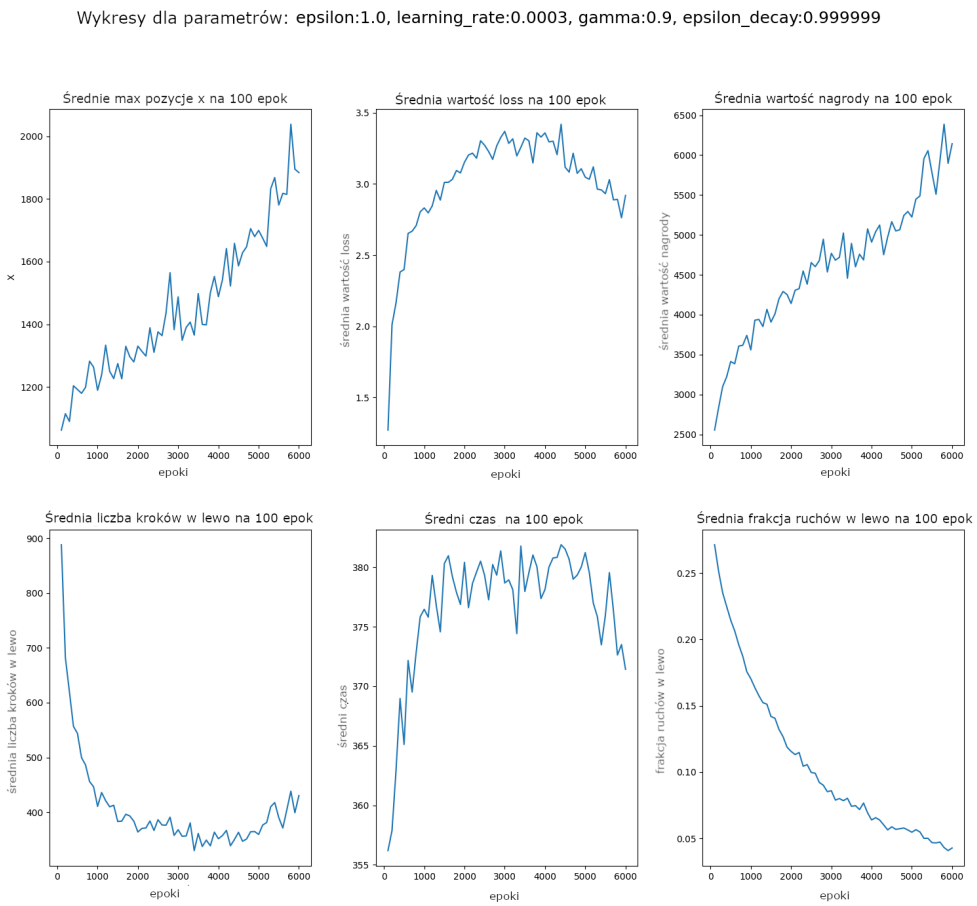
Można zauważyć wzrost wraz z kolejnymi epokami wartości average max x position, czyli Mario docierał coraz dalej, jednocześnie korzystając z coraz mniejszej liczby ruchów (Average Number of Steps) i średni czas na każde podejście także się zmniejszał, co można interpretować, jako coraz rzadsze występowanie sytuacji utknięcia przy przeszkodzie lub próby przejścia poziomu idąc w lewo (co dodatkowo potwierdza wykres "Average fraction of left movements").
Na podstawie analizy wyników stworzono usprawnienie polegające na dodatkowym nagradzaniu agenta, gdy pokona przeszkody, na których często utyka - w szczególności dotyczy to rur blokujących przejście,

które agent powinien przeskakiwać aby dalej przechodzić poziom. Ponadto był jeszcze bardziej karany za ruchy w lewo, aby zniechęcić go do ich wykonywania. Ponieważ końcowy epsilon okazał się za mały, zmniejszono wartość epsilon-decay.

Rysunek 6: Utknięcie na pierwszej rurze.

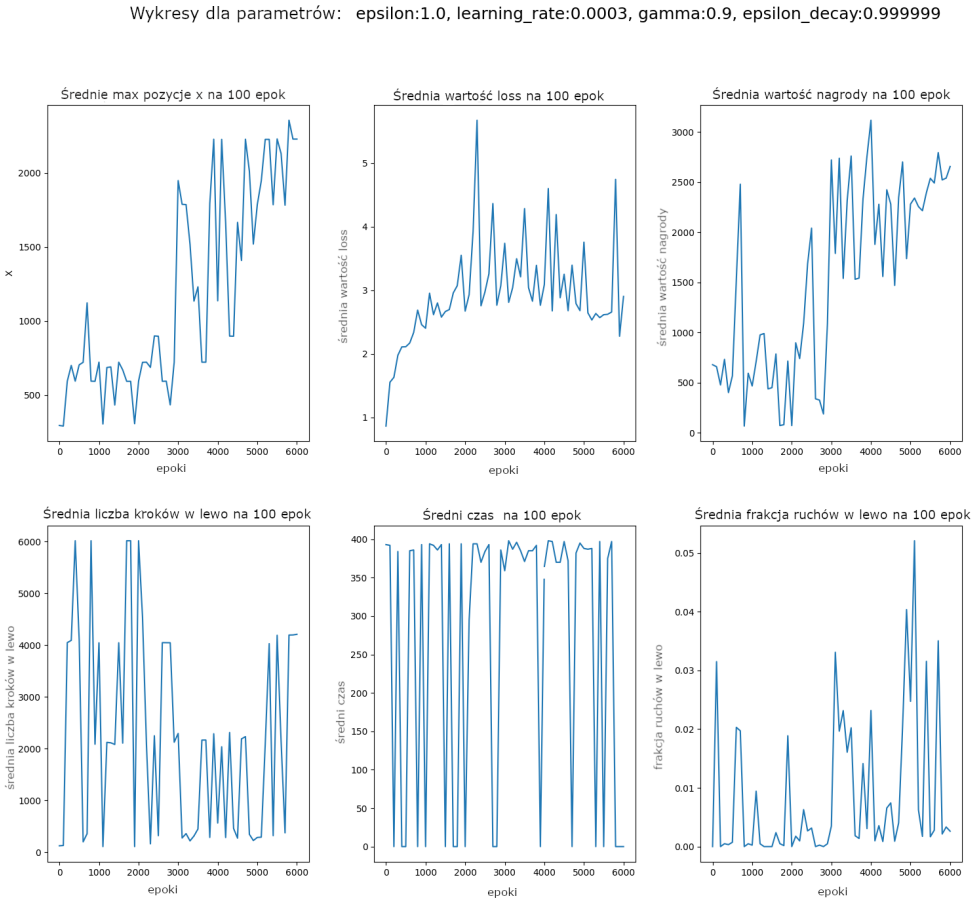


Rysunek 7: Utknięcie na drugiej rurze.



Rysunek 8: Wyniki trenowania modelu z epsilon = 1.0, learning rate = 0.0003, gamma = 0.9, epsilon-decay = 0.999999 na kolejnych epokach z uwzględnieniem dodatkowej nagrody.

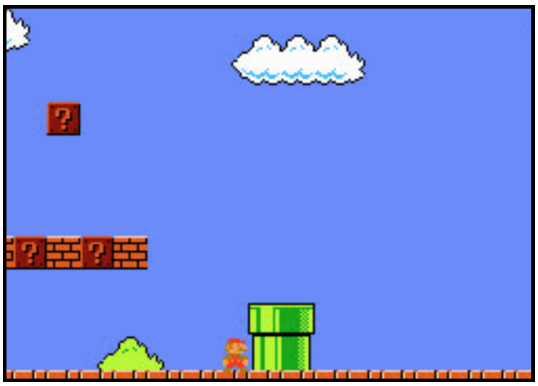
Na wykresie 8 można zauważyć lepsze wyniki- między innymi większe wartości average max x position. Dodatkowo wykresy są "gładsze" co może świadczyć o bardziej efektywnym uczeniu modelu.
Dodatkowo śledziliśmy wyniki nauczonego agenta w kolejnych epokach- wykres 9. Oznacza to, że w tym przypadku nie były wykonywane żadne losowe ruchy. Agent dokonywał decyzji na podstawie swojego doświadczenia i dostarczanego obrazu gry. Wykresy są bardziej chaotyczne, więc oprócz tego, że widać tendencje sugerujące polepszanie wyników modelu, niewiele wniosków można z nich wyciągnąć.
Ciekawą obserwacją jest, że nie zawsze najlepsze wyniki były osiągane na koniec uczenia - w ramach testowania modeli końcowych dla różnych parametrów zauważono, że ostatni model często próbował przechodzić grę szybciej, jednak zwykle kończyło się to śmiercią agenta przed ukończeniem poziomu lub utknięciem w miejscu. Model ulegał zjawisku przeuczania i czasami mimo przejścia całego poziomu, w następnej iteracji nie zdołał tego powtórzyć.



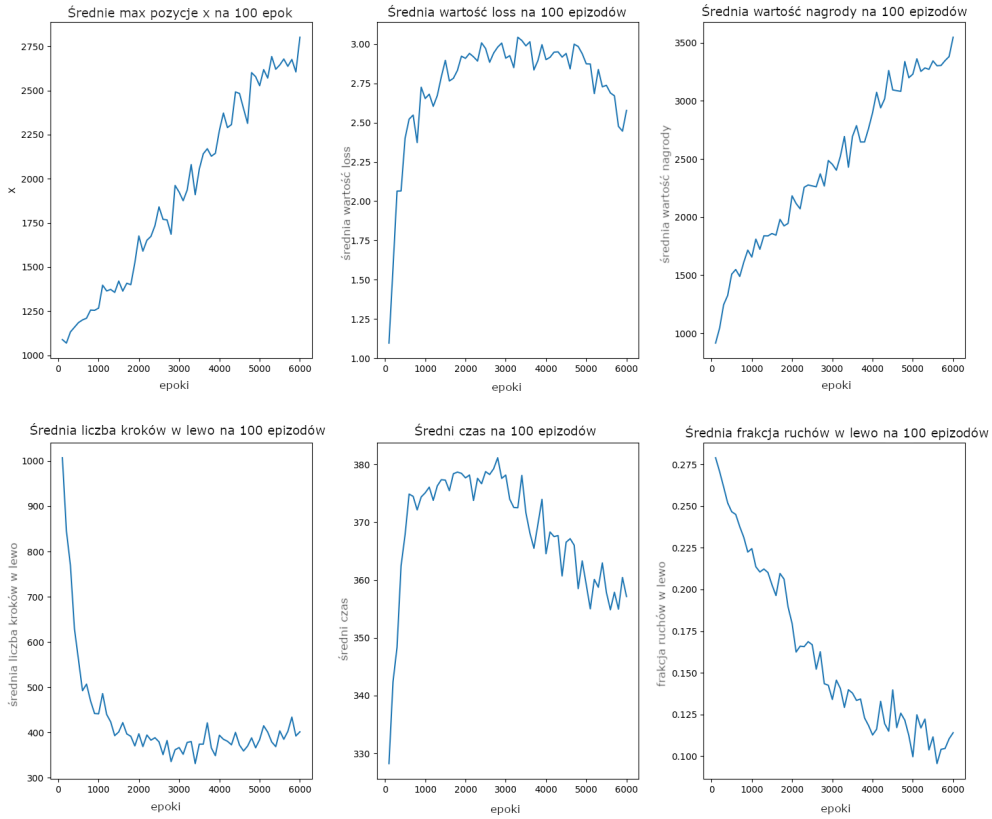
Rysunek 9: Wyniki wytrenowanego modelu z epsilon = 1.0, learning rate = 0.0003, gamma = 0.9, epsilon-decay = 0.999999 na kolejnych epokach z uwzględnieniem dodatkowej nagrody.

Ostatnim usprawnieniem, o którym wspomniano w rozdziale 2.2.2 było ograniczenie obrazu gry przekazywanego do sieci. Ucięty został widok informacji w górnej części ekranu oraz widok podłogi, przez co złożoność danych wejściowych była mniejsza a wyniki trenowania lepsze, co można zauważyć na wykresie 12.

Rysunek 10: Początkowy widok przekazywany do modelu.



Rysunek 11: Ograniczony widok przekazywany do modelu.



Rysunek 12: Wyniki trenowania modelu z epsilon = 1.0, learning rate = 0.0003, gamma = 0.9, epsilon-decay = 0.9999991 na kolejnych epokach z uwzględnieniem dodatkowej nagrody z ograniczonym widokiem.

Wyniki z tymi usprawnieniami były najlepsze z dotychczas otrzymanych. Mario przechodził poziom bardzo sprawnie unikając przeszkód i docierając do końca w dobrym czasie.

5 Podsumowanie i wnioski

Projekt polegający na stworzeniu agenta, który przy użyciu technik uczenia ze wzmocnieniem (Reinforcement Learning) nauczy się przechodzić pierwszy poziom klasycznej gry "Super Mario Bros" (świat 1-1), przyniósł wiele interesujących wyników. Agent został wytrenowany przy użyciu algorytmu Double Deep Q-Learning (DDQN). Dzięki odpowiednio zaprojektowanemu systemowi nagród i kar, agent nauczył się skutecznych strategii przechodzenia poziomu, minimalizując czas potrzebny na jego ukończenie.

5.1 Kluczowe wyniki

- **Implementacja środowiska gry:** Udało się stworzyć środowisko gry "Super Mario Bros" przy użyciu biblioteki gym-super-mario-bros, co umożliwiło kontrolowaną interakcję agenta z grą.
- **Projektowanie i implementacja agenta:** Opracowano model agenta oparty na głębokiej sieci neuronowej, który podejmował decyzje na podstawie aktualnego stanu gry. Wdrożono dwie sieci neuronowe, które współpracowały w celu stabilizacji procesu uczenia.
- **Mechanizmy uczenia się:** Zastosowano algorytm Double Deep Q-Learning, który minimalizował funkcję straty między przewidywanymi a rzeczywistymi wartościami Q, przy użyciu dwóch sieci neuronowych. Wprowadzono mechanizm "epsilon-greedy" do balansowania eksploracji nowych akcji z eksploatacją znanych strategii.
- **System nagród i kar:** Opracowano i wdrożono system nagród motywujący agenta do podejmowania działań prowadzących do szybkiego przejścia poziomu. Nagrody były przyznawane za ruchy w prawo, czas czy ukończenie poziomu. Wprowadzono również mechanizm kar za stanie w miejscu oraz powolny postęp agenta.
- **Optymalizacja przetwarzania obrazów:** Aby zredukować złożoność danych wejściowych, przeskalowano obrazy do niższej rozdzielczości oraz przekonwertowano je na skalę szarości, usuwając informacje o kolorze. Nałożono na siebie cztery kolejne klatki widoku z gry, co umożliwiło agentowi lepsze uchwycenie dynamiki ruchu. Usunięto górną część obrazu zawierającą informacje o czasie oraz dolną część, która nie zawierała ważnych informacji, co zmniejszyło złożoność danych wejściowych i poprawiło wyniki trenowania.

5.2 Wnioski

- **Skuteczność DDQN:** Algorytm Double Deep Q-Learning wykazał się dużą skutecznością w trenowaniu agenta do przechodzenia poziomu gry "Super Mario Bros". Dzięki zastosowaniu dwóch sieci neuronowych do estymacji wartości Q, algorytm znacznie zmniejszył ryzyko przeszacowania tych wartości, co pozwoliło na stabilniejszy i bardziej dokładny proces uczenia się. Algorytm ten efektywnie radził sobie z kompleksowością środowiska gry, prowadząc do wyraźnych postępów w umiejętnościach agenta.
- **Rola systemu nagród:** System nagród i kar miał kluczowe znaczenie dla motywowania agenta do optymalnych zachowań. Wprowadzenie nagród za ruch w prawo oraz kar za niepożądane działania (np. ruch w lewo) okazało się skuteczne w poprawie wyników agenta.
- **Znaczenie parametrów modelu:** Parametr epsilon-decay, który decyduje o szybkości zmniejszania się prawdopodobieństwa eksploracji, miał istotny wpływ na efektywność procesu uczenia się. Odpowiednie dostosowanie tego parametru pozwoliło na zachowanie balansu między eksploracją a eksploatacją, co przyczyniło się do osiągnięcia lepszych wyników w krótszym czasie.
- **Adaptacja wejściowych danych obrazowych:** Przetwarzanie obrazów wejściowych było kluczowym elementem w poprawie wydajności agenta. Skalowanie obrazów do niższej rozdzielczości, konwersja do skali szarości, usunięcie zbędnych elementów (takich jak tło, górna i dolna część ekranu) oraz użycie nałożonych na siebie klatek pozwoliło agentowi lepiej analizować otoczenie i podejmować bardziej trafne decyzje. Te zmiany zmniejszyły złożoność danych, co przyczyniło się do bardziej efektywnego uczenia się.

6 Dalsze możliwości rozwoju

Podczas trenowania naszego agenta za pomocą Double Deep Q-Learningu niekiedy natrafialiśmy na sytuację że agent po nauczeniu się odpowiedniej polityki w dalszej fazie treningu potrafił od niej odbiegać. Warto było by przetestować więć inny algorytm Reinforcement Learningowy, który bezpieczniejsze byłby w swoich aktualizacjach polityki.
Naszym celem było wytrenowanie agenta który w jak najszybszym czasie przechodzi poziom pierwszy. Nasz najlepszy uzyskany agent często skakał zbyt wysoko i w trakcie skoku blokował się o pojawiające się półki. Warto było by więc również przetestować inne funkcje nagrody aby jeszcze bardziej usprawnić jego przejścia.

6.1 Inne algorytmy Reinforcement Learningowe

6.1.1 SARSA

SARSA [3] jest algorytmem Reinforcement Learningu, który jest podobny do Q-Learning, ale różni się sposobem aktualizacji wartości Q. W SARSA wartość Q jest aktualizowana na podstawie rzeczywiście wybranej akcji w następnym stanie, a nie na podstawie maksymalnej możliwej akcji. SARSA jest bardziej "ostrożny" niż Q-Learning, ponieważ bierze pod uwagę politykę eksploracyjną podczas aktualizacji wartości Q.

Dlaczego warto przetestować SARSA:

- **Uwzględnienie rzeczywistej polityki eksploracyjnej:** SARSA aktualizuje wartości Q w oparciu o rzeczywiście wybrane akcje, co oznacza, że bierze pod uwagę bieżącą politykę eksploracyjną. Jest to szczególnie korzystne, gdy agent działa w środowisku, gdzie bezpieczeństwo i ostrożność są kluczowe, ponieważ aktualizacje są bardziej konserwatywne.
- **Stabilność w eksploracji:** Dzięki uwzględnianiu rzeczywistych akcji, SARSA może być bardziej stabilny w trakcie procesu eksploracji. W porównaniu do Q-Learningu, który aktualizuje wartości Q w oparciu o maksymalnie możliwe nagrody, SARSA jest mniej podatny na przeszacowanie wartości Q, co może prowadzić do bardziej stabilnych wyników. Faktem jest, że kwestia została częściowo rozwiązana poprzez zastosowanie Double Q-Learning'u, jednak zastosowanie algorytmu z natury cechującego się zachowawczością może wskazać nową ścieżkę w przebiegu trenowania agentów.
- **Ostrożność w niepewnych środowiskach:** SARSA jest bardziej odpowiedni w środowiskach, gdzie agent musi podejmować decyzje ostrożnie, co wynika z tego, że SARSA bierze pod uwagę politykę eksploracyjną.

6.2 Nowe funkcje nagrody

6.2.1 Wysokość skoku

Problemem niektórych wytrenowanych agentów jest ich wysokość skoku. Często aby przeskoczyć jakąś przeszkodę lub wroga skaczą oni zbyt wysoko przez co blokują się o przeszkody nad nimi (półki). Warto spróbować dodać do funkcji nagrody modyfikację która karała by naszego agenta za oddawanie zbyt wysokich skoków oraz nagradzała za skoki odpowiedniej wysokości czyli takie dzięki którym agent pokona przeszkodę przemieszczając się do przodu.

6.2.2 Czas przejścia

W naszym podejściu kluczową rolę odgrywa czas przejścia poziomu przez agenta. Aby polepszyć jego wyniki można dodać nową nagrodę po wstępnym wytrenowaniu agenta. Najpierw agent jest trenowany przy użyciu podstawowej funkcji nagrody, która może obejmować nagrody za ruch w prawo, unikanie przeszkód, pokonywanie przeciwników itp. Po wstępnym wytrenowaniu agenta można dodać dodatkową nagrodę, która będzie motywować agenta do ukończenia poziomu w jak najkrótszym czasie.

$$r_{time} = t_{max} - t$$

gdzie t to czas, w którym agent ukończył poziom, a t_{max} to całkowity czas na przejście poziomu.

Literatura

[1] Christian Kauten. Super Mario Bros for OpenAI Gym. GitHub, 2018.

[2] KW MK NS, WG. SuperMarioBrosRL. `https://github.com/WojtekGrbs/SuperMarioBrosRL`, 2024.

[3] SARSA:. https://www.javatpoint.com/sarsa-reinforcement-learning.

[4] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.