JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 20 (2021) 197–204
Received: 14 June 2021
Accepted: 28 June 2021

Analysis of the possibilities for using machine learning algorithms in the Unity environment

Analiza możliwości wykorzystania algorytmów uczenia maszynowego w środowisku Unity

Karina Litwynenko*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, ul. Nadbystrzycka 38, 20-618 Lublin, Poland

Abstract

Reinforcement learning algorithms are gaining popularity, and their advancement is made possible by the presence of tools to evaluate them. This paper concerns the applicability of machine learning algorithms on the Unity platform using the Unity ML-Agents Toolkit library. The purpose of the study was to compare two algorithms: Proximal Policy Optimization and Soft Actor-Critic. The possibility of improving the learning results by combining these algorithms with Generative Adversarial Imitation Learning was also verified. The results of the study showed that the PPO algorithm can perform better in uncomplicated environments with non-immediate rewards, while the additional use of GAIL can improve learning performance.

Keywords: reinforcement learning; imitation learning; Unity

Streszczenie

Algorytmy uczenia ze wzmocnieniem zyskują coraz większą popularność, a ich rozwój jest możliwy dzięki istnieniu narzędzi umożliwiających ich badanie. Niniejszy artykuł dotyczy możliwości zastosowania algorytmów uczenia maszynowego na platformie Unity wykorzystującej bibliotekę Unity ML-Agents Toolkit. Celem badania było porównanie dwóch algorytmów: Proximal Policy Optimization oraz Soft Actor-Critic. Zweryfikowano również możliwość poprawy wyników uczenia poprzez łączenie tych algorytmów z metodą uczenia przez naśladowanie Generative Adversarial Imitation Learning. Wyniki badania wykazały, że algorytm PPO może sprawdzić się lepiej w nieskomplikowanych środowiskach o nienatychmiastowym charakterze nagród, zaś dodatkowe zastosowanie GAIL może wpłynąć na poprawę skuteczności uczenia.

Słowa kluczowe: uczenie ze wzmocnieniem; uczenie przez naśladowanie; Unity

*Corresponding author
Email address: karina.litwynenko@pollub.edu.pl (K. Litwynenko)

1. Wstęp

Na zaobserwowany w ostatnim czasie duży postęp w dziedzinie algorytmów uczenia ze wzmocnieniem wpływ miało istnienie odpowiednich do badań nad nimi narzędzi [1]. Zapewniają one dostęp do środowisk o coraz większym stopniu złożoności, jak i o bardziej realistycznej grafice, czy też dokładniejszej fizyce. Są też one dla badaczy głównym narzędziem umożliwiającym projektowanie, a także testowanie algorytmów. Z tego powodu, jakość oraz możliwości tych platform są bardzo często istotnym czynnikiem postępu w dziedzinie uczenia ze wzmocnieniem. Niniejszy artykuł dotyczy platformy Unity, przedstawiając jego możliwości w kontekście badań nad algorytmami uczenia ze wzmocnieniem. Unity to narzędzie pozwalające na tworzenie gier i symulacji zawierających dokładną fizykę, a także realistyczną oprawę graficzną. Platforma ta pozwala na tworzenie rozgrywki posługując się w dużej mierze wygodnym, graficznym interfejsem użytkownika. Połączenie jej z otwartoźródłową biblioteką Unity ML-Agents Toolkit sprawia, że staje się ona rozbudowanym i elastycznym narzędziem do pracy z algorytmami uczenia maszynowego. W ramach pakietu ML-Agents Toolkit dostępne są dwa algorytmy uczenia ze wzmocnieniem — Proximal Policy Optimization (PPO) [2] oraz Soft Actor-Critic (SAC) [3]. Narzędzie wspiera również uczenie przez naśladowanie, a wśród udostępnianych algorytmów można wskazać Generative Adversarial Imitation Learning (GAIL) [4,5].

Pomimo że początki algorytmów uczenia ze wzmocnieniem sięgają już lat 50. [6], to właśnie w ciągu ostatnich kilku lat zaobserwowano znaczny postęp w tej dziedzinie. Przejawia się on w szczególności powstawaniem nowych algorytmów, takich jak PPO oraz SAC. Pierwszy z nich powstał na podstawie równie niedawno przedstawionego algorytmu Trust Region Policy Optimiza-

tion (TRPO) [7]. Metoda PPO okazała się nie tylko wydajniejsza od swojego poprzednika, ale też o wiele prostsza w implementacji [2]. Z kolei algorytm SAC został porównany z kilkoma innymi współczesnymi algorytmami uczenia ze wzmocnieniem, w tym PPO. W rezultacie, SAC wykazał się wyższą wydajnością oraz szybkością uczenia w stosunku do pozostałych badanych metod [3]. Dostępne są też wyniki uczenia obu algorytmów PPO i SAC na platformie Unity. Obie metody zastosowano w siedemnastu przykładowych środowiskach wchodzących w skład biblioteki ML-Agents Toolkit [1]. Nie podjęto jednak próby analizy tych rezultatów. Unity wykorzystywano również do treningu agentów w innych, indywidualnych projektach środowisk [8]. Jednak nie zostały one szczegółowo opisane, ani nie wskazano zastosowanych algorytmów uczenia. Platforma Unity oraz biblioteka Unity ML-Agents Toolkit posłużyła również do badania skuteczności sposobów definiowania przestrzeni akcji agenta [9]. Wśród narzędzi pozwalających na testowanie algorytmów uczenia ze wzmocnieniem w symulowanych środowiskach, obok Unity, można wyróżnić platformę Arcade Learning Environment [7, 10–13] oraz OpenAI Gym [14]. Algorytm uczenia przez naśladowanie GAIL okazał się skuteczniejszy od kilku innych porównywanych metod, takich jak Behavioral Cloning (BC), Feature Expectation Matching (FEM), czy Game-Theoretic Apprenticeship Learning (GTAL) [4]. Jako podsumowanie aktualnego stanu wiedzy warto zwrócić uwagę na to, że dotychczas wykonane porównania PPO z SAC opierały się na eksperymentach przeprowadzonych w predefiniowanych środowiskach, bez możliwości wprowadzania w nich zmian. Trudno jest odnaleźć prace dotyczące zależności pomiędzy specyfiką środowiska agenta, a skutecznością algorytmów PPO oraz SAC. Brakuje też badań nad możliwościami łączenia algorytmów uczenia ze wzmocnieniem z uczeniem przez naśladowanie.

Celem przedstawionego w niniejszej pracy badania było porównanie algorytmów uczenia maszynowego udostępnianych w ramach biblioteki Unity ML-Agents Toolkit, współpracującej z platformą Unity. Badanie skupiło się na dwóch współczesnych algorytmach uczenia ze wzmocnieniem — PPO oraz SAC. Weryfikacji poddano również hipotezę o możliwości poprawy skuteczności uczenia poprzez łączenie tych algorytmów z metodą GAIL. Uczenie z wykorzystaniem badanych metod przeprowadzono w środowisku zaprojektowanym za pomocą platformy Unity.

Kolejny rozdział artykułu zawiera opis metody badań. Dalsza część przedstawia projekt wykonanego w Unity środowiska agentów sztucznej inteligencji. Ostatnie dwa rozdziały zostały poświęcone na wyniki badania, a także wnioski, w których zawarto krótkie podsumowanie pracy oraz wynikłe konkluzje. W końcowej treści wskazano również możliwe dalsze kierunki prac.

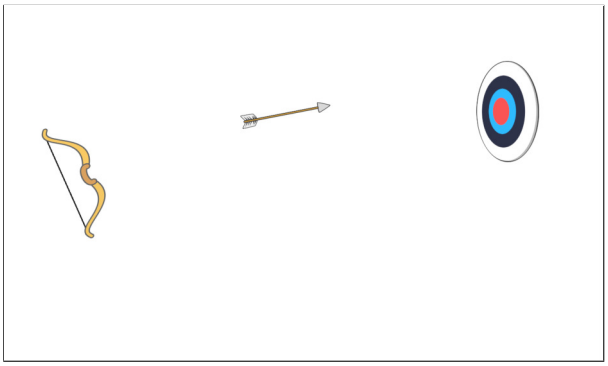
2. Metoda badań

Porównanie algorytmów zostało przeprowadzone na podstawie rezultatów uczenia agentów w grze 2D zaprojektowanej za pomocą platformy Unity wykorzystującej bibliotekę ML-Agents Toolkit. Przygotowana gra składała się z trzech wariantów, które charakteryzowały się innym poziomem trudności rozgrywki. Głównym zadaniem agenta było strzelanie z łuku do tarczy treningowej, zdobywając przy tym jak największą sumę punktów (Rysunek 1). Poszczególne warianty środowiska można scharakteryzować następująco:

1. Siła oddawanego strzału była zawsze stałą, ponadto tarcza nigdy nie zmieniała swojego położenia.
2. Siła strzału była stałą, jak w przypadku wariantu pierwszego, jednak po każdym celnym strzale tarcza zmieniała swoją pozycję, pojawiając się losowo w jednym z 25 możliwych miejsc.
3. Siła strzału była nadawana przez agenta, kontrolującego naciągnięcie cięciwy łuku. Tarcza zmieniała też swoje położenie, jak w wariancie drugim.

Algorytmy zostały więc porównane na podstawie rezultatów eksperymentu, zakładającego zwiększanie złożoności środowiska agenta, rozpoczynając od zadania najbardziej podstawowego. Każda z wersji przygotowanego środowiska charakteryzowała się również tym, że zdobywane przez agenta nagrody nie były nigdy natychmiastowe, a wymagały wykonania korzystnej sekwencji kroków. Porównanie opierało się na wynikach przebiegu uczenia, jak również nauczonego przez agentów zachowania. Wytrenowane modele sprawdzono w rozgrywce uwzględniającej liczbę zdobytych punktów, celność (stosunek strzałów trafionych w cel do wszystkich oddanych), a także czas ukończenia zadania przy ograniczonym zasobie strzał.

Możliwości łączenia algorytmów PPO i SAC z GAIL w celu poprawy wyników uczenia zbadano w trzecim, najbardziej złożonym wariancie środowiska. Zastosowane demonstracje pochodziły z rozgrywki człowieka, trwającej 8 i 48 minut. Rezultaty uczenia łączonych metod zostały porównane w ten sam sposób, co samodzielne algorytmy PPO oraz SAC — na podstawie przebiegu uczenia i rozgrywki wytrenowanych agentów.



Rysunek 1: Widok z rozgrywki przygotowanej na potrzeby badania gry.

3. Projekt środowiska

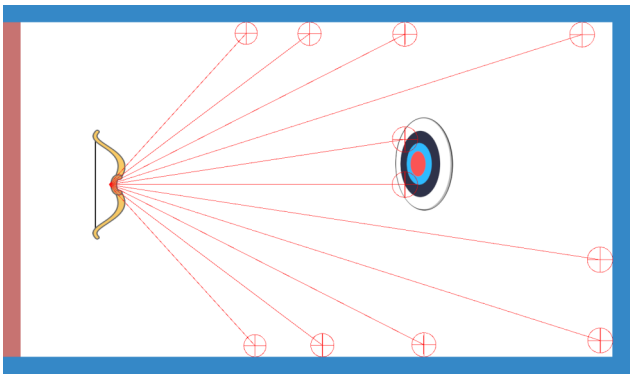
Zadaniem agenta w przygotowanym na potrzeby eksperymentu środowisku jest strzelanie z łuku do tarczy treningowej. Każdy celny strzał jest nagradzany odpowiednią liczbą punktów, zależną od koloru trafionego okręgu tarczy — od 0,7 do 1 (Rysunek 2). Sterowanie dla gracza jest realizowane za pomocą klawiatury. Użycie strzałki lewej lub prawej obraca łuk, zaś sterowanie cięciwą odbywa się za pomocą spacji. W przypadku dwóch prostszych wariantów gry do oddania strzału wystarczy pojedyncze wciśnięcie tego klawisza, zaś w najtrudniejszym trybie rozgrywki możliwe jest jego przytrzymanie w celu nadania większej siły wystrzału.

Rysunek 2: Punktacja dla gracza, a także wartości wzmocnienia zdobywane przez agenta w zależności od trafionego okręgu tarczy.

Obserwacje agenta, zbierane w każdym kroku uczenia, składają się z następujących elementów:

- gotowość łuku do oddania strzału (1 dla prawdy, 0 dla fałszu),
- kąt obrotu łuku,
- wartości zarejestrowane przez 10 promieni komponentu *RayPerceptionSensor3D*, wykrywające tarczę oraz ściany otaczające agenta (Rysunek 3),
- wartość naciągu cięciwy (tylko w przypadku 3. wariantu środowiska).

Ponadto obserwacje te są kumulowane dwukrotnie, co oznacza, że agent podejmuje decyzje na podstawie bieżącego oraz poprzedniego stanu środowiska.



Rysunek 3: Rozmieszczenie promieni komponentu *RayPerceptionSensor3D* z widocznymi ścianami otaczającymi agenta.

Akcje agenta zaimplementowano jako dyskretne, stosując gałęzie akcji (z ang. *action branching*), które pozwalają uprościć przestrzeń akcji, rozdzielając ją na grupy, z których agent może wybierać wartości jednocześnie [9,15]. Ponadto akcje podejmowane są co 5 kroków środowiska i są one automatycznie powtarzane pomiędzy kolejnymi decyzjami. Maksymalna długość epizodu została ustawiona na 1000 kroków.

Środowisko przewiduje następujące wartości wzmocnienia:

- od 0,7 do 1 w zależności od koloru trafionego okręgu tarczy (Rysunek 2),
- - 0,01 za trafienie w ścianę za agentem (czerwona na rysunku 3)
- - 0,001 za trafienie w ścianę boczną lub przednią,
- - 0,001 za zwłokę, gdy została podjęta akcja inna niż oddanie strzału, gdy łuk jest w stanie gotowości lub napięcie cięciwy osiągnęło maksymalny poziom.

Przy projektowaniu systemu nagród, zdecydowano, aby nie zniechęcać agenta do prób oddawania strzałów. Dlatego zastosowano niewielkie wartości kar za trafianie w otaczające ściany. Należy też dodać, że agent nie otrzymuje przedstawionych nagród i kar natychmiastowo. Co oznacza, że nie następuje to od razu po podjęciu korzystnej akcji, lecz dopiero po wielu kolejnych krokach, w których również podejmowane są decyzje. Najdłuższa sekwencja sprzyjających działań agenta wymagana jest w najtrudniejszym wariancie środowiska, co wynika z dodatkowej konieczności naciągnięcia cięciwy łuku w celu oddania strzału.

Dobór hiperparametrów uczenia został przeprowadzony w sposób eksperymentalny, ręcznie regulując ich wartości i obserwując rezultaty uczenia. W badaniu wykorzystano następnie te parametry, dla których uczenia osiągnęło najwyższy wynik skumulowanej wartości nagrody, przy zachowaniu stabilności uczenia. W celu oceny skuteczności testowanych hiperparametrów wykorzystano dodatkowo funkcję strat polityki, która pozwoliła określić jak bardzo w trakcie uczenia zmieniała się polityka agenta, a także wyznaczyć odpowiedni moment zakończenia treningu. Maksymalną liczbę kroków uczenia ustalono na podstawie najmniejszej wartości wspólnej dla obu algorytmów, dla której nastąpiło ustalenie się maksymalnej skumulowanej wartości nagrody. Zakres testowanych wartości hiperparametrów dostosowano na podstawie zaleceń umieszczonych w dokumentacji projektu ML-Agents Toolkit [16]. Najistotniejsze hiperparametry zastosowane w badaniu zebrano w tabelach 1 (PPO) oraz 2 (SAC). Ich nazwy pozostawiono zgodnie z definicją pliku konfiguracyjnego *yaml*. W przypadku algorytmu GAIL zastosowano stałą uczenia (parametr *learning_rate*) równą 0,0006. Wagę sygnału wzmocnienia (parametr *strength*) ustawiono na 0,01, co pozwoliło zapobiec bezpośredniemu kopiowaniu przez agenta zachowania przedstawionego w demonstracjach.

Trening w przypadku każdego wariantu gry uruchomiono na dziesięciu instancjach środowiska. Zrealizowa-

Tabela 1: Hiperparametry uczenia zastosowane dla algorytmu PPO

	Wariant środowiska		
	1.	2.	3.
learning_rate	0,001	0,0006	
batch_size	32		
buffer_size	5000		12000
beta	0,005		
epsilon	0,2		
lambd	0,95		
num_epoch	3		
normalize	true		
hidden_units	32	64	128
num_layers	2		
max_steps	504000	900000	1008000

Tabela 2: Hiperparametry uczenia zastosowane dla algorytmu SAC

	Wariant środowiska		
	1.	2.	3.
learning_rate	0,0008	0,0006	
batch_size	32		
buffer_size	50000		100000
tau	0,005		
steps_per_update	10		
init_entcoef	0,5		
normalize:	true		
hidden_units	32	64	256
num_layers	2		
max_steps	504000	900000	1008000

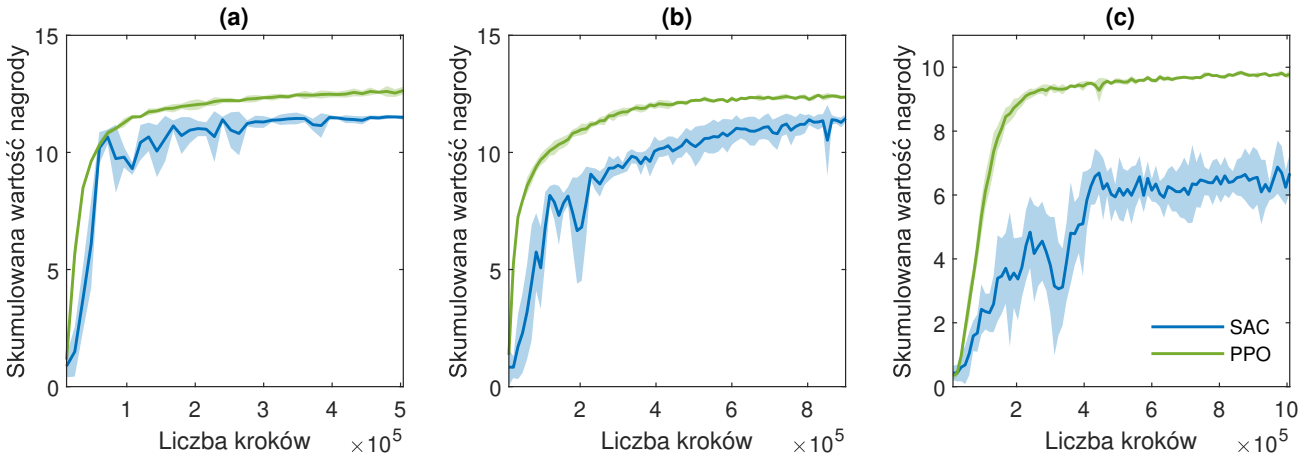
no to poprzez powielenie obiektu agenta oraz środowiska w pojedynczej scenie. Każda z instancji agenta podejmuje niezależne decyzje na podstawie lokalnych obserwacji, ale wykorzystuje i przyczynia się do aktualizowania wspólnej polityki. Zabieg ten wpływa na szybsze zapełnianie bufora doświadczeń, a tym samym na skrócenie czasu oczekiwania na rezultat uczenia.

4. Wyniki

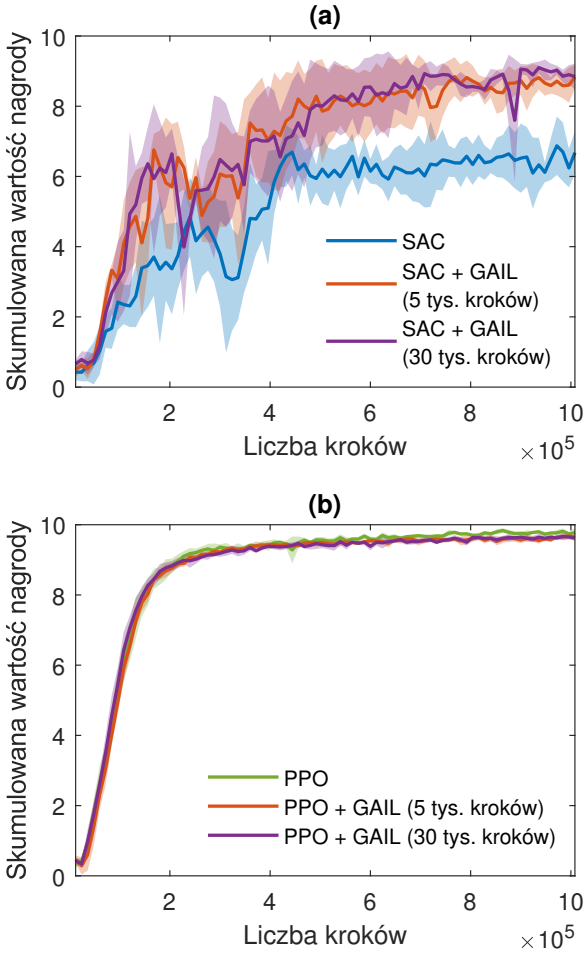
Wyniki badań przedstawiające skumulowaną wartość nagrody zdobywaną przez agenta opracowano na podstawie średniej z pięciu prób pochodzących z niezależnych procesów uczenia z przedziałem ufności 95%. Przebiegi treningów w poszczególnych wariantach środowiska z wykorzystaniem metod PPO oraz SAC przedstawiono na rysunku 4. Na ich podstawie można zaobserwować, że algorytm PPO uzyskał bardzo dobry rezultat w każdym z testowanych środowisk, czyniąc duże podstępy już na samym początku uczenia. Chociaż dalszy trening przebiegał mniej intensywnie, skumulowana wartość nagrody rosła konsekwentnie, aby osiągnąć swoją największą wartość w pobliżu maksymalnej liczby kroków. Wynik ten był też za każdym razem wyższy niż w przypadku uczenia z wykorzystaniem SAC. Algorytm PPO okazał się też bardzo stabilny. W przypadku uczenia z zastosowaniem SAC, zauważyć można zdecydowanie większe wahania wartości nagrody, które stają się coraz bardziej widoczne wraz ze wzrostem złożoności środowiska. Największe różnice pomiędzy algorytmami są widoczne w przypadku najtrudniejszego trybu gry, w którym występują zarówno największe dysproporcje skumulowanej wartości nagrody pomiędzy dwoma algorytmami, jak i bardziej widoczna niestabilność algorytmu SAC.

Przebiegi uczenia algorytmów PPO oraz SAC wspomaganych GAIL w trzecim wariancie środowiska zebrano na rysunku 5. Rozgrywka trwająca 8 minut odpowiada 5 tysiącom kroków demonstracji, z kolei ta o długości 48 minut dostarczyła 30 tysięcy przykładów. Dodatkowe zastosowanie uczenia przez naśladowanie pozwoliło znacząco poprawić wynik uczenia w przypadku algorytmu SAC. Maksymalna skumulowana wartość nagrody zwiększyła się z 6,5 do zakresu w okolicy 8,5. Pomimo że zastosowanie GAIL nie wpłynęło na stabilizację przebiegu, zyskał on bardziej wzrostową tendencję, gdy samodzielny algorytm SAC posiadał skłonność do zahamowania wzrostu skumulowanej wartości nagrody już w połowie treningu. Warto też zauważyć, że zastosowanie GAIL mogłoby pozwolić na otrzymanie lepszego rezultatu w mniejszej liczbie kroków. Pomimo pozytywnego wpływu wykorzystania GAIL w połączeniu z SAC, nie jest to już widoczne w przypadku algorytmu PPO. Przebiegi uczenia samodzielnej metody PPO, jak i tych łączonych z GAIL, okazały się być do siebie bardzo podobne, co sprawia, że trudno jest wskazać poprawę wyniku. Zastosowanie większej liczby demonstracji nie wpłynęło na skuteczność uczenia w przypadku algorytmu PPO. Niewielką różnicę można zaś zaobserwować w przebiegu uczenia SAC przy wykorzystaniu 30 tysięcy przykładów dla GAIL, w którym nieznacznie zwiększyła się maksymalna skumulowana wartość nagrody (o mniej niż ok. 0,5).

Wyniki rozgrywki agentów z modelami wytrenowanymi za pomocą algorytmów PPO oraz SAC w poszczególnych wariantach środowiska przedstawiono na rysunku 6. Rezultat poszczególnych algorytmów otrzymano na podstawie 500 rozgrywek w danym trybie gry. Pomimo słabszego przebiegu uczenia algorytmu SAC, agenci z wynikowym modelem zachowania osiągnęli bardzo wyrównany rezultat w stosunku do PPO w przypadku pierwszego wariantu gry, a nawet zdobywali średnio prawie dwa punkty więcej. Różnice zaczęły się jednak pojawiać w drugim wariancie środowiska, w którym SAC posiadał przewagę jedynie w postaci krótszego średniego czasu ukończenia zadania. Algorytm PPO z kolei osiągnął wyższą średnią celność strzału, jak i większą średnią liczbę zdobytych punktów. Największe dysproporcje pomiędzy dwiema badanymi metodami można zaobserwować jednak w przypadku najtrudniejszego wariantu środowiska. Agenci z modelami wytrenowanymi za pomocą SAC kończyli zadanie średnie o prawie 4 sekundy później, a także posiadali niską celność, trafiając w cel tylko w 80% strzałów. Algorytm PPO okazał



Rysunek 4: Przebieg uczenia algorytmów SAC oraz PPO w poszczególnych wariantach środowiska: (a) pierwszym, (b) drugim, (c) trzecim.



Rysunek 5: Przebieg uczenia algorytmów (a) SAC oraz (b) PPO, wspomaganych GAIL dla różnej liczby wykorzystanych demonstracji.

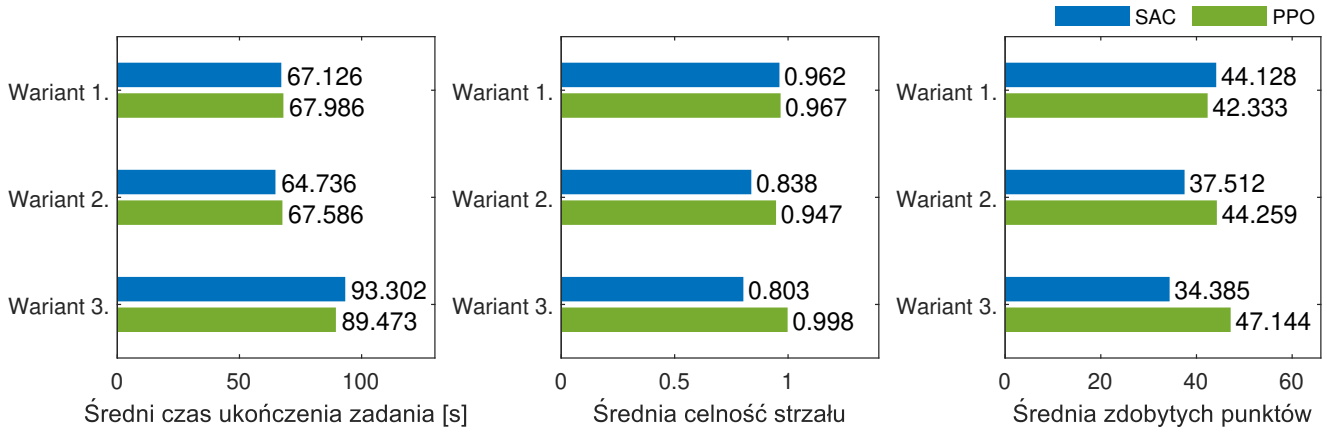
się bardzo skuteczny, uzyskując niemal idealną celność (0,998), a także zdobywając wysoką średnią liczbę punktów (47,144). Metoda SAC uzyskała w tym czasie średnio jedynie 34,385 punktów.

Wyniki rozgrywki agentów z zachowaniem nauczonym z wykorzystaniem łączenia metod PPO oraz SAC z GAIL w trzecim wariancie środowiska przedstawiono w tabeli 3. Podobnie jak w przypadku samego przebiegu uczenia, tak i wynik pochodzący z gry agentów nie zyskał wyraźnych różnic po zastosowaniu GAIL z PPO. Wykorzystanie demonstracji przyniosły jednak poprawę rezultatów w przypadku SAC w każdej z ocenianych kategorii, skracając czas ukończenia zadania, a także zwiększając celność i liczbę zdobytych punktów. Nie wystarczyło to jednak na wyrównanie do wyników osiąganych przez PPO.

Tabela 3: Wyniki rozgrywki agentów w trzecim wariancie środowiska trenowanych za pomocą samodzielnych algorytmów PPO i SAC oraz tych samych metod łączonych z GAIL dla zastosowanej różnej liczby demonstracji

	Średni czas ukończenia [s]		Średnia celność strzału		Średnia suma punktów	
	PPO	SAC	PPO	SAC	PPO	SAC
Bez GAIL	89,473	93,302	0,998	0,803	47,144	34,385
GAIL (5000 kroków)	90,557	92,118	0,995	0,958	47,162	43,824
GAIL (30000 kroków)	90,577	90,568	0,995	0,961	47,043	44,294

Średni czas trwania treningu z wykorzystaniem algorytmów PPO, SAC oraz tych metod łączonych z GAIL zebrano w tabeli 4. Jak można zauważyć, czas ten wydłużał się wraz ze wzrostem złożoności środowiska, czego przyczyną była konieczność stosowania coraz to większej sieci neuronowej, jak i stopniowego zwiększania liczby kroków uczenia. Algorytm PPO charakteryzował się krótszym czasem treningu niż SAC w każdym z wariantów gry. Różnica ta rosła wraz ze zwięk-



Rysunek 6: Wyniki rozgrywki w poszczególnych wariantach środowiska uzyskane przez agentów trenowanych za pomocą algorytmów SAC i PPO.

szaniem trudności zadania wykonywanego przez agenta. W najtrudniejszym wariancie środowiska SAC potrzebował średnio około 6 minut więcej na ukończenie uczenia w przewidzianej liczbie kroków. W przypadku PPO łączonego z GAIL dla 5 tysięcy kroków czas uczenia został wydłużony o prawie 19%. Zwiększenie liczby przykładów wpłynęło zaś na wzrost tego czasu o blisko 22%. Większą różnicę można zaobserwować jednak przy zastosowaniu GAIL z SAC. Dla małej liczby demonstracji średni czas uczenia zwiększył się o 36%, wydłużając trening z 37,5 do około 51 minut. Wykorzystanie przykładów odpowiadających 30 tysiącom kroków poskutkowało wzrostem czasu uczenia do niewielu ponad 50 minut.

Tabela 4: Średnia oraz odchylenie standardowe czasu uczenia algorytmów PPO oraz SAC w trzecim wariancie środowiska oraz wynik tych samych algorytmów łączonych z metodą GAIL

	PPO [s]	σ	SAC [s]	σ
Wariant 1.	927,6	26,1	968,2	37,2
Wariant 2.	1719,4	14,6	1789,2	33
Wariant 3.	1881,4	19,9	2250,6	70,6
Wariant 3. z GAIL (5000 kroków)	2233	27,6	3065,4	70,7
Wariant 3. z GAIL (30000 kroków)	2300,8	32,9	3003,6	37,7

Przy podsumowaniu rezultatów badania, należy podkreślić istotny wpływ wyboru hiperparametrów na wyniki uczenia. Metodę ręcznej ich regulacji nie można uznać za najskuteczniejszą, a bardziej wynikającą z konieczności. Pakiet ML-Agents Toolkit nie posiada, jak dotąd, narzędzi umożliwiających automatyzację tego procesu. Zawodność ręcznego doboru hiperparametrów mogła mieć tym bardziej istotny wpływ na wynik badania ze względu na większe odchylenia przebiegów uczenia z SAC. Mogło to wpłynąć na niekorzystną interpretację wyników podczas doboru konfiguracji parametrów wykorzystanych we właściwym badaniu. Dodatkowym czynnikiem wpływającym na znaczenie sposobu wyboru hiperparametrów, a wskazującym konieczność stosowania metod zautomatyzowanych, jest zaobserwowana czułość algorytmu SAC na zmianę ich wartości. W ostatecznej ocenie wyników badania, należy więc uwzględniać możliwość istnienia bardziej odpowiednich hiperparametrów, których wykorzystanie mogłoby potencjalnie przynieść poprawę skuteczności metody SAC.

5. Wnioski

Przeprowadzone badanie pozwoliło dokonać porównania dwóch współczesnych algorytmów uczenia ze wzmocnieniem — PPO oraz SAC, a także weryfikacji możliwości poprawy skuteczności uczenia poprzez łączenie tych metod z algorytmem uczenia przez naśladowanie GAIL. Zaprojektowane w tym celu środowisko powstało przy użyciu platformy Unity oraz biblioteki Unity ML-Agents Toolkit. Składało się ono z kilku wariantów, sprawdzających skuteczność algorytmów w zależności od stopnia złożoności prezentowanego zadania. Celem agenta w utworzonej grze było strzelanie z łuku do tarczy treningowej, przy czym poszczególne tryby rozgrywki stawiały przed agentem większe wymagania. Algorytmy PPO i SAC zostały porównane na podstawie przebiegu uczenia, jak i skuteczności nauczonego zachowania. W przypadku sprawdzenia możliwości łączenia uczenia ze wzmocnieniem z metodą GAIL zbadano dodatkowo wpływ liczby demonstracji na wynik uczenia.

Na podstawie wyników przeprowadzonego badania, algorytm PPO okazał się skuteczniejszy od SAC. Pomimo wyrównanych rezultatów obu metod w przypadku środowiska o najniższym poziomie trudności zadania, znaczące różnice pomiędzy algorytmami zaczęły występować w dwóch trudniejszych wersjach przygotowanej gry. Agenci wytrenowani za pomocą PPO osiągali znacznie wyższy rezultat podczas rozgrywki, zwłaszcza w najbardziej złożonym wariancie środowiska. Ana-

liza przebiegu procesu uczenia pozwoliła zaobserwować znacznie większą niestabilność skumulowanej wartości nagrody w przypadku algorytmu SAC, występującą najsilniej w najtrudniejszym trybie rozgrywki. Analiza czasu uczenia wykazała, że PPO był nieznacznie szybszy w przypadku dwóch pierwszych wariantów środowiska. Okazał się on jednak o wiele wydajniejszy czasowo w najtrudniejszym trybie rozgrywki, ale przyczyną takiego rezultatu była najprawdopodobniej konieczność zastosowania większej sieci neuronowej dla SAC.

Uzyskane wyniki badania pozwoliły potwierdzić hipotezę o możliwości poprawy rezultatu treningu poprzez łączenie algorytmów uczenia ze wzmocnieniem z GAIL. Efekt ten udało się osiągnąć w przypadku SAC, dla którego otrzymano znacznie lepsze wyniki zarówno przebiegu, jak i rozgrywki, przy zastosowaniu jedynie kilkuminutowej demonstracji. Poprawa rezultatów nie była jednak wystarczająca, aby osiągnąć skuteczność porównywalną z algorytmem PPO. Zaobserwowano też, że zastosowanie GAIL znacznie wydłużyło czas uczenia. W przypadku PPO nie udało się zaobserwować istotnej zmiany wyników treningu, ale należy wziąć pod uwagę to, że już samodzielny algorytm charakteryzował się dużą skutecznością, a zastosowane przykłady odpowiadały zachowaniu suboptymalnemu. Wykorzystanie większej liczby demonstracji nie przyniosło istotnej poprawy wyników treningu w przypadku obu algorytmów.

Zebrane rezultaty badania pozwoliły wskazać, że algorytm PPO jest metodą bardziej uniwersalną, zdolną sprawdzić się w większości środowisk, posiadającą przy tym sporą stabilność i skuteczność uczenia. Poprzez ręczny dobór hiperparametrów zauważono, że jest to metoda prosta w konfiguracji, znacznie mniej wrażliwa na zmiany wartości hiperparametrów niż SAC. Wniosek ten może mieć większe znaczenie biorąc pod uwagę przeznaczenie wykorzystanych narzędzi, czyli platformy Unity i pakietu ML-Agents Toolkit, które są dedykowane dla szerszej grupy użytkowników, nie tylko badaczy, ale i programistów. Dodatkową przeszkodą w przeprowadzeniu skutecznego uczenia z wykorzystaniem SAC może być brak dostępnych w Unity rozwiązań pozwalających na automatyzację poszukiwania najbardziej korzystnych wartości hiperparametrów. Możliwe jest, że wybór parametrów posiadał istotny wpływ również na wynik przeprowadzonego badania i istnieje konfiguracja, która przyniosłaby lepszy rezultat, zwłaszcza w przypadku algorytmu SAC. Okazało się też, że niezadowalający wynik uczenia można skutecznie poprawić wykorzystując udostępniany przez pakiet algorytm uczenia przez naśladowanie GAIL. Pamiętać jednak należy, że wybór tej metody wiąże się z koniecznością dodatkowej rejestracji demonstracji oraz możliwym dłuższym czasem uczenia.

Zaprezentowane w niniejszej pracy badanie może być początkiem dalszych eksperymentów, wykorzystujących bardziej skomplikowane środowiska, o większym zapotrzebowaniu zasobów sprzętowych, czy czasu uczenia. Ciekawą hipotezą, wartą weryfikacji może być to, że algorytm SAC mógłby sprawdzić się lepiej w bardzo powolnym środowisku. Dalsze badania mogą dotyczyć również pozostałych możliwości oferowanych przez pakiet ML-Agents Toolkit, czego przykładem może być moduł *Curiosity*, zalecany w środowiskach z rzadkimi nagrodami, czy zastosowanie rekurencyjnej sieci neuronowej, która może być wykorzystywana jako pamięć agentów.

Literatura

[1] A. Juliani, V. P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, D. Lange, Unity: A General Platform for Intelligent Agents, arXiv preprint arXiv:1809.02627v2 (2020).

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).

[3] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, In Proceedings of Machine Learning Research 80 (2018) 1861–1870.

[4] J. Ho, S. Ermon, Generative Adversarial Imitation Learning, Advances in neural information processing systems (2016) 4565–4573.

[5] A. Hussein, M. M. Gaber, E. Elyan, C. Jayne, Imitation Learning: A Survey of Learning Methods, ACM Computing Surveys 50(2) (2017) 1–35, https://doi.org/10.1145/3054912.

[6] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, The MIT Press, Cambridge, 2018.

[7] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, In International conference on machine learning (2015) 1889–1897.

[8] M. Urmanov, M. Alimanova, A. Nurkey, Training Unity Machine Learning Agents using reinforcement learning method, In 2019 15th International Conference on Electronics, Computer and Computation (2019) 1–4, https://doi.org/10.1109/ICECCO48375.2019.9043194.

[9] M. Pleines, F. Zimmer, V. Berges, Action Spaces in Deep Reinforcement Learning to Mimic Human Input Devices, In 2019 IEEE Conference on Games (2019) 1–8, https://dx.doi.org/10.1109/CIG.2019.8848080.

[10] V. Mnih, K. Kavukcuoglu, D. Silver et al., Human-level control through deep reinforcement learning, Nature 518(7540) (2015) 529–533.

[11] M. G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The Arcade Learning Environment: An Evaluation Platform for General Agents, Journal of Artificial Intelligence Research 47 (2013) 253–279.

[12] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, C. Blundell, Agent57: Outperforming the Atari Human Benchmark, In International Conference on Machine Learning (2020) 507–517.

[13] A. Defazio, T. Graepel, A Comparison of learning algorithms on the Arcade Learning Environment, arXiv preprint arXiv:1410.8620 (2014).

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, arXiv preprint arXiv:1606.01540 (2016).

[15] A. Tavakoli, F. Pardo, P. Kormushev, Action branching architectures for deep reinforcement learning, In Proceedings of the AAAI Conference on Artificial Intelligence 32(1) (2018).

[16] Dokumentacja biblioteki ML-Agents Toolkit — opis i zalecany zakres wartości hiperparametrów uczenia, https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md, [04.05.2021].