

Politechnika Warszawska

W Y D Z I A Ł E L E K T R Y C Z N Y



INSTYTUT STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ
ZAKŁAD STEROWANIA

Praca dyplomowa magisterska

na kierunku INFORMATYKA STOSOWANA
w specjalności Inżynieria oprogramowania

Sterowanie grą wieloagentową przy pomocy algorytmów
uczenia ze wzmocnieniem

Piotr Klukowski

nr albumu 252662

promotor

dr inż. Grzegorz Sarwas

Warszawa 2021

Sterowanie grą wieloagentową przy pomocy algorytmów uczenia ze wzmacnieniem

Streszczenie

Niniejsza praca dotyczy problemu wykorzystania algorytmów uczenia przez wzmacnianie w sterowaniu wieloma agentami w grze. Jako środowisko testowe zostało wykorzystane opracowane przez Google środowisko gfootball, które jest uproszczoną wersją gry komputerowej w piłkę nożną. Środowisko to zostało poddane analizie pod kątem doboru zaimplementowanych w nim funkcji nagrody, jak również ich wpływu na prędkość uczenia algorytmów. Część z analizowanych konfiguracji była odtworzeniem badań i wyników zaprezentowanych przez Google. W celu dokonania badań została przeprowadzona integracja środowiska z bibliotekami uczenia maszynowego.

W niniejszej pracy dokonano również przeglądu algorytmów nieujętych w wynikach zaprezentowanych przez Google. Do przeprowadzenia tych badań wykorzystana została biblioteka stable-baselines stworzona przez Ashley'a Hill'a. Przeanalizowany został również wpływ ustawienia hiper-parametrów algorytmu PPO2 na prędkość procesu uczenia. Podjęta została również próba opracowania nowej funkcji nagrody.

Słowa kluczowe: uczenie maszynowe, uczenie ze wzmacnieniem, sterowanie grą wieloagentową, sieci głębokie

Multi-agent game control using reinforcement learning algorithms

Abstract

This thesis addresses the problem of using reinforcement learning algorithms in controlling multiple agents in a game. The gfootball environment developed by Google, which is a simplified version of the computer game of soccer, was used as a test environment. This environment was analyzed in terms of the choice implemented reward functions, as well as their effect on the learning speed of the algorithms. Some of the analyzed configurations replicated research and results presented by Google. The research required an integration of the environment with machine learning libraries.

This study also reviews algorithms not included in the results presented by Google. The stable-baselines library created by Ashley Hill was used to conduct this research. The effect of setting hyper-parameters of the PPO2 algorithm on the speed of the learning process has also been analyzed. In addition, an attempt was made to develop a new reward function.

Keywords: machine learning, reinforcement learning, multiagent game control, deep neural network

Warszawa, 12 lutego 2021

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa magisterska pt. Sterowanie grą wieloagentową przy pomocy algorytmów uczenia ze wzmacnieniem:

- została napisana przeze mnie samodzielnie,
- nie narusza niczyich praw autorskich,
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej. Jestem świadom, że praca zawiera również wyniki stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Piotr Klukowski.....

Spis treści

1 Wstęp	1
2 Uczenie przez wzmacnianie	4
2.1 Podział uczenia przez wzmacnianie	7
2.2 Metoda Q-Learning	9
2.3 Metoda Optymalizacji Polityk	9
2.4 Kompromis pomiędzy Q-learning i Optymalizacją Polityk	10
3 Środowisko gfootball	11
3.1 Rodzaje obserwacji	11
3.2 Funkcje nagrody	12
3.3 Scenariusze	13
4 Problemy ze środowiskiem gfootball	16
4.1 Niekompatybilność z oficjalną dokumentacją - artykułem	16
4.2 Biblioteka Baselines od OpenAI	17
5 Nakładka na środowisko	18
5.1 Instalacja środowiska na systemie Linux	18
5.2 Opis Kodu	19
5.2.1 Trener	20
5.2.2 Evaluator	24
5.2.3 Skrypt Bash	28
5.3 Implementacja w COLAB	29

6	Badania	30
6.1	Porównanie funkcji nagrody	32
6.2	Badanie wpływu wysokości nagrody pomocniczej na prędkość uczenia	38
6.3	Porównanie algorytmów dostępnych w bibliotece stable-baselines	41
6.4	Badanie wpływu hiperparametrów na prędkość uczenia	45
7	Propozycja nowego rodzaju nagrody dla środowiska gfootball	49
7.1	Opis teoretyczny	49
7.2	Implementacja	51
7.3	Wyniki Badań	55
7.4	Podsumowanie	57
8	Podsumowanie	59
A	Hiperparametry PPO2	62
B	Wyniki badań porównania wysokości nagrody pomocniczej 'checkpoints'	64
C	Wyniki badań porównania algorytmów: PPO2, A2C i ACER z defaultowymi hiperparametrami z biblioteki stable-baselines	68
D	Wyniki badań porównania hiperparametrów dla algorytmu PPO2	72
E	Wyniki badań nagrody typu 'possession'	77

Rozdział 1

Wstęp

Uczenie maszynowe jest dynamicznie rozwijającą się dziedziną nauki. Możemy podzielić je na trzy klasy problemów: uczenie nadzorowane, uczenie nienadzorowane i uczenie przez wzmacnianie (ang. Reinforcement Learning - RL). W czasie pisania niniejszej pracy uczenie nadzorowane oraz uczenie nienadzorowane są w dużym stopniu tematami dojrzałymi, natomiast problem uczenia przez wzmacnianie jest cały czas rozwijany. Fakt ten przyczynił się właśnie do podjęcia w niniejszej pracy tej tematyki. Dodatkowym aspektem jest wieloagentowość środowisk, które są trudniejsze, ale bardziej praktyczne, ponieważ są bliższe środowisku rzeczywistemu. Cały świat i wiele jego składowych można określić mianem systemów wieloagentowych, począwszy od gier, a skończywszy na giełdzie czy otaczającej nas przyrodzie. Celem uczenia przez wzmacnianie jest wytrenowanie inteligentnych agentów, którzy mają dokonywać interakcji z otoczeniem, dzięki czemu możliwe staje się rozwiązywanie bardzo złożonych problemów.

Do badania tego typu algorytmów doskonałe są gry komputerowe, gdyż agent będący programem komputerowym może bezpośrednio wykonywać swoje akcje w środowisku oraz widzi rezultaty swoich działań. Ponadto środowiska cyfrowe są bardzo elastyczne i modyfikalne. Dodatkowym argumentem przemawiającym za grami komputerowymi jest to, iż można w łatwy sposób zaobserwować wyniki działań agenta.

Gry komputerowe mogą być również wykorzystywane jako symulatory. Taki symulator znacznie poprawia prędkość uczenia się algorytmu, gdyż wyniki otrzymywane poprzez obliczenia procesora są niejednokrotnie szybsze od tych zarejestrowanych z układów fizycznych. Ponadto symulator jest bezpieczniejszy i jest odporny na ludzkie błędy.

Przykładem gry wieloagentowej jest piłka nożna. Środowisko gfootball jest uproszczoną grą komputerową w piłkę nożną. Zostało ono stworzone przez firmę Google i bazuje na silniku gry w football stworzonym przez Bastiaan'a Schuiling'a [1].

Celem niniejszej pracy jest sterowanie grą wieloagentową (środowiskiem gfootball) przy pomocy algorytmów uczenia ze wzmacnieniem. Dodatkową motywacją do pracy była chęć przetestowania środowiska Google'a w innych konfiguracjach i przy użyciu innych algorytmów niż zrobili to autorzy projektu. Zakresem pracy jest:

1. Analiza środowiska wieloagentowego do gry w piłkę nożną;
2. Implementacja połączenia środowiska gfootball z bibliotekami algorytmów uczenia maszynowego;
3. Przegląd algorytmów uczenia maszynowego pod kątem środowiska gfootball oraz badanie ich efektywności uczenia się;
4. Badanie wpływu hiper-parametrów algorytmów na prędkość uczenia;
5. Dodatkowo: Koncepcja, implementacja oraz badanie funkcji nagrody nowego typu.

Pierwszym elementem pracy jest analiza i badanie środowiska gfootball. Składał się on z dwóch części. Pierwsza dotyczy badania istniejących funkcji nagrody i ich wpływu na proces uczenia. W efekcie jest to powtórzenie badań przedstawionych w benchmarku dostarczonym przez Google. Druga część badań skupia się na analizie wpływu wysokości nagrody pomocniczej na prędkość uczenia przy użyciu bibliotek uczenia maszynowego. Okazało się, że implementacja połączenia środowiska gfootball z bibliotekami algorytmów uczenia maszynowego była niezbędna do wykonania pierwszej części badań.

Trzeci element pracy, czyli przeglądem algorytmów wraz z ich badaniem, będzie przeprowadzony na innych algorytmach, które nie zostały ujęte w analizach przedstawionych przez Google. W tym celu wykorzystana zostanie biblioteka stable-baselines stworzona przez Ashley'a Hill'a (github: hill-a) [2], która jest rozwidleniem znanej, choć w momencie pisania pracy niestabilnej, biblioteki opracowanej przez organizację OPEN.AI - baselines [3]. Czwartym elementem pracy jest badanie wpływu doboru hiper-parametrów algorytmów na prędkość uczenia się algorytmu. Do eksperymentów wykorzystany został algorytm PPO2. Ostatnim, piątym elementem jest koncepcja i implementacja nowej funkcji nagrody. Badania te zostały również przeprowadzony przy zastosowaniu algorytmu PPO2.

Praca została zorganizowana w następujący sposób. W rozdziale 2 zostały opisane podstawy teoretyczne dotyczące uczenia przez wzmacnianie. Rozdział 3 przedstawia szczegóły środowiska gfootball. Rozdział 4 porusza problemy zaistniałe podczas pracy ze środowiskiem Google'a. W rozdziale 5 zostało zaprezentowane rozwiązanie wcześniej wspomnianych problemów. Rozdział 6 przedstawia wszystkie badania środowiska gfootball. W rozdziale 7 została w pełni opisana nowa funkcja nagrody typu 'possession'.

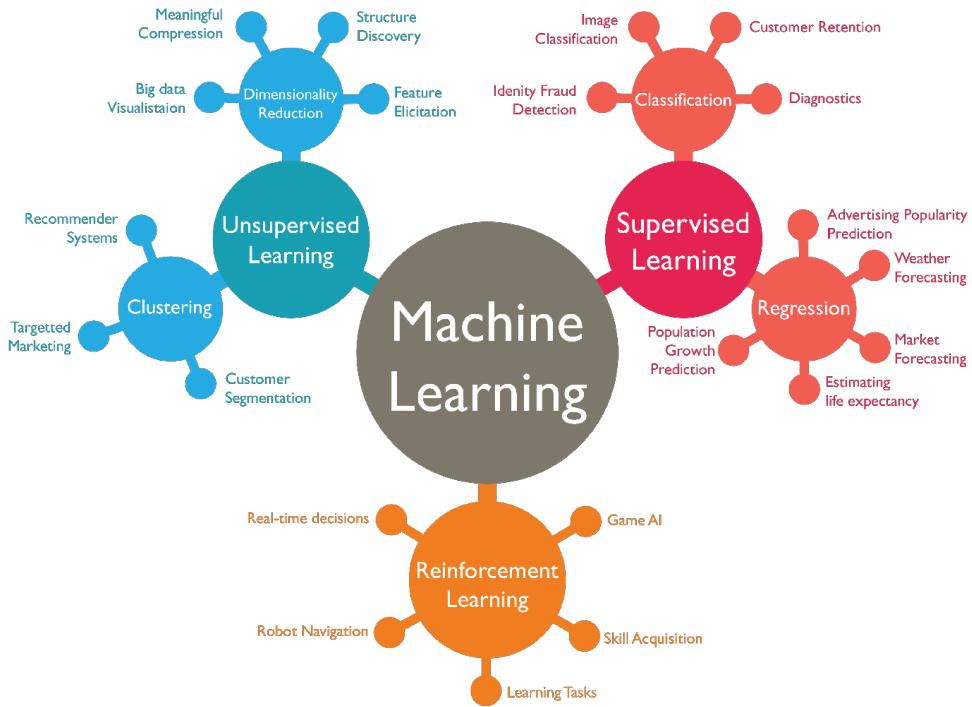
Rozdział 2

Uczenie przez wzmacnianie

Uczenie przez wzmacnianie jest jednym z elementów uczenia maszynowego (ang. machine learning - ML), które jest częścią szeroko pojętej sztucznej inteligencji (ang. artificial Intelligence - AI). Dokładny podział AI został zaprezentowany na rysunku 2.1. Schemat ten został stworzony poprzez zastosowanie algorytmów sztucznej inteligencji.

Problemy rozwiązywane przez algorytmy uczenia przez wzmacnianie zwykle nie są rozwiązywane tylko i wyłącznie przy użyciu algorytmów RL. Często do optymalizacji procesu uczenia wykorzystuje się techniki uczenia nadzorowanego, rzadziej nienadzorowanego.

Uczenie nadzorowane to najczęściej klasyfikacja danych do pewnych zbiorów, czyli na podstawie cech wejściowych obiektów przyporządkowywanie im etykiet poznanych w procesie uczenie lub regresja, czyli na podstawie wektora cech wejściowych obiektu określanie liczbowej wartości wyjściowej.



Rysunek 2.1: Podział machine learningu, źródło: [4]

Uczenie nienadzorowane to w prostym ujęciu podział danych na zbiory (tzw. klastrowanie lub grupowanie). Dodatkowo techniki uczenia nienadzorowanego wykorzystywane są do redukcji wymiarowości danych (np. dzięki redukcji wymiarów z 10000 parametrów wejściowych możemy wyodrębnić 100, tak aby później móc stworzyć algorytm predykcyjny bazujący tylko na 100 cechach opisujących). Jest to bardzo istotne z punktu widzenia ograniczenia kosztu obliczeniowego, gdzie czas obliczeń może rosnąć wykładniczo w stosunku do ilości cech wejściowych.

Trzecią gałęzią jest uczenie przez wzmacnianie (ang. Reinforcement Learning). Idea uczenia najlepiej opisana jest w [5]. Autor opisuje ją w następujący sposób:

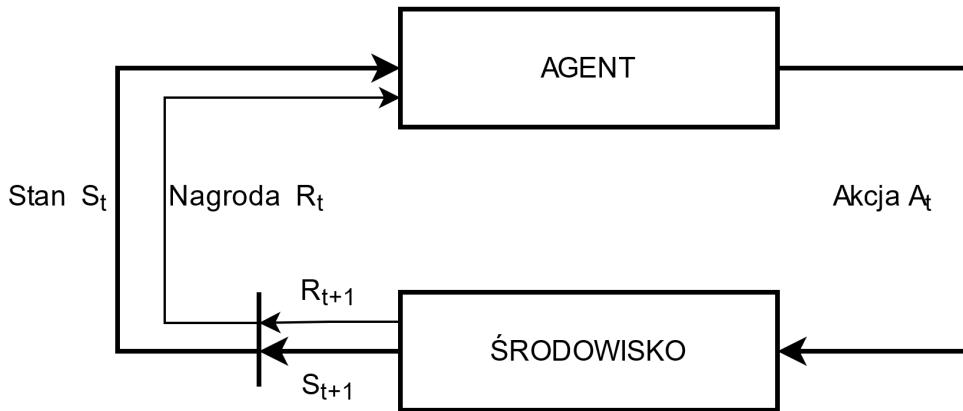
Idea uczenia przez wzmacnianie towarzyszy nam już od pierwszych dni życia. W momencie gdy człowiek uczy się prowadzić pierwsze interakcje z otaczającym go środowiskiem. Wykonując

różne ruchy uzyskuje informacje zwrotne od środowiska zmysłami takimi jak zmysł wzroku czy dentyku. Korzystając z tych informacji zaczyna pojmować konsekwencje swych działań i zaczyna dostrzegać co należy robić, aby osiągnąć założone cele. W taki właśnie sposób człowiek poznaje otaczający go świat. Możemy nauczyć się jeździć rowerem lub samochodem. Po nauczeniu stajemy się w pełni świadomi jak reaguje środowisko i jak możemy na nie wpływać naszym zachowaniem.

Podsumowując cytat, uczenie przez wzmacnianie jest to nauka interaktywna, która jest fundamentalną ideą leżącą u podstaw większości teorii uczenia i sztucznej inteligencji. Obecnie eksplorujemy świat coraz to w bardziej skomplikowany sposób. Tworzymy nowe konstrukcje maszyn, rozwiązujemy problemy naukowe czy techniczne, a nawet ekonomiczne. Takie właśnie eksplorowanie świata i rozwiązywanie problemów w oparciu o cel jest uczeniem przez wzmacnianie.

Omawiane uczenie przez wzmacnianie jest w uproszczeniu metodologią, w której mapujemy sytuacje do akcji tak, aby zmaksymalizować nagrodę. Uczący nie dostaje informacji jaką akcję ma podjąć, ale to on sam musi odkryć poprzez prowadzone w procesie uczenia eksperymenty, które akcje należy podjąć aby uzyskać jak najwyższą nagrodę. W bardzo zaawansowanych przypadkach akcje nie tylko mogą powodować nagrody cząstkowe, ale także nowe możliwości akcji i kolejne nagrody. Te charakterystyczne cechy: przeszukiwanie i opóźnione nagrody są kluczowe w uczeniu przez wzmacnianie.

Uczenie przez wzmacnianie jest formalizowane przy użyciu teorii układów dynamicznych, na przykład przy użyciu procesów decyzyjnych Markowa. Idea ta polega na wyodrębnieniu najbardziej kluczowych elementów rzeczywistego problemu poprzez agenta, który aby osiągnąć cel będzie prowadził interakcję ze środowiskiem. Agent musi uzyskiwać niezbędne dane ze środowiska, w którym się znajduje i móc podejmować akcje, które na nie wpływają. Ponadto agent musi mieć cel związany ze stanem środowiska. Tak właśnie w olbrzymim



Rysunek 2.2: Agent w procesie decyzyjnym Markowa

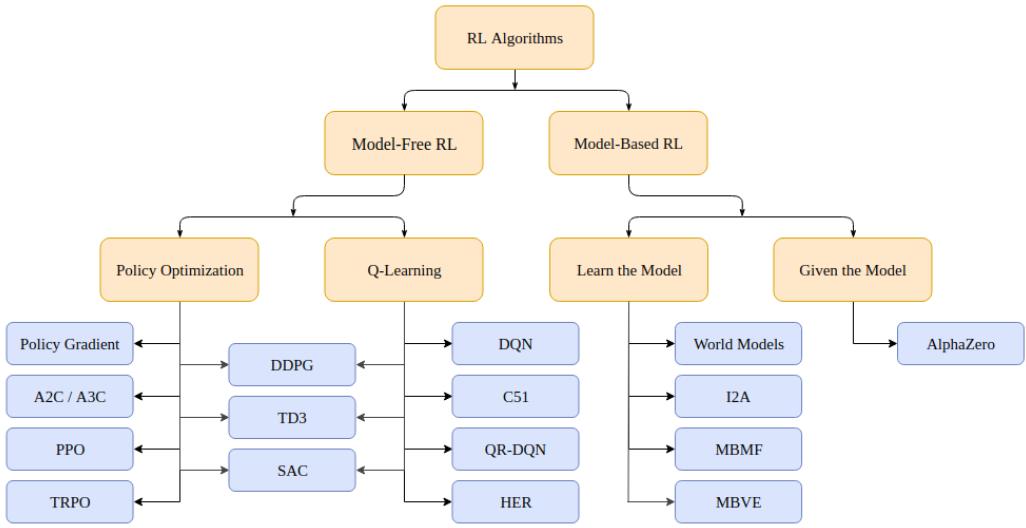
skrócie wygląda proces decyzyjny Markowa, który zawiera pomiar, akcje i cel. Jest on ogólnym schematem Uczenia Maszynowego. Jego uproszczona architektura została przedstawiona na rysunku 2.2.

W dalszej części tego rozdziału zostanie szczegółowo omówione uczenie przez wzmacnianie. Zostanie przedstawione jak obecnie dzieli się tę gałąź sztucznej inteligencji oraz jakie metody stosuje się do rozwiązywania problemów w omawianym zagadnieniu.

2.1 Podział uczenia przez wzmacnianie

W tym podrozdziale zostanie poruszony podział algorytmów RL. W środowisku uczenia maszynowego przyjęto podział zaprezentowany przez organizację Open.AI, który został przedstawiony na rysunku [6].

Algorytmy dzielą się na te 'bez modelu' (ang. Model-Free RL) oraz 'oparte na modelu' (ang. Model-Based RL). Model w tym wypadku oznacza funkcję, która przewiduje stan środowiska po wykonaniu akcji. Różnica pomiędzy algorytmami 'bez modelu' i 'opartymi na modelu' polega na dostępie agenta do środowiska. W pierwszym przypadku agent nie ma pojęcia o środowisku,



Rysunek 2.3: Podział uczenia ze wzmacnieniem, źródło: [6]

w którym istnieje. Drugi przypadek to taki, że agent doskonale zna otoczenie w jakim się znajduje.

Główną zaletą posiadania modelu jest to, że pozwala on agentowi planować poprzez przewidywanie. Agent posiada wiedzę na temat tego co może się stać w przyszłości w szeregu możliwych wyborów. Agenci następnie mogą wyodrębniać rezultaty planowania wprost do uczonej polityki.

Główna niedogodnością z jaką borykają się algorytmy z modelem jest fakt, iż zazwyczaj model środowiska jest niedostępny dla agenta. Jeżeli agent chce użyć modelu musi nauczyć się go na podstawie własnych doświadczeń, co stwarza bardzo duży problem. Największym wyzwaniem jest to, że tendencjonalność modelu może być wykorzystana przez agenta. Skutkuje to tym, że agent działa dobrze w odniesieniu do wyuczzonego modelu, jednakże nie zachowuje się optymalnie w środowisku rzeczywistym.

Metody bez modelu, chociaż wydają się być bardziej skomplikowane, ponieważ rezygnują z potencjalnego zysku z używania modelu, są łatwiejsze do wdrożenia. Ponadto przez ich uniwersalność stały się bardziej popularne i obszernie opracowywane oraz testowane, dlatego też w dalszej części tej

pracy został podjęty temat uczenia przez wzmacnianie przez wykorzystanie algorytmów bez modelu.

Jak można zauważyc na rysunku 2.3 algorytmy wolne od modelu dzieli się następnie na algorytmy bazujące na algorytmy optymalizujące politykę (opisane w 2.3) oraz na tzw. algorytmy Q-learning (opisane w 2.2). Oba podejścia zostaną omówione szerzej w dalszej części tego rozdziału. Warte uwagi są algorytmy hybrydowe, które poniekąd stanowią kompromis pomiędzy wcześniejszym przedstawionymi podejściami. Zostaną one szerzej omówione w 2.4.

2.2 Metoda Q-Learning

Metody z tej rodziny uczą się aproksymatora $Q_\theta(s, a)$ dla optymalnej funkcji $Q^*(s, a)$. Zazwyczaj używają one funkcji celu opartej na równaniu Bellmana. Optymalizacja ta jest niemalże zawsze wykonywana w trybie bez polityki. Tryb ten oznacza, że każda aktualizacja może wykorzystywać dane zebrane podczas całego treningu, niezależnie od wyboru sposobu eksploracji środowiska przez agenta w danym momencie. Odpowiednią politykę uzyskuje się poprzez połączenie między Q^* i π^* . Akcje podejmowanie przez agenta przy algorytmie Q-learningowym są podktowane równaniem:

$$a(s) = \arg \max_a Q_\theta(s, a)$$

Przykładem użycia Q-learningu jest popularny algorytm DQN.

2.3 Metoda Optymalizacji Polityk

Algorytmy tego typu reprezentują politykę w sposób jawny jako zmienna $\pi_{\text{theta}}(a|s)$. Optymalizują one parametr θ poprzez metodę gradientu prostego w celu optymalizacji $J(\pi_{\text{theta}})$, lub też pośrednio poprzez maksymalizację lokalnych przybliżeń $J(\pi_{\text{theta}})$. Optymalizacja ta jest wykonywana niemalże zawsze w trybie z polityką. Tryb ten oznacza, że każda aktualizacja wykorzystuje dane zebrane tylko podczas działania z najnowszą wersją polityki.

Optymalizacja polityki często wiąże się również z uczeniem aproksymatora $V_{phi}(s)$ dla funkcji wartości $V^\pi(s)$. Zmienne te są wykorzystywane do ustalenia w jaki sposób aktualizować politykę.

Najbardziej znanymi algorytmami tego typu są: A2C czy też jego nowsza wersja A3C oraz PPO i jego nowsza wersja PPO2. PPO2 w niniejszej pracy będzie bardzo intensywnie używane w badaniach przedstawionych w rozdziałach 6 oraz 7.

2.4 Kompromis pomiędzy Q-learning i Optymalizacją Polityk

Największą zaletą metody optymalizacji polityk jest ich ścisłe oparcie na określonych i niezmiennych zasadach. Dzięki temu są one stabilne i niezawodne. Ponadto w przypadku Q-learningu jest wiele przypadków awaryjnych, przez co jest to grupa algorytmów mniej stabilnych niż grupa z optymalizacją polityk. Jednakże metody Q-learningu zyskują w przypadku wydajności, gdyż efektywniej wykorzystują dane.

Ciekawym rozwiązaniem jest interpolacja metod Q-learningu i optymalizacji polityk. Okazuje się, że metody te są ze sobą kompatybilne. Algorytmy, które implementują w sobie oba rodzaje metod, potrafią odpowiednio zarządzać wynikami Q-learningu i metody Optymalizacji Polityk oraz radzić sobie ze słabymi stronami każdej z tych metod poprzez odpowiednie wspomaganie drugiej.

Bardzo dobrze znanym przykładem z zastosowaniem tzw. hybrydy obu metod jest algorytm DDPG, który uczy się deterministycznie polityki oraz funkcji Q i odpowiednio aktualizuje jedną wartość drugą, tak aby się wzajemnie usprawniały.

Rozdział 3

Środowisko gfootball

Środowisko gfootball zostało oryginalnie stworzone przez [1]. Zostało ono napisane w języku C++ przez Bastiaan'a Schuiling'a. Google Research użyło tego środowiska tworząc na nie nakładkę w języku python, pozwalającą na łatwą implementację algorytmów uczenia przez wzmacnianie.

W tym rozdziale zostanie omówione środowisko gfootball [7]. Zostaną przedstawione rodzaje obserwacji środowiska, czyli typ obserwacji jaka jest dokonywana po każdym kroku czasowym. Następnie zostaną przedstawione dostępne w projekcie funkcje nagrody. Ostatecznie w ostatnim podrozdziale zostaną przedstawione wszystkie dostępne scenariusze.

3.1 Rodzaje obserwacji

Aby móc użyć jakiegokolwiek algorytmu w danym środowisku należy je w jakiś sposób odczytywać i prezentować w formie danych. W tym celu zostały stworzone 4 sposoby obserwacji:

1. `'pixels'` - obserwacja środowiska w postaci pikseli RGB;
2. `'pixel_gray'` - obserwacja środowiska w postaci pikseli w odcieniach szarości;
3. `'extracted'` - obserwacja składająca się z 4 płaszczyzn o wymiarach reprezentacji pixelowej. Na pierwszej płaszczyźnie są pozycje zawodni-

ków pierwszej drużyny. Na drugiej płaszczyźnie zawodników z drugiej drużyny. Na trzeciej płaszczyźnie pozycja piłki, a na czwartej pozycja aktualnie sterowanego zawodnika. Ten rodzaj reprezentacji jest również nazywany minimapą;

4. `'simple115v2'` - obserwacja w której wszystkie dane przetrzymuje się w jednowymiarowym wektorze o długości 115, składającym się z:

- pozycja piłki (x, y, z),
- wektor binarny o długości 3 mówiący o tym kto dokładnie posiada piłkę (nikt lub prawa drużyna lub lewa drużyna),
- wektor o długości 11 mówiący który gracz obecnie jest aktywny,
- 11 pozycji zawodników lewej drużyny (x, y),
- 11 prędkości zawodników drużyny lewej w kierunkach x i y,
- 11 pozycji zawodników prawej drużyny (x, y),
- 11 prędkości zawodników drużyny prawej w kierunkach x i y,
- wektor binarny o długości 7 z informacją o stanie gry (rzut różny, rzut karny, normalna gra itp.).

Twórcy zalecają używać tego trybu jedynie dla scenariuszy polegających na pełnej rozgrywce meczowej.

3.2 Funkcje nagrody

Oryginalnie w środowisku zaimplementowane zostały dwie funkcje nagrody:

- `'scoring'`,
- `'checkpoint,scoring'`.

Pierwsza z nich to zliczanie bramek. Wadą tego rozwiązania jest to, że algorytm na początku nie posiada informacji co jest poprawne, a co nie jest. Dopiero gdy którakolwiek z drużyn (czy sterowania przez agenta, czy drużyna

przeciwna) zdobędzie bramkę, to od wtedy algorytm wie co musi robić lub czego nie. Jednakże dopóki nie zdobędzie gola nie wie jak zdobyć pozytywną nagrodę.

Z wymienionych przyczyn została w środowisku zaimplementowana druga funkcja nagrody uwzględniająca przesunięcie się piłki w kierunku bramki przeciwnika. Jest to analogiczne do futbolu amerykańskiego, w którym punkty są zdobywane nie tylko przez zdobycie gola, ale także przez przeniesienie piłki przez linie punktową. Dzięki temu rozwiązaniu mamy nagrody częściowe za wykonane akcje, a algorytm wie, że trzeba kierować się w kierunku bramki przeciwnika, by zmaksymalizować nagrodę.

Przyznawanie nagrody za przesunięcie się piłki w kierunku bramki polega na podzieleniu boiska na 10 części i przyznaniu określonej premii wynoszącej 0,1 za to, że piłka w posiadaniu kontrolowanej drużyny przebędzie drogę w kierunku bramki przeciwnika. Dokładne obliczenia są bardziej skomplikowane, aczkolwiek ze względu na brak jakiekolwiek dokładnej dokumentacji w tej pracy nie będzie tworzona inżynieria wstępna środowiska gfootball.

3.3 Scenariusze

W środowisku [7] zostało zaimplementowanych 19 scenariuszy. Cześć z nich to gra 5 minutowa z zaimplementowanym botem, ale zdecydowana większość to scenariusze akademii, gdzie można trenować specyficzne urywki meczów piłkarskich, jak na przykład rzut różny czy kontratak. Poniżej lista wszystkich dostępnych scenariuszy w gfootball:

1. '11_vs_11_competition',
2. '11_vs_11_stochastic',
3. 'academy_corner',
4. 'academy_empty_goal',
5. 'academy_run_to_score_with_keeper',
6. '11_vs_11_easy_stochastic',

```
7. '1_vs_1_easy',
8. 'academy_counterattack_easy',
9. 'academy_pass_and_shoot_with_keeper',
10. 'academy_single_goal_versus_lazy',
11. '11_vs_11_hard_stochastic',
12. '5_vs_5',
13. 'academy_counterattack_hard',
14. 'academy_run_pass_and_shoot_with_keeper',
15. '11_vs_11_kaggle',
16. 'academy_3_vs_1_with_keeper',
17. 'academy_empty_goal_close',
18. 'academy_run_to_score',
19. 'test_example_multiagent'.
```

Bardzo dokładne opisy scenariuszy można znaleźć w repozytorium [7] w folderze 'scenarios'. Znajduje się tam dokładna konfiguracja początkową środowiska dla każdego scenariusza, czyli:

- wszystkie pozycje zawodników biorących udział w scenariuszu,
- pozycja początkowa piłki,
- poziom trudności bota,
- czas gry,
- true/false czy środowisko jest deterministyczne,
- true/false włączone spalone (domyślnie: true),
- true/false czy zakończyć scenariusz po golu (domyślnie: false),
- true/false czy zakończyć scenariusz po wybiciu piłki poza boisko (domyślnie: false),

- true/false czy zakończyć scenariusz po zabraniu piłki przez przeciwnika (domyślnie: false).

Rozdział 4

Problemy ze środowiskiem gfootball

Pierwszym problemem z gfootball była już sama instalacja. Pomimo wykonania kroków zawartych w 5.1 może zdarzyć się, że użytkownik nie będzie w stanie uruchomić przykładów implementacji uczenia ze wzmacnieniem z repozytorium gfootball [7] ze względu na biblioteki pythonowe i ich złe definicje w projekcie. Otóż brak jest dokładnie zdefiniowanych wymaganych wersji bibliotek, które zaś w swojej historii miały tzw. breaking changes, czyli zmiany niekompatybilne wstecznie. Kolejne poważne problemy dotyczą przedstawionych w projekcie przykładów, które zostaną omówione w 4.1. Ponadto niedogodności z użytką w przykładach biblioteką baselines od OPEN.AI [3] zostaną omówione w 4.2.

4.1 Niekompatybilność z oficjalną dokumentacją - artykułem

Oficjalną dokumentacją projektu jest [8]. Niestety nie jest to rzetelna dokumentacja, gdyż jest to artykuł naukowy. Wysokopoziomowo opisuje on środowisko. Ponadto nie ma tam dokładnego przedstawienia "know how".

Jedynym przykładem, który zgadzał się z artykułem był algorytm PPO2

z biblioteki baselines od OPEN.AI [3], aczkolwiek rozwinięcie tego tematu znajduje się w 4.2. W repozytorium [7] brakowało przykładu z APE-X DQN, co więcej w artykule brakowało informacji z jakiego repozytorium skorzystano podczas badań i gdzie można szukać algorytmu do powtórzenia benchmarku z [8].

Dodatkowo w przykładach IMPALA została zaimplementowana jako polityka do algorytmu, nie został użyty algorytm impala. Polityka została stworzona od zera, nie została ona zainportowana z żadnego repozytorium. Polityka ta polegała na stworzeniu sieci neuronowej, której architektura jest udostępniona w [9].

4.2 Biblioteka Baselines od OpenAI

Wiosną 2020 biblioteka baselines była usunięta z możliwości instalacji przez narzędzie pythonowe "pip". Wzbudziło to wątpliwości dlatego biblioteka została bardzo dokładnie przeanalizowana. Okazało się, że instalacja "pipem" nie działa od 2018 roku. Dodatkowo okazało się, że biblioteka ma bardzo dużo otwartych błędów (w porównaniu do konkurencyjnych bibliotek). Faktem, który przesądził o rezygnacji z tego rozwiązania było to, że biblioteka w momencie rozpoczęcia pracy była w statusie "failing". To znaczy budowa repozytorium nie kończyła się sukcesem. Jak się później okazało od pół roku biblioteka nie zbudowała się z sukcesem.

Rozwiązanie tego problemu zostanie przedstawione w 5. Jest to nowa biblioteka Stable-Baselines [2], która jest rozwidleniem biblioteki baselines od OPEN.AI [3]

Rozdział 5

Nakładka na środowisko

W poprzednim rozdziale zostały omówione problemy ze środowiskiem i brakiem pokrycia badaniami przedstawionych w artykule z kodem udostępnionym w [7]. Rozwiązaniem problemów z brakiem pełnego kodu benchmarku oraz problemów z biblioteką Baselines od Open.AI jest stworzenie własnego repozytorium do badań bazującego na bibliotece Stable-Baselines [2], które zostanie przedstawiona w 5.2. Dodatkowo w 5.1 zostanie opisana instrukcja przygotowania środowiska do przeprowadzania badań na gfootball [7]

5.1 Instalacja środowiska na systemie Linux

Na natywnym systemie Linux, czyli takim, który działa na komputerze bezpośrednio na hardwarze, instalacja przechodzi bezproblemowo. Jednakże jeżeli nie jest się posiadaczem sprzętu z zainstalowanym natywnie Linuxem najczęściej sięga się po rozwiązanie w postaci maszyny wirtualnej, czyli emulatora realnego komputera. System operacyjny na maszynie wirtualnej pracuje tak jakby był zainstalowany na realnym sprzęcie.

Początkowo w niniejszej pracy był używany OracleVM. Szybko okazało się, że maszyna wirtualna Oracle nie jest dopracowana po kątem współpracy z kartami graficznymi, które w uczeniu maszynowym odgrywają kluczową rolę. Rozwiązaniem okazał się VMWare, który umożliwia współpracę z kartą

graficzną oraz wirtualizacje wewnątrz zwirtualizowanego środowiska. Kolejnym napotkanym problemem była niemożliwość przekonwertowania istniejącej maszyny wirtualnej OracleVM na maszynę wirtualną VMWare. Z tego powodu całą instalację i konfigurację środowiska wykonywaną w ramach prowadzonych prac badawczych trzeba było przeprowadzić ponownie.

Po zainstalowaniu maszyny wirtualnej bardzo ważne jest upewnienie się, że python jest zainstalowany w wersji 3.7. W przeciwnym wypadku środowisko nie uruchomi się. Jeżeli maszyna wirtualna ma automatycznie zainstalowanego pythona w wersji 3.8 lub wyższego należy zainstalować wersję 3.7. Następnym krokiem jest instalacja podstawowych narzędzi takich jak git, python dev tools oraz numpy. Można to zrobić za pomocą komendy:

```
sudo apt-get install git python3-dev python3-numpy
```

Następnie zgodnie z instrukcją zawartą w [7] należy zainstalować wszystkie niezbędne narzędzia poniższą komendą:

```
sudo apt-get install git cmake build-essential \
libgl1-mesa-dev libSDL2-dev libSDL2-image-dev \
libSDL2-ttf-dev libSDL2-gfx-dev libboost-all-dev \
libdirectfb-dev libst-dev mesa-utils xvfb x11vnc \
libsdl-sge-dev python3-pip
```

Po wykonaniu wyżej wymienionych kroków środowisko powinno uruchamiać się bezproblemowo.

5.2 Opis Kodu

Z uwagi na nieścisłości pomiędzy [8] oraz [7] powstała dodatkowa nakładka na środowisko gfootball [7]. Nakładka ta została udostępniona w [10]. Składa się ona z dwóch części. Pierwsza część służy do trenowania modeli, zaś druga do ich ewaluacji. Python to język prosty jednakże z poważnym problemem tzw. "wyspami pamięci". Problem ten polega na tym, że śmieciarz pythona (ang. garbage collector) nie układa obiektów w drzewo zależności

tak jak robi to np. Java. Uściślając kiedy obiekt przestaje mieć połączenie z obiektem głównym (ang. rootem) następuje przekazanie odciętych obiektów do śmieciarza. Śmieciarz Pythona polega na sprawdzaniu czy dany obiekt nie ma referencji. Jeśli to założenie okaże się prawdziwe zostaje on usuwany z pamięci. Problem "wysp pamięci" w dużym uproszczeniu polega na tym że różne obiekty mają referencje do siebie nawzajem i tworzą zamknięty cykl referencji. W takim wypadku powstaje tzw. wyspa pamięci i śmieciarz nie usuwa żadnego obiektu. We wcześniejszej wersji pisania kodu występował problem z bardzo wysokim użyciem pamięci RAM, a nawet zamrożenie całego systemu operacyjnego i zabicie procesu uczenia z powodu za wysokiego użycia RAM. Z tego powodu został stworzony skrypt Bash do trenowania wielu modeli naraz, dla dowolnej ilości kroków czasowych oraz różnych scenariuszy. Skrypt Bashowy dodatkowo tworzy strukturę plików, tak aby nawigacja po wytrenowanych modelach była jak najbardziej przejrzysta.

Dodatkowo w kodzie udostępnionym w [10] znajduje się wersja runnera w postaci czysto pythonowej. Można ją znaleźć w 'rl_project\runner.py'.

Obie wersje implementacji uruchamiania kodu korzystają z niemalże identycznego kodu. Dlatego w następnych podrozdziałach 5.2.1 i 5.2.2 zostanie omówiony kod wspólny, zaś jeżeli wystąpią małe różnice, to zostaną one wspomniane lecz nie będą opisane. Kod skryptu bashowego, który zarządza całym cyklem uczenia, został omówiony w 5.2.3.

5.2.1 Trener

Pierwszą częścią implementacji nakładki jest tzw. trener. W wersji z runnerem w pythonie jest on koncepcyjnie taki sam, aczkolwiek z delikatnie inną implementacją. W przypadku treningiem skryptem Bashowym głównym plikiem trenującym jest plik 'baselines_run.py', którego implementacja jest widoczna na rysunku 5.1. Implementacja z runnerem w pythonie znajduje się w pliku 'rl_project\trainer.py'. Częścią wspólną obu implementacji jest kreator modelu algorytmu, który również znajduje się w pliku 'rl_project\trainer.py'. Implementacja kreatora modelu algorytmu jest wi-

doczna na rysunku 5.2.

Patrząc wysokopoziomowo trening polegał na wywołaniu środowiska gfootball w odpowiedniej konfiguracji. Środowisko mogło być wielowątkowe (wektorowe) tzn. mającym wiele równoległych środowisk z tymi samymi ustawieniami. Następnie stworzeniu modelu odpowiedniego algorytmu wraz z przekazaniem do algorytmu wywołanego środowiska. Ostatecznie wytrenowaniu modelu oraz zapisaniu go do pliku.

W tym miejscu należy przywołać kod 'baselines_run.py'. W pierwszych liniach zaraz po importach można zauważać flagi (linie od 5 do 22). Służą one do definiowania parametrów wejściowych z wywoływania programu z linii polecen. Przechodząc dalej w linii 25 zaczyna się metoda main(_), która jest metodą wywoływaną po uruchomieniu pliku pythona. Metoda wewnętrzna 'get_run_name()' w liniach 28-35 na obrazku 5.1 zwraca nazwę służącą do zapisania plików wynikowych oraz logów. Składa się ona z nazwy algorytmu, numeru scenariusza, ilości kroków czasowych oraz ilości środowisk równoległych użytych do treningu. Metoda jest adapterem do metody '_get_run_name()' w '__init__.py' w której zdefiniowane jest dokładnie jak działa generacja nazw. Zrobiono to, aby dwie wersje: z runnerem pythonowym i runnerem bashowym korzystały z tego samego kodu do tworzenia nazw plików.

Następne linijki na rysunku 5.1 37-40 służą do skonfigurowania logowania do pliku oraz zapisaniu, że rozpoczęto proces uczenia. Następnie w liniach 43-50 następuje wywoływanie metody kreacji modelu algorytmu na podstawie danych przekazywanych przez flagi absl. Cała logika wywoływania znajduje się w tej metodzie. Musiała ona zostać wyodrębniona aby móc być współdzielona pomiędzy runnerem pythonowym i bashowym. W linijce 50 można dodatkowo zobaczyć wywołanie metody 'learn(number_of_time_steps)' do modelu algorytmu, która rozpoczyna proces trenowania. Metoda 'learn' jest najbardziej czasochlonną częścią całego kodu.

```

1  import time
2
3  from absl import app
4  from absl import flags
5  from absl import logging as logger
6
7
8  FLAGS = flags.FLAGS
9
10 flags.DEFINE_integer('scenario_number', int(17),
11                      'Defines scenario number - look at Readme.md', int(1), int(19))
12 flags.DEFINE_enum('algorithm', 'PPPO2', ['PPPO2', 'DQN', 'A2C', 'ACER', 'GAIL', 'TRPO'],
13                  'Algorithm to be used for training - only some algorithms from stable-baselines')
14 flags.DEFINE_string('algorithm_policy', 'CnnPolicy',
15                     'The policy model to use (MlpPolicy, CnnPolicy, CnnLstmPolicy...)')
16 flags.DEFINE_integer('number_of_envs', int(8),
17                      'Number of env run parallelly to get result. Available only for ', int(1))
18 flags.DEFINE_integer('number_of_steps', int(1e5),
19                      'Total number of steps of the algorithm', int(1))
20 flags.DEFINE_enum('representation', 'extracted', ['extracted', 'pixels', 'pixels_gray', 'simple115v2'],
21                  'Definition of the representation used to build the observation. More info in gfootball.env.__init__')
22 flags.DEFINE_boolean('stacked', True,
23                      'If True, stack 4 observations, otherwise, only the last observation is returned '
24                      'by the environment. Stacking is only possible when representation is one of the '
25                      'following: "pixels", "pixels_gray" or "extracted"')
26
27 def main():
28     def get_run_name() -> str:
29         from rl_project import _get_run_name
30         return _get_run_name(
31             algorithm=FLAGS.algorithm,
32             scenario_number=FLAGS.scenario_number,
33             number_of_steps=FLAGS.number_of_steps,
34             number_of_envs=FLAGS.number_of_envs,
35         )
36
37     run_name = get_run_name()
38     logger.get_absl_handler().use_absl_log_file(run_name, './')
39     logger.info("STARTED")
40     time_start = time.time()
41
42     from rl_project.trainer import create_rl_algorithm_model
43     trained_model = create_rl_algorithm_model(
44         algorithm=FLAGS.algorithm,
45         algorithm_policy=FLAGS.algorithm_policy,
46         scenario_number=FLAGS.scenario_number,
47         number_of_envs=FLAGS.number_of_envs,
48         representation=FLAGS.representation,
49         stacked=FLAGS.stacked,
50     ).learn(FLAGS.number_of_steps)
51     time_elapsed = time.time() - time_start
52     logger.info("Time elapsed " + time.strftime("%H:%M:%S", time.gmtime(time_elapsed)))
53     trained_model.save(save_path=run_name)
54
55
56 if __name__ == '__main__':
57     app.run(main)
58

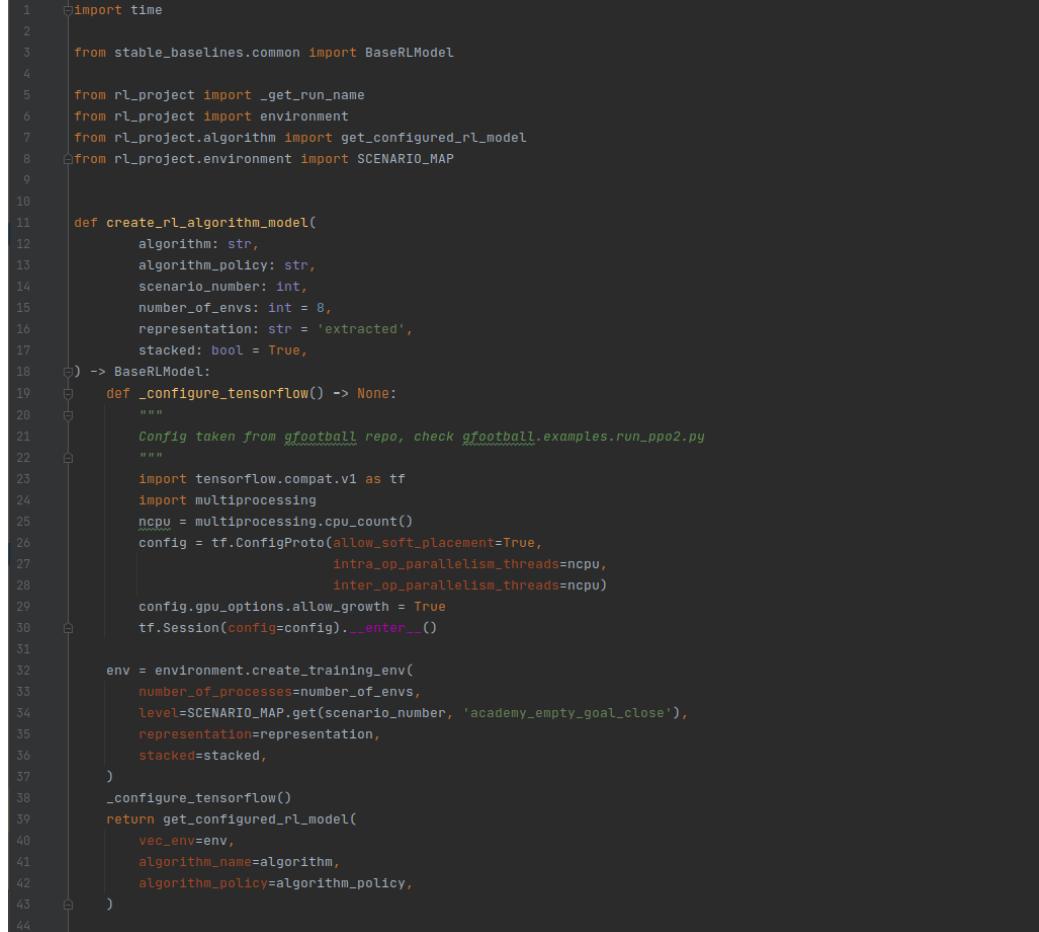
```

Rysunek 5.1: Implementacja trenera w pliku 'baselines_run.py'

Po wytrenowaniu modelu w linijkach 51-52 następuje zapisanie czasu i zalogowanie go. Ostatecznie w linijce 53 następuje zapisanie modelu w postaci pliku pod nazwą wygenerowaną przez metodę '_get_run_name()' w folderze

w którym zostało włączone wykonywanie pliku pythonowego.

Wracając do linijek 43-50 na rysunku 5.1. Jest to wywołanie metody kreacji modelu, która przedstawiona na rysunku 5.2.



```
1 import time
2
3 from stable_baselines.common import BaseRLModel
4
5 from rl_project import _get_run_name
6 from rl_project import environment
7 from rl_project.algorithm import get_configured_rl_model
8 from rl_project.environment import SCENARIO_MAP
9
10
11 def create_rl_algorithm_model(
12     algorithm: str,
13     algorithm_policy: str,
14     scenario_number: int,
15     number_of_envs: int = 8,
16     representation: str = 'extracted',
17     stacked: bool = True,
18 ) -> BaseRLModel:
19     def __configure_tensorflow() -> None:
20         """
21             Config taken from gfootball repo, check gfootball/examples/run_ppo2.py
22         """
23         import tensorflow.compat.v1 as tf
24         import multiprocessing
25         ncpu = multiprocessing.cpu_count()
26         config = tf.ConfigProto(allow_soft_placement=True,
27                                intra_op_parallelism_threads=ncpu,
28                                inter_op_parallelism_threads=ncpu)
29         config.gpu_options.allow_growth = True
30         tf.Session(config=config).__enter__()
31
32         env = environment.create_training_env(
33             number_of_processes=number_of_envs,
34             level=SCENARIO_MAP.get(scenario_number, 'academy_empty_goal_close'),
35             representation=representation,
36             stacked=stacked,
37         )
38         __configure_tensorflow()
39         return get_configured_rl_model(
40             vec_env=env,
41             algorithm_name=algorithm,
42             algorithm_policy=algorithm_policy,
43         )
44     
```

Rysunek 5.2: Implementacja kreatora modelu algorytmu w pliku 'rl_project\trainer.py'

W linijkach 19-30 została zdefiniowana wewnętrzna metoda 'configure_tensorflow()', została ona skopiowana z części kodu udostępnionego w [7]. Powodem skopiowania, a nie użycia jest zły design kodu w repo [7], gdyż jako przykład jest tam przedstawione użycie bez wydzielenia używalnych metod. Metoda ta jest wywoływana w linijce 38, po kreacji śro-

dowiska treningowego zgodnie z koncepcją z [7].

W linijkach 32-37 znajduje się kreacja środowiska testowego. Jest to wywołanie adaptera do [7]. Jest to zwykły adapter z przepisaniem parametrów.

W linijkach 39-43 jest wywoływany adapter do stworzenia modelu uczenia ze wzmacnieniem. Nie został on przedstawiony ze względu na swoją prostotę. Jest to pewnego rodzaju mapowanie, które na podstawie nazwy algorytmu tworzy odpowiedni model. Dodatkowo ustawia modelowi środowisko w jakim będzie się uczył.

5.2.2 Evaluator

Sposób wywoływania evaluatora z kodu pythonowego i bashowego delikatnie się różni. Różnica to implementacja logowania. W przypadku treningu skryptem bashowym głównym plikiem trenującym jest plik 'baselines_evaluator.py', którego implementacja jest widoczna na rysunku 5.3. Implementacja z runnerem w pythonie znajduje się w pliku 'rl_project\evalutor.py'. Częścią wspólną obu tych implementacji jest metoda 'evaluate_model()' w pliku 'rl_project\evalutor.py', która została przedstawiona na rysunku 5.4.

Patrząc wysokopoziomowo ewaluacja polegała na wywołaniu z odpowiednimi ustawieniami środowiska gfootball. Środowisko wywoływanie jest tylko z użyciem jednego wątku i przeprowadzeniu na nim X tzw. runów (uruchomień), czyli przeprowadzeniu scenariusza od początku do końca. O ilości uruchomień decyduje zmienna 'ACCURACY'. Funkcją nagrody jest 'only scoring'. Po każdym uruchomieniu jest zapisywana różnica bramek obu drużyn. Ostatecznie wyniki wszystkich uruchomień sumuje się co daje wynik ewaluacji.

Wałą informacją w przypadku evaluatora jest to, że tworzy on środowisko, w którym funkcja nagrody to sam 'scoring'. Jest to konieczne, gdyż do porównywania modeli uczenia przez wzmacnianie nie należy używać funkcji pomocniczych, tylko tych, które jasno definiują pożądany rezultat.

Wracając do kodu 'baselines_evalutor.py'. W pierwszych liniach zaraz po

importach można zauważyc flagi (linie od 5 do 22). Flagi służą do definiowania parametrów wejściowych z wywoływania programu z linii poleceń. Od linijki 25 zaczyna się główna metoda evaluatora. Najpierw w linijkach od 26 do 31 zdefiniowana jest metoda 'get_run_name()', która podobnie jak w przypadku runnera zwraca nazwę danego uruchomienia skryptu, służącą do zapisywania logów.

```

1  from absl import app
2  from absl import flags
3  from absl import logging as logger
4
5  FLAGS = flags.FLAGS
6
7  flags.DEFINE_enum('algorithm', 'PPO2', ['PPO2', 'DQN', 'A2C', 'ACER', 'GAIL', 'TRPO'],
8                  'Algorithm used for model training - only some algorithms from stable-baselines')
9  flags.DEFINE_string('path', None, 'path to stored model')
10 flags.DEFINE_integer('scenario_number', int(17),
11                      'Defines scenario number (look at Readme.md), which will be used for evaluation', int(1), int(19))
12 flags.DEFINE_integer('accuracy', int(1e3),
13                      'Number of runs to evaluate model correctness', int(1))
14 flags.DEFINE_boolean('render', False,
15                      'if enabled then game screen pop up and you can observe how agent behaves')
16 flags.DEFINE_enum('reward', 'scoring', ['scoring', 'checkpoints', 'scoring,checkpoints'],
17                   'Option defines how each run should be rewarded')
18 flags.DEFINE_enum('representation', 'extracted', ['extracted', 'pixels', 'pixels_gray', 'simple115v2'],
19                   'Definition of the representation used to build the observation. More info in gfootball.env._init_')
20 flags.DEFINE_boolean('stacked', True, 'If True, stack 4 observations, otherwise, only the last observation is returned
21                      by the environment. Stacking is only possible when representation is one of the
22                      following: "pixels", "pixels_gray" or "extracted")
23
24
25 def main():
26     def get_run_name() -> str:
27         return "EVALUATOR_{algorithm}_{scenario}-{accuracy}-{accuracy}".format(
28             algorithm=FLAGS.algorithm,
29             scenario=FLAGS.scenario_number,
30             accuracy=str(FLAGS.accuracy),
31         )
32
33     if FLAGS.path is None:
34         raise ValueError("path to trained model must be given. Please run script with --help option")
35     logger.get_absl_handler().use_absl_log_file(get_run_name(), './')
36     from rl_project.algorithim import load_model
37     model = load_model(FLAGS.path, FLAGS.algorithm)
38     from rl_project.evaluator import evaluate_model
39     total_reward = evaluate_model(
40         model=model,
41         scenario_number=FLAGS.scenario_number,
42         accuracy=FLAGS.accuracy,
43         reward=FLAGS.reward,
44         representation=FLAGS.representation,
45         stacked=FLAGS.stacked,
46         render=FLAGS.render,
47         logging=logger.info,
48     )
49     logger.info(
50         "EVALUATOR: ended with {runs} runs and receive in total reward: {reward}. Average reward: {average}".format(
51             runs=FLAGS.accuracy,
52             reward=total_reward,
53             average=total_reward / FLAGS.accuracy,
54         ))
55
56
57 if __name__ == '__main__':
58     app.run(main)
59

```

Rysunek 5.3: Implementacja evaluatora w pliku 'baselines_evaluator.py'

```

1  from stable_baselines.common import BaseRLModel
2
3  from rl_project import _get_run_name
4  from rl_project.environment import create_demo_env, SCENARIO_MAP
5
6
7  def evaluate_model(
8      model: BaseRLModel,
9      scenario_number: int,
10     accuracy: int,
11     logging,
12     reward: str = 'scoring',
13     representation: str = 'extracted',
14     stacked: bool = True,
15     render: bool = False,
16 ) -> int:
17     """
18         Function returns average goal difference after number of scenario runs - accuracy parameter
19     """
20
21     if model is None:
22         raise ValueError("Trained model must be given")
23     logging("EVALUATOR STARTED")
24     env = create_demo_env(
25         level=SCENARIO_MAP.get(scenario_number, 'academy_empty_goal_close'),
26         reward_experiment=reward,
27         representation=representation,
28         stacked=stacked,
29         render=render,
30     )
31     number_of_runs = int(0)
32     total_reward = float(0)
33     while number_of_runs < accuracy:
34         obs = env.reset()
35         index = 0
36         done = False
37         rewards = float(0.0)
38         while not done:
39             index = index + 1
40             action, _states = model.predict(obs)
41             obs, rewards, done, info = env.step(action)
42             logging("EVALUATOR: run-{run} ended with reward: {reward}".format(
43                 run=number_of_runs,
44                 reward=rewards,
45             ))
46             total_reward = total_reward + rewards
47             number_of_runs = number_of_runs + 1
48     return total_reward
49

```

Rysunek 5.4: Implementacja ewaluatora modelu w pliku 'rl_project\evaluator.py'

W linijkach 33-34 jest sprawdzenie czy ścieżka do modelu została podana jako argument do programu. Następnie w linijce 35 następuje zdefiniowanie loggera. W linijce 37 następuje wczytanie modelu z pliku podanego jako parametr wejściowy poprzez adapter, który na podstawie nazwy algorytmu dobiera odpowiedni konstruktor algorytmu z biblioteki stable-baselines. Za część obliczeniową ewaluatora odpowiada metoda 'evaluate_model', która

odpowiada za ewaluację modelu i zapisanie w logach wszystkich otrzymanych wyników. Implementacja metody 'evaluate_model' została zaprezentowana na rysunku 5.4. Ostatnią częścią omawianej metody jest zapisanie końca ewaluacji i jej sumarycznego wyniku w linijkach 48-54.

Wchodząc w szczegóły metody 'evaluate_model' z pliku 'rl_project\evalutor.py', która jest widoczna na rysunku 5.4, rozpoczyna się ewaluację czy model istnieje, w przeciwnym razie jest rzucany błąd (liniki 21-22). Następnie w linijce 23 następuje zapis do logów informacji o starcie ewaluacji. W linijkach 24-30 następuje stworzenie środowiska gfootball. Jest to zrobione podobnie jak w przypadku runnera i tak jak w tamtym przypadku nie zostanie opisane, jest to bardzo prosty adapter służący do wywoływania metod udostępnionych w [7].

Następnie w wierszach 31-47 następuje pętla, w której na żądaną ilość razy określoną parametrem 'accuracy' zostaje wykonane kolejno:

1. utworzenie za każdym razem nowego środowiska - linijka 34,
2. utworzenie parametrów pomocniczych - linijki 35-37,
3. pod-pętla sterująca środowiskiem do skończenia scenariusza składająca się z:
 - (a) obliczenia następnej akcji na podstawie obecnego stanu środowiska - linijka 40,
 - (b) wykonania kroku na podstawie wcześniej wyliczonej akcji oraz nadpisanie zmiennej 'done' w celu sprawdzenia czy scenariusz został zakończony,
4. zalogowanie wyniku rozegranego scenariusza - linijki 42-45,
5. dodanie do sumy nagród zdobytej nagrody w obecnym scenariuszu.

W linijce 48 zostaje zwrócony ostateczny wynik ewaluacji, czyli suma wszystkich zdobytych przez dany model nagród.

5.2.3 Skrypt Bash

Skrypt Bash został zaprezentowany na obrazku 5.5. Niestety skrypty bashowe nie są zbyt czytelne, dlatego też w ramach ułatwienia korzystania ze skryptu wszystkie potrzebne zmienne zostały wyodrębnione i zapisane w liniach 1-5. Są to:

- Nazwa algorytmu - ALGORITHM NAME,
- Lista numerów scenariuszy - SCENARIOS,
- Liczba środowisk uruchamianych równolegle do treningu - NUMBER_OF_PARALLEL_ENVS,
- Lista kroków czasowych treningów w jakich algorytm będzie uruchamiany - TIME_STEPS.

Skrypt oprócz uruchamiania treningów tworzy także strukturę plików. Jako folder główny jest ustawiana kombinacja nazwy algorytmu oraz numeru scenariusza. W środku zaś umieszczone są dodatkowe foldery o nazwach ilości kroków czasowych. W każdym z tych folderów znajduje się wytrenowany model oraz powstałe przy procesie logi.

```
1 #!/bin/bash
2
3 ALGORITHM_NAME="PP02"
4 SCENARIOS="13 14 17"
5 NUMBER_OF_PARALLEL_ENVS="8"
6 EVALUATION_ACCURACY="1000"
7 TIME_STEPS="10000 20000 50000 100000 200000 500000 1000000 2000000 5000000"
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

Rysunek 5.5: Skrypt bash do uruchamiania wielu treningów wraz z ewaluacjami

5.3 Implementacja w COLAB

Colab jest to środowisko stworzone przez google research do trenowania modeli w pythonie z dziedziny uczenia maszynowego. Środowisko to jest bardzo elastyczne i pozwala na uruchamianie go z akceleracją GPU. Dlatego też do prostego ponownego użycia stworzonego podczas pisania tej pracy kodu źródłowego zostało stworzony plik COLAB udostępniony w [10]. Darmowa wersja COLAB jest stosunkowo powolna w porównaniu z przeciętnym komputerem programisty, co więcej jest zależna od obciążenia serwerów COLAB. Innym problemem trenowania modeli w środowisku COLAB jest to, iż wytrenowane modele są bezpowrotnie usuwane gdy zostanie rozłączona sesja. Problem ten uczynił COLAB bezużytecznym w badaniach omówionych szczegółowo w 6. Zwłaszcza, że czas treningu wynosi od kilku do kilkunastu godzin.

Plik Jupyter (używany w COLAB) został podzielony na 3 części:

1. Initial Setup - po uruchomieniu środowiska Colab należy je odpowiednio skonfigurować. Należy to zrobić w następujących krokach:
 - (a) 'Install all linux required packages' - instalacja wszystkich wymaganych paczek do podstawowego obrazu Linux dostarczonego przez Colab,
 - (b) 'Clone github repository to /content directory and install all packages' - sklonowanie z repozytorium github kodu stworzonego na potrzebę tej pracy.
2. 'Train model' - sekcja ta zawiera dokładne instrukcje jak uruchamiać kod stworzony w [10] do wytrenowania modelu algorytmem ze stable-baselines, [2]
3. 'Result Model validation' - sekcja ta zawiera dokładne instrukcje jak uruchamiać ewaluacje stworzonego modelu w poprzedniej sekcji.

Rozdział 6

Badania

W rozdziale tym zostaną omówione badania prowadzone w środowisku gfootball. Najpierw zostanie przeanalizowana funkcja nagrody i porównanie następujących modeli nagród: 'scoring' oraz 'scoring,checkpoints'. Następnie zostanie zbadany wpływ wysokości nagrody 'checkpoints' na szybkość uczenia się algorytmu. Trzecią częścią będzie przedstawienie algorytmów dostępnych w bibliotece stable-baselines [11] oraz przebadanie ich pod kątem prędkości uczenia się algorytmu. Ostatnia - czwarta część badań jest badaniem wpływu wartości hiper-parametrów na prędkość uczenia algorytmu na przykładzie algorytmu PPO2.

Dokładne wyniki badań dla podrozdziałów 2, 3, 4 zostały przedstawione w dodatkach B, C oraz D w postaci tabel. Znajdują się tam informacje na temat wysokości wyniku ewaluacji modelu oraz czasu trwania obliczeń.

Badania przeprowadzone zostało na następującym sprzęcie:

- Płyta główna: ASUS PRIME Z370-P
- Procesor: Intel Core i7-8700
- Pamięć RAM: 32GB DDR4, 3000MHZ CL16
- Karta graficzna: NVIDIA GeForce GTX 1060 (6144 MB GDDR5)

Obliczenia były uruchamiane na maszynie wirtualnej VMware przez program 'VMware Workstation 16 Player' z przeznaczeniem:

- połowy wątków procesora, czyli 6 na maszynę wirtualną,
- połową pamięci RAM, czyli 16GB,
- połową pamięci karty graficznej, czyli 3GB.

Na maszynie wirtualnej był zainstalowany system operacyjny Ubuntu 18.04.

6.1 Porównanie funkcji nagrody

W tym podrozdziale zostanie porównanie funkcji nagrody bazującej tylko na zdobytych bramkach (flaga reward = 'scoring') oraz funkcji nagrody bazującej na zdobytych bramkach wraz z uwzględnieniem przesunięcia się piłki w stronę bramki drużyny przeciwej (flaga reward = 'scoring,checkpoints'). Będzie to częściowe powtórzenie benchmarku z [8].

Funkcja nagrody bazująca na samych checkpointach nie ma sensu, gdyż wtedy środowisko nie zwraca informacji o najważniejszej nagrodzie, czyli zdobyciu bramki. Z tego powodu poniżej zostaną przeanalizowane tylko i wyłącznie dwa modele. Poza tym w środowisku nie ma możliwości stworzenia nagrody tylko z 'checkpoints'.

Porównanie to zostanie przeprowadzone przy użyciu algorytmu PPO2 z biblioteki stable-baselines, z użyciem hiperparametrów udostępnionych w kodzie z przykładami w [7], z jedną różnicą we wspólniku vf_coefficient, którego wartość wynosiła 1, a nie jak to było oryginalnie 0,5. Różnica ta nastąpiła w wyniku zwykłej pomyłki, jednakże nie wpływa w żaden sposób na jakość tego badania.

Badanie będzie opierać się na scenariuszach:

1. 'Sam na sam' - numer scenariusza: 17,
2. 'Kontratak' - numer scenariusza: 13,
3. 'Podaj biegij strzel' - numer scenariusza: 14.

Scenariusze, te zostały wybrane, gdyż reprezentują stosunkowo zróżnicowane poziomy trudności.

Badanie będzie polegało na wytrenowaniu modelu dla wyżej wymienionych scenariuszy. Każdy scenariusz będzie posiadał kilka modeli, które będą różnić się ilością kroków. Dla każdego z nich zostanie zewaluowany nauczony model z dokładnością równą 1000, Ewaluacja została opisana w 5.2.2. Ilość kroków dla każdego ze scenariuszy będzie następująca: 0,01M, 0,02M, 0,05M, 0,1M, 0,2M 0,5M, 1M, 2M, 5M (gdzie M = 1 milion kroków).

W tabelach 6.1 oraz 6.2 zostały przedstawione wyniki badania dla scenariusza 17. Dodatkowo dane dla scenariusza 17 zostały skumulowane na wykresie 6.1. W tabelach 6.3 oraz 6.4 znajdują się wyniki badania dla scenariusza 14, ich dane skumulowane zostały przedstawione na wykresie 6.2. Ostatnia część badania dotyczy scenariusza 13 i wyniki zostały przedstawione w tabelach 6.5 oraz 6.6. Skumulowane dane zostały przedstawione na wykresie 6.3.

Jak można łatwo zauważyc na wykresach 6.1, 6.2 i 6.3 nagroda typu 'checkpoints', a dokładniej 'scoring,checkpoints' osiąga znacznie lepsze rezultaty. Co więcej w scenariuszach 13 i 14, nawet po 5M kroków czasowych algorytm PPO2 w opcji nagrody 'scoring' nie był w stanie osiągnąć nawet 10% średniej różnicy goli po 1000 epizodach.

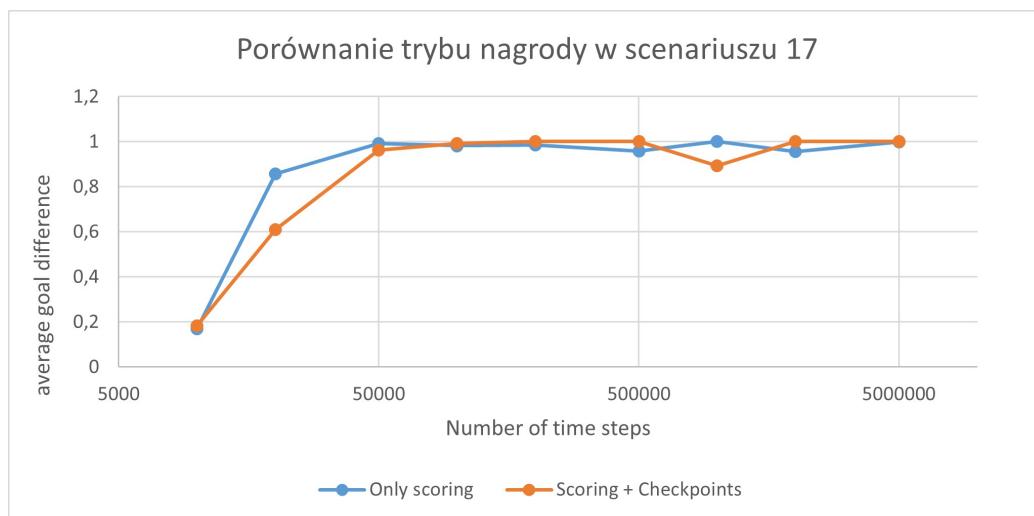
Wniosek z tego badania jest bardzo prosty. Nadając małe nagrody, które w pewnym sensie nakierowują algorytm w kierunku "główniej nagrody", możemy znacząco przyspieszyć proces uczenia algorytmu. W omawianym przypadku nie można dokładnie powiedzieć o ile proces uczenia został przyspieszony w scenariuszach 13 i 14, gdyż bez drobnych nagród algorytm nie był w stanie nauczyć się jak dojść do nagrody głównej. Ciekawym jest fakt, że przy prostych scenariuszach takich jak scenariusz 17 brak drobnych nagród zwiększał prędkość nauki algorytmu.

Tablica 6.1: Wyniki ewaluacji modeli algorytmem PPO2, dla nagrody 'scoring', dla scenariusza 17

ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,01M	0:00:46	0,169
0,02M	0:01:26	0,858
0,05M	0:03:49	0,993
0,1M	0:08:05	0,982
0,2M	0:15:59	0,986
0,5M	0:40:53	0,959
1M	1:26:20	1,0
2M	2:44:08	0,957
5M	6:51:27	0,999

Tablica 6.2: Wyniki ewaluacji modeli algorytmem PPO2, dla nagrody 'scoring,checkpoints', dla scenariusza 17

ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,01M	0:00:41	0,184
0,02M	0:01:20	0,609
0,05M	0:03:34	0,962
0,1M	0:07:19	0,992
0,2M	0:16:06	1
0,5M	0:40:41	1
1M	1:23:40	0,894
2M	2:45:03	1
5M	7:01:49	1



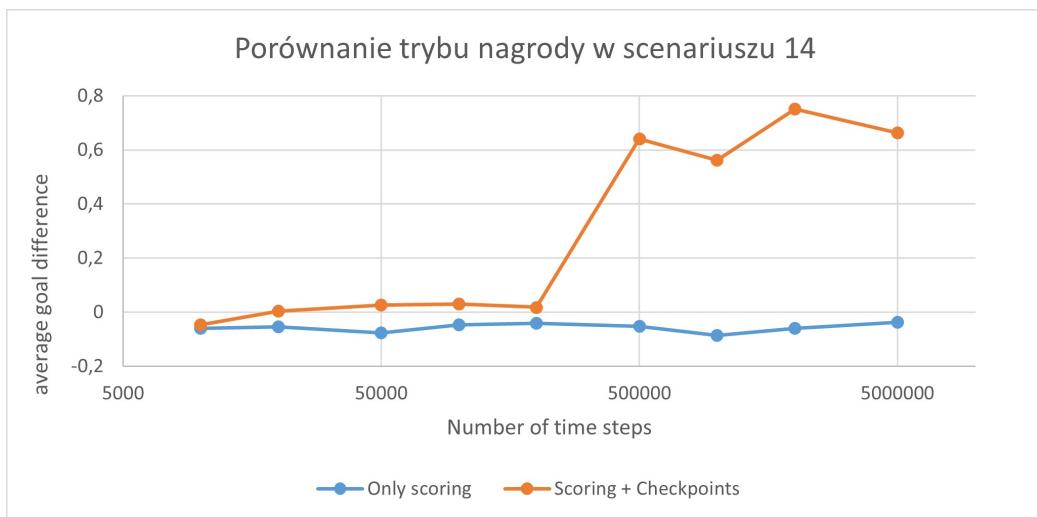
Rysunek 6.1: Porównanie funkcji nagrody 'scoring' i 'checkpoints' dla scenariusza 17

Tablica 6.3: Wyniki ewaluacji modeli algorytmem PPO2, dla nagrody 'scoring', dla scenariusza 14

ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,01M	0:00:40	-0,06
0,02M	0:01:21	-0,055
0,05M	0:03:23	-0,076
0,1M	0:06:47	-0,047
0,2M	0:13:17	-0,041
0,5M	0:33:12	-0,052
1M	1:13:58	-0,085
2M	2:22:09	-0,059
5M	5:51:24	-0,038

Tablica 6.4: Wyniki ewaluacji modeli algorytmem PPO2, dla nagrody 'scoring,checkpoints', dla scenariusza 14

ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,01M	0:00:41	-0,047
0,02M	0:01:22	0,004
0,05M	0:03:29	0,026
0,1M	0:06:52	0,03
0,2M	0:13:50	0,018
0,5M	0:36:38	0,64
1M	1:13:32	0,562
2M	2:24:07	0,752
5M	6:00:51	0,663



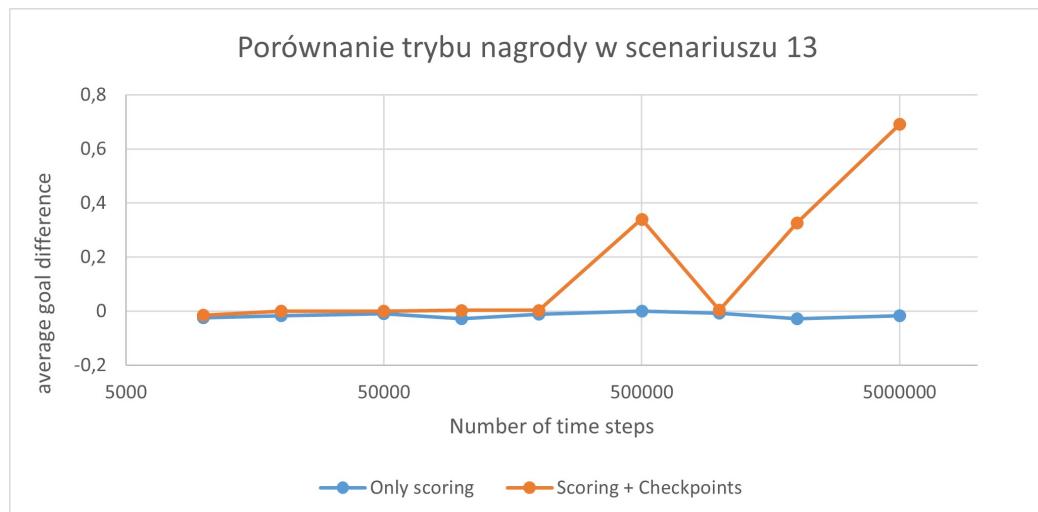
Rysunek 6.2: Porównanie funkcji nagrody 'scoring' i 'checkpoints' dla scenariusza 14

Tablica 6.5: Wyniki ewaluacji modeli algorytmem PPO2, dla nagrody 'scoring', dla scenariusza 13

ilosc krokow	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,01M	0:00:56	-0,025
0,02M	0:01:54	-0,016
0,05M	0:04:34	-0,01
0,1M	0:09:20	-0,028
0,2M	0:18:34	-0,011
0,5M	0:47:34	0,001
1M	1:28:22	-0,007
2M	3:00:32	-0,028
5M	7:24:09	-0,017

Tablica 6.6: Wyniki ewaluacji modeli algorytmem PPO2, dla nagrody 'scoring,checkpoints', dla scenariusza 13

ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,01M	0:00:53	-0,015
0,02M	0:01:50	0
0,05M	0:04:25	0
0,1M	0:09:14	0,003
0,2M	0:18:27	0,003
0,5M	0:44:58	0,34
1M	1:30:00	0,004
2M	2:56:08	0,327
5M	7:25:36	0,692



Rysunek 6.3: Porównanie funkcji nagrody 'scoring' i 'checkpoints' dla scenariusza 13

6.2 Badanie wpływu wysokości nagrody pomocniczej na prędkość uczenia

W tym podrozdziale zostanie omówiony wpływ wysokości nagrody cząstkowej typu 'checkpoints' na prędkość uczenia. Badanie to odbyło się dla scenariuszy:

1. 'Sam na sam' - numer scenariusza: 17,
2. 'Kontratak' - numer scenariusza: 13,
3. 'Podaj biegnij strzel' - numer scenariusza: 14.

Dla każdego ze scenariuszy zostały stworzone modele z następującą ilością kroków czasowych: 0,01M, 0,02M, 0,05M, 0,1M, 0,2M, 0,5M, 1M, 2M, 5M (gdzie M = 1 milion kroków).

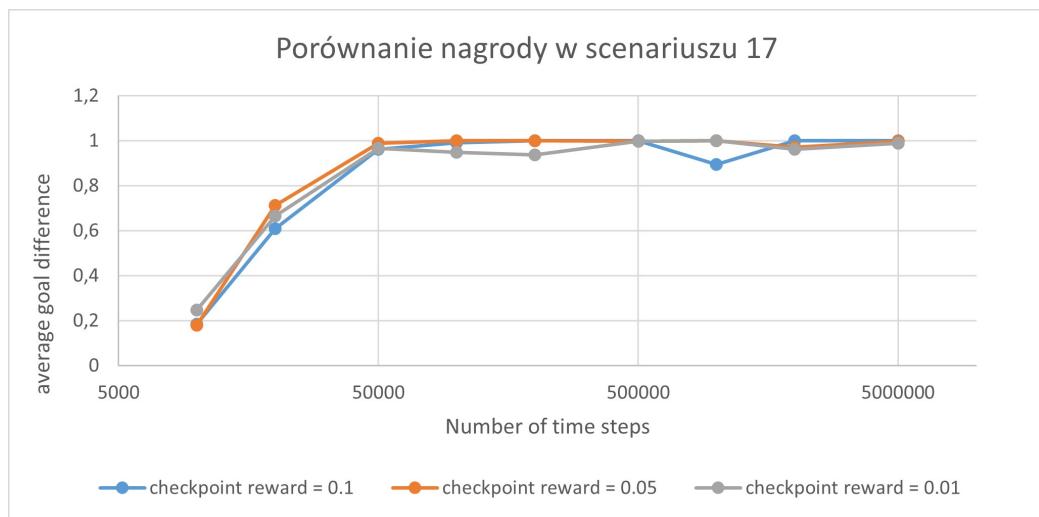
Nagroda typu checkpoints polega na podzieleniu boiska na 10 części i nadawaniu nagrody za każdy zdobyty checkpoint. Wysokość nagrody za jeden checkpoint oryginalnie wynosiła 0,1. W badaniu tym zostały także przebadane dwie wartości: 0,05 oraz 0,01. Wszystkie wyniki tego badania zostały przedstawione w dodatku B w postaci tabel. Zagregowane dane zostały przedstawione na wykresach 6.4, 6.5 oraz 6.6.

Jak możemy zauważyc na wykresie 6.4 w przypadku najprostszego scenariusza (17) wysokość nagrody cząstkowej nie odgrywa za dużej roli. Prawdopodobnie spowodowane jest to małym wpływem nagrody typu 'checkpoints' w tym scenariuszu.

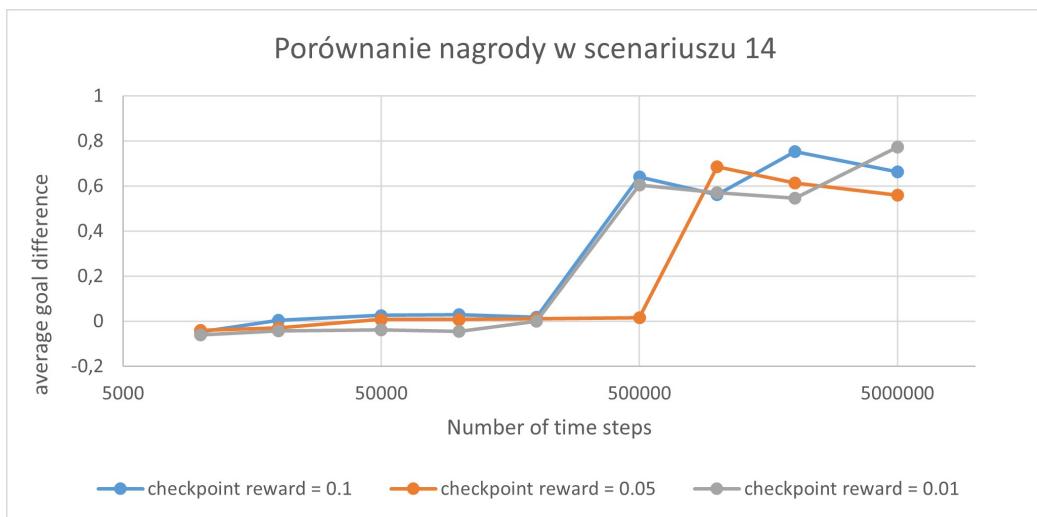
W przypadku scenariusza bardziej złożonego (14) widzimy na wykresie 6.5, że dla nagrody cząstkowej 0,1 i 0,01 nie ma zbyt dużej różnicy w prędkości uczenia. Natomiast widzimy różnicę dla nagrody cząstkowej wynoszącej 0,05. W tym przypadku skok średniej różnicy goli następuje po dwukrotnie większej ilości kroków czasowych. Najprawdopodobniej zjawisko to wynika z błędu statycznego, gdyż wysokość średniej różnicy goli dla nagrody cząstkowej wynoszącej 0,05 powinien być interpolacją średniej różnicy goli dla nagród cząstkowych 0,1 i 0,01.

Ostatnim zbadanym scenariuszem jest scenariusz 13, który jest także najbardziej zaawansowany. Jak można łatwo zauważyć na wykresie 6.6 w przypadku najmniejszej nagrody cząstkowej wynoszącej 0,01 nawet po 5M kroków nie udało się uzyskać więcej niż 0,005 średniej różnicy goli, czyli 5 razy zdobyć bramkę na 1000 przetworzonych prób scenariusza. W przypadku nagrody cząstkowej wynoszącej 0,1 oraz 0,05 prędkość uczenia była porównywalna, aczkolwiek minimalnie wyższa była w tym pierwszym wariantie.

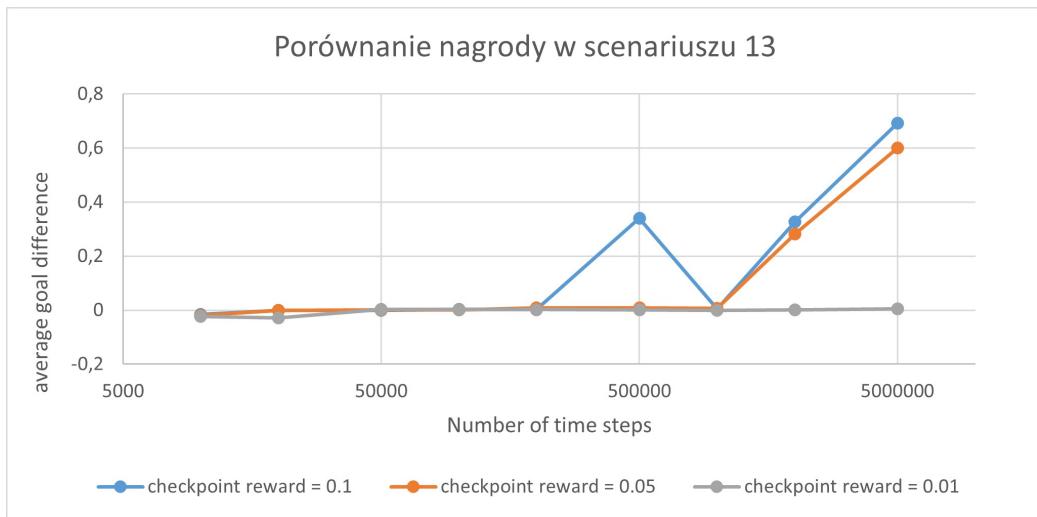
Wniosek z tego badania jest taki, że nagrody cząstkowe, które nakierowują algorytm na znajdywanie rozwiązania nie mogą być zbyt niskie, gdyż w przeciwnym razie algorytm nie zwraca na nie uwagi i nie dochodzi do rozwiązania problemu w racjonalnym czasie.



Rysunek 6.4: Porównanie prędkości uczenia dla różnych wysokości nagrody cząstkowej 'checkpoints' dla scenariusza 17



Rysunek 6.5: Porównanie prędkości uczenia dla różnych wysokości nagrody cząstkowej 'checkpoints' dla scenariusza 14



Rysunek 6.6: Porównanie prędkości uczenia dla różnych wysokości nagrody cząstkowej 'checkpoints' dla scenariusza 13

6.3 Porównanie algorytmów dostępnych w bibliotece stable-baselines

W tym podrozdziale zostanie omówiona szybkość uczenia dla różnych algorytmów dostępnych w bibliotece stable-baselines [2]. Wszystkie dostępne algorytmy z biblioteki stable-baselines zostały przedstawione na rysunku 6.7.

Name	Refactored [1]	Recurrent	Box	Discrete	Multi Processing
A2C	✓	✓	✓	✓	✓
ACER	✓	✓	✗ [4]	✓	✓
ACKTR	✓	✓	✓	✓	✓
DDPG	✓	✗	✓	✗	✓ [3]
DQN	✓	✗	✗	✓	✗
HER	✓	✗	✓	✓	✗
GAIL [2]	✓	✓	✓	✓	✓ [3]
PPO1	✓	✗	✓	✓	✓ [3]
PPO2	✓	✓	✓	✓	✓
SAC	✓	✗	✓	✗	✗
TD3	✓	✗	✓	✗	✗
TRPO	✓	✗	✓	✓	✓ [3]

Rysunek 6.7: Algorytmy dostępne w bibliotece stable-baselines, źródło: [11]

Środowisko akceptuje tylko akcje typu dyskretnego (eng. discrete), nie pudelkowego (eng. box). Z tego powodu nie można było implementować żadnego algorytmu który nie akceptował typu dyskretnego akcji, czyli: DDPG, SAC i TD3. Dodatkowo nie ma sensu implementować algorytmów, które nie wspierają wielowątkowości (eng. multi processing) gdyż czas obliczeniowy nie jest skalowalny i trwa za długo aby można było przeprowadzić racjonalne badanie. Niestety część z tych algorytmów wspiera wielowątkowość poprzez MPI,

z tego powodu również nie zostały wybrane do badania algorytmy DDPG, GAIL, PPO1 oraz TRPO.

Z przedstawionych powodów do dalszych badań zostały wybrane następujące algorytmy: A2C, ACER, ACKTR oraz PPO2. Niestety sam czas badania algorytmu jest stosunkowo długi (do 72h, jeżeli nie wystąpią błędy), nie licząc czytania dokumentacji i zimplementowania go w kodzie. Wszystkie te algorytmy są z rodziny tzw. modelu z aktorem (eng. Actor Critic Model). Dlatego z uwagi na powyższe argumenty zdecydowano o przebadaniu następujących algorytmów: A2C, ACER oraz PPO2. Dodatkowo częściowo został przebadany algorytm DQN ze względu na największą popularność, aczkolwiek proces uczenia algorytmu jednowątkowego jest tak powolny, że ze względu na ograniczenia czasowe eksperyment został przerwany. Okres w jakim algorytm przerabiał 1 milion kroków czasowych to prawie 12h. Przy takim tempie pełny eksperyment potrwał by około miesiąca.

Hiper-parametry algorytmu PPO2 zostały przedstawione w dodatku A. Do badań zostały użyte hiper-parametry domyślne dla biblioteki stable-baselines (tabela 3 w dodatku). Hiper-parametry pozostałych algorytmów były domyślnymi parametrami biblioteki stable-baselines.

Badanie to odbyło się dla scenariuszy:

1. 'Sam na sam' - numer scenariusza: 17,
2. 'Kontratak' - numer scenariusza: 13,
3. 'Podaj biegij strzel' - numer scenariusza: 14.

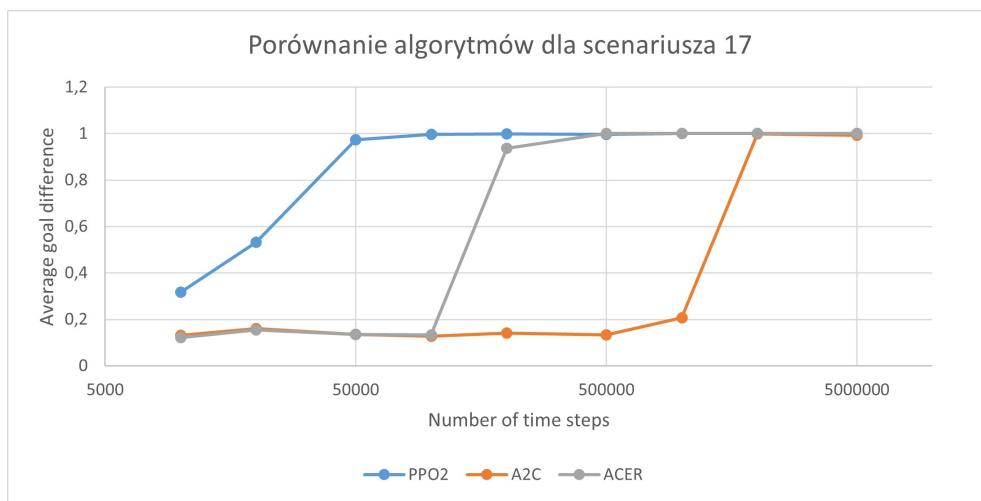
Dla każdego ze scenariuszy zostały stworzone modele z następującą ilością kroków czasowych: 0,01M, 0,02M, 0,05M, 0,1M, 0,2M 0,5M, 1M, 2M, 5M (gdzie M = 1 milion kroków). Szczegółowe wyniki wszystkich badań zostały zamieszczone w dodatku C. Wyniki zostały skumulowane i przedstawione na wykresach 6.8, 6.9 oraz 6.10.

Po analizie wykresów przedstawiających badania dla określonego scenariusza można zauważyc, że najbardziej skutecznym algorytmem z parametrami domyślnymi jest PPO2. Dlatego też algorytm PPO2 będzie w dalszej

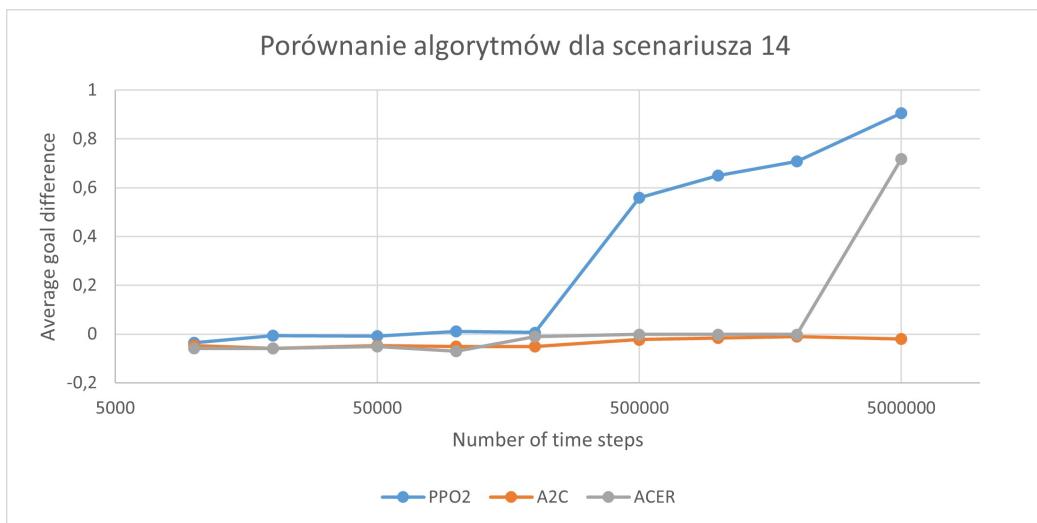
części tej pracy analizowany pod kątem skuteczności przy różnych hiperparametrach.

Wartym uwagi jest fakt iż żaden algorytm oprócz PPO2 z domyślnymi parametrami w scenariuszu 13 nie był w stanie osiągnąć nawet 1% średniej różnicy goli. Nawet po 5 milionach kroków czasowych było to zaledwie 0,1% i 0,2% średniej różnicy goli dla kolejno algorytmów A2C i ACER. PPO2 natomiast po 5 milionach kroków czasowych osiągnęło aż 63,1% skuteczności tzn. średniej różnicy goli.

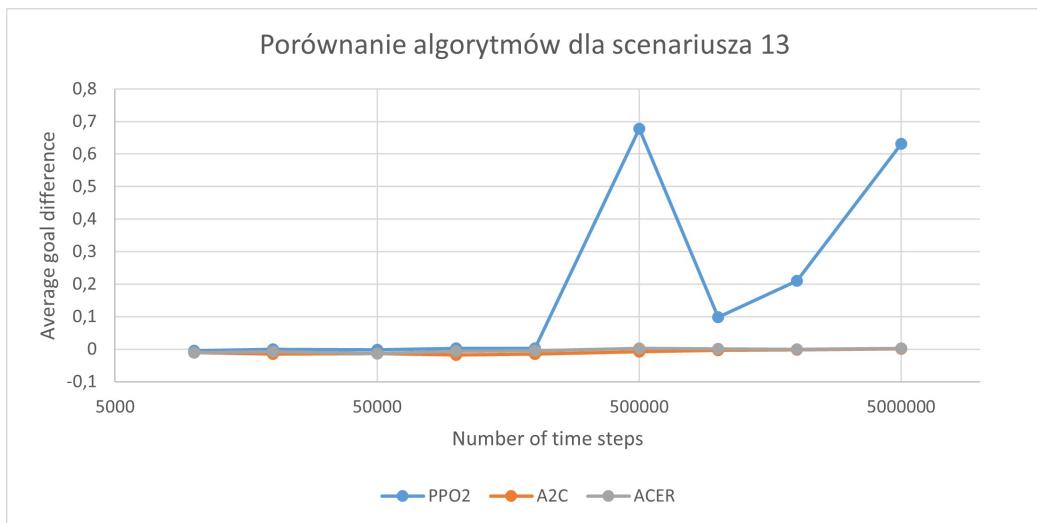
Ciekawe zjawisko występuje na wykresie 6.10. Jest to pik średniej różnicy goli dla algorytmu PPO2 dla 0,5M kroków czasowych. Pierwszym przypuszczeniem był błąd statystyczny, dlatego też badanie zostało powtórzone. Wyniki były porównywalne. Po dalszej analizie okazuje się, że najprawdopodobniej jest to spowodowane faktem iż algorytm mimo, że statystycznie w większości przypadków dochodzi do rozwiązań to szuka on innej drogi, która może okazać się lepsza tzn. kończyć się wyższą nagrodą całkowitą. Dzieje się tak gdyż algorytm nie jest zależny od modelu i nie wie on, czy zdobytą nagrodą była optymalna globalnie, czy tylko lokalnie.



Rysunek 6.8: Porównanie prędkości uczenia dla różnych algorytmów z biblioteki stable-baselines z domyślnymi parametrami dla scenariusza 17



Rysunek 6.9: Porównanie prędkości uczenia dla różnych algorytmów z biblioteki stable-baselines z domyślnymi parametrami dla scenariusza 14



Rysunek 6.10: Porównanie prędkości uczenia dla różnych algorytmów z biblioteki stable-baselines z domyślnymi parametrami dla scenariusza 13

6.4 Badanie wpływu hiperparametrów na prędkość uczenia

Ostatnim badaniem przeprowadzonym w tej pracy jest badanie wpływu hiper-parametrów na prędkość uczenia algorytmu. W związku z wynikami otrzymanymi w rozdziale 6.3 do badania został użyty algorytm PPO2, gdyż z domyślnymi parametrami osiągnął on najlepsze rezultaty.

Eksperymenty zostały przeprowadzone dla następujących konfiguracji hiperparametrów:

1. 'Hyperparamters from google code' - hiper-parametry udostępnione w [7],
2. 'Hyperparamters from Google Code (with vf_coef = 1)' - hiper-parametry udostępnione w [7] ze zmianą parametru vf_coef z 0,5 na 1,0,
3. 'Default Hyperparameters from stable-baselines' - domyślne hiper-parametry dla algorytmu PPO2 z biblioteki stable-baselines [2],
4. 'Hyperparamters from Google Benchmark' - hiper-parametry udostępnione w [8].

W rozdziale 4.1 wyjaśniono że kod udostępniony w [7] nie jest w pełni zgodny z benchmarkiem przeprowadzonym w [8]. Dlatego osobno zostaną zbadane konfiguracje z benchmarku [8] oraz z kodu udostępnionego w [7].

Wszystkie hiper-parametry omawianych przypadków zostały przedstawione w tabelach umieszczonych w dodatku A.

Badanie to odbyło się dla scenariuszy:

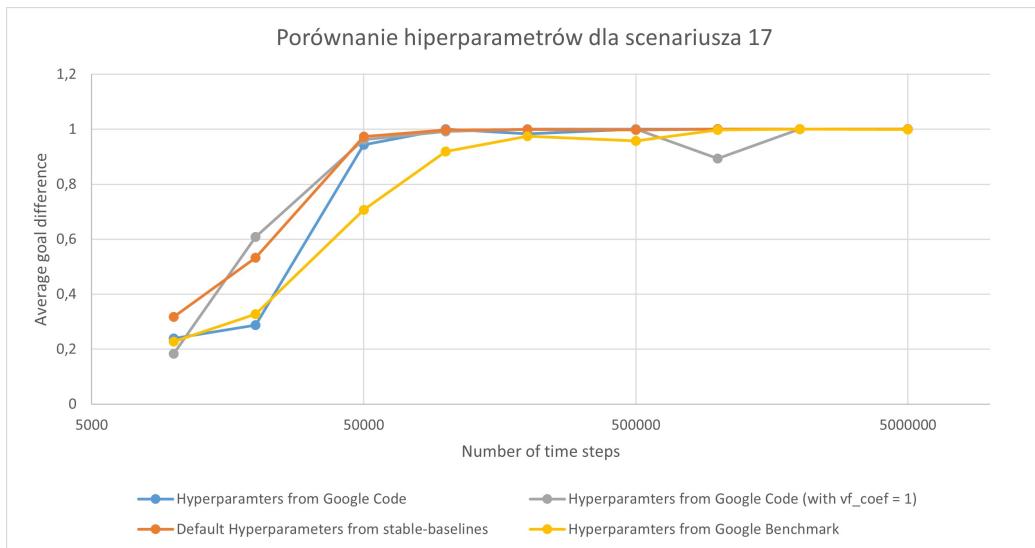
1. 'Sam na sam' - numer scenariusza: 17,
2. 'Kontratak' - numer scenariusza: 13,
3. 'Podaj biegnij strzel' - numer scenariusza: 14.

Dla każdego ze scenariuszy zostały stworzone modele z następującą ilością kroków czasowych: 0,01M, 0,02M, 0,05M, 0,1M, 0,2M, 0,5M, 1M, 2M, 5M

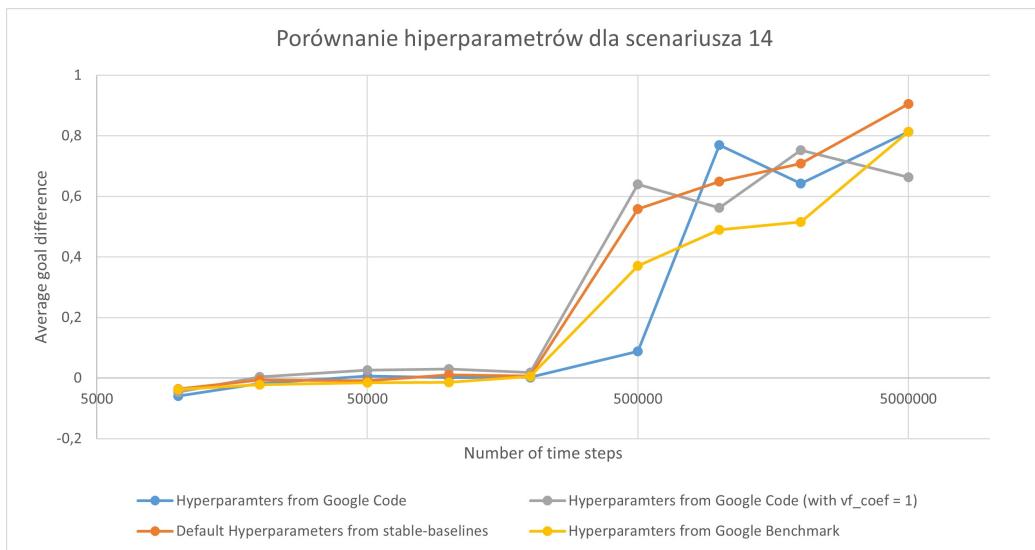
(gdzie $M = 1$ milion kroków). Szczegółowe wyniki wszystkich badań zostały zamieszczone w dodatku D. Wyniki zostały skumulowane i przedstawione na wykresach 6.11, 6.12 oraz 6.13.

Na wspomnianych wykresach można zauważyć, że dla scenariusza 17 i 14 prędkość uczenia jest porównywalna dla wszystkich przypadków, bez większych odstępstw. Aczkolwiek w scenariuszu 14 możemy zauważyć, że algorytm zdecydowanie najwolniej uczył się z hiper-parametrami udostępnionymi w [8] - Benchmark Google (Podpunkt 4 listy). Natomiast najlepsze efekty ucznia zostały otrzymane dla domyślnych hiper-parametrów z biblioteki stable-baselines [2] (Podpunkt 3).

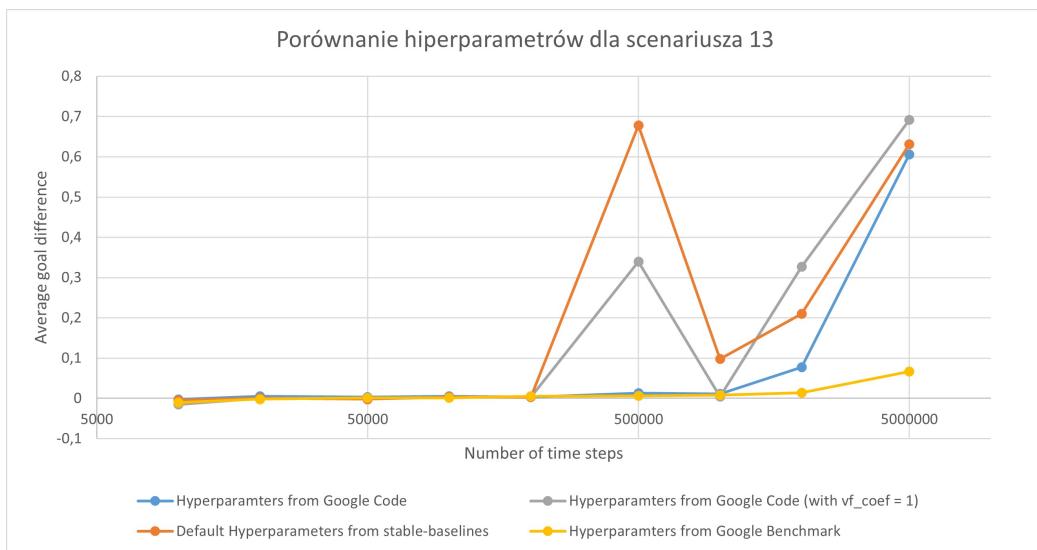
W przypadku scenariusza 13 można zauważyć, że zdecydowanie najgorsze wyniki zostały otrzymane dla hiper-parametrów z benchmarku Google (Podpunkt 4). Po 5 milionach kroków algorytm osiągnął skuteczność średniej różnicy bramek wynoszącą zaledwie 6,7%, gdzie wszystkie inne konfiguracje osiągnęły rezultat powyżej 60%. Z powodu tak dużej rozbieżności pomiędzy przypadkami zdecydowano, aby badanie powtórzyć. Wynik powtózonego badania był niemalże taki sam jak pierwszy. Wynika z tego jasno iż przypadek z podpunktu 4 najwolniej się uczył. Można poddać to pewnym wątpliwościom, gdyż zespół R&D z Googla robił badanie z doborem najlepszych hiper-parametrów dla 50 milionów kroków czasowych. Są dwa możliwe wyjaśnienia tej sytuacji. Pierwszym z nich jest fakt, że hiper-parametry z podpunktu 4 zostały ustawione na podstawie pełnych meczów, a nie scenariuszy treningowych. Prawdopodobnie w przypadku pełnych rozgrywek będą one dawały lepsze rezultaty. Drugim możliwym powodem jest prawdopodobna pomyłka. Wytrenowane hiper-parametry mogły być w kodzie udostępnionym na gitub'ie [7], nie w artykule [8].



Rysunek 6.11: Porównanie prędkości uczenia dla różnych hiper-parametrów algorytmu PPO2 dla scenariusza 17



Rysunek 6.12: Porównanie prędkości uczenia dla różnych hiper-parametrów algorytmu PPO2 dla scenariusza 14



Rysunek 6.13: Porównanie prędkości uczenia dla różnych hiper-parametrów algorytmu PPO2 dla scenariusza 13

Rozdział 7

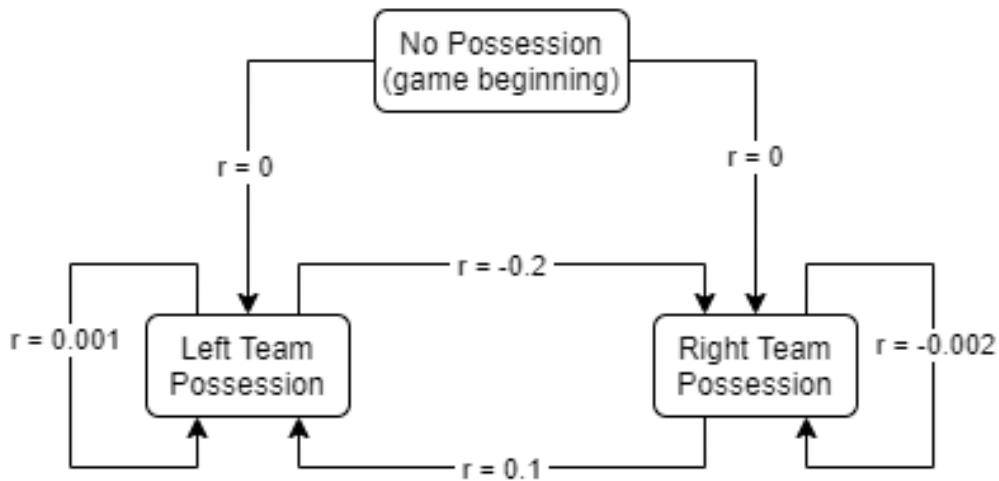
Propozycja nowego rodzaju nagrody dla środowiska gfootball

W tym rozdziale zostanie podjęta próba opracowania nowej funkcji nagrody dla środowiska gfootball. Będzie to nagroda polegająca na nagradzaniu agenta za posiadanie piłki i karanie go za posiadanie piłki przez przeciwnika. W podrozdziale 7.1 zostanie dokładnie omówiony mechanizm powyższego postępowania. Następnie w podrozdziale 7.2 zostanie przedstawiona implementacja omawianej koncepcji. Ostatecznie algorytm zostanie przetestowany, a wyniki przedstawione w podrozdziale 7.3.

Konfiguracja sprzętowa jest identyczna jak dla badań przeprowadzonych w rozdziale 6. Porównywalna jest również koncepcja uruchamiania obliczeń na maszynie wirtualnej VMware z systemem operacyjnym Ubuntu 18.04. Dodatkowo podobny jest podział połowy zasobów sprzętowych dla maszyny wirtualnej.

7.1 Opis teoretyczny

Koncepcja ta polega na nagradzaniu agenta po każdym kroku czasowym w momencie jeśli posiada piłkę oraz na karaniu go jeżeli piłkę posiada przeciwnik. Karanie polega na nagradzaniu agenta ujemną nagrodą. Koncepcyjny



Rysunek 7.1: Koncepcja nagrody typu 'Possession'

schemat został przedstawiony na rysunku 7.1. Koncepcja polega na tym żeby kara była większa niż nagroda. Na rysunku 7.1 kara jest dwukrotnie wyższa niż nagroda. Celem było, aby agent zrozumiał, że bardziej opłacalne jest posiadanie piłki niż jej strata.

Nagroda oraz kara muszą być na tyle małe, aby agent przez cały scenariusz nie zdobył nagrody wyższej niż nagroda główna. Aczkolwiek nagroda także nie może być zbyt mała, gdyż jak wiadomo z 6.2, jeżeli jej wartość jest zbyt mała to słabo wpływa ona na algorytm. Dlatego zdecydowano, że nagroda cząstkowa za posiadanie piłki będzie wynosić 0,001 w każdym kroku czasowym, kara zaś odpowiednio wyższa w zależności od testu (w przypadku rysunku 7.1 0,002).

Dodatkowo agent jest nagradzany jeżeli odbierze piłkę drużynie przeciwniej i karany jeżeli ją traci. Nagroda za pozyskanie i kara za stratę piłki jest 100 razy wyższa niż kolejno nagroda za posiadanie piłki oraz kara za posiadanie piłki przez przeciwnika, czyli wynosi 0,1, oraz strata odpowiednio wyższa (w przypadku rysunku 7.1 0,2). Nagroda ta jest stosunkowo wysoka jeżeli chodzi cały mecz, gdyż w przeciętnym meczu 5 minutowym potrafi

być nawet 50 przejęć. Wtedy strata wynosi dwie i pół bramki. W kontekście badanych scenariuszy jest to najczęściej jedno przejęcie drużyny przeciwej kończące scenariusz. Z powyższych powodów przedstawiono, że nagroda za przejęcie piłki i strata będą rzędu dziesiątej części nagrody głównej czyli zdobytej bramki.

7.2 Implementacja

Implementacja nowej funkcji nagrody nastąpiła bezpośrednio w [7], aczkolwiek tylko i wyłącznie lokalnie. Nie ma możliwości opublikować rozwiązania w ramach jednej pracy w [10], ponieważ dodając nową funkcję nagrody nie ma innej możliwości jak ingerencja w kod źródłowy. Do implementacji 'possession reward' należało zmodyfikować dwa pliki. Pierwszy z nich to 'gfootball/env/__init__.py', drugi zaś to 'gfootball/env/wrappers.py'.

Pierwszy z nich został zmodyfikowany w celu dodania dekoratora nowego typu do ustawień. Z racji, że był to warunek z odnośnikiem do klasy dodanej w ramach drugiego pliku nie został on uwzględniony w żadnym z obrazów dołączonych do tej pracy.

Drugi z plików, czyli 'gfootball/env/wrappers.py' zawiera wszystkie dekoratory środowiskowe projektu gfootball. Na obrazkach 7.2 i 7.3 została przedstawiona implementacja nagrody jako nowa klasa 'PossessionRewardWrapper' która dziedziczy po klasie 'gym.RewardWrapper'. Klasa 'gym.RewardWrapper' jest interfejsem do implementacji dekoratorów nagrody dla środowisk dziedziczących po klasie 'Gym' od OpenAI. Klasa 'PossessionRewardWrapper' została stworzona na podobieństwo klasy 'CheckpointRewardWrapper' znajdującej się w tym samym pliku, dlatego też podobnie jak przodek 'PossessionRewardWrapper' dziedziczy po 'gym.RewardWrapper'.

Warty podkreślenia jest fakt, że część z metod zaimplementowanych w klasie 'PossessionRewardWrapper' nie jest częścią kontraktu z 'gym.RewardWrapper', mowa o metodach 'get_state' i 'set_state'. Brak implementacji tych metod skutkuje niedziałaniem programu. Okazuje się, że

środowisko gfootball łamie zasady dobrego programowania SOLID (zasady te dobrze opisują [12] i [13]), ponieważ dekorator musi mieć zaimplementowane dodatkowe metody, które nie są nigdzie zdefiniowane. W przyszłości należałoby wyodrębnić klasę 'FootballRewardReward' dziedziczącą po klasie 'gym.RewardWrapper', w której byłyby wszystkie dodatkowe metody, które musi mieć zaimplementowane każdy nowy dekorator. Niestety python w swojej składni nie posiada interfejsów ani klas abstrakcyjnych, dlatego należałoby wyżej wspomniane metody zaimplementować w postaci klasy.

Przed opisem samego kodu warto na wstępnie dodać, że widoczna implementacja na rysunkach 7.2 i 7.3 jest implementacją z wysokościami kar 5 krotnie większymi niż nagroda.

```

350
351
352     class PossessionRewardWrapper(gym.RewardWrapper):
353         """A wrapper that adds a dense checkpoint reward. TEST"""
354
355     def __init__(self, env):
356         gym.RewardWrapper.__init__(self, env)
357         self._last_possession = {}
358         self._ball_pickup_reward = 0.1
359         self._ball_loose_reward = -0.5
360         self._ball_possession_reward = 0.001
361         self._team_possession_reward = 0.0
362         self._ball_enemy_possession_reward = -0.005
363
364     def reset(self):
365         self._last_possession = {}
366         return self.env.reset()
367
368     def get_state(self, to_pickle):
369         to_pickle['PossessionRewardWrapper'] = self._last_possession
370         return self.env.get_state(to_pickle)
371
372     def set_state(self, state):
373         from_pickle = self.env.set_state(state)
374         self._last_possession = from_pickle['PossessionRewardWrapper']
375         return from_pickle
376
377

```

Rysunek 7.2: Implementacja nagrody typu 'possession'

Na rysunku 7.2 w linii 332 widać deklarację nowej klasy. W liniach od 335 do 342 znajduje się konstruktor, w którym zawierają się wszystkie deklaracje stałych ze schematu pokazanego na rysunku 7.1. Dodatkowo w linii 337 znajduje się deklaracja zmiennej, która przechowuje informacje na temat poprzedniego stanu. W następnych liniach widać implementacje metod 'reset',

'get_state' i 'set_state'. Są te metody zaimplementowane analogicznie do metod przedstawionych w klasie 'CheckpointRewardWrapper' ze środowiska gfootball.

Na rysunku 7.3 znajduje się ostatnia metoda klasy 'PossessionRewardWrapper', czyli metoda 'reward'. Jest to dokładna implementacja 7.1. Tutaj warto wspomnieć, że w sytuacjach, gdzie wynik akcji jest zależny od stanu, warto tworzyć implementację w oparciu o wzorzec projektowy maszyny statycznej. Jednakże z uwagi iż są tylko 3 stany i nie jest to kod, który będzie wspierany, to podjęto decyzje o implementacji w sposób prosty, czyli przy użyciu warunków. Dodatkowo można zauważać, że wszędzie są żółte podkreślenia. Dzieje się tak ponieważ PyCharm ma ustalone domyślne wcięcia na 4 spacje. Główny projekt tej pracy był robiony z wcięciem 4 spacje. Projekt gfootball i wszystkie inne projekty google w pythonie mają wcięcia 2 spacje. Globalny standard pythonowy PEP 8 mówi o wcięciach wynoszących 4 spacje.

Wracając do kodu implementacji metody 'reward' widocznej na 7.3. Argumentem tej funkcji jest lista wartości zmienoprzecinkowych w postaci zmiennej 'reward'. Jest to lista gdyż środowisko gfootball to środowisko wielowątkowe.

W linijce 358 można zauważać deklaracje zmiennej 'observation', która jest obserwacją z pierwotnego środowiska. Jest to lista z tego samego powodu co 'reward'. Widzimy tu także kolejne złamanie zasad dobrego programowania, a dokładniej złamanie wzorca projektowego dekorator. Otóż dekorator powinien korzystać tylko i wyłącznie z metod i zmiennych zdefiniowanych przez interfejs dekoratora. Niestety w tym przypadku to biblioteka Gym nie udostępnia żadnej klasy generycznej 'reward wrapper' w takiej postaci aby móc zaobserwować środowisko według dobrych praktyk. Z drugiej strony gdyby obserwacja była przekazywana przez dekorator to mogła być ona nadpisywana przez inny dekorator. Język Python pozwala na nadpisywanie zmiennych nawet jeżeli nie powinno się tego robić.

```

356
357     def reward(self, reward):
358         observation = self.env.unwrapped.observation()
359         if observation is None:
360             return reward
361
362         assert len(reward) == len(observation)
363
364         for rew_index in range(len(reward)):
365             if reward[rew_index] >= 1:
366                 continue
367             o = observation[rew_index]
368
369             if self._last_possession.get(rew_index, None) is not None:
370                 # if we have first measure
371                 self._last_possession[rew_index] = o.get('ball_owned_team', 1)
372                 continue
373
374             if 'ball_owned_team' not in o or o['ball_owned_team'] != 0:
375                 # your team do not posses the ball
376                 if self._last_possession.get(rew_index, 1) != 0:
377                     # previously enemy possessed the ball
378                     reward[rew_index] += self._ball_enemy_possession_reward
379                     pass
380                 else:
381                     # previously our team possessed the ball
382                     reward[rew_index] += self._ball_loose_reward
383                     pass
384             else:
385                 # your team possess ball
386                 if ('ball_owned_player' not in o or o['ball_owned_player'] != o['active']):
387                     # team possess the ball, but not an active player
388                     reward[rew_index] += self._team_possession_reward
389                 else:
390                     # active player took the ball from enemy
391                     if self._last_possession.get(rew_index, 1) != 0:
392                         # previously enemy possessed the ball
393                         reward[rew_index] += self._ball_pickup_reward
394                         pass
395                     else:
396                         # previously our team possessed the ball
397                         reward[rew_index] += self._ball_possession_reward
398                         pass
399                     self._last_possession[rew_index] = o.get('ball_owned_team', 1)
400
401         return reward
402

```

Rysunek 7.3: Implementacja nagrody typu 'possession' (2)

Następnie w linijkach od 364 do 399 widzimy pętlę iterującą po każdej

nagrodzie z poszczególnych wątków (sub-środowisk). Generalnie implementacja tej części kodu jest w pełni opisana rysunkiem 7.1, który odzwierciedla co dzieje się w kodzie w tej sekcji.

7.3 Wyniki Badań

Badanie nowej funkcji nagrody przebiegało podobnie jak badania przedstawione w rozdziale 6. Zostało ono przeprowadzone dla dwóch wartości stosunku kary do nagrody:

- 2 - nagroda za posiadanie: 0,001, nagroda za przejęcie: 0,1, nagroda za posiadanie piłki przez przeciwnika: -0,002, nagroda za przejęcie piłki przez przeciwnika: -0,2,
- 5 - nagroda za posiadanie: 0,001, nagroda za przejęcie: 0,1, nagroda za posiadanie piłki przez przeciwnika: -0,005, nagroda za przejęcie piłki przez przeciwnika: -0,5.

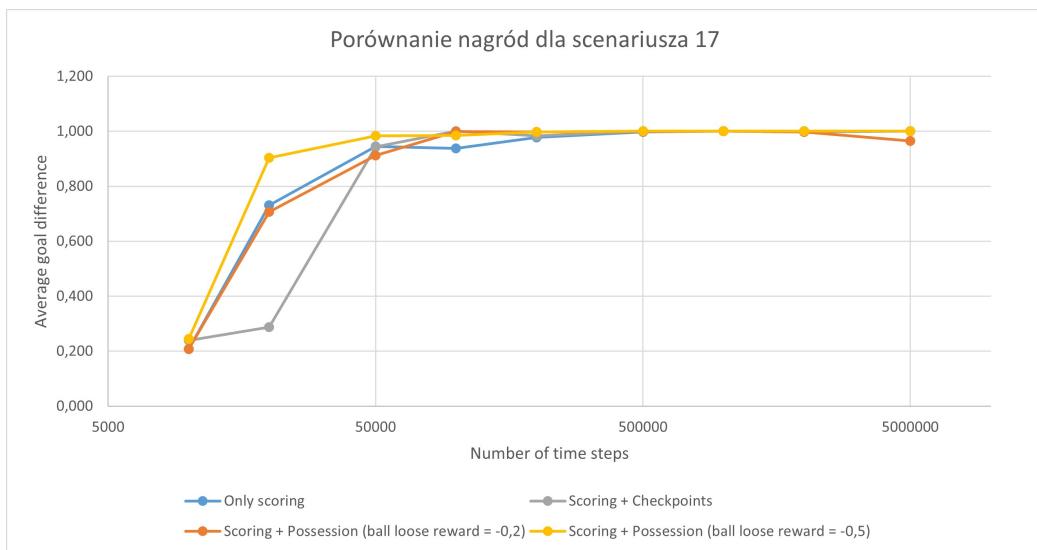
Badanie to odbyło się dla scenariuszy:

1. 'Sam na sam' - numer scenariusza: 17,
2. 'Kontratak' - numer scenariusza: 13,
3. 'Podaj biegij strzel' - numer scenariusza: 14.

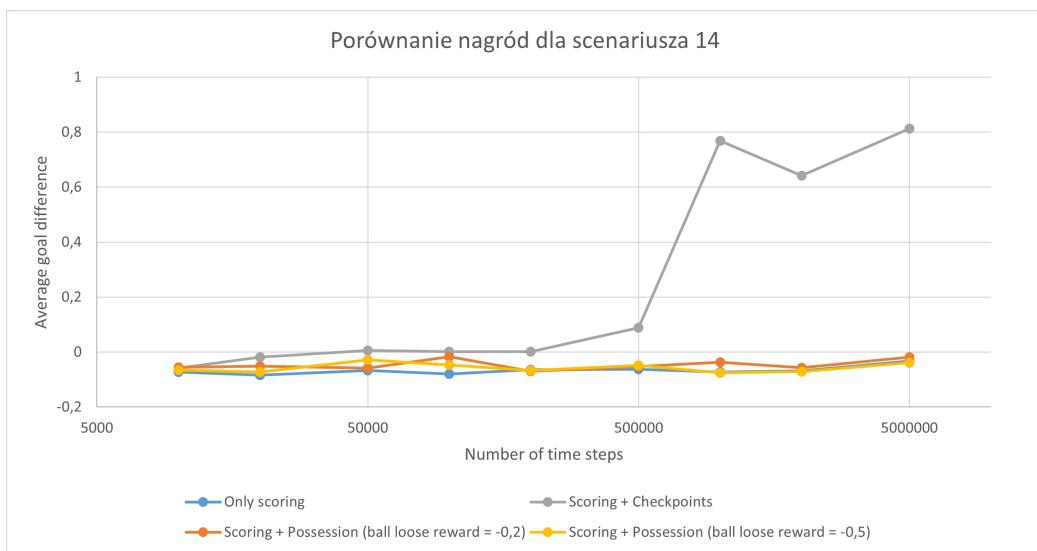
Dla każdego ze scenariuszy zostały opracowane modele z następującą liczbą kroków czasowych: 0,01M, 0,02M, 0,05M, 0,1M, 0,2M, 0,5M, 1M, 2M, 5M (gdzie M = 1 milion kroków).

Szczegółowe wyniki wszystkich badań zostały zamieszczone w dodatku E. Zostały one skumulowane i przedstawione na wykresach 7.4, 7.5 oraz 7.6. Na wykresach znajdują się nie tylko wyniki badania ale także wyniki nagrody typu 'scoring' oraz 'scoring, checkpoints' wzięte z podrozdziału 6.2.

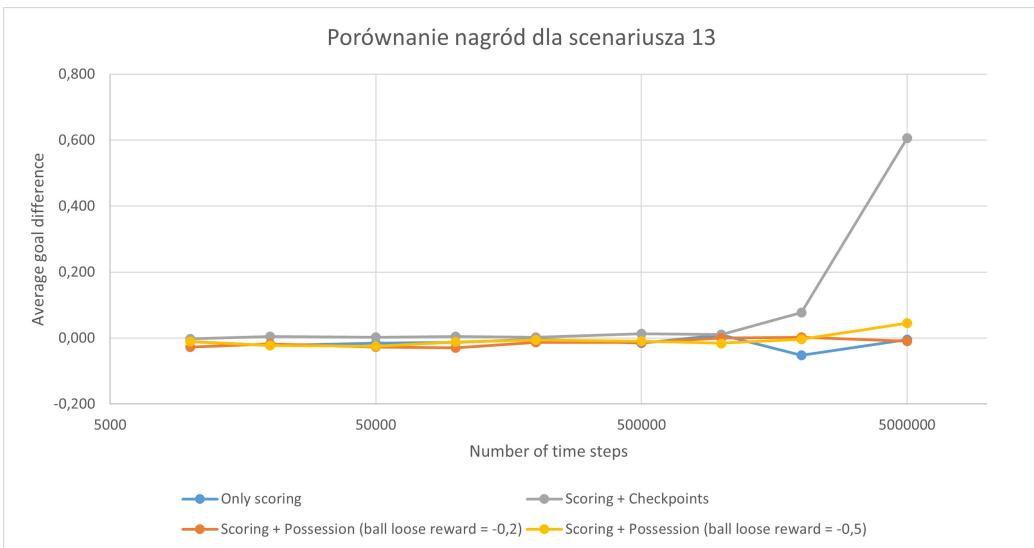
Interpretacja i szczegółowe wnioski z badania zostały przedstawione w podrozdziale 7.4.



Rysunek 7.4: Wyniki badań nowej nagrody 'possession reward' dla scenariusza 17



Rysunek 7.5: Wyniki badań nowej nagrody 'possession reward' dla scenariusza 14



Rysunek 7.6: Wyniki badań nowej nagrody 'possession reward' dla scenariusza 13

7.4 Podsumowanie

Jak można zauważyć na pierwszym z wykresów (rysunek 7.4 dotyczący scenariusza 17) najszybciej uczącym się algorytmem był ten z funkcją nagrody 'scoring + possession' z karą 5 krotnie wyższą niż nagroda. W tym przypadku karanie przyniosło pozytywny rezultat. Już po 20 tys. kroków czasowych uzyskana została efektywność algorytmu powyżej 90%.

Analizując wykres 6.9 można zauważyć, że żaden algorytm w tym scenariuszu nie osiągnął pozytywnych rezultatów nawet po 5M kroków czasowych oprócz trybu z funkcją nagrody 'scoring + checkpoints'. Okazuje się, że funkcja nagrody 'possession' nie wniosła żadnego pozytywnego wpływu na prędkość uczenia algorytmu.

Ostatni wykres dotyczył badania scenariusza 13 przedstawionego na rysunku 7.6. Widać na nim, że podobnie jak w przypadku scenariusza 14 najlepiej uczącym się algorytmem był algorytm z funkcją nagrody 'Scoring +

'Checkpoints' z efektywnością lekko ponad 60%. Inne funkcje nagrody nie osiągnęły nawet 10% skuteczności. Można zauważyć, że nagroda typu 'Scoring + Possession' z karą 5 krotnie wyższą niż nagroda, osiągnęła drugi najlepszy wynik wynoszący 4,5%, gdzie pozostałe funkcje nagrody nie osiągnęły nawet 1% skuteczności.

Podsumowując, nowa funkcja nagrody w obecnej konfiguracji nie dała zadowalających rezultatów, które wpływały by na znaczącą poprawę prędkości uczenia algorytmu w porównaniu do nagrody bazującej tylko i wyłącznie na różnicy zdobytych goli. Aczkolwiek wpływa ona nieznacznie na prędkość uczenia, co można było zauważyć w przypadku scenariusza 13.

Wartym dodania jest to, że funkcja nagrody 'possession reward' została przebadana tylko dla dwóch przypadków testowych, gdyż proces uczenia zajmuje znaczną ilość czasu. Pełne przebadanie jednej konfiguracji stosunku wysokości kary do nagrody to około 36 godzin. Być może dalsze prace i badania pozwoliłyby na rozwój tej metody i osiągnięcie tak zadowalających wyników jak w przypadku funkcji nagrody 'Scoring + Checkpoints'. Jednakże w takim wypadku potrzebne były by znaczne zasoby sprzętowe lub dodatkowe finansowanie badań w celu skorzystania z chmury obliczeniowej.

Rozdział 8

Podsumowanie

W trakcie realizacji niniejszej pracy dyplomowej zaistniały problemy ze środowiskiem w postaci niezgodności kodu z artykułem, który go opisywał. Zostały one rozwiązane w postaci własnej nakładki na środowisko gfootball łączącej środowisko wraz z bibliotekami algorytmów uczenia ze wzmacnianiem.

Pomimo trudności udało się przeprowadzić badania nad funkcją nagrody. Potwierdzone zostało badanie z [8] iż dobra funkcja pomocnicza znacząco poprawia prędkość uczenia się algorytmu. W przypadku [8] funkcją nagrody było 'scoring + checkpoints' i w pełni zostało udowodnione, iż znacząco wpływa na prędkość uczenia. Bez wspomnianej dodatkowej nagrody algorytm PPO2 nie mógł sobie poradzić z problematycznym scenariuszem nr 13 nawet po 5 milionach kroków czasowych.

Wspomniana funkcja nagrody 'scoring + checkpoints' została również przebadana pod kątem wysokości nagrody cząstkowej. Okazało się, że nagroda o wartość 0,1 za pokonanie 1/10 długości boiska jest najbardziej optymalna. Jeżeli wartość ta zostanie obniżona z 0,1 do 0,01 wtedy algorytm PPO2 w przypadku scenariusza nr 13 nie mógł dojść do 10% skuteczności nawet po 5 milionach kroków czasowych.

W pracy został poruszony temat bibliotek użytych w [7]. W trakcie pisania okazało się, że biblioteka od OPEN.AI, organizacji znanej na całym

świecie za swój wkład w rozwój dziedziny uczenia maszynowego, nie buduje się i że przestała być dostępna w oficjalnych globalnych repozytoriach. Zamiennikiem biblioteki od OPEN.AI okazała się biblioteka stable-baselines [11]. Z tego powodu należało zrobić dodatkową nakładkę na kod [7]. Udało się to, a nowa nakładka została udostępniona w publicznym repozytorium [10].

Wszystkie poprzednie przypadki były uczone przy użyciu algorytmu PPO2. W pracy zostały także przeanalizowane inne algorytmy dostępne w bibliotece stable-baselines i porównane z algorytmem PPO2 w opcji z domyślnymi parametrami. Okazało się, że żaden algorytm nie był tak skuteczny jak algorytm PPO2. Trzeba zaznaczyć, że wszystkie algorytmy były testowane z domyślnymi hiper-parametry z biblioteki stable-baselines. Być może możliwy jest dobór takich konfiguracji algorytmów, które osiągnęłyby lepszą skuteczność niż algorytm PPO2, jednakże żeby je znaleźć potrzeba dużo więcej zasobów sprzętowych lub środków finansowych na publiczną chmurę obliczeniową w celu zrównoleglenia obliczeń.

Ostatnim przeprowadzonym badaniem było badanie hiper-parametrów. Po rozpoczęciu badania okazało się, że hiper-parametry z benchmarku googla [8] oraz z repozytorium [7] różnią się, co więcej dają inną efektywność uczenia. Z badania nie można wyznaczyć jednoznacznie, która przebadana konfiguracja osiągnęła najlepsze rezultaty, jednakże łatwo można wskazać która miała najgorsze. Najgorsze były hiper parametry udostępnione w benchmarku googla [8], które miały mniejszą skuteczność niż domyślne parametry biblioteki stable-baselines. Mogło do tego dojść z powodu że benchmark googla został zrobiony na pełnych meczach, nie na scenariuszach testowych.

Ostatnią częścią tej pracy była nowa funkcja nagrody nazwana 'possession'. Jest to funkcja, która nagradza za każdy krok czasowy posiadania piłki i karze za brak posiadania piłki oraz dodatkowo każda strata piłki jest karana, zaś odbiór nagradzany. Implementacja została zakończona sukcesem, jednakże po badaniach okazało się, że funkcja nagrody 'possession' przynosi stosunkowo małe rezultaty w porównaniu do bazowej - bez dodatkowych

częstkowych funkcji nagrody. Porównując to do funkcji nagrody 'scoring + checkpoints' wręcz znikome. W tym wypadku pojawia się problem sprzętowy, gdyż dla tej funkcji nagrody niekoniecznie są optymalne te same hiperparametry algorytmu. Dodatkowo można by manipulować wysokościami kar i nagród, aby sprawdzić jaka konfiguracja daje najlepsze rezultaty. Niestety takie badania są kosztowne czasowo jak i finansowo.

Podsumowując, cel pracy wraz z wszystkimi celami pobocznymi zostały spełnione. Największym osiągnięciem pracy było stworzenie i zbadanie nowej funkcji nagrody, co wymagało dogłębnej analizy środowiska gfootball.

Dalsze prace powinny być ukierunkowane na:

1. Pełne badania hiper-parametrów różnych algorytmów na środowisku gfootball;
2. Implementacja lub zastosowanie, najnowszych algorytmów np. REINBOW oraz przeprowadzenie badań na środowisku gfootball;
3. Badania pełnych meczów.

Jak można zauważyć w tej pracy to wcale nie szerokość tematu stanowi największe wyzwanie, lecz skończona ilość zasobów sprzętów lub środków finansowych do użycia publicznej chmury obliczeniowej. Problemem jest to, że ciężko jest znaleźć optymalne hiper-parametry dla algorytmów. Każdy hiper-parametr można dowolnie modyfikować i sprawdzać jak wpływa na zachowanie danego algorytmu. Przy 10 hiper-parametrach typu logicznego daje to 2^{10} możliwości. Jest to czas 4 lat i 2,5 miesiąca przy konfiguracji sprzętowej wykorzystywanej w pracy.

Dodatek A

Hiperparametry PPO2

Hyperparamters from Google Code		
nazwa w stable-baselines	nazwa w football RL	wartość
gamma	gamma	0,993
n_steps	number_of_steps_per_epoch	128
ent_coef	entropy_coefficient	0,01
learning_rate	learning_rate	0,00008
vf_coef	vf_coefficient	0,5
max_grad_norm	max_grad_norm	0,5
nminibatches	number_of_mini_batches_in_epoch	8
noptepochs	number_of_updates_per_epoch	4
cliprange	clip_range	0,27

Hyperparamters from Google Code (with vf_coef = 1)		
nazwa w stable-baselines	nazwa w football RL	wartość
gamma	gamma	0,993
n_steps	number_of_steps_per_epoch	128
ent_coef	entropy_coefficient	0,01
learning_rate	learning_rate	0,00008
vf_coef	vf_coefficient	1
max_grad_norm	max_grad_norm	0,5
nminibatches	number_of_mini_batches_in_epoch	8
noptepochs	number_of_updates_per_epoch	4
cliprange	clip_range	0,27

Default Hyperparameters from stable-baselines		
nazwa w stable-baselines	nazwa w football RL	wartość
gamma	gamma	0,99
n_steps	number_of_steps_per_epoch	128
ent_coef	entropy_coefficient	0,01
learning_rate	learning_rate	0,00025
vf_coef	vf_coefficient	0,5
max_grad_norm	max_grad_norm	0,5
nminibatches	number_of_mini_batches_in_epoch	4
noptepochs	number_of_updates_per_epoch	4
cliprange	clip_range	0,2

Hyperparamters from Google Benchmark		
nazwa w stable-baselines	nazwa w football RL	wartość
gamma	gamma	0,993
n_steps	number_of_steps_per_epoch	128
ent_coef	entropy_coefficient	0,03
learning_rate	learning_rate	0,000343
vf_coef	vf_coefficient	0,5
max_grad_norm	max_grad_norm	0,64
nminibatches	number_of_mini_batches_in_epoch	8
noptepochs	number_of_updates_per_epoch	2
cliprange	clip_range	0,08

Dodatek B

**Wyniki badań porównania
wysokości nagrody pomocniczej
'checkpoints'**

CHECKPOINT REWARD = 0,1		
-------------------------	--	--

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:41	0,184
0,002M	0:01:20	0,609
0,005M	0:03:34	0,962
0,01M	0:07:19	0,992
0,05M	0:40:41	1,000
0,02M	0:16:06	1,000
1M	1:23:40	0,894
2M	2:45:03	1,000
5M	7:01:49	1,000

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:41	-0,047
0,002M	0:01:22	0,004
0,005M	0:03:29	0,026
0,01M	0:06:52	0,030
0,02M	0:13:50	0,018
0,05M	0:36:38	0,640
1M	1:13:32	0,562
2M	2:24:07	0,752
5M	6:00:51	0,663

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:53	-0,015
0,002M	0:01:50	0,000
0,005M	0:04:25	0,000
0,01M	0:09:14	0,003
0,02M	0:18:27	0,003
0,05M	0:44:58	0,340
1M	1:30:00	0,004
2M	2:56:08	0,327
5M	7:25:36	0,692

CHECKPOINT REWARD = 0,05

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:46	0,180
0,002M	0:01:33	0,712
0,005M	0:04:18	0,989
0,01M	0:08:57	1,000
0,02M	0:18:40	1,000
0,05M	0:43:43	0,999
1M	1:25:18	1,000
2M	2:50:26	0,971
5M	7:05:58	0,998

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:41	-0,041
0,002M	0:01:24	-0,030
0,005M	0:03:28	0,008
0,01M	0:07:05	0,008
0,02M	0:14:09	0,012
0,05M	0:36:36	0,015
1M	1:13:53	0,686
2M	2:28:30	0,614
5M	6:51:43	0,560

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:55	-0,019
0,002M	0:01:50	-0,001
0,005M	0:04:35	0,001
0,01M	0:09:24	0,001
0,02M	0:18:56	0,008
0,05M	0:47:10	0,008
1M	1:42:44	0,007
2M	3:12:43	0,282
5M	7:48:58	0,601

CHECKPOINT REWARD = 0,01

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:43	0,247
0,002M	0:01:27	0,666
0,005M	0:04:01	0,967
0,01M	0:07:48	0,948
0,02M	0:16:05	0,938
0,05M	0:41:19	0,999
1M	1:24:05	1,000
2M	2:48:55	0,962
5M	7:25:57	0,988

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:54	-0,061
0,002M	0:01:48	-0,043
0,005M	0:04:14	-0,038
0,01M	0:08:59	-0,044
0,02M	0:17:53	0,000
0,05M	0:37:00	0,604
1M	1:12:15	0,570
2M	2:22:01	0,546
5M	6:11:34	0,774

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:57	-0,023
0,002M	0:01:52	-0,029
0,005M	0:04:30	0,003
0,01M	0:09:16	0,002
0,02M	0:19:31	0,002
0,05M	0:47:57	0,001
1M	1:35:34	0,000
2M	3:18:22	0,001
5M	8:05:12	0,005

Dodatek C

**Wyniki badań porównania
algorytmów: PPO2, A2C i ACER
z defaultowymi hiperparametrami
z biblioteki stable-baselines**

PPO2 with default hyperparameters from stable-baselines

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:46	0,318
0,002M	0:01:22	0,533
0,005M	0:03:37	0,974
0,01M	0:07:41	0,997
0,02M	0:15:42	0,999
0,05M	0:41:58	0,997
1M	1:24:20	1,000
2M	3:05:33	1,000
5M	7:01:59	1,000

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:58	-0,036
0,002M	0:01:41	-0,006
0,005M	0:03:27	-0,009
0,01M	0:06:51	0,010
0,02M	0:13:44	0,006
0,05M	0:40:12	0,558
1M	1:14:52	0,649
2M	2:23:45	0,708
5M	6:18:20	0,905

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:56	-0,004
0,002M	0:01:49	0,000
0,005M	0:04:24	-0,002
0,01M	0:08:50	0,003
0,02M	0:17:35	0,003
0,05M	0:44:06	0,678
1M	1:28:02	0,098
2M	2:57:28	0,210
5M	7:43:08	0,631

A2C with default hyperparameters from stable-baselines

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:28	0,132
0,002M	0:00:48	0,161
0,005M	0:01:57	0,136
0,01M	0:03:36	0,128
0,02M	0:07:28	0,142
0,05M	0:17:25	0,134
1M	0:35:14	0,206
2M	1:31:13	0,998
5M	4:14:19	0,993

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:27	-0,047
0,002M	0:00:55	-0,059
0,005M	0:02:10	-0,048
0,01M	0:04:18	-0,051
0,02M	0:07:50	-0,051
0,05M	0:19:53	-0,023
1M	0:38:11	-0,016
2M	1:18:52	-0,011
5M	3:33:55	-0,020

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:59	-0,011
0,002M	0:01:55	-0,015
0,005M	0:03:07	-0,013
0,01M	0:06:08	-0,018
0,02M	0:12:06	-0,015
0,05M	0:30:37	-0,007
1M	1:00:06	-0,003
2M	2:00:10	-0,002
5M	5:02:14	0,001

ACER with default hyperparameters from stable-baselines

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:50	0,122
0,002M	0:02:06	0,154
0,005M	0:06:06	0,135
0,01M	0:12:30	0,133
0,02M	0:27:33	0,937
0,05M	1:20:14	1,000
1M	2:47:38	1,000
2M	5:17:32	1,000
5M	13:32:01	1,000

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:46	-0,058
0,002M	0:02:05	-0,059
0,005M	0:06:28	-0,051
0,01M	0:13:12	-0,071
0,02M	0:26:12	-0,010
0,05M	1:08:52	0,000
1M	2:25:44	0,000
2M	4:42:40	0,000
5M	12:04:56	0,718

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:01:22	-0,010
0,002M	0:02:47	-0,008
0,005M	0:07:24	-0,013
0,01M	0:15:56	-0,006
0,02M	0:30:44	-0,005
0,05M	1:27:53	0,003
1M	2:46:50	0,001
2M	5:10:22	0,000
5M	14:23:34	0,002

Dodatek D

Wyniki badań porównania
hiperparametrów dla algorytmu
PPO2

Hyperparamters from Google Code

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:43	0,239
0,002M	0:01:23	0,287
0,005M	0:03:29	0,944
0,01M	0:07:33	1,000
0,02M	0:15:24	0,984
0,05M	0:40:49	1,000
1M	1:22:56	1,000
2M	2:47:07	1,000
5M	7:15:05	1,000

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:52	-0,059
0,002M	0:01:43	-0,018
0,005M	0:04:06	0,006
0,01M	0:07:52	0,001
0,02M	0:14:31	0,002
0,05M	0:38:34	0,088
1M	1:12:48	0,769
2M	2:25:21	0,642
5M	6:17:56	0,813

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:01:01	-0,002
0,002M	0:01:56	0,005
0,005M	0:04:28	0,003
0,01M	0:08:59	0,005
0,02M	0:18:05	0,003
0,05M	0:44:59	0,013
1M	1:39:57	0,011
2M	3:13:30	0,077
5M	7:44:14	0,606

Hyperparamters from Google Code (with vf_coef = 1)

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:41	0,184
0,002M	0:01:20	0,609
0,005M	0:03:34	0,962
0,01M	0:07:19	0,992
0,02M	0:16:06	1,000
0,05M	0:40:41	1,000
1M	1:23:40	0,894
2M	2:45:03	1,000
5M	7:01:49	1,000

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:41	-0,047
0,002M	0:01:22	0,004
0,005M	0:03:29	0,026
0,01M	0:06:52	0,030
0,02M	0:13:50	0,018
0,05M	0:36:38	0,640
1M	1:13:32	0,562
2M	2:24:07	0,752
5M	6:00:51	0,663

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:53	-0,015
0,002M	0:01:50	0,000
0,005M	0:04:25	0,000
0,01M	0:09:14	0,003
0,02M	0:18:27	0,003
0,05M	0:44:58	0,340
1M	1:30:00	0,004
2M	2:56:08	0,327
5M	7:25:36	0,692

Default Hyperparameters from stable-baselines

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:46	0,318
0,002M	0:01:22	0,533
0,005M	0:03:37	0,974
0,01M	0:07:41	0,997
0,02M	0:15:42	0,999
0,05M	0:41:58	0,997
1M	1:24:20	1,000
2M	3:05:33	1,000
5M	7:01:59	1,000

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:58	-0,036
0,002M	0:01:41	-0,006
0,005M	0:03:27	-0,009
0,01M	0:06:51	0,010
0,02M	0:13:44	0,006
0,05M	0:40:12	0,558
1M	1:14:52	0,649
2M	2:23:45	0,708
5M	6:18:20	0,905

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:56	-0,004
0,002M	0:01:49	0,000
0,005M	0:04:24	-0,002
0,01M	0:08:50	0,003
0,02M	0:17:35	0,003
0,05M	0:44:06	0,678
1M	1:28:02	0,098
2M	2:57:28	0,210
5M	7:43:08	0,631

Hyperparamters from Google Benchmark

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:30	0,227
0,002M	0:00:57	0,328
0,005M	0:02:21	0,707
0,01M	0:04:58	0,919
0,02M	0:11:00	0,975
0,05M	0:29:31	0,957
1M	1:01:05	0,998
2M	2:03:11	1,000
5M	5:23:17	0,999

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:31	-0,038
0,002M	0:01:01	-0,022
0,005M	0:02:23	-0,015
0,01M	0:04:41	-0,014
0,02M	0:09:25	0,005
0,05M	0:24:14	0,371
1M	0:49:54	0,489
2M	1:40:02	0,515
5M	4:21:24	0,813

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:45	-0,010
0,002M	0:01:29	-0,002
0,005M	0:03:29	0,001
0,01M	0:06:54	0,001
0,02M	0:13:50	0,005
0,05M	0:34:19	0,006
1M	1:08:09	0,008
2M	2:17:57	0,014
5M	5:42:38	0,067

Dodatek E

Wyniki badań nagrody typu 'possession'

Scoring + Possession (ball loose reward = -0,2)

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:41	0,207
0,002M	0:01:29	0,707
0,005M	0:03:49	0,912
0,01M	0:07:38	0,999
0,02M	0:16:12	0,998
0,05M	0:41:50	0,999
1M	1:23:10	1,000
2M	2:48:51	0,997
5M	7:08:58	0,965

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:41	-0,056
0,002M	0:01:23	-0,051
0,005M	0:03:46	-0,059
0,01M	0:07:01	-0,017
0,02M	0:13:42	-0,069
0,05M	0:34:33	-0,052
1M	1:08:44	-0,037
2M	2:17:19	-0,057
5M	5:52:47	-0,019

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:58	-0,027
0,002M	0:01:58	-0,018
0,005M	0:04:49	-0,027
0,01M	0:09:29	-0,030
0,02M	0:18:15	-0,013
0,05M	0:45:27	-0,013
1M	1:30:35	0,000
2M	3:02:59	0,002
5M	7:47:34	-0,010

Scoring + Possession (ball loose reward = -0,5)

academy empty goal close (17)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:43	0,245
0,002M	0:01:24	0,903
0,005M	0:03:46	0,984
0,01M	0:07:42	0,985
0,02M	0:16:03	0,997
0,05M	0:41:32	1,000
1M	1:23:55	1,000
2M	2:50:48	1,000
5M	7:13:00	1,000

academy run pass and shoot with keeper (14)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:45	-0,065
0,002M	0:01:23	-0,072
0,005M	0:03:23	-0,029
0,01M	0:06:43	-0,047
0,02M	0:13:54	-0,067
0,05M	0:35:20	-0,048
1M	1:09:20	-0,076
2M	2:24:37	-0,071
5M	5:57:48	-0,038

academy counterattack hard (13)		
ilość kroków	czas treningu (hh:mm:ss)	średni wynik ewaluacji
0,001M	0:00:58	-0,011
0,002M	0:02:04	-0,023
0,005M	0:05:10	-0,025
0,01M	0:09:01	-0,012
0,02M	0:18:00	-0,006
0,05M	0:45:03	-0,009
1M	1:29:15	-0,016
2M	3:00:14	-0,003
5M	7:41:44	0,045

Bibliografia

- [1] Bastiaan Schuiling. URL: <https://github.com/BazkieBumpercar/GameplayFootball>. (accessed: 30.11.2020).
- [2] Ashley Hill. URL: <https://github.com/hill-a/stable-baselines>. (accessed: 30.11.2020).
- [3] Open.AI. URL: <https://github.com/openai/baselines>. (accessed: 30.11.2020).
- [4] Abdul Wahid. *Big data and machine learning for Businesses*. URL: <https://www.slideshare.net/awahid/big-data-and-machine-learning-for-businesses>. (accessed: 23.11.2020).
- [5] Richard S. Sutton i Andrew G. Barto. *Reinforcement Learning: An Introduction second edition*. Bradford Books; second edition (November 13, 2018), 2018. ISBN: 9780262039246.
- [6] Joshua Achiam. „Spinning Up in Deep Reinforcement Learning”. W: (2018).
- [7] Google research. *google-research/football source code*. URL: <https://github.com/google-research/football>. (accessed: 24.10.2020).
- [8] Karol Kurach i in. „Google Research Football: A Novel Reinforcement Learning Environment”. Wer. 2. W: *CoRR* abs/1907.11180 (2020). arXiv: 1907.11180. URL: <http://arxiv.org/abs/1907.11180>.

- [9] Lasse Espeholt i in. „IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures”. Wer. 3. W: abs/1802.01561 (2018). arXiv: 1907 . 11180. URL: <https://arxiv.org/abs/1802.01561>.
- [10] Piotr Klukowski. *Master Thesis source code*. URL: <https://github.com/Klukov/FootballRL>. (accessed: 30.11.2020).
- [11] Ashley Hill. URL: <https://stable-baselines.readthedocs.io/en/master/index.html>. (accessed: 30.11.2020).
- [12] Robert C. Martin. *Clean Code A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2009. ISBN: 978-0-13-235088-4.
- [13] Robert C. Martin. *Clean Architecture A Craftsman’s Guide to Software Structure and Design*. Prentice Hall, 2018. ISBN: 978-0-13-449416-6.
- [14] Ashley Hill. URL: <https://github.com/hill-a/stable-baselines>. (accessed: 30.11.2020).
- [15] Gianmario Spacagna Ivan Vasilev Daniel Slater. *Python Deep Learning*. Packt Publishing; second edition, 2019. ISBN: 978-1-78934-846-0.
- [16] Maxim Lapan. *Deep Reinforcement Learning Hands-On*. Packt Publishing; second edition, 2020. ISBN: 978-1-83882-699-4.