POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AiR)
SPECJALNOŚĆ: Technologie informacyjne w systemach automatyki (ART)

PRACA DYPLOMOWA MAGISTERSKA

Sterowanie platformy mobilnej za pomocą uczenia ze wzmocnieniem

Mobile platform control using reinforcement learning

AUTOR:
Jakub Mateusz Szczyrk

PROWADZĄCY PRACĘ:
dr inż. Krzysztof Halawa, W4/K30

WROCŁAW, 2021

Spis treści

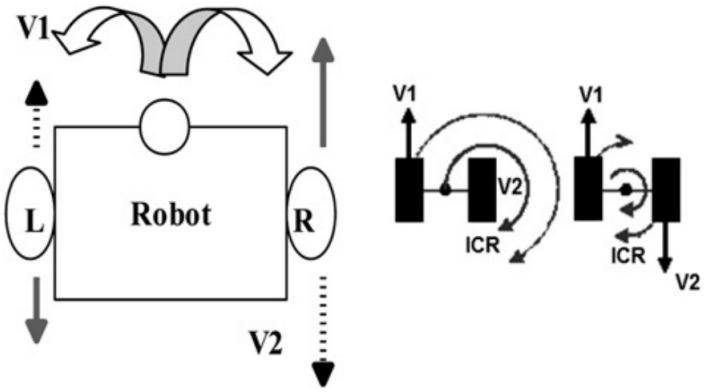
Spis rysunków

Listings

Rozdział 1

Cel i zakres pracy

Celem przedstawianej pracy magisterskiej było badanie sterowania platformą mobilną za pomocą uczenia ze wzmocnieniem. Problem poruszany w pracy dotyczy robota mobilnego, który zbiera dane o środowisku i wysyła polecenia ruchu do prawego i lewego koła. Sterowanie robotem odbywa się poprzez obracanie silników z różnymi prędkościami kątowymi. Kiedy koła poruszają się z tą samą prędkością, robot jedzie na wprost, natomiast ruch kół w przeciwnych kierunkach skutkuje obrotem robota w miejscu. Kiedy jedno koło obraca się wolniej niż drugie, robot przesuwa się po łuku w stronę wolniejszego obrotu. Szczegółowa analiza przedstawionego sterowania robotem znajduje się w artykule [23]:



Rys. 1.1: Sterowanie robotem mobilnym Klasy (2,0)

Projekt pokaże, że sterowanie platformą mobilną będzie możliwe poprzez wyuczenie odpowiednich reguł przyswojonych w uczeniu ze wzmocnieniem.

Badania zostały przeprowadzone na zaprogramowanej symulacji w zintegrowanym środowisku Unity dla robota typu sumo. Szczegółowy opis symulatora znajduje się w pracy inżynierskiej [29]. Każda symulowana platforma zwana też agentem, posiadała własne symulowane środowisko do uczenia.

Rozdział 2

Wstęp

Robotyka jest bardzo mocno związana z integracją systemów. Wykorzystuje ona funkcjonalne urządzenie mechaniczne do wykonania zadania poprzez inteligentną integrację komponentów. Wiele z tych elementów jest współdzielonych z systemami i sterowaniem, obliczeniami, animacją postaci, sztuczną inteligencją i innymi obszarami. Dlatego w związku z rosnącą liczbą koncepcji i algorytmów robotyki, wykorzystuje się ją w coraz większej liczbie aplikacji zewnętrznych, prze co granic robotyki nie można jasno zdefiniować.

Robot to urządzenie, które potrafi naśladować ruchy na wzór człowieka, a także wyposażony jest w programowalny manipulator umożliwiający dokonanie wielu operacji, zgodnie z zaprogramowaną ścieżką do realizowania różnych zadań. Terminem ruchoma platforma określa się również urządzenie zawierające mechanizm ruchu, na który mają wpływ komponenty wykrywające, planujące, wyzwalające i sterujące, zgodnie z opisem w [17]. Minimalna liczba takich elementów nie musi być uwzględniana w oprogramowaniu ani modyfikowana przez użytkownika korzystającego z urządzenia - można skojarzyć charakterystykę ruchu do urządzenia na stałe przez producenta. Robot to urządzenie, które potrafi naśladować ruchy na wzór człowieka, a także wyposażony jest w programowalny manipulator umożliwiający dokonanie wielu operacji, zgodnie z zaprogramowaną ścieżką do realizowania różnych zadań.

Urządzenia mobilne zawierają się w typie robotów, które przemieszczają się poprzez koła bądź gąsienice. Poprzez otoczenie robota rozumie się obszar, gdzie funkcjonuje robot, natomiast w przypadku robotów stacjonarnych, środowisko to roboczy obszar. Robot mobilny posiada takie cechy, jak mobilność (poruszanie się po powierzchni płaskiej), pewien stopień autonomii (ograniczona interakcja międzyludzka) czy zdolność postrzegania (wykrywania i reagowania na otoczenie).

W książce [26] autorzy szczegółowo przedstawiają technologię robotów mobilnych, polegającą na tworzeniu urządzeń zdolnych do przemieszczania się w obrębie otoczenia takiego jak powietrze, woda czy przestrzenie otwarte. Urządzenia mobilne są zwykle sterowane przy użyciu konkretnego oprogramowania oraz wykorzystują czujniki i inne urządzenia do wykrywania otaczającego środowiska. Roboty mobilne scalają ewolucje w zakresie AI (sztucznej inteligencji) oraz roboty fizyczne, aby mogły poruszać się po środowisku. Wśród technologii robotów mobilnych wyszczególnia się dwa typy: autonomiczne oraz nieautonomiczne roboty mobilne. Robota można określić autonomicznym, jeżeli jest zdolny do zdobywania swoich celów bez pomocy z zewnątrz, natomiast mobilność dotyczy możliwości całkowitego poruszania się w środowisku. Autonomiczny robot mobilny ze wszystkimi komponentami zintegrowanymi w strukturze mechanicznej może badać swoje otoczenie bez zewnętrznego kierowania. Robot kierowany wykorzystuje do poruszania się jakiś system naprowadzania. Na robot mobilny składają się elementy fizyczne (sprzęt) jak i komputerowe (oprogramowanie). Jeśli chodzi o elementy sprzętowe, robot mobilny można traktować jako grupę podsystemów, odpowiadającą na pytania w jaki

sposób robot przemieszcza się po swoim otoczeniu czy jak wygląda jego porozumiewanie się ze środowiskiem. Ruch autonomicznych robotów mobilnych jest możliwy w środowisku, które nie jest kontrolowane bez wyposażenia naprowadzającego, podczas gdy zwykłe roboty mobilne polegają na urządzeniach nawigacyjnych. Wyposażenie nawigacyjne pozwala takim robotom poruszać się po z góry określonej ścieżce nawigacji w przestrzeni względnie kontrolowanej.

Urządzenie mobilne - robot, składa się ze sterownika, oprogramowania odpowiadającego za sterowanie, czujników i komponentów wykonawczych. Kontroler stanowi zwykle mikroprocesor bądź komputer. Oprogramowanie do zarządzania urządzeniami mobilnymi może wykorzystywać języki asemblera lub wysokiego poziomu - C++ czy C.

Czujniki, które należy wybrać dla robota zależą od jego wymagań takich jak zapobieganie kolizjom, pozycjonowanie i inne specyficzne zastosowania, pomiar dotyku i bliskości. Robotics Simulator stanowi narzędzie do generowania aplikacji przeznaczonych dla fizycznych robotów, bez względu na maszynę rzeczywistą. Takie aplikacje mogą być bezpośrednio przeszczepiane (lub odtwarzane) na fizycznych robotach bez modyfikacji. Ze względu na to, że od robotów mobilnych wymaga się osiągania wyznaczonych celów w otoczeniach, które są skomplikowane, algorytm wykorzystywany do kontroli zachowania robota powinien sprawdzać się w ciężkich warunkach stanu robota czy otoczenia. Konstrukcja algorytmu umożliwiającego manipulację robotem powinien zostać poddany testom w zmiennych warunkach środowiska, w których funkcjonuje robot. Proces ten musi być opatrzony dużą liczbą eksperymentów na rzeczywistych robotach, dlatego też wymagane jest wykonanie realistycznego symulatora do testowania algorytmów behawioralnych.

Funkcjonalnie na robota składają się: system mechaniczny, system sterowania i system zasilania. System sterowania definiuje sygnał sterujący, który należy przyłożyć do systemu (robota), aby uzyskać określone właściwości. Sygnał sterujący generowany jest na podstawie danych o tym systemie. Zadaniem układu sterowania dla robota jest wygenerowanie sygnału sterującego, aby system osiągnął wymagane położenie i kierunek przestrzenny. Niezbędne jest uwzględnienie przeszkód i podstawowej kinematyki oraz dynamiczna kontrola parametrów. Sterowanie takim robotem polega na zapewnieniu określonej sekwencji kroków w programie jak również upewnieniu się, że poszczególne ruchy są wykonywane poprawnie.

Rozdział 3

Narzędzia i technologie

3.1. UNITY

Unity, dokładnie przedstawiony w [24], to silnik gier, który został specjalnie zaprojektowany do ułatwienia tworzenia aplikacji wieloplatformowych. Po wygenerowaniu projektu wyświetlany jest w edytorze pierwszy plik sceny. Dokument [5] opisuje część „Hierarchia", która reprezentuje hierarchiczną strukturę obiektów w grze. Na scenie znajdują się przede wszystkim dwa obiekty – kamera i światło kierunkowe. Main Camera wykorzystywany jest jako wizja użytkownika, który szczegółowo opisano w dokumentacji [6]. Widok działania kamery tworzonego projektu jest pokazany w oknie "Game", przedstawionym w dokumencie [4]. Directional Light, opisane w [8], służy do prawidłowego oświetlenia obiektów w scenie, której szczegółowy opis można znaleźć na stronie [11]. Środowisko pozwala na tworzenie elementów w widoku sceny, np. dodawanie modeli czy kamery, jak opisano w dokumentacji [11]. To trójwymiarowe okno umożliwia wizualne umieszczenie wszystkich wykorzystanych zasobów oraz obserwacje obiektów pod różnymi kątami:

- Obrót - należy przesunąć kursor, trzymając wciśnięty prawy przycisk myszy.
- Przybliżanie i oddalanie - należy obracać kółko myszki komputerowej.
- Przemieszczenie - należy przytrzymać kółko myszy, aby przesunąć kursor.

W większości przypadków do nawigacji po scenie używa się myszy. Najważniejsze operacje nawigacyjne to przeciąganie, orbitowanie oraz powiększanie i pomniejszanie. Każdy z tych ruchów bazuje na technice „kliknij i przeciągnij", a określona kombinacja klawiszy Alt i Ctrl jest przytrzymywana w tym samym czasie, w zależności od typu myszy. Przesunięcie stanowi translację kamery, orbita to obrót kamery, a przybliżanie i oddalanie to skalowanie kamery.

W Unity występuje możliwość uruchomienia projektu w edytorze Unity, gdzie widok sceny jest aktualizowany do obecnego stanu. Po zatrzymaniu aplikacji widok i ustawienia powracają do swojego początkowego stanu. Oprócz tego możliwe jest kopiowanie elementów, aby nie utracić zmian podczas uruchomionego projektu. Okno „Game"pokazuje aktualnie uruchomiony projekt, co pozwala sprawdzić, jak działają różne mechanizmy. Unity pozwala na dostosowanie rozmiaru okna do określonej rozdzielczość i proporcji urządzenia. Dzięki temu utworzone aplikacje prezentują się dobrze na każdym sprzęcie.

Sekcja „Inspector", opisana w dokumencie [14], wyświetla właściwości wybranego obiektu. Dokument [3] wyjaśnia, że GameObjects to puste kontenery, które można formatować, dodając komponenty. GameObjects w swoim zachowaniu przypominają foldery zawierające inne GameObjects, dlatego przydają się do organizowania scen. Wszystkie GameObjects, z których aplikacja aktywnie korzysta w bieżącej scenie, zostaną wyświetlone w oknie „Hierarchia". Za każdym razem, gdy GameObject zostanie usunięty z Hierarchi, zniknie również ze sceny.

Aby dostosować dowolny obiekt GameObject zaleca się użycie okna Inspektor. Prezentuje ono wszystkie komponenty GameObject wraz z ich właściwościami (po uwzględnieniu GameObject w hierarchi). Składniki te pozwalają GameObjects wyświetlać kształty geometryczne, emitować światło, działać jak kamery, a nawet używać skryptów do tworzenia złożonych zachowań. Za manipulację obiektami w widoku sceny odpowiada pasek narzędzi.

Unity jako system modułowy bazuje na komponentach do tworzenia obiektów w aplikacji. Komponenty te są powiązane z funkcjami przedstawionymi szczegółowo na stronie internetowej [19]. Obiekty aplikacji nie opierają się na hierarchii klas, lecz są tworzone jako kolekcje komponentów. Ogólnie rzecz biorąc, obiekty istnieją na płaskiej płaszczyźnie, na której pojedynczy obiekt może mieć różne zestawy komponentów. Pojedynczy obiekt nie znajduje się na różnych gałęziach drzewa, jak w przypadku metody struktury dziedziczenia. Rozwiązanie to ułatwia wydajne prototypowanie, ponieważ może szybko dostosować się do różnych komponentów, bez konieczności refaktoryzacji łańcucha dziedziczenia w miarę zmian obiektu. Unity korzysta również z bardzo wydajnego wizualnego trybu pracy i dużej niezależności od platformy. Przy pomocy zaawansowanego edytora graficznego możliwa jest sprawna praca wizualna. Edytor umożliwia przygotowanie sceny w aplikacji oraz połączenie grafiki i kodu w interaktywne obiekty napisane w $C\#$, jak opisano w artykule [25]. Można konfigurować obiekty w edytorze, manipulować nimi podczas działania aplikacji i dostosowywać sam edytor za pomocą skryptów, które dodają nowe funkcje do menu i interfejsu użytkownika. Niezależność od platformy przejawia się nie tylko w wyborze platformy docelowej dla tworzonej aplikacji, ale także w katalogu narzędzi deweloperskich, gdyż grę można tworzyć na platformie Windows lub macOS. Edytor Unity pozwala na projektowanie gier 2D i 3D oraz treści interaktywnych. Unity jest dostarczany z pełnym zestawem narzędzi do modelowania i konstrukcji, w tym interfejsami graficznymi, dźwiękowymi i narzędziami do tworzenia środowiska. Te interfejsy wymagają przynajmniej zewnętrznego oprogramowania do zarządzania projektem.

Unity umożliwia najprostsze z dostępnych na rynku wdrożenie międzyplatformowe, gdyż stosunkowo mało aplikacji obsługuje tak wiele platform wdrażania docelowego.

Widok Scena pozwala na manipulację obiektami, co umożliwia ujrzenie całkowitej liczby obiektów reprezentowanych przez różne ikony i kolorowe linie, takie jak kamery, światła i strefy kolizji. Pokazany tutaj widok różni się od uruchomionej symulacji, ponieważ można oglądać scenę bez ograniczania się widokiem symulacji. Scena zawiera środowiska oraz menu aplikacji - każdy plik sceny to inny poziom. Otoczenie, przeszkody i tło są umieszczane w każdej scenie, zasadniczo poprzez projektowanie i tworzenie aplikacji jedna po drugiej.

3.1.1. ML-Agents

Uczenie maszynowe (ML) to bardzo szeroki temat, dlatego na potrzeby pracy dyplomowej przedstawione zostaną jedynie metody i algorytmy używane przez agentów ML. Korzystając z wtyczki ML-Agents, możliwe jest szkolenie inteligentnych agentów w zakresie używania sieci neuronowych do rozwiązywania różnych problemów. Jak stwierdzono w oficjalnej dokumentacji, wykonuje się to za pomocą uczenia ze wzmocnieniem (RL), uczenia się naśladowania, neuroewolucji (NE) i innych technologii, które można napisać poprzez API. Pyton. Uczenie ze wzmocnieniem (RL) można również uruchomić przy użyciu różnych algorytmów, ale domyślnie agent ML stosuje algorytm o nazwie Proximal Policy Optimization (PPO). Jest to algorytm opracowany przez OpenAI i obowiązuje tam jako domyślny algorytm RL.

Idea RL jest bardzo prosta - przeszkolony agent nie ma danych szkoleniowych, a wszystko, czego się nauczył, pochodzi z jego własnego doświadczenia poprzez wielokrotne próby maksymalizacji długoterminowych zysków.

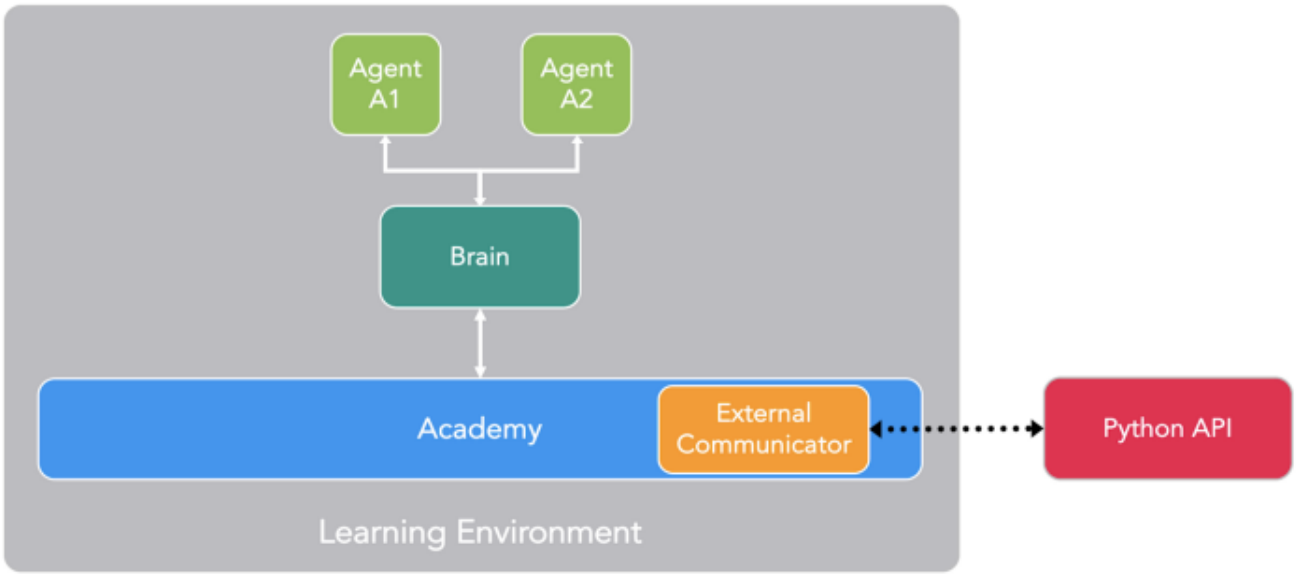
Agentowi należy udzielić porady w formie nagrody, stwierdzając czy jego czynności były poprawne czy błędne. Następnie, gdy rozpoczyna się szkolenie, agent podejmuje działania, na

podstawie których system nagradza lub karze agenta, dostosowując sieć neuronową do otoczenia. Sieci neuronowe są odpowiedzialne za podejmowanie decyzji, czyli za zachowanie.

Dzięki agentom uczenia maszynowego unika się pisania kodu sieci neuronowej od podstaw. Wystarczy powiedzieć agentowi, które parametry mają być użyte jako dane wejściowe, przypisać wyjścia i nagrody - wtyczka wykonuje resztę pracy za programistę.

Aby skonfigurować środowisko uczenia się w ML-Agent, wystarczy skonfigurować trzy komponenty: agenta, mózg i akademie:

- Agent może być dowolną postacią na scenie. Agent obserwuje, działa i jest nagradzany.
- Brain stanowi komponent, który określa zachowanie agenta - otrzymuje obserwacje i nagrody od agenta oraz zwraca operacje. Przykład komponentu znajduje się 6.2
- Academy to komponent, który kontroluje wszystkie ustawienia środowiska i współdziała z API Pythona.



Rys. 3.1: Schemat uczenia maszynowego ML-Agent

Źródło: Dokumentacja Unity ML-Agents

PyTorch

Biblioteka PyTorch umożliwia przeprowadzanie obliczeń na bazie wykresów przepływu danych. Platforma ta zapewnia elastyczność i szybkość w kwestii prac badawczych dotyczących głębokiego uczenia. PyTorch oferuje funkcje takie jak obliczenia tensora z silnym wsparciem przyspieszania GPU i budowanie głębokich sieci neuronowych na taśmach opartych na systemach typu autograd, które służą do automatycznego wyznaczania elementów wektora gradientu. Biblioteka ta, przeznaczona jest do języka Python, pomaga w przetwarzaniu języka naturalnego w aplikacjach. PyTorch wyposażony jest w łatwy w obsłudze interfejs oraz prostą konfigurację. Można korzystać z jego usług na platformach chmurowych i nie wymaga definiowania całego wykresu a priori, dzięki czemu debuggowanie jest przystępniejsze.

TensorBoard

W przypadku uczenia modeli za pośrednictwem biblioteki PyTorch, należy określić wartości hiperparametrów. Proces znajdowania tych atrybutów może zająć kilka iteracji, dlatego stosuje

się narzędzie TensorBoard, które odpowiada za wizualizację. Pozwala ono zobrazować atrybuty agentów takie jak nagrody przydzielane podczas szkolenia, co usprawnia precyzowanie optymalnych wartości dla środowiska. Ponadto TensorBoard zapewnia śledzenie metryk eksperymentu, strat, gradientów i wyjść pośrednich.

3.2. Github

Git to oprogramowanie, szerzej opisane w [15], które śledzi zmiany w kodzie źródłowym i przechowuje modyfikacje w centralnym repozytorium. Stanowi to duże ułatwienie dla deweloperów, gdyż mogą pobrać nową wersję oprogramowania, wprowadzić zmiany i przesłać najnowszą wersję. Z kolei GitHub, przedstawiony na stronie internetowej [16], stanowi witrynę obsługującą projekty programistyczne przy użyciu systemu kontroli wersji Git. Z tego względu należy rozgraniczyć Git jako narzędzie oraz Github jako usługę, w której można przechowywać projekty Git.

Commit to zmiana pliku lub zestawów plików, którą zapisuje się wewnątrz repozytorium za pomocą akceptacji. Taka akceptacja przybiera postać migawki danego repozytorium, a wszystkie akceptacje składają się na historię takiego repozytorium.

Pull oznacza proces pobierania i scalania zmian, który dokonuje uaktualnienia obecnej lokalnej gałęzi oraz gałęzi przynależących do śledzenia zdalnego. Termin push odnosi się do wysyłania zatwierdzonych zmian, które są dokonywanych lokalnie do repozytorium zdalnego np. hostowanego na Github.

Repozytorium stanowi podstawowy element Github – analogiczny do folderu projektu, w którym przechowywany jest program. W obrębie takiego repozytorium znajdują się wszystkie pliki w projekcie oraz historia zmian dla każdego pliku - tutaj także uruchamiany jest Git.

Rozdział 4

Część teoretyczna

4.1. Wstęp

Uczenie ze wzmocnieniem tzw. reinforcement learning (RL) ma na celu uczenie się wiedzy proceduralnej - jest to obliczeniowe podejście do uczenia się celowego zachowania na podstawie interakcji. Algorytm w uczeniu przez wzmacnianie otrzymuje informację zwrotną, która prowadzi użytkownika do uzyskania najlepszego wyniku. Uczenie przez wzmacnianie różni się od innych rodzajów uczenia maszynowego tym, że system nie jest trenowany za pomocą przykładowego zestawu danych, tylko metodą prób i błędów. Z tego powodu sekwencja właściwych decyzji spowoduje wzmocnienie procesu, gdyż stanowi on najlepsze rozwiązanie danego problemu. Krytyk jest tutaj źródłem informacji trenującej, która ma charakter wartościujący jakość działania ucznia. Podstawą uczenia ze wzmocnieniem są dynamiczne interakcje ucznia wykonującego zadanie ze środowiskiem w którym to zadanie realizuje.

Tego rodzaju oddziaływania odbywają się przy dyskretnych krokach czasowych, gdzie uczeń jest zobowiązanych obserwować kolejne stany środowiska i stosować się do obowiązującej metody działania. Nagroda przyznawana jest uczniowi, kiedy przeprowadzi on akcję w sposób prawidłowy. Dzięki temu nauczanie polega na zmianie strategii w oparciu o monitoring kolejności nagród lub akcji. Poniższy schemat prezentuje ogólne podejście uczenia się ze wzmocnieniem zaczerpnięte ze strony[22].

Scenariusz uczenia się ze wzmocnieniem

W każdym kroku czasu t:

1. obserwuj aktualny stan x_t;
2. wybierz akcję a_t do wykonania w stanie x_t;
3. wykonaj akcję a_t;
4. obserwuj wzmocnienie r_t i następny stan x_{t+1};
5. ucz się na podstawie doświadczenia $\langle x_t, a_t, r_t, x_{t+1} \rangle$.

Rozpatrując proces nauczania ze wzmocnieniem, zauważyć można pojawienie się niepewności otoczenia, czyli metamorfozy wzmocnienia oraz stanów. Ponadto, jeśli uczeń nie jest zaznajomiony z danymi na temat rozkładu prawdopodobieństwa, nie będzie on posiadał wiedzy o otoczeniu. Ze względu na fakt, że otoczenie nie jest kontrolowane, wpływ ucznia na rozkłady prawdopodobieństwa będzie znikomy.

Częstym podejściem jest założenie, że krytyk wydający nagrody stanowi część środowiska, dlatego uczeń nie może w żaden sposób na te nagrody wpłynąć. W praktyce krytyk ten może przyjąć postać inteligentnego agenta i wchodzić w skład architektury ucznia. Zadaniem ucznia

jest nauczenie się strategii, która maksymalizuje konkretne kryterium jakości definiowane poprzez dostarczane uczniowi nagrody, a rodzaj tego kryterium mówi o tym jaki typ uczenia ze wzmocnieniem przyjęto. Zadania epizodyczne będące podklasą zadań uczenia ze wzmocnieniem dzielą interakcję ucznia ze środowiskiem na serię niezależnych epizodów lub prób, zatem akcje wykonywane w danej próbie nie wpływają na nagrody wręczane w innych próbach. Szczególnym typem zadań epizodycznych jest zadanie do-sukcesu, gdzie uczeń w ramach każdej próby ma do osiągnięcia pewien cel, a próba kończy się, gdy osiągnie on sukces. Nagrody i współczynnik dyskontowania określone są w taki sposób, aby maksymalizacja kryterium jakości pozwoliła na osiągnięcie celu w jak najmniejszej liczbie kroków.

4.2. Głębokie uczenie ze wzmocnieniem

Nowy model głębokiego uczenia dla uczenia ze wzmocnieniem przedstawiono w artykule [27], w którym zademonstrowano jego zdolność do opanowania trudnych zasad sterowania w grach komputerowych Atari 2600, wykorzystując tylko surowe piksele jako dane wejściowe. Zaprezentowano również wariant online Q-learning, który łączy stochastyczne aktualizacje minibatchów z pamięcią powtórek, aby ułatwić szkolenie głębokich sieci pod kątem RL. Wdrożone podejście dało najlepsze wyniki w sześciu z siedmiu gier, w których było testowane, bez dostosowywania architektury lub hiperparametrów. Ponieważ skala wyników różniła się znacznie w zależności od gry, ustawiono wszystkie pozytywne nagrody na 1, a wszystkie negatywne na -1, pozostawiając 0 niezmienionych. W tych eksperymentach użyto algorytm RMSProp z minibatchami o rozmiarze 32. Przeszkolono łącznie 10 milionów klatek i wykorzystano pamięć odtwarzania jednego miliona ostatnich klatek. Zgodnie z wcześniejszymi podejściami do grania w gry na Atari, zastosowano również prostą technikę pomijania klatek. Dokładniej, agent widzi i wybiera akcje na każdej k-tej ramce zamiast na każdej klatce, a jego ostatnia akcja jest powtarzana na pominiętych klatkach. Wykorzystano k = 4 we wszystkich grach z wyjątkiem Space Invaders, w których zauważono, że wybór k = 4 sprawiał, że lasery były niewidoczne z powodu okresu, w którym migają. Z tego względu zdecydowano o wyborze k = 3, aby lasery były widoczne, a ta zmiana była jedyną różnicą w wartościach hiperparametrów między którąkolwiek z gier. Oprócz zauważenia stosunkowo płynnej poprawy przewidywanego Q podczas treningu, nie napotkano żadnych problemów z rozbieżnością w żadnym z wykonywanych eksperymentów. Sugeruje to, że pomimo braku jakichkolwiek teoretycznych gwarancji zbieżności, metoda jest w stanie trenować duże sieci neuronowe przy użyciu sygnału uczenia się wzmocnienia i stochastycznego spadku gradientu w stabilny sposób. Na koniec pokazano, że zaimplementowana metoda osiąga lepsze wyniki niż doświadczony ludzki gracz w Breakout, Enduro i Pong i daje wyniki bliskie ludzkim w Beam Rider. Gry Q * bert, Seaquest, Space Invaders, w których metoda jest daleko od ludzkiej wydajności, są trudniejsze, ponieważ wymagają sieci do znalezienia strategii, która rozciąga się na długie skale czasowe.

Przeglądu uczenia ze wzmocnieniem (RL) oraz głębokiego uczenia ze wzmocnieniem (DRL) dokonano w artykule [20]. Autorzy sugerowali, że w uczeniu ze wzmocnieniem należy skupić się na nauce bez dostępu do podstawowego modelu środowiska, ale interakcje ze środowiskiem można wykorzystać do poznania funkcji wartości, zasad, a także modelu. Metody RL bez modeli uczą się bezpośrednio na podstawie interakcji ze środowiskiem, natomiast metody RL oparte na modelach mogą symulować przejścia przy użyciu wyuczonego modelu, co skutkuje zwiększoną wydajnością próby. Wiele sukcesów w głębokim uczeniu ze wzmocnieniem wynikało ze skalowania wcześniejszej pracy nad RL do problemów wielowymiarowych. Wynikało to z uczenia się niskowymiarowych reprezentacji cech i potężnych możliwości aproksymacji funkcji przez sieci

neuronowe. Za pomocą uczenia się reprezentacji DRL może skutecznie radzić sobie z przekleństwem wymiarowości, w przeciwieństwie do tabelarycznych i tradycyjnych metod nieparametrycznych. Na przykład konwolucyjne sieci neuronowe (CNN) mogą być wykorzystywane jako składniki agentów RL, umożliwiając im naukę bezpośrednio na podstawie surowych, wysokowymiarowych danych wejściowych. Ogólnie rzecz biorąc, DRL opiera się na uczeniu głębokich sieci neuronowych w celu przybliżenia optymalnej polityki i/lub optymalnych funkcji wartości V (stanu), Q (jakości) i A (przewagi). Pomimo sukcesów DRL metodami bez gradientowymi, zdecydowana większość obecnych prac opiera się na gradientach, a tym samym na algorytmie wstecznej propagacji. Podstawową motywacją jest to, że gradienty, jeśli są dostępne, zapewniają silny sygnał uczenia się. W rzeczywistości gradienty te są szacowane na podstawie przybliżeń, próbkowania lub w inny sposób, i jako takie musimy opracować algorytmy z użytecznymi błędami indukcyjnymi, aby były one wykonalne.

4.3. Uczenie ze wzmocnienie w narzędziu Unity

Metoda uczenia ze wzmocnieniem została rozwinięta w artykule [31], gdzie autorzy dokonali szkolenia agentów uczenia maszynowego przy pomocy platformy Unity. W artykule zaprezentowano środowisko, w którym agent szkolony metodą uczenia się przez wzmacnianie próbuje uciec od tradycyjnej sztucznej inteligencji zaimplementowanej w nowoczesnym komercyjnym silniku gier wideo. Na początku opisano narzędzie jakim się posługiwano - ML-Agent, czyli wtyczkę Unity, która umożliwia połączenia silnika Unity z uczeniem ze wzmocnieniem. Następnie przedstawiono trzy scenariusze: ukrywanie, unikanie oraz ucieczkę. W ukrywaniu agent reprezentowany przez maszynę uczącą musiał unikać pola widzenia agenta sztucznej inteligencji. Na początku efekty uczenia były słabe i niestabilne. Po zwiększeniu rozmiaru bufora (dane wejściowe), wyniki uległy poprawie. Ostatecznie chowający się agent nie miał szans z agentem szukającym, więc porzucono dalsze badania tego scenariusza. Scenariusz unikania nie był rozpatrywany. Scenariusz ucieczki polegał na tym, że jeden agent maszyny uczącej uciekał przed agentem sztucznej inteligencji. Badania te przyniosły zadowalające efekty. Prace testowe pokazały, że badania nad uczeniem się przez wzmacnianie można zastosować do opracowania agenta szkoleniowego. Wyniki szkolenia agentów uczenia maszynowego w dużym stopniu zależą od wielu czynników, takich jak ustawienie hiperparametrów, zasady środowiska, projektowanie nagród i kar oraz czas szkolenia, zatem w większości przypadków lepiej jest opracować tradycyjną sztuczną inteligencję zaimplementowaną poprzez maszyny stanowe lub drzewa zachowań. Istnieją jednak pewne zalety stosowania tej metody - wydaje się, że takie podejście daje naturalne i nieoczekiwane zachowanie lub przynajmniej zachowanie, które może być postrzegane jako bardziej autentyczne niż niektóre programy tradycyjnej sztucznej inteligencji. W scenariuszu ucieczki, przy ustaleniu odpowiednich hiperparametry i inteligentnej konfiguracji nagród/kar w połączeniu z programem nauczania, optymalizacja w całkiem prostych scenariuszach jest możliwa.

4.4. Reguła rozmyta w uczeniu ze wzmocnieniem

Wiarygodność reguł rozmytych otrzymanych w procesie uczenia ze wzmacnianiem opisano w artykule [32], gdzie obiektem badań był miniaturowy robot mobilny Khepera, który ze względu na niewielkie wymiary i małą wagę stanowił dobry obiekt do testowania algorytmów sterowania. Robot ten wyposażony był w osiem czujników RC, pozwalających na wykrywanie przeszkód. Wartość sygnału z czujnika stanowiła liczbę całkowitą z przedziału [0, 1023], gdzie 0 to brak przeszkody, a 1023 to bardzo mała odległość czujnika od przeszkody. W artykule zaprezentowano układ nawigacyjny realizujący zadanie omijania przeszkód w oparciu o regulator rozmyty Takagi-Sugeno, zdefiniowano współczynnik wiarygodności reguł otrzymanych w wyniku uczenia oraz zaprezentowano wyniki sterowania robotem mobilnym Khepera. Działanie reguły rozmytej polegało na zdefiniowaniu i przydzieleniu odpowiednio szeregu wartości sygnału z czujnika do wartości Big, gdzie otrzymywano informację czy przeszkoda jest blisko i Small mówiącą że przeszkoda znajdowała się daleko od robota, bądź w ogóle jej nie było. Uczenie następników reguł odbywało się w trakcie poruszania się robota w otoczeniu nieruchomych, różnokolorowych przeszkód, a przerywano je po pokonaniu przez robota odległości 1m (czas uczenia = 1,5 min). Na podstawie badań wyciągnięto wnioski, że liczba przeszkód jest zbyt niska, ponieważ najczęściej uczeniu poddawano regułę pierwszą odpowiedzialną za sytuację braku reguł. Dodatkowo należałoby „zagęścić" przeszkody w taki sposób, aby uczone były również reguły opisujące stan, gdy w pobliżu robota występuje więcej niż jedna przeszkoda. Informacja o tym, które reguły nie były uczone w wystarczającym stopniu służy zmianie warunków uczenia, co doprowadzić może do bardziej równomiernego ich uczenia. Zaproponowany przez autora współczynnik znajduje zastosowanie w większości systemów wnioskowania rozmytego oraz przy różnych algorytmach uczenia.

4.4.1. Reguła rozmyta w robotach Sumo

Artykuł [23], który również podejmuje tematykę reguły rozmytej prezentuje zastosowanie hybrydowego systemu Neuro-Fuzzy (NF) do sterowania robotem Sumo Robot (SR), który jest często projektowany przez studentów do zawodów robotów. Architektura SR oparta jest na nieholonomicznej autonomii robota z napędem różnicowym. Robot ten zbiera dane o środowisku i wysyła polecenia ruchu do prawego i lewego silnika prądu stałego. Sterowanie robota odbywa się poprzez obracanie silników z różnymi prędkościami kątowymi. Kiedy silniki poruszają się z tą samą prędkością, robot jedzie na wprost, natomiast ruch silników w przeciwnych kierunkach skutkuje obrotem robota w miejscu. Kiedy jeden silnik obraca się wolniej niż drugi, robot przesunie się po łuku w stronę wolniejszego obrotu. W zawodach SR każdy robot stara się jak najszybciej znaleźć przeciwnika i wypchać go z ringu. Ze względu na fakt, że relacja między sygnałami wyjściowymi czujników i impulsami sterującymi silnikami jest wysoce nieliniowa w SR, można wykorzystać techniki obliczeń miękkich do zdefiniowania takiej nieliniowej zależności i sterowania robotem w ringu zawodowym. Podanie inteligentnych metod sterowania SR oprócz uproszczenia sterowania robotem i poprawy jego reakcji podczas zawodów, zachęca również programistów do korzystania z inteligentnych metod rozwiązywania rzeczywistych światowych problemów. W związku z tym zaproponowano i wdrożono kontroler NF do sterowania SR. Opracowano rozmyty system wnioskowania (FIS) do wykrywania i śledzenia przeciwnika na ringu, który wiąże sygnały wyjściowe czujnika z impulsami sterowania silnikiem. Do ekstrakcji reguł i dostrajania parametrów FIS posłużono się algorytmem opartym na sztucznych sieciach neuronowych (ANN). W trakcie badań zastosowano nowy algorytm kontroli do nadzorowania SR. W wykorzystanym kontrolerze zaaplikowano kombinację zalet kontrolera rozmytego i ANN, a następnie opracowano i wdrożono sterownik oparty na technologii NEFCLASS za pomocą rozmytego mikrokontrolera. NEFCLASS dostraja parametry

FIS, dodając i usuwając reguły – pomaga przy budowie FIS z danych systemu i zapewnia interpretowalność opracowanych klasyfikatorów. W przeprowadzonym eksperymencie SR próbował znaleźć nieruchomego przeciwnika, który znajdował się po lewej stronie robota. Kolejno robot rozpoczynał ruchy w trybie wędrówki, obracając się w lewo. W porównaniu z poprzednim badaniem zastosowanie algorytmu uczenia się zapewniło 30% redukcję reguł, co poprawiło szybkość reakcji robota na ringu zawodów i umożliwiło zaatakowanie przeciwnika przy minimalnych manewrach. Wydajność zastosowanej kontroli można ulepszyć, dodając dodatkowe czujniki do wykrywania ataku przeciwnika podczas zawodów. Przedstawiono także w sposób szczegółowy podejście projektowe proponowanego kontrolera oraz skuteczność kontrolera wykazaną przez implementację sprzętu i wyniki eksperymentalne, co dowiodło, że metody sterowania mogą być łatwo stosowane przez studentów w różnych zawodach robotów.

4.5. Metody Q-learning: podstawowa, ϵ-greedy i symulowane wyżarzanie w robotach Line Follower

Twórcy artykułu [28] w celu sterowania robotem podążającym za linią (Line Follower Robot) zaproponowali metodę Q-learning. Konwencjonalnym kontrolerem dla tego typu robotów jest regulator proporcjonalny (P). Biorąc pod uwagę nieznane mechaniczne charakterystyki robota i niepewności, takie jak tarcie i śliskie powierzchnie, modelowanie systemu i projektowanie kontrolerów mogą być niezwykle trudne. W artykule przedstawiono modelowanie matematyczne robota, a na jego podstawie zaprojektowano symulator. Podstawowe metody Q-learning - czysta eksploatacja i ϵ-greedy metody, które pomagają w eksploracji, mogą również zaszkodzić działaniu kontrolera po zakończeniu nauki przez odkrywanie nieoptymalnych działań. Metoda Q-learning oparta na symulowanym wyżarzaniu (SA) rozwiązuje tę wadę, zmniejszając tempo eksploracji, gdy nauka wzrasta. Najpopularniejszym kontrolerem dla robotów śledzących linię jest kontroler proporcjonalny. Aby zaprojektować regulator P, należy znać dokładne parametry systemu, co stanowi wyzwanie. Dlatego metoda prób i błędów jest najczęstszą techniką dostrajania parametrów sterownika, ale nie jest ona optymalna. W artykule przedstawiono oparty na SA kontroler Q-learning do optymalnego sterowania robotem. Symulacja i wyniki eksperymentalne pokazują skuteczność bazowego Q-learning MIMO (Multiple Input Multiple Output) SA. Ponadto porównanie proponowanej metody z trzema różnymi metodami, w tym Q-learning na bazie MISO (Multiple Input Single Output) SA, ϵ-greedy Q-learning MISO oraz regulatora proporcjonalnego P ma na celu wyjaśnienie zalet proponowanej metody. Zgodnie z oczekiwaniami, tradycyjny robot sterowany P działał podobnie we wszystkich odcinkach, jednak wydajność robotów opartych na Q-learning poprawiała się w miarę przechodzenia przez proces uczenia się. Jak pokazały wyniki symulacji, wydajność zarówno ϵ-greedy Q-learning MISO, jak i Q-learning na bazie MISO SA poprawiła się przez iterację, dopóki kontroler nie został przeszkolony. Po nauczeniu się procesu eksploracja w ϵ-greedy była kontynuowana i powodowała nieoptymalne wyniki, przeciwnie, technika oparta na SA zmniejszała eksplorację przez wzrost uczenia się. Ograniczenie kontroli zmniejszyło skuteczność MISO przy zaledwie pięciu trybach kontroli. Niemniej jednak kontroler MIMO, który miał 421 trybów sterowania, działał lepiej niż pięć trybów uczenia MISO. Proces uczenia się kontrolera MIMO był dłuższy w porównaniu do kontrolera MISO, jednak po wystarczającym treningu wydajność osiągnęła lepsze rezultaty niż we wszystkich symulowanych kontrolerach. Wszystkie wyniki kontrolerów mieściły się w przedziale od 70 do 80, co udowodniło, że wszyscy byli w stanie dotrzeć do punktu końcowego, ale metoda ϵ-greedy miała tendencję do uzyskiwania nieoptymalnych wyników, ponieważ nie przerywała eksploracji. Niewielka różnica wynika z kary czasowej, a uczenie Q-learning MIMO SA z dokładniejszymi odważnikami było w stanie płynniej i szybciej podążać za linią.

Rozdział 5

Implementacja

5.1. Komponent Unity Wheel Collider

Wheel Collider jest przeznaczony dla pojazdów z uziemieniem, głównie kołowych. Jego celem jest wykrywanie kolizji kół poprzez rzutowanie promienia środka w dół przez oś Y (lokalną). Wyposażony jest on we wbudowane wykrywanie kolizji, właściwości fizyczne kół i wzorce tarcia opon. Koła posiadają promień i rozciągają się w dół w zależności od odległości zawieszenia. Model tarcia bazujący na poślizgu, stosowany jest w zderzakach koła do określenia tarcia bez uwzględniania reszty silnika fizycznego. Pomaga to osiągnąć realizm w zachowaniu pojazdu, ale skutkuje zarazem zignorowaniem domyślnych właściwości fizycznych materiału. Zapewnienie poprawnej geometrii kolizji z torem wyścigowym jest kluczowe, gdyż pojazdy mogą osiągać duże prędkości.

Wśród właściwości i funkcji zderzaka koła wyszczególnia się:

- Force App Point Distance - odległość od punktu przyłożenia projektowanej siły - określić parametry punktu przyłożenia siły koła (przyjmując odległość do siły działającej na koło jest wyrażona w metrach). Spód koła, nacisk położony jest na kierunek ruchu zawieszenia)
- Center - środek koła
- Mass - masa koła
- Promień - promień koła
- Target Position - statyczna odległość zawieszenia wzdłuż długości zawieszenia (domyślnie 0,5, co odpowiada osiągom konwencjonalnego zawieszenia samochodowego)
- Spring - siła sprężyny
- Suspension Spring - sprężyna zawieszenia
- Forward/Sideways Friction - siła tarcia opon, gdy koło porusza się do przodu i z boku
- Wheel Damping Rate - stopa amortyzacji koła przedmiotu-Kwota amortyzacji zastosowana do koła
- Suspension Distance - odległość zawieszenia - maksymalny rozstaw kół zawieszenia Odległość mierzona w przestrzeni lokalnej (zawieszenie zawsze rozciąga się w dół wzdłuż lokalnej osi Y)
- Damper - amortyzator ciągu, należy zmniejszyć prędkość zawieszenia, a wyższa wartość tego ustawienia spowolni ruch sprężyny zawieszenia.

5.2. Silnik

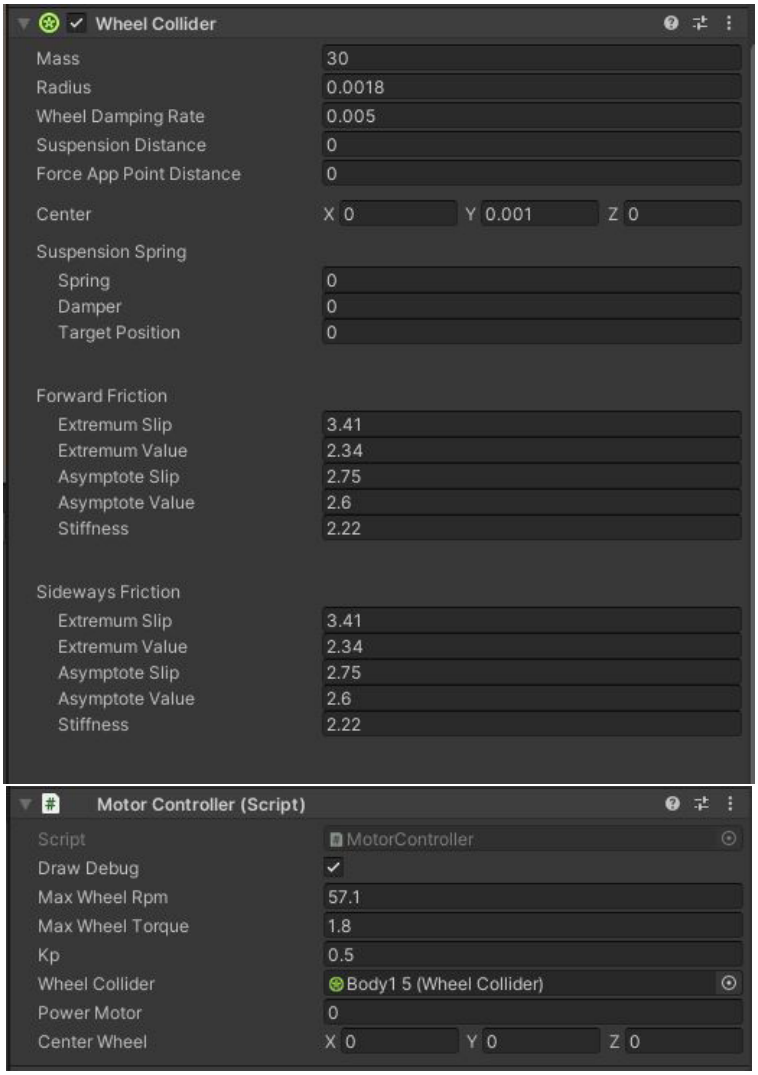
Za sterowanie silnikiem odpowiada zmienna powerMotor, co obrazuje kod 5.1. Jeżeli jest to liczba dodatnia, koło będzie się obracać do przodu, natomiast dla liczby ujemnej koło obróci się do tyłu. Należy ustalić Torque, będący momentem obrotowym ograniczonym przez maksymalny moment obrotowy silnika.

Funkcja Mathf.Clamp pozwala ograniczyć moc silinków do wartość przyjętej przez zmienną motorTorque.

Listing 5.1: Funkcja poruszania się koła: Run.

```
public void Run() {
 torqueWheelCollider = (powerMotor * maxWheelColliderRpm - wheelCollider.rpm) * Kp;
 wheelCollider.motorTorque = Mathf.Clamp(torqueWheelCollider,
                          -maxWheelColliderTorque, maxWheelColliderTorque);
}
```

Komponent Wheel Collider przedstawiony w dokumencie [10], odpowiada za modelowanie silnika, jak pokazano na obrazku 5.1. Oprócz parametrów środka i promienia, które przedstawiono w sekcji 5.1, podawane są również parametry dotyczące masy i prędkości tłumienia kół, których modyfikacja może zmieniać ruch symulowanego robota.

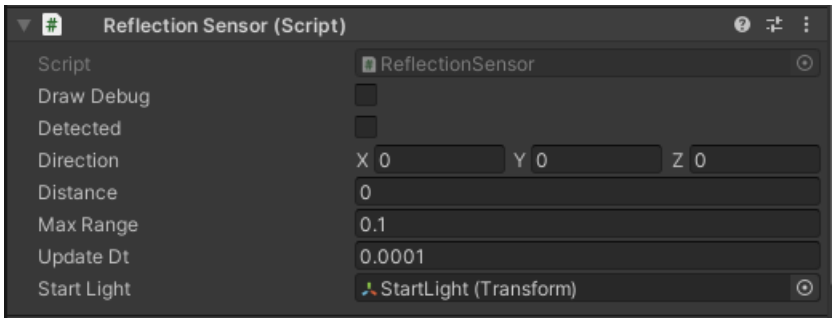


Rys. 5.1: Komponenty do symulowania silnika

5.3. Czujnik odbiciowy

Czujnik modelowany jest przez skrypt Reflection Sensor, jak pokazano na rysunku 5.2.
Jego główne parametry to:

- Max Range - maksymalna odległość wyrażona w metrach. [m]
- Update Dt - czas aktualizacji czujnika, w sekundach [s]



Rys. 5.2: Komponent do symulowania czujnika odbiciowego

Symulacja czujnika odbiciowego wskazanego w kodzie 5.2 jest wykonywana przez rysowanie pojedynczej linii w ograniczonym zasięgu maxRange. Czujnik zwraca informację "prawdę"w przypadku wykrycia czarnej powierzchni, a "fałsz"w przypadku białej.

Listing 5.2: Funkcja skanująca obszar widzenia sensora odbiciowego: `Scanning`.

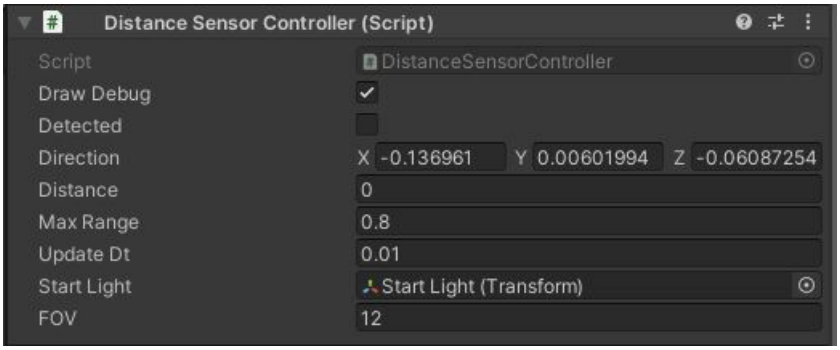
```
private IEnumerator Scanning()
{
    while (scanning)
    {
        accessibility = maxRange;
        Quaternion rotation = Quaternion.Euler(new Vector3(0, 1, 0));
        Vector3 Direction = rotation * startLight.forward;
        scan = Cast(Direction);
        if (scan < accessibility) accessibility = scan;
        Direction = startLight.forward * accessibility;
        distance = accessibility;
        if (accessibility < maxRange) Detected = true;
        else Detected = false;
        yield return new WaitForSeconds(updateDt);
    }
}
```

5.4. Czujnik odległościowy

Czujnik modelowany jest przez skrypt Distance Sensor Contoller, jak pokazano na rysunku 5.3.
Jego główne parametry to:

- Fov (field of view) - pole widzenia, w stopniach [°]
- Max Range - maksymalna odległość wyrażona w metrach. [m]
- Update Dt - Czas aktualizacji czujnika, w sekundach [s]

Symulacja czujnika odległościowego wskazanego w kodzie 5.3 jest wykonywana przez nakreślenie linii obrazujących pole widzenia FOV w ograniczonym zasięgu maxRange. W celu



Rys. 5.3: Komponent do symulowania czujnika odległościowego

zdefiniować kierunek dla każdej linii ustalany jest punkt kierunkowy. Czujnik zwraca informację "prawdęlub "fałsz"w przypadku wykrycia obiektu. Dostarcza także dane o odnośnie odległości od znalezionego elementu

Listing 5.3: Funkcja skanująca obszar widzenia sensora odległościowego: `Scanning`.

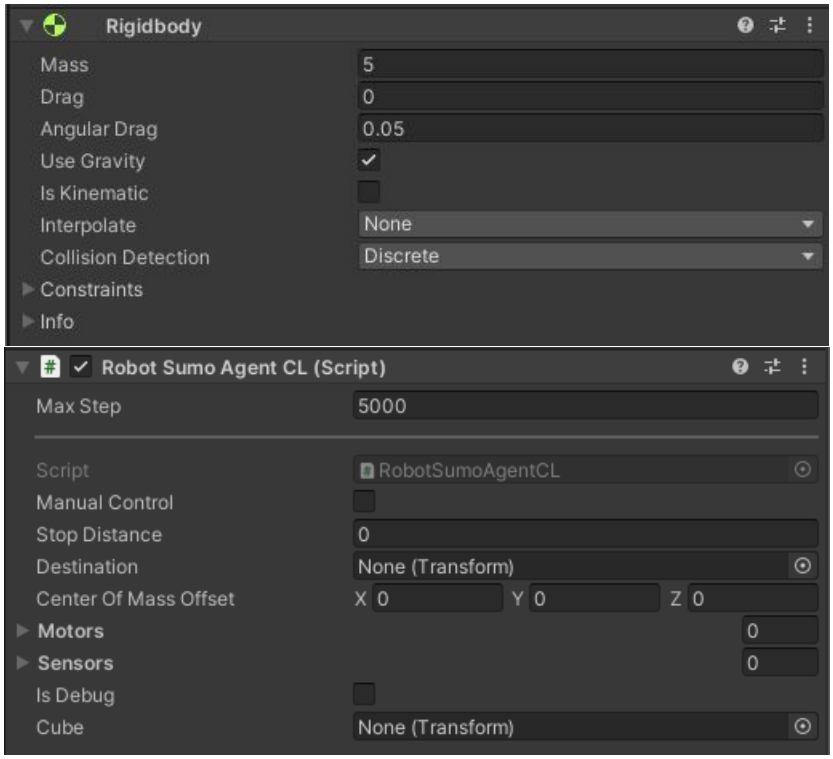
```
private IEnumerator Scanning()
{
    while (scanning)
    {
        accessibility = maxRange;
        for (float a = -FOV; a < FOV; a += 0.5f)
        {
            Quaternion rotation = Quaternion.Euler(new Vector3(0, a, 0));
            Vector3 Direction = rotation * startLight.forward;
            scan = Cast(Direction);
            if (scan < accessibility) accessibility = scan;
        }

        distance = accessibility;
        Direction = startLight.forward * accessibility;
        if (accessibility < maxRange ) Detected = true;
        else Detected = false;
        yield return new WaitForSeconds(updateDt);
    }
}
```

5.4.1. Środowisko

Wszystkie elementy, poza robotem, posiadające Collider stanowią środowisko. Collider przedstawiony w dokumentacji [7] stanowi komponent zderzaka zdefiniowany dla fizyki. Unity umożliwia dodawanie obiektów online za pośrednictwem zakładki GameObject, a następnie wybranie obiektów 3D. Przedmiotami tymi mogą być podłoga, ściany lub przeszkody robota. Jeśli robot musi poruszać obiektami umieszczonymi na scenie, należy uwzględnić elementy sztywnego ciała (Rigidbody) z odpowiednimi ustawieniami masy. Aby dodać sztywną bryłę, wystarczy skierować się do Inspector, wybrać Add Component i kliknąć Rigidbody. Rigidbody jest głównym komponentem, który zapewnia fizyczne zachowanie obiektu, opisanym w [9].

5.4.2. Robot

Robot pełni rolę hierarchicznego rodzica wszystkich komponentów robota - silników i czujników odległości. Robot ten symulowany jest poprzez komponent Rigidbody, przedstawiony w dokumentacji [9] i zaprezentowany na rysunku 5.4, dzięki czemu prawidłowo reaguje na grawitację. Ważnym parametrem jest masa (modyfikowana w Rigidbody) i środek masy robota - określany przez Unity, ale można go zmienić, modyfikując parametr Center Of Mass Offset w Robot.



Rys. 5.4: Komponenty do symulowania robota

Rozdział 6

Roboty Sumo - uczenie ze wzmocnieniem

Zawody robotów w kategorii sumo bazują są na prawdziwych walkach sumo w Japonii i stylem przypominają zapasy. Walka między dwoma autonomicznymi lub zdalnie sterowanymi robotami odbywa się na powierzchni czarnego koła (ringu) ograniczonego białą boczną linią (marginesem). Zwycięzcą zostaje robot, któremu uda się wypchnąć przeciwnika z ringu. Zwykle walka robotów sumo składa się z kilku rund ograniczonych czasowo (zwykle 3 rundy, każda runda trwa 3 minuty).

Zawody robotów sumo można podzielić na kilka podkategorii. Dotyczą one układu sterowania robota (sterowanie autonomiczne lub zdalne), typu robota (robot humanoidalny, robot Lego Mindstorms i konstrukcja mobilna DIY), średnicy ringu oraz dopuszczalnych maksymalnych rozmiarów i mas robotów.

Zasady dotyczące konkurencji mówią o tym, że w czasie ich trwania roboty wystawiane są przez uczestników do kolejnych rozgrywek. Zwycięski tytuł trafia do rąk uczestnika, którego robot zwycięży w największej liczbie rozgrywek.

Zasady walk wskazują, że w każdej bitwie biorą udział dwa roboty. Na walkę składają się trzy trzyminutowe rundy. Przed rozpoczęciem pierwszej rundy losowany jest uczestnik, który jako pierwszy ustawia robota w obrębie ringu. W drugiej i trzeciej rundzie pierwszego robota umieszcza zwycięzca poprzedniej rundy. Po ulokowaniu robota w ringu uczestnicy nie mogą w jakikolwiek sposób poruszać robotem. Robota można umieścić w dowolnym miejscu na ringu za linią startu wyznaczoną przez sędziego. Gdy pozycje obu robotów są prawidłowe, sędzia sygnalizuje rozpoczęcie meczu. Po wysłaniu sygnału gracz aktywuje swojego robota. Od tego momentu zawodnicy nie mogą już dotykać konstrukcji podczas swojej tury. Robot musi odczekać pięć sekund przed rozpoczęciem walki, ponieważ zbyt wczesny start skutkuje faulem (runda zostaje wznowiona, kolejność robotów na ringu nie zmienia się). Uczestnik, który uruchomi robota przed zezwoleniem sędziego, otrzymuje ostrzeżenie. Po drugim ostrzeżeniu runda zostaje oddana na rzecz rywala (punkty dla przeciwnika). W bitwach robotów autonomicznych roboty grają niezależnie, podczas gdy w bitwach robotów zdalnie sterowanych są one instruowane przez samych graczy za pośrednictwem komunikacji bezprzewodowej. Celem bitwy jest obezwładnienie wroga – wypchnięcie go poza obszar ringu. Jeśli jakakolwiek część ramy robota dotknie powierzchni poza pierścieniem, robot przegrywa walkę. Walkę zwycięża robot, który wygrał więcej rund niż rywal w danej walce. Podczas rundy gracze mogą w dowolnym momencie wspólnie zdecydować, czy natychmiast zakończyć rundę i rozpocząć następną. Przerwana w ten sposób runda kończy się remisem. Rundę można przerwać i kontynuować w wyznaczonym czasie, gdy roboty są zablokowane i nie mogą wykonać żadnej czynności, oba roboty dotykają zewnętrznego obszaru ringu w tym samym czasie lub kiedy sędzia nie może

określić zwycięzcy. Jeśli robot ulegnie uszkodzeniu podczas rundy i nie jest w stanie kontynuować walki, zawodnik może zażądać zawieszenia rozgrywki. W każdej fazie zawodów całkowita liczba przerw wniesionych przez zespół wynosi maksymalnie 2, a maksymalny czas przerwy w rundzie wynosi pięć minut. Walka zdalnie sterowanymi robotami odbywa się za pomocą komunikacji bezprzewodowej wykonywanej przez uczestnika, gdzie roboty mogą poruszać się tylko na podstawie poleceń wysyłanych przez zawodników.

Zasady odnoszące się do ringu określają go jako arenę będącą czarnym kołem oddzielonym białym marginesem. Powierzchnia areny jest okrągła, a średnica odpowiada regułom gry i znajduje się ponad powierzchnią strefy zewnętrznej ringu. Powierzchnia z której wykonany jest ring to materiał o niskiej wartości współczynnika tarcia. Ponadto obszar ten musi być płaski, gładki i czarny.

Zasady ukierunkowane na konstrukcję robota dopuszczają wymiary takie jak: długość, szerokość i wysokość, a także masę robota sprecyzowane w regulaminie konkursu. Po rozpoczęciu walki (co najmniej pięć sekund po sygnale o rozpoczęciu rundy) robot może zmienić swój rozmiar (zwiększyć swój rozmiar poprzez wysunięcie pozostałych ramion). Robot musi poruszać się autonomicznie, dlatego zabrania się jakiejkolwiek komunikacji z robotem w trakcie trwania rundy. System sterowania urządzenia musi umożliwiać automatyczne uruchomienie robota co najmniej pięć sekund po włączeniu zasilania przez operatora. Używanie materiałów powodujących przywieranie robota do podłoża (klej, przyssawki) jest niedopuszczalne, a robot nie może zawierać żadnych urządzeń, które aktywnie zakłócają działanie układu sterowania konkurenta. Zabrania się również używania magnesów lub elektromagnesów w celu zwiększenia przyczepności robota. Stosowanie elementów, które mogłyby uszkodzić arenę, jest niedozwolone, zatem robot nie może zawierać żadnych pocisków ani wyposażenia emitującego dużo ciepła (miotaczy ognia). Rywalizacja robotów, a zwłaszcza zawody sumo, cieszą się coraz większą popularnością wśród miłośników robotów w Polsce i na świecie. Świadczy o tym wzrost liczby zawodów i postęp opracowywanych corocznie robotów.

6.1. Opis techniczny robota

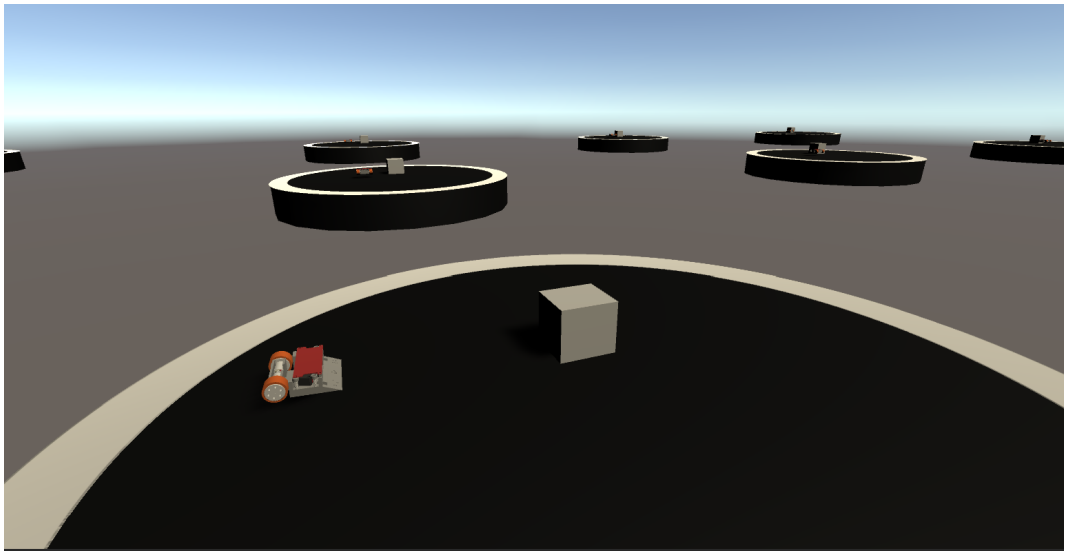
Robot wykorzystany do badań należy do klasy robotów 2.0, zatem wyposażony jest w dwukołowy napęd różnicowy. Urządzenie zawiera trzy czujniki odległościowe oraz dwa czujniki odbiciowe zamieszczone na przedniej stronie modelu. Rozmieszczenie czujników odległościowych przedstawiono na rys. 6.3. Takie ułożenie pozwało utrzymywać przeciwnika idealnie na środku trasy.

Czujniki odległościowe bazują na Sharp GP2Y0A21YK0F z doumentacją dostępną [1]. Czujnik podczerwony pozwala wykrywać obiekty w odległości od 10 do 80 cm. Wyjściem jest sygnał analogowy, którego wartość zależna jest od odległości pomiędzy wykrytym obiektem a sensorem. Im obiekt znajduje się bliżej, tym napięcie na wyjściu jest wyższe. W symulowanym czujniku odległość tą określono od 0 do 80 cm. W przypadku gdy obiekt jest poza zasięgiem czujnika, zwraca on wartość maksymalną. Pominięto także konwersję z jednostki miary odległości na napięcie - zastosowano od razu pomiar odległości w metrach. W rzeczywistym czujniku podczerwonym, do prawidłowego działania potrzebna jest poprawna jakość czujników. Można zastosować technikę dopasowania krzywej dla czujników podczerwieni lub filtrowanie Kalmana dla czujników podczerwieni i ultradźwiękowych. Jest to przydane ze względu na to, że rzeczywiste czujniki nigdy nie działają identycznie. W symulowanym czujniku nie ma takiego problemu. Z dokumentacji zaczerpnięto również informacje o czasie odświeżania, który wynosi 0.38ms oraz polu widzenia (FOV) 12°.

Czujniki odbiciowe za symulowano na podstawie czujnika KTIR0711S z dokumentacją dostępną na stronie [2]. Nadajnik podczerwieni służy do transportu wiązki promieniowania kierowanej z czujnika, dzięki czemu można określić natężenie światła, które odbija fototranzystor. Sygnał napięciowy stanowi wyjście uzależnione od natężenia światła padającego na fotodetektor. Jeżeli dużo światła zostanie odbite i trafi do detektora - napięcie na wyjściu wzrośnie. Jako że promieniowanie świetlne lepiej odbija powierzchnię jasną (a ciemną pochłania), dlatego napięcie będzie wyższe na białym materiale. W symulacji czujnika znajduje się wartość bool. Na czarnej powierzchni zwracana jest wartość jeden, a na białej wartość zero. Z dokumentacji zaczerpnięto również informację o czasie odświeżania, który wynosi $20\mu s$.

6.2. Proces uczenia

Proces uczenia dla każdej z metod wyglądał tak samo. Na początku robot typu sumo mierzył się z nieruchomym przeciwnikiem w postaci sześcianu, jak pokazano na obrazie 6.1. Widać na nim, że środowisko wraz z robotem zduplikowano wielokrotnie, aby przyspieszyć proces uczenia. Liczba kopii była różna, gdyż na początku powielanie wynosiło 16, ale ze względu na sprzęt widać było znaczący spadek klatek na sekundę, dlatego też później liczba ta wynosiła 6-8 duplikatów.



Rys. 6.1: Pierwszy etap uczenia

Kolejny etap uczenia polegał na zmierzeniu się z ruchomym celem, który był dokładną kopią robota. Spowodowało to dwukrotne zwiększenie agentów. Ostatni etap uczenia polegał na walce z robotem, który był trenowany inną metodą.

Ważnym elementem uczenia była zmiana położenia początkowego robotów na arenie. Odbywało się to zawsze w sposób losowy z zachowaniem innego obszaru dla każdego robota. Jest to doskonała metoda w walce z przeuczeniem. Zostało to wprowadzone ze względu na zróżnicowanie działania robotów. Dzięki temu nie były one w stanie nauczyć się jednej konkretnej sytuacji.

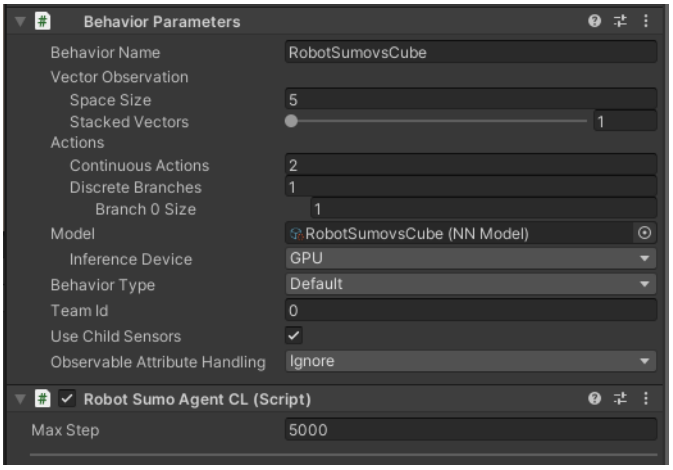
Za każdym razem robot, który spadł z ringu był karany wartością -1, z kolej jeżeli przeciwnik wypadł pierwszy, robot był nagradzany wartością +1. Takie zasady są adekwatne do przebiegu zawodów sumo, ale niewystarczające, dlatego wprowadzono dodatkowe systemy nagradzania oraz przetwarzania danych wejściowych:

- Logika wielowartościowa opisana w sekcji 6.3.1
- Funkcja nagradzania opisana w sekcji 6.3.2

Agent

Agent to robot, który obserwuje i podejmuje działania w otoczeniu. Bazowy obiekt Agent ma kilka właściwości, które wpływają na jego zachowanie:

- Behavior Parameters - każdy agent musi mieć komponent określający zachowanie. Zachowanie określa, w jaki sposób agent podejmuje decyzje. Każdy robot posiadał identyczny komponent 6.2 różniły się tylko nazwą przy parametrze "Behavior Name".
- Max Step - określa, ile kroków symulacji może wystąpić przed zakończeniem odcinka agenta. Każdy robota uruchamia się ponownie po 5000 krokach. Jest to bardzo pomocne przy rozwiązywaniu problemów opisanych 6.6.



Rys. 6.2: Komponent Behavior Parametrs (Brain) definiujący zachowanie robota sumo

Z ważniejszych elementów komponentu Behavior Parametrs są:

- Behavior Parameters : Vector Observation Space
  Agent przed podjęciem decyzji zbiera obserwacje dotyczące swojego stanu na świecie. Obserwacja wektora jest wektorem liczb zmiennoprzecinkowych, które zawierają istotne informacje potrzebne agentowi do podejmowania decyzji.
  Parametry zachowania dla robotów sumo używa Space Size równego 5. Oznacza to, że wektor funkcja zawiera na wejściu procesu uczącego wartości otrzymane z czujników. Były one różnie przetwarzane ze względu na metody.
- Behavior Parameters : Actions
  Agent otrzymuje instrukcje w formie czynności. Zestaw narzędzi ML-Agents dzieli akcje na dwa typy: ciągłe i dyskretne. Robot sumo jest zaprogramowany do używania ciągłych działań, które są wektorem liczb zmiennoprzecinkowych i mogą się zmieniać w sposób ciągły. Mówiąc dokładniej, używa on Space Size o wartości 2 do kontrolowania wyjścia procesu uczącego użytego do sterowania mocą silników. Dla wszystkich metod wyglądało to tak samo jak w przedstawionym kodzie 6.1. Wartość zwracana przez `actions.ContinuousActions` mieści się w przedziale 0-1. W celu prawidłowego działania kół wartość ta została przemnożona przez 30.

Listing 6.1: Funkcja sterująca robotem przy pomocy przez Agenta: `OnActionReceived`.

```
public override void OnActionReceived(ActionBuffers actions)
{
    base.OnActionReceived(actions);
    for (int i = 0; i < motors.Length; i++)
        motors[i].powerMotor = actions.ContinuousActions[i] * 30f;
}
```

Metody uczenia w Ml agents

ML-Agents Toolkit stanowi narzędzie przeznaczone do tworzenia gier, które wymagają implementowania złożonych zachowań postaci w przypadku głębokiego uczenia ze wzmocnieniem (DRL). DRL stosuje się, aby nauczyć postaci zachowań bazujących na fizyce czy rozwiązywania zagadek. Niestety metoda ta potrzebuje informacji na temat gry, które pozwolą na prawidłową naukę zachowań, co często stanowi problem, ponieważ rzeczywiste gry są ograniczone względem przyspieszenia. Algorytmy głębokiego uczenia ze wzmocnieniem dzielą się na dwa typy - on-policy i off-policy.

Algorytm, który stosuje się do strategii kontroli np. PPO, działa w oparciu o gromadzenie odpowiedniej liczby próbek. Algorytm ten poddawany jest procesowi uczenia, gdzie jego zadaniem jest nauczyć się w jaki sposób polepszyć strategię kontroli na postawie tych próbek. Musi on zwiększyć szanse na wykonanie prawidłowych czynności i zmniejszyć ryzyko wystąpienia działań, które nie są satysfakcjonujące. Algorytmy bazujące na strategii kontroli przechodzą przez proces uczenia się funkcji oceny np. szacowania wartości czy funkcji Q nagród. Zgodnie z informacjami znajdującymi się na stronie internetowej [30], oszacowanie wartości to oczekiwana zdyskontowana suma nagród do końca odcinka, biorąc pod uwagę, że agent jest w określonym stanie, natomiast Q-funkcja stanowi oczekiwaną zdyskontowaną sumę nagród, jeśli dana akcja zostanie podjęta w określonym stanie. Algorytmy działające w myśl on-policy, dokonują szacowania nagród mając na uwadze zaimplementowaną strategię kontroli.

Algorytm PPO do przybliżania funkcji stosuje sieć neuronową. Dzięki temu możliwe jest odtworzenie obserwacji agenta na akcję, która jest najkorzystniejsza z punktu widzenia stanu w jakim znajduje się agent. Metoda ML-Agents PPO uwzględniona została w TensorFlow, gdzie funkcjonuje w osobnym procesie Pythona.

Algorytmy przynależące do grupy off-policy np. SAC, działają w oparciu o określoną dynamikę oraz funkcję nagrody w środowisku. Znajomość zależności pomiędzy daną czynnością w konkretnym stanie, a otrzymaniem nagrody zapewniłaby prosty proces uczenia. Metody opierające się na off-policy nie wymagają sprawdzania skuteczności obecnej polityki, ponieważ algorytmy te poddawane są uczeniu się optymalnej funkcji oceny w każdej strategii kontroli. Proces ten jest trudniejszy do przeprowadzenia niż w przypadku algorytmów on-policy, gdyż rzeczywista funkcja może być bardzo złożona. Jednak z racji tego, że jest to uczenie funkcji globalnej, zgromadzone próbki można wykorzystać do pomocy w nauce. Z tego względu algorytmy, które nie uwzględniają strategii kontroli są wydajniejsze w kwestii próbek niż algorytmy on-policy.

ML-Agents Toolkit wyposażony jest w oryginalny algorytm SAC, przeznaczony do wykonywania zadań lokomocji w trybie ciągłego działania, tak aby obsługiwał wszystkie funkcje takie jak rekurencyjne sieci neuronowe czy rozgałęzione działania dyskretne, co opisano na stronie internetowej [30].

Narzędzie ML-Agents posiada trenera MA-POCA (MultiAgent POsthumous Credit Assignment), szkolącego sieć neuronową. Sieć ta zachowuje się jak „coach” w stosunku do grupy agentów, których może nagradzać zespołowo lub indywidualnie. W przypadku nagród indywidualnych, cała grupa agentów stara się pomóc jednostce w zdobyciu celu jakim jest nagroda. Każdy z agentów może zostać dodany lub usunięty podczas odcinka. W momencie, gdy któryś z agentów zostanie usunięty z powodu unicestwienia pozostałych agentów z grupy lub ich usunięcia z gry, agent ten nadal dostaje informacje o tym, czy jego czynność przyniosła zysk w postaci wygranej zespołu. Trenera MA-POCA można także zastosować we własnej grze, aby wyszkolić zespoły agentów do gry przeciwko sobie.

Konfiguracja Ml-agents

Plik konfiguracyjny configuration.yaml to bazowa sekcja pliku konfiguracyjnego trenera. Plik ten precyzuje funkcje, które mają być wykorzystywane w trakcie uczenia. Znajdują się tam hiperparametry szkolenia dla każdego zachowania, a także ustawienia parametrów środowiska.

Dla wszystkich metod uczenia została zastosowana konfiguracja pokazana w kodzie 6.2.

Listing 6.2: Konfiguracja Ml-agents.

```
trainer_type:   ppo
hyperparameters:
  batch_size:   1024
  buffer_size:  10240
  learning_rate:    0.0003
  beta: 0.005
  epsilon:  0.2
  lambd:    0.95
  num_epoch:    3
  learning_rate_schedule:   linear
network_settings:
  normalize:    False
  hidden_units: 128
  num_layers:   2
  vis_encode_type:  simple
  memory:   None
reward_signals:
  extrinsic:
    gamma:  0.99
    strength:   1.0
init_path:  None
keep_checkpoints:   5
checkpoint_interval:    500000
max_steps:  500000
time_horizon:   64
summary_freq:   50000
threaded:   True
self_play:  None
behavioral_cloning: None
framework:  pytorch
```

Jedną z pierwszych decyzji, jaką trzeba podjąć w związku z przebiegiem treningowym, jest wybór trenera:

- Proximal Policy Optimization (PPO)
- Soft Actor-Critic (SAC)
- MultiAgent POsthumous Credit Assignment (POCA)

Istnieją pewne konfiguracje treningowe, które są wspólne zarówno dla trenerów, jak i inne, które zależą od wyboru trenera. Dla ułatwienia badań zastosowano domyślnego trenera PP0.

Parametry konfiguracji przedstawione 6.2 odpowiadają za:

- trainer_type
  Wybór rodzaju trenera: ppo, sac, lub poca (domyślnie = ppo).
- hyperparameters
  - batch_size
    Stanowi ilość danych w każdej iteracji gradientu prostego, która powinna być kilka rzay mniejsza od buffer_size - w przypadku akcji ciągłych jest rzędu 1000, natomiast dla dyskretnych akcji, liczba ta powinna być rzędu 10.

- buffer_size
  Jest to ilość danych, które należy dostarczyć przed aktualizacją modelu strategii kontroli, zatem są to dane, które zbiera się przed procesem uczenia - buffer_size powinien być wielokrotnie większy od batch_size.
- learning_rate
  To hiperparametr odnoszący się do początkowej szybkości uczenia się dla gradientu prostego, który odpowiada sile każdego kroku aktualizacji spadku gradientu. Wartość ta powinna zostać zmniejszona, gdy trening jest niestabilny, a nagroda nie rośnie konsekwentnie.
- beta
  Opisuje siłę regularyzacji entropii, która sprawia, że strategia kontroli jest bardziej losowa. Dzięki temu agenci mogą prawidłowo eksplorować przestrzeń akcji podczas treningu. Większa wartość beta umożliwia podejmowań większej liczby losowych działań. Entropia, mierzona na podstawie TensorBoard, powinna powoli maleć z jednoczesnym wzrostem nagrody. Jeżeli spadek entropii jest zbyt szybki, hiperparametr beta ulega zwiększeniu. Analogicznie przy zbyt wolnym spadku entropii, beta maleje. W badaniach hiperparametr beta przyjmuje wartość 5.0e-3.
- epsilon
  Hiperparametr epsilon ma wpływ na szybkość procesu ewolucji strategii szkolenia podczas uczenia. Odpowiada on dopuszczalnemu progowi rozbieżności między starą i nową polityką podczas aktualizacji gradientu spadku. Niska wartość epsilon prowadzi do bardziej stabilnych aktualizacji, ale jednocześnie spowalnia proces uczenia. Przyjęto w badaniach wartość hiperparametru epsilon równą 0.2.
- lambd
  lambd jest parametrem regularyzacyjnym wykorzystywanym przy obliczaniu uogólnionego oszacowania przewagi (GAE). Mówi on o tym. w jakim stopniu agent polega na swojej aktualnej ocenie wartości podczas obliczania zaktualizowanej szacunku wartości. Jeśli parametr ten posiada niską wartość, oznacza to, że polega w większym stopniu na bieżącej szacunkowej wartości. W przypadku wysokich wartości lambd, agent opiera się mocniej na rzeczywistych nagrodach otrzymywanych w środowisku. Odpowiednia wartość lambd zapewnia bardziej stabilny proces uczenia. W badaniach lambd stanowi wartość 0.95.
- num_epoch
  Parametr num_epoch opisuje liczbę przebiegów do wykonania przez bufor doświadczenia w trakcie optymalizacji zstępowania gradientu. Bardziej stabilna aktualizacja jest możliwa poprzez pomniejszenie num_epoch, ale zaskutkuje to wolniejszym procesem uczenia. W badaniach wartość tego parametru to 3.
- learning_rate_schedule
  Mówi o tym, w jaki sposób zmianie ulega szybkość uczenia się w czasie. Jeżeli dla PPO tempo procesu uczenia zostanie zmniejszone do max_steps, nauka będzie bardziej stabilna. W badaniach parametr learning_rate_schedule przyjęto wartości linear.

- network_settings:
  - normalize
    Ten parametr decyduje o tym, czy normalizacja jest stosowana do danych wejściowych obserwacji wektora. Normalizacja ta opiera się na średniej kroczącej i wariancji obserwacji wektora. W badaniach normalize przyjmuje wartość false.

  - hidden_units
    Parametr ten to ilość jednostek w ukrytych warstwach sieci neuronowej, odpowiadający liczbie jednostek w każdej w pełni połączonej warstwie sieci neuronowej. Wartość tego parametru dla prostych problemów, gdzie prawidłowe działanie jest prostą kombinacją danych wejściowych obserwacji, powinna być niska, a w przypadku problemów złożonych, powinna być wysoka. W badaniach wartość hidden_units wynosi 128.
  - num_layers
    num_layers określa liczbę ukrytych warstw w sieci neuronowej i odpowiada on liczbie ukrytych warstw po wprowadzeniu obserwacji. Mniejsza liczba warstw przyczyni się do szybszego i wydajniejszego treningu w przypadku prostych problemów, a przy złożonych sterowaniach jest potrzebna większa liczba warstw. W badaniach wartość parametru num_layers to 2.
  - vis_encode_type
    To typ enkodera do kodowania obserwacji wizualnych. Domyślnie wartość tego parametru to simple, który składa się z dwóch warstw konwolucyjnych.
- reward_signals -> extrinsic:
  - strength
    Stanowi współczynnik, przez który mnoży się nagrodę przyznaną przez środowisko. Typowe zasięgi różnią się w zależności od sygnału nagrody. a w badaniach wartość strength to 1.0.
  - gamma
    Gamma to współczynnik rabatu na przyszłe nagrody pochodzące ze środowiska. Parametr ten opisuje jak daleko w przyszłości agent powinien przejmować się możliwymi nagrodami. Jeżeli agent ma działać w teraźniejszości, aby przygotować się na nagrody w odległej przyszłości, wartość gamma powinna być duża. Natomiast, gdy nagrody przydzielane są szybciej, wartość gamma powinna być mniejsza (< 1). W badaniach wartość gamma wynosi 0.99.
- init_path
  Parametr ten inicjalizuje trenera z wcześniej zapisanego modelu. Poprzedni bieg powinien wykorzystywać te same konfiguracje trenera, co bieżący bieg i zostać zapisany z taką samą wersją ML-Agent. W badaniach init_path dla pierwszego etapu uczenia nie przyjmował inicjującego trenera, z kolej w kolejnych etapach przyjmował trenerów z poprzedniego etapu.
- keep_checkpoints
  Jest to maksymalna liczba punktów kontrolnych modelu do zachowania. Po liczbie kroków określonej przez opcję checkpoint_interval, zapisywane są punkty kontrolne. Jeżeli osiągnie się maksymalną liczbę punktów kontrolnych, najstarszy z nich jest usuwany podczas zapisywania nowego punktu kontrolnego. W badaniach wartość keep_checkpoints to 5. Aczkolwiek nie jest wykorzystywany ponieważ należałoby w scenie zdefiniować jakieś obiekty jako checkpoints co nie jest potrzebne dla robotów sumo, ale mogłoby być przydatne dla Line Follower.
- checkpoint_interval
  Parametr ten stanowi liczbę doświadczeń zebranych między każdym punktem kontrolnym przez trenera. Zanim stare punkty kontrolne zostaną usunięte, maksymalna liczba keep_checkpoints punktów kontrolnych jest zapisywana. Pliki .onnx zapisywane są w results/folderze. W badaniach checkpoint_interval przyjmuje wartość 500000.
- max_steps
  To całkowita liczba kroków, jakie należy wykonać w środowisku przed zakończeniem procesu uczenia. W badaniach max_steps ustawia się na 500000.

- time_horizon
  Mówi o tym, ile kroków doświadczenia należy zebrać dla każdego agenta przed dodaniem go do bufora doświadczenia. Oszacowanie wartości jest używane do przewidywania ogólnej oczekiwanej nagrody na podstawie obecnego stanu agenta, w przypadku gdy ten limit zostanie osiągnięty przed końcem odcinka. Parametr ten zmienia się pomiędzy mniej obciążonym, ale wyższym oszacowaniem wariancji (długi horyzont czasowy) i bardziej obciążonym, ale mniej zróżnicowanym oszacowaniem (krótki horyzont czasowy). Jeżeli w odcinku często pojawiają się nagrody lub odcinki są zbyt duże, mniejsza wartość time_horizon może okazać się lepsza. Ta liczba powinna być wystarczająco duża, aby uchwycić wszystkie ważne zachowania w sekwencji działań agenta. W badaniach time_horizon osiąga wartość 64.
- summary_freq
  Jest to liczba doświadczeń, które należy zebrać przed wygenerowaniem i wyświetleniem statystyk treningu. Opisuje ona ziarnistość wykresów w Tensorboard. W badaniach summary_freq wynosi 50000.
- threaded
  Zezwala on środowiskom na działanie podczas aktualizacji modelu. Skutkuje to przyspieszeniem treningu, szczególnie w przypadku korzystania z SAC. W badaniach threaded ma wartość false.

6.3. Metody nagradzania

6.3.1. Logika wielowartościowa wykorzystana w uczeniu ze wzmocnieniem

Wartości wejściowe dla tej metody zostały przetworzone przez poniższy wzór:

$$f(x) = ROUND(\frac{3 \cdot x}{maxRange}),$$

gdzie ROUND to funkcja zaokrąglająca do liczb całkowitych, maxRange to maksymalny zasięg czujników odległościowych(w tym przypadku 0.8m). Zastosowanie tego wzoru miało na celu przydzielenie odległości do odpowiedniej wartości lingwistycznej:

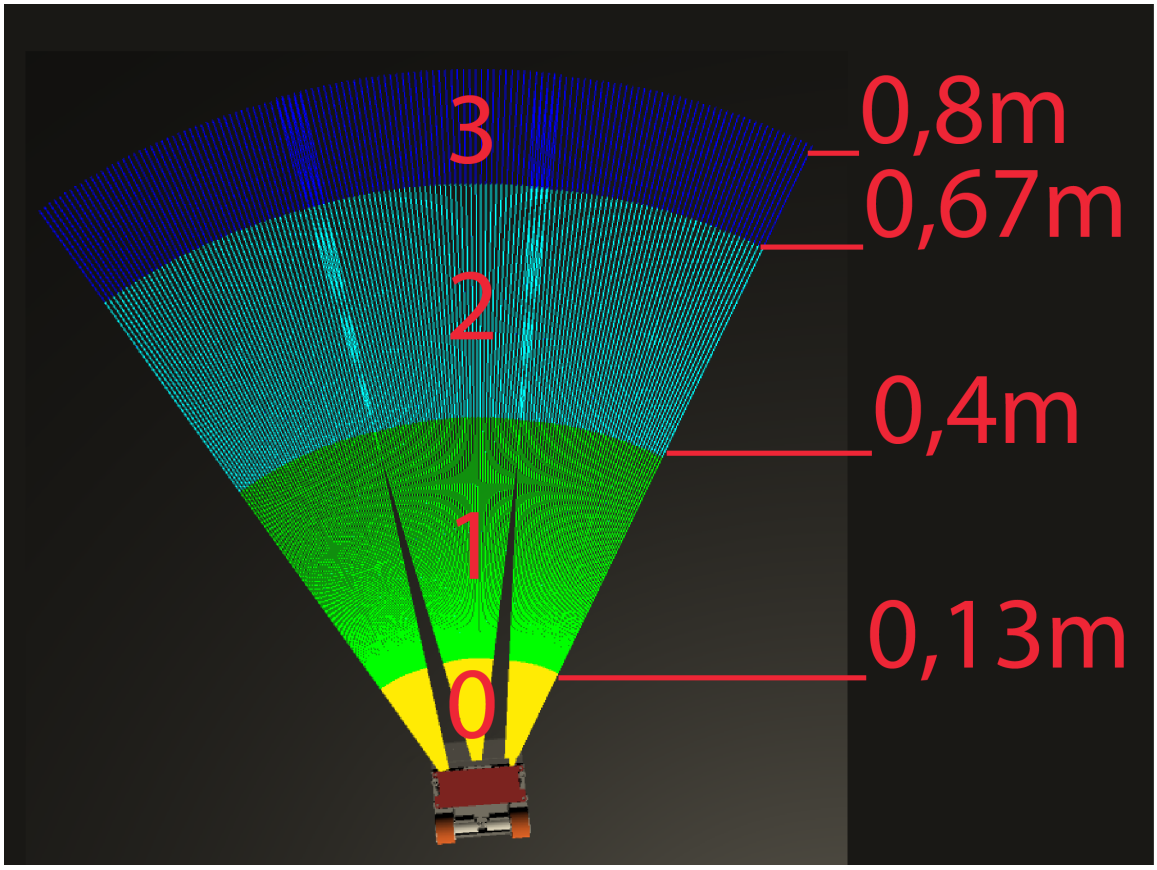
- "0"$\in [0; 0,13)$m
- "1"$\in [0,13; 0,4)$m
- "2"$\in [0,4; 0,67)$m
- "3"$\in [0,67; \infty)$m

Podział został przedstawiony graficznie na obrazie 6.3.

Przydzielenie do odpowiedniej wartości lingwistycznej dla każdego z czujników odbywa się przez funkcję _classicalLogic.Fuzzy(sensors[i].distance) widoczną w kodzie 6.3.

Listing 6.3: Funkcja gromadząca dane z obserwacji środowiska przez Agenta dla metody z logiką wielowarościową: `CollectObservations`.

```
public override void CollectObservations(VectorSensor sensor)
{
    base.CollectObservations(sensor);
    for (int i = 0; i < sensors.Length; i++)
    {
        arrayFuzzy[i] = _classicalLogic.Fuzzy(sensors[i].distance);
        sensor.AddObservation(arrayFuzzy[i]);
    }
}
```



Rys. 6.3: Zakresy dla wartości lingwistycznej

Na podstawie uzyskanych wartość i reguł przedstawionych w 6.4, określano nagrodę dla agenta. Reguł jest wystarczająco dużo, aby robot był w stanie nauczyć się odpowiednio wyśrodkowywać przeciwnika, a następnie wypchnąć go z ringu.

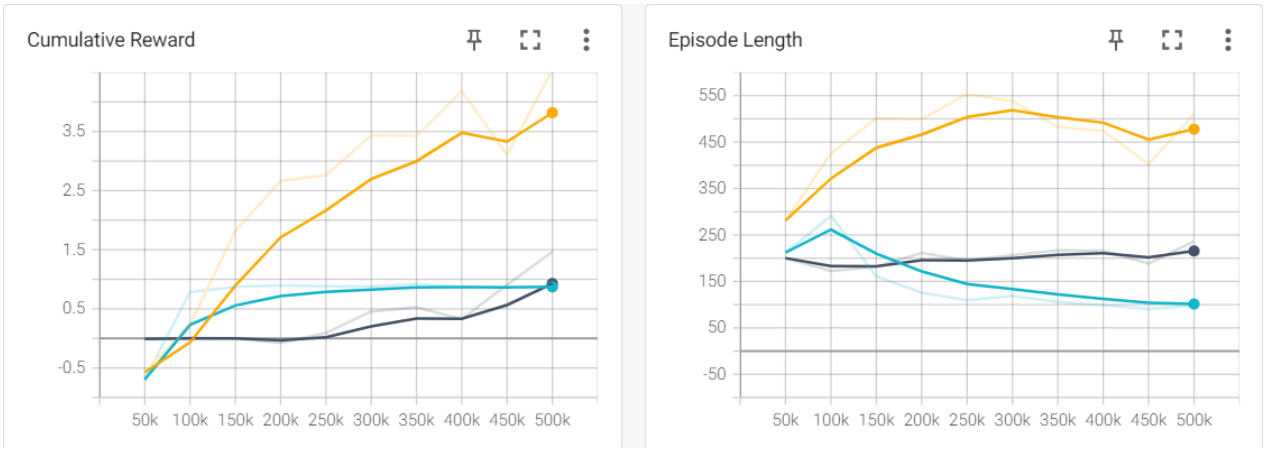
Listing 6.4: Reguły uczenia.

```
if (arrayFuzzy[2] < 3 && arrayFuzzy[3] == 3 && arrayFuzzy[4] == 3 &&
motors[1].powerMotor > 20f)
{
    SetReward(0.1f);
}
else if (arrayFuzzy[2] == 3 && arrayFuzzy[3] == 3 && arrayFuzzy[4] < 3 &&
motors[0].powerMotor > 20f)
{
    SetReward(0.1f);
}
else if (arrayFuzzy[2] == 3 && arrayFuzzy[3] < 3 && arrayFuzzy[4] == 3 &&
motors[0].powerMotor > 20f && motors[1].powerMotor > 20f)
{
    SetReward(0.1f);
}
```

Dzięki narzędziu TesnorBoard opisanym w rozdziale 3.1.1 możliwe jest zobrazowanie przebiegu procesu uczenia. Wykresy przedstawiają wszystkie trzy etapy:

- niebieski pierwszy etap
- żółty drugi etap
- szary trzeci etap

Na wykresie przedstawiającym kumulację nagród (Cumulative Reward) 6.4 widać, że wszystkie etapy działały prawidłowo, ponieważ rosła ilość otrzymywanej wartości nagrody. W pierwszym etapie uczenia można było zakończyć uczenie na około 350000 kroku ze względu na znikomy wzrost otrzymywanych nagród. Średnia długość rundy (Episode Length) maleje przy pierwszym etapie, ponieważ agent coraz szybciej znajduję statycznego przeciwnika, wypychając go z ringu i kończąc tym samym rundę. W przypadku drugiego etapu występuje ten sam przeciwnik, co powoduje ciągłe zrównoważone starcia, przedłużające rundę.



Rys. 6.4: Kumulacja nagród i średnia długość rundy dla uczenia z logiką wielowartościową

Średnia wartość funkcji straty polityki (Policy Loss) przedstawionej na wykresie 6.5 wchodzi w korelacje z tym, jak bardzo zmienia się polityka (proces podejmowania decyzji). Przy udanej sesji treningowej, wartość ta powinna ulec zmniejszeniu. Wykresy mają charakter oscylacyjny podczas treningu, a ich wartości są mniejsze niż 1,0.

Wykres obok to średnia utrata aktualizacji funkcji wartości (Value Loss). Pokazuje, jak dobrze model jest w stanie przewidzieć wartość każdego stanu. Wartość rośnie, gdy agent się uczy. Oznacza to również, że nagroda wzrasta. Stabilizacja wartości następuje, gdy nagroda utrzymuje się na tym samym poziomie.

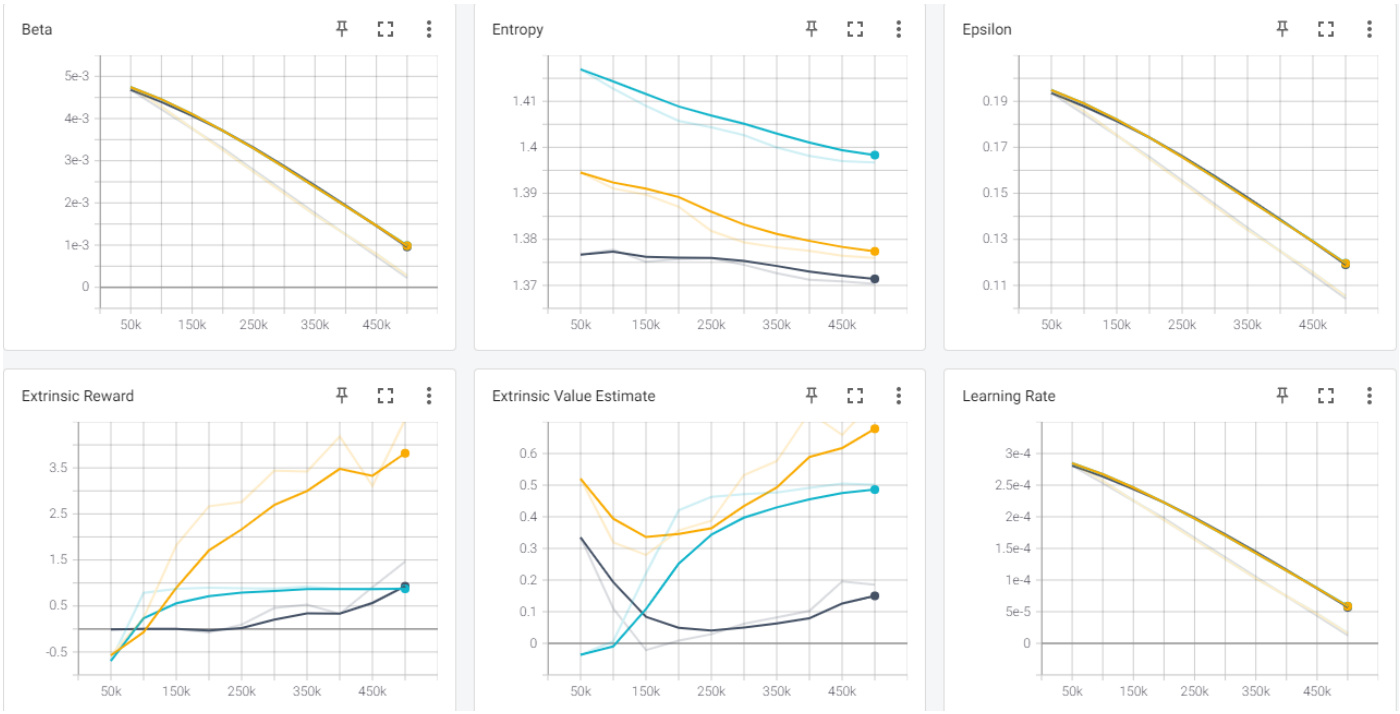
Rys. 6.5: Średnia wielkość funkcji straty polityki i utraty wartości w uczeniu z logiką wielowartościową

Wykres 6.6 entropii (Entropy) przedstawia jak losowe są decyzje modelu. Powinien powoli spadać podczas udanego procesu treningowego, co widać na wykresie w przypadku wszystkich trzech etapów. Można zauważyć także, że w kolejnych etapach wartość entropii była coraz mniejsza, ze względu na kontynuację etapów. Wartość β (Beta) pozwala na regulację entropii,

z kolei ϵ (Epsilon) umożliwia kontrolowanie polityki uczenia. Obie te wartość maleją z kolejnymi krokami.

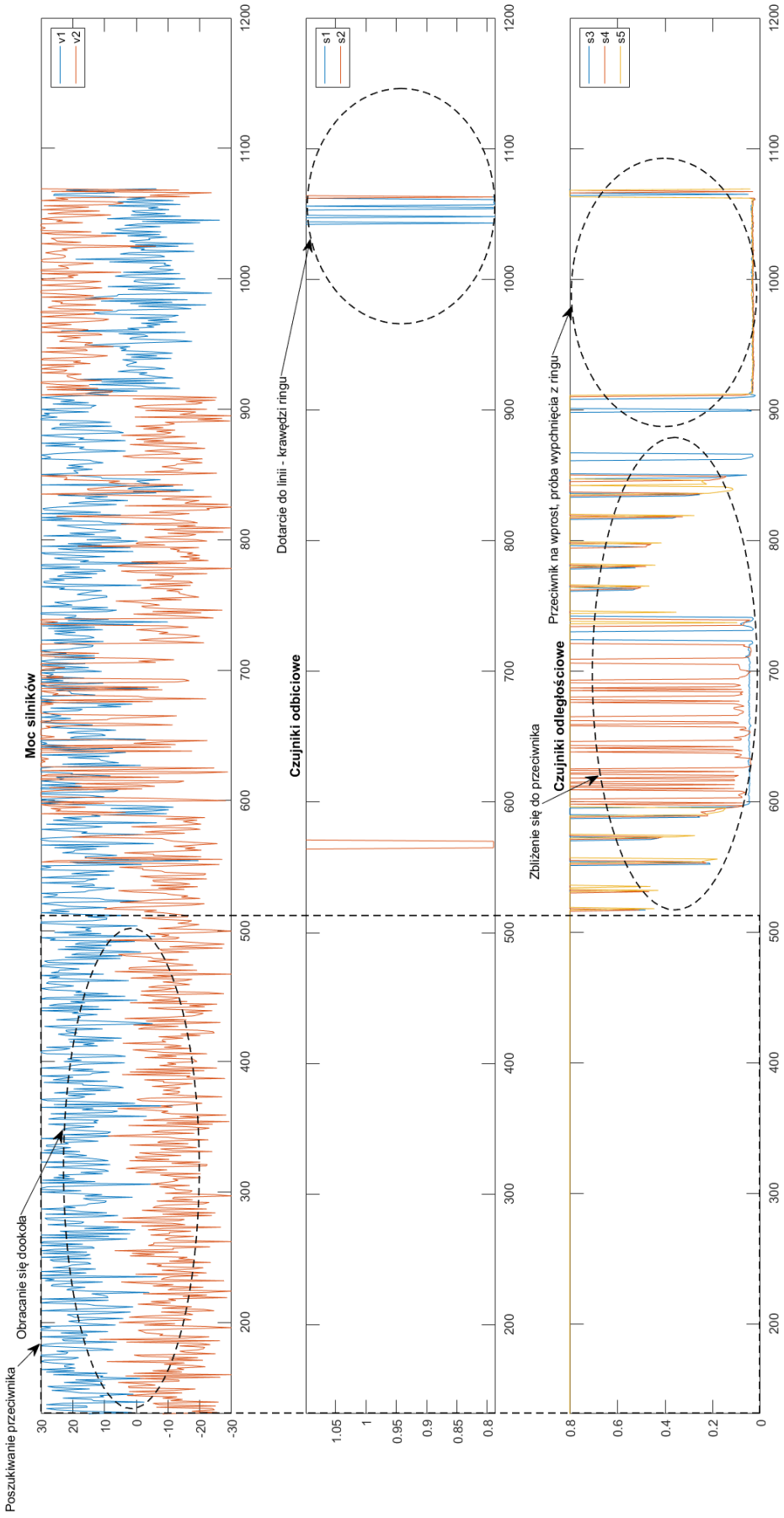
Średnia oszacowana wartość (Extrinsic Value Estimate) powinna wzrosnąć podczas udanej sesji treningowej. W przypadku drugiego i trzeciego etapu na początku widać spadek, a następnie wzrost wartości. Jest to uwarunkowane tym, że robot sumo walczył z przeciwnikiem, który mógł go zepchnąć z ringu i tym samym otrzymać karę. Te wartości powinny rosnąć wraz ze wzrostem skumulowanej nagrody. Odpowiadają one wysokości przyszłej nagrody, którą agent przewiduje, że otrzyma w danym momencie.

Wykres szybkości uczenia się (Learning Rate) określa jak duży krok wykonuje algorytm uczący podczas wyszukiwania optymalnej polityki. Z czasem maleje według rozkładu liniowego.



Rys. 6.6: Parametry polityki w uczeniu z logiką wielowartościową

Robot po przeprowadzeniu uczenia powyższą metodą był zdolny do wykrycia i zepchnięcia przeciwnika z ringu. Efekty można zobaczyć na wykresie 6.7, gdzie za pomocą mocy silników można określić czy robot jechał prosty czy też obracał się. Czujniki odbiciowe pokazują, kiedy robot stykał się z krawędzią ringu, a czujniki odległościowe wskazują z której strony i jak daleko znajduje się przeciwnik.



Rys. 6.7: Analiza danych z silników i czujników

6.3.2. Pojedyncza funkcja nagrody wykorzystana w uczeniu ze wzmocnieniem

Metoda ta opiera się na wyznaczaniu nagrody przy pomocy poniżej funkcji

$$f(v1, v2, s1, s2, s3, s4, s5) = \frac{\mid v1 \mid + \mid v2 \mid}{(s1 + s2 + s3 + s4 \cdot 2 + s5) \cdot 30},$$

gdzie v1 i v2 to moc silników w zakresie [0-30], s1 i s2 to wartość Boolowska (0,1) z czujników odbiciowych, s3, s4, s5 to wartości odległości od przeciwnika w polu widzenia w zakresie [0-0.8] z czujników odbiciowych. Jeśli nie ma przeciwnika zwracana jest wartość 1. Parametry te wyznaczane były w kodzie 6.5, w którym należało odpowiednio wyznaczyć wartości z czujników.

Listing 6.5: Funkcja gromadząca dane z obserwacji środowiska przez Agenta dla metody z pojedynczą funkcją nagradzania: `CollectObservations`

```
public override void CollectObservations(VectorSensor sensor)
{
    base.CollectObservations(sensor);
    for (int i = 0; i < sensors.Length; i++)
        if (sensors[i].GetType() == typeof(ReflectionSensor))
            if (sensors[i].distance < _maxRangeReflection)
                arrayFuzzy[i] = 1;
            else
                arrayFuzzy[i] = 0;
        else
            if (sensors[i].distance < _maxRange)
                arrayFuzzy[i] = sensors[i].distance / _maxRange;
            else
                arrayFuzzy[i] = 1;

        sensor.AddObservation(arrayFuzzy[i]);
}
```

Funkcja nagradzania przedstawiona w fragmencie kodu 6.6 powoduję, że agent jest nagradzany najbardziej, gdy używa pełnej mocy silników, przeciwnik styka się z przodem robota oraz nie znajduje się w obramowaniu ringu (przy krawędzi).

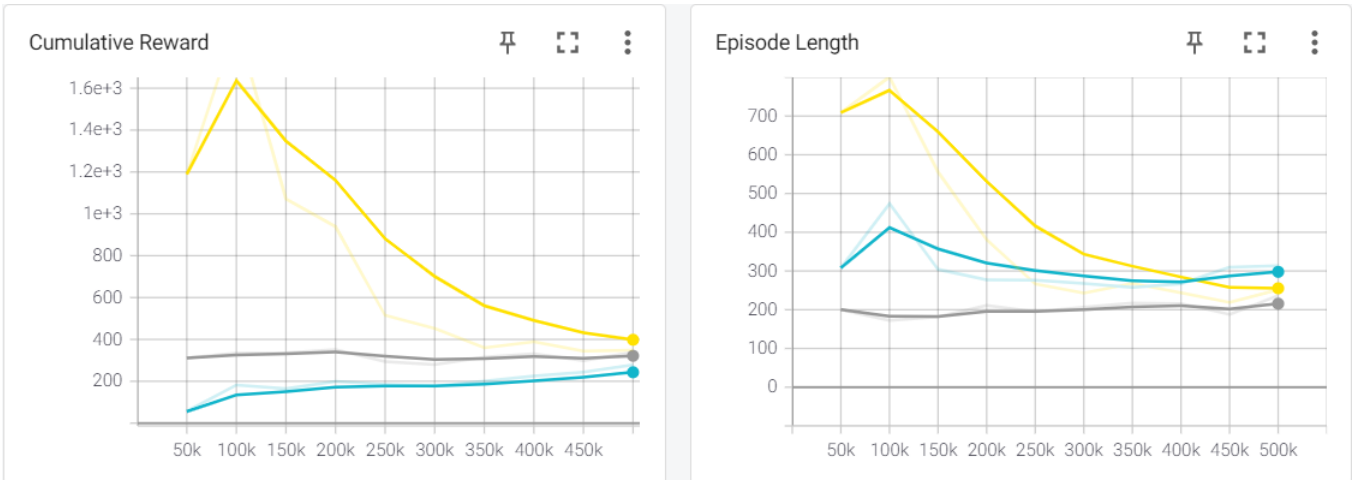
Listing 6.6: Funkcja nagradzania.

```
powerMotors = motors.Select(m => m.powerMotor).ToArray();
powerMotorSum = powerMotors.Select(m => Math.Abs(m)).Sum() / 30f;
reward = powerMotorSum / (arrayFuzzy.Sum() + arrayFuzzy[3]);
SetReward(reward);
```

TesnorBoard opisany w rozdziale 3.1.1, pozwala przejrzeć przebieg procesu uczenia. Wykresy przedstawiają wszystkie trzy etapy:

- niebieski pierwszy etap
- żółty drugi etap
- szary trzeci etap

Wykres obrazujący kumulację nagród (Cumulative Reward) 6.8 pokazuje, że drugi etap nie dział prawidłowo, ponieważ wartość nagrody zaczęła maleć w okolicy 100 tysięcznego kroku. Pierwszy i trzeci etap uczenia prezentuje się poprawnie. Średnia długość rundy (Episode Length) w przypadku pierwszego i trzeciego etapu właściwie utrzymuje się na jednym poziomie. Porównując to do wykresu 6.4 można stwierdzić, że rundy trwały dłużej, co mogłoby świadczyć o gorszym działaniu, aczkolwiek w rozdziale następnym 6.3.3 okaże się to niesłusznym porównaniem. W drugim etapie roboty walcząc ze sobą, uczyły się znajdywać ruchomy cel, dlatego też rundy na początku trwały znacznie dużej.



Rys. 6.8: Kumulacja nagród i średnia długość rundy dla uczenia z pojedynczą funkcją nagrody

Średnia wartość funkcji straty polityk (Policy Loss) przedstawionej na wykresie 6.9 charakteryzuje się oscylacjami podczas treningu, co jest pożądanym zjawiskiem. Wykres ten jest podobny do wykresu 6.9 związanego z uczeniem z logiką wielowartościową.

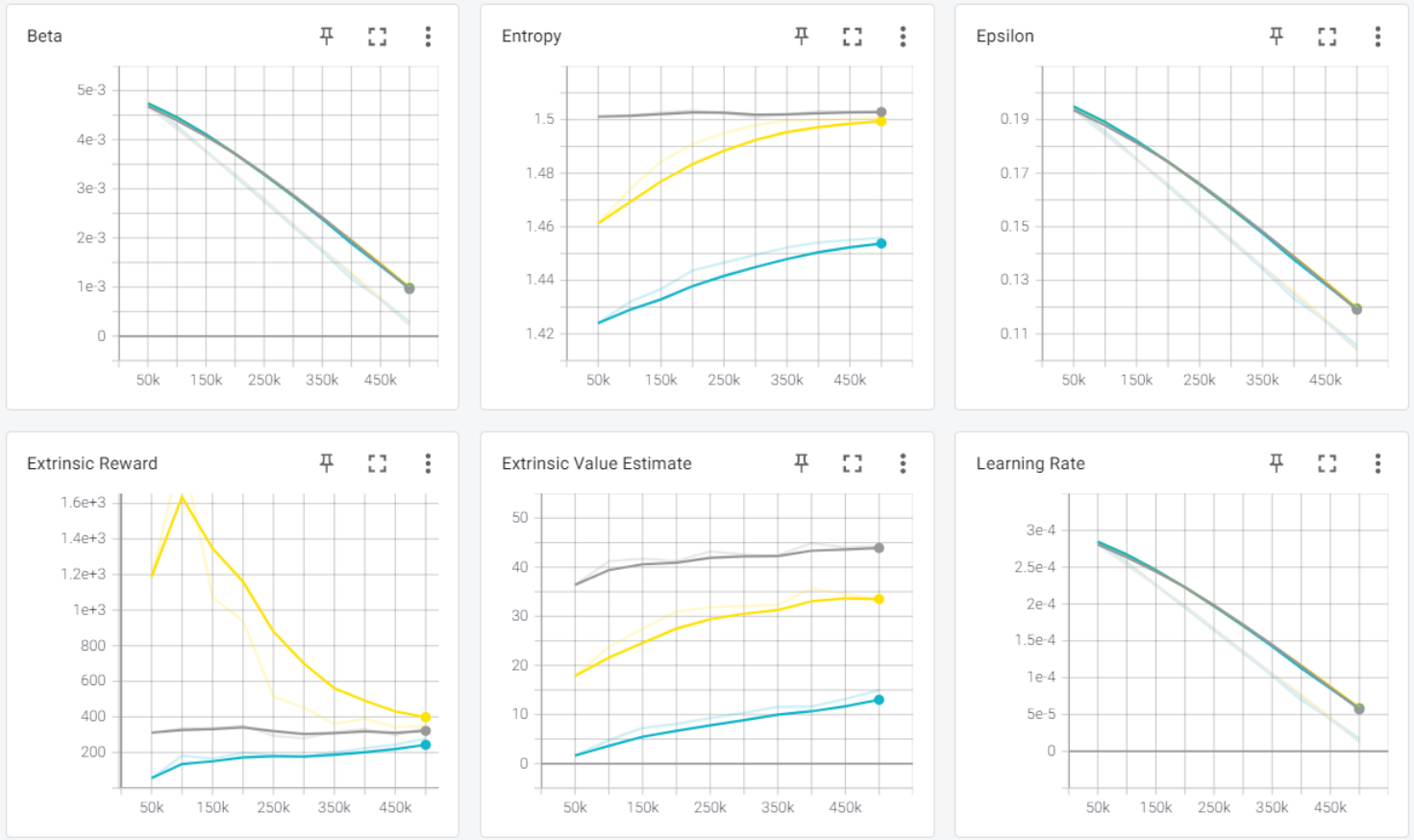
Wykres średniej utraty aktualizacji funkcji wartości (Value Loss) rośnie w przypadku pierwszego i drugiego etapu, z kolei maleje w przypadku etapu trzeciego.

Rys. 6.9: Średnia wielkość funkcji straty polityki i utraty wartości w uczeniu z pojedynczą funkcją nagrody

Wykres 6.10 entropii (Entropy) wskazuje na nieudany proces treningowy, ponieważ wartość rośnie. Wartość β (Beta) i ϵ (Epsilon) maleją z kolejnymi krokami prawidłowo.

Średnia oszacowana wartość (Extrinsic Value Estimate) prawidłowo wzrasta, co świadczy o udanej sesji treningowej.

Wykres szybkości uczenia się (Learning Rate) maleje według rozkładu liniowego dla wszystkich etapów.



Rys. 6.10: Parametry polityki w uczeniu z pojedynczą funkcją nagrody

Po zakończonych treningach robot pomimo niewłaściwych wykresów z przebiegu uczenia, poprawnie poruszał się po ringu w poszukiwaniu przeciwnika, a gdy go znalazł starał się wypchnąć go z ringu.

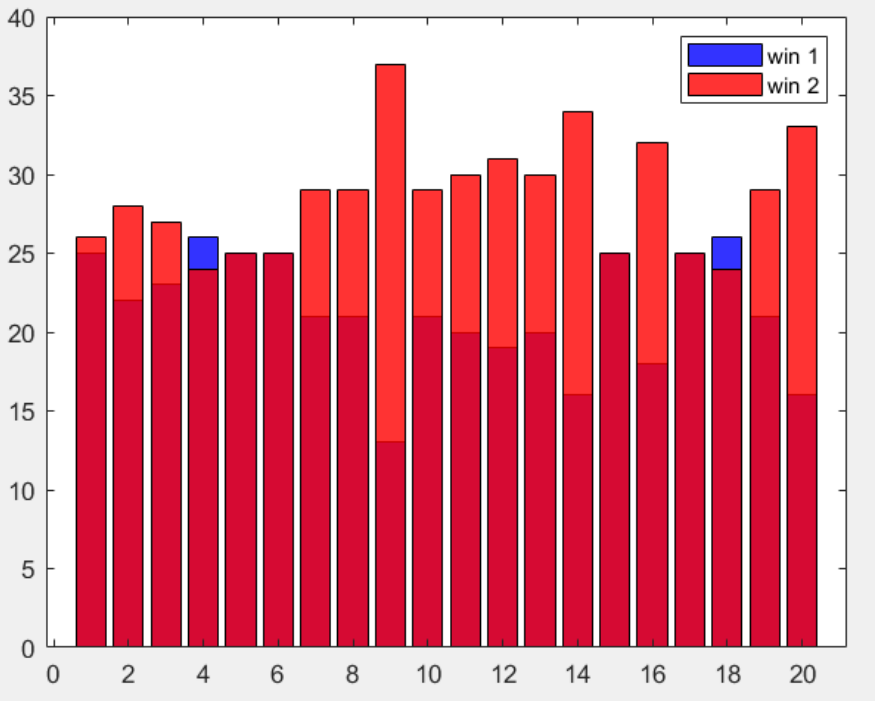
6.3.3. Porównanie logiki wielowartościowej z funkcją nagradzania

Po drugim etapie (walki robotów z samym sobą), we wcześniej przedstawionych sposobach uczenia, przeprowadzono test walki między robotami wyuczonymi metodą logiki wielowartościowej, a funkcją nagradzania. Walki rozegrały się, aż liczba wszystkich wygranych wynosiła 1000, co znaczy, że pomijano przypadki remisów. Rezultaty wyglądają następująco:

- Niebieski robot uczony z logiką wielowartościową wygrał: 428
- Pomarańczowy robot uczony z funkcją nagradzania wygrał: 572

Widać znaczącą przewagę dla robota uczonego z pojedynczą funkcją nagradzania, co może świadczyć o lepszej skuteczności uczenia.

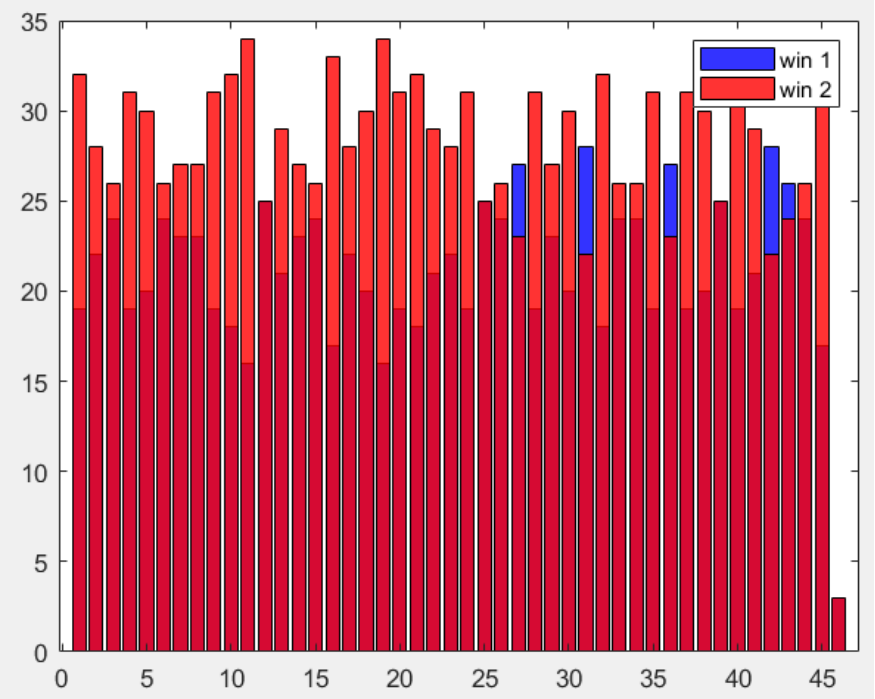
Przyjrzano się bliżej rezultatom, dzieląc ilość przeprowadzonych walk na serie po 50 wygranych. Efekty przedstawiono na wykresie 6.11, gdzie każdy słupek to 50 rund podzielonych na dwa kolory. Niebieskim kolorem zaznaczono robota posługującego się metodą z logiką wielowartościową, a pomarańczowym metodą z funkcją nagradzania. Jak widać niebieskiemu udało się dwa razy wygrać serię i cztery razy zremisować - słupki całe czerwone. Oznacza to, że robot oznaczony pomarańczowym kolorem nie jest niepokonany.



Rys. 6.11: Wykres serii walk (po 50) dla uczenia z logiką wielowartościową i funkcją nagradzania - po drugim etapie

Wykres 6.12 przedstawia wyniki podczas uczenia w etapie trzecim. Widać, że robot uczony przy pomocy logiki wielowartościowej zaczął wygrywać w późniejszym procesie uczenia. Podsumowując wygrane wyglądają następująco:

- Niebieski robot uczony z logiką wielowartościową wygrał: 974
- Pomarańczowy robot uczony z funkcją nagradzania wygrał: 1284



Rys. 6.12: Wykres serii walk (po 50) dla uczenia z logiką wielowartościową i funkcją nagradzania - podczas etapu trzeciego

6.4. LSTM dla pojedynczej funkcji nagradzania wykorzystanej w uczeniu ze wzmocnieniem

Długoterminowa pamięć krótkoterminowa (LSTM), opisana w artykule [21], to rodzaj rekurencyjnej sieci neuronowej (RRN - recurrent neural network), która potrafi uczyć się zależności długoterminowych. Sieć ta jest w stanie zapamiętywać informacje przez długi okres czasu, ponieważ stanowi to jej domyślne zachowanie. W przypadku sieci RNN, powtarzające się moduły sieci cechuje prosta budowa.

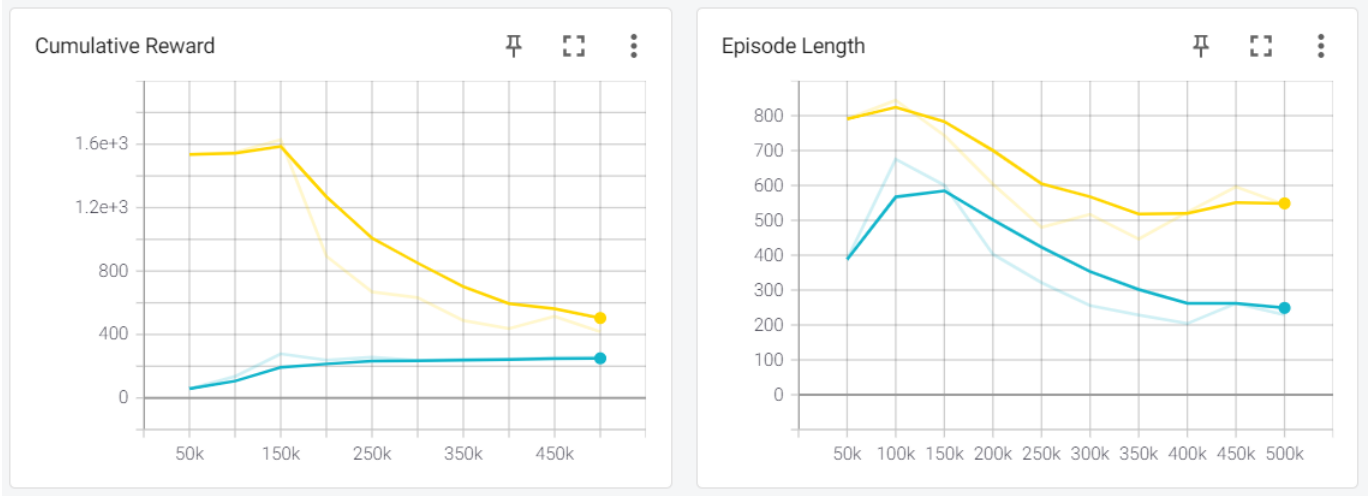
Bazą LSTM jest stan komórki, który działa na zasadzie przenośnika taśmowego. Podczas drogi przez łańcuch wchodzi on w delikatne interakcje liniowe, ale zazwyczaj pokonuje tę drogę bez większych zmian. Sieć LSTM może dodać bądź usunąć informacje ze stanu komórki, kontrolowanej poprzez bramki. W skład bramek, zgodnie z informacjami zawartymi w artykule [21], wchodzą warstwa sigmoidalna sieci neuronowej oraz operacja mnożenia punktowego. Za sprawą warstwy sigmoidalnej otrzymywane są dane na temat części składnika, która ma zostać przepuszczona przez bramki. Liczba zero oznacza komunikat zabraniający przepuszczenia przez bramki, natomiast wartość jeden pozwala na przepuszczenie wszystkiego.

Tworząc nowego agenta przy wykorzystaniu LSTM pominięto etap drugi. W etapie trzecim przeprowadzono trening z robotem wytrenowanym poprzez metodę z pojedynczą funkcją nagradzania, opisaną w sekcji 6.3.2.

Z pomocą TesnorBoard, opisanego w rozdziale 3.1.1, uzyskano wykres przedstawiający dwa etapy uczenia:

- niebieski pierwszy etap
- żółty trzeci etap

Wykres kumulacji nagród (Cumulative Reward) 6.13 pokazuje, że pierwszy etap uczenia przebiegał prawidłowo. W trzecim etapie widać znaczący spadek nagradzania, spowodowany tym, że przeciwnik spychał trenowanego agenta z ringu. Średnia długość rundy (Episode Length) dla pierwszego i trzeciego etapu malała.



Rys. 6.13: Kumulacja nagród i średnia długość rundy - LSTM

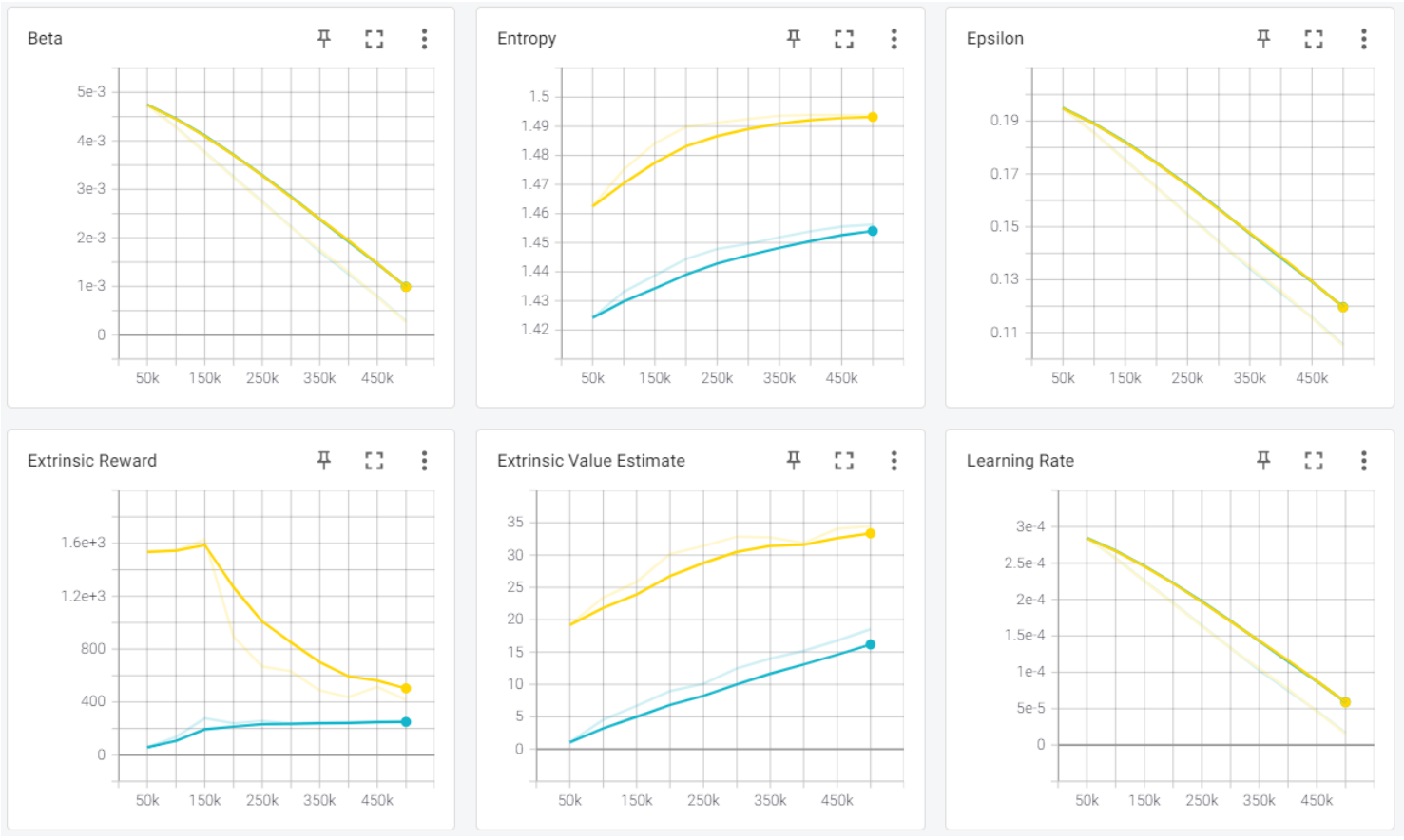
Średnia wartości funkcji straty polityki (Policy Loss) przedstawiona na wykresie 6.14, oscyluje, wskazując na prawidłowe działanie.

Uśredniona utrata aktualizacji funkcji wartości (Value Loss) dla trzeciego etapu maleje, ponieważ nagradzanie zmniejsza się. Sytuacja ta jest przeciwna do etapu pierwszego, gdzie agent dostawał coraz większą nagrodę.

Rys. 6.14: Średnia wielkość funkcji straty polityki i utraty wartości w uczeniu - LSTM

Entropia (Entropy) na wykresie 6.15 rośnie w etapie pierwszym i trzecim, co świadczy o nieudanym procesie uczenia.

Średnia oszacowana wartość (Extrinsic Value Estimate) wzrasta dla obu etapów uczenia.

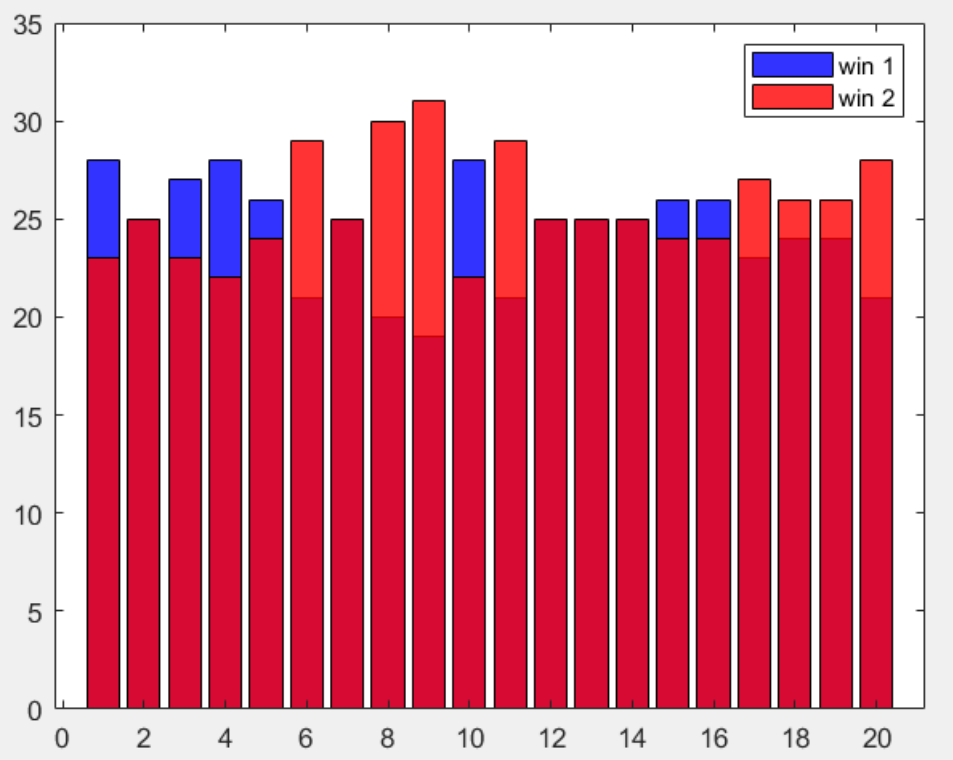


Rys. 6.15: Parametry polityki w uczeniu - LSTM

Tak wyuczony robot podjął próbę walki z robotem poddanym procesowi uczenia w sekcji 6.3.2. Wyniki prezentują się następująco:

- Niebieski robot uczony bez LSTM wygrał: 487
- Czerwony robot uczony z LSTM wygrał: 513

Na wykresie 6.16 przedstawiono przebieg sparingów w seriach po 50 walk dla powyższych wyników. Widać, że walka była wyrównana, ale ostatecznie robot czerwony wygrał z przewagą 13 rund.



Rys. 6.16: Wykres serii walk (po 50) dla uczenia z funkcją nagradzania - LSTM

6.5. Zmiana parametrów czujnika odległościowego

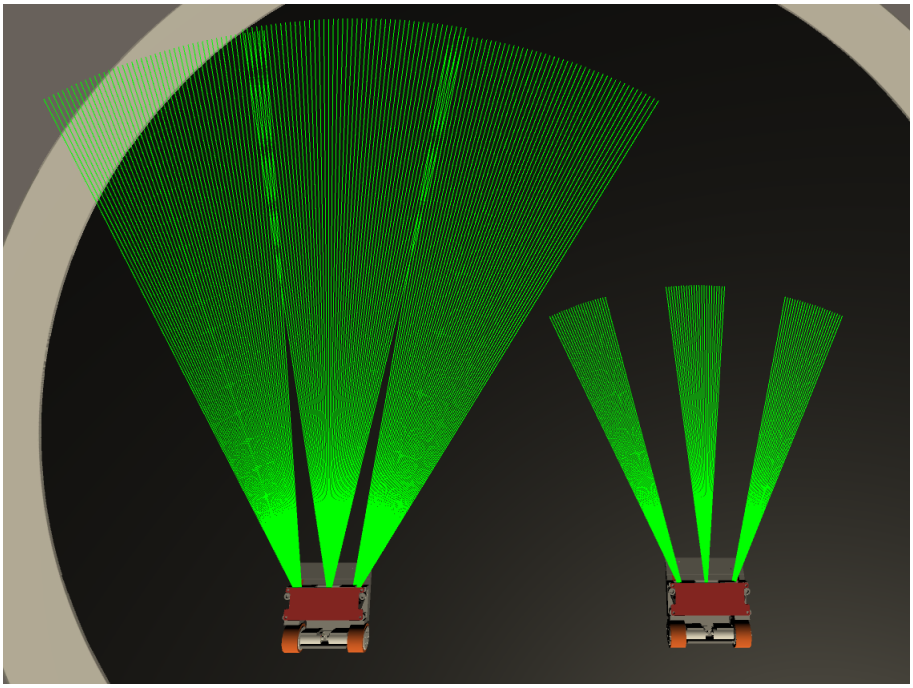
6.5.1. Pełne uczenie dla robota ze zmniejszonym polem widzenia

Dalsze badania zostały przeprowadzone na metodzie uczenia ze wzmocnieniem z pojedynczą funkcją nagradzania, opisaną w sekcji 6.3.2, ze względu na lepsze rezultaty przedstawione w sekcji 6.3.3. Zmieniono parametry czujnika odległościowego na:

- maksymalny zasięg 40 cm
- pole widzenia 6°
- czas odświeżania 8ms

zgodnie ze specyfikacją czujnika Sharp GP2Y0D340K, dostępną [12]. Porównanie pola widzenia czujników przedstawiono na zdjęciu 6.17.

Przeprowadzono dwa etapy uczenia opisane w poprzednich rozdziałach. Efekty treningu agenta nie były zadowalające, ponieważ robot jedynie kręcił się w kółko. Nie reagował on również na zbliżającego się przeciwnika. Z tego powodu zrezygnowano z przeprowadzenia trzeciego etapu uczenia.

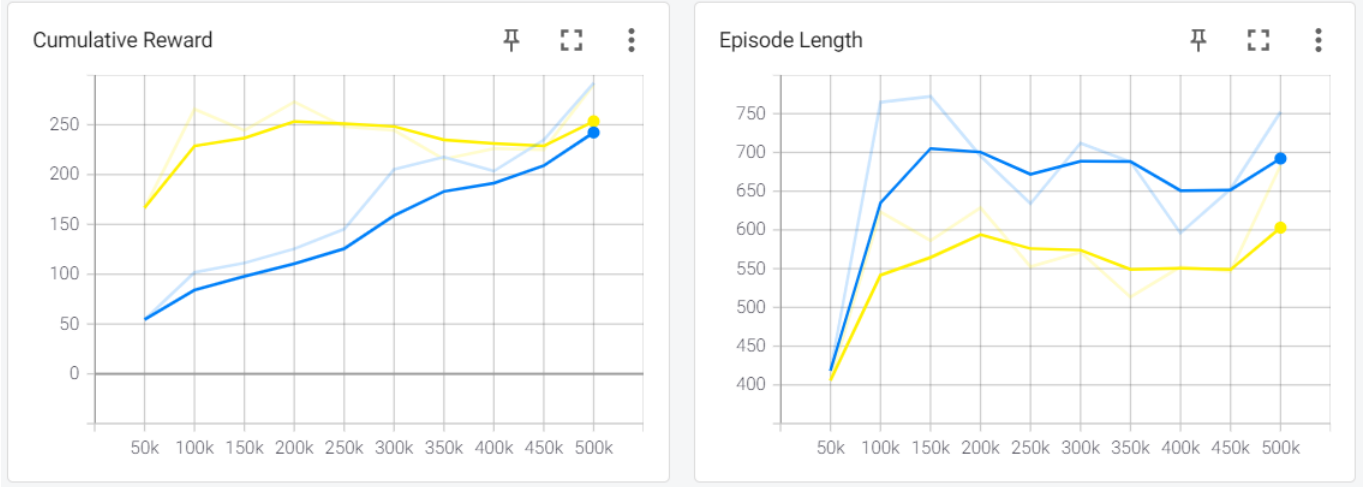


Rys. 6.17: Porównanie pola widzenia czujników odległościowych przy zmniejszonym polu widzenia

Wykres przedstawiający poniższe dwa etapy uczenia, uzyskano dzięki TesnorBoard, przedstawiony w rozdziale 3.1.1:

- niebieski pierwszy etap
- żółty drugi etap

Wykres obrazujący kumulację nagród (Cumulative Reward) 6.18 pokazuje, że pierwszy i drugi etap uczenia prezentują się poprawnie. Średnia długość rundy (Episode Length) w przypadku pierwszego etapu jest większa od drugiego etapu ze względu na znajdujące się dwa roboty na arenie - gdy jeden wypadnie runda jest przerywana.



Rys. 6.18: Kumulacja nagród i średnia długość rundy przy zmniejszonym polu widzenia

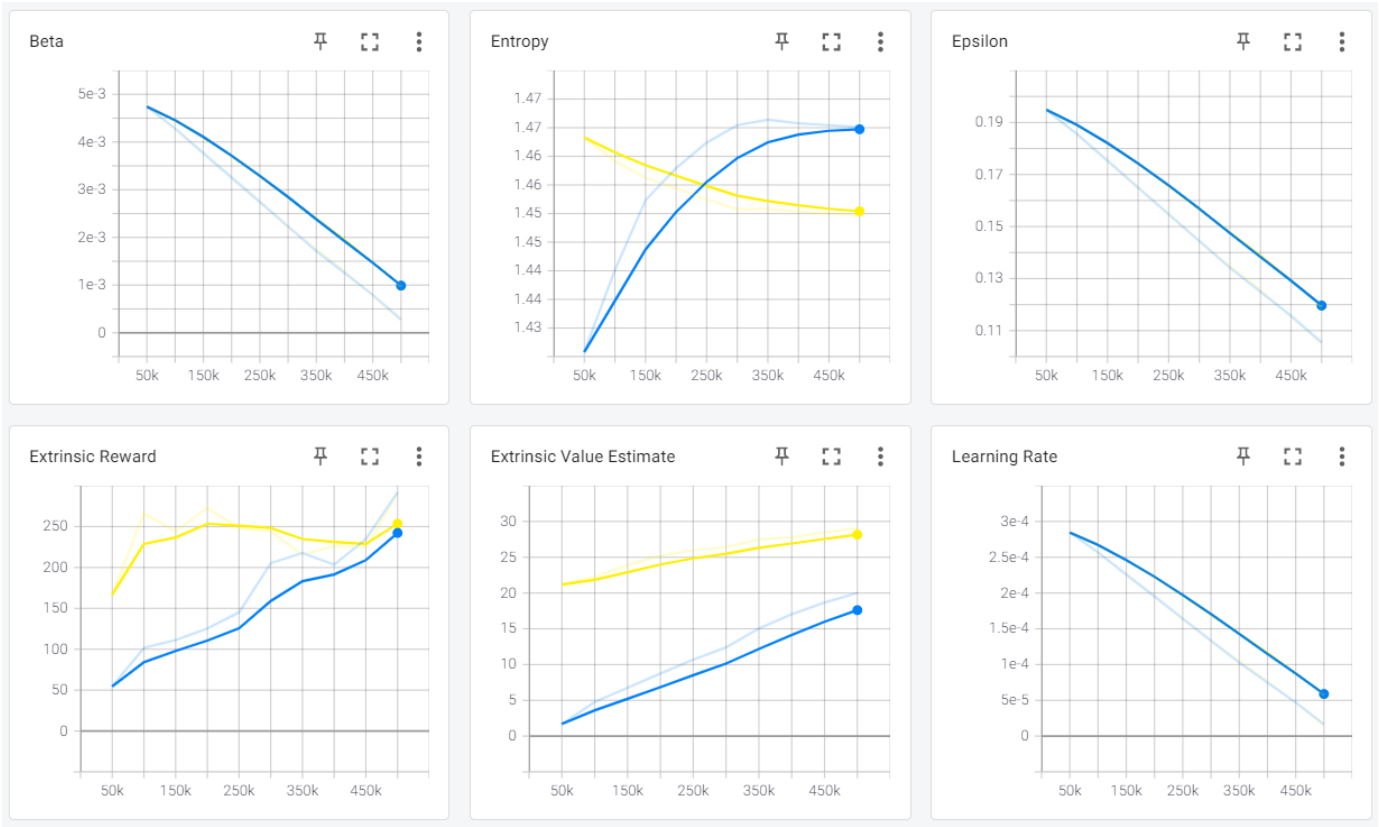
Wykres średniej wartości funkcji straty polityki (Policy Loss) przedstawiony jest na 6.19. Zauważyć można, że proces podejmowania decyzji oscyluje i maleje w obu etapach, świadcząc o udanej sesji treningowej.

Średnia utrata aktualizacji funkcji wartości (Value Loss), pokazuje dla etapu pierwszego, że wartości rosną, zatem agent uczy się. Przeciwnie do etapu drugiego, w którym agent nie osiągnął lepszych rezultatów.

Rys. 6.19: Średnia wielkość funkcji straty polityki i utraty wartości w uczeniu przy zmniejszonym polu widzenia

Entropia (Entropy) na wykresie 6.20 rośnie w etapie pierwszym, co świadczy o nieudanym procesie uczenia. Z kolei w przypadku etapu drugiego widać, że wartość entropii maleje, tym samym informując o udanym treningu.

Średnia oszacowana wartość (Extrinsic Value Estimate) rośnie dla obu etapów uczenia.

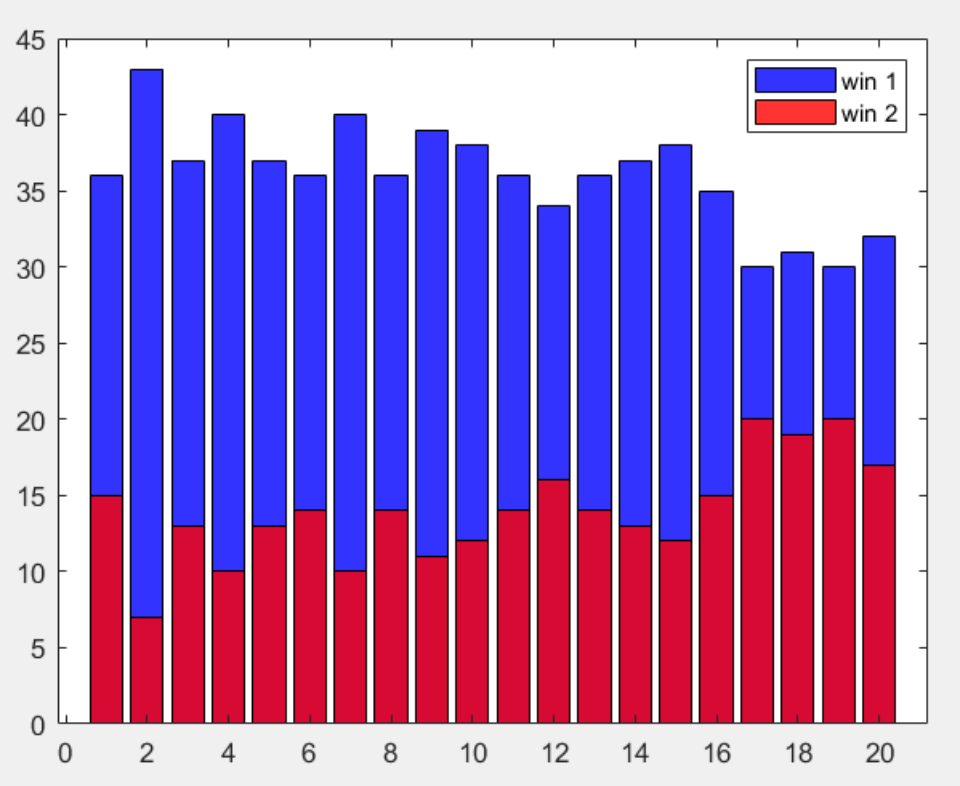


Rys. 6.20: Parametry polityki w uczeniu przy zmniejszonym polu widzenia

Wyuczony w ten sposób robot podjął próbę walki z robotem poddanym procesowi uczenia w sekcji 6.3.2. Wyniki prezentują się następująco:

- Niebieski robot uczony z funkcją nagradzania o parametrach czujników odległościowych równych, maksymalny zasięg 80cm FOV 12° wygrał: 721
- Czerwony robot uczony z funkcją nagradzania o parametrach czujników odległościowych równych, maksymalny zasięg 40cm FOV 6° wygrał: 279

Na wykresie 6.21 przedstawiono przebieg sparingów w seriach po 50 walk dla powyższych wyników. Widać, że robot niebieski wygrał wszystkie serie.



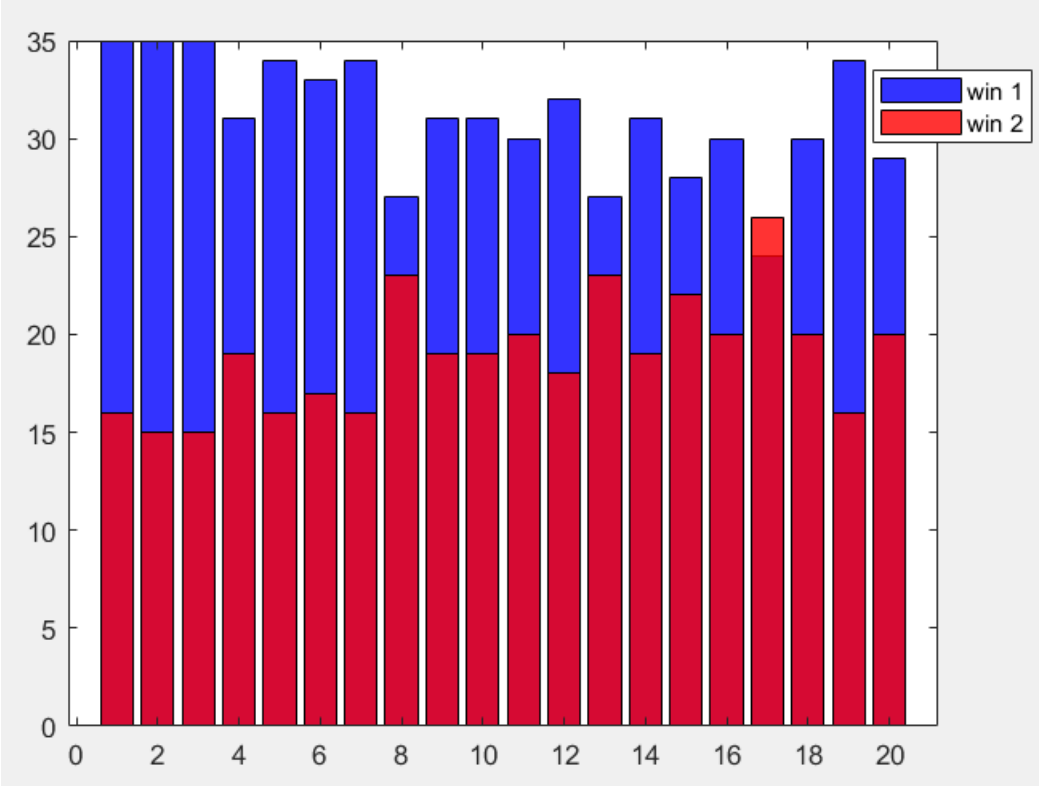
Rys. 6.21: Wykres serii walk (po 50) dla uczenia z funkcją nagradzania przy zmniejszonym polu widzenia

6.5.2. Wykorzystanie wyuczonego agenta dla robota ze zmniejszonym polem widzenia

Dla paramentów czujnika odległościowego, analogicznych do tych opisanych w sekcji 6.5.1, przeprowadzono badania nie zmieniając agenta wyczonego w sekcji 6.3.2. Otrzymane rezultaty wyglądają następująco:

- Niebieski robot o parametrach czujników odległościowych równych, maksymalny zasięg 80cm FOV 12° wygrał: 621
- Czerwony robot o parametrach czujników odległościowych równych, maksymalny zasięg 40cm FOV 6° wygrał: 379

Na wykresie 6.22 przedstawiono przebieg sparingów w seriach po 50 walk dla powyższych wyników. Widać, że robot czerwony wygrał tylko jedną serię.



Rys. 6.22: Wykres serii walk (po 50) wyuczonego agenta, przy zmniejszonym polu widzenia

6.5.3. Podsumowanie zmniejszonych parametrów czujnika odległościowego

Porównując dwie poprzednie sekcje 6.5.1 i 6.5.2 można stwierdzić, że wyuczony robot na wcześniej dobranych parametrach osiągnął lepsze wyniki, niż agent uczony od podstaw. Jest to spowodowane tym, że strategia opisana w sekcji 6.3.2, była przeznaczona dla parametrów przedstawionych w 6.1 oraz testowana metodą prób i błędów.

6.5.4. Pełne uczenie dla robota ze zwiększonym polem widzenia

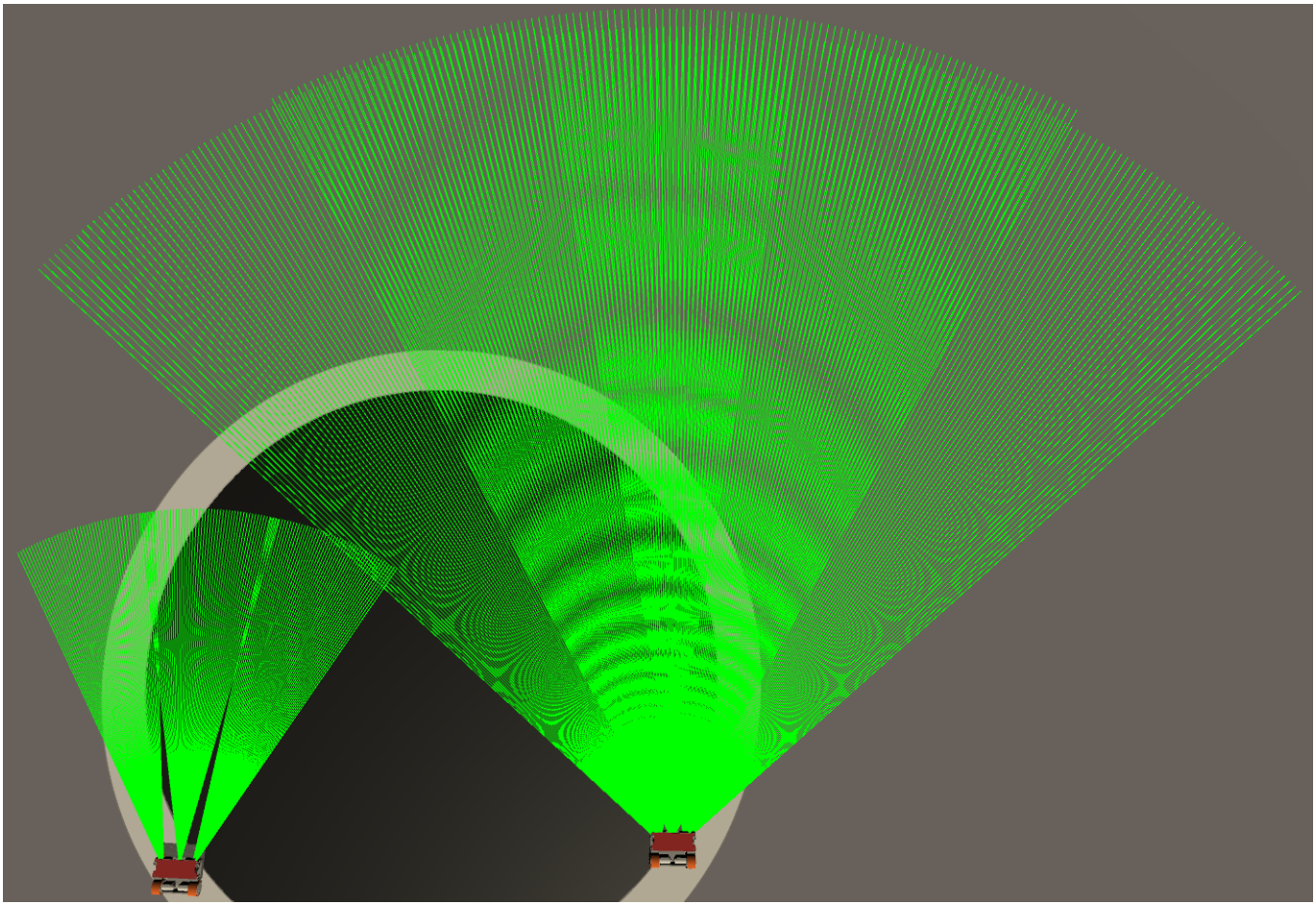
Dalsze badania zostały przeprowadzone na metodzie uczenia ze wzmocnieniem z pojedynczą funkcją nagradzania, opisaną w sekcji 6.3.2, ze względu na lepsze rezultaty przedstawione w sekcji 6.3.3. Zmieniono parametry czujnika odległościowego na:

- maksymalny zasięg 200 cm
- pole widzenia 30°
- czas odświeżania 10ms

zgodnie ze specyfikacją czujnika ultradźwiękowego HC-SR04, dostępną w [13]. Porównanie pola widzenia czujników przedstawiono na zdjęciu 6.23.

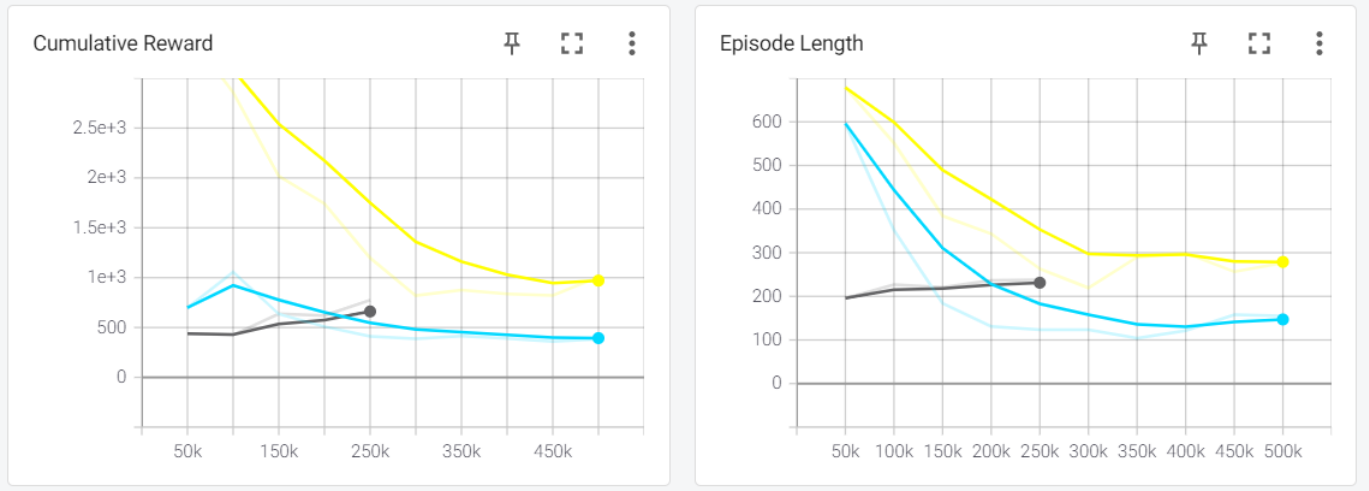
Przeprowadzono trzy etapy uczenia opisane w poprzednich rozdziałach. Trzeci etap polegał na walce z robotem opisanym w sekcji 6.3.2. Efekty treningu agenta były zadowalające, robot poszukiwał przeciwnika, a gdy na niego natrafił, starał się wypchnąć go z ringu. Z pomocą TesnorBoard, opisanego w rozdziale 3.1.1, uzyskano wykres przedstawiający trzy etapy uczenia:

- niebieski pierwszy etap
- żółty drugi etap
- szary trzeci etap - skrócony do 250 tys. kroków



Rys. 6.23: Porównanie pola widzenia czujników odległościowych przy zwiększonym polu widzenia

Kumulacja nagród (Cumulative Reward) przedsawiona na wykresie 6.24 pokazuje, że w pierwszym i drugim etapienagroda agenta wzrastała. uczenia agent nagroda agenta wzrastała. coraz mniejszą ilość nagród. W trzecim etapie uczenia widać, że nagroda agenta zwiększała się. Średnia długość rundy (Episode Length) w przypadku pierwszego i drugiego etapu malała.



Rys. 6.24: Kumulacja nagród i średnia długość rundy przy zwiększonym polu widzenia

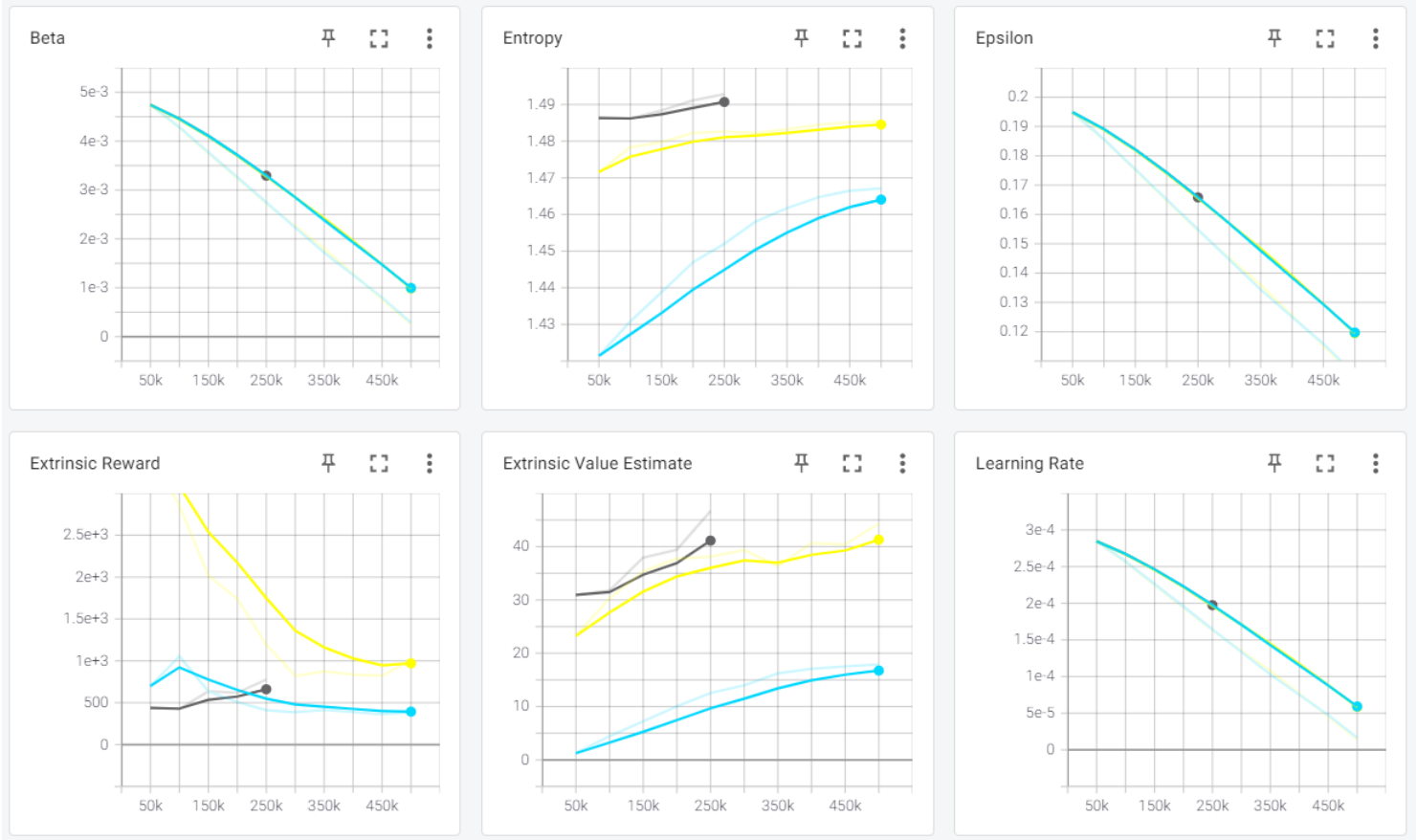
Średnia wartość funkcji straty polityki (Policy Loss) przedstawiona na wykresie 6.25 oscyluje i maleje dla pierwszego i drugiego eatpu.

Wykres średnia utrata aktualizacji funkcji wartości (Value Loss), rośnie dla wszystkich trzech etapów.

Rys. 6.25: Średnia wielkość funkcji straty polityki i utraty wartości w uczeniu przy zwiększonym polu widzenia

Wykres Entropii (Entropy) 6.26 rośnie we wszystkich etapach, co świadczy o nieudanym procesie uczenia.

Średnia oszacowana wartość (Extrinsic Value Estimate) wzrasta dla wszystkich etapów uczenia.

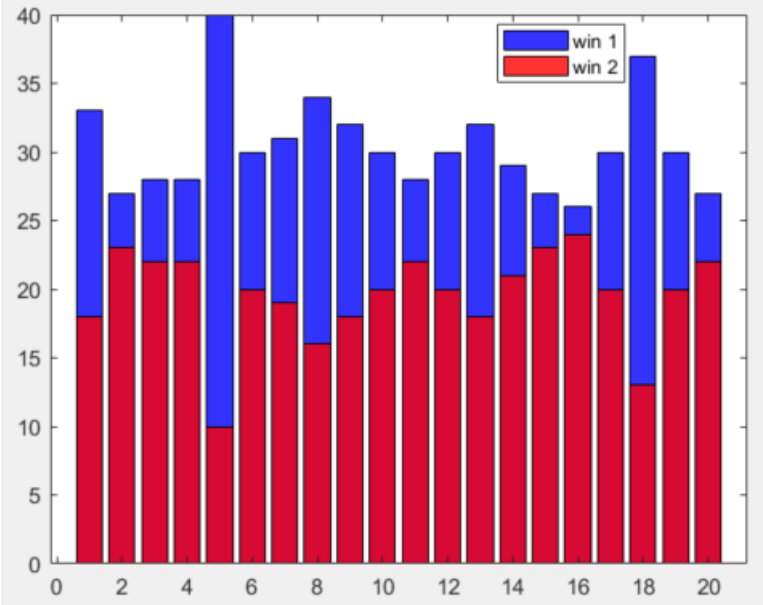


Rys. 6.26: Parametry polityki w uczeniu przy zwiększonym polu widzenia

Robot po drugim etapie uczenia podjął próbę walki z robotem wyuczonym w sekcji 6.3.2. Wyniki prezentują się następująco:

- Niebieski robot uczony z funkcją nagradzania o parametrach czujników odległościowych równych, maksymalny zasięg 80cm FOV 12° wygrał: 609
- Czerwony robot uczony z funkcją nagradzania o parametrach czujników odległościowych równych, maksymalny zasięg 200cm FOV 30° wygrał: 391

Na wykresie 6.27 przedstawiono przebieg sparingów w seriach po 50 walk dla powyższych wyników. Widać, że robot niebieski wygrał wszystkie serie.



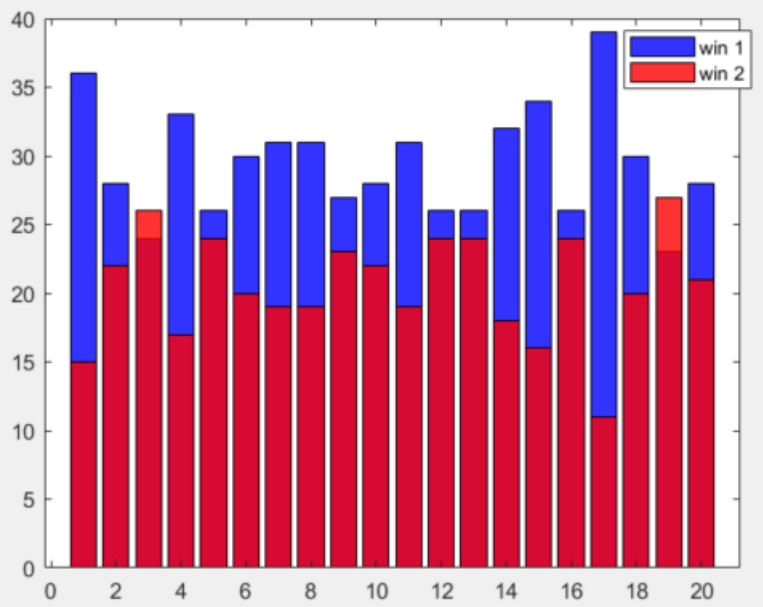
Rys. 6.27: Wykres serii walk (po 50) dla uczenia z funkcją nagradzania przy zwiększonym polu widzenia - po drugim etapie

Robot po trzecim etapie uczenia podjął próbę walki z robotem wyuczonym w sekcji 6.3.2. Wyniki prezentują się następująco:

- Niebieski robot uczony z funkcją nagradzania o parametrach czujników odległościowych równych, maksymalny zasięg 80cm FOV 12° wygrał: 589
- Czerwony robot uczony z funkcją nagradzania o parametrach czujników odległościowych równych, maksymalny zasięg 200cm FOV 30° wygrał: 411

Na wykresie 6.28 przedstawiono przebieg sparingów w seriach po 50 walk dla powyższych wyników. Widać, że robot niebieski przegrał dwie serie. Wskazuje to na poprawę działania robota w trzecim etapie uczenia.

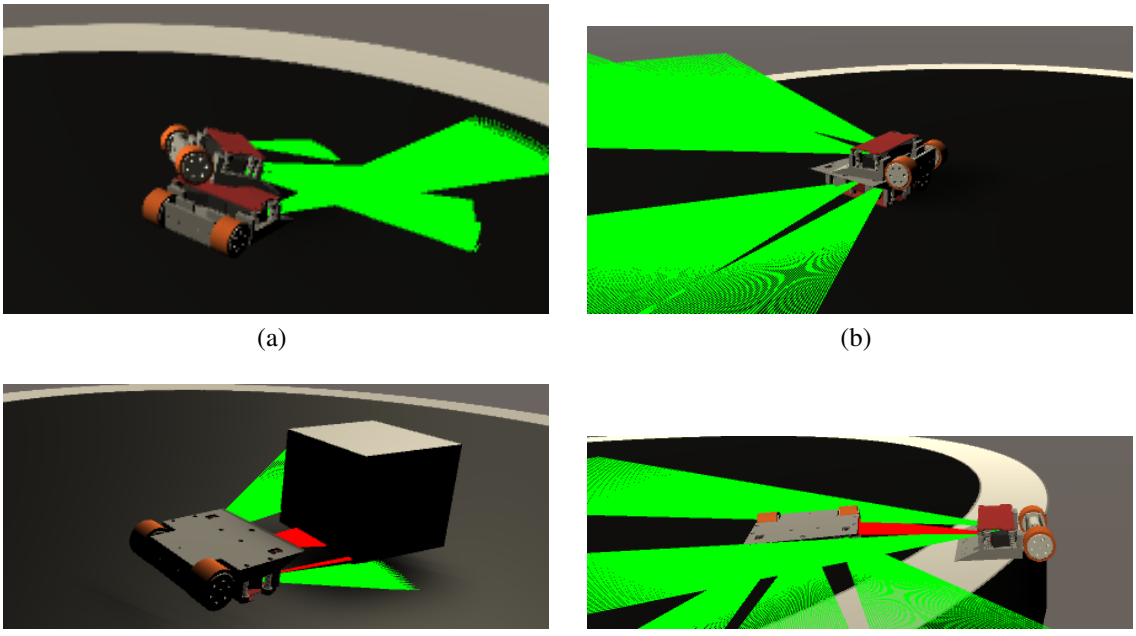
Zwiększenie pola widzenia czujników odległościowych przyniosło lepsze rezultaty niż ich pomniejszenie. Nie można było zastosować wyszkolonego agenta dla badań ze zwiększonym polem widzenia, tak jak było to zrobione dla zmniejszonego pola widzenia, ponieważ przy próbie zastosowania robot od razu spadał z ringu, ze względu na inny zakres parametrów wejściowych. Problem ten mógłby być rozwiązany poprzez normalizację danych, czego zabrakło w powyższej pracy.



Rys. 6.28: Wykres serii walk (po 50) dla uczenia z funkcją nagradzania przy zwiększonym polu widzenia - po trzecim etapie

6.6. Problemy z robotami sumo

Roboty sumo w symulacji jak i w rzeczywistości zaklinowują się, co przedstawiono na obrazach 6.29. W takich sytuacjach zwykle powtarza się rundę. Tego typu przypadki mają również znaczący wpływ na uczenie. Gdy agent stoi w miejscu, może bezproblemowo wykonywać reguły, które odpowiednio go nagrodzą, szczególnie w przypadkach 6.29c i 6.29d.



(a) (b)

(c) (d)

Rys. 6.29: Problemy zaklinowania robotów sumo

Rozdział 7

Podsumowanie

W ramach pracy dyplomowej udało się zaprezentować agenta AI, który nauczył się wygrywać walki w zawodach typu robot sumo. Ponadto porównano różne metody nagradzania (6.3), wpływ zmiany parametrów czujnika odległościowego (6.5) oraz wybrane sposoby sterowań wraz z użyciem LSTM (6.4). Zestawienie to pokazało trudność w samym sposobie nagradzania oraz zasygnalizowało jak ważne są nowe środowiska i różne kombinacje działań dla testowania, a także tworzenia nowych algorytmów ML. Badania ze zmianą paramentów czujników odległościowych wykazały, że dla innych parametrów należy dobrać odmienną metodę nagradzania. Zestaw narzędzi Unity ML-Agents Toolkit przyniósł obiecujące wyniki. Pracę można rozwijać o dalsze badania metod uczenia w ML: SAC, POCA, PPO + Generative Adversarial Imitation Learning (GAIL), + Random Network Distillation (RND), + Behavioral Cloning (BC) opisane w [18]. Analizę tę można przeprowadzić na innych typach robotów np. Line Follower, zaprezentowanym w sekcji 4.5 lub spróbować rozwiązać inny problem, taki jak autonomiczne parkowanie pojazdem za pomocą ML.

Środowisko Unity umożliwia wielokrotne powielanie agenta, a także przyspieszanie symulacji. Możliwości te są jednak uwarunkowane sprzętowo. Nie mniej jednak może ono znacząco przyspieszyć naukę agenta, w przeciwieństwie do rzeczywistych treningów.

Unity pozwala na generowanie obiektu, którego działanie jest realistyczne poprzez manipulację parametrami fizycznej symulacji. Środowisko to daje możliwość kontroli fizyki i zapewnia dynamikę pojazdu w przypadku obiektów. Rigidbody to element dzięki któremu można uzyskać fizyczne zachowanie obiektu. Ciało sztywne przypięte do obiektu, umożliwia obiektowi stosowanie się do grawitacji. Ważną rolę spełnia także Collider, jako jeden z najważniejszych wbudowanych komponentów Unity3D, używany do określenia formy fizycznego zderzenia.

Proces debuggownia nie sprawia trudności w Unity, gdyż pozwala deweloperom na debugowanie procesu w czasie wykonywania. Sprawna modyfikacja iteracji możliwa za pomocą odtwarzania, pozwala wstrzymać działanie aplikacji w przypadku błędu i zmodyfikować jej kod. Usprawnienie procesu debuggowania zapewnia także opcję dokładnego monitoringu symulacji (klatka po klatce) oraz wykrycia niezgodności.

Symulacja wykonana w pracy dyplomowej może być rozszerzana o dodatkowe czujniki takie jak enkoder, akcelerometr czy kompas, dzięki którym możliwe będzie kontrolowanie poruszania się robota. Gdyby wykorzystać żyroskop, symulacja robotów balansujących byłaby możliwa, a tym samym opcja trenowania nowego agenta poprzez ML. Dodanie kamer pozwoliłoby na testowanie trudnych algorytmów rozpoznawania obrazów, których nie da się używać w wielu robotach akcelerometr, ze względu na mało wydajny sprzęt. W połączeniu z uczeniem ze wzmocnieniem, mogłoby to stanowić ciekawy aspekt badawczy.

Literatura

[1] Dokumentacja czujnika odległościowego gp2y0a21yk0f. Źródło: https://www.pololu.com/file/0J85/gp2y0a21yk0f.pdf [dostęp dnia 23 marca 2021].

[2] Dokumentacja czujnika odległościowego gp2y0a21yk0f. Źródło: https://www.kingbright.com/attachments/file/psearch/000/00/00/KTIR0711S(Ver.17).pdf [dostęp dnia 23 marca 2021].

[3] Dokumentacja gameobjects. Źródło: https://docs.unity3d.com/Manual/GameObjects.html [dostęp dnia 29 stycznia 2021].

[4] Dokumentacja gameview. Źródło: https://docs.unity3d.com/Manual/GameView.html [dostęp dnia 3 marca 2021].

[5] Dokumentacja hierarchy. Źródło: https://docs.unity3d.com/Manual/Hierarchy.html [dostęp dnia 3 marca 2021].

[6] Dokumentacja komponentu camera. Źródło: https://docs.unity3d.com/Manual/class-Camera.html [dostęp dnia 3 marca 2021].

[7] Dokumentacja komponentu colliders. Źródło: https://docs.unity3d.com/Manual/CollidersOverview.html [dostęp dnia 25 stycznia 2021].

[8] Dokumentacja komponentu lighting. Źródło: https://docs.unity3d.com/Manual/Lighting.html [dostęp dnia 3 marca 2021].

[9] Dokumentacja komponentu rigidbody. Źródło: https://docs.unity3d.com/Manual/class-Rigidbody.html [dostęp dnia 25 stycznia 2021].

[10] Dokumentacja komponentu wheelcollider. Źródło: https://docs.unity3d.com/Manual/class-WheelCollider.html [dostęp dnia 29 stycznia 2021].

[11] Dokumentacja scenes. Źródło: https://docs.unity3d.com/Manual/CreatingScenes.html [dostęp dnia 25 stycznia 2021].

[12] Dokumentacja sharp gp2y0d340k. Źródło: http://www.farnell.com/datasheets/1679910.pdf [dostęp dnia 9 czerwca 2021].

[13] Dokumentacja ultrasonic ranging module hc - sr04. Źródło: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf [dostęp dnia 14 czerwca 2021].

[14] Dokumentacja usingtheinspector. Źródło: https://docs.unity3d.com/Manual/UsingTheInspector.html [dostęp dnia 3 marca 2021].

[15] Git – co, gdzie, jak i dlaczego? Źródło: https://www.nettecode.com/git-dlaczego-materialy-start/ [dostęp dnia 28 stycznia 2021].

[16] Github – podstawy. Źródło: http://devfoundry.pl/github-podstawy/ [dostęp dnia 28 stycznia 2021].

[17] Teoria robotyki. Źródło: https://www.robotyka.com/teoria.php/teoria.5 [dostęp dnia 12 stycznia 2021].

[18] Training configuration file - ml agents. Źródło: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md [dostęp dnia 13 maja 2021].

[19] Unity opis bibliotek. Źródło: https://docs.unity3d.com/Manual/index.html [dostęp dnia 25 stycznia 2021].

[20] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath. *A Brief Survey of Deep Reinforcement Learning*. arXiv, 2017.

[21] J. Cheng, L. Dong, M. Lapata. *Long Short-Term Memory-Networks for Machine Reading*. arXiv, 2016.

[22] P. Cichosz. Uczenie się ze wzmocnieniem (część 1). Źródło: http://staff.elka.pw.edu.pl/~pcichosz/um/wyklad/arch/wyklad12/wyklad12.html [dostęp dnia 28 styczeń 2021].

[23] H. Erdem. *Application of Neuro-Fuzzy Controller for Sumo Robot control*. Elsevier, 2011.

[24] J. Hocking. *Unity w akcji*. Helion, 2017.

[25] K. JAGACIAK. C# w unity. poznaj narzędzie do tworzenia gier. Źródło: https://www.komputerswiat.pl/poradniki/programy/c-w-unity-poznaj-narzedzie-do-tworzenia-gier/7yx77qg [dostęp dnia 12 lutego 2021].

[26] W. Mariusz J. Giergiel, Zennon Hendzel. *Modelowanie i sterowanie mobilnych robotów kołowych*. Wydawnictwo Naukowe PWN, 2013.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. arXiv, 2013.

[28] S. Saadatrnand, S. Azizi, M. Kavousi, D. Wunsch. *Autonomous Control of a Line Follower Robot Using a Q-Leaming Controller*. IEEE, 2020.

[29] J. Szczyrk. *Symulator robotów mobilnych w narzędziu Unity*. 2019.

[30] E. Teng. Training your agents 7 times faster with ml-agents. Źródło: https://blog.unity.com/technology/training-your-agents-7-times-faster-with-ml-agents [dostęp dnia 25 maja 2021].

[31] M. Urmanov, M. Alimanova, A. Nurkey. *Training Unity Machine Learning Agents using reinforcement learning method*. IEEE, 2019.

[32] R. Zajdel. *Wiarygodność reguł rozmytych otrzymanych w procesie uczenia*. Wydawnictwo PAK, 2003.