POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA
SPECJALNOŚĆ: SYSTEMY I SIECI KOMPUTEROWE

PRACA DYPLOMOWA MAGISTERSKA

Analiza porównawcza metod reinforcement i imitation learning na potrzeby autonomicznego przemieszczania się w środowisku 3D

Comparative analysis of reinforcement and imitation learning for autonomous movement in 3D environment

AUTOR :
Oskar Speruda

OPIEKUN PRACY:
Dr inż. Marek Woda, W4/K9

OCENA PRACY:

WROCŁAW 2019

Spis treści

1 Wstęp

W ostatnich latach autonomiczne samochody stanowią ważny punkt rozwoju technologii. Wielkie koncerny takie jak BMW, *Tesla* czy *Google* poświęcają olbrzymie środki na opracowywanie prototypów samojezdnych pojazdów. Rosnące zainteresowanie tym kierunkiem rozwoju branży motoryzacyjnej nie jest przypadkowe. Specjaliści nie mają wątpliwości, że czeka nas era samochodów autonomicznych [1]. Gdy to nastąpi, firmy, które opracują najlepsze rozwiązania mogą liczyć na wysoki popyt na tego typu produkty, a co za tym idzie wymierne zyski.

Przewidywania co do wprowadzenia tej technologii różnią się [1]. Optymistyczne predykcje zakładają, że już w 2030 roku pojazdy autonomiczne będą wystarczająco dostępne, aby w większości zastąpić tradycyjne samochody. Wykorzystanie samojezdnych pojazdów może przynieść wymierne korzyści. Zwolnienie kierowcy z odpowiedzialności za prowadzenie samochodu pozwali ograniczyć w ruchu drogowym tzw. czynnik ludzki, tj. zmęczenie, rozkojarzenie czy braki w umiejętnościach prowadzącego. Pomoże to zatem w zwiększeniu bezpieczeństwa na drodze, zmniejszy liczbę wypadków, a tym samym liczbę ofiar śmiertelnych tych zajść. Ponadto zastosowanie na szeroką skalę autonomicznych samochodów może ograniczyć zatory i utrudnienia na drodze, pozwalając zminimalizować czas podróży oraz pośrednio ograniczyć straty finansowe spowodowane na przykład spóźnieniami do pracy, czy opóźnieniami w dostawie.

1.1 Cel i zakres pracy

Przedmiotem pracy jest problem autonomicznego przemieszczania się w przestrzeni 3D powiązany w szczególności z technologią samojezdnych pojazdów oraz zastosowanych w nich modelach sztucznej inteligencji. Zdobyta wiedza w pewnym stopniu może być przydatna również w innych dziedzinach, tj. zastosowanie sztucznej inteligencji w grach komputerowych symulujących ruch pojazdów.

Posiadając zasoby nieporównywalnie mniejsze od koncernów pracujących nad rozwojem technologii autonomicznego przemieszczania się, zdecydowano się na zasymulowanego problemu w komputerowym środowisku 3D. Ponadto ze względu na ograniczenia sprzętowe oraz czasowe zdecydowano się na pewne uproszczenia problemu. Jednakże część ustaleń oraz uzyskanych informacji będzie mogła znaleźć odniesienia do problemu ogólnego.

W pracy skoncentrowano się na aspekcie zastosowania nauczania maszynowego w omawianym problemie. Uwzględniono przy tym dwie strategie reprezentujące odmienne podejścia do zagadnienia uczenia maszyn. *Imitation learning* prezentuje bardziej klasyczną drogę nauczania maszynowego, budując model na podstawie dostarczonych mu przykładów. W *reinforcement learning*, komputer uczy się na podstawie własnych obserwacji i akcji, opierając się na sygnale zwrotnym, który może przyjmować wartości pozytywne oraz negatywne w zależności działań podejmowanych przez agenta.

W celu porównania wyżej wymienionych metod, zaimplementowane zostanie środowisko badawcze, w którym symulowany będzie ruch pojazdu w przestrzeni 3D. Model pojazdu sterowany przez sztuczną inteligencję ma za zadanie pokonać trasę przygotowanego toru, unikając przy tym kolizji z przeszkodami. Przed przystąpieniem do symulacji konieczne jest oprogramowanie środowiska oraz pojazdu w szczególności opracowując metody zbierania

obserwacji oraz wykonywania akcji. W przypadku metody *reinforcement learning*, ważnym aspektem jest implementacja metody przyznawania sygnału nagrody.

Dla obu strategii zostaną wykorzystane gotowe algorytmy nauczania maszynowego *Proximal Policy Optimization* oraz *Behavioral Cloning*. Przed przystąpieniem do procesu nauki konieczna jest odpowiednia konfiguracja oraz zapewnienie komunikacji pomiędzy wykorzystywanymi narzędziami.

Wielokrotne powtarzanie procesu nauczania i dla różnych wartości parametrów pozwoli na ich odpowiedni dobór. Uzyskane modele, zostaną przetestowane, w odpowiednich próbach, podczas których zostaną zebrane dane pozwalające na ocenę modeli według określonych kryteriów porównawczych. Po zakończeniu procesu parametryzacji, konieczne będzie przeprowadzenie treningu dla wybranego zestawu wartości parametrów oraz ponowne przetestowanie modeli. Następnie zostanie przeprowadzana analiza uzyskanych danych. Ostatecznie dokonane zostanie podsumowanie przydatności i efektywności testowanych metod dla opracowywanego problemu.

1.2 Układ pracy

Praca podzielona jest na 7 rozdziałów. Po słowach wstępu oraz omawianiu celów oraz zakresu pracy przedstawiony jest obszerny opis tematyki związanej z badaniami. Najpierw opisywana jest tematyka autonomicznego przemieszczania się, następnie omawiane jest zagadnienie nauczania maszynowego z wyróżnieniem metod *reinforcement* i *imitation learning*. Ostatnim punktem rozdziału jest przegląd literatury dotyczącej tematyki nauczania maszynowego w autonomicznym poruszaniu się. W kolejnym rozdziale przedstawiana jest koncepcja realizacji, gdzie przedstawione są wybrane technologie oraz algorytmy. W rozdziale 4 opisano implementację środowiska, która została podzielona na implementację przestrzeni treningu, oraz znacznie dłuższy rys implementacji agenta sztucznej inteligencji. Następnie opisano sposób konfiguracji środowiska. Rozdział 6 poświęcono przeprowadzonym badaniom. Na początku opisano kryteria i założenia badawcze. Samo badanie podzielono na 3 etapy: porównanie początkowe, proces parametryzacji, by w końcu przejść do prób porównawczych. Wraz z przedstawionymi wynikami zamieszczono ich omówienie. W ostatnim rozdziale dokonano podsumowania uzyskanych wyników oraz spostrzeżenia dotyczące możliwości dalszego rozwoju badań.

2 Przedstawienie tematyki pracy

2.1 Autonomiczne przemieszczanie

Najbardziej popularnym, łączonym z problemem autonomicznego przemieszczaniem się, jest zagadnienie samojezdnych samochodów określanych również jako samochody bezzałogowe, autonomiczne lub samojezdne. Pojazdy takie mają możliwość zbierania informacji z otoczenia i poruszają się, bez udziału lub z niewielkim udziałem człowieka w sterowaniu nimi. Kwestią sporną pozostaje stopień, kiedy wkład ludzki jest na tyle mały, że samochód można określić mianem autonomicznego. Jeden z czołowych producentów samochodów na Świecie, niemiecka marka BMW, wyróżnia pięć stopni takiej autonomiczności [2]. Każdy z nich opisuje poziom, w jaki samochód przejmuje obowiązki kierowcy oraz sposób interakcji samochodu i kierowcy.

Pierwszy poziom, to systemy asystujące kierowcy, wspomagające wygodę i bezpieczeństwo. Duża część obecnie produkowanych pojazdów wyposażona jest w takie systemy. Możemy do nich zaliczyć m.in. aktywny tempomat, która mierzy i kontroluje dystans pomiędzy pojazdem znajdującym się przed samochodem, hamując w przypadku, gdy odległość ta staje się zbyt mała.

Poziomem drugim jest częściowo zautomatyzowane prowadzenie, poprzez zaawansowane systemy wspomagające. Przykładem może być asystent kontroli pasa, który monitoruje pas jezdni oraz zapobiega niezamierzonej zmianie pasa ruchu przez samochód. W takie systemy wyposażane są obecnie produkowane samochody klasy wyższej.

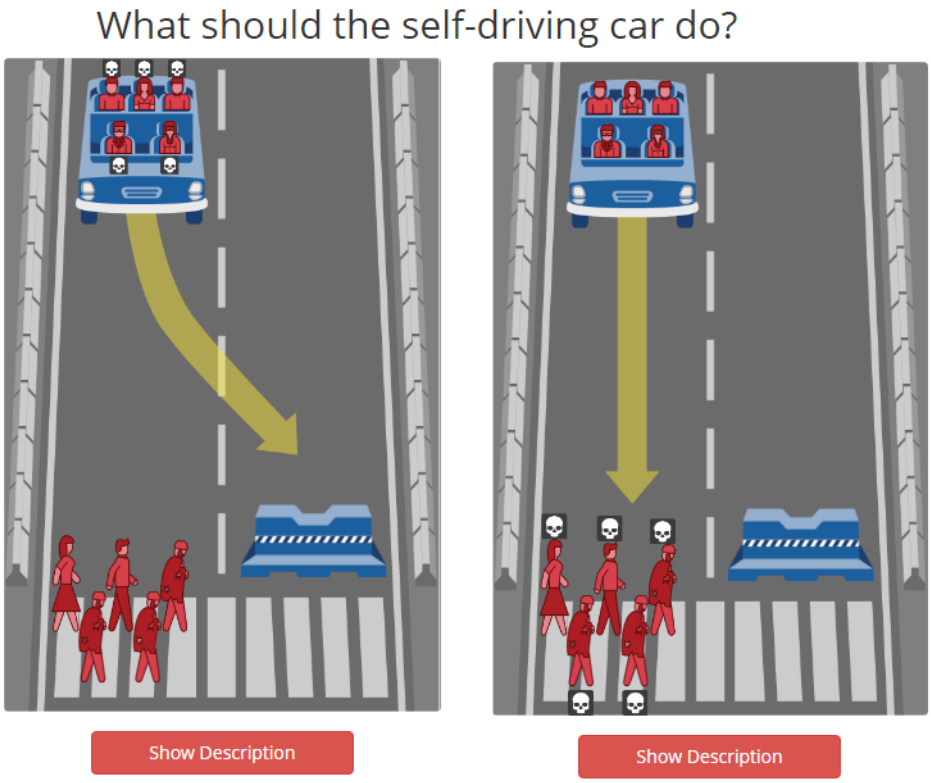
Kolejnym wyróżnionym stopniem jest wysoce zautomatyzowane prowadzenie. Są to systemy współprowadzące, które w pewnych warunkach pozwalają kierowcy na kompletne odwrócenie uwagi od prowadzenia samochodu i przejęcie sterowania przez komputer. Jednak, w niektórych warunkach kierujący musi mieć możliwość przejęcia ponownej kontroli nad pojazdem w ciągu kilku sekund. Takie systemy są od kilku lat intensywnie testowane, również w publicznym ruchu drogowym.

Czwarty stopień to w pełni zautomatyzowane pojazdy. Pomimo, że samochód ma kompletnie wyposażony kokpit i kierowca wciąż na własne życzenie może przejąć kontrolę nad pojazdem, komputer jest w stanie obsłużyć zdecydowaną większość sytuacji na drodze bez konieczności ingerencji człowieka. Kierowca musi jednak wciąż zasiadać w kokpicie samochodu i być gotowym na reagowanie w sytuacjach krytycznych.

W najwyższym, piątym stopniu kierowca nie jest potrzeby, wszystkie osoby pozostające w pojeździe stają się pasażerami, a co za tym idzie takie pojazdy nie muszą posiadać nawet kokpitu. Implementacja takiego poziomu autonomiczności wymaga jednak spełnienia najwyższych standardów bezpieczeństwa. BMW przewiduje, że tego typu pojazdy będą poruszały się tylko po specjalnie do tego przeznaczonych przestrzeniach.

Oczywiście docelowym i pożądanym poziomem jest stopień najwyższy. Jednak niesie on za sobą najwięcej niebezpieczeństw i obaw. Poruszanie się w pełni autonomicznego pojazdu, bez kierowcy, który mógłby przejąć kontrolę w sytuacjach awaryjnych, oznacza konieczność podejmowania przez komputer wielu ważnych decyzji, niekiedy o charakterze etycznym. Jednym z najbardziej popularnych problemów jest ten związany z eksperymentem naukowców z *Massachusetts Institute of Technology* pod nazwą *Moral Machine* [3]. Porusza on problem podjęcia wyboru, w przypadku, gdy nie ma możliwości w pełni uniknięcia

wypadku, jednak istnieje możliwość wyboru jego skutków. Przed ankietowanymi postawiono szereg ilustrowanych pytań, w odpowiedzi, na które należało zdecydować m.in., która grupa uczestników ruchu powinna zostać uderzona przez samochód, a bezpieczeństwo której z grup uczestników powinno stanowić priorytet. Pojawił się między innymi problem czy samochód powinien w pierwszej kolejności chronić bezpieczeństwo pasażerów, czy też przechodniów.



Rysunek 2.1 Jedna z instancji problemu autonomicznych samochodów przedstawionych w eksperymencie Moral Machine (źródło: [3])

Takie problemy są szczególnie trudne, gdyż na tego typu pytania trudno postawić jednoznaczną odpowiedź. Kolejną kwestią jest odpowiedzialność za wypadek. Czy powinna spoczywać na właścicielu samochodu autonomicznego, czy na jego producencie? Obecnie nie ma regulacji prawnych w zakresie sytuacji związanych z udziałem w ruchu drogowym pojazdów samojezdnych. Jeżeli ktoś chciałby takowe wprowadzić, bez względu na ich treść, zapewne wywołałby dyskusje, kontrowersje oraz głosy krytyczne. To właśnie takie kwestie są jednym z największych problemów w odniesieniu do przyszłości samochodów autonomicznych.

Za kamień milowy w dziejach samojezdnych samochodów, można uznać konkurs *Gran Challenge*, zorganizowany w 2004 roku przez amerykańską, rządkową agencję obronny DARPA (*Defense Advanced Research Projects Agency*) [4]. 15 samochodów stanęło na linii startu na kalifornijskiej pustyni. Celem było przejechanie 142 milowego toru w trybie autonomicznym. Zespół, którego pojazd pokona trasę w najkrótszym czasie miał otrzymać nagrodę pieniężną w wysokości 1 miliona USD. Niestety, żaden samochód nie ukończył wyścigu. Najwyżej oceniony pojazd pokonał zaledwie 7,5 mil [4]. Nie osiągnięto zakładanego celu, jednak eksperyment ten pozwolił na zawiązanie społeczności naukowców, studentów i pasjonatów skupionych wokół problemu, a ponadto konkurs wpłynął na popularyzację problemu autonomicznego przemieszczania. W kolejnej edycji imprezy, która

odbyła się w 2005 roku przejazd udało się pomyślnie ukończyć 5 spośród aż 195 startujących zespołów. Z czasem 6 godzin i 53 minut zwyciężył samochód *Stanley* Uniwersytetu *Stanford*. W 2007 roku zorganizowano *Urban Challenge*, w którym samochody, miały pokonać tor mający symulować warunki miejskie. Trudnością było manewrowanie w ruchu drogowym, między poruszającymi się innymi samochodami, jednocześnie stosując się do przepisów i znaków. 6 spośród 11 zespołów pomyślnie ukończyło tę próbę [4].

W 2009 roku *Google* rozpoczęło pracę nad swoim projektem samochodu autonomicznego zatrudniając zespół projektantów spośród uczestników biorących wcześniej udział w wyzwaniach DARPA. Po zaledwie 18 miesiącach ich system potrafił operować na publicznych drogach stanu Kalifornia. W ciągu kilku lat w ślad za *Google* poszła *Tesla* czy *Uber*. Do wyścigu technologicznego dołączyły także znani producenci samochodów, tacy jak *Nissan, Mercedes* czy *Ford* [5].

Założyciel firmy *Telsa*, Elon Musk zapowiedział [6], że w 2020 roku autopilot montowany w samochodach tej marki będzie tak zaawansowany, że kierowca nie będzie musiał zwracać uwagę na wydarzenia na drodze (4 poziom autonomiczności). Stwierdził również, że w ciągu najbliższych lat mogą pojawić się pierwsze *Robotaxi*, nieposiadające kierownicy oraz osiągające 5 poziom autonomiczności. Wprowadzenie takich pojazdów na drogi wymagałoby zmiany w prawie o ruchu drogowym.

Prototypy samochodów samojezdnych wykorzystują różne technologie w celu zbierania danych z otoczenia [7]. Możemy wyróżnić między innymi lidary i radary, które są w stanie w dokładny sposób określić pozycję otaczających samochód przeszkody. Ważnym aspektem jest również analiza obrazu kamer, która pozwala między innymi na rozpoznawanie znaków drogowych. W kwietniu 2019 roku firma *Tesla* poinformowała [6], że wszystkie produkowane przez nią samochody są wyposażane w sprzęt wymagany dla w pełni autonomicznej jazdy. Pojazdy *Tesli* wyposażane są w 8 kamer wizyjnych, 12 czujników ultradźwiękowych radar oraz specjalnie zaprojektowany komputer.

Komputery pokładowe często posiadają stały dostęp do Internetu, dzięki czemu mogą komunikować się z innymi pojazdami lub systemami ostrzegania. Samochód zna również swoją pozycję i prędkość dzięki np. powszechnie stosowanemu systemowi GPS.

Zbierane w ten sposób dane są przekazywane do komputera. Algorytmy przetwarzają dane wejściowe i decydują o tym jak zachować się ma samochód, co odpowiada konkretnym ruchom siłowników i sterowników odpowiedzialnych za układy jezdny, hamowania czy moc silnika.

2.2 Nauczanie maszynowe

Nauczanie maszynowe to dział sztucznej inteligencji, w której komputer prowadzi do pewnej optymalizacji operacji bazując na przykładach lub doświadczeniach. Aby uzyskany model był inteligentny, powinien móc się dostosowywać do zmieniającego się środowiska. Taka adaptacyjność jest możliwa dzięki umiejętności nauki [8].

Miliony ludzi codziennie generuje ogromne ilości danych. Przedsiębiorstwa informatyczne, takie jak *Google* nieustannie gromadzą dane o użytkownikach swoich usług, które stanowią prawdziwą cenę pozornie darmowych usług. Posiadanie takich zasobów danych stymuluje rozwój *machine learning*-u.

W nauczaniu maszynowym prowadzący badanie nie musi sam szukać zależności i tworzyć modeli matematycznych. Często wystarczy odpowiednio duży zestaw danych

wejściowych powiązanych z danymi wyjściowymi. Dzięki odpowiedniemu algorytmowi i iteracyjnym powtórzeniom, komputer jest w stanie dostrzec powiązania, jakie człowiekowi trudno byłoby zauważyć, a tym bardziej opisać matematycznie. Taka operacja możne pomóc w zrozumieniu badanego procesu lub postawieniu pewnych założeń i znalezieniu regularności

Warto jednak nadmienić, że pomimo ogromnym możliwością jakie jest w stanie posiąść odpowiednio wyszkolony model, maszyna nie zyskuje świadomości swoich czynów, tworzy jedynie model matematyczny mapując dane wejściowe do danych wyjściowych.

Machine learning powszechnie wykorzystywany jest między innymi w medycynie (diagnozowanie chorób), bankowości (wykrywanie oszustw), telekomunikacji (optymalizacja efektywności przepływu danych) czy robotyce. Nawet w urządzeniach codzienne użytku, jak telefon czy samochód, możemy spotkać się z usługami opartymi o nauczanie maszynowe, takimi jak rozpoznawaniu mowy, twarzy lub pisma.

W uczeniu maszyn wyróżnić można trzy główne klasy metod [9]:

- *Unsupervised Learning* (Uczenie nienadzorowane)

W tych metodach algorytmowi nie są zapewniane dodatkowe dane wyjściowe, takie jak etykiety czy przykłady. Wkładem ze strony przeprowadzającego w proces treningu jest odpowiedni dobór atrybutów. Często zadaniem algorytmu jest podzielenie danych na mniejsze grupy, klastry (*clustering*). Podejście to jest szczególnie przydatne, wtedy, kiedy odpowiednie etykietowanie danych jest niemożliwe lub kosztowne (tzn. wymagałoby na przykład dużych nakładów pracy i czasu).

- *Supervised Learning* (Uczenie nadzorowane)

W przeciwieństwie do nauczania nienadzorowanego, zakłada obecność nadzoru (nauczyciela) np. poprzez zadanie pożądanej przez nadzorcę odpowiedzi (etykiety) przypisanej do obiektów należących do zbioru uczącego. Na podstawie danych wyjściowych algorytm jest w stanie zaobserwować wzorce i wygenerować odpowiedni model matematyczny. Tak wytworzony model pozwolić może na późniejszą predykcję dla nieetykietowanych obiektów. Do klasy tej zaliczyć można metodę *imitation learning* (rozdział 2.2).

- *Reinforcement Learning* (Uczenie przez wzmacnianie)

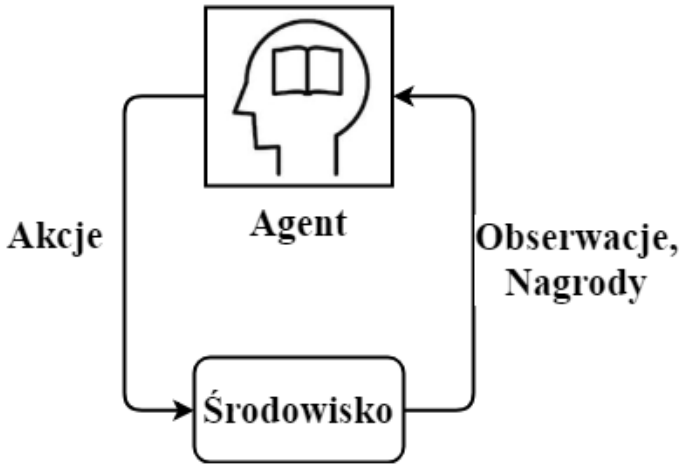
Omówione w podrozdziale 2.2.1.

2.2.1 Reinforcement learning

W języku polskim metoda określana jako *uczenie przez wzmacnianie* lub *nauka z wzmacnianiem*. W tej strategii nauczania maszynowego polityka modelu (sekwencja działań prowadzących do osiągnięcia celu), tworzona jest na podstawie zebranych doświadczeń. Najczęściej nie jest ważna pojedyncza akcja, lecz to, że sekwencja poprawnych akcji doprowadzi do osiągnięcia celu. Z tego powodu *reinforcement learning* często jest stosowany w badaniach związanych z grami, gdzie łatwo jest opisać cel rozgrywki, jednak problemem jest ustalenie sekwencji prowadzącej do wygranej (trudno jest grać w te gry z wysoką skutecznością) [8].

W tym podejściu agent sam generuje dane treningowe poprzez interakcje ze środowiskiem i zbieranie obserwacji. Wyróżnia to omawiane podejście od pozostałych strategii uczenia maszynowego, np. uczenia nadzorowanego, gdzie dostarczany jest statyczny zestaw danych treningowych. Na podstawie działań agenta, według zadanego algorytmu przyznawane jest

sprzężenie zwrotne w postaci sygnału nagrody, które jest podstawą do opracowywania polityki modelu.



Rysunek 2.2 Schemat przebiegu nauki reinforcement learning (źródło własne)

Proces nauki można porównać do szkolenia szczeniaka, który uczy się w myśl zasady *prób i błędów*. Za pożądane od niego działania jest nagradzany, za złe karcony. Szczenięcia ciekawość sprawia, że zwierzę próbuje różnych działań i obserwuje ich wyniki. W ten sposób kształtowany jest jego charakter.

Trening *reinforcement learning* wymaga wielu prób i literacyjnego aktualizowania polityki. Początkowe działania podejmowane przez agenta są w dużym stopniu losowe, a sekwencje działań bardzo rzadko prowadzą do oczekiwanych rezultatów. Kiedy jednak wykonana sekwencja działań doprowadza do otrzymania pozytywnej nagrody, algorytm zaktualizuje politykę w ten sposób, aby te akcje były wykonywane z większym prawdopodobieństwem w przyszłości. Czas potrzebny do wygenerowania modelu potrafiącego wykonywać postawione mu zadanie jest zależny od złożoności problemu.

Problemem często napotykanym w tym podejściu jest kwestia rzadkiej nagrody (*sparse reward)* [10]. W przypadku skomplikowanych problemów, sekwencja działań wymaganych do osiągnięcia celu może być na tyle długa, że pomimo wielu prób agentowi bardzo trudno jest w sposób losowy wygenerować sekwencję, która zaowocuje pożądanym rezultatem. Przykładem takiego problemu może być precyzyjne operowanie ramieniem robota, w celu przemieszczania obiektów przedstawione w [11]. Może to spowodować niską lub zerową efektywnością procesu nauki. Rozwiązaniem może być odpowiednie kształtowanie nagrody (*reward shaping).* Zabieg ten polega na odpowiednim zaprojektowaniu funkcji przyznawania nagrody, w ten sposób, aby nie tylko nagradzać agenta w momencie wykonania poprawnie całej sekwencji działań, ale również dawać mu odpowiednie sprzężenie zwrotne w trakcie jego wykonywania, tak aby niejako nakierować go na rozwiązanie. Taka funkcja musi być zaprojektowana specjalnie z myślą o danym środowisku testowym. W przypadku błędnej implementacji funkcji nagrody może dojść do sytuacji, w której agent odnajdzie sposób na maksymalizację wartości otrzymywanej nagrody, nie wykonując przy tym oczekiwanego zadania. Ponadto należy unikać sytuacji, w której strategia przyznawania nagrody, narzuci konkretny sposób działania agenta (np. zbliżony do wzorca ludzkiego), co może doprowadzić do nieoptymalnego zachowania agenta [9]. Innymi słowy, polityka przyznawania nagrody nie powinna wskazywać drogi jego rozwiązania. Pogodzenie tych warunków często jest bardzo trudne, a odpowiednia implementacja funkcji nagrody może decydować o sukcesie lub porażce procesu uczenia.

Nauczanie przez wzmacnianie, mimo iż przez długi czas znajdowało się w cieniu strategii nauczania nadzorowanego, obecnie zyskuje coraz większe uznanie oraz prezentowane są jego kolejne aplikacje. Gry od dawna są wykorzystywane jako ważny sposób do testowania, oceniania oraz demonstrowania wydajności systemów sztucznej inteligencji. Warto wspomnieć tu słynny komputer *Deep Blue* opracowany przez firmę IBM w latach 90-tych 20 wieku [12]. Podobnie rozwojowi *reinforcement learning* towarzyszy implementacja kolejnych systemów mających rywalizować z człowiekiem.

W 2013 roku przedsiębiorstw *DeepMind* zaprezentowało [13] jak opracowanemu przez nich modelowi, uzyskanego z zastosowaniem *reinforcement learning* udało się opanować gry na konsolę z lat 70-tych *Atari 2600*. Co ważne jako dane wejściowe wykorzystano surowe, w żadne sposób nieprzetworzone piksele generowane przez grę. Mimo dużego narzutu danych nadmiarowych, algorytm pomyślnie opanował 6 z 7 testowanych gier.

Ta sama organizacja opracowała również program *AlphaGo* [14], mający pojedynkować się w chińskiej grze logicznej *Go*. Specyfika gry sprawiała, że dotąd pomimo wielu lat pracy nawet najmocniejsze komputery, nie były w stanie opanować jej lepiej niż do poziomu amatora. *DeepMind* w swoim projekcie połączyło bardziej tradycyjne podejście, takie jak zaawansowane przeszukiwanie drzew z głębokimi sieciami neuronowymi. Ponadto przed tradycyjnym procesem nauczania metodą *reinforcement learning* pozwoliła komputerowi przeanalizować dużą liczbę gier amatorskich w celu rozwinięcia *rozumienia* podstawowych i uzasadnionych w danej sytuacji zagrań. W tym przypadku dokonano niejako połączenia dwóch strategii: uczenia nadzorowanego oraz uczenia przez wzmocnienie. W 2015 roku system *AlphaGo* pokonał mistrza Europy, a w 2016 roku w Seulu Pana *Lee Sedol* uznawanego za najlepszego gracza *Go* tej dekady.

Pod koniec 2016 roku, organizacja non-profit *OpenAI* rozpoczęła pracę nad projektem *Five* [15]. Celem była realizacja systemu sztucznej inteligencji, który mógłby rywalizować w popularnej, internetowej grze *DOTA 2*. Opiera się ona na rozgrywaniu meczy w trybie wieloosobowym. 10 graczy podzielonych na dwie drużyny bronią bazę swojego zespołu oraz próbują zniszczyć fortyfikacje przeciwnika [16]. Jest to jedna z najpopularniejszych gier komputerowych na świecie. Decyzję o jej wyborze wspierał fakt, że gra udostępnia interfejs API oraz może być również uruchomiona na komputerze z systemem *Linux*. *DOTA 2* stanowiła wyzwanie, gdyż charakteryzuje się wysokim poziomem trudności, koniecznością strategicznego planowania oraz wymaga ścisłej współpracy między graczami. Już w sierpniu 2017 roku uzyskany model był w stanie pokonać topowego, profesjonalnego gracza w trybie pojedynku 1 kontra 1. Rok później sztuczna inteligencja kilkukrotnie pokonała różne zespoły złożone z graczy zarówno amatorskich, półprofesjonalnych oraz profesjonalnych. Warto nadmienić, że drużyna złożona z agentów sztucznej inteligencji poniósł również porażkę. Na rozgrywki takie nakładane są pewne ograniczenia, na przykład, program ma czas reakcji wydłużony do ludzkiego, tak aby pojedynek był bardziej wyrównany. Danymi wejściowymi dla komputera w jednym kroku pracy algorytmu jest lista 20 000 liczb, które odpowiadają zadanemu stanowi gry ograniczonemu do takich, do których dostęp ma każdy gracz w czasie meczu. Dane wyjściowe to lista liczb o długości 8. Podczas rozgrywki zespołu 5 agentów, wykorzystywane są zasoby 128 000 rdzeni procesora oraz 256 procesorów graficznych. *OpenAI* dla projektu *Five* stosuje *reinforcement learning* z wykorzystaniem autorskiego algorytmu *Proximal Policy Optimization* [17]

W styczniu 2019 *DeepMind* zaprezentowało projekt *AlphaStar* [18], w ramach którego zaimplementowany został system mający na celu rywalizować w popularną, komputerową grę strategiczną *StarCraft*, która uważana jest za jedną z najtrudniejszych i wymagających gier kompetytywnych. Największymi wyzwaniami w grze może być fakt konieczności

opracowywania długoterminowych strategii, która musi być modyfikowana na bieżąco w zależności od działań przeciwnika, brak pełnej informacji o zachowaniu przeciwnika (najważniejsze z nich ukryte są za *mgłą wojny*), konieczność podejmowania decyzji w czasie rzeczywistym czy olbrzymi wektor możliwych akcji wynikających, m.in. z mnogości dostępnych jednostek i modyfikacji. Podobnie jak w przypadku opisywanego projektu *Five*, model najpierw został przeszkolony z użyciem strategii uczenia nadzorowanego analizując powtórki meczy rozegranych przez ludzi, co pozwoliło mu na imitację zachowań graczy. W ten sposób wygenerowano wiele modeli, które następnie rywalizowały między sobą w specjalnie opracowanej lidze. Zaprezentowano tutaj również podejście czerpiące z algorytmu genetycznego, gdyż liga była wciąż uzupełniana kolejnymi mutacjami istniejących agentów. Parametrami poszczególnych modeli podczas tego iteracyjnego treningu zarządzał algorytm strategii *reinforcement learning*, który na podstawie wyników poszczególnych starć odpowiednio je dostrajał. W trakcie ewolucji agenci skłaniali się do wielu strategii od tych najbardziej obronnych i zachowawczych, do najbardziej agresywnych i szybkich zagrań. Najbardziej radykalne strategie (przejawiające się na przykład w bardzo szybkim ataku na bazę wroga wszystkimi dostępnymi jednostkami), zostały zweryfikowane i odrzucone w procesie populacyjnym. Taka rywalizacja pozwoliła wyłonić kilka najlepiej radzących sobie w lidze modeli. To właśnie je postawiono w bezpośrednim starciu z dwoma profesjonalnymi graczami *StarCraft*-a: Grzegorz *MaNa* Komincz oraz Dario *TLO* Wünsch. Oba mecze zakończyły się zwycięstwem 5-0 dla *AlphaStar*.

Opisane powyżej przykłady to bardzo medialne i popularne projekty, jednak implementacja nauczania przez wzmacniania nie ogranicza się jedynie do gier. Swoje zastosowanie znalazło między innymi w dziedzinach, takich jak: energetyka, transport, finanse czy medycyna [19].

2.2.2 Imitation learning

Nauka jedynie na zasadzie *prób i błędów*, może nie zawsze być efektywna. W niektórych problemach, łatwiej jest zademonstrować pewien sposób zachowania niż ręcznie zaprogramować odpowiadający temu model lub funkcję nagrody. W takich przypadkach można zastosować podejście *imitation learning*, w którym agent szuka sposób użycia zestawu treningowego dostarczonego przez nauczyciela w celu poznania i uogólnienia zasad jego działania [20]. Metoda to przypomina tą znaną z klasyfikatorów wykorzystywanych na przykład w detekcji obrazu [9].

Takie podejście niejednokrotnie pozwala na uzyskanie wysokiej efektywności nauki. Nie jest to jednak metoda uniwersalna. Dla niektórych problemów człowiek nie jest w stanie wykonać sprawnie danego zadania i dostarczenie odpowiednich przykładów może być bardzo kosztowne lub niemożliwe. Kolejną wadą tego podejścia może być fakt, że wzorując się na przykładach model może przenieść na swoją politykę również ludzkie błędy, wynikające m.in. ze złych nawyków czy braku umiejętności. Chcąc opanować rozwiązanie problemu w wysokim stopniu, takie podejście niejednokrotnie może okazać się niesatysfakcjonujące.

Występującym w tym podejściu problemem może być niedopasowanie dystrybucji danych [21]. Sytuacja ta może zajść, gdy agent z różnych względów znajdzie się w stanie, który jest niepokryty przez dane demonstracyjne. Wtedy algorytm musi określić działanie jakie w tej sytuacji powinien podjąć agent. Ograniczeniem tej metody jest to, że model może być tylko tak dobry, jak nauczyciel dostarczający przykłady.

2.3 Nauczanie maszynowe w autonomicznym przemieszczaniu

W literaturze można odnaleźć przykłady aplikacji metod uczenia maszyn dla zastosowań związanych z autonomicznym przemieszczaniem się. Większość z wykorzystywanych materiałów to prace powstałe w ciągu ostatnich kilku lat. Powyższe wskazuje, jak stosunkowo nowy jest to problem.

W artykule [22] Kaspar Sakmann prezentuje implementacje metody *imitation learning* z wykorzystaniem algorytmu *behavioral cloning* dla autonomicznego samochodu. Jako środowisko testowe autorowi posłużył gotowy projekt *Udacity Self-Driving Car Nanodegree* dostarczany do jednego z kursów dostępnego na stronie kursu *Udacity*.

Dane zbierane przez trzy kamery umieszczone na pójście przekazywane są do algorytmu. W procesie nauki autor wykorzystuje konwencyjną sieć neuronową (CNN), składającej się 4 warstw. Neuron reprezentujący wyjście odpowiedzialny jest za kąt skrętu. W rezultacie udało się uzyskać model, który pokonał tor treningowy i testowy bez kolizji.

W swojej pracy Sakmann zauważa pewien problem związany z symulacją ruchu samochodu. Podczas jazdy przez większość czasu kąt skrętu kół przednich jest bliski zeru. Jednak to momenty, kiedy samochód skręca są najważniejsze i stanowią największe zagrożenia kolizji. Paradoksalnie w zebranych danych treningowych informacji o takich sytuacjach jest najmniej. Rozwiązaniem przedstawionym przez autora jest syntetyczne wytworzenie takich sytuacji, poprzez duplikację danych zebranych w newralgicznych częściach toru oraz umiejętną ich obróbkę poprzez m.in. odbicie obrazu.

Jerry Qu w swoim artykule [23] udowadnia strategię, że *reinforcement learning* może być wykorzystywany do implementacji samojezdnych samochodów. Qu prezentuje proste środowisko 2D, zaimplementowane z wykorzystaniem pakietu *Kivy* języka *Python*. Model sztucznej inteligencji po zakończonej nauce opanował poruszanie pojazdu do wskazywanego punktu końcowego, unikając przy tym dynamicznie tworzonych (poprzez rysowanie), przeszkód. Autor w swoich badaniach zastosował poniższy system nagród: -5 uderzenie w przeszkodę, -0,1 oddalenie się od celu, 0,1 zbliżenie się do celu. Wykorzystano model o *Deep Q-Learning* o architekturze składającej się z 30 ukrytych warstw.

W [24] autorzy wykorzystują jako środowisko testowe symulator *The Open Racing Car Simulator*, takie podejście zdecydowanie może przyspieszyć badania. Autorzy wykorzystywali dostarczone wraz z symulatorem sensory zwracające informacje na temat: kąta osi samochodu względem linii toru, dystansu między samochodem, a brzegiem toru, dystansu między samochodem, a linią toru, prędkość samochodu w 3 wymiarach. Funkcja nagrody została napisana w taki sposób, aby zachęcać agenta do szybkiego pokonywania trasy, biorąc pod uwagę jego prędkość oraz pozycję względem linii toru. W procesie nauki *reinforcement learning* autorzy wykorzystują algorytm *Deep Deterministic Policy Gradient*.

3 Koncepcja Realizacji

3.1 Wybór technologii

Analizując literaturę dotyczącą nauczania maszynowego dla problemu autonomicznego poruszaniu się (rozdział 2.3), można zauważyć dwie drogi realizacji środowiska testowego. Pierwszą z nich jest wykorzystanie gotowych programów symulacyjnych, jak to miało miejsce w [24], [22]. Podejście takie pozwala zaoszczędzić wiele czasu, niesie jednak ze sobą duże ograniczenia. Środowiska takie są mało elastyczne, często uniemożliwiając modyfikacje oraz tym samym narzucając np. sposób zbierania obserwacji czy wykonywania akcji przez agenta. Drugą możliwością jest samodzielna implementacja środowiska z wykorzystaniem różnego rodzaju bibliotek czy narzędzi. Takie podejście zaprezentowano w [23], jednak w tym przypadku środowisko było bardzo umowne oraz zbudowane w widoku 2D.

Aby przygotować środowisko mogące symulować zachowanie obiektów w przestrzeni 3D możliwe jest napisanie własnego silnika graficznego, żeby symulacja była wiarygodna konieczne jest implementacja silnika fizycznego, mogącego symulować poruszanie się pojazdu czy wykrycie zderzeń. Takie podejście jest bardzo czasochłonne. Dlatego często korzystniejszym podejściem jest wykorzystanie gotowych narzędzi stosowanych do tworzenia na przykład do gier czy symulacji, zapewniających API i wiele gotowych rozwiązań, które w znacznym stopniu mogą ułatwić przygotowanie odpowiedniego środowiska. Takie środowisko powinno ponadto umożliwić komunikację z bibliotekami wspierającymi uczenie maszynowe tj. *TensorFlow*.

Jednym z takich narzędzi jest *Gym* [25] udostępniane przez organizację *OpenAi*. Zostało ono opracowane z myślą o implementacji i testowaniu agentów sztucznej inteligencji z zastosowaniem nauczania maszynowego oraz z wykorzystaniem dowolnej biblioteki obliczeń numerycznych, takich jak *TensorFlow* lub *Theano*. Pozwala ono na implementację prostych środowisk testowych zarówno w widoku 2D i 3D.

W pracy zdecydowano się na wykorzystanie narzędzia *Unity*, które pozwala na wygodną implementację zaawansowanych środowisk 3D. Dzięki wtyczce *ML-Agents*, pozwala na implementację nauczania maszynowego z wykorzystaniem biblioteki *Tensorflow*. Na wybór narzędzia istotny wpływ miała znajomość tego narzędzia wynikająca z doświadczenia w posługiwaniu się nim przez autora pracy.

Wykorzystane narzędzia i biblioteki zostały opisane poniżej.

3.1.1 Unity

Unity jest popularnym środowiskiem stosowanym do tworzenia gier oraz materiałów interaktywnych. Swoją popularność *Unity* zawdzięcza między innymi swojej wszechstronności. Stanowi zbiór funkcjonalności, takich jak silnik fizyczny i graficzny, zawiera również czytelny edytor. Zastosowanie narzędzia nie ogranicza się do tworzenia gier komputerowych. W ostatnich latach rozwijane są nowe funkcjonalności pozwalające wykorzystać *Unity* w celu tworzenia animacji oraz symulacji. Niewątpliwą zaletą *Unity* jest fakt, że program oferowany jest w formie darmowej, dla firmy oraz użytkowników prywatnych o zarobkach rocznych, nieprzekraczających 100 000 dolarów amerykańskich [26].

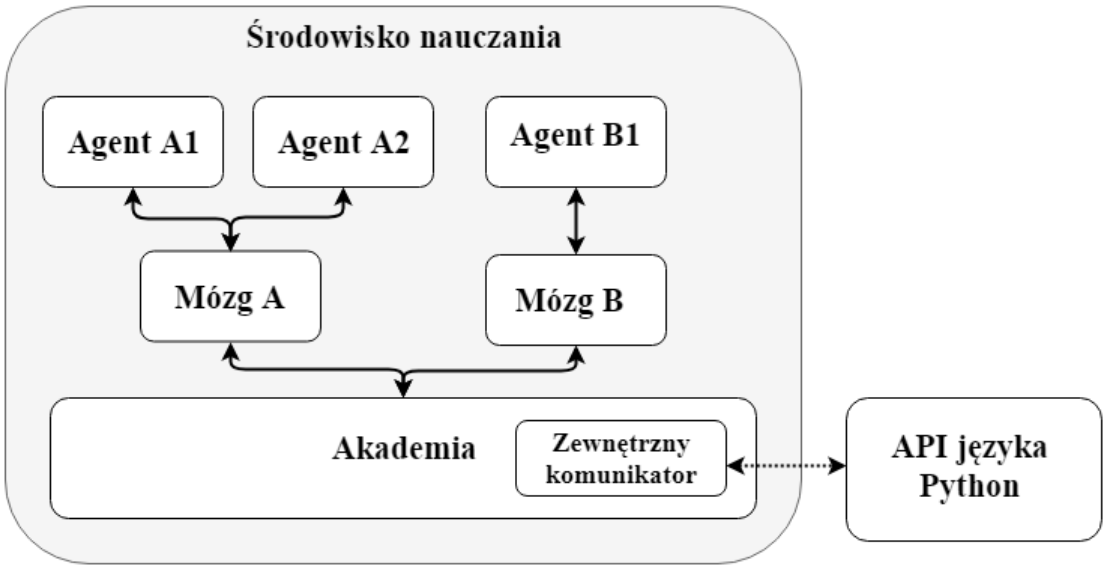
3.1.2 The Unity Machine Learning Agents Toolkit

Jest to wtyczka, która pozwala wykorzystać gry i symulacje *Unity* jako środowisko treningu agentów sztucznej inteligencji. Pozwala na nauczenie maszynowe metodami: *reinforcement learning, imitation learning, neuroevolution* [9]. *ML-Agents* zostało opublikowane na platformie *Github* w październiku 2017, od tego czasu jest rozwijane i aktualizowane. W trakcie realizacji projektu znajdowało się wciąż w wersji testowej (*beta) v0.81*.

Środowisko nauki składa się z trzech głównych komponentów [9]:

- Środowisko nauczania – zawiera scenę *Unity* wraz z wszystkimi jej obiektami. Wyróżnić w nim można kluczowe elementy:
  - *Akademia* (*Academy*): zarządza środowiskiem i organizuje proces nauczania
  - *Mózg* (*Brain*): jednostka decyzyjna, określa politykę działań a*genta* i decyduje jakie działania powinien podjąć, dla każdej jego instancji
  - *Agent*: zbiera obserwacje otoczenia, które przekazuje do mózgu i wykonuje akcje, na podstawie informacji uzyskanych z jednostki mózgu
- API języka *Python* – zawiera algorytmy nauczania maszynowego używane do treningu. Uruchamiane jest ono jako zewnętrzny program poza edytorem *Unity*.
- Zewnętrzny komunikator – łączy środowiska nauczania *Unity* z API języka *Python*.

Warto zaznaczyć, że środowisko treningu może składać się z kilku jednostek mózgu oraz agentów (co zobrazowano na rys.3.1). Takie działanie może prowadzić do zwielokrotnienia efektywności czasowej nauki, zwiększając przy tym narzut obliczeniowy.



Rysunek 3.1Wizualizacja komponentów środowiska ML-Agents (źródło własne)

Występują 3 typy mózgu agenta:

- *Learning* – wykorzystywany w procesie uczenia (kontrola nad mózgiem przekazywana jest do algorytmów znajdujących się w zewnętrznym API języka *Python*) lub gdy chcemy użyć wygenerowanego już modelu
- *Player* – agent jest sterowany przez użytkownika poprzez sygnały urządzeń wejściowych (np. klawiatury), każdej akcji agenta należy przypisać odpowiedni sygnał wejściowy (np. klawisz klawiatury), stosowany do testowania i debugowania środowiska testowego oraz przez nauczyciela w celu demonstracji wykonania zadania
- *Heuristic* – umożliwia zaprogramowanie za pomocą kodu, decyzji wykonywanych przez agenta

3.1.3 Tensorflow

To otwartoźródłowa biblioteka, przeznaczona do obliczeń o wysokiej wydajność, zapewniając wsparcie dla uczenia maszynowego. Charakteryzuje się elastyczną architekturą, pozwalającą na wdrażanie na wielu platformach tj. komputery stacjonarne, urządzenia mobilne czy serwery [27]. Biblioteka została opracowana przez naukowców i inżynierów z zespołu *Google Brain* w ramach *AI Google*. Obecnie jest to zdecydowanie najbardziej popularna otwartoźródłowa biblioteka wykorzystywana w nauczaniu maszynowym i *deep learning*-u [28], używana przez przedsiębiorstwa takie jak *Google, AMD, NVIDIA, Uber, Coca-Cola, Twitter, Ebay* czy *Intel* [27].

Opisywana w 3.1.2 wtyczka *ML-Agents* bazuje na bibliotece *Tensorflow*. Po zakończonym treningu generowany jest model omawianej biblioteki jako plik w formacie *.nn,* który później może być wykorzystywany przez agenta do podejmowania decyzji w czasie rzeczywistym.

3.1.4 Tensorboard

To narzędzie przeznaczone dla użytkowników biblioteki *Tensorflow* pozwalające na wizualizacje danych zbieranych podczas procesu nauki [29]. Korzystając z *ML-Agents* po zakończonym treningu wraz z modelem *.nn* generowany jest plik *.summaries*, który pozwala wykorzystywany jest przez aplikację *Tensorboard* do wyświetlenia statystyk treningu, tj. rozkład łącznej nagrody dla kolejnych kroków treningu.

3.2 Wybór algorytmów

Dla obu testowanych metod zostały wybrane algorytmy. Na decyzję wpływ miała popularność algorytmów w analizowanej literaturze oraz kompatybilność z wtyczką *ML-Agents*.

3.2.1 Proximal Policy Optimization

W procesie nauki *reinforcement learning* wykorzystany zostanie algorytm *Proximal Policy Optimization* (PPO). Jest to algorytm zaprezentowany w 2017 roku przez naukowców z organizacji *OpenAI*. Należy do grupy metod *policy gradient* (gradient polityki), która stanowi fundament przełomowych osiągnięć w wykorzystaniu sztucznych sieci neuronowych w implementacjach agentów sztucznej inteligencji (omawianych częściowo w podrozdziale 2.2.1).

PPO stanowi modyfikację metody *Trust Region Policy Optimization* (TRPO). Po pierwowzorze algorytm przejął wiele zalet, w tym wydajność, jest przy tym łatwiejszy w implementacji, prezentując bardziej ogólne podejście oraz stosując wydajniejsze metody próbkowania. Założeniami twórców PPO było osiągniecie balansu między łatwością implementacji, złożonością próbkowania i możliwościami dostrajania algorytmu [17].

Chcąc zrozumieć działanie algorytmu, konieczna jest znajomość ogólnych podstaw działania metod *policy gradient*, do których należy *Proximal Policy Optimization*. *Gradient policy* to specjalny przypadek bardziej ogólnego *estymatora gradientu funkcji*. Podejście to stosowane jest w strategii nauczania przez wzmacnianie. Celem działania jest wymodelowanie optymalnej polityki, maksymalizującej wartość sygnału nagrody [30]. Algorytmy należące do tej grupy to metody bardzo ogólne. Algorytm może być zastosowany dla zróżnicowanych problemów jedynie po odpowiednim doborze parametrów.

Dla problemu dana jest pewna dystrybucja, która umożliwia próbkowanie. Dla próbki możemy ocenić funkcję wyniku, która pobiera próbkę i zwraca wynik o wartościach skalarnych. Informuje to, jak powinno się przesunąć rozkład, aby próbki osiągnęły wyższe wyniki [31]. Problemem jest odpowiedni dobór wielkości kroku. Zbyt mała aktualizacja poskutkuje znikomymi postępami w nauce, zbyt duża może doprowadzić do zaszumienia sygnału lub do dużych spadków w wydajności. Drugim problemem jest niska efektywności próbkowania, która powoduje, że nawet proste zadania mogą wymagać milionów kroków treningu [31]. Metody *policy gradient* wykorzystują podejście *online policy*. W przeciwieństwie do *offline policy*, które stosowane jest np. w rozwiązaniach *Q-learning*, gdzie algorytm może być uczony z zebranych wcześniej danych, metody wykorzystujące *online policy* nie posiadają bufora powtórnego (*replay buffer*), uczą się bezpośrednio na podstawie zdarzeń, które agent napotka w środowisku testowym.

Omawiany algorytm dzieli wiele cech wspólnych z innymi metodami policy gradient, takimi jak *Vanilla policy gradient* czy TRPO. Cechą wyróżniającą PPO jest funkcja celu wykorzystująca *clipped surrogate objective*, ograniczający wielkość aktualizacji *policy gradient* [17].

Algorytm w momencie powstania posiadał wiele alternatyw, inne rozwiązania aproksymacji z wykorzystaniem sztucznych sieci neuronowej strategią uczenia ze wzmocnieniem, to m.in *Actor Critic with Experience Replay* (ACER), *Q-learning*, *Vanilla Policy Gradient* czy wspomniany już *Trust Region Policy Optimization* (TRPO) [17]. Znalazł on jednak liczne zastosowanie dzięki wysokiej wydajności oraz relatywnie prostej implementacji. Przeprowadzone w [17] badania opierające się na problemie nauki gier konsoli *Atari* na podstawie analizy pikseli ekranu oraz zadaniu ciągłej kontroli (*continuous control*) wykazały wyższą wydajność PPO, względem konkurencyjnych algorytmów.

Implementację algorytmu z wykorzystaniem języka *Python* i biblioteki *TensorFlow*, znaleźć można w repozytorium publicznym *OpenAI*. PPO jest domyślnym algorytmem w wykorzystywanym w badaniach środowisku *Unity ML-Agents* [9].

3.2.2 Behavioral Cloning

W czasie przeprowadzania nauczania *imitation learning* wykorzystany zostanie algorytm *Behavioral Cloning*. Algorytm ten stanowi jedną z głównych metod tego podejścia. Agent jako dane treningowe odbiera zarówno obserwacje, jak i akcje nauczyciela, a następnie używa klasyfikatora by odtworzyć działania nauczyciela. Działa podobnie do algorytmów nauczania nadzorowanego wykorzystywanych w klasyfikatorach służących na przykład do analizy

zdjęć [9], z tym że przypisuje (mapuje) on akcje do obserwacji. Wykorzystuje sztuczne sieci neuronowe. Jego zaletą jest możliwość natychmiastowego naśladowania nauczyciela, bez konieczności interakcji z otoczeniem [32]. Algorytm znalazł liczne zastosowania m.in. w omawianej w rozdziale 2.3 implantacji samojezdnego samochodu [22].

Od wersji *v0.6 ML-Agents* umożliwienie szkolenie *imitation learning* w dwóch trybach [9]. *Offline training* pozwala na bazowanie na nagranych wcześniej przykładach, zapisanych w pliku demonstracyjnym. Przed przystąpieniem do treningu należy z wykorzystaniem agenta podłączonego do mózgu w trybie *player* (sterowanego przez gracza), nagrać przykłady rozwiązań problemu. Tryb *online training* pozwala na naukę obserwując zachowania nauczyciela w czasie rzeczywistym. Wymaga on w czasie treningu obecności na scenie przynajmniej dwóch agentów: sterowanego przez gracza *nauczyciela* oraz (przynajmniej jednego) agenta - *ucznia*. Tryb ten lepiej sprawdza się w przypadku prostych problemów, w których wystarczają kilka przykładów demonstrujących. Zaletą trybu *offline* nad *online training* jest to, że w przypadku konieczności powtórzenia treningu, nie ma potrzeby ponownej demonstracji przez gracza, gdyż pliki demonstracyjne mogą być używane wielokrotnie. Z tych względów zdecydowano wykorzystaniu trybu *offline training*.

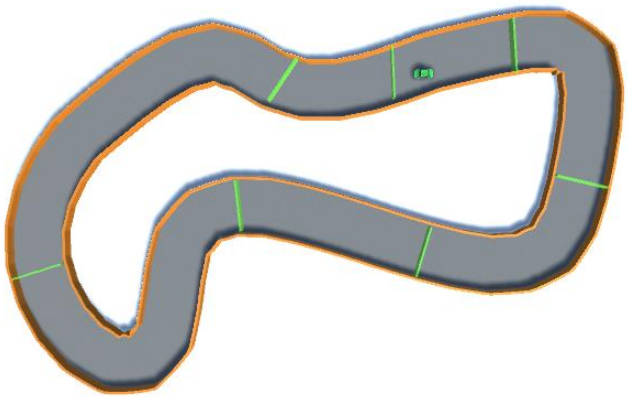
4 Implementacja środowiska testowego

Problemem symulowanym w przestrzeni treningowej jest autonomiczne przemieszczanie w przestrzeni 3D. Z wykorzystaniem środowiska *Unity* zostanie zaimplementowany model poruszającego się pojazdu-agenta. Ze względu na ograniczone zasoby mocy obliczeniowej komputera oraz zasoby czasowe, przestrzeń treningowa zostanie przedstawiona w sposób uproszczony oraz w pewnym stopniu umowny. Pojazd przypominać będzie samochód autonomiczny, mając możliwość zbierania informacji o otoczeniu. Agent poruszać będzie się po zamkniętym torze. Nitka toru powinna być zmienna, tak aby zapewnić zmienność środowiska podczas procesu treningu, jak również testowania modelu.

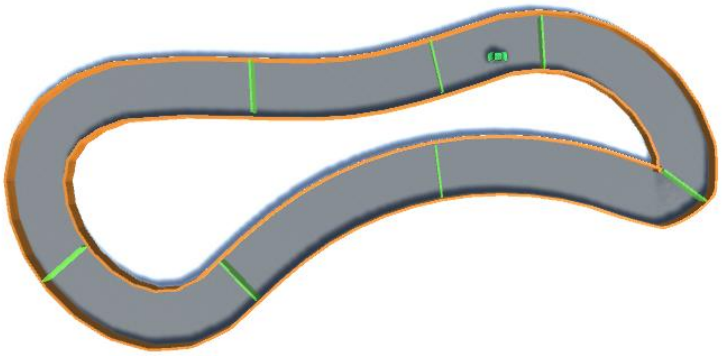
4.1 Implementacja przestrzeni treningu

Przestrzeń treningu stanowią obiekty, z którymi w interakcję może wchodzić agent. Wyróżnić należy elementy, takie jak podłoże toru (droga), po której agent może się poruszać, ściany ograniczające trasę, z którymi zderzenia agent powinien unikać. Na torze rozmieszczone są również punkty kontrolne, które służą do przedstawienia agentowi odpowiedniego kierunku jazdy. Wykorzystywane są one również do obliczania sygnału zwrotnego nagrody przyznawanej agentowi w czasie treningu. Ponadto po długotrwałym kontakcie z przeszkodą, agent może zostać cofnięty do ostatniego z osiągniętych punktów kontrolnych, a następnie kontynuować dalszą jazdę. Zabieg ten ma na celu zwiększenie efektywności treningu. W ten sposób ograniczony zostanie czas, w którym agent próbuje sforsować ścianę poruszając się wprost na nią.

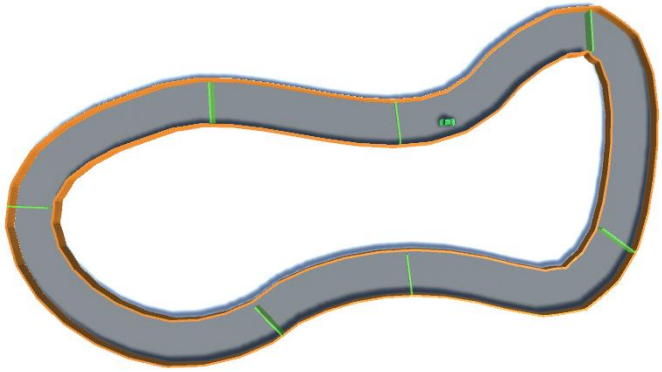
Trasa przejazdu posiada 3 konfiguracje, które charakteryzują się odmienną nitką toru, a co za tym idzie innym rozkładem punktów kontrolnych.



Rysunek 4.1 Rzut toru 1



Rysunek 4.2 Rzut toru 2



Rysunek 4.3 Rzut toru 3

Przy wykorzystywaniu wtyczki *ML-Agents*, konieczne jest posiadanie na scenie obiektu *akademii* odpowiedzialnego za organizację procesu treningu. Rola tego obiektu została omówiona w podrozdziale 3.1.2. Jest to pusty obiekt *Unity*, posiadający komponent, którego klasa dziedziczy po klasie *ML-Agents SDK: Academy*. W tym przypadku jest to klasa *CarAcademy*.

4.2 Implementacja agenta

Skrypt agenta, implementujący klasę *TrackCarAgent* dziedziczącą po klasie *ML-Agents SDK: Agent*, odpowiedzialny jest m.in. za zbieranie obserwacji czy interpretacji wektora akcji przekazywanego przez mózg. Implementuje również algorytm przyznawania nagrody, określając, kiedy i z jakimi wartościami parametru ma być wywoływana metoda *AddReward*, zdefiniowana w klasie nadrzędnej *Agent.*

Tabela 4.1 Metody pochodne klasy Agent nadpisywane w klasie TrackCarAgent

Nazwa funkcji	Opis
InitializeAgent	Wywoływana w czasie inicjalizacji agenta zbierając referencję do pozostałych komponentów agenta
CollectObservations	Implementuje sposób zbierania obserwacji przez agenta
AgentAction	Implantuje sposób wykonywania akcji przez agenta
AgentReset	Wywoływana w celu przywrócenia wartości domyślnych agenta, tj. pozycja w celu przygotowania agenta do rozpoczęcia kolejnego epizodu nauki

Ponadto klasa zawiera szereg metod pomocniczych wywoływanych np. w czasie zderzenia z przeszkodami, czy też w czasie pokonywania punktów kontrolnych. Metody te mają za zadanie m.in. zapisywanie czasu pokonania przez pojazd całego okrążenia, jak również implementują funkcjonalność przywracania pojazdu do punktu kontrolnego w przypadku długotrwałego kontaktu z przeszkodą.

4.2.1 Gromadzenie obserwacji

Zbieranie obserwacji to jeden z najważniejszych elementów systemów autonomicznego przemieszczania się. Sposób jego implementacji wpływa w znaczący sposób na efektywność procesu nauczania W przypadku jego niepoprawnego działania, wygenerowany model nie będzie mógł działać poprawnie.

Jak opisano w rozdziale 2.1. w świecie rzeczywistym stosuje się różnego rodzaju sensory i czujniki, tj. radary czy kamery. Dużym problemem jest rozpoznanie obiektów, takich jak przeszkody, granica drogi, piesi czy znaki drogowe. Wykorzystywane są zawansowane algorytmy analizy danych i klasyfikatory. W przypadku tej symulacji komputerowej, ze względu na ograniczone zasoby zdecydowano się na uproszczenie, które umożliwia środowisko *Unity*. Obiekty sceny, zostały oznaczone specjalnymi znacznikami (tagami). Wykorzystano dostarczony razem z wtyczką *ML-Agents* komponent, *RayPerception3D*, który umożliwia detekcję obiektów oznaczonych tagami, wykorzystując do tego symulowane promienie lasera, które docierając do przeszkód nie tylko zwracają informację na temat odległości od napotkanego obiektu, ale również na temat znacznika obiektu. Klasa *RayPerception3D* dostarcza metodę *Perceive*, której parametry opisuje tablica 4.2 zwraca listę liczb rzeczywistych zawierającą informacje o wykrytych obiektach oraz odległości od nich. Lista jest przekazywana do mózgu, za pomocą dostarczanej przez *ML-Agents SDK* funkcji *AddVectorObs*.

Zastosowane podejście do zbierania obserwacji ogranicza zaszumienie danych, jak również pozwala na pominięcie kosztownego procesu klasyfikacji obiektów. Proces ten jest w stanie znacznie przyśpieszyć proces nauczania w stosunku do treningu z wykorzystaniem obserwacji wizualnych (dane zbierane za pomocą wirtualnej kamery).

Tabela 4.2 Parametry metody Perceive, klasyRayPerception

Parametr	Typ	Opis
rayDistance	Liczba rzeczywista	Długość promienia lasera
rayAngles	Tablica liczb rzeczywistych	Kąty w jakich mają padać lasery, wokół agenta
detectableObjects	Tablica napisów	Lista znaczników jakie mają być rozpoznawane
startOffset	Liczba rzeczywista	Początkowe przesunięcie wysokości promienia od środka agenta
endOffset	Liczba rzeczywista	Końcowe przesunięcie wysokości promienia od środka agenta

Rozmiar tablicy *rayAngles* decyduje o tym, jak dużo wiązek lasera będzie prowadzonych w każdym kroku treningu. Wartość tę przyjęto na 10. Liczba zmiennych przekazywanych przez *AddVectorObs* na każdym kroku treningu decyduje o wielkości wektora obserwacji mózgu. W opisanej powyżej konfiguracji ma on rozmiar 50.

4.2.2 Wykonywanie akcji

Akcje to działania agenta wywoływane na podstawie poleceń mózgu. Konieczne jest zdefiniowanie przestrzeni akcji agenta. W *ML-Agents* przestrzeń wektora akcji może być zdefiniowany jako ciągła (*continuous*) lub dyskretna (*discrete*). W postaci ciągłej parametr akcji jest tablicą sygnałów o rozmiarze *Vector Action Space Size*. W postaci dyskretnej parametr akcji to tablica liczb całkowitych. Każda z nich odwołuje się do indeksu listy lub tablicy komend. Akcję definiuje się nadpisując metodę *AgentAction(),* przyjmującą jako parametr tablicę akcji przekazywaną przez mózg.

Ze względu to, że agent ma prezentować uproszony model samochodu, naturalną jest decyzja o wyborze ciągłej przestrzeni akcji. W tym przypadku konieczne było zdefiniowanie przestrzeni wielkości 2.

Tabela 4.3 Opis wektora akcji agenta

Indeks	Opis	Oś wejścia trybie *Player*
0	Przyspieszenie	Wertykalna
1	Rotacja	Horyzontalna

W celu symulacji ruchu pojazdu wykorzystano narzędzie *Vehicle Tools*, dostarczaną za darmo przez *Unity Technologies* w *Asset Store*. Umożliwiła ona implementację prostego modelu samochodu, posiadającego fizykę kół oraz napędu, implementowanego przez komponent *Wheel Drive*. Został on zmodyfikowany w taki sposób, aby kontrola nad pojazdem, mogła odbywać się za pomocą interfejsu spoza komponentu. W tym celu,

zaprogramowano funkcję publiczną *Drive*, przyjmującą jako parametry dwie liczby rzeczywiste odpowiedzialne za sterowanie kątem kół oraz przyspieszeniem samochodu. Dzięki czemu posiadając referencję do komponentu *Wheel Drive*, kontrola nad pojazdem mogła zostać przekazana mózgowi za pośrednictwem skryptu *TracCarAgent*, gdzie funkcja *Drive* wywoływana jest metodą *AgentAction*.

4.2.3 Strategia przyznawania nagrody

Określenie polityki przyznawania nagród to kluczowy aspekt nauki metodą *reinforcement learning*. Sygnały nagrody nie są natomiast wykorzystywane w metodzie *imitation learning*. Metodyka nagradzania wpływa bezpośrednio na skuteczność treningu oraz rezultaty osiągane przez agenta. Nawet drobne zmiany w przyznawaniu nagród mogą sprawić, że zachowanie agenta po treningu zmieni się diametralnie, gdyż to właśnie sygnał nagrody kształtuje końcowe zachowanie agenta. Ze względu na swoją wagę w procesie *reinforcement learning*, kształtowanie nagrody (*reward shaping*) stanowi bardzo istotny problem, co zostało opisane w podrozdziale 2.2.1.

Problematyka przyznawania nagród obejmuje ponadto omawiany problem rzadkiej nagrody (*sparse reward*), wynikający ze złożoności zagadnienia autonomicznego poruszania się. Celem samojezdnego samochodu jest dotarcie z punktu startowego do punktu końcowego (pokonanie okrążenia), w jak najkrótszym czasie, unikając przy tym zderzenia z przeszkodami. Jednak przyznanie nagrody dopiero w momencie osiągnięcia przez agenta celu końcowego spowoduje, że nagroda przyznawana będzie zbyt rzadko. Co skutkowało będzie nieefektywnością procesu treningu lub na jego kompletnym niepowodzeniem. Istnieje bardzo małe prawdopodobieństwo, aby w sposób losowy wygenerować sekwencję akcji prowadzącą do pokonania całej trasy. Powoduje to konieczność zastosowania odpowiedniego algorytmu przyznawania nagrody, który będzie nie tylko nagradzała agenta za dotarcie do punktu końcowego, ale również nakierowywała go w trakcie pokonywania trasy.

W celu uniknięcia wystąpienia skutków rzadkiej nagrody, zdecydowano się na rozbicie zadania na mniejsze podzadania, poprzez rozmieszczenie punktów kontrolnych na trasie przejazdów. Dzięki takiemu podejściu, nagroda może zostać przyznawana w trakcie przejazdu, w momencie osiągnięcia kolejnego punktu. Konieczne jest karanie agenta, za zderzenie z przeszkodami, nie tylko w momencie uderzenia, ale również w trakcie kontaktu z nią. Ponadto, w celu osiągnięcia modelu, który będzie pokonywał trasę w sposób efektywny czasowo, konieczne jest promowanie przejazdów, w którym agent pokonał całe okrążenie w jak najkrótszym czasie. W celu osiągnięcia takiego rezultatu możliwe jest opisane w [9] przyznawanie niewielkiej kary, w każdym kroku treningu.

W przypadku implementacji z użyciem *ML-Agents*. Nagrody przyznawane są poprzez metodę *AddReward*() lub *SetReward()*, przyjmują wartości typu *float* i są zdefiniowane w klasie *Agent*.

Tabela 4.4 Polityka przyznawania nagrody

Zdarzenie	Nagroda
Dotarcie do punktu kontrolnego	1
Rozpoczęcie kolizji z przeszkodą	-0,5
Kontakt z przeszkodą	-0,1
Każdy krok treningu (kara pasywna)	-0,001

5 Konfiguracja środowiska

5.1 Konfiguracja i uruchamianie procesu nauczania

Przed uruchomieniem treningu w edytorze *Unity*, należy pamiętać, aby przypisać agentowi obiekt mózgu (w trybie *learning brain*), następnie ten sam obiekt przypisać jako parametr skryptu akademii zaznaczając przy nim opcję *Control.* W inspektorze tego skryptu można ustawić również dodatkowe parametry treningu, tj. skalę czasu (*time scale),* która pozwala przyspieszyć proces symulacji.

W konsolowym programie *Anaconda* należy aktywować środowisko komendą *activate ml-agents,* a następnie wprowadzić polecenie, rozpoczynające proces nauki:

```
mlagents-learn config\<scieżka_pliku_konfiguracyjnego> --train --run-id=<identyfikator>
```

Polecenie to posiada również inne, dodatkowe parametry, które opisano w dokumentacji wtyczki [9]. Po chwili ukazać się powinna informacja o aktywacji treningu i konieczności uruchomienia symulacji w edytorze. Po wykonaniu tej czynności, jeżeli nie wykryte zostaną żadne błędy, proces nauki powinien się rozpocząć.

5.2 Parametry treningu

Przed przestąpieniem do treningu konieczne jest odpowiednie uzupełnienie plików konfiguracyjnych. Domyślnie ich lokalizacją jest folder *config* znajdującym się w głównym folderze repozytorium *ML-Agents*. Dla procesu *reinforcement learning* domyślny plik konfiguracyjny to *trainer_config.yaml,* a dla *imitation learning* jest to *offline_bc_config.yaml* oraz *online_bc_config.yaml*. Szczegółowe instrukcje dotyczące instalacji i obsługi można znaleźć w dokumentacji wtyczki *ML-Agents* [9]. Zarówno PPO, jak i BC, to bardzo uniwersalne algorytmy, i aby przystosować je do danego problemu należy przeprowadzić proces parametryzacji. W czasie dostrajania parametrów w szczególności należy uwzględnić złożoności problemu, długość epizodu treningu, wielkości wektora obserwacji oraz przestrzeni akcji. Poniżej przedstawiono wybrane parametry treningu, informacje o wszystkich dostępnych parametrach można znaleźć w [9].

- *learning_rate* - wielkość aktualizacji metody *gradient descent*, którą algorytm wykonuje w celu poszukiwania kolejnego rozwiązania, przekłada się na to jak szybko sieć neuronowa porzuca starą politykę na rzecz nowej
- *batch_size* - liczba doświadczeń wykorzystywanych w jednej iteracji aktualizacji metody *gradient descent*
- *max_steps* - maksymalna liczba kroków symulacji podczas sesji treningu, po przekroczeniu jej trening jest zatrzymywany i automatycznie generowany jest model, im bardziej skomplikowany jest problem, tym większa powinna być wartość tego parametru
- *num_layers* - liczba ukrytych warstw w sztucznej sieci neuronowej

5.3 Śledzenie i analiza procesu treningu

W procesie badawczym można wyróżnisz dwa etapy:

- **Proces treningu** – jest to czasochłonny proces mający na celu wygenerowanie modelu, z wykorzystaniem zadanych parametrów
- **Ocena modelu** – uruchomienie agenta z wykorzystywaniem wygenerowanego w procesie treningu modelu, w celu oceny sprawność modelu a tym samym efektywność przeprowadzonego nauczania

W celu śledzenia rozkładu metryk procesu treningu dla kolejnych kroków treningu, zostanie wykorzystany opisany w podrozdziale 3.1.4 program *TensorBoard*. Metryką, której rozkład śledzony będzie w szczególności jest nagroda łączna (*cumulative reward)*. Jest to średnia, łączna nagroda za epizod treningu w stosunku do wszystkich agentów, dla których jest przeprowadzane nauczanie. Kryterium to pokazuje w jakim stopniu agentowi udaje się maksymalizować wartość nagrody. Wartość nagrody łącznej powinna wzrosnąć podczas udanej sesji treningowej [9]. Sygnał nagrody jest niezbędne w procesie *reinforcement learning*. W metodzie *imitation learning*, nagroda nie wpływa bezpośrednio na kształtowanie modelu, jednak metryka ta jest dalej przydatna do oceny procesu nauki.

W celu oceny modelu w klasie *CarAcademy*, została zaimplementowana metoda *GenerateRaport*, która po zakończonej sesji generuje plik tekstowy wraz z zebranymi metrykami. Metoda zapisuje metryki odpowiadające kryteriom porównawczym przedstawionym w rozdziale 6.1 oraz uśrednia dane dla wszystkich agentów znajdujących się na scenie.

6 Badania

6.1 Kryteria porównawcze

Podczas badań wykorzystywane w celu oceny będą poniższe kryteria :

- **Ukończone okrążenia** – uśredniona liczba ukończonych okrążeń, jaką agent był w stanie pokonać w czasie testowym
- **Uderzenia w przeszkody** - uśredniona liczba zderzeń agenta z przeszkodami w czasie testowym
- **Nagroda łączna modelu** – uśredniona wartość nagrody agenta dla jednego epizodu w czasie testowym
- **Czas okrążenia** – średni czas okrążenia pokonanego w czasie testowym
- **Czas treningu** – czas rzeczywisty jaki upłynął w czasie nauki modelu

6.2 Założenia

Na jednej scenie *Unity* umieszczone zostały 4 przestrzenie badawcze. Każda z przestrzeni składa się z jednego agenta oraz 3 torów jazdy. Takie podejście zapewni większą efektywność czasową treningu, gdyż w tym samym czasie rzeczywistym agenci są w stanie zebrać 4 razy więcej doświadczeń. Tory 1 i 2 są przełączane w trakcie treningu w sposób losowy, tak aby zapewnić zmienność środowiska treningu. Tor 3 wykorzystany jedynie w części badań.

Tak jak opisano w podrozdziale 5.3, w procesie badawczym możemy wyróżnić dwa etapy: proces treningu oraz ocena modelu. W procesie treningu skala czasu symulacji zostanie ustawiona na wartość 100, dzięki czemu efektywny czas treningu zostanie zwielokrotniony. W czasie przejazdów mających na celu ocenę modelu skala ta przyjmie wartość 1. W tej fazie badawczej środowisko testowe zostanie uruchomione na czas 180 sekund. W trakcie procesu treningu czas nauki będzie uzależniony od konkretnego badania i będzie przedstawiony w sekcji *konfiguracja treningu*. Wyniki zostaną zaokrąglone do dwóch miejsc po przecinku.

W czasie badania próby będą często powtarzane, w takim przypadku zostanie wprowadzone pojęcie przebiegu, odwołującego się do numeru modelu. Ponadto w czasie opisu badań wykorzystane zostaną skrótowce- RL (*reinforcement learning*), IL (*imitation learning*).

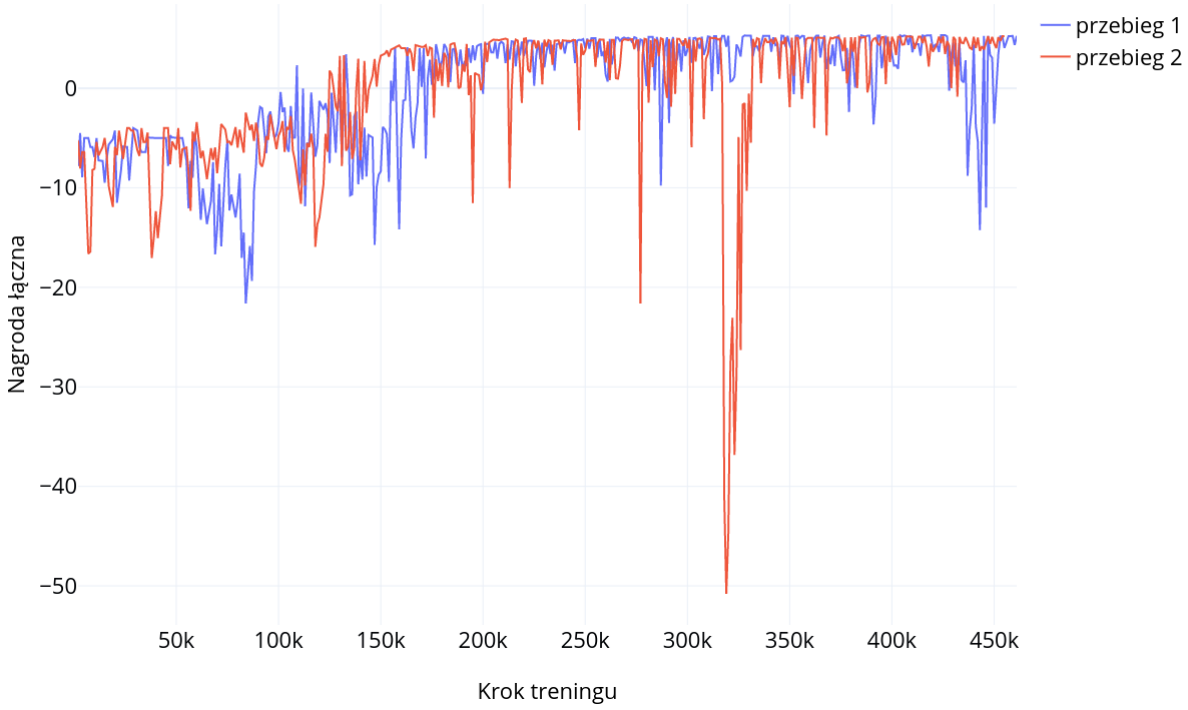
6.3 Porównanie początkowe

Przed przystąpieniem do procesu parametryzacji zdecydowano się wykonać początkowe porównanie, mające na celu test środowiska, jak również znalezienie pierwszych zależności, mogących wpłynąć na późniejsze etapy badań. Algorytmy zostaną uruchomione z domyślnymi wartościami parametrów.

Konfiguracja treningu:

- Ograniczenie czasowe: 60 minut
- Interwał decyzji: 5

a) Reinforcement learning

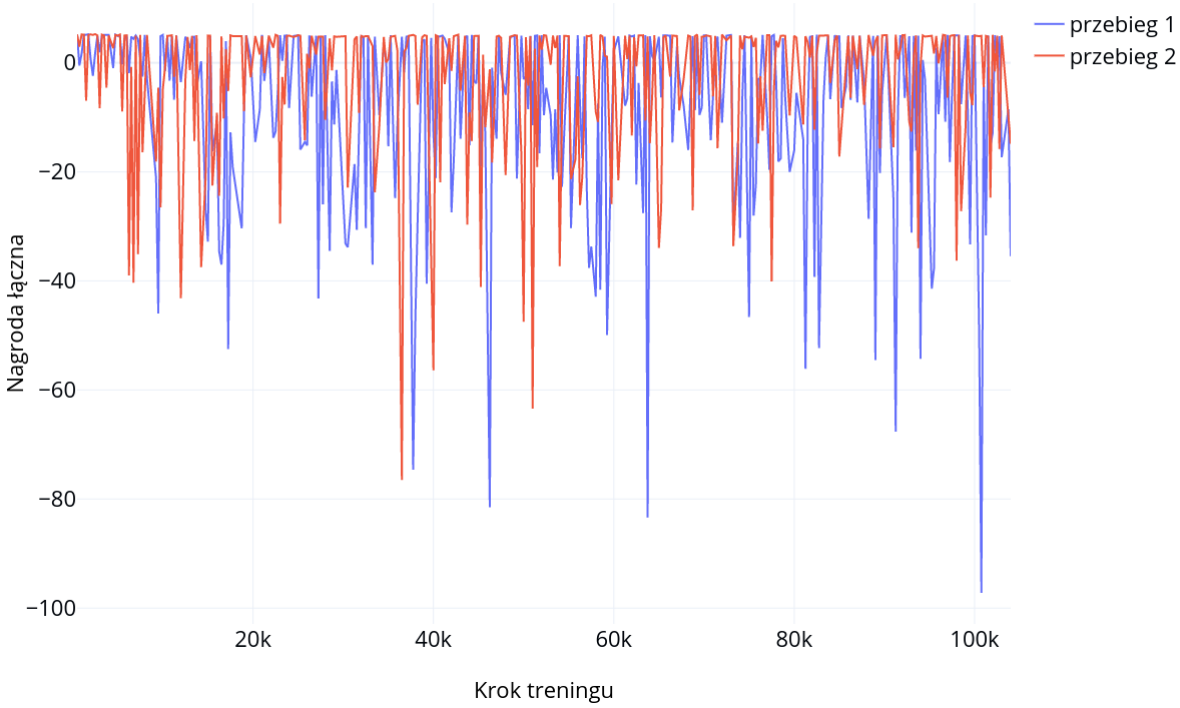


Wykres 6.1 Rozkład nagrody łącznej dla kolejnych kroków nauki dla przebiegów porównania początkowego metodą RL

Tabela 6.1 Wyniki oceny modelu dla przebiegów porównania początkowego metodą RL

Przebieg	**1**	**2**
Ukończone okrążenia	8,75	10
Liczba kolizji	10,75	6,25
Nagroda łączna	1,07	2,71
Czas okrążenia	17,93	18,22

b) Imitation learning



Wykres 6.2 Rozkład nagrody łącznej dla kolejnych kroków nauki dla przebiegów porównania początkowego metodą IL

Tabela 6.2 Wyniki oceny modelu dla przebiegów porównania początkowego metodą IL

Przebieg	**1**	**2**
Ukończone okrążenia	1,75	4,75
Liczba kolizji	19,75	26,25
Nagroda łączna	-15,59	-12,91
Czas okrążenia	22,08	16,63

Próby te wykazały, że modele wygenerowane podczas treningu trwającego 60 minut, są w stanie pokonywać całe okrążenia toru. Jednak dla takiej samej konfiguracji i danych wejściowych, tak samo długo trwający trening może wygenerować modele różniące się pod względem efektywności. Dlatego podczas dalszych badań prezentowane wyniki będą wartościami uśrednionymi kilku przebiegów nauczania. Warto nadmienić, że w tym samym czasie (60 minut), algorytmy wykonały różną liczbę kroków. Algorytm PPO 458000 i 461000, natomiast dla algorytmu BC 105000 i 104000, średnia długość kroku algorytmu BC była zatem ponad 4 razy większa niż PPO. W trakcie treningu wielokrotnie dochodzi do aktualizacji polityki, które często powodowały spadek w uzyskiwanych wartościach nagrody. Śledząc przebieg wartości nagrody łącznej dla kolejnych kroków treningu, można wyróżnić mniej oraz bardziej radykalne zmiany polityki. Te drugie często powodowały gwałtowny

spadek wartości nagrody łącznej. Warto zaznaczyć, że dla RL można zauważyć stopniowy wzrost średniej uzyskanej nagrody dla kolejnych kroków treningu, natomiast dla IL, algorytm bardzo szybko uzyskuje wartości maksymalne, nie widać jednak wyraźnej poprawy średniej wartości nagrody wraz z kolejnymi krokami nauki. Taka niestabilność treningu może to być spowodowana błędnymi parametrami nauczania.

6.4 Parametryzacja

Przed przystąpieniem do bezpośredniego porównania metod konieczny jest odpowiedni dobór parametrów nauczania. Ze względu na ograniczenia czasowe i czasochłonność treningu nie zdecydowano się na parametryzację dla wszystkich dostępnych parametrów. Wyróżniono poniższe czynniki, dla których wpływ na efektywność modelu będzie zbadany:

- **Kształtowanie sygnału nagrody**

Sposób przyznawania nagrody został określony w rozdziale 4.2.3. Na tym etapie zbadany zostanie wpływ nagrody pasywnej na uzyskiwane przez model wyniki. Ponieważ sygnał nagrody wykorzystywany jest jedynie w metodzie RL, czynnik ten nie będzie testowany dla strategii IL.

- **Interwał podejmowania decyzji**

Nie dla każdego problemu konieczne jest podejmowanie decyzji, w każdym jego kroku. Parametr ten pozwala wpłynąć na częstotliwość podejmowania decyzji przez agenta, poprzez określenie liczby kroków między żądaniami decyzji.

- **Liczba ukrytych warstw w sztucznej sieci neuronowej** (parametr *num_layers*)

Liczba ukrytych warstw występujących po wejściu obserwacyjnym sieci neuronowej. Dla prostszych problemów niższa liczba warstw może zaowocować szybszym i bardziej efektywnym treningiem. Dla bardziej złożonych problemów może być konieczna większa liczba warstw [9].

- **Learning rate** (parametr *learning_rate)*

Odpowiada wielkości kroku (aktualizacji), metody *gradient descent*, który zostaje wykonany w celu poszukiwania kolejnego rozwiązania. Metryka przekłada się na to, jak szybko sieć neuronowa porzuca starą politykę na rzecz nowej. Według [9] wartość ta powinna być zmniejszona w przypadku, gdy trening jest niestabilny.

- **Batch size (**parametr *batch_size*)

Liczba doświadczeń wykorzystywanych w jednej iteracji aktualizacji metody *gradient descent.*

Dla podanych parametrów zostanie zbadany wpływ dla metod *reinforcement learning* oraz *imitation learning*. Na podstawie informacji znalezionych w dokumentacji [9] dla każdego z parametrów określono zakresy, a następnie wartości parametru, dla których przeprowadzone zostanie proces nauczania. Dla każdej z wartości, trening trwający 30 minut, zostanie powtórzony 3-krotnie. Następnie, dla każdego z uzyskanego modelu zostanie przeprowadzona ocena modelu. Dla każdej wartości parametru, wyliczona zostanie średnia na podstawie rezultatów 3 modeli. Końcowe wartości zostaną przedstawione w tabelach. Wybrane zostaną wartość parametru, dla której uśrednione rezultaty dla kryteriów porównawczych okażą się najkorzystniejsze.

6.4.1 Reinforcement learning

- **Kształtowanie sygnału nagrody**

Wpływ metody przyznawania nagrody na skuteczność procesu treningu opisano w podrozdziale 2.2.1. W 4.2.3 przedstawiono zaimplementowany sposób nagradzania i karania agenta. Celem tego punku badań jest przetestowanie, jaki wpływ na efektywność modelu ma przyznawanie niewielkiej nagrody negatywnej w każdym kroku treningu. Ten zabieg opisany w 4.2.3 ma na celu zmuszenie agenta do wykonywania zadania w jak najkrótszym czasie.

Tabela 6.3 Wyniki oceny modelu w przypadku nauki z oraz bez kary pasywnej dla metody RL

Kara pasywna	Tak	Nie
Ukończone okrążenia	11,37	6,5
Liczba kolizji	16	6
Nagroda łączna	2,39	3,75
Czas okrążenia	14,57	16,10

Uśrednione wyniki oceny modelu wykazują, że pomimo uzyskiwania wyższych wartości nagrody łącznej (co spowodowane jest modyfikacją metody przyznawania nagrody), modele wytrenowane z przyznawaniem nagrody pasywnej uzyskały lepsze czasy okrążenia. Wynikać to może z faktu, że taka polityka przyznawania nagrody, promowała szybsze ukończenie zadania. Rezultaty potwierdziły więc przewidywania postawione w 4.2.3.

- **Interwał podejmowanych decyzji**

Tabela 6.4 Wyniki oceny modelu dla zmiennej wartości interwału decyzji dla metody RL

Interwał decyzji	1	5	10	15	20
Ukończone okrążenia	0	1	8,75	11	9,75
Liczba kolizji	21,37	13,38	8,5	1,25	0,63
Nagroda łączna	-23,52	-6,84	4,39	6,62	6,75
Czas okrążenia	-	20,3	15,97	15,12	17,20

Dla wartości interwału 1 i 5 uzyskiwane wyniki były bardzo słabe. Dla wyższych wartości interwału, modele uzyskiwały znacznie korzystniejsze rezultaty. Może to wynikać z wyższej efektywności takiego treningu spowodowanej ograniczeniem nadmiarowych decyzji. Najbardziej korzystne wyniki uzyskano dla wartości interwału 15. Taki przyjęto dla kolejnych pomiarów.

- **Liczba ukrytych warstw w sztucznej sieci neuronowej**

Tabela 6.5 Wyniki oceny modelu dla zmiennej wartości parametru liczby ukrytych warstw sztucznej sieci neuronowej dla metody RL

Liczba ukrytych warstw	1	2	3	4
Ukończone okrążenia	10,63	11,25	10,58	10,16
Liczba kolizji	0,88	0	1,41	2,66
Nagroda łączna	4,77	5,04	3,80	3,08
Czas okrążenia	15,36	15,18	15,42	15,52

Dla liczby ukrytych warstw w sztucznej sieci neuronowej różnice w uzyskiwanych wynikach były mniejsze niż w przypadku wcześniej testowanych parametrów. Najbardziej korzystne wyniki uzyskano dla 2 ukrytych warstw sieci neuronowej.

- **Learning rate**

Tabela 6.6 Wyniki oceny modelu dla zmiennej wartości learning rate dla metody RL

Learning rate	1,00E-05	5,00E-05	1,00E-04	5,00E-04	1,00E-03
Ukończone okrążenia	0	0	5,88	11,13	10,63
Liczba kolizji	4,5	13,25	8	4,13	3,38
Nagroda łączna	-10,11	-14,33	-3,25	3,87	3,61
Czas okrążenia	-	-	28,31	14,88	15,67

Zmiana wartości tego parametru znacząco wpłynęła na efektywność treningu oraz tym samym na uzyskiwane przez agentów rezultaty. Dla niskich wartości *learning rate*: 1,00E-5 oraz 5,00E-05 agenci nie byli w stanie pokonać nawet jednego okrążenia toru. Dla wyższych wartości stopnia nauki agenci uzyskiwali znacznie korzystniejsze wyniki. Najlepsze rezultaty odnotowano dla wartości 5,00E-04.

- **Batch size**

Tabela 6.7 Wyniki oceny modelu dla zmiennej wartości parametru batch size dla metody RL

Batch size	32	64	128	256	512
Ukończone okrążenia	10,75	11,41	10,5	10,38	6,75
Liczba kolizji	6,875	3,58	4,08	7,5	1,25
Nagroda łączna	3,07	3,79	3,68	2,34	3,1
Czas okrążenia	15,28	15,08	14,86	14,57	23,11

Podobnie, jak parametr liczby ukrytych warstw sieci neuronowej, tak wartość *batch size*, nie miała tak znaczącego wpływu na uzyskiwane przez agentów rezultaty, jak chociażby *learning rate*. Najkorzystniejsze rezultaty uzyskano dla wartości 64 (choć to dla wartości parametru 128 oraz 256 uzyskano niższe średnie czasy okrążeń).

6.4.2 Imitation learning

Ze względu na wysoką niestabilność treningu zaobserwowaną dla *imitation learning* w próbach początkowych 6.3, zdecydowano się na zmianę kolejności parametryzowanych parametrów, celem jak najszybszego ograniczenia niestabilności treningu, poprzez odpowiedni dobór *learning rate*.

- **Learning rate**

Tabela 6.8 Wyniki oceny modelu dla zmiennej wartości learning rate dla metody IL

Learning rate	1,00E-05	5,00E-05	1,00E-04	5,00E-04	1,00E-03
Ukończone okrążenia	9,63	9	3,75	0,5	0,88
Liczba kolizji	7	7,38	25,38	36,5	23,25
Nagroda łączna	1,3	1,94	-11,16	-36,26	-25,35
Czas okrążenia	17,54	16,63	19,64	33,48	28,14

W przypadku *imitation learning*, a więc algorytmu BC, korzystniejsze rezultaty osiągnięto dla niższych wartości *learning rate* (przeciwnie do *reinforcement learning*). Najlepsze wyniki osiągnął agent wykorzystujący *learning rate* wynoszący 5,00E-05. Modele uczone z tą wartością parametru okazał się nieznacznie lepsze od tych wykorzystującego parametr przyjmujący wartość 1,00E-05. Porównując uśrednione wyniki przedstawione w tabeli zauważyć można, że odpowiedni dobór tego parametru miał kluczowe znaczenie na efektywność procesu treningu.

- **Liczba ukrytych warstw**

Tabela 6.9 Wyniki oceny modelu dla zmiennej wartości parametru liczby ukrytych warstw sztucznej sieci neuronowej dla metody IL

Liczba ukrytych warstw	**1**	**2**	**3**	**4**
Ukończone okrążenia	10,63	9	4,88	2,75
Liczba kolizji	0,88	7,38	15,5	21,88
Nagroda łączna	4,52	1,94	-6,09	-13
Czas okrążenia	15,89	16,63	19,51	21,14

Dla parametru liczby ukrytych warstw sieci neuronowej różnice w osiąganych przez modele uśrednionych wynikach, przedstawionych w tabeli 6.9 nie są tak znaczące, jak w przypadku parametryzacji *learning rate* (tabela 6.8). Można jednak zaobserwować zdecydowaną różnicę między wartościami maksymalnymi i minimalnymi, np. średniej nagrody łącznej, dla której to wynosiła ona 17,52. Najlepsze rezultaty osiągnięto dla najniższej wartości parametru, wynoszącej 1.

- **Batch size**

Tabela 6.10 Wyniki oceny modelu dla zmiennej wartości parametru batch size dla metody IL

Batch size	**32**	**64**	**128**	**256**	**512**
Ukończone okrążenia	9	7,25	10,63	10,13	9,75
Liczba kolizji	4,13	4,5	0,88	2,13	3,25
Nagroda łączna	2,83	1,41	4,52	4,84	3,69
Czas okrążenia	17,73	20,96	15,89	16,08	17,13

W przypadku parametru *bath size* najkorzystniejsze wyniki odnotowano przy wartości parametru wynoszącej 128. Były one zbliżone do tych osiągniętych dla wartości 256, gdzie odnotowano jeszcze wyższą średnią wartość nagrody łącznej. Jednak to dla parametru 128 osiągnięto korzystniejsze rezultaty dla kryteriów ukończonych okrążeń, czasu okrążenia oraz liczby kolizji.

- **Interwał podejmowania decyzji**

Tabela 6.11 Wyniki oceny modelu dla zmiennej wartości interwału decyzji dla metody IL

Interwał decyzji	**1**	**5**	**10**	**15**	**20**
Ukończone okrążenia	10	11,25	11	11,25	9,25
Liczba kolizji	1,2	0	0	0	3,5
Nagroda łączna	5,05	5,06	5,07	5,08	3,1
Czas okrążenia	15,04	14,91	15,81	14,88	17,75

Dla metody *imitation learning* różnice w uśrednionych wynikach uzyskiwanych dla kolejnych wartości interwału decyzji były bardzo zbliżone, jednak najlepsze rezultaty uzyskano dla wartości 15.

6.4.3 Podsumowanie parametryzacji

Końcowy zestaw wartości parametrów przedstawiono w tabeli 6.12. Zostaną one zastosowane w dalszych badaniach.

Tabela 6.12 Wartości uzyskane w procesie parametryzacji

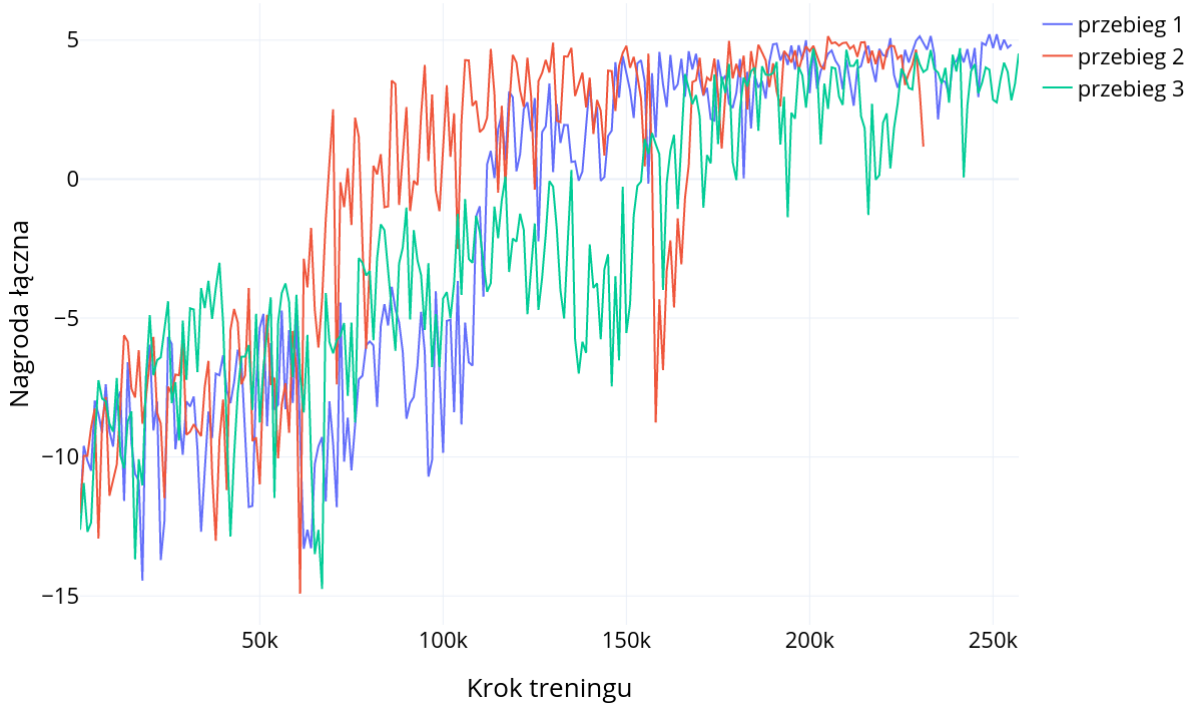
Nazwa parametru	**Wartość**	
	RL	**IL**
Interwał decyzji	15	15
Learning rate	5,00E-04	5,00E-05
Batch size	64	128
Liczba ukrytych warstw	2	1
Nagroda pasywna	Tak	-

6.5 Próby porównawcze

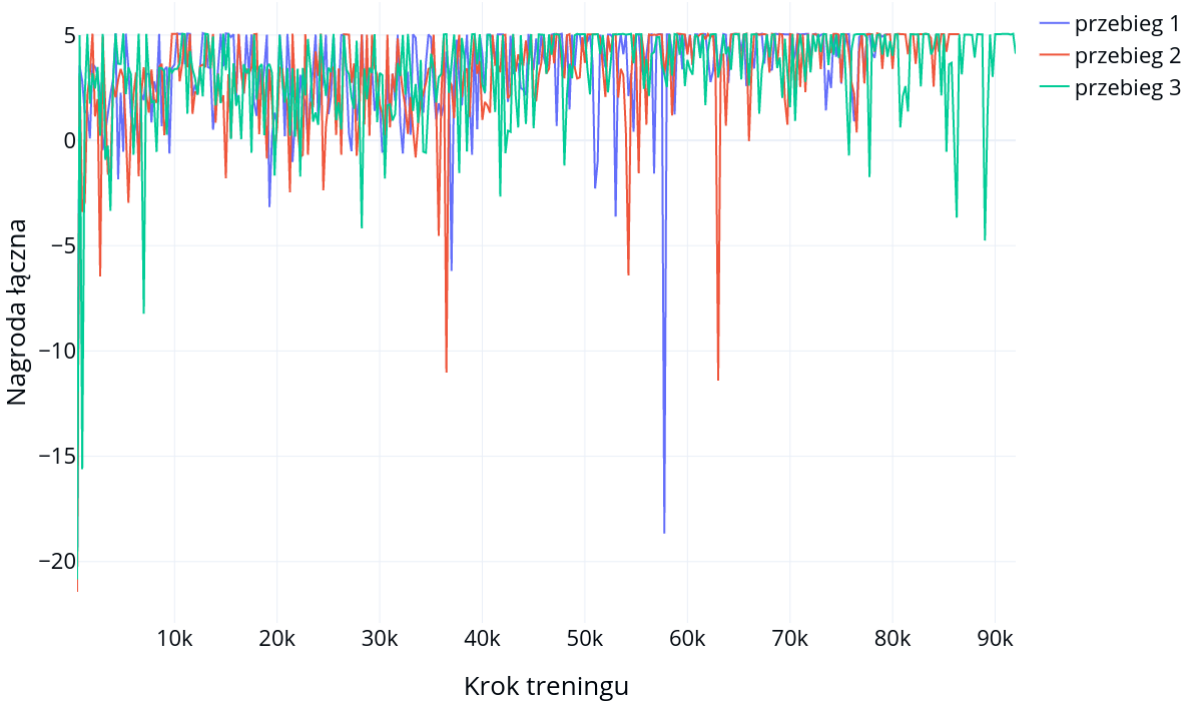
W tej części badań przeprowadzone zostanie bezpośrednie porównanie modeli wygenerowanych z wykorzystaniem metod *imitation* i *reinforcement learning*. Pierwszym krokiem będzie przeprowadzenie dla każdej z metod 3 sesji nauczania, trwających po 60 minut. W celu uzyskania lepszego punktu odniesienia pomiary przeprowadzone zostaną również dla agenta sterowanego przez gracza. Modele zostaną przetestowane najpierw na torach, które były wykorzystywane w procesie ich nauki. Następnie podjęta zostanie próba przetestowania ich na torze 3, z którym agenci nie mieli styczności podczas procesu nauki. Ostatnią częścią badań będzie porównanie modeli w zależności od czasu trwania procesu nauki. Należy pamiętać, że przedstawione wyniki są wartościami uśrednionymi. Dodatkowo podane zostaną wartości odchylnia standardowego pomiędzy wartościami kryteriów porównawczych.

Konfiguracja treningu:

- Ograniczenie: czasowe, 60 minut



Wykres 6.3Rozkład nagrody łącznej dla kolejnych kroków nauki dla przebiegów porównania głównego metodą RL



Wykres 6.4 Rozkład nagrody łącznej dla kolejnych kroków nauki dla przebiegów porównania głównego metodą IL

Analizując rozkład nagrody łącznej dla poszczególnych kroków treningu, dla obu metod można zauważyć tendencje zaobserwowane w rozdziale 6.2. Modele uzyskane metodą *imitation learning* szybciej osiągają wartości maksymalne, lecz ich nauka jest mniej stabilna. Modele *reinforcement learning* początkowo uzyskują znacznie niższe nagrody, jednak wraz z postępem nauki średnia rośnie.

- **Porównanie modeli w środowisku nauki**

Modele w czasie porównania poruszać się będą po dwóch torach, na których odbywała się ich nauka.

Tabela 6.13 Wyniki oceny modelu przy porównaniu w środowisku nauki dla metody RL

Przebieg	**1**	**2**	**3**	**Średnia**
Ukończone okrążenia	11	10,25	10,5	10,58
Liczba kolizji	3,75	7,75	1,5	4,33
Nagroda łączna	4,24	1,69	4,58	3,50
Czas okrążenia	15,26	16,57	15,99	15,94

Tabela 6.14 Wyniki oceny modelu przy porównaniu w środowisku nauki dla metody IL

Przebieg	**1**	**2**	**3**	**Średnia**
Ukończone okrążenia	9	10	10,5	9,83
Liczba kolizji	4,5	0	3,5	2,67
Nagroda łączna	2,31	4,97	3,24	3,51
Czas okrążenia	18,77	16,44	16,57	17,26

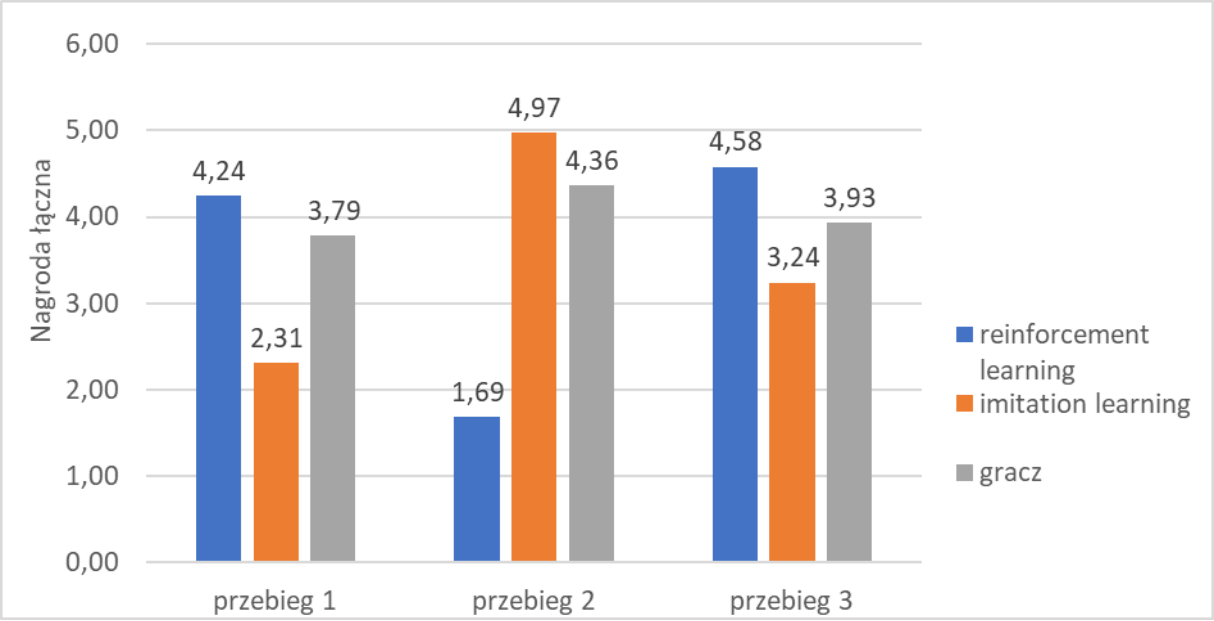
Tabela 6.15 Uśrednione wyniki oceny modelu dla wszystkich metod przy porównaniu w środowisku nauki

Metoda	**RL**	**IL**	**Gracz**
Ukończone okrążenia	10,58	9,83	9,83
Liczba kolizji	4,33	2,67	5
Nagroda łączna	3,50	3,51	4,02
Czas okrążenia	15,94	17,26	16,34

Tabela 6.16 Odchylenie standardowe dla wyników oceny modelu dla wszystkich metod przy porównaniu w środowisku nauki

Metoda	**RL**	**IL**	**Gracz**
Ukończone okrążenia	0,38	0,76	0,58
Liczba kolizji	3,17	2,36	2,00
Nagroda łączna	1,58	1,35	0,30
Czas okrążenia	0,66	1,31	0,40

Uśrednione wyniki przedstawione w tabeli 6.15 wykazują, że dla kryteriów liczby i czasu okrążeń najbardziej korzystniejsze rezultaty uzyskano dla metody *reinforcement learning*. Najmniej kolizji uzyskano dla metody *imitation learning*, natomiast najwyższą nagrodę łączną uzyskał agent sterowany przez człowieka.



Wykres 6.5 Średnia nagroda łączna dla kolejnych przebiegów przy porównaniu w środowisku nauki

- **Porównanie modeli w nowym środowisku**

Wygenerowane wcześniej modele zostaną poddane próbie przejechania toru, na którym nie poruszały się w czasie procesu nauki. Tym samym zostanie sprawdzone, czy algorytmy dokonały odpowiedniego uogólnienia problemu.

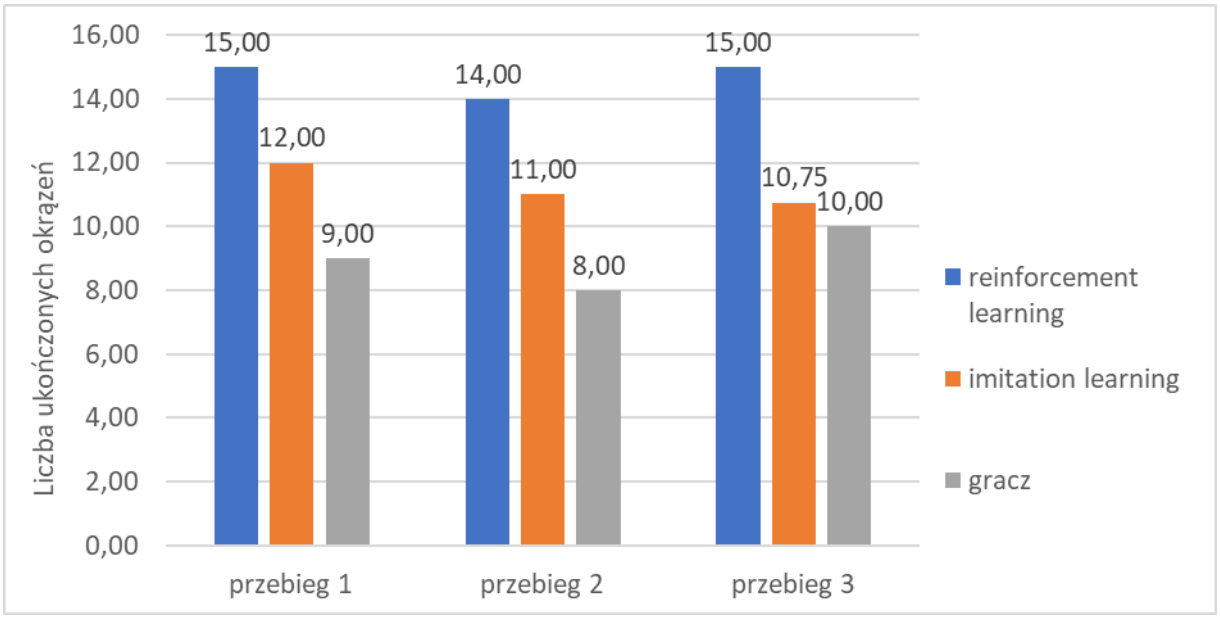
Tabela 6.17 Uśrednione wyniki oceny modelu dla wszystkich metod przy porównaniu w nowym środowisku

Metoda	RL	IL	Gracz
Ukończone okrążenia	14,67	11,25	9
Liczba kolizji	0	0	9
Nagroda łączna	5,38	4,76	2,23
Czas okrążenia	11,71	15,16	19,19

Tabela 6.18 Odchylenie standardowe dla wyników oceny modelu dla wszystkich metod przy porównaniu w nowym środowisku

Metoda	RL	IL	Gracz
Ukończone okrążenia	0,58	0,66	1
Liczba kolizji	0	0	5
Nagroda łączna	0,03	0,11	1,01
Czas okrążenia	0,25	0,79	2,17

Modele uzyskały dobre wyniki pomimo poruszania się po torze, który nie był wykorzystywany w procesie nauki. Powyższe wskazuje, że wygenerowane polityki są uniwersalne. Podczas badania żaden z agentów korzystających z wygenerowanych modeli nie dopuścił do ani jednej kolizji. Uzyskane rezultaty są lepsze niż dla przejazdów testowych w pierwszej części badania. Może to wynikać z charakterystyki toru, który posiada łagodniejsze zakręty. Warto zauważyć, że biorąc pod uwagę kryteria ukończonych okrążeń, nagrody łącznej oraz czasu okrążenia, korzystniejsze rezultaty uzyskano dla metody *reinforcement learning*. Ponadto obie metody uzyskały korzystniejsze rezultaty od tych, które zanotowano dla agenta sterowanego przez gracza, cechując się ponadto znacznie większą powtarzalnością, o czym może świadczyć różnica w odchyleniu standardowym wyników przedstawionym w tabeli 6.18.



Wykres 6.6 Liczba ukończonych okrążeń dla kolejnych przebiegów przy porównaniu w nowym środowisku

- **Badanie wpływu czasu nauki na sprawność modelu**

Ostatnim punktem badań było przetestowanie sprawności model, w zależności od czasu trwania procesu nauki. Do porównania zostały wykorzystane prezentowane już wyniki modeli, które trenowane były przez okres 60 minut. Dla każdej z metod dodatkowo przeprowadzono ocenę 3 modeli, których trening trwał przez okres 15 oraz 120 minut.

Tabela 6.19 Uśrednione wyniki oceny modeli przy różnych czasach treningu dla metody RL

Czas nauki [minuty]	**15**	**60**	**120**
Ukończone okrążenia	6,67	10,58	0
Liczba kolizji	3,08	4,33	0,63
Nagroda łączna	2,19	3,50	-6,62
Czas okrążenia	21,45	15,94	-

Tabela 6.20 Odchylenie standardowe dla wyników oceny modeli przy różnych czasach treningu dla metody RL

Czas nauki [minuty]	**15**	**60**	**120**
Ukończone okrążenia	0,63	0,38	0
Liczba kolizji	1,77	3,17	0,88
Nagroda łączna	1,21	1,18	0,86
Czas okrążenia	1,78	0,66	-

Analizując wyniki przedstawione w tabeli 6.19 możemy zauważyć, że długi czas treningu nie przełożył się na poprawę przystosowania modeli, wręcz przeciwnie- uzyskane w ten sposób modele w fazie oceny modelu nie pokonały, ani jednego okrążenia toru. Tak słabe wyniki mogą spowodowane być przeuczeniem modeli. Zbyt długi trening doprowadził do niepoprawnego ukształtowania modelu, w rezultacie tracąc umiejętności sprawnego wykonywania zadania. Jak wynika ze znikomych wartości odchylenia standardowego (tabela 6.20) problem przeuczenia powtórzył, się w każdym testowanym modelu, którego okres treningu wynosił 120 minut.

Modele trenowane przez okres 15 minut, radziły sobie gorzej niż te trenowane przez okres 60 minut. Wyjątkiem jest liczba kolizji, która okazała się być niższa dla krócej trenowanych modeli. Może to być spowodowane bardziej zachowawczą jazdą agenta, co może potwierdzać duża różnica w metrykach czasu okrążenia oraz liczby pokonanych okrążeń. Modele, których nauka trwała dużej cechowały się niższymi wartościami odchylenia standardowego pomiędzy modelami.

Tabela 6.21 Uśrednione wyniki oceny modeli przy różnych czasach treningu dla metody IL

Czas nauki [minuty]	**15**	**60**	**120**
Ukończone okrążenia	10,83	9,83	8,13
Liczba kolizji	4,17	2,67	8,00
Nagroda łączna	5,44	3,51	0,55
Czas okrążenia	15,35	17,26	16,76

Tabela 6.22 Odchylenie standardowe dla wyników oceny modeli przy różnych czasach treningu dla metody RL

Czas nauki [minuty]	**15**	**60**	**120**
Ukończone okrążenia	0,14	0,76	0,53
Liczba kolizji	3,79	2,36	0,35
Nagroda łączna	4,13	1,35	0,52
Czas okrążenia	0,26	1,31	0,38

Analizując wyniki dla metody *imitation learning* przedstawione w tabelach 6.21 oraz 6.22 możemy zauważyć, że różnice pomiędzy uśrednionymi wynikami są znacznie mniejsze niż w przypadku tych uzyskanych dla metody *reinforcement learning* (tabele 6.19 i 6.20). Najkorzystniejsze wyniki odnotowano dla modeli trenowanych najkrócej- przez 15 minut, najgorsze dla tych trenowanych najdłużej- 120 minut. Jednak dla uczonych dłużej odnotowano niższe wartości odchyleń standardowych dla kryteriów porównawczych.

7 Podsumowanie

7.1 Ocena uzyskanych rezultatów

Jednym z najważniejszych etapów prac badawczych był proces parametryzacji. Wpłynął on na znaczną poprawę osiąganych przez modele rezultatów, zwłaszcza w przypadku strategii *imitation learning*. Największy wpływ odnotowano dla parametrów *learning rate* oraz dla interwału podejmowanych decyzji. Parametryzacja jest bardzo czasochłonnym procesem, gdyż wymaga wielokrotnego powtarzania długiego procesu nauczania. Ponadto dla każdej z wartości parametru trening powinien być powtórzony kilka razy, a wyniki powinny być uśrednione. Jest to spowodowane tym, że nawet dla takiej samej konfiguracji proces nauki może wygenerować modele o różnych politykach, co zostało wykazane już w porównaniu początkowym.

Podczas badań dla obu metod udało się wygenerować modele, które w sprawny sposób wykonywały postawione przed nimi zadanie pokonywania trasy o zmiennej nitce toru. Jak wykazano, modele nauczania maszynowego osiągały rezultaty lepsze niż te zanotowane dla przejazdów gracza. Co ważne modele uzyskały dobre wyniki, również wtedy, gdy poruszały się po torze, który nie był wykorzystywany w procesie nauki. Powyższe wskazuje, że wygenerowane polityki zachowania są uniwersalne, a algorytmy dokonały odpowiedniej generalizacji. Oznacza to również, że według kryterium przedstawionego w rozdziale 2.2 uzyskane modele można określić jako inteligentne, gdyż potrafią dostosować się do zmieniającego środowiska.

W przypadku krótkiego czasu treningu (15 minut), korzystniejsze rezultaty uzyskano dla metody *imitation learning*. W przypadku dłuższego procesu nauczania (60 minut) lepsze wyniki odnotowano dla modeli wygenerowanych z wykorzystaniem *reinforcement learning*. Ta metoda lepiej radziła sobie z maksymalizacją osiąganych przez agentów wyników. Długa nauka nie zawsze wpływa jednak korzystnie na efektywność modelu. Po treningu trwającym 120 minut dla metody *reinforcement learning* odnotowano objawy przeuczenia- wydajność modeli drastycznie spadła- agenci nie byli wstanie pokonać ani jednego okrążenia toru.

7.2 Możliwości rozwoju

W czasie prowadzonych badań dużo czasu poświęcono na parametryzację algorytmów. Pomimo tego, wciąż istnieją możliwości lepszego doboru parametrów. Porównania dokonywano dla ograniczonego zbioru wartości parametrów, którego rozmiar nie przekraczał 5 wartości. Dokładniejszym rozwiązaniem byłoby testowanie wartości parametrów np. w sposób liniowy, wymagałoby to jednak automatyzacji procesu parametryzacji oraz jeszcze większych nakładów czasowych. Ciekawym podejściem jest sposób parametryzacji zastosowany w programie *AlphaStar*, który opisany został w podrozdziale 2.2.1. Parametrami algorytmów podczas iteracyjnego treningu zarządzał algorytm strategii *reinforcement learning*, który na podstawie uzyskiwanych przez model rezultatów odpowiednio dostrajał parametry, celem maksymalizacji wydajności. Jest to interesujące rozwiązanie, wymagające jednak dużych zasobów zarówno czasowych, jak i sprzętowych.

Aby zmaksymalizować wydajność agenta w środowisku badawczym interesującym rozwiązaniem może być połączenie dwóch porównywanych strategii. Algorytmowi wskazano by plik demonstracyjny, którego analiza pomogłaby w rozpoczęciu treningu, demonstrując

podstawową politykę zachowania. Jak wykazały badania to właśnie podejście *imitation learning* jest bardziej efektywne w początkowych fazach nauki. W późniejszych fazach nauki w celu maksymalizacji wydajności agenta zastosowano by strategię *reinforcement learning*.

Dalszy rozwój badań, wiązać mógłby się również z modyfikacją środowiska testowego. Zamknięte tory wyścigowe, zastąpione mogłyby zostać poprzez otwarte trasy, symulujące drogi spotykane w ruchu ulicznym. Na trasach pojawić mogłyby się bardziej zróżnicowane przeszkody takie jak inne pojazdy czy piesi. W bardziej zawansowanym wariancie sztuczna inteligencja musiałaby stosować się do zasad ruchu drogowego uwzględniając czynniki takie jak znaki drogowe. Wiązałoby się to z koniecznością modyfikacji metod obserwacji otoczenia przez agenta oraz sposobu obliczania sygnału nagrody.

Literatura

1. Litman, Todd. *Autonomous vehicle implementation predictions*. Victoria, Canada: Victoria Transport Policy Institute, 2019.
2. *Autonomous driving – 5 steps to the self-driving car*
   https://www.bmw.com/en/automotive-life/autonomous-driving.html
   [dostęp dnia 20 maja 2019]
3. *Moral Machine*
   http://moralmachine.mit.edu/
   [dostęp dnia 20 maja 2019]
4. *The DARPA Grand Challenge: Ten Years Later*
   https://www.darpa.mil/news-events/2014-03-13
   [dostęp dnia 20 maja 2019]
5. *Self-Driving Cars: The Complete Guide | WIRED*
   https://www.wired.com/story/guide-self-driving-cars/
   [dostęp dnia 20 maja 2019]
6. *Aggressive Tesla Autopilot Mode With "Slight Chance Of A Fender Bender" Apparently Coming*
   https://www.carthrottle.com/post/aggressive-tesla-autopilot-mode-with-slight-chance-of-a-fender-bender-apparently-coming/
   [dostęp dnia 20 maja 2019]
7. *Self-Driving Cars Explained | Union of Concerned Scientists*
   https://www.ucsusa.org/clean-vehicles/how-self-driving-cars-work
   [dostęp dnia 20 maja 2019]
8. Alpaydin, Ethem. *Introduction to machine learning*. MIT press, 2014.
9. *Unity ML-Agents Toolkit Documentation*
   https://github.com/Unity-Technologies/ml-agents/tree/master/docs
   [dostęp dnia 20 maja 2019]
10. *Overcoming Sparse Rewards in Reinforcement Learning – Frank's World of Data Science & AI*
    http://franksworld.com/2018/10/01/overcoming-sparse-rewards-in-reinforcement-learning/
    [dostęp dnia 20 maja 2019]
11. Gudimella, Aditya, et al. "Deep reinforcement learning for dexterous manipulation with concept networks." *arXiv preprint arXiv:1709.06977* (2017).
12. *IBM100 - Deep Blue*
    https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/
    [dostęp dnia 20 maja 2019]
13. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

14. *AlphaGo | DeepMind*
https://deepmind.com/research/alphago/
[dostęp dnia 20 maja 2019]

15. *OpenAI Five*
https://openai.com/five/
[dostęp dnia 20 maja 2019]

16. *Dota 2 | Eurogamer.net*
https://www.eurogamer.net/articles/2011-08-19-dota-2-preview
[dostęp dnia 20 maja 2019]

17. Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).

18. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II | DeepMind*
https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/
[dostęp dnia 20 maja 2019]

19. Yuxi Li, *Reinforcement Learning Applications*
https://medium.com/@yuxili/rl-applications-73ef685c07eb
[dostęp dnia 20 maja 2019]

20. Attia, Alexandre, and Sharone Dayan. "Global overview of imitation learning." *arXiv preprint arXiv:1801.06503* (2018).

21. *Katerina Fragkiadaki, Imitation Learning | Deep Reinforcement Learning and Control,*
https://katefvision.github.io/katefSlides/immitation_learning_I_katef.pdf
[dostęp dnia 20 maja 2019]

22. *Kaspar Sakmann, Behavioral Cloning — make a car drive like yourself*
https://medium.com/@ksakmann/behavioral-cloning-make-a-car-drive-like-yourself-dc6021152713
[dostęp dnia 20 maja 2019]

23. *Jerry Qu, I Created A Virtual Self Driving Car With Deep Q-Networks*
https://towardsdatascience.com/i-created-a-virtual-self-driving-car-with-deep-q-networks-7f87c0aad7c8
[dostęp dnia 20 maja 2019]

24. Wang, Sen, Daoyuan Jia, and Xinshuo Weng. "Deep Reinforcement Learning for Autonomous Driving." *arXiv preprint arXiv:1811.11329* (2018).

25. *Gym | OpenAI*
https://gym.openai.com/
[dostęp dnia 20 maja 2019]

26. *Informacje dotyczące edytora Unity.*
https://unity3d.com/unity/editor
[dostęp dnia 20 maja 2019]

27. *Strona internetowa biblioteki TensorFlow*
https://www.tensorflow.org
[dostęp dnia 20 maja 2019]

28. *Top 16 Open Source Deep Learning Libraries and Platforms*
https://www.kdnuggets.com/2018/04/top-16-open-source-deep-learning-libraries.html
[dostęp dnia 20 maja 2019]

29. *TensorBoard: Visualizing Learning | TensorFlow*
https://www.tensorflow.org/guide/summaries_and_tensorboard
[dostęp dnia 20 maja 2019]

30. Lilian Weng, *Policy Gradient Algorithms*
https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html
[dostęp dnia 20 maja 2019]

31. Andrej Karpathy, *Deep Reinforcement Learning: Pong from Pixels*
http://karpathy.github.io/2016/05/31/rl/
[dostęp dnia 20 maja 2019]

32. Torabi, Faraz, Garrett Warnell, and Peter Stone. "Behavioral cloning from observation." *arXiv preprint arXiv:1805.01954*(2018).