POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: AUTOMATYKA I ROBOTYKA
SPECJALNOŚĆ: TECHNOLOGIE INFORMACYJNE W SYSTEMACH AUTOMATYKI

PRACA DYPLOMOWA INŻYNIERSKA

System wizualizacji działania algorytmów sztucznej inteligencji do zagadnień sterowania obiektami symulowanymi w środowisku Unity

A system for visualization of performance of Artificial Intelligence algorithms applied for control of objects simulated in the Unity environment

AUTOR:
Jakub Pawłowski

PROWADZĄCY PRACĘ:
dr Marek Bazan, Katedra Informatyki Technicznej

OCENA PRACY:
bardzo dobry

WROCŁAW, 2019

Spis treści

Spis rysunków

Spis tabel

Skróty

IT (ang. *Information Technology*)
ML (ang. *Machine Learning*)
AI (ang. *Artyficial Inteligence*)
RL (ang. *Reinforcement Learning*)
GA (ang. *Genetic Algorithm*)
mr (ang. *mutation rate*)
lr (ang. *learning rate*)

Rozdział 1

Wprowadzenie

1.1. Cel pracy inżynierskiej

Od lat sztuczna inteligencja (dalej AI) rewolucjonizuje branżę technologiczną, a swój początek miała w latach 50 XX wieku.

Narodziny dziedziny AI [6] miały miejsce podczas konferencji w Dartmouth w 1956r., gdzie niewielka grupa informatyków po raz pierwszy poruszyła to zagadnienie. Na przestrzeni dekad, AI zaczęło być uważane jako klucz do świetlanej przyszłości i porzucenie starej technologi.

W ujęciu podanym przez Słownik Języka polskiego PWN [20] sztuczna inteligencja oznacza dział informatyki badający reguły rządzące zachowaniami umysłowymi człowieka i tworzący programy lub systemy komputerowe symulujące ludzkie myślenie

Udział algorytmów uczenia maszynowego zwiększa się w zaskakującym tempie. W chwili obecnej tempo rozwoju oprogramowania hamowane jest przez ograniczone zasoby ludzkie. Czas programisty jest coraz droższy, co skutkuje coraz większą automatyzacją procesów wytwórczych.

Celem niniejszej pracy jest wizualizacja tego, w jaki sposób AI potrafi zastąpić developera w roli tworzenia oprogramowania. W symulacjach konwencjonalne układy sterowania obiektami zastąpione zostaną wielowarstwowymi sieciami neuronowymi. Zadaniem symulatora jest odzwierciedlenie warunków rzeczywistych zadania i umożliwienie algorytmom uczenia maszynowego wyznaczenia optymalnego modelu.

1.2. Zakres pracy inżynierskiej

Poniżej przedstawiono problemy, które zostaną poruszone w dalszej części pracy:

- Uczenie maszynowe przez wzmacnianie
- Symulacje w Unity
- Serwis ds. sztucznej inteligencji
- Komunikacja sieciowa z serwisem
- Archiwizacja modeli
- Uruchomienie i wdrożanie serwisu

W celu wizualizacji działania stworzonego serwisu zostaną zaprezentowane przykładowe problemy inżynierskie. Do każdego z nich została opracowana symulacja w wirtualnym środowisku Unity. Szczegółowy opis podejścia do problemu symulacji ma na celu zrozumienie sposobu myślenia w tworzeniu kolejnych bardziej złożonych projektów. Zastosowanie AI do ich realizacji może okazać się jedynym rozwiązaniem, bądź nieporównywalnie korzystniejszym pod względem nakładów pracy.

Rozdział 2

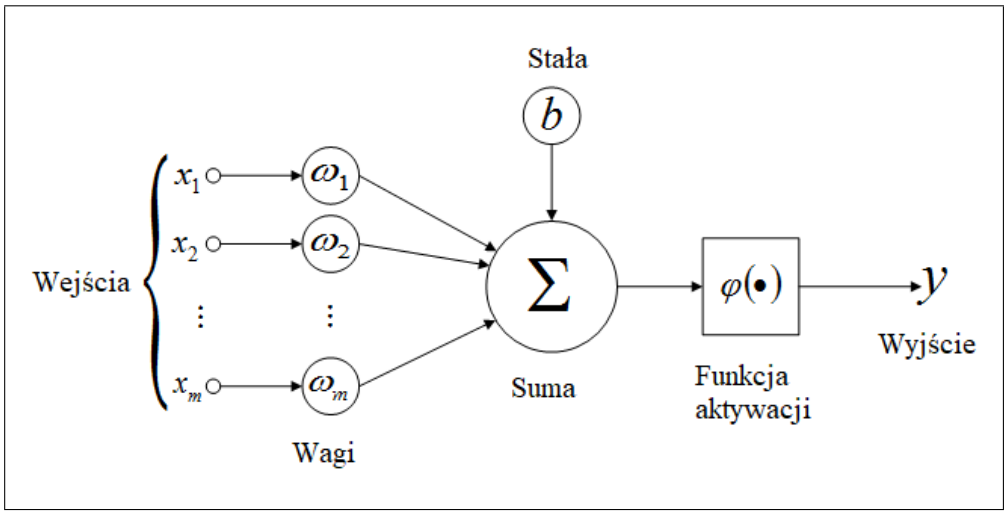
Zagadnienia sztucznej inteligencji

2.1. Wprowadzenie do sieci neuronowych

Sieć neuronowa [31][36] jest algorytmem uczenia maszynowego opierającym się na modelu naturalnego, ludzkiego neuronu. Mózg człowieka, w zależności od wieku i fazy rozwoju zawiera 15–33 miliardów neuronów, z których każdy może mieć do 10 tysięcy połączeń synaptycznych [28][27]. Wysyła i przetwarza sygnały w formie elektryczno-chemicznych sygnałów. Neurony te połączone są specjalną strukturą znaną jako synapsy. Synapsy pozwalają neuronom na przekazywanie sygnałów. Z wielu neuronów powstaje sieć neuronowa. Jest ona złożoną i trudną do wytłumaczenia strukturą, mogącą wykonywać skomplikowane i niemożliwe do zaprogramowania operacje.

2.1.1. Model matematyczny pojedynczego neuronu

W roku 1969 [24], przedstawiony został model matematyczny neuronu. Był to pierwszy krok w kierunku wytłumaczenia zasady działania ludzkiego mózgu. Analogicznie do rzeczywistości, model matematyczny reaguje na pobudzenie sygnałem wejściowym X, którego ostateczną odpowiedzią jest stan wyjściowy Y. Zakłada się, że X może być dowolnym wektorem informacji o rzeczywistości, natomiast Y wektorem odpowiedzi *sztucznej inteligencji* na dany stan.



Rys. 2.1: Model matematyczny pojedynczego neuronu

Omówienie rysunku 2.1 [26]

- Z prawej strony schematu modelu znajduje się wektor wejść X

- Każda z wartości mnożona jest przez pojedynczą wagę wejścia

$$x_n w_n$$

- Iloczyny, oraz wartość stała (wyznaczana w procesie uczenia) sumowane są w punkcie wspólnym do postaci

$$sum = \sum_{i=0}^{N} x_i w_i + b \tag{2.1}$$

- Następnie zsumowana wartość poddawana jest *funkcji aktywacji* φ, której wynikiem jest wyjście *Y*.

$$Y = \varphi(sum) \tag{2.2}$$

2.1.2. Wielowarstwowy perceptron

[26][36] Utworzona, struktura warstwowa perceptronów tworzy tzw. *perceptron wielowarstwowy* z *ang. Multi Layer Perceptron* dalej **MLP**. [32] MLP jest to jednokierunkowa sieć neuronowa z liniowymi funkcjami potencjału postsynaptycznego i (zwykle nieliniowymi funkcjami aktywacji. Można ją podzielić na 3 główne rodzaje warstw.

2.1.3. Interpretacja odpowiedzi sieci neuronowej

Znaczenie odpowiedzi sieci zależy od jej zastosowania. Oto kilka najpopularniejszych przykładów:

- Przynależność do danej klasy
- Regresja wartości zmiennych
- Rozpoznawanie wzorców
- Podejmowanie decyzji

W dziedzinie sterowania obiektami kluczową rolę odgrywa zagadnienie ***podejmowania decyzji*** przez sztuczną inteligencję. Ma ona za zadanie wysterować obiekty wykonujące ściśle określony typ zadań. Zadaniem obiektu jest przekazanie swego ***stanu*** na wejście ***X*** modelu sieci, oraz wykonanie czynności zwane ***akcjami***, które model zwróci na wyjściu ***Y***

Przy założeniu, że model jest wyuczony i na zadany stan podejmuje decyzje w postaci wartości akcji, które są prawidłową czynnością dla obiektu z punktu widzenia maksymalnej korzyści, to możemy go zdefiniować jako sztuczną inteligencję ograniczoną co do funkcjonalności obiektu.

Czynnikiem decydującym o zaistnieniu takiej sytuacji jest zdefiniowanie odpowiedniej struktury sieci neuronowej, mogącej zrozumieć obsługiwane zjawiska dla obiektu i posiadając wystarczającą ilość wag, które stanowią *pamięć neuronową* dobranych optymalnie do funkcji celu.

2.2. Uczenie maszynowe

Zagadnienie uczenia maszynowego sięga przełomu lat 50. i 60. Kluczowym momentem w rozwoju tej technologii było powstanie systemu eksperckiego Dendral na Uniwersytecie Stanforda[1].

Rolą uczenia maszynowego w przypadku sieci neuronowych jest ustalenie właściwych wag każdego z neuronu, tak aby średni błąd odpowiedzi modelu na dany stan był jak najmniejszy. W

tym celu stosuje się wiele sposobów uczenia oraz algorytmów optymalizacji całego procesu. Jednak w pierwszej kolejności należy rozróżnić sposoby w jaki sieć jest uczona.

2.2.1. Rodzaje uczenia maszynowego

Każdy z rodzajów uczenia charakteryzuje się całkowicie odmiennym podejściem zarówno do sposobu uczenia, jak i wartości które nauczona sieć ma wyznaczać.
W uczeniu maszynowym można wyodrębnić trzy główne rodzaje:

- Uczenie nadzorowane
- Uczenie nienadzorowane
- Uczenie przez wzmacnianie

Przedmiotem pracy jest zagadnienie uczenia przez wzmacnianie. Celem tej metody jest wyznaczenie właściwego modelu sterowania (podejmowania decyzji) przez obiekt w środowisku czasu rzeczywistego.

2.2.2. Uczenie przez wzmacnianie

Podstawą uczenia maszynowego są dane. Posiadając skończony i skorelowany zbiór informacji o zjawisku współczesne algorytmy uczenia maszynowego są w stanie: przewidywać informacje, rozpoznawać wzorce, czy podejmować decyzje. Jednak ostatni z przytoczonych rodzajów uczenia *uczenie przez wzmacnianie* charakteryzuje się zupełnie innym podejściem do zagadnień wyznaczania optymalnych wartości wag, a jego założeniem jest brak danych do uczenia.

W realnym świecie nie posiadamy wystarczającej ilości danych dotyczących każdego z istniejących zjawisk. Funkcjonują dziedziny w których nie ma możliwości zgromadzenia danych uczących i wyuczenia na ich podstawie pożądanych modeli. Natomiast zawsze istnieje sposób do oceny stanu rzeczywistego i skuteczności systemu sterowania. W przypadku uczenia przez wzmacnianie, najważniejszą różnicą w porównaniu z pozostałymi typami uczenia jest całkowity brak wymagania danych do uczenia. Elementem kluczowym w dążeniu do minimum błędu jest sposób ewaluacji. czyli metoda testująca skuteczność modelu w czasie rzeczywistym.

2.3. Q-learning

Sterowanie obiektami odbywa się na zasadzie podejmowania decyzji przez agenta [2]. **Agentem** nazywamy sterowany obiekt w wirtualnym środowisku. Każdy z agentów może posiadać unikalny stan i obserwacje. Za każdym zapytaniem agent podejmuje decyzję, którą z akcji podjąć w oparciu o stan rzeczywisty. Na tej podstawie zbiór reprezentuje wyrażenie 2.3. Jego rolą jest przekazywanie informacji na temat stanu do modelu i wykonywanie wskazanej przez niego akcji [34].

$$A = \{1, ..., K\} \tag{2.3}$$

[15] [35]Agent podejmuje określoną akcje spośród skończonego zbioru *K* możliwości, z największą przewidywaną nagrodą. Następnie porównuje swoją pozycję z pozycją poprzednią i na jej podstawie wyznacza wartość nagrody wykonanej akcji, wynik tzw. **funkcji Q** [36].

$$stan, \quad działanie => oczekiwana \quad korzyść \tag{2.4}$$

Przy czym *oczekiwana korzyść* jest wynikiem *funkcji Q*.

2.3.1. Uczenie przez wzmacnianie sieci neuronowych, metodą najszybszego spadku

[34] W celu wyznaczenia gradientu do metody najszybszego spadku, oprócz akcji wykonanych będą potrzebne wartości akcji, które powinny być podjęte. Z powodu braku takich informacji, rolę akcji prawdziwych zastępuje nagroda. Wskazuje ona kierunek, w którym model powinien dążyć w celu zminimalizowania funkcji strat i osiągnięciu jak największej nagrody po wykonanym zadaniu. Przyjęta wartość prawidłowej akcji przyjmuje wzór 2.5 [15].

$$action_{true} = r + \gamma \quad Q^*(s', a') \tag{2.5}$$

[16] Wyrażenie Q^* oznacza podjętą akcję z przypisaną do niej nagrodą r. Suma ma za zadanie wyznaczyć kierunek akcji prawidłowej do wykonania. W celu zneutralizowania zbędnych akcji, które nie wpływają na stan nagrody służy współczynnik γ, którego wartość dąży do 1. Standardową wartością jest $\gamma = 0.999$. Mnożenie akcji przez współczynnik w kolejnych epokach zmniejsza jej wpływ i wartość w kolejnych poprawionych modelach sterowania. Przy założeniu, dużego zróżnicowanego zbioru akcji i nagród, przedstawiona metoda jest w stanie określić właściwy kierunek spadku. [13] Proces optymalizacji dąży do zminimalizowania błędu wynikającego z podjętych decyzji.

2.4. Algorytm genetyczny

[29] Jest to uniwersalny sposób do wyznaczenia optymalnego modelu sterowania przy posiadaniu metody jego ewaluacji. Założeniem jest traktowanie wag sieci neuronowych jako materiał genetyczny zwany chromosomem [23]. Pojedynczym osobnikiem jest sieć neuronowa z własnymi niepowtarzalnymi wagami. Grupa osobników tworzy populacje, ściele określonej wielkości. W skutek ewaluacji każdego z osobników sprawdzana jest skuteczność każdej z sieci. Populacje należy sortować tak, aby na górze znajdowały się osobniki o największym wyniku. Zaczynając od najlepszego osobnika zachodzi krzyżowanie z następnym w populacji. Opisana procedura [23] widoczna jest na schemacie 2.2

Krzyżowanie polega na dokonaniu wymieszania puli genowej dwóch krzyżowanych osobników. Dokonuje się to za pomocą uśredniania ich genów lub losowych zapożyczeń z dwóch osobników. Odwołując się do źródła [18], najbardziej efektywną metodą jest sekwencyjne zapożyczanie fragmentów z dwóch krzyżowanych gatunków. Tak utworzony potomek poddawany jest mutacji.

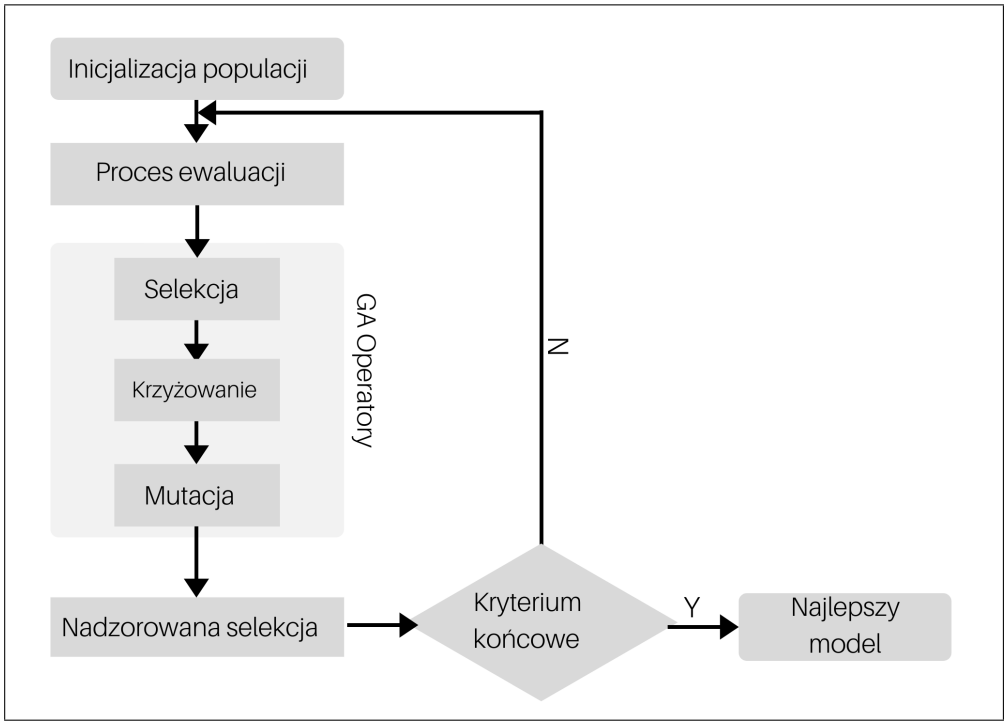
Mutacja jest to dodanie losowej zmiany w genach osobnika. Dokonuje się to poprzez utworzenie macierzy o kształcie wag potomka z wartościami czysto losowymi o rozkładzie normalnym. Elementy macierzy których, wartość jest co najmniej tej samej wielkości co wartość progowa 2.6 pozostają bez zmian.

$$x >= mr, \quad \text{gdzie} \quad mr - \text{prawdopodobieństwo mutacji} \tag{2.6}$$

Utworzona w ten sposób *macierz mutacji*, definiuje wybrane wagi poddawane zmianie losowej w przedziale wartość $[-\delta\ ;\ \delta]$ 2.7.

$$W' = W + \delta_w W_{mutable}, \quad \text{gdzie} \quad W_{mutable} - \text{macierz mutacji} \tag{2.7}$$

Siłą mutacji kieruje współczynnik δ , który można porównać do prędkości uczenia standardowej metody uczenia sieci neuronowych na podstawie zbioru danych. Ważne jest właściwe określenie mocy mutacji w algorytmie, aby zmiany w strukturze sieci nie pomijały minimum lokalnych, a w jak najszybszym tempie znalazły rozwiązanie. Na samym początku uczenia



Rys. 2.2: Schemat algorytmu genetycznego

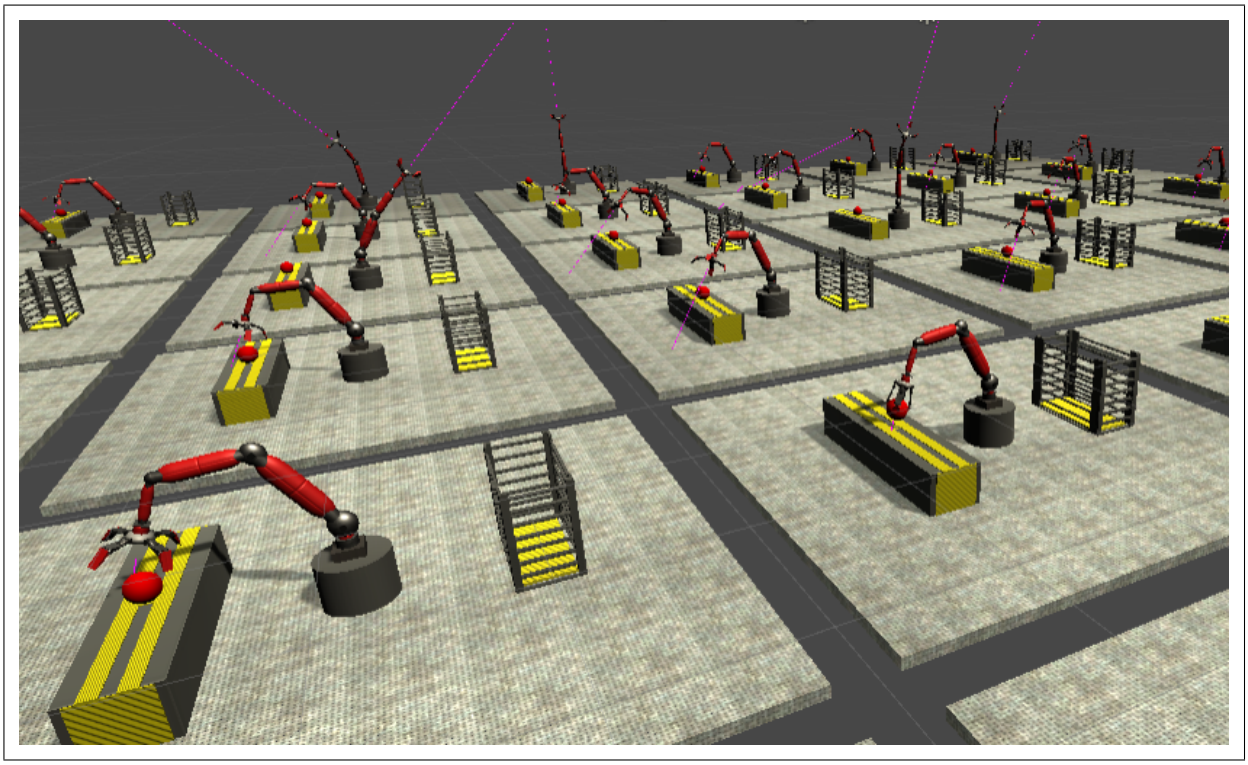
warto zwiększyć współczynnik, aby stworzyć jak najróżniejsze osobniki, a w kolejnych etapach przyjąć mniejsze wartości współczynnika.

Nadzorowana selekcja jest procesem porównania każdego z osobników populacji w danym pokoleniu. Znając wynik każdego z nich, tworzona jest list ułożona w kolejności malejącej co do wyniku osobnika. Do nowej populacji dodawany jest osobnik najlepszym wynikiem, oraz pouczony model na podstawie zebranych danych (Q-learning). Pozostałe miejsca mają szanse zająć osobniki w pozycji $2...N, gdzie N = [1, ..., P]$. Przetrwanie danego gatunku osobnika deprymuje ograniczony rozmiar populacji oraz ilość dzieci jaka powstaje na skutek krzyżowań. Brane są kolejno pary $(2, 3), (3, 4), ..., (N - 1, N)$, i poddawane operacji $Krzyzowanie(x, y) -> Mutacja(x, y)$ z warunkiem zatrzymania procesu przy wyczerpaniu się wolnych miejsc w populacji. W skutek selekcji naturalnej, najlepsze modele przekazują swoje geny (wagi) potomkom. Kolejne pokolenia osiągają taki sam, bądź lepszy wynik. W połączeniu z mechanizmem *Q-learningu* zgromadzone dane mają ogromną wartość w gromadzeniu zbioru uczącego w każdej iteracji. Pozwala to testować niekontrolowane i czysto losowe zmiany w wagach sieci. Tym samym wyznaczając właściwy kierunek spadku.

2.5. Federated Learning i system wieloagentowy

W celu przyspieszenia procesu uczenia oraz usprawnienia procesu, jedną z możliwości jest zwiększenie liczby agentów. Nazywa się to systemem wieloagentowym, z mechanizmem uczenia rozproszonego. [33] Temat uczenia rozproszonego w podejściu *Federated Learning*, jest wciąż rozwijającym i innowacyjnym zagadnieniem. Jego istotą jest wykluczenie posiadania prywatnych danych klienta i oddelegowanie do niego części metod uczących. Zaletą jest zachowanie całkowitej prywatności przez użytkowników i tym samym spełnienie obowiązujących na świecie standardów. Rozproszony system uczenia znajduje jednak zastosowanie nie tylko w komercyjnych projektach. Jest niezbędnym elementem w procesie wieloagentowego systemu uczącego. [22] Model oblicza straty na pojedynczym niezrównoważonym zbiorze danych jednego użytkownika. Za pomocą metody optymalizacji *SGD* wyznaczany jest gradient funkcji

strat i zwracany do serwisu macierzystego. Oddelegowanie obliczanie spadku do każdego z agentów niesie za sobą pewnie konsekwencje. Występuje duża wrażliwość na niezrównoważenie podzbioru. W takim celu nowy model przeliczany jest po zsumowaniu gradientów od każdego z agentów. W ten sposób dysponujemy pewną informacją o zjawisku z niejednorodnej dyspozycji.



Rys. 2.3: System wieloagentowy - manipulator przemysłowy

[14] [21] Zastosowanie systemu wieloagentowego ma za zadanie połączyć dwa światy uczenia poprzez *Q-learning* oraz *genetycznego*. Jak już zauważono 2.4 jedną z roli, populacji jest zgromadzenie dużego zbioru uczącego. W etapie selekcji algorytm *wędruje* po osobnikach, sumując gradient funkcji strat. Średni gradient całej populacji określa kierunek spadku [5]. Tak wytrenowany model, dodawany jest bez mutacji do następnej populacji osobników [19].

Rozdział 3

Środowisko Unity

Unity jest największą światową platformą developerską 3D czasu rzeczywistego. Przy pomocy tego zintegrowanego środowiska twórcy z całego świata posiadają narzędzia do tworzenia bogatych, interaktywnych doświadczeń w 2D, 3D, VR i AR. Skład 1000 inżynierów rozwija silnik Unity, aby posiadał najnowsze rozwiązania. Współpracują z nimi takie firmy jak Facebook, Google, Microsoft i Oculus.

Unity3D pozwala na stworzenie uniwersalnej aplikacji na wszystkie obsługiwane platformy sprzętowo-programistyczne.

Zastosowanie Unity ma zasięg 3 miliardów urządzeń na całym świecie. A w roku 2019 został zainstalowany 24 miliony razy. Stosowany jest w architekturze, branży *automotive*, konstrukcjach, projektowaniu, filmach, grach i innych dziedzinach. Co ważne pozwala uzyskać finalną projektowaną aplikację. [3]

3.1. Wprowadzenie

Aby stworzyć wirtualne środowisko należy zastosować *Unity Editor*. Jest to ekosystem służący do edycji świata wirtualnego, kompilacji oraz testowania rozwiązania. Silnik graficzny *Unity* oparty jest na następującej technologi:

- OpenGL - dla zastosowań stacjonarnych
- WebGL - w celu kompatybilności z przeglądarkami internetowymi

Backend silnika zapewnia skompilowany program napisany we wspieranych językach.

- C#
- UnityScript

Zrealizowane symulacje zostały w całości napisane przy pomocy języka C#, w oparciu o wersję stacjonarną z technologią OpenGL. Edytor zawiera gotowe narzędzia, które wystarczają do stworzenia większości mechanik potrzebnych do stworzenia symulacji, gier i wielu innych produktów. Ponadto posiada wiele funkcjonalności fizyki, efektów wizualnych i audio. Dzięki oparciu o język w pełni obiektowy jakim jest C#, dozwolone są podejścia o specyfice polimorfizmu i efektywnej formie wywarzania oprogramowania.

3.2. Fizyka

Każde prezentowane wirtualne środowisko określa się jako scena. Składa się ona z wielu elementów. Dla przykładu warto podać, że mogą być nimi: modele 3D, efekty wizualnie, dźwiękowe, widoki użytkownika.

Obiekt 3D może być rzeczywistym pojedynczym ciałem fizycznym bądź też abstrakcyjną grupą wielu obiektów. Jego podstawową cechą jest położenie w przestrzeni w sześciu współrzędnych 3.1

```
position = new Vector3(X, Y, Z);
rotation = new Vector3(rotX, rotY, rotZ);
```

Listing 3.1: Wektor położenia i rotacji

3.3. Programowanie w Unity

Programowanie w silniku Unity polega na osadzeniu tzw. skryptów w każdym żądanym obiekcie na scenie. Pojedynczy skrypt jest klasą obiektu Unity. Przykładową klasę obrazuje kod zamieszczony poniżej 3.2.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Globalization;

namespace driven_car {

public class drive : MonoBehaviour
{
    void Start()
    {
        TurnOnEngine();
    }

    void Update()
    {
        Drive();
    }
}
```

Listing 3.2: Skrypt Unity - przykład

Każdy ze skryptów jest częścią wspólną projektu w języku C#. Z tego powodu możliwe jest wzajemne dziedziczenie oraz praca obiektów zależnych od siebie.

3.4. Klient HTTP

Środowisko Unity posiada dedykowane wsparcie dla biblioteki TensorFlow, a także wykorzystanie jej w symulacjach [17]. Założeniem pracy jest stworzenia niezależnego serwisu do zagadnień sztucznej inteligencji oraz uczenie modeli sterowania z rozbudowaną wizualizacją procesu. Z tego punktu widzenia podejścia kompleksowego, symulacja ma nie odpowiadać za algorytmy uczące. Odpowiada ona jedynie za wydawanie zapytania do systemu co obiekt powinien zrobić.

Adres *127.0.0.1* jest hostem lokalnym serwisu. Każda jedna z operacji wykonywana jest poprzez wysłanie polecenia GET pod ten właśnie adres. Obsługiwane zapytania:

- **Zapytanie serwisu o wolny token**:

  `127.0.0.1/token/ai_1`

Każdy wolny aktor ma za zadanie zapytać serwis wolny *token* (klucz do zapytań) odpowiadający za konkretny gatunek poddawany procesowi ewaluacji przez środowisko. Aktor będzie wykorzystywał go w każdym z kolejnych zapytań do serwisu.

- **Zapytanie serwisu o akcje**:

  `127.0.0.1/use/ai_1/5,501237;5,501237;2,53845`

  Wartość *stanu* odczytana ze środowiska, zapisywana jest w formie *wektora informacji* 4.2.1. Korzystając z funkcji */use/uid/data*, gdzie *uid* jest identyfikatorem serwisu, a *data* to wektor informacji, uzyskujemy odpowiedź od serwisu. Odpowiedź zapisywana jest do zmiennej znakowej i odpowiednio interpretowana jako wektor liczb. Odpowiada on wektorowi akcji, które obiekt ma za zadanie podjąć poprzez interpretację części skryptowej Unity.

- **Przekazanie informacji o stanie, akcji, nagrodzie**:

  `127.0.0.1/use/ai_1/v9i9acno42x/5,5;5,1;2,5*4,2;3,4,2,5*1;0*0.001`

  Realizując proces Q-learningu, koniecznie jest gromadzenie danych o stanach, akcjach i nagrodach. Posługując się przypisanym *tokenem*, odczytane wartości podawane są kolejno w postaci wektorów informacji, wraz z separatorem $*$, pozwalającym serwisowi rozdzielić ciąg znaków na pojedyncze wektory.

- **Raport do serwisu**:

  `127.0.0.1/use/ai_1/v9i9acno42x/$0,5820`

  Ostatnim etapem pojedynczej symulacji jest przekazanie informacji o końcowej całkowitej nagrodzie do serwisu. Posługując się przypisanym *tokenem*, wartość nagrody podawana jest po znaku $, dzięki czemu serwis jest w stanie rozróżnić zapytanie o akcję od raportu

3.5. Proces uczenia

Przebiega w powtarzalnej (zapętlonej) procedurze inicjalizacji, pracy i restartu [29].

1. Ustawienie położenia i warunków początkowych dla pojedynczej symulacji
2. Praca obiektu
    1. Odczytanie i zapisanie aktualnego stanu obiektu
    2. Określenie nagrody aktualnego stanu obiektu
    3. Wysłanie zapytania do serwisu, przekazując mu aktualny stan, wcześniejszy stan, wcześniejszą akcje, przyrost nagrodę za wykonanie poprzedniej czynności
    4. Odczytanie odpowiedzi i wykonanie operacji
    5. Zapis stanu, podjętej akcji oraz aktualnej nagrody
3. Wykonanie zadania / koniec czasu symulacji / niewybaczalny błąd obiektu
    1. Odcztanie stanu i określenie nagrody
    2. (w przypadku należenia do populacji)Wysłanie wyniku końcowego (nagrody) pojedynczej symulacji.

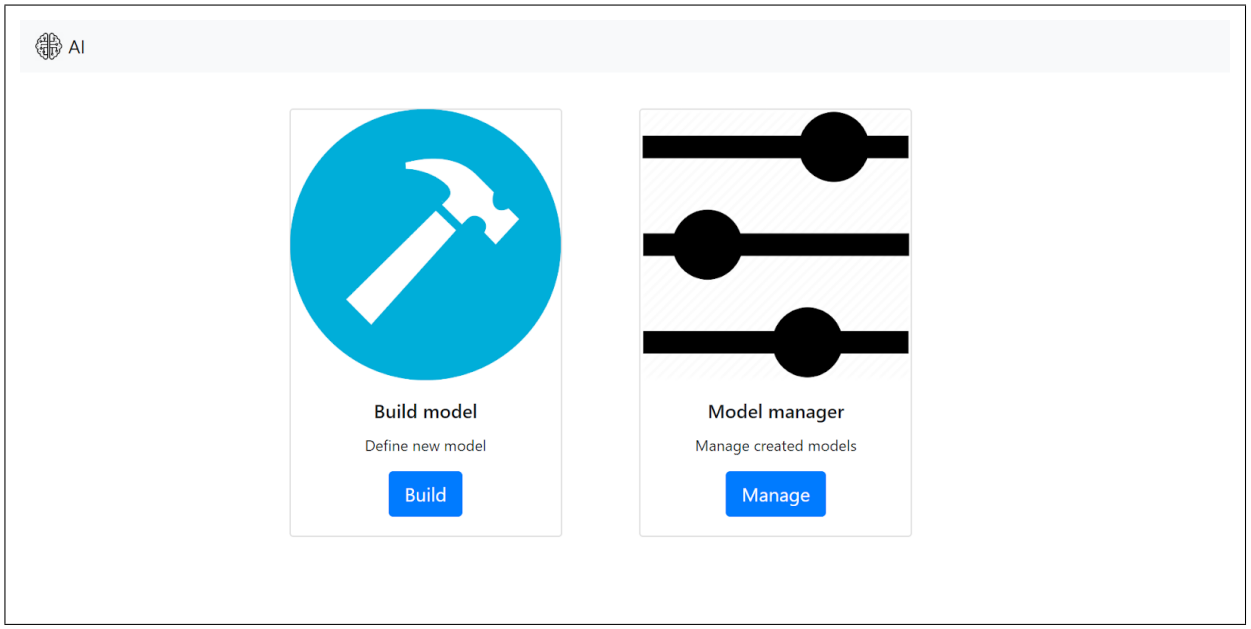
Wymieniony przebieg wynika z zastosowania algorytmów uczących typu genetycznego oraz Q-learningu. Każda z symulacji będzie opierać się na dostosowanej wersji przedstawionego procesu, z własnymi indywidualnymi zmianami wynikającymi ze specyfiki zadania obiektu.

Rozdział 4

Platforma AI

4.1. Założenia platformy i użyte technologie

Na potrzeby pracy została stworzona własna platforma, której głównym zadaniem jest tworzenie serwisów w celu zapewnienie dostępu do sterowania za pomocą sztucznej inteligencji dowolnego urządzenia mającego dostęp do Internetu bądź połączenia lokalnego.



Rys. 4.1: Zmiana struktury sieci

Wykorzystane technologie w serwisie:

- Flask - silnik aplikacji webowej[8]
- Pytorch - framework uczenia maszynowego[12]
- Pickle - archiwizacja modeli w formacie JSON[10]
- Plotly - wizualizacja procesu uczenia[11]
- NumPy - obliczenia matematyczne oraz generacja liczb losowych [9]

Adres repozytorium: `https://github.com/jkpawlowski96/AI`

4.2. REST API do komunikacji klient-serwer, przy użyciu protokołu HTTP

Aby możliwe było oddelegowanie mechaniku ML do osobnego serwisu, konieczne jest stworzenie sposobu komunikacji części symulacyjnej Unity z platformą obsługującą serwisy, gdzie:

- serwer - jest platformą AI
- klient - jest użytkownikiem platformy (środowisko Unity)

Istotą komunikacji pomiędzy klientem a serwerem, jest protokół HTTP. Dzięki czemu serwis jest dostępny również z poziomu przeglądarki internetowej, w którą jest wyposażone każde urządzenie z dostępem do Internetu lub sieci lokalnej. Klient wysyła polecenia GET do serwera, w których zawarta jest nazwa identyfikacyjna wykorzystywanego serwisu oraz informacja dla modelu na temat stanu lub inne zależne od aktualnie wykonywanej funkcjonalności w procesie uczenia.

Przykładowy wystawiony adres URL, w pliku `app.py` aplikacji webowej:

```
@app.route("/use/<string:uid>/<string:data>")
def service_use(uid, data):
    if uid not in db.uids:
        return 'null'
    return service_work(data, db.services[uid])
```

Powyższy kod przedstawia przykładową funkcjonalność - zapytania klienta do konkretnego serwisu, jaką akcję ma podjąć na podstawie podanego stanu, gdzie:

- uid - jest to identyfikator serwisu
- data - zawiera wektor informacji na temat aktualnego stanu obiektu

gdzie każdy z wektorów sporządzony jest w zdefiniowanym systemie rozumienia znaków.

4.2.1. Wektor informacyjny

Wektor informacyjny jest pojedynczą zmienną typu string składającą się ze znaków alfanumerycznych. Jest to uniwersalny sposób wymiany informacji między klientem a serwerem, a tym samym różnymi językami programowania. W przypadku projektu jest to strona serwera stworzona w języku Python oraz strona klienta stworzona w języku C#.

Pojedyncze wartość wektora oddzielone są separatorem "; " , dlatego przykładowy wektor zawierający informacje o położeniu obiektu w trójwymiarowej przestrzeni względem osi x, y, z. Dla przykładu $(10.5, 0, -6.9)$ będzie posiadać wartość zmiennej typu string w następującej postaci:

```
10.5;0;-6.9
```

Odpowiedź serwisu przyjmuje dokładnie takie same zasady dzielenia zmiennych liczbowych jak w przypadku wektora informacji o stanie. Dlatego przykładowa odpowiedź serwisu w postaci akcji do podjęcia (0.5 , 1) , będzie:

```
0.5;1
```

Istnieją różnice pomiędzy zapisem typu *string* liczby zmiennoprzecinkowej. Jest to niespójny znak decymalny, który w przypadku Pythona przyjmuje znak ".", natomiast C# ten sam znak

występuje z użyciem znaku ”, ”. dlatego zarazem zadaniem klienta jak i serwera jest dostosowanie wektora informacji poprzez zamianę każdego ze znaków decymalnych na swój wewnętrzny określony językiem programowania.

4.3. Tworzenie serwisu na platformie

Zadaniem platformy jest umożliwienie tworzenie i obsługiwane wielu serwisów. Serwis jest zdefiniowanym i stworzonym modelem sterowania obiektu. Na przykładzie omawianych zastosowań będą to serwisy:

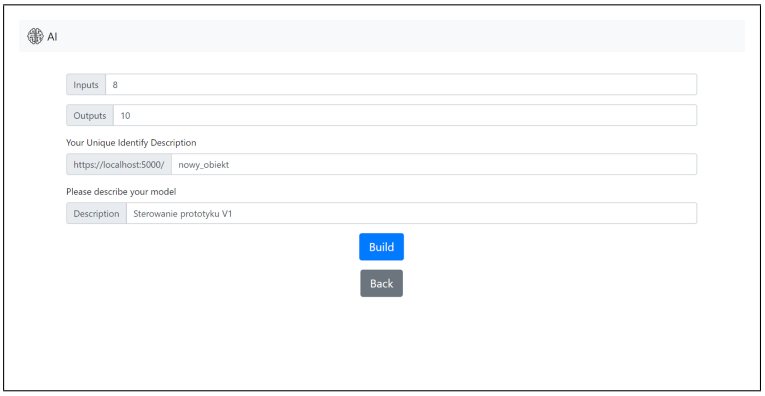
- *ai_1* dla samochodu autonomicznego
- *ai_2* dla manipulatora przemysłowego

Dla konkretnego problemu, należy własnoręcznie skonfigurować strukturę modelu serwisu:

- Ilość wejść
- Ilość wyjść

Oraz właściwości serwisu

- Nazwa własna serwisu - stanowi adres dostępu do modelu oraz konfiguracji serwisu
- Opis serwisu - widoczny w liście aktywnych serwisów



Rys. 4.2: Tworzenie serwisu

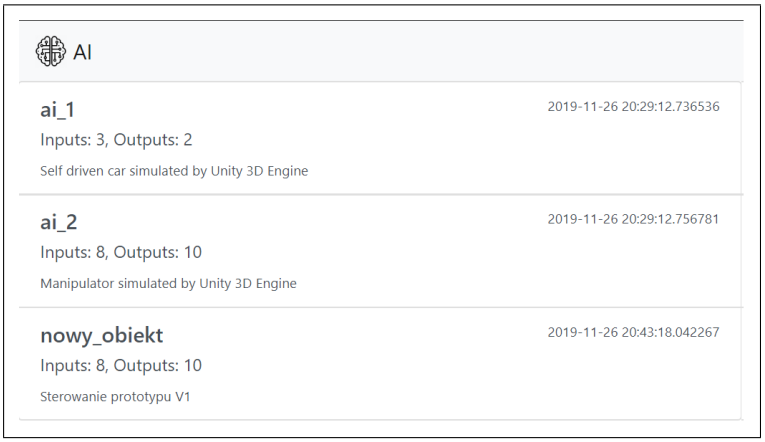
4.4. Zarządzanie serwisami

Aktualnie działające serwisy sterowania dostępne są w podstronie *manage*, która prezentuje interaktywną listę dostępnych i gotowych do użycia serwisów o unikalnej nazwie *uid*. Każde z pól posiada wymienioną nazwę oraz opis, który został podany w procesie tworzenia. Dodatkowo widoczna jest specyfikacja na temat modelu w postaci ilości obsługiwanych wejść oraz wyjść.

Dalsza konfiguracja dostępna jest poprzez wybór pola konkretnego serwisu.

4.5. Konfiguracja modelu i procesu uczenia

Każdy z serwisów może być edytowany oraz obserwowany poprzez zintegrowany jednolity interfejs 4.4. Można go podzielić na poniższe komponenty.



Rys. 4.3: Zarządzanie serwisami

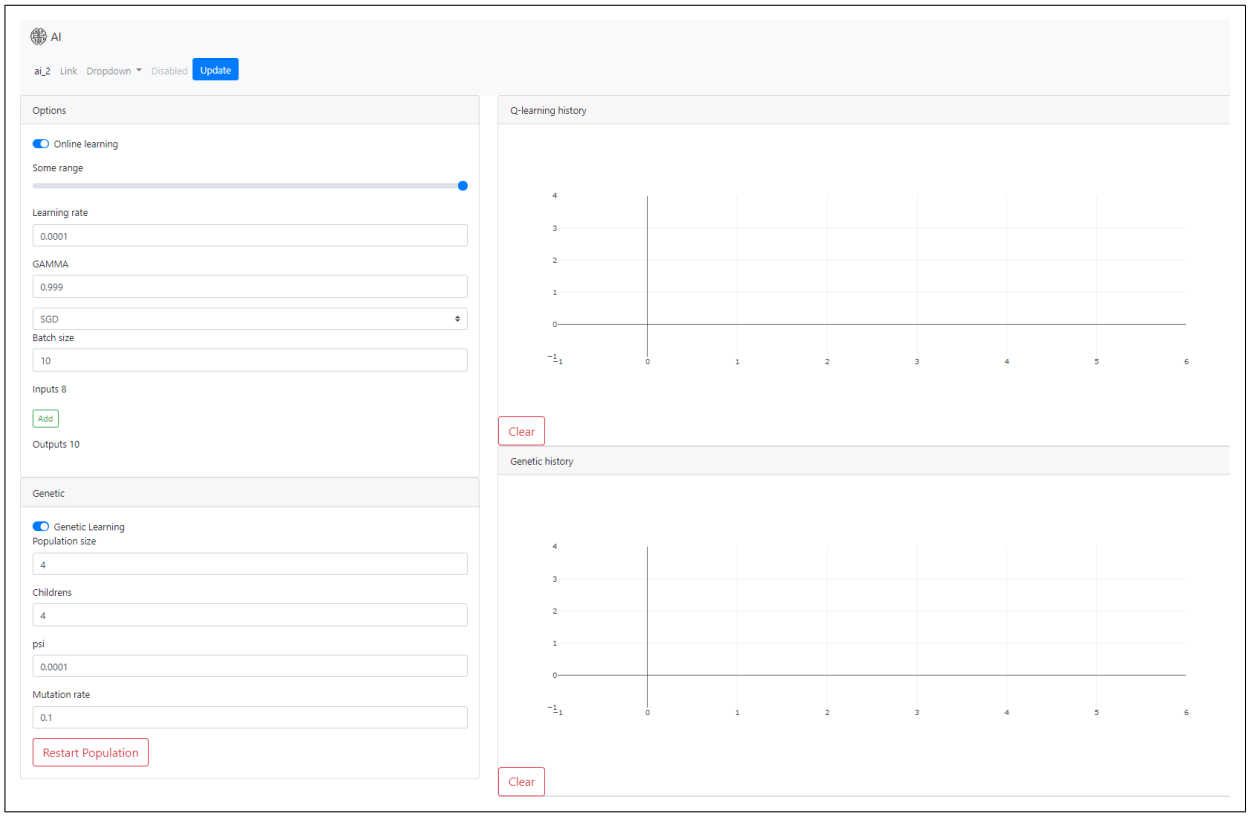
4.5.1. Tryb genetyczny

Okno opcji genetycznych 4.5, pozwala na zmianę parametrów takich jak:

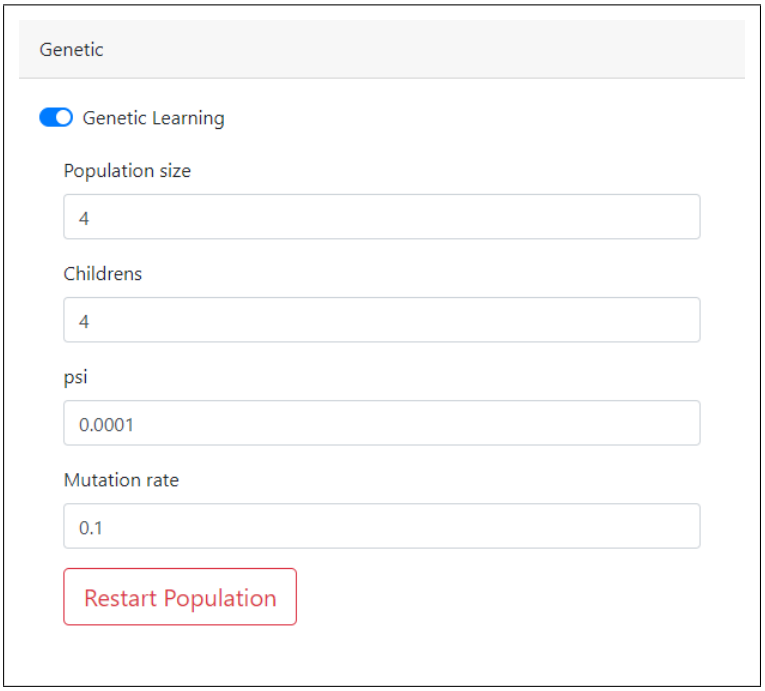
- Population size - rozmiar populacji
- Mutation rate - siła mutacji
- Children - liczba dzieci po krzyżowaniu

4.5.2. Opcje uczenia

Okno opcji uczenia 4.6, pozwala na zmianę parametrów takich jak:



Rys. 4.4: Widok serwisu



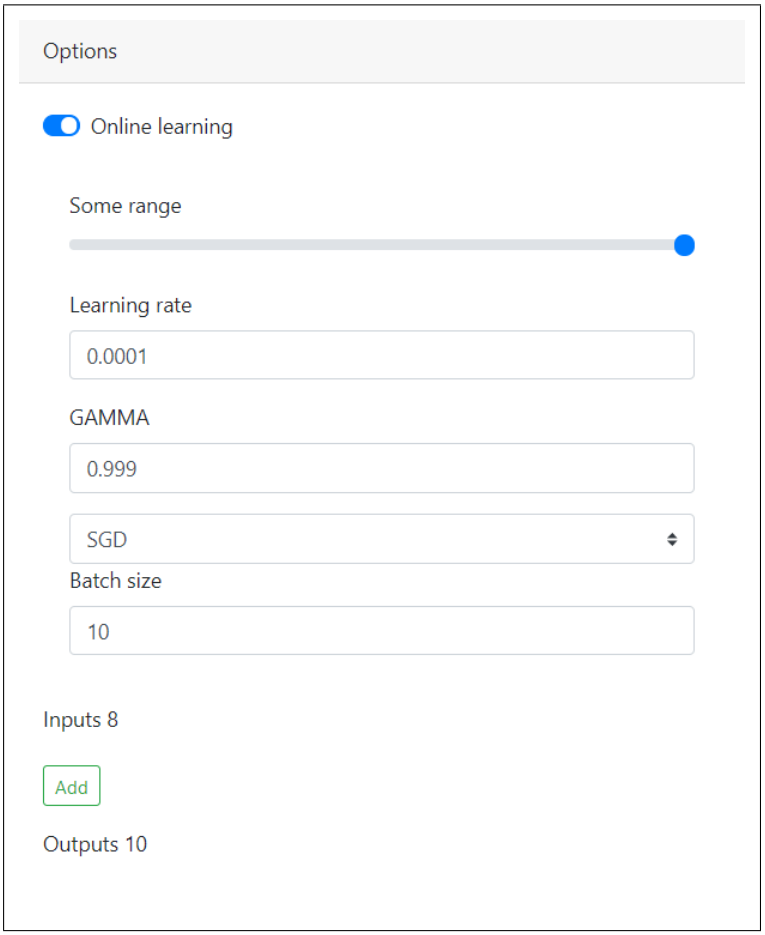
Rys. 4.5: Opcje genetyczne

- Online Learning - metoda q-learning. Model uczy się w czasie rzeczywistym. Każda z odpowiedzi klienta z informacją o stanie, akcji oraz nagrodzie, jest gromadzona i wykorzystywana do uczenia modelu na końcu bieżącej populacji
- Learning rate - prędkość uczenia
- GAMMA - wartość współczynnika *gamma*
- Batch size - rozmiar pojedynczego batcha w procesie uczenia
- Optimizer - metoda optymalizacji gradientu
- Struktura sieci - pozwala na modyfikacji warstw ukrytych, dodawanie, odejmowanie neuronów i warstw

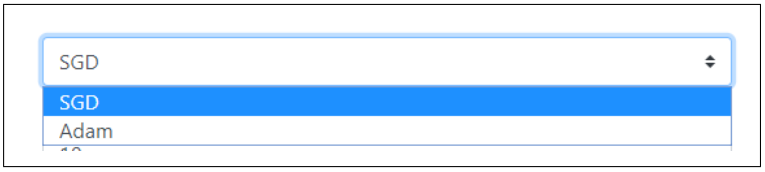
4.6. Wizualizacja procesu uczenia

W uczeniu przez wzmacnianie, czas oczekiwania na rozwiązanie względnie prostych problemów zdecydowanie przewyższa standardowe podejście uczenia na podstawie zebranych danych. W celu obserwacji procesu uczenia, niezbędny jest system akumulacji informacji i wyników, do późniejszego wglądu przez użytkownika. Serwis zbiera dane w na temat:

- **Maksymalnej całkowitej nagrody w generacji**
  Z założenia procesu uczenia algorytmu genetycznego 2.4, kolejne generacje spełniają założenie $max(reward_{n+1}) >= max(reward_n)$. Powodem jest pozostawianie najlepszego osobnika (modelu) w puli. Dlatego wykres *Maksymalnej całkowitej nagrody* przejmuje kształt wykresu schodkowego.
- **Sumy strat całej populacji osobników**
  Kryterium oceniające błąd między akcją porządną a dokonaną, sumowane jest do postaci jednolitej sumy gradientów błedu. W celu zauważenia powiązań między występującym błędem a aktualnym wynikiem, dla użytkownika dostępny jest wykres ww. sumy w każdej generacji (epoce).



Rys. 4.6: Podstawowe opcje uczenia



Rys. 4.7: Wybór optymalizatora

4.6.1. Interaktywne wykresy

Serwis prezentuje dwa wykresy procesu uczenia:

- Q-learning history - wartość sumy strat w każdej generacji
- Genetic history - maksymalna nagroda w każdej iteracji

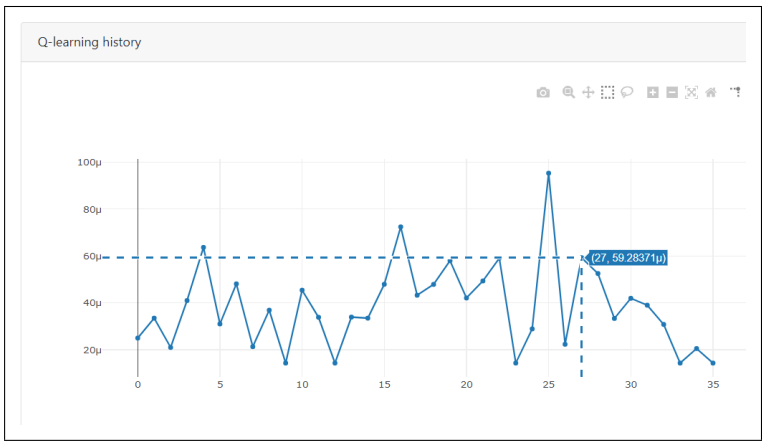
Dzięki bibliotece Plotly [11] okna pozwalają na interaktywną analizę danych, obserwacje przebiegu uczenia i zapis widoku do pliku graficznego.

4.7. Archiwizacja wyuczonych modeli

[10] Przechowywanie wyuczonych modeli działania na zasadzie eksportu do pliku tekstowego w formacie *Pickle*. Pickle jest biblioteką do archiwizacji, czyli zapisywania, przechowywania i ponownego wykorzystywania obiektów programów w języku *Python*.

Zapisanie do pliku odbywa się poprzez wykonanie polecenia :
pickle.dump(object).

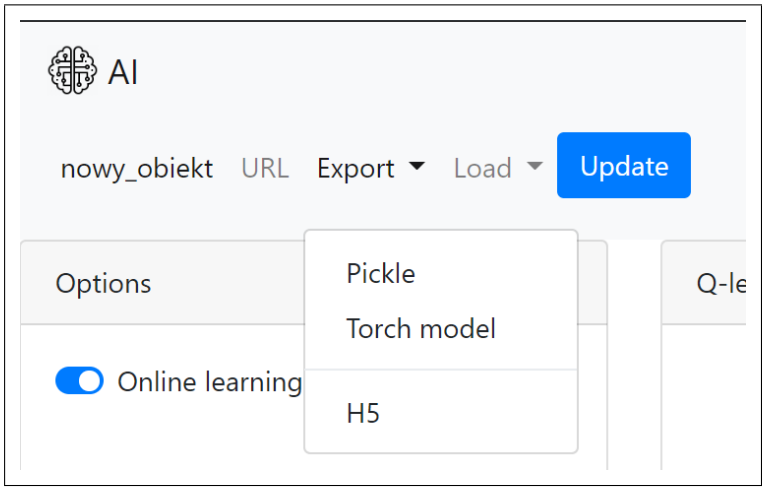
Rys. 4.8: Zmiana struktury sieci



Rys. 4.9: Przykładowy wykres

```
pickle.dump(object)
```

Obiekt zostaje dzielony na składowe zmienne wszystkich podstawowych typów języka *Python*. Następnie stworzony z nich słownik zapisywany jest w bezpieczny, zaszyfrowany wewnętrznie plik tekstowy formacie JSON.

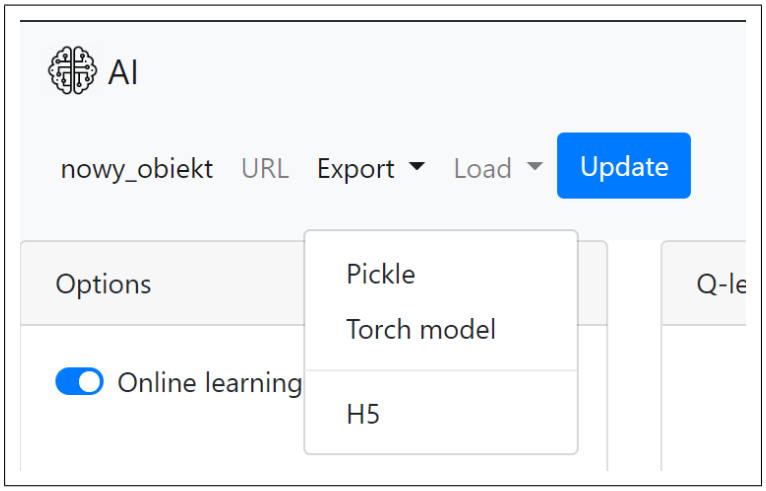


Rys. 4.10: Zapis modelu do pliku

Import, czyli wczytanie obiektu z pliku wykonuje się poprzez polecenie:

```
pickle.load(file)
```

Gdzie w poleceniu, *file* jest strumieniem wczytującym lub ciągiem znaków. Jako wynik operacji zostaje zwrócony model z wszystkimi informacjami i polami jak, zapisany wcześniej do pliku.



Rys. 4.11: Wczytanie modelu z pliku

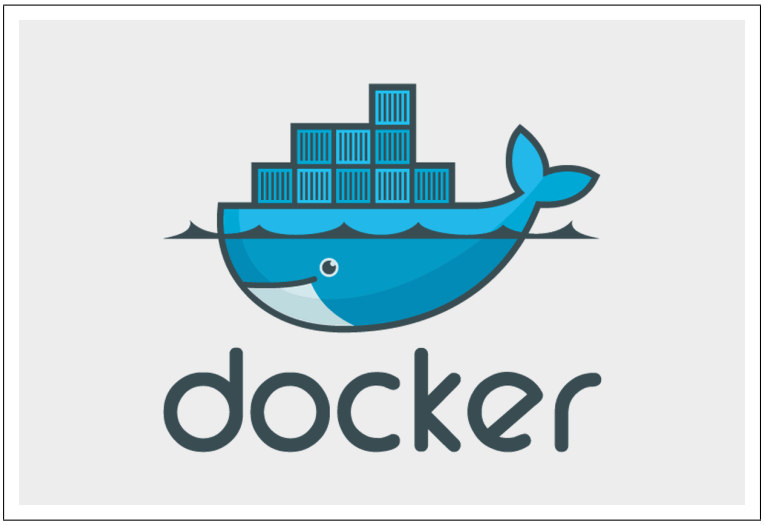
4.8. Zautomatyzowane wdrożenie przy pomocy obrazu platformy Docker

[7] Kontener ma służyć szybkiemu uruchomieniu serwisu na dowolnej maszynie z obsługiwaną platformą Docker.

Istotą kontenera jest korzystanie z gotowych elementów (innych kontenerów) tj.

- system Linux
- wirtualne środowiska języka Python

- instancje baz danych
- serwisy i usługi sieciowe



Rys. 4.12: Logo platformy Docker `https://pl.euro-linux.com/wp-content/uploads/Docker_w_Linuksie.png` [dostęp dnia 20 października 2019]

Na potrzeby serwisu do spraw AI. Został użyty:

- gotowy kontener z systemu Ubuntu w wersji 18.04, wraz z interpreterem języka Python w wersji 3
- wirtualne środowisko Conda

Zawartość pliku `Dockerfile` w folderze aplikacji webowej:

```
FROM continuumio/miniconda:latest

WORKDIR /home/ai

COPY environment.yml ./
RUN conda env create -f environment.yml

RUN echo "source activate ai" > ~/.bashrc

ENV PATH /opt/conda/envs/ai/bin:$PATH

COPY . ./app
COPY run.py ./

EXPOSE 5000

ENTRYPOINT  ["python3"]

CMD ["run.py"]
```

Po utworzeniu zaplecza systemowego dla serwisu, kolejne czynności wykonywane przez plik Dockerfile to

1. Zmiana katalogu roboczego na dedykowany folder dla projektu w systemie
2. Skopiowanie pliku environment.yml do katalogu projektu
3. Tworzenie środowiska Conda zgodnie z wymaganiami wykorzystywanych bibliotek zdefiniowanych w pliku environment.yml
4. Aktywacja utworzonego wirtualnego środowiska Conda o nazwie *ai*
5. Dodanie folderu conda/bin do ścieżki systemowej
6. Skopiowanie zawartości projektu do utworzonego folderu
7. Wystawienie portu (5000) jako dedykowany dla serwisu, w celu widoczności przez system usługi kontenera
8. Uruchomienie aplikacji poprzez wykonanie skryptu języka Python *run.py*

4.8.1. Sposób uruchomienia

W celu utworzenia kontenera z działającym serwisem, należy zainstalować na swoim komputerze pakiet *docker* oraz *docker-compose*. Następnie przejść do głównego katalogu projektu i wykonać dwie komendy:

```
sudo docker-compose build
sudo docker-compose up
```

Pierwsza buduje kontener według pliku docker-compose, natomiast druga uruchamia go.

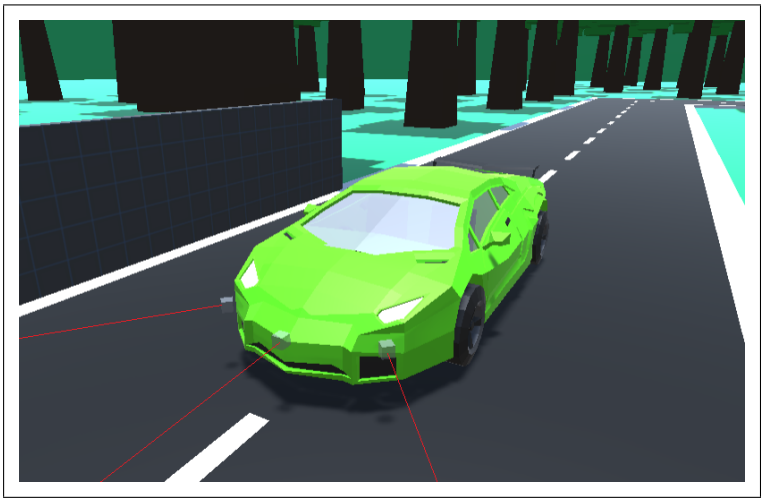
Rozdział 5

Symulacje rozwiązań problemów zagadnień sterowania obiektami

5.1. Uproszczony pojazd autonomiczny

5.1.1. Opis

Symulowanym obiektem będzie uproszczony pojazd autonomiczny. Pojazd, zwany dalej obiektem, wyposażony jest w trzy czujniki mierzące odległość. Elementem poddawanym sterowaniu jest kierunek w jakim pojazd będzie się poruszał. Każdy z elementów symulowany jest w środowisku Unity. Podejmowanie decyzji oddelegowane zostanie serwisowi na platformie *AI*, wykorzystujący sieć typu wielowarstwowy perceptron 2.1.2 o strukturze 5.1.



Rys. 5.1: Wygląd obiektu - pojazd autonomiczny

Na podstawie zebranych informacji według wzoru 5.1, liczona jest odległość d która stanowi jedną ze składowych wektora stanu $state$

$$d = |P_0 - P_{hit}| \tag{5.1}$$

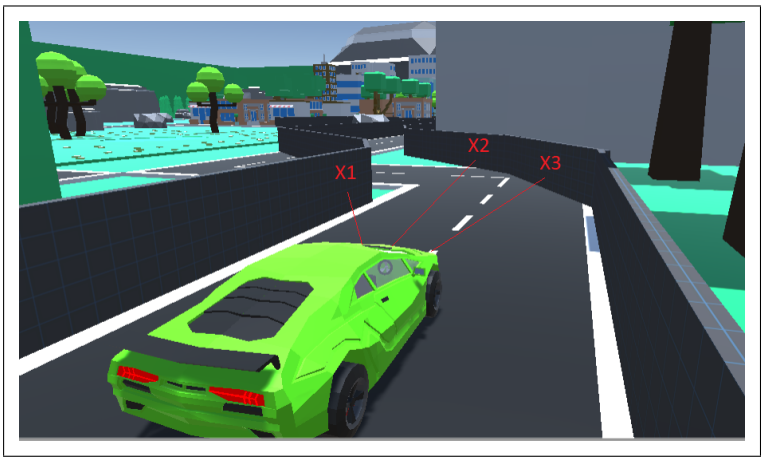
Gdzie P_0 oznacza miejsce położenia czujnika laserowego, a P_{hit} punkt w którym laser napotkał ciało obce. Kod realizujący wymienione wyliczenia, wykonywany jest w każdej odświeżanej klatce prze system. Jedna z podpiętych do pojedynczego czujnika funkcji 5.1 prezentuje następującą składnię.

```
IEnumerator Laser()
{
    while (true)
    {
        Ray ray = new Ray(transform.position, transform.forward);
        RaycastHit hit;
        line.SetPosition(0, ray.origin); // set laser on startup position

        if (Physics.Raycast(ray, out hit, 100))
        {
            // hit
            line.SetPosition(1, hit.point);
            var distance3d = transform.position - hit.point;
            distance = distance3d.magnitude;
        }
        else
            // move laser forward
            line.SetPosition(1, ray.GetPoint(100));

        yield return null;
    }
}
```

Listing 5.1: Mierzenie odległości - pojazd autonomiczny



Rys. 5.2: Stan obiektu - pojazd autonomiczny

Zebrane w taki sposób dane z trzech czujników tworzą wektor stanu $state$.

[4]Obiekt posiadając tak przygotowaną informacje, wydaje zapytanie do serwisu, jaką powinie wykonać akcję. sytuacji pojazdu na drodze. Akcje jakie obiekt podejmuje to:

- Skręt w lewo
- Jazda prosto
- Skręt w prawo

[25]Każda z możliwych przyjmuje zasięg od 0 do 1. Kierunek jazdy wyznaczany jest poprzez wynik operacji $action_{lewo} - action_{prawo}$. Natomiast Wyznaczenie następnego położenia obiektu wykonywane jest na podstawie wzoru 5.2. Przesunięcie obiektu wyznacza się przez wektor położenia, rotacje (y modelu) stałej prędkość,oraz przyrost czasu.

$$\vec{P}_{n+1} = \vec{P}_n + \vec{V}(\vec{R}_n + \vec{R}_{action})\Delta t \tag{5.2}$$

Zadaniem obiektu jest osiągnięcie pozycji docelowej, widocznej na mapie 5.3. Rozpoczynając trasę z pozycji A, zapamiętywany jest punkt początkowy, który może posłuży do obliczania nagrody. Nagroda rozumiana jest jako część pokonanej drogi, która dzieli obiekt do celu w pozycji początkowej, w taki celu stosowany jest wzór 5.3.

$$R = 1 - \frac{d_0 - d}{d_0} \tag{5.3}$$

Jednak przedstawione podejście do problemu oceny jakości sieci wydaje się nie być jednoznacznie. Jak można zauważyć na mapie 5.3. W miarę pokonanej odległości przez obiekt wartość funkcji zysku (nagrody) nie byłaby w porost proporcjonalna. W sytuacji pokonywania trasy nie będącej w linii prostej do celu, odległość do niego będzie maleć mimo pokonywania dalszej trasy. Na potrzeby zadania w którym obiekt przemieszcza się ze stałą, niezmienną prędkością. Wystarczającym kryterium wydaje się być czas całkowity w jakim obiekt przemiesza od początku trasy. Wartość nagrody będzie zatem rozumiana jako następujące wyrażenie 5.4.

$$R = T_{now} - T_0 \tag{5.4}$$

Rys. 5.3: Mapa otoczenia - pojazd autonomiczny

Rys. 5.4: Cel obiektu - pojazd autonomiczny

Wejścia	Warstwa ukryta 1	Warstwa ukryta 2	Wyjścia
3	10	10	3

Tab. 5.1: Struktura sieci - pojazd autonomiczny

5.1.2. Przebieg uczenia

Zbiorem czynności wykonywanych przez obiekt oraz program sterujący mają charakter cykliczny. W takim procesie używany jest konkretny model wystawiony przez serwis. Całość cyklu przebiega wedle następującego schematu:

Faza 1 Inicjalizacja

1. Ustawienie obiektu w pozycji początkowej
2. Zapamiętanie czasu początkowego

Faza 2 Praca obiektu Wykonywanie pętli:

1. Ustalenie *stanu* obiektu
2. Zapytanie serwisu o *akcję* do podjęcia na podstawie *stanu*
3. Wykonanie *akcji*

Faza 3 Zakończenie pracy

1. Pomiar przyrostu czasu
2. Obliczenie nagrody
3. Wysłanie podsumowania pracy do serwisu

Do wyznaczenia modelu sterującego pracą pojazdu autonomicznego, wybrano metodę genetyczną , bez mechanizmu q-learningu [30]. Ze względu na trudną do określenia wartość podjętej akcji przez obiekt, algorytm *Q-learningu* mógłby być w najlepszym razie nie opłacalny. W tym konkretnym przypadku stosując losowe mutacje i selekcje najlepszych modeli sterowania, serwis jest w stanie wyznaczyć optymalne wagi sieci.

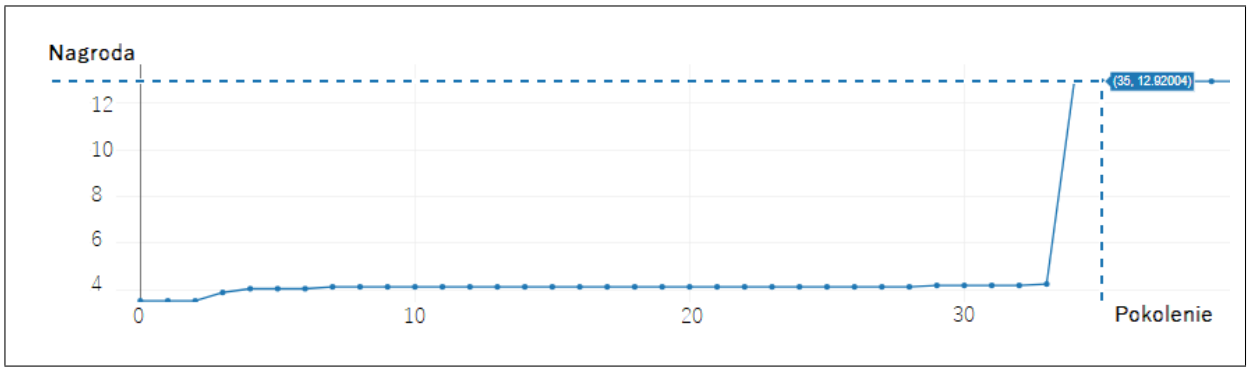
Szybkość uczenia	Prawdopodobieństwo mutacji	Siła mutacji	Rozmiar populacji	Ilość dzieci*
1E-07	1E-2	1E-3	32	4

Ilość dzieci - ilość potomków jaka powstaje z krzyżowania jednej pary

Tab. 5.2: Parametry uczenia - uproszczony pojazd autonomiczny

5.1.3. Wyniki

Wyniki uczenia prezentuje następujący wykres nagród.



Rys. 5.5: Wyniki uczenia - pojazd autonomiczny

Obliczenia wykonywano na komputerze o specyfikacji:

- CPU: 2.6GHz Intel Core i7-6700HQ (quad-core, 6MB cache, up to 3.5GHz with Turbo Boost)

- GPU Nvidia GeForce GTX 960M (4GB DDR5 VRAM)

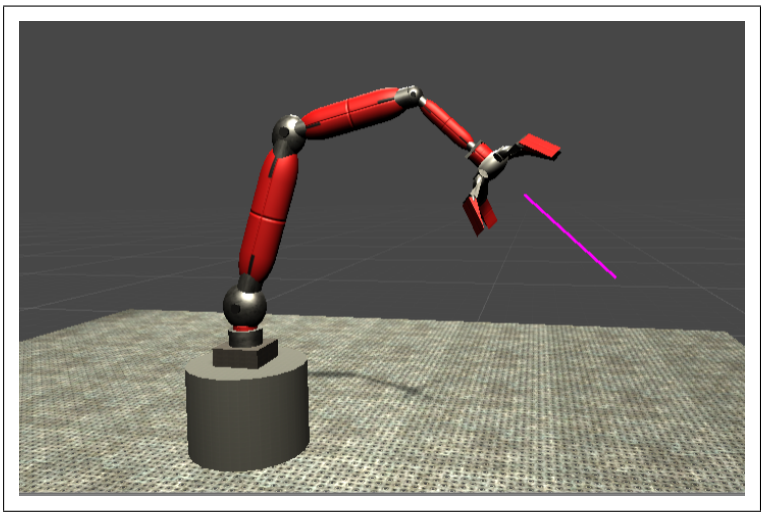
Nauka modelu sterowania zajęła 35 pokoleń w populacji 32 osobników. W systemie jednoagentowym (jedna symulacja w tym samym czasie) był to czas mieszczący się w przedziale 18-20 minut.

Adres do materiałów filmowych: `https://www.youtube.com/playlist?list=PLxS-l8aR8N2F7TcbeynxSaYDBNRCzwRzM`

5.2. Manipulator przemysłowy

5.2.1. Opis

Rolę obiektu w opisywanym zadaniu pełni manipulator przemysłowy. Zadaniem manipulatora jest przeniesienie przedmiotu z taśmy do skrzyni. W symulacji rolą sztucznej inteligencji będzie wysterowanie przegubów manipulatora tak, aby obiekt wykonał powierzone mu zadanie. Składa się on z czterech przegubów oraz efektora w postaci chwytaka wraz z systemem detekcji. Dzięki temu posiada 4 stopnie swobody w konfiguracji RRRR. Rolą systemu detekcji jest wykrycie, czy w zasięgu chwytaka znajduje się przedmiot.



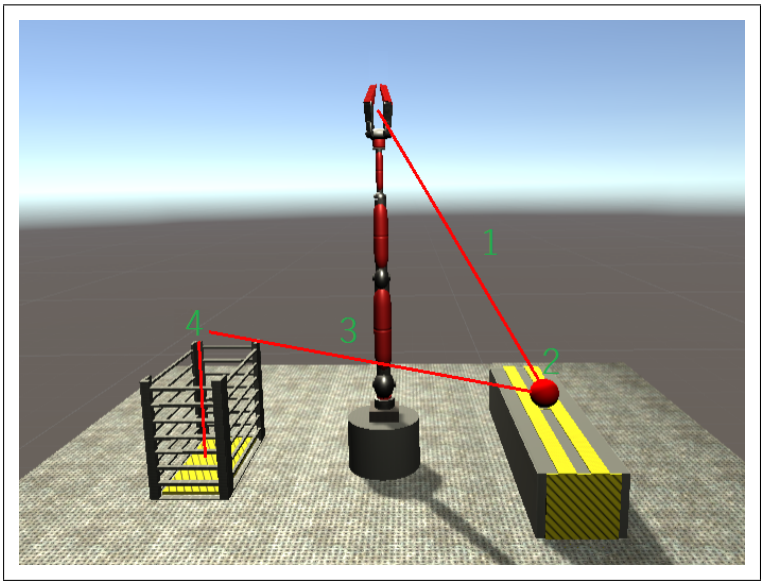
Rys. 5.6: Wygląd obiektu - manipulator przemysłowy

Każdy z przegubów oraz efektor osiąga wskazaną mu pozycję w zakresie od 0 do 1 z marginesem bezpieczeństwa między wartością minimalną i maksymalną 0.01 w celu uniknięcia niepożądanych błędów mechanicznych. W opisywanym zadaniu manipulator znajdzie zastosowanie w transporcie przedmiotów z taśmy produkcyjnej do kosza z produktami.

Rysunek 5.7 przedstawia schemat pracy manipulatora, którą można podzielić na 4 etapy:

Etap 1 Zbliżenie się **chwytaka** do **przedmiotu**
Manipulator z pozycji początkowej ma za zadanie zmienić swoje ustawienie tak, aby umożliwić chwycenie przedmiotu w następnym etapie. W tym celu, korzystając ze sztucznej inteligencji, serwis ustala odpowiednią konfiguracje przegubów. Nagroda obliczana jest na podstawie wzoru (5.5).

Etap 2 Zaciśnięcie się **chwytaka**.
Realizowane jest automatycznie poprzez wbudowany system detekcji przedmiotu. Po wykryciu przedmiotu w zasięgu chwytaka detektor zmienia swój stan z 0 na 1. Tym samym następuje zaciśnięcie się chwytaka i chwycenie obiektu.



Rys. 5.7: Zadanie do wykonania - manipulator przemysłowy

Etap 3 Przeniesienie **przedmiotu** nad **skrzynię**
W tym celu, korzystając ze sztucznej inteligencji, serwis ustala odpowiednią konfiguracje przegubów. Nagroda obliczana jest na podstawie wzoru (5.5).

Etap 4 Opuszczenie **przedmiotu** przez **chwytak**
Polega na opuszczeniu przedmiotu przez chwytak. Wykonanie zadania przeniesienia skutkuje zmianą wartości zadanej na zaciskach chwytaka z 1 na 0.

$$nagroda = 1 - \frac{d_0 - d}{d_0}, \quad \text{gdzie} \quad d \quad \text{jest odległością obiektu do pozycji docelowej} \tag{5.5}$$

Każdą ze składowych nagrody całkowitej oblicza się na zasadzie osiągnięcia pozycji docelowej w odniesieniu do pozycji początkowej, czyli w chwili wstąpienia w dany stan. Nagroda przyjmuje zatem wartość 0 w pozycji początkowej oraz bliską 1, jeśli etap zostanie wykonany.

Nagroda całkowita poddawana ocenie przez algorytmy uczące, wyznaczana jest według wzoru (5.6), jest to średnia z dwóch nagradzanych etapów (1 i 3).

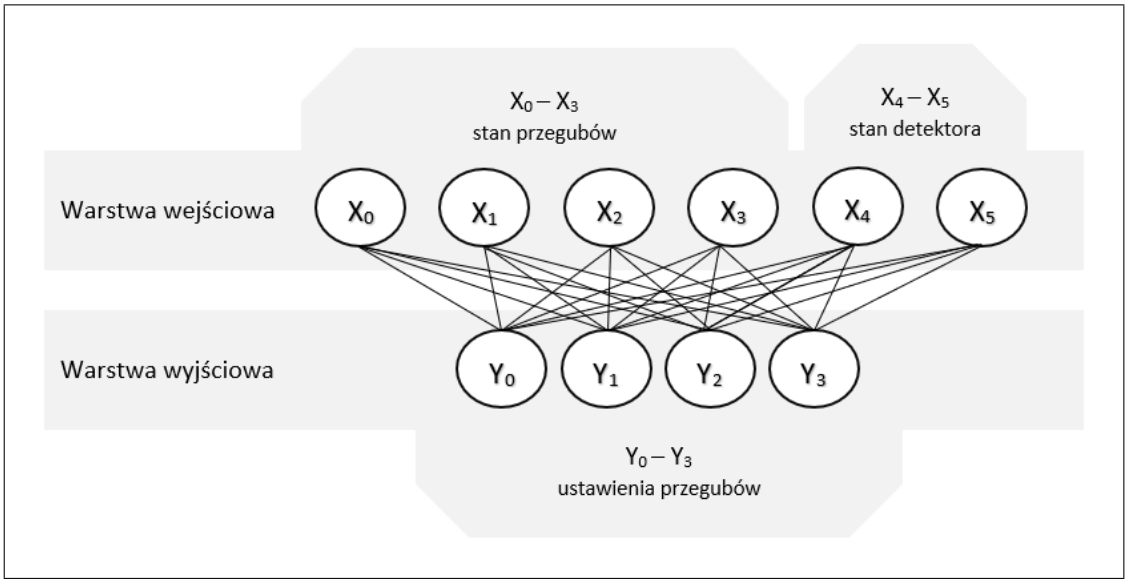
$$nagroda_{całkowita} = \frac{nagroda_{etap1} + nagroda_{etap3}}{2} \tag{5.6}$$

Dodatkowo symulacja przewiduje zastosowanie kary. Jest ona rozumiana jako ujemna nagroda. Jeśli obiekt uderzy w ciało obce, nie będące celem decyzja oznaczona jest z nagrodą wynoszącą -1, czyli karą za złą decyzje.

Schemat 5.8 przedstawia strukturę sieci typu wielowarstwowy perceptron wykorzystywaną przez serwis. W opisywanym przykładnie użyto warstwy wejściowej i wyjściowej, bez warstw ukrytych. Na wejście sieci podawane są aktualne stany ustawień przegubów od x_0 do x_3 oraz stan detektora przedmiotu x_4 (brak przedmiotu) i x_5 (wykryto przedmiot). Odpowiedzią sieci jest konfiguracja przegubów które sterowany manipulator ma aktualnie przyjąć.

5.2.2. Przebieg uczenia

W celu zwiększenia efektywności, zastosowany został system wieloagentowy [21]. Porównując problem manipulatora do uczenia uproszczonego pojazdu autonomicznego 5.1. Konieczne było



Rys. 5.8: Struktura sieci - manipulator przemysłowy

zastosowanie algorytmu *Q-learningu*. Pojedyncze akcje obiektu są łatwe do ocenienia. Każda oczekiwana akcja oznacza zwiększenie nagrody całkowitej, a tym samym nie istnieje problem niejednoznaczności w ocenie stanu jak w przykładzie poprzednim. Połączenie podejścia genetycznego [29] gwarantuje kompleks niejednorodnych danych typu $stan, nagroda$. Metodą sumowanych gradientów funkcji strat każdego osobnika. Wyznaczany jest kierunek spadku dla uczonego modelu. Opisany proces przedstawia poniższy schemat.

Faza 1 Inicjalizacja

1. Ustawienie obiektu w pozycji początkowej
2. Ustawienie przedmiotu w pozycji początkowej

Faza 2 Praca obiektu Wykonywanie pętli:

1. Ustalenie *stanu* obiektu
2. Obliczenie nagrody i ustalenie etapu zadania
3. Zapytanie serwisu o *akcję* do podjęcia na podstawie *stanu*
4. Wysłanie danych na temat poprzedniego stanu i przyrostu nagrody
5. Wykonanie *akcji*

Faza 3 Zakończenie pracy

1. Obliczenie nagrody
2. Wysłanie podsumowania pracy do serwisu

Znaczenie *Q-learningu* okazało się być kluczowe w wychodzeniu z minimum lokalnych. Podstawowe podejście genetyczne bywa problematyczne jeśli model nie jest w stanie osiągnąć lepszego wyniku w zasięgu ograniczonych co do swej siły mutacji. Zbierając gradient z każdego osobnika dostajemy informację w którym kierunku powinny zmierzać zmiany w wagach, a tym samym przyszłe mutacje w następnych pokoleniach. W tym celu, najlepszy bieżący model po dokonaniu wstecznej propagacji błędu, dodawany jest do populacji. Zastosowane zostały następujące parametry uczenia 5.3

Szybkość uczenia	Prawdopodobieństwo mutacji	Siła mutacji	Rozmiar populacji	Ilość dzieci
1E-07	1E-1	5E-2	64	4

Tab. 5.3: Parametry uczenia - manipulator przemysłowy

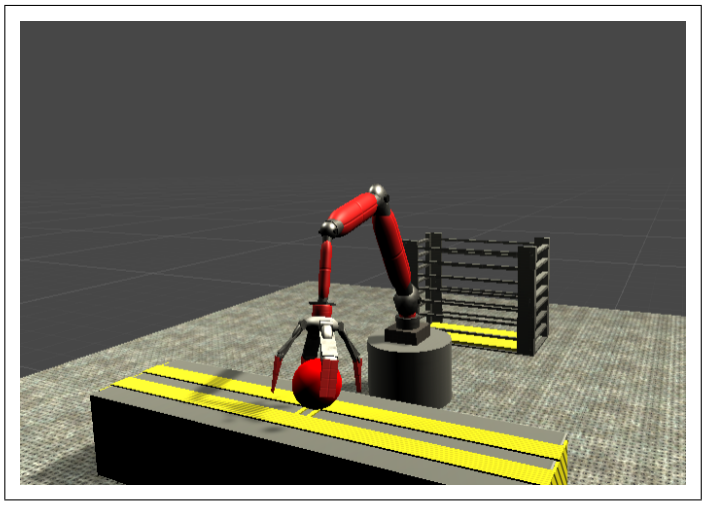
5.2.3. Wyniki

Obliczenia wykonywano na komputerze o specyfikacji:

- CPU: 2.6GHz Intel Core i7-6700HQ (quad-core, 6MB cache, up to 3.5GHz with Turbo Boost)
- GPU Nvidia GeForce GTX 960M (4GB DDR5 VRAM)

Nauka modelu sterowania zajęła 51 pokoleń w populacji 64 osobników. Dzięki zastosowaniu systemu wieloagentowego (wiele symulacji w tym samym czasie), mimo większej złożoności zadania niż miało to miejsce przy samochodzie autonomicznym 5.1, czas potrzebny na wykonanie rozwiązanie zadania wyniósł 14-15 minut.

Pierwszym zadaniem algorytmów uczących było osiągnięcie położenia umożliwiającego podniesienie przedmiotu. **Etap 1** został pierwszy raz wykonany prawidłowo w 8 pokoleniu, co wynika z osiągnięcia wartości nagrody > 0.5. Efekt przedstawia ilustracja 5.9

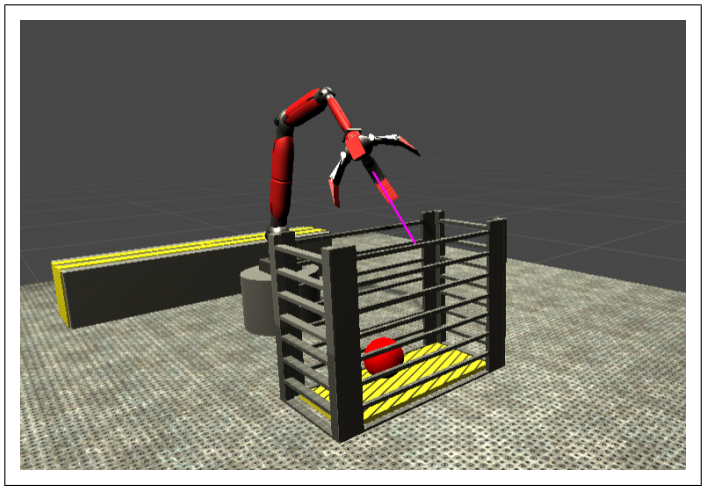


Rys. 5.9: Etap 1 - manipulator przemysłowy

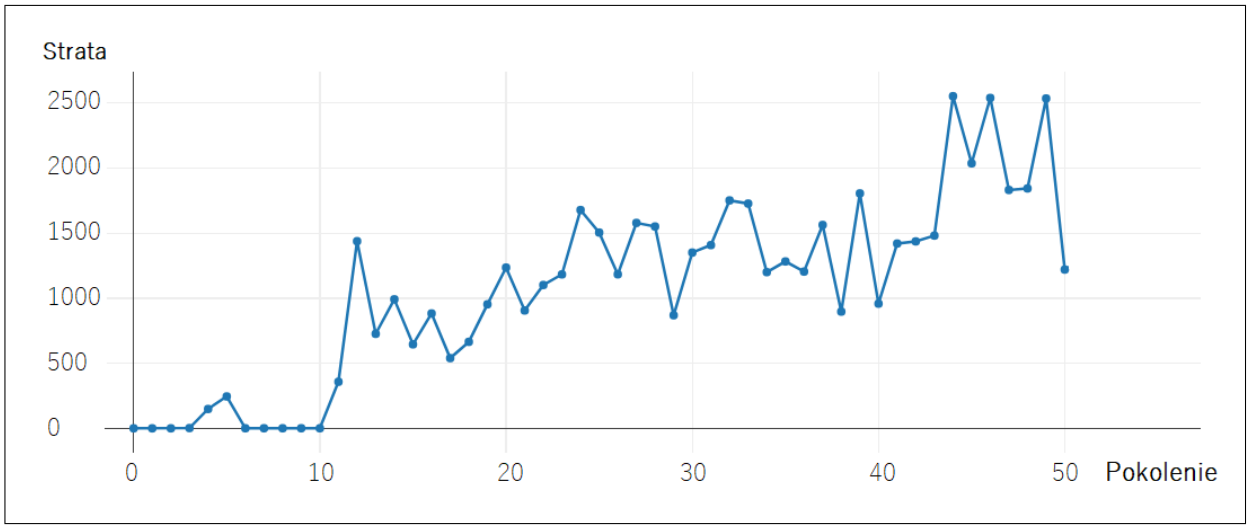
Ostatnim krokiem był **Etap 3**, czyli przeniesienie przedmiotu z taśmy nad kosz, będący miejscem docelowym. Konieczność wykonania *Etapu 1*, aby model mógł przystąpić do próby w *Etapie 3* skutkowała dłuższym czasem uczenia. Po 50 pokoleniu w populacji znalazł się pierwszy model realizujący ostatni etap 5.10, a tym samym całe zadanie. Oznacza to, że został wyznaczony ostateczny optymalny model sterowania obiektem.

Zebrane dane statystyczne w czasie uczenia posłużyły do wygenerowania wykresów łącznej sumy błędów w populacji 5.11 oraz całkowitej nagrody 5.12.

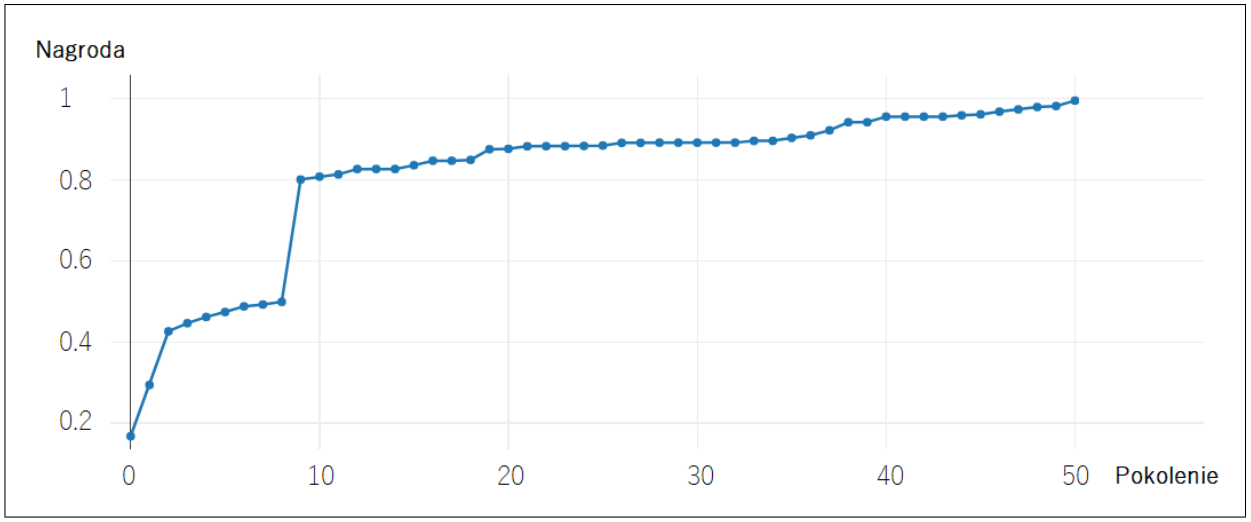
Postęp w procesie uczenia trwał 51 generacji. W związku z podziałem zastosowania modelu na dwa zadania na wykresie całkowitej nagrody 5.12 zauważalne są dwie asymptoty. Pierwsza zmierza ku wartości 0.5, przy czym druga ku wartości końcowej 1. Otrzymany kształt wykresu nagrody zgadza się z wcześniejszymi oczekiwaniami. W przypadku wykresu łącznej sumy błędów w populacji 5.11, wartość nie osiągnęła zamierzonego 0. Oznacza to że model był bardzo wrażliwy na zmiany w jego wagach, co jest skutkiem mało rozbudowanej struktury sieci oraz współzależnemu zastosowaniu w podnoszeniu i przenoszeniu.



Rys. 5.10: Etap 4 - manipulator przemysłowy



Rys. 5.11: Suma funkcji strat - manipulator przemysłowy



Rys. 5.12: Wyniki uczenia - manipulator przemysłowy

Istotne z punktu widzenia dynamiki zmian są za to miejsca w których wartość spada względem poprzedniej populacji oraz wzrasta. Analizując dwa wykresy jednocześnie można zauważyć dwie ciekawe zależności:

- Wraz ze spadkiem sumy strat następuje wzrost nagrody całkowitej. Jest to jednoznaczne z wyjściem z tzw. minimum lokalnego.
- Suma strat wzrasta, jeśli nie następuje znaczący wzrost nagrody całkowitej. Wynika to z nakładających się na siebie mutacji w populacji osobników. Tak długo jak żaden nie wyróżni się z pozostałych większą nagrodą silniejsze mutacje oznaczają więcej popełnianych błędów i więcej znaczących danych do uczenia.

Adres do materiałów filmowych: `https://www.youtube.com/playlist?list=PLxS-l8aR8N2F7TcbeynxSaYDBNRCzwRzM`

Rozdział 6

Podsumowanie

Przedstawione zagadnienia sztucznej inteligencji posłużyły na opracowanie metody sterowania obiektami w symulowanym środowisku Unity. Uczenie przez wzmacnianie jest sposobem na uporanie się z problemem braku danych uczących. Dostarczając zsynchronizowane środowisko Unity, sztuczna inteligencja jest w stanie nauczyć się kontrolować obiekty. Wytrenowane w ten sposób mogą posłużyć do opracowania prototypów rzeczywistych urządzeń i douczenia ich w realnym świecie.

Aby sztuczna inteligencja mogła poradzić sobie z powierzonym jej zadaniem konieczne jest zapewnienie niezbędnych informacji o stanie otoczenia. Jak można zauważyć na przykładzie projektu *pojazdu autonomicznego* genetyczne algorytmy uczące byłby w stanie wyznaczyć optymalny model sterowania. Oznacza to, że realizacja procesu uczącego została wykonana prawidłowo, a dane o otoczeniu w formie stanu były wystarczające do nauki modeli. Przykład *manipulatora przemysłowego* został wzbogacony o mechanizm *Q-learningu*. Metody genetyczne i uczące doprowadziły obiekt do ostatecznego położenia i umożliwiły wykonanie zadania.

Wizualizacja działania algorytmów sztucznej inteligencji, pozwala w efektowny sposób zaprezentować drzemiący potencjał w dziedzinie AI i ML. Uniwersalna platforma ucząca okazuje się być doskonałym narzędziem do prototypownia obiektów sterowanych. Dalszy rozwój projektu pozwoliłby na zwiększenie funkcjonalności o sieci rozpoznające obrazy oraz usprawnienie komunikacji w celu zwiększenia wydajności. Z całą pewnością na przestrzeni lat udział w tego rodzaju technologii zacznie usprawniać procesy wytwórcze w branży IT, przemysłowych i wielu innych.

Literatura

[1] https://aspolska.pl/cztery-typy-uczenia-maszynowego/ [dostęp dnia 13 października 2019].

[2] https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents [dostęp dnia 5 grudnia 2019].

[3] https://unity3d.com/public-relations [dostęp dnia 17 listopada 2019].

[4] A. Baldominos, Y. Sáez, G. Recio, F. Calle. *Learning Levels of Mario AI Using Genetic Algorithms*, wolumen 9422, strony 267–277. 11 2015.

[5] Z. Chen, Y. Zhou, X. He, S. Jiang. A restart-based rank-1 evolution strategy for reinforcement learning. strony 2130–2136, 08 2019.

[6] M. V. Copeland. *Deep Learning Explained*. 2019.

[7] Dokumentacja. Docker. https://docs.docker.com. [dostęp dnia 30 września 2019].

[8] Dokumentacja. Flask. https://palletsprojects.com/p/flask. [dostęp dnia 2 lipca 2019].

[9] Dokumentacja. Numpy. https://docs.scipy.org/doc/numpy. [dostęp dnia 1 lipca 2019].

[10] Dokumentacja. Pickle. https://docs.python.org/3/library/pickle. [dostęp dnia 20 listopada 2019].

[11] Dokumentacja. Plotly. https://plot.ly/python. [dostęp dnia 30 sierpnia 2019].

[12] Dokumentacja. Pytorch. https://pytorch.org. [dostęp dnia 1 lipca 2019].

[13] T. Geijtenbeek, M. van de Panne, A. F. van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32(6), 2013.

[14] A. Greenwald, K. Hall. Correlated-q learning. 02 2004.

[15] L. Jin, S. Li, J. Yu, J. He. Robot manipulator control using neural networks: A survey. *Neurocomputing*, 285:23 – 34, 2018.

[16] I. John, C. Kamanchi, S. Bhatnagar. Generalized speedy q-learning, 2019.

[17] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, D. Lange. Unity: A general platform for intelligent agents, 2018.

[18] S. Kenneth, M. Risto. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10:99–127, 02 2002.

[19] S. Khadka, S. Majumdar, S. Miret, S. McAleer, K. Tumer. Evolutionary reinforcement learning for sample-efficient multiagent coordination, 2019.

[20] D. L., S. E. Słownik Języka Polskiego PWN, 2019.

[21] M. Lauer, M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. *In Proceedings of the Seventeenth International Conference on Machine Learning*, strony 535–542. Morgan Kaufmann, 2000.

[22] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data, 2016.

[23] Z. Michalewicz. *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Warszawa: Wydawnictwa Naukowo-Techniczne, 1996.

[24] M. Minsky, S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015.

[26] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[27] M. Parsell. Steven m. platek, julian paul keenan and todd k. shackelford (eds), evolutionary cognitive neuroscience. *Minds and Machines*, 19:275–278, 05 2009.

[28] D. Pelvig, H. Pakkenberg, A. Stark, B. Pakkenberg. Neocortical glial cell numbers in human brain. *Neurobiology of aging*, 29:1754–62, 06 2007.

[29] A. Sallab, M. Abdou, E. Perot, S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, Jan 2017.

[30] A. Sallab, M. Abdou, E. Perot, S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, Jan 2017.

[31] S. Sharma. Artificial neural network in machine learning, 08 2017. `https://www.datasciencecentral.com/profiles/blogs/artificial-neural-network-ann-in-machine-learning` [dostęp dnia 10 listopada 2019].

[32] StatSoft. Internetowy Podręcznik Statystyki, 1984-2011. `https://www.statsoft.pl/textbook/stathome_stat.html` [dostęp dnia 8 grudnia 2019].

[33] G. Tesauro, J. Kephart. Pricing in agent economies using multi-agent q-learning. *Autonomous Agent and Multi-Agent Systems*, 5, 10 1999.

[34] D. S. A. G. I. A. D. W. M. R. Volodymyr Mnih, Koray Kavukcuoglu. Playing atari with deep reinforcement learning, 2013.

[35] C. J. C. H. Watkins, P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[36] V. Zocca, G. Spacagna, D. Slater, P. Roelants. *Python Deep Learning*. Packt Publishing, 2017.

Dodatek A

Opis załączonej płyty CD/DVD

Płyta DVD zawiera następujące elementy:

- Kod źródłowy projektu AI, serwisu ds. sztucznej inteligencji
- Nagrania video, prezentujące działanie obiektów i procesy uczenia