**Akademia Finansów i Biznesu Vistula**

**Wydział Sztuki, Techniki i Komunikacji**

**Kierunek studiów Informatyka**

**Michał Bałtowski**

Numer albumu 45050

***Projekt i implementacja aplikacji wykorzystującej algorytm genetyczny w silniku Unity***

Praca inżynierska

napisana pod kierunkiem

doktora Piotra Zaremby

Warszawa 2022

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie (w zakresie wyszczególnionym we wstępie) i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Spis treści

[Streszczenie 3](#_Toc95042120)

[Wstęp 4](#_Toc95042121)

[Rozdział 1 Projekt 6](#_Toc95042122)

[Rozdział 2 Zagadnienia dotyczące sztucznej inteligencji 7](#_Toc95042123)

[2.1 Historia sztucznej inteligencji 7](#_Toc95042124)

[2.2 Podstawy sieci neuronowych 10](#_Toc95042125)

[2.3 Podstawy algorytmów ewolucyjnych 13](#_Toc95042126)

[Rozdział 3 Użyte narzędzia i technologie 16](#_Toc95042127)

[3.1 UNITY 16](#_Toc95042128)

[3.2 C# 18](#_Toc95042129)

[Rozdział 4 Implementacja 19](#_Toc95042130)

[4.1 Analiza problemu 19](#_Toc95042131)

[4.2 Zaimplementowanie środowiska w Unity 19](#_Toc95042132)

[4.3 Implementacja osobnika 21](#_Toc95042133)

[4.4 Stworzenie sieci neuronowych 25](#_Toc95042134)

[4.5 Implementacja zachowania sieci neuronowej 30](#_Toc95042135)

[4.6 Dodanie technik ewolucyjnych 34](#_Toc95042136)

[4.7 Implementacja mechanizmu zapisywania wyników 38](#_Toc95042137)

[Rozdział 5 Prezentacja wyników i pomysłów na rozwinięcie projektu/wykorzystanie 40](#_Toc95042138)

[5.1 Wyniki 40](#_Toc95042139)

[5.2 Pomysły na rozwój i wykorzystanie projektu 45](#_Toc95042140)

[Bibliografia 47](#_Toc95042141)

[Spis Ilustracji 48](#_Toc95042142)

[Spis kodów 48](#_Toc95042143)

# Streszczenie

W mojej pracy opisuje projekt i implementację aplikacji symulującej wyścig samochodowy. Celem symulacji jest by choć jeden samochód przejechał cały tor nie uderzając w żadną przeszkodę. Samochody biorące udział w wyścigu są sterowane przez sztuczne sieci neuronowe. Przetwarzają one sygnały odbierane przez sensory pojazdu na wartości, od których zależy prędkość i kierunek skrętu. Algorytm genetyczny szkoli całą populację osobników i generuje kolejne, lepiej przystosowane do przejechania toru osobniki.

Do celów mojej pracy inżynierskiej zaliczam:

* Implementacje sztucznej sieci neuronowej w celu kierowania pojazdami.
* Implementacje algorytmu genetycznego, który zapewni wystąpienie sytuacji w której w generacji n przynajmniej jeden osobnik dojedzie do mety

Do pracy załączam :

Projekt do uruchomienia w programie UNITY, instrukcje uruchomienia projektu, trzy pliki tekstowe z wynikami zarejestrowanymi przez program, plik excel z wynikami.

Wykaz skrótów

SI – Sztuczna inteligencja

Fitness – wartość funkcji przystosowania/ fitness value

Wstęp

Sztuczna inteligencja to wieloznaczne pojęcie. W informatyce oznacza tworzenie modeli i programów symulujących choć częściowo cechy ludzkiego umysłu, takie jak umiejętność rozumienia języka, rozpoznawania obrazów, rozwiązywania problemów i uczenia się. Przez uczenie się rozumiem zaprogramowanie programu w taki sposób by do pewnego stopnia był w stanie dostosować swoje zachowanie, analizując skutki wcześniejszych działań w celu osiągnięcia określonego przez twórcę celu. Aby to osiągnąć informatycy łączą wiedzę z takich dziedzin jak filozofia, psychologia, matematyka czy neurobiologia z wiedzą dotyczącą dziedziny nauki, która ma związek z konkretnym projektem np. ekonomia lub lingwistyka. Pierwsze efekty wyżej wspomnianych prac specjalistów zostały opublikowane już w latach 40. Tempo rozwoju sztucznej inteligencji przyśpieszyło za sprawą zwiększenia mocy obliczeniowej komputerów, nowych algorytmów oraz dużej ilości danych. Dzięki temu już dziś efekty pracy nad Si pojawiają się w życiu codziennym każdego z nas pod np. postacią algorytmów wspomagających aparaty w smartphonach lub sterujących tłumaczem. Sztuczna inteligencja jest wykorzystywana między innymi do: dostarczania spersonalizowanych rekomendacji reklamowych na podstawie historii wyszukiwania i zakupów w sklepach internetowych, do podpowiadania szukanej frazy w wyszukiwarkach internetowych, w oprogramowaniu tłumaczącym teksty pisane lub mówione oraz do automatycznego generowania napisów. Firmy stworzyły również oparte na sztucznej inteligencji mechanizmy do analizowania wpisów publikowanych na portalach społecznościowych i fraz wpisywanych do wyszukiwarki internetowej w celu oceny potencjału samobójczego[[1]](#footnote-1). Poza tym SI wykorzystywana jest także w wielu gałęziach gospodarki. W medycynie jest wykorzystywana np. do wykrywana objawów koronawirusa na podstawie obrazów tomografii komputerowej[[2]](#footnote-2). W dziedzinie transportu działa już pierwsza firma, która udostępnia usługi przewozu przez autonomiczne pojazdy, a na rynku już teraz najnowsze pojazdy posiadają zaawansowane systemy wspomagania kierowcy[[3]](#footnote-3) lub systemy wykrywające niebezpieczne sytuacje na drodze. W rozwoju technologii opartych na sztucznej inteligencji ludzie upatrują ogromny potencjał na rozwiązanie problemów w wielu dziedzinach takich jak rolnictwo, bezpieczeństwo, medycyna czy administracja, a Unia Europejska postrzega sztuczną inteligencje jako centralny element transformacji społeczeństwa[[4]](#footnote-4).

Powodem wyboru tematu pracy była chęć poszerzenia własnej wiedzy z zakresu sztucznej inteligencji, chęć tworzenia gier, doskonalenie umiejętności programistycznych oraz zainteresowanie tematyką sztucznych sieci neuronowych. Dlatego też by pogłębić swoją wiedzę w tym temacie oraz ze względu na jej doceniony na całym świecie potencjał na rozwiązanie niektórych problemów naszego świata podjąłem się w niniejszej pracy odwzorowania wyścigu, w którym będą brać udział samochody sterowane sztuczną inteligencją. Ich celem będzie przejechanie krętego toru w taki sposób by nie zderzyć się z żadną przeszkodą. Samochody powinny dokonać tego bez ingerencji programisty.

# **Rozdział 1 Projekt**

Projekt jaki realizuje w tej pracy inżynierskiej to odwzorowanie wyścigu pod postacią aplikacji. Jednak nie jest to wyścig rozumiany jako rywalizacja między uczestniczącymi samochodami, a między sztuczną inteligencją sterującą samochodami a czasem. Dlatego w dalszej części pracy opisując ten projekt będę posługiwać się również słowem symulacja, ponieważ projekt symuluje sytuacje, która wygląda jak wyścig między samochodami.

Za przywołaną sztuczną inteligencje posłuży sztuczna sieć neuronowa oraz algorytm genetyczny. Każdy samochód posiada własną sieć neuronową, która w czasie rzeczywistym przetwarza dane odbierane przez sensory umieszczone na jego krawędziach. Wynik wyjściowy sieci jest interpretowany przez samochód i przejawia się zmianą kierunku bądź poruszeniem do przodu. Algorytm genetyczny wykorzystałem jako czynnik mobilizujący grupę samochodów do nauki poruszania się po torze. Szczegółowym zadaniem algorytmu jest wygenerowanie populacji osobników. Następnie wybór najlepszych osobników, które dalej posłużą jako wzór do stworzenia kolejnej populacji. Aby utworzyć środowisko, w którym wyścig może się odbyć posłużyłem się Unity czyli silnikiem do tworzenia gier, animacji, a wszystkie skrypty napisałem w języku C#.

Struktura pracy wygląda następująco. W drugim rozdziale opisałem historię i genezę sztucznej inteligencji oraz jej obecne zastosowania. Zaprezentowałem też opis sieci neuronowych i algorytmu genetycznego. Rozdział trzeci zawiera opis narzędzi wykorzystanych do stworzenia aplikacji. W czwartym rozdziale przedstawiłem szczegółową implementacje najważniejszych funkcjonalności symulacji. Piąty rozdział poświęciłem analizie wyników działania symulacji oraz opisałem tam pomysły na rozwój i wykorzystanie aplikacji.

# **Rozdział 2 Zagadnienia dotyczące sztucznej inteligencji**

W tym rozdziale opisałem wybrane wydarzenia z historii sztucznej inteligencji, podstawowe zagadnienia z tematu sztucznych sieci neuronowych oraz podstawy algorytmów ewolucyjnych.

2.1 Historia sztucznej inteligencji

Pomysł stworzenia sztucznego tworu, który wyręczyłby twórcę w wykonywaniu zadania, mimo futurystycznych skojarzeń z filmami SCI Fi nie jest zagadnieniem nowym. Sama idea stworzenia autonomicznej maszyny, która byłaby zdolna do samodzielnego myślenia pojawiła się już w starożytności. Jednym z najpopularniejszych przykładów ukazujący chęć ożywienia sztucznego obiektu jest mit o Pigmalionie. Opowiada on o młodym królu który wyrzeźbił z kości słoniowej statuę kobiety idealnej. Statua okazała się tak piękna że król zakochał się w niej i prosił bogów o ożywienie swojego obiektu westchnień. Bogini Afrodyta spełniła jego prośbę dzięki czemu król zyskał prawdziwą, żywą towarzyszkę życia. W innych opowiadaniach Hefajstos bóg kowalstwa miał wielokrotnie tworzyć automatyczne roboty. Jednym razem były to posągi lwów i byków, które służyły jako strażnicy, innym razem złote stoły które zastawione jedzeniem przychodziły na wezwania bogów. Według podań Hefajstos stworzył również Talosa, olbrzyma z brązu którego zadaniem było strzec Krety przed cudzoziemcami. W średniowieczu idea stworzenia sztucznego bytu, który obdarzony byłby możliwością myślenia, nie zginęła. Alchemicy próbowali wytworzyć w swoich laboratoriach homunkulusa, a żydowska koncepcja ulepienia z gliny sługi golema, stała się w naszych czasach popularnym motywem fantastyki[[5]](#footnote-5).

Niestety przez brak możliwości technicznych dawne pomysły nie przybrały nigdy realnej namacalnej postaci. Dopiero rozwój techniki w XX w. umożliwił hiszpańskiemu inżynierowi Leonardo Torres y Quevedo stworzenie w 1914 El Ajedrecista (hiszp. Gracz szachowy). Był to pierwszy automat, który był w stanie samodzielnie rozgrywać końcówki szachowe i przy tym sygnalizować gdy przeciwnik wykonał niedozwolony ruch.

Wraz z postępem oraz zwiększającą się mocą obliczeniową komputerów rozwój sztucznej inteligencji przyśpieszał, a naukowcy pracujący nad SI opracowywali coraz to lepsze algorytmy i techniki. Poniżej wypiszę kilka wydarzeń w porządku chronologicznym, które stanowią kamienie milowe, pokazujące tempo oraz skale rozwoju sztucznej inteligencji

W 1943 roku neuropsychiatra Warren McCulloch i matematyk Walter Pitt publikują artykuł „A Logical Calculus of the Ideas Immanent in Nervous Activity” (Logiczne obliczenie idei immanentnych w aktywności nerwowej), który zostanie później uznany za jedną najważniejszych prac, które zapoczątkowały tworzenie sztucznej inteligencji. Autorzy formułują w nim pierwszą formalną definicje sztucznego neuronu opartą na uproszczonym modelu biologicznym. Istotnym elementem modelu jest sumowanie sygnałów wejściowych z odpowiednią wagą i poddanie otrzymanej sumy działaniu nieliniowych funkcji aktywacji. Sieci wykorzystujące ten model wywarły wpływ na późniejszy rozwój tej dziedziny[[6]](#footnote-6).

1949 rok, psycholog Donald Oldigin Hebb w trakcie swoich badań zauważył, że połączenie pomiędzy dwiema komórka jest wzmacnianie, jeżeli w tym samym czasie obie komórki stają się aktywne[[7]](#footnote-7). Jako pierwszy zaproponował metodę uczenia sieci polegającą na zmianach wag połączeń pomiędzy neuronami. Metoda ta znana jest dzisiaj jako reguła Hebba.

1950 rok matematyk Alan Turing zaproponował „sposób określenia [zdolności](https://mfiles.pl/pl/index.php/Zdolno%C5%9Bci) maszyny do posługiwania się językiem naturalnym i pośrednio mającym dowodzić opanowania przez nią [umiejętności](https://mfiles.pl/pl/index.php/Umiej%C4%99tno%C5%9Bci) myślenia w sposób podobny do ludzkiego”[[8]](#footnote-8) zwany testem Turinga. Test polega na rozmowie między sędzią a człowiekiem i sztuczną inteligencją. Sędzia zadaje dowolne pytanie jednak nie wie, który adresat w danej chwili będzie udzielać odpowiedzi. Aby test nie był zbyt łatwy dialog jest przeprowadzany w formie tekstowej. Testowana sztuczna inteligencja ma na celu w takim sposób odpowiadać na pytania zadane przez sędziego, by ten nie był w stanie określić czy rozmawia z człowiekiem, wtedy też można powiedzieć że zdała test.

W 1951 roku Marvin Minsky i Dean Edmonds tworzą pierwszą sztuczną sieć neuronową nazwaną SNARC (ang. Stochastic Neural Analog Reinforcement Computer). Sieć zawierała 40 neuronów i była w stanie dopasowywać wagi połączeń między neuronami. Cały system miał za zadanie wyjść z labiryntu imitując zachowanie szczura.

1956 rok w Dartmouth College odbyła się konferencja na której uczestnicy mieli zastanowić się nad rozwojem inteligentnych maszyn. To wtedy organizator konferencji, informatyk John McCarthy jako pierwszy w historii użył pojęcia Artificial Inteligence(AI) (Sztuczna inteligencja). Wydarzenie to uznawane jest za moment, w którym sztuczna inteligencja stała się dziedziną badań naukowych[[9]](#footnote-9).

1966 informatyk prof. Joseph Weizenbaum napisał program ELIZA, był to jeden z pierwszych programów zdolny do podjęcia próby zaliczenia testu Turinga. Program był jednym z pierwszych chatbotów, czyli programów, których zadaniem jest prowadzić konwersacje. Do działania potrzebował skryptu, w którym zakodowane były słowa kluczowe wraz z informacją określającą jak są ważne oraz instrukcje jak na nie odpowiedzieć. Na podstawie skryptu ELIZA symulowała rozmowę, dzięki przestawianiu słów i zmiany słów kluczowych[[10]](#footnote-10).

1973 za sprawą lekarza i informatyka Edwarda Shortliffa powstaje MYCIN czyli jeden z pierwszych systemów eksperckich do diagnostyki leczenia zakażeń bakteryjnych opartych na sztucznej inteligencji. System by pomagać lekarzom właściwie stosować antybiotyki został wyposażony w bogatą bazę danych i finalnie zaprogramowane 600 reguł stworzone przez konsylium lekarskie. MYCIN operował dużą liczbą niezwiązanych ze sobą faktów, potrafił wykorzystać dane nieścisłe i niekompletne, a także był w stanie uzasadniać rady, których udzielał lekarzom[[11]](#footnote-11).

W 1989 badacz Yann Adne LeCun tworzy nowy rodzaj sieci neuronowej zwany ConvNet lub CNN (Convolutional Neural Network), której specjalnym zastosowaniem jest rozpoznawanie wzorów wizualnych bezpośrednio z pikseli obrazu. CNN swą nazwę zawdzięcza operacji matematycznej nazywanej konwolucja lub splotem. Warstwa w sieci, która wykorzystuje tą operację jest głównym elementem sieci. Systemów, którym dały początek badania LeCuna, wciąż używa się w bankach do odczytywania czeków.

W 2004 roku DARPA (amerykańska agencja rządowa zajmująca się rozwojem technologii wojskowej) zorganizowała wyzwanie na stworzenie pojazdu, który samodzielnie przejedzie 240 kilometrową wyznaczoną trasę na pustyni Mojave. Żaden z pojazdów zgłoszonych do wyzwania nie dojechał do mety, a pojazd który dojechał najdalej przebył zaledwie 12 kilometrów. Rok później zespół badaczy pod kierownictwem prof. Sebastiana Thruna wystawił do drugiej edycji wyzwania pojazd o nazwie Stanley. Stanley jako pierwszy dojechał do mety z najlepszym czasem. Poza Stanleyem do mety dojechało 5 innych pojazdów co pokazuje tempo rozwoju sztucznej inteligencji używanej do kierowania pojazdami.

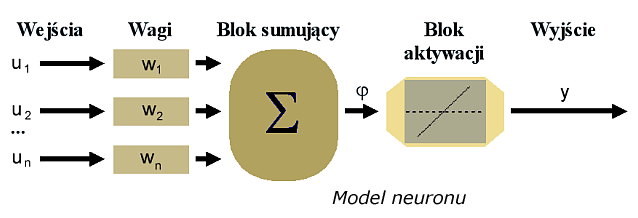
W 2016 roku program AlphaZero pokonał Mistrza TCEC (Top Chess Engine Championship czyli komputerowy turniej szachowy, który odbywa się od 2010 roku) z roku 2016 program Stockfish 8. AlphaZero przed meczem potrzebował jedynie 9 godzin treningu przeciwko samemu sobie, aby osiągnąć poziom mistrzowski i wygrać pojedynek. Programiści projektujący AlphaZero ograniczyli naukę jedynie do zasad gry, taktyka jaką program się posługiwał została wyuczona na podstawie treningu[[12]](#footnote-12).

2017 zorganizowano 20 dniowy turniej pokerowy w którym udział wzięło czterech najwyższej klasy graczy pokera oraz Libratus to program komputerowy wykorzystujący sztuczną inteligencje. Libratus wygrywał już od pierwszego dnia turnieju a gdy uczestnicy odpoczywali po grze on wykorzystywał ten czas na analizę wcześniejszej rozgrywki i swoje straty doskonaląc swoją technikę. Finalnie Libratus wygrał uzyskując wyjątkowo wysoką wygraną, a jego sukces oznacza że „Zdolność sztucznej inteligencji do strategicznego rozumowania w oparciu o niekompletne dane przekroczyła zdolność najlepszych ludzi – skomentował sukces Tuomas Sandholm, profesor informatyki w Carnegie Mellon University i współtwórca Libratusa”[[13]](#footnote-13)

2020 Amerykańska firma Waymo wywodząca się z Alphabet (szerzej znana jako Google) udostępniła w Arizonie pierwszą na świecie usługę w pełni zautomatyzowanego wezwania przejażdżki, bez człowieka za kierownicą[[14]](#footnote-14) - Waymo One. Firma działa do dziś i rozszerza swoje usługi na kolejne miasta w Stanach Zjednoczonych.

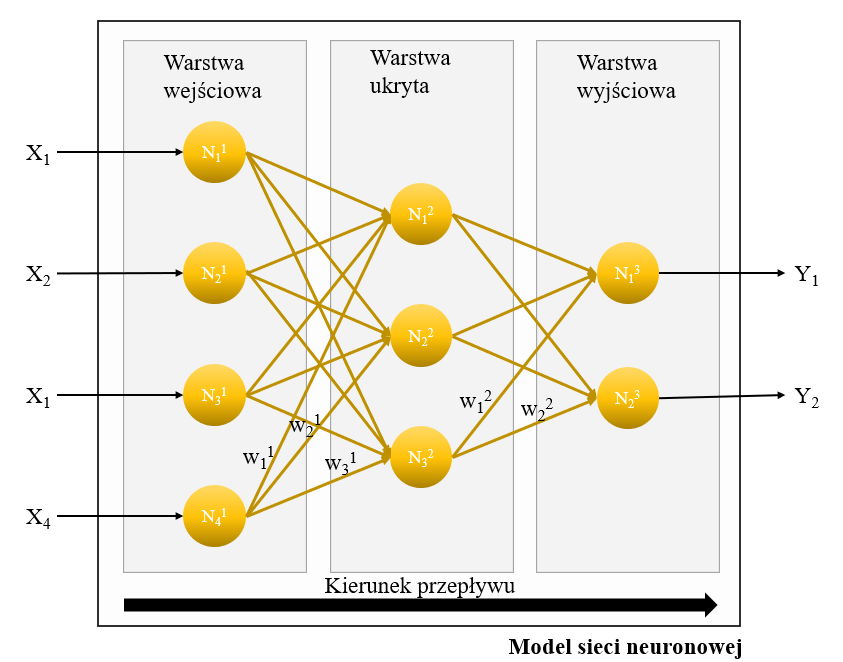
2.2 Podstawy sieci neuronowych

Sztuczna sieć neuronowa jest to uporządkowany zbiór sztucznych neuronów, który można przedstawić za pomocą skryptu napisanego w języku programowania. Koncept sieci oraz pojedynczego neuronu jest oparty na podstawie ich biologicznych odpowiedników. Sztuczny neuron jest to obiekt który posiada: n wejść, wagę każdego z wejść, blok sumujący, blok aktywacji oraz jedno wyjście.



Ilustracja 1 Model neuronu

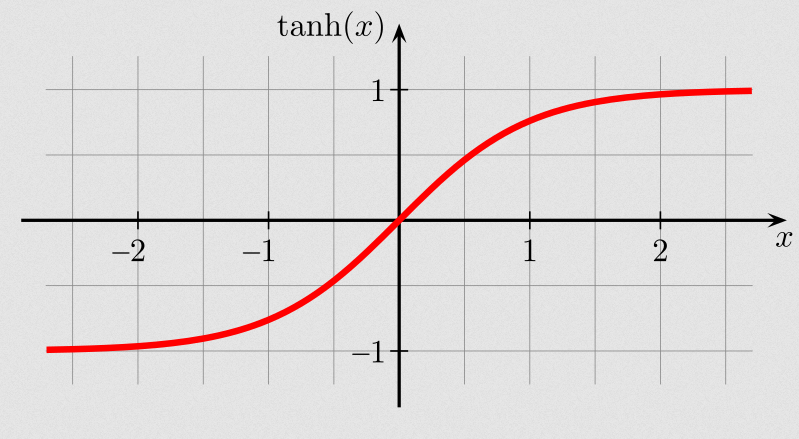
W celu przetworzenia sygnałów neuron odbiera je przez wejścia i przekazuje do bloku sumującego, którego celem jest zsumowanie iloczynów wartości sygnałów i wag wejścia przez które te sygnały zostały przesłane. Drugim działaniem jest obliczenie wartości funkcji aktywacji dla argumentu przesłanego przez pierwsze działanie. Obliczona wartość jest następnie przekazywana do wyjścia i jest wynikiem przetwarzania całego neuronu. Istnieje kilka sposobów organizacji neuronów w sieć, takich jak sieci jednokierunkowe, sieci rekurencyjne, czy sieci radialne. W niniejszej pracy zaimplementowałem sieć jednokierunkową wielowarstwową zwaną perceptronem wielowarstwowym[[15]](#footnote-15), dlatego w dalszej części to ten rodzaj sieci będę opisywać. W celu przetwarzania danych przez sieć, neurony składające się na nią podzielone są na trzy rodzaje warstw, ale jest to podział formalny ponieważ różnice między typami nie występują.



Ilustracja 2 Model sieci neuronowej zaimplementowany w projekcie

Schemat działania całej sieci przypomina sposób w jaki działa pojedynczy neuron. Neurony znajdujące się w tej samej warstwie nie są ze sobą bezpośrednio połączone. Pierwsza z nich to warstwa wejściowa. Neurony które znajdują się w tej warstwie odbierają sygnały z zewnątrz i przekazują je dalej do sieci, dlatego liczba neuronów jest uzależniona od liczby źródeł przesyłających sygnały. Drugim rodzajem warstwy jest warstwa ukryta, która może posiadać zero lub wiele warstw ukrytych, oznacza to, że każdą warstwę między warstwą wejścia, a warstwą wyjścia określa się jako warstwę ukrytą. Liczba warstw, oraz liczba neuronów w każdej z nich oraz liczba połączeń między neuronami jest zależna tylko od wiedzy i doświadczenia programisty. Dobór tych ilości można dobrać według metody piramidy geometrycznej, która zakłada że liczba neutronów w kolejnych warstwach tworzy kształt piramidy i maleje od wejścia do wyjścia[[16]](#footnote-16). Na ogół najlepsze wyniki otrzymuje się wybierając te liczebności w sposób empiryczny[[17]](#footnote-17). Ostatnia trzecia warstwa odpowiada za wykonanie ostatnich obliczeń z wartości przeliczonych i przekazanych od poprzednich warstw i obliczenie wyniku[[18]](#footnote-18). Ilość neuronów w tej warstwie zależy od przeznaczonego działania całej sieci. W moim przypadku sieć wysyła sygnał sterujący samochodem, z związku z tym jeden na ostatniej warstwie znajdują się dwa neurony. Jeden odpowiada za ruch samochodu do przodu, a drugi za skręt w lewo bądź w prawo.

Wspomniana wcześniej funkcja aktywacji jest implementowana w strukturę sztucznego neuronu w celu obliczenia wartości na jego wyjściu. Jest wiele typów funkcji aktywacji: funkcja liniowa, funkcja skoku jednostkowego, funkcja sigmoidalna lub tangens hiperboliczny. Wybór funkcji aktywacji zależy od rodzaju problemu jaki stawiamy przed siecią do rozwiązania. W niniejszej pracy użyłem dwóch funkcji aktywacji. Na drugim neuronie w warstwie wyjściowej użyłem funkcji progowej unipolarnej przyjmującej wartość 0 lub 1. W rozdziale 4.4 Stworzenie sieci neuronowej opisałem powód. Natomiast na pozostałych neuronach użyłem funkcji tangens hiperboliczny, ponieważ funkcja posiada ciągłe, łagodne przejście między wartością minimalną, a maksymalną ( -1 i 1).



Ilustracja 3 Wykres funkcji Tangens hiperboliczny[[19]](#footnote-19)

Wagi połączeń neuronów zostały oznaczone na modelu sieci neuronowej symbolem wij. To one zawierają cała wiedzę jaką zgromadziła sieć neuronowa. Ich zmiana poprzez metody uczenia ma związek ze zmianą zachowania obiektu którym steruje sieć.

2.3 Podstawy algorytmów ewolucyjnych

Terminem algorytm ewolucyjny określamy algorytm optymalizacyjny, w którym stosowane są mechanizmy selekcji, reprodukcji i mutacji, inspirowane przez biologiczny proces ewolucji zwany doborem naturalnym. W naturalnym środowisku mechanizm ten zachodzi gdy na daną populacje osobników wywierana jest presja otoczenia. W takich warunkach tylko najlepiej przystosowane osobniki mają szanse przetrwać. To one będą mieć możliwość przedłużenia gatunku poprzez zapoczątkowanie kolejnych populacji, które dzięki genom rodziców mogą okazać się jeszcze lepiej przystosowane do warunków, w których żyją. Algorytmy ewolucyjne działają na podstawie tego konceptu. Problem, który algorytm ma za zadanie rozwiązać jest implementowany jako środowisko, w którym żyje populacja osobników, a każdy z tych osobników reprezentuje potencjalne rozwiązanie problemu. Tak jak i w biologii proces ewolucji z biegiem czasu wykształca coraz to lepiej przystosowane osobniki, tak algorytmy ewolucyjne stopniowo tworzą coraz to lepsze rozwiązania problemu. Stąd wynika, że algorytmy ewolucyjne mogą być odpowiednim sposobem rozwiązywania problemów optymalizacyjnych, czyli takich charakteryzujących się bardzo dużą liczbą potencjalnych rozwiązań. W takich przypadkach wydajniejsze od sprawdzania każdego rozwiązania po kolei, okazuje się zastosowanie technik probabilistycznych wykorzystywanych przez algorytmy ewolucyjne, które używają wyboru losowego jako narzędzia do ukierunkowania procesu poszukiwań[[20]](#footnote-20).

Schemat działania algorytmu ewolucyjnego rozpoczyna się od wygenerowania populacji osobników o losowych parametrach. Zbiór tych parametrów określa się mianem genotypu, natomiast wynik osiągnięty przez każdego osobnika to jego fenotyp. Fenotyp podlega ocenie funkcji przystosowania (w pracy jako synonimy tego wyrażenia używam również zwrot funkcja celu lub fitness value). Każda wygenerowana populacja nazywana jest generacją. Gdy cała generacja wypracowała już swój fenotyp następuje proces selekcji osobników na podstawie ich wyników. Te osobniki, które poradziły sobie najgorzej zostają odrzucone i nie będą brać udziału w dalszych etapach, natomiast osobniki, które poradziły sobie najlepiej zostają powielone by kolejna generacja posiadała taką samą liczbę osobników. Genotyp osobników z nowej generacji zostaje poddany działaniu operatorów genetycznch. Wyróżniamy dwa operatory genetyczne: operator krzyżowania oraz operator mutacji. Operatory te różnią się od siebie prawdopodobieństwem występowania oraz sposobem w jaki zmieniają osobniki. Zazwyczaj przyjmuje się duże prawdopodobieństwo na wystąpienie krzyżowania w granicach 0,5 ≤ Pk ≤ 1. Natomiast dla mutacji przyjmuje się o wiele mniejsze prawdopodobieństwo w granicach 0 ≤ Pm ≤ 0.1[[21]](#footnote-21). Dzięki krzyżowaniu ewolucyjny proces przesuwa się w kierunku obiecujących obszarów w przestrzeni poszukiwań. Dodatkowo w trakcie operacji krzyżowania może wystąpić operacja mutacji, która polega na rzadko występującej, losowej zmianie parametrów każdego osobnika.

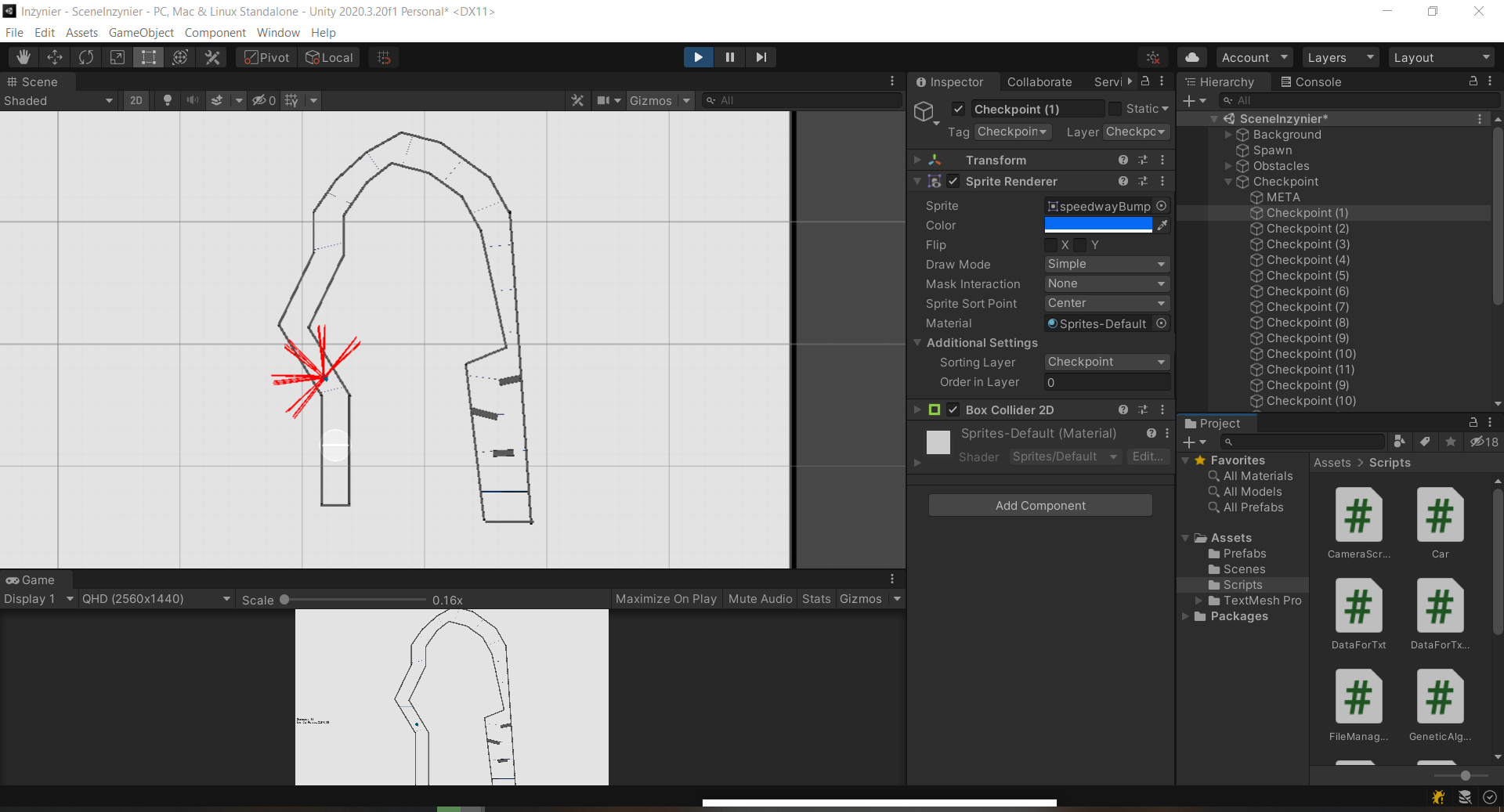
W przypadku mojej aplikacji genotypem jest zbiór wag zawartych w sztucznej sieci neuronowej. Za fenotyp uznaję składową kilku wartości, jednak wartością, która ma największy wpływ jest długość drogi jaką samochód pokonał po torze, ponieważ im dłuższa pokonana droga tym większa jest szansa na to, że samochód dojedzie do mety. W podrozdziale 4.3 opisałem szczegółowo implementacje i cel każdej wartości. Zaimplementowałem tylko operator mutacji, ponieważ operator krzyżowania jest przystosowany do algorytmów, w których osobniki posiadają genotyp w postaci np. jednowymiarowej tablicy. Uznałem, że krzyżowanie wag z sieci należących do poszczególnych osobników będzie niepotrzebnie skomplikowane, dlatego wymyśliłem inny mechanizm. Po etapie weryfikacji algorytm genetyczny wybiera na podstawie wartości funkcji przystosowania część najgorszych oraz część najlepszych osobników. Następnie klonuje wagi z tej lepszej części i wkleja je do części gorszej nadpisując ich poprzednie wartości. Dodatkowo, aby przyśpieszyć tempo w jakim populacja przystosowuje się do toru, zaimplementowałem mechanizm który powiela geny najlepszego osobnika na kilka pojazdów. Gdy powielenie dobiega końca, następuje mutacja wszystkich osobników. Implementacje powyższych mechanizmów opisałem w podrozdziale 4.6.

# **Rozdział 3 Użyte narzędzia i technologie**

W tym rozdziale opisałem jakich narzędzi i technologii użyłem w celu realizacji projektu. Opisałem silnik graficzny Unity, w którym utworzyłem symulacje, oraz język programowania C#, w którym napisałem kod aplikacji.

## 3.1 UNITY

Unity[[22]](#footnote-22) to zintegrowane środowisko będące jednocześnie silnikiem gry, które jest wykorzystywane do tworzenia gier 2d i 3d oraz animacji. Powstało z myślą uczynienia tworzenia gier powszechnie dostępnym i z każdym kolejnym uaktualnieniem przybliżało się do tego celu dodając obsługę kolejnych platform wraz z nowymi funkcjonalnościami. Obecna najnowsza wersja 2020.3.20f1 obsługuje 25 platform w tym Android, Windows, IOS, konsole, zestawy do rozszerzonej rzeczywistości, a nawet systemy operacyjnie przeznaczone dla telewizorów smart. Z Unity można korzystać na warunkach darmowej licencji gdy wykorzystujemy je do użytku własnego lub gdy nasza firma generuje przychód mniejszy niż $100 000 rocznie. Dzięki temu platforma szybko zyskała dużą popularność i już w 2018 połowa gier mobilnych oraz 60% treści dotyczących rozszerzonej rzeczywistości tworzone było w Unity. Dzięki zaawansowanej strukturze oraz możliwościom jakie posiada, duże studia chętnie korzystają z silnika tworząc uznane tytuły takie jak AmongUs (jedna z najpopularniejszych gier 2020 roku[[23]](#footnote-23)), Cities: Skylines[[24]](#footnote-24)(jedna z lepszych gier w swoim gatunku) oraz Hearthstone[[25]](#footnote-25)(gra która przeniknęła do e-sportu i zgromadziła 100 milionów graczy). Mimo tego że jest to zaawansowane narzędzie do tworzenia gier i animacji to użytkownik nie powinien mieć problemów z nauką obsługi ponieważ silnik jest intuicyjny a duże grono użytkowników we współpracy z twórcami utworzyło obszerną bazę wiedzy składającą się z dokumentacji, poradników oraz wątków na forach internetowych. Dodatkowo twórcy udostępnili za darmo kilkanaście gotowych projektów stworzonych w celu nauki podstawowych funkcjonalności.



Ilustracja 4 Zrzut ekranu z edytora Unity podczas uruchomionej symulacji

Interfejs Unity składa się z kilku podstawowych widoków które przedstawione są na rysunku 1 w kolejności od lewego górnego rogu Scene, Inspektor, Hierarchy, Consola(schowany) i rząd nizej Game oraz Project. Scene prezentuje środowisko, w którym tworzymy grę z perspektywy twórcy, a Game pokazuje grę z perspektywy gracza przekazując obraz z kamery umieszczonej przez projektanta. Można to porównać do widoku reżysera, który widzi cały plan zdjęciowy, do perspektywy widza, który ogląda film. W inspektorze widać właściwości oraz komponenty dodane do wybranego elementu z sekcji Hierarchy. Hierarchy prezentuje listę wszystkich komponentów, które umieszczone są na scenie. Widok Projekt wyświetla bibliotekę zasobów dostępnych w projekcie. Na rysunku 1, w prawym dolnym rogu widać otworzony folder Scripts, który zawiera pliki z klasami w języku C#. Console to widok w którym program prezentuje komunikaty błędów oraz logi. Ustawienie widoków można modyfikować w dowolny sposób według uznania i preferencji kreatora.

Projektowanie w Unity pola w uproszczeniu przeciąganiu elementów na scenę z poziomu hierarchy lub project i edytowanie ich komponentów w oknie inspektora. Komponenty to funkcjonalna część obiektu, która definiuje jego zachowanie. Właściwości komponentów można edytować poprzez interfejs oraz jest możliwość odwołania się do nich w kodzie skryptu. Każdy dodany obiekt określany jest jako „GameObject” i posiada domyślnie dodany component Transform[[26]](#footnote-26). Transform zawiera pozycje obiektu, wartość rotacji oraz wartość skali dzięki czemu łatwo jest umieścić obiekt w dowolnym miejscu w przestrzeni. Unity zapewnia szereg wbudowanych komponentów zapewniających obiektowi np. zachowanie według praw fizyki lub system kolizji. Unity umożliwia utworzenie własnego componentu w postaci skryptu napisanego w języku C#. Aby dodany skrypt mógł wywoływać zdarzenia, reagować na dane wejściowe użytkownika lub zmieniać parametry komponentu musi zawierać klasę, która dziedziczy po klasie MonoBehaviour. W momencie dołączania skryptu jako nowy komponent do elementu GameObject, Unity tworzy nową instancje obiektu typu Component według kodu skryptu. Inicjalizacja obiektów w Unity nie jest obsługiwana tradycyjnie przez programistę w kodzie lecz przez edytor. Każda klasa dziedzicząca po MonoBehaviour posiada wbudowane metody wywoływane automatycznie przez edytor. Są to metody Start(), Update(), FixedUpdate(). Start jest wywoływana przez edytor przed rozpoczęciem rozgrywki i dzięki temu jest idealnym miejscem do inicjalizacji zmiennych. Update i FixedUpdate są wywoływane automatycznie w interwałach w czasie trwania programu, dlatego to dobre miejsce do implementacji kodu aktualizującego stan obiektu. Szczegóły dotyczące różnic między powyższymi klasami oraz ich praktyczne wykorzystanie opisałem w rozdziale 3.3 Implementacja Osobnika,

## 3.2 C#

C# jest to wieloparadygmatowy język programowania stworzony przez firmę Microsoft. Kod napisany w języku C# jest tłumaczony na język CLI (Common Intermediate Language) który następnie jest wykonywany przez specjalne środowisko uruchomieniowe takie jak np. Net Framewok[[27]](#footnote-27). Kod który napisałem w ramach niniejszej pracy jest napisany w języku C# ponieważ jest to główny język programowania wspierany przez Unity.

Ponieważ Unity nie posiada opcji edytora kodu, mój projekt napisałem przy pomocy Visual Studio 2019. Jest to zintegrowane środowisko programistyczne, stworzone przez firmę Microsoft obsługujące kilka języków programowania między innymi: C#, JavaScript, Python. Wybrałem to narzędzie, ponieważ posiada darmowe rozszerzenie przeznaczone do integracji z Unity, które zapewnia możliwość debugowania oraz posiada IntelliSense czyli funkcjonalność automatycznego uzupełniania kodu[[28]](#footnote-28).

# **Rozdział 4 Implementacja**

W tym rozdziale opisałem działanie funkcjonalności, które są najważniejsze z perspektywy całego projektu.

4.1 Analiza problemu

W celu realizacji przedstawionego problemu implementacji algorytmu genetycznego podzieliłem główny problem na mniejsze dające się wyszczególnić etapy prac i uporządkowałem w celu usystematyzowania procesu wytwórczego. Wyszczególniłem następujące postępujące po sobie etapy: stworzenie toru wyścigowego, implementacja pojedynczego osobnika, zaprojektowanie sztucznej sieci neuronowej wraz jej zachowaniem, implementacja algorytmu genetycznego oraz implementacja mechanizmu zapisującego wynik po skończeniu symulacji. Wzorując się na metodach wytwarzania oprogramowania opartego na programowaniu iteracyjno-przyrostowym, każdy etap prac zakończony był testami weryfikującymi dodane funkcjonalności. W kodzie aplikacji znajdują się metody lub fragmenty zakomentowanego kodu, których zadaniem było pokazywać stan aplikacji w czasie testów lub podczas rozwiązywania błędów. Ze względu na to, że ich zastosowanie ma mniejsze znaczenie z perspektywy działającej symulacji nie opisałem ich działania.

4.2 Zaimplementowanie środowiska w Unity

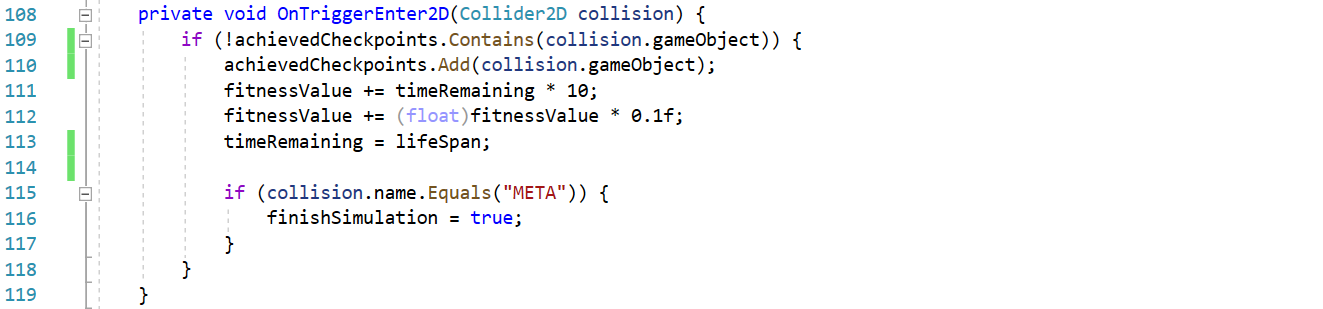
W pierwszej kolejności utworzyłem środowisko w programie Unity. Przez środowisko mam na myśli nowy projekt utworzony za pomocą szablonu dla aplikacji dwuwymiarowych. Aby zasymulować tor, po którym będą się przemieszczać pojazdy, utworzyłem tło, na którym umieściłem obiekty: bariery, checkpointy, punkt spawnu oraz canvas. Poniżej wyjaśnię po krótce zastosowanie tych obiektów.

Canvas to niewidoczne tło, na którym prezentowane są statystyki w czasie działania symulacji takie jak obecny numer populacji, oraz wartość funkcji przystosowania(fitness value) najlepszego pojazdu w danym momencie.

Bariery to obiekty, z których zbudowany jest tor oraz przeszkody które utrudniają dojechanie do mety. Pojazdy pojawiają się na jednym końcu zamkniętego toru w miejscu spawnu i próbują dotrzeć do mety znajdującej się na jego drugiej stronie.

Bariery, checkpointy i samochody to obiekty o typie GameObject. Wszystkie posiadają komponent Box-Collider 2D, który umożliwia wykrywanie okoliczności w której obiekty nachodzą na siebie. Dzięki temu możliwe było zasymulowanie sytuacji zderzenia samochodu z barierą oraz najechania samochodu na checkpoint. Każdy pojazd w momencie pojawienia się na torze posiada ustawioną flagę collided na wartość ‘false’. Jest to zmienna instancji klasy Car, od której zależy czy dany osobnik funkcjonuje. W momencie zderzenie z barierą flaga jest ustawiana na wartość ‘true’. Ma to na celu sprawić by pojazdy poruszały się po torze bez odbijania się lub przesuwaniu wzdłuż barier. Żeby program mógł uczyć się efektywnie nawet przy ilości osobników przekraczającej np. 100, należało zagwarantować, aby pojazdy nie kolidowały ze sobą. Zapewniłem to przy pomocy jednej z funkcjonalności Unity czyli warstw. Warstwy definiują, które obiekty mogą wchodzić ze sobą w interakcje. Utworzyłem trzy nowe warstwy i ustawiłem w kolejności od najniższej: „Cars”, „Speedway”, „ Checkpoint”, a następnie każdemu rodzajowi obiektu przypisałem inną warstwę. Każdy obiekt ma możliwość zdefiniowania dwóch warstw. Pierwsza ‘Layer’ która odpowiada za to w której warstwie występuje obiekt, druga ‘Sorting layer’ określa z jaką warstwą obiekt wchodzi w interakcje. Pojazd posiada ustawioną Layer na warstwę Cars oraz sorting layer na warstwę speedway, dzięki temu pojazdy wykrywają tor jednak nie kolidują ze sobą.

Checkpointy czyli punkty kontrolne to obiekty rozstawione na całej długości toru w celu wzmocnienia procesu nauki. By zrozumieć powód zastosowania checkpointów ważne, by dowiedzieć się w jaki sposób osobniki zdobywają wartość funkcji przystosowania. Każdy pojazd w czasie rzeczywistym zlicza całkowitą odległość jaką pokonał, realizuje to metoda countTotalDistanceTravelled w klasie Car.cs. Metoda calculateFitnessValue z klasy Car.cs zwiększa wartość fitness o odległość obliczoną przez poprzednią metodę. Odległość ta jest jednych z głównych składowych wartości fitnessValue. Jednak przejechana odległość nie zawsze może oznaczać, że osobnik dotarł do mety, dlatego na wartość fitness wpływa również operacja uruchamiana w momencie osiągnięcia punktu kontrolnego. Może zdarzyć się, że pojazd będzie jeździł w kółko lub na pewnym odcinku toru zawracał, by wyjeździć dłuższą odległość. W tym przypadku pojazd, który dojechałby do mety mógłby mieć kilkukrotnie mniejszą wartość funkcji przystosowania od pojazdów, które nieskończenie jeździłby w kółko. Aby się przed tym zabezpieczyć wprowadziłem checkpointy. W momencie inicjalizacji pojazdu każdy osobnik otrzymuje 5 sekundy by dotrzeć do punktu kontrolnego. Jeśli nie zdąży i jego czas się skończy, wtedy automatycznie zmieni się jego flaga collided na true. Jeśli zdąży i przejedzie checkpoint, uruchomiona zostanie metoda OnTriggerEnter2D() którą wklejam poniżej by omówić jej działanie.



Kod 1 Zrzut ekranu metoda onTriggerEnter2D z klasy Car.cs

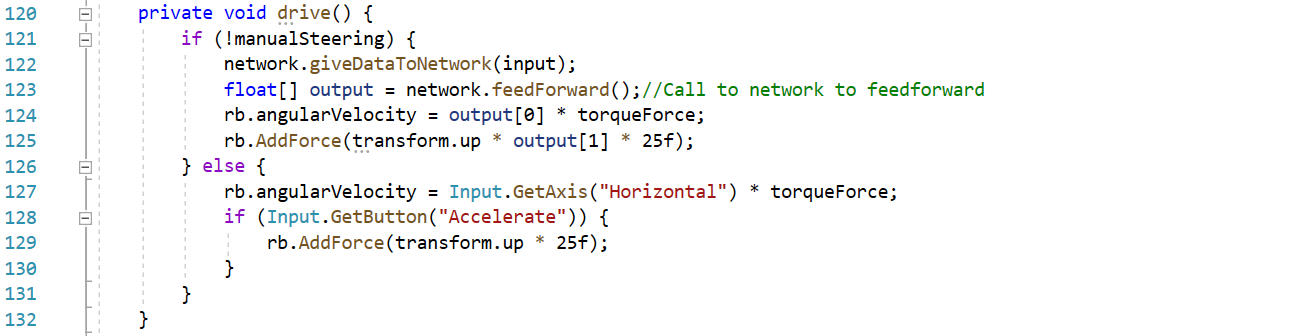
Każda instancja klasy Car zawiera prywatną listę achievedChekpoints. Lista przechowuje referencje do obiektów (o typie checkpoint) z którymi samochód już kolidował. By zabezpieczyć się przed sytuacją, w której samochód będzie raz za razem przekraczał ten sam checkpoint, metoda na początku sprawdza czy obiekt kolidujący znajduje się w obecnej kolekcji. Jeśli tak, nie wpływa to w żaden sposób na samochód. Jeśli nie, checkpoint zostanie dodany do kolekcji oraz wartość fitnessValue zostanie powiększona o dwie wartości. Najpierw zostanie powiększona o iloczyn pozostałego czasu\*10. Ma na to celu przydzielać więcej punktów osobnikom które poruszają się po najszybszej trasie. Następnie wartość fitnessValue zostaje powiększona o 10%. Po tych operacjach metoda zapewnia by osobnik miał szanse dotrzeć do następnego checkpointu, dlatego odnawia jego pozostały czas ‘życia’ zapisany w zmiennej timeRemaining. Przypisuje jej wartość ze zmiennej lifeSpan, która jest zadeklarowana na poziomie komponentu obiektu Car. Na koniec metoda sprawdza czy punkt kontrolny, który osiągnął samochód to META. Jeśli tak to zmienia wartość zmiennej finishSimulation, której użycie opisałem w rozdziale 3.9 Implementacja mechanizmu zapisywania wyników.

4.3 Implementacja osobnika

Obiekt pojazd posiada dodany komponent RigidBody2D który umożliwia symulowanie prostej fizyki poruszania się. Komponent posiada parametry za pomocą których można zmieniać fizyczne właściwości obiektu. W wyniku testów dobrałem parametry w taki sposób, aby fizyka poruszania się osobnika była zbliżona do fizyki jazdy samochodu. Mass = 1, Linear Drag = 2, Angular Drag = 120, Gravity Scale = 0. Warto tutaj wspomnieć o parametrze Gravity Scale. Musi on być ustawiony na wartość równą zero, ponieważ Unity w przypadku generowania obrazu 2D, ustawia plansze(scene) prostopadle do podłoża. W przypadku ustawienia parametru na wartości większe od zera, pojazdy będą opadać w dół planszy, proporcjonalnie z siłą ustawionej siły grawitacji.

Unity udostępnia szeroki wybór komponentów, które po dołączeniu do obiektu wpływają na jego zachowanie lub stan, jednak najczęściej niestandardowe zachowania należy zakodować samemu. W tym celu tworzymy nowy skrypt, czyli nowy plik z klasą i dołączamy go jako komponent. Domyślnie klasa utworzona w Unity dziedziczy po klasie MonoBehaviour . Należy wyszczególnić trzy metody ważne dla sposobu działania projektu w Unity. Są to Start(), Update(), FixedUpdate(). Wszystkie te metody wywoływane są automatycznie, różnią się głównie kolejnością i częstotliwością wywołań. Metoda Start wołana jest na samym początku przed Update i FixedUpdate gdy tylko obiekt korzystający ze skryptu jest dostępny. To dobre miejsce na zakodowanie inicjalizacji np. zmiennych obiektu. Update i FixedUpdate to jedne z najczęściej używanych metod w Unity. Wywoływane są po wielokroć w czasie działania programu. Różnicą między nimi jest częstotliwość wywołań. Update jest wywoływana po każdej wygenerowanej klatce. To ile czasu zajmie, aby silnik graficzny wygenerował każdą klatkę zależy od mocy obliczeniowej i ilości obliczeń, dlatego czas między kolejnymi wywołaniami metody Update może się różnić. Metoda FixedUpdate natomiast jest wywoływana w regularnych odstępach czasu, a po każdym jej wywołaniu silnik graficzny wykonuje obliczenia. Z tego powodu FixedUpdate jest przeznaczona do wywoływania czynności które wpływają na obiekty z zaimplementowaną fizyką tj. np. z komponentem RigidBody[[29]](#footnote-29).

Do obiektu Car dołączyłem plik Car.cs, ta klasa zawiera kod odpowiadający za sterowanie, inicjalizacje, działanie pojedynczego samochodu. Po pierwsze należy zakodować sterowanie czyli zapewnić by pojazd pod wpływem czynników zewnętrznych zmieniał swoje położenie. Poniżej prezentuje metodę drive() która odpowiada za sterowanie osobnikiem.

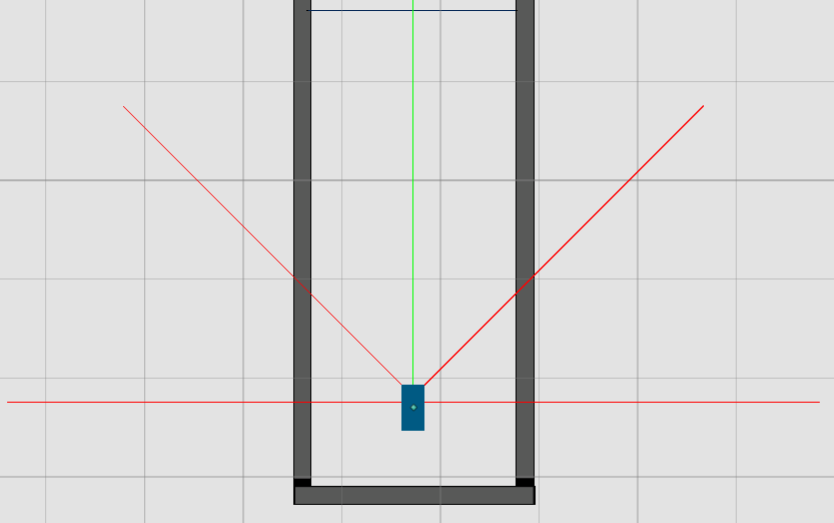


Kod 2 Zrzut ekranu metoda drive z klasy Car.cs

Jako że metoda wpływa na obiekt z fizyką, jest wywoływana w metodzie FixedUpdate(). Pierwszy warunek sprawdza wartość zmiennej bool manualSteering. Zmienna ta jest utworzona na potrzeby testów i ma na celu oznaczenie czy pojazdem kieruje programista.

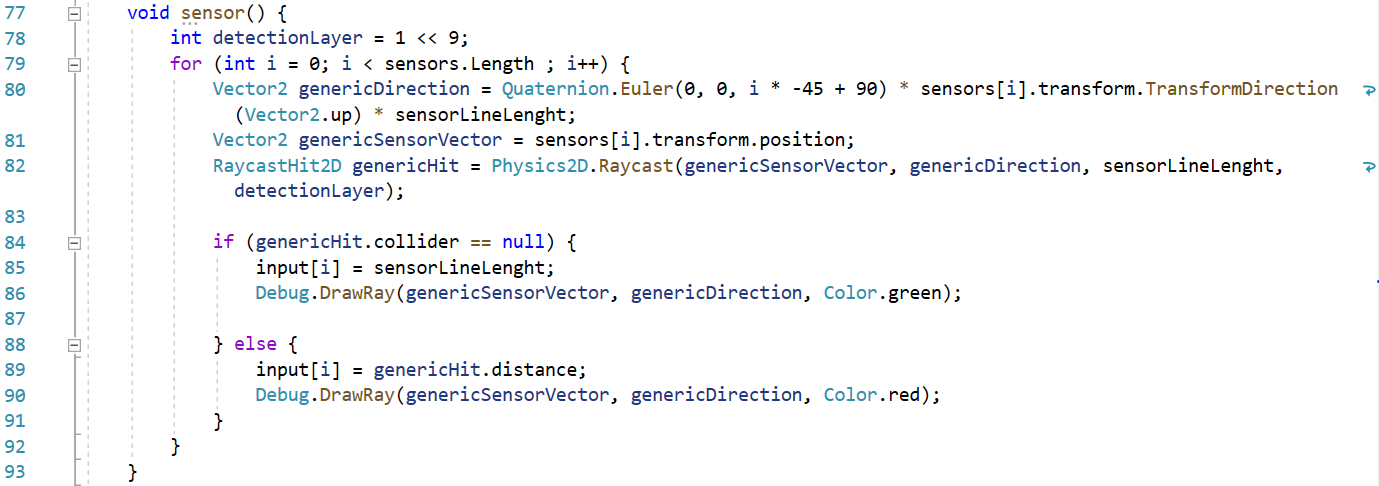
Zmienna domyślnie ustawiona jest na false. Jej główne celem było zapewnienie kontroli nad pojazdem w trakcie budowania toru i testowania parametrów dotyczących fizyki jazdy. Dalej omówię działanie od linii 122. Jest to odwołanie do sieci neuronowej danego osobnika i wywołanie na niej metody która przekazuje tablice 5 wartości pobraną z pięciu sensorów auta. Następnie zostaje wywołanie wykonania obliczeń w sieci. To w jaki sposób to następuje opisuję w podrozdziale 3.4. W wyniku działania tej metody sieć zwraca tablice zawierającą dwie wartości. Wartość na zerowym indeksie odpowiada za skręt samochodu, a wartość z indeksu pierwszego odpowiada za ruch samochodu do przodu, tzn. za potoczne przyciśnięcie pedału gazu.

Każdy samochód posiada wspomniane wyżej 5 sensorów dzięki którym zbiera dane dla sieci neuronowej. Poniżej prezentuje jak wygląda osobnik z wiązkami sensorów.



Ilustracja 5 Pojazd z 5 wiązkami czujników odległości

Sensory rejestrują odległość jaka dzieli samochód i bariery. Jeśli na trasie wiązki wystąpi bariera, wtedy sensor zapisuje odległość i zapala się na czerwono. Jeśli wiązka nie natrafi na przeszkodę wtedy sensor jako wartość zwraca długość wiązki, a jej kolor jest ustawiany na zielony. Jako iż ważna jest odległość zwracana przez sensor postanowiłem, że nie będzie ona emitowana ze środka pojazdu. Takie zastosowanie wymagało by korekty jej długości o szerokość pojazdu od środka do wybranej krawędzi. Zamiast tego każdy pojazd posiada pięć osobnych malutkich obiektów umieszczonych na samych krawędziach pojazdu i to od nich wystrzeliwana jest wiązka. Rozwiązanie to wymagało tylko o kilka linii kodu więcej przy inicjalizacji sensorów, aby przypisać obiekty do konkretnych referencji i umieścić je w jednej kolekcji sensors, a jego główną zaletą jest prosty sposób pobierania danych z sensorów bez obróbki danych, tak jak miałoby to miejsce w przypadku wcześniej wspomnianego rozwiązania. Obsługą sensorów zajmuje się metoda sensor() której kod prezentuje poniżej.



Kod 3 Zrzut ekranu metoda sensor z klasy Car.cs

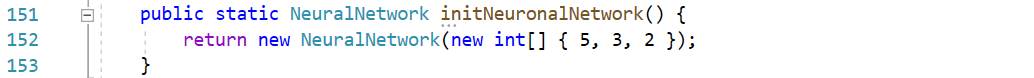
W linii 78, ustawiona zostaje zmienna która przechowuje numer warstwy na której operować będą sensory. Unity przechowuje numer warstwy pod postacią bitową o zakresie Integer[[30]](#footnote-30), czyli 32 bitów. Każdy z bitów może być osobną warstwą. Z edytora Unity wiem, że bariery które powinny być wykrywane przez sensory znajdują się na 9 warstwie. Oznacza to że bit powinien być ustawiony na 9 pozycji licząc od prawej strony. Gdy przedstawimy ten numer jako liczbę w systemie dziesiętnnym by zapisać ją do zmiennej otrzymamy 512. Sprawia to dwa problemy, po pierwsze w przypadku zmiany warstwy należałoby ponownie przeliczać numer warstwy na liczbę w systemie dziesiętnym, po drugie ustawienie numeru warstwy na wartość o wielkości 512 stwarza ryzyko nieporozumienia(od kiedy ten program ma 512 warstw?). Aby rozwiązać oba problemy prościej jest wykorzystać operator przesunięcia bitowego w lewo „<<”. 1 << 9 oznacza przesunięcie jednego bitu o 9 pozycji w lewo. Dzięki temu ustawienie warstwy jasno wskazuje na kolejność warstwy, a ewentualna zmiana nie wymaga przeliczenia.

Następnie w pętli po kolei program odwołując się za pomocą iteratora, odnosi się do tabeli sensor która przechowuje referencje do sensorów. Sensory są ustawione w tabeli w kolejność i są generowane według ruchów wskazówek zegara zaczynając od lewego dolnego. Przy okazji iterator wykorzystywany jest przy obliczaniu kąta zwrócenia wiązki przez i \* -45 + 90. Dla przykładu kąt dla pierwszej wiązki to będzie 0 \* -45 + 90 = 90 stopni. Każda wiązka po wygenerowaniu od razu zapisuje wartość do tabeli input[] która jest wykorzystywana przez wcześniej wspomnianą metodę drive().

4.4 Stworzenie sieci neuronowych

Każdy osobnik posiada zmienną o typie NeuralNetwork przechowującą obiekt sztucznej sieci neuronowej. Jest to główne odwołanie do sieci za pomocą którego osobnik przesyła do sieci wartości odczytane przez sensory oraz pobiera wartości wyjściowe neuronów z ostatniej warstwy które odpowiadają za sterowanie. Jednak w celu rozdzielenia logiki odpowiadającej za np. tworzenie sieci lub zachowania sieci postanowiłem, aby cała sztuczna sieć neuronowa składała się z 3 klas. Pierwsza to klasa główna będąca centrum sieci neuronowej NeuralNetwork.cs. Pozostałe dwie to klasy pomocnicze, które są składową klasy głównej: NnLayer.cs, Neuron.cs. Hierarchia ta odwzorowuje model sztucznej sieci neuronowej z rozdziału 1.2. Obiekt klasy NeuralNetwork zawiera prywatną listę obiektów o typie NnLayer. Z kolei każdy obiekt klasy NnLayer zawiera listę obiektów klasy Neuron. Za utworzenie sieci i przydzielenie jej do konkretnego osobnika odpowiada klasa Manager. Jest ona przypisana do punktu spawnu jako komponent i to ona odpowiada za koordynowanie działań wszystkich klas w programie.

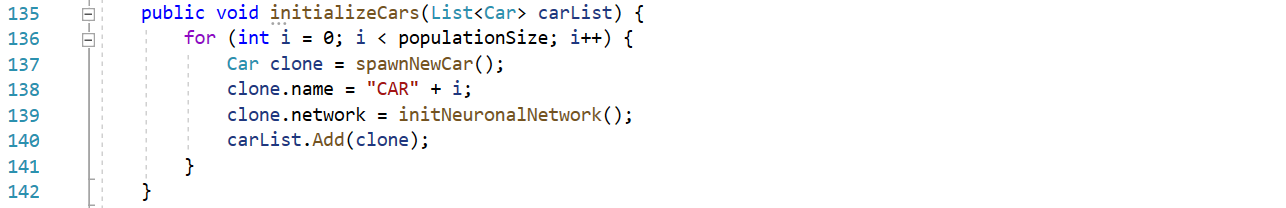
Inicjalizacja sieci neuronowej. Utworzenie sieci wiąże się z utworzeniem obiektów wszystkich klas, jednak klasa Manager jako klasa sterująca działaniem całej symulacje powinna skupić się jedynie na utworzeniu sieci i przypisaniu jej do obiektu o typie Car. Aby ta operacja było możliwe najprostsza, kod odpowiadający za inicjalizacje całej sieci wywoływany jest przez konstruktor klasy NeuralNetwork.



Kod 4 Zrzut ekranu metoda initNeuronalNetwork z klasy Manager.cs

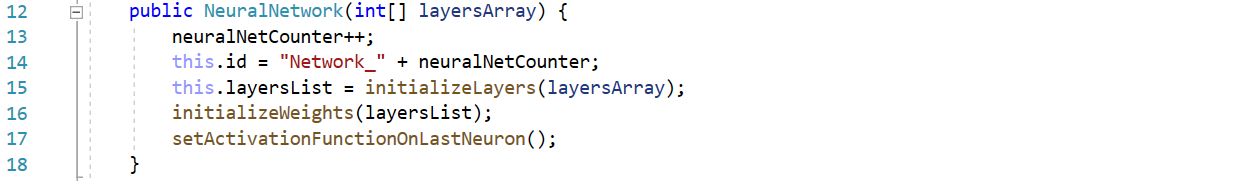
Dzięki temu klasa Manager nie musi wiedzieć w jaki sposób tworzyć nowe warstwy sieci, inicjalizować neurony i przypisywać im wagi itp. Jedyne co potrzebuje zrobić to utworzyć nowy obiekt przy pomocy operatora new i przekazać do konstruktora tablice o typie int. Przekazany parametr reprezentuje strukturę sieci, każda liczba reprezentuje liczbę neuronów w danej warstwie. Sieć utworzona przez konstruktor z załączonego powyżej kodu, będzie zawierać 3 warstwy sieci, pierwsza warstwa będzie składać się z pięciu neuronów, druga z trzech i ostatnia z dwóch.

Na samym początku po uruchomieniu symulacji, Manager w metodzie Start inicjalizuje nową listę która przechowuje referencje do wszystkich osobników. Ta lista jest przekazywana do metody initializeCars, która za pomocą pętli iteruje i uzupełnia kolekcję tworząc nowy obiekt klasy Car.cs po czym od razu tworzy nową sieć neuronową i przypisuje ją do danego samochodu. Metoda jest wykorzystywana zarówno przy tworzeniu pierwszej jak i każdej następnej generacji.



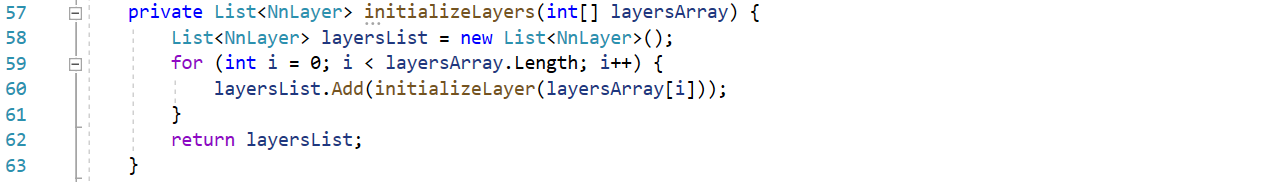
Kod 5 Zrzut ekranu metoda initializeCars z klasy Manager,cs

Wracając do konstruktora klasy NeuralNetwork, jego cel jakim jest inicjalizacja obiektu sieci neuronowej można podzielić na dwa etapy. W etapie pierwszym powstaje struktura tj. obiekty składające się na sieć. Tworzone są warstwy sieci, a do każdej z nich przypisywane są nowe utworzone neurony. Odpowiada za to kod w linii 15 na rysunku 8. W etapie drugim struktura zostaje wypełniona danymi. Neuronom przypisuje się wylosowane wagi połączeń. Odpowiada za to kod z linii 16. Intencja metody z linii 17 zostanie wytłumaczona w kolejnej części opisującej etap pierwszy.



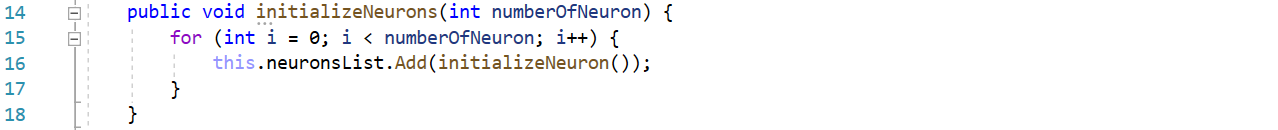
Kod 6 Zrzut ekranu konstruktor klasy NeuralNetwork.cs

Etap pierwszy konstruktor sieci rozpoczyna od wywołania metody initializeLayers. Tworzy ona nową listę przechowującą obiekty o typie NnLayer, która będzie przechowywać liczbę obiektów odpowiadającej wielkości tablicy przesłanej w parametrze. W każdej iteracji dodawany jest nowy obiekt zwrócony przez metodę initializeLayer, która wywołuje kontruktor klasy NnLayer przesyłając mu w parametrze ilość neuronów jaka powinna zostać zainicjalizowana w tej warstwie.



Kod 7 Zrzut ekranu metoda initalizeLayers z klasy NeuralNetwork.cs

Idąc głębiej zgodnie z wywołaniem do konstruktora klasy NnLayer, tworzy on kolekcje neuronsList przechowującą typ Neuron oraz wywołuje metodę initializeNeurons. Odpowiada ona za wypełnienie kolekcji neuronami które tworzy za pomocą metody initlializeNeuron. Wewnątrz niej wywoływany jest konstruktor bezparametrowy klasy Neuron.

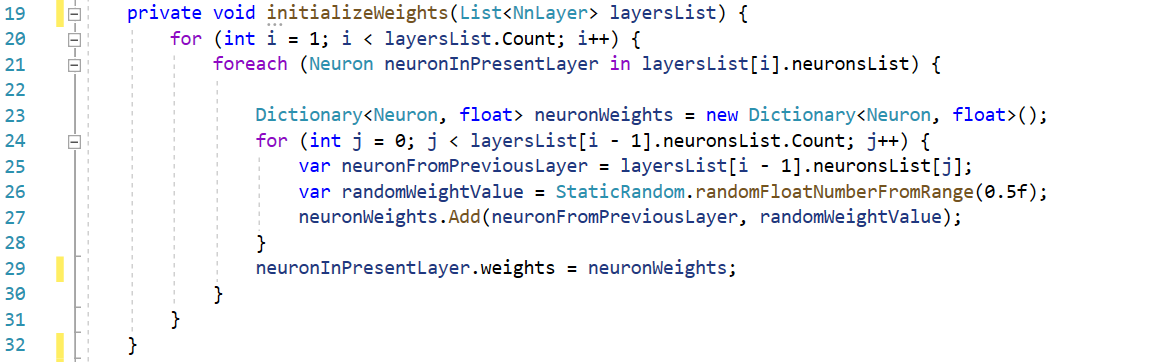


Kod 8 Zrzut ekranu metoda initializeNeurons z klasy NnLayer.cs

Zawiera on tylko jedną istotną operacje, jest to przypisanie funkcji aktywacji. Wszystkie neurony domyślnie posiadają przypisany tangens hiperboliczny.

Jednak jest jeden neuron w każdej sieci który posiada inną funkcje aktywacji. W trakcie testów symulacji zauważyłem, że samochody często nie zdążały wyhamować i zbyt rozpędzone podchodziły do zakrętu. Dlatego postanowiłem zmienić wartość funkcji aktywacji na ostatnim neuronie w ostatniej warstwie, którego wartość wyjściowa kieruje prędkością samochodu. To zadanie realizuje metoda setActivationFunctionOnLastNeuron. Ustawia ona „BinaryStep” jako jego funkcje aktywacji. Po zmianie zauważyłem poprawę w prowadzeniu pojazdów widoczną szczególnie przy pokonywaniu ostrych zakrętów.

Etap drugi rozpoczyna się od inicjalizacji wag. Zastanawiałem się czy pojedyncza waga powinna być reprezentowana przez osobną klasę. Jednak ze względu na to, że waga połączenia jest ściśle skorelowana z neuronami oraz fakt że to neuron wykorzystuje wartość wagi przy obliczaniu swojej wartości wyjścia, zdecydowałem że waga między neuronem A i neuronem B, będzie zapisana w neuronie B (zakładając że neuron A znajduje się w warstwie x, a neuron B w warstwie x + 1). Wagi zapisane są pod postacią kolekcji o typie Dictionary o nazwie „weights”. Przechowuje ona zbiór par kluczy i przypisanych im wartości. Typ klucza to Neuron a typ wartości to float. Oznacza to, że każdy neuron przechowuje w swojej pamięci kolekcje, która zawiera bezpośrednie referencje do neuronów z poprzedniej warstwy oraz wartości przypisane do każdego neuronu reprezentujące połączenia. Metoda odpowiedzialna za inicjalizacje wartości wag to initializeWeights. Jako parametr przyjmuje referencje do struktury całej sieci i dzięki niej ma dostęp do każdego jej elementu.



Kod 9 Zrzut ekranu metoda initializeWeights z klasy NeuralNetwork.cs

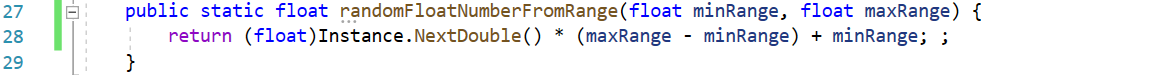
InitializeWeights posiada 3 pętle. Pierwsza z linii 20 iteruje po warstwach sieci neuronowej, druga z linii 21 iteruje po neuronach w danej warstwie. Dla każdego neuronu tworzy nowy obiekt kolekcji neuronWeights. To w nim neuron będzie przechowywał wagi. Trzecia pętla z linii 24, iteruje po neuronach z warstwy poprzedniej względem neuronu z drugiej pętli. Uzupełnia kolekcje referencją do neuronu oraz wylosowaną wartością wagi. Metoda randomFloatNumberFromRange losuje wartość połączenia między -0.5 a 0.5. Gdy pętla dojdzie do ostatniego neuronu, kolekcja zapisywana jest do neuronu.

Do losowania wartości losowych wykorzystuje oddzielną klasę statyczną StaticRandom. Wykorzystuje ona klasę Random. Klasa ta przy inicjalizacji korzysta z inicjatora, który może zostać podany jawnie lub nie. „Jeśli ten sam inicjator jest używany w odniesieniu do oddzielnych Random obiektów, spowoduje to wygenerowanie tej samej serii liczb losowych.”[[31]](#footnote-31) Taka sytuacja mogła by znacząco przeszkodzić w symulacji, ponieważ mogło by dojść do sytuacji gdy wiele wartości wag byłoby takich samych dla wielu osobników. Skutkiem tego mogłaby być populacja jednakowych osobników, które podczas mutacji zmieniałyby się o takie same wartości. Aby do tego nie doprowadzić klasa StaticRandom przy każdym wywołaniu inicjalizuje nowy obiekt klasy Random z inicjatorem.



Kod 10 Zrzut ekranu deklaracja zmiennej threadLocal w klasie StaticRandom.cs

Role inicjatora przyjmuje statyczna zmienna seed, która przy pierwszym uruchomieniu programu przyjmuje wartość równą liczbie milisekund, które upłynęły od momentu uruchomienia systemu. Dodatkowo sama zmienna jest przechowywana jako zmienna atomowa, która przy każdym wywołaniu jest inkrementowana przez Interlocked.Increment(ref seed). Samo wywołanie zostało opakowane w operacje lambda przypisaną do zmienną o typie ThreadLocal, który zapewnia magazynowanie danych w lokalnym wątku[[32]](#footnote-32). Jedynymi publicznymi metodami wystawionymi na zewnątrz są randomFloatNumberFromRange i randomFloatNumberDefaultRange. Odwołują się do metody Instance, która zwraca wywołanie do zmiennej threadLocal. Dzięki temu, że dla każdego odpytania klasa Random zostanie utworzona z nowym, zawsze nowym, różnym od innych inicjatorem (czyli inkrementowaną dla każdego wywołania zmienną seed), mam gwarancje, że generator jest odporny na wielowątkowe wywołania lub bardzo wiele zapytań w bardzo krótkim czasie. Poniżej prezentuje jedną z dwóch publicznych metod klasy StaticRandom.



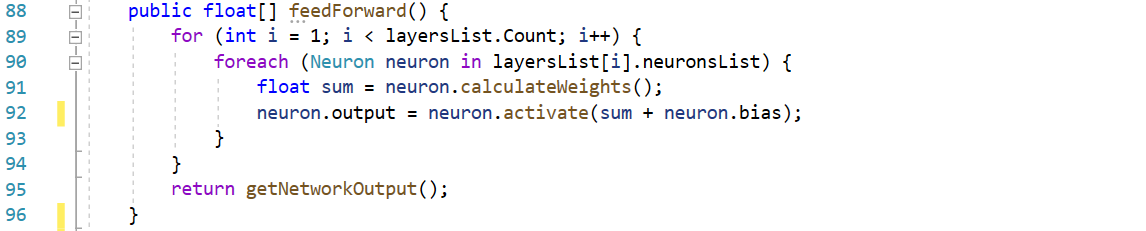
Kod 11 Zrzut ekranu metoda randomFloatNumberFromRange z klasy StaticRandom.cs

Zwraca ona losową liczbę z zakresu określanego przez parametry przesyłane do metody minRange i maxRange. Druga metoda zwraca domyślną wartość zwracaną przez metodę klasy Random, NextDouble. Zwraca ona wartość od 0.0 do 1.0.

4.5 Implementacja zachowania sieci neuronowej

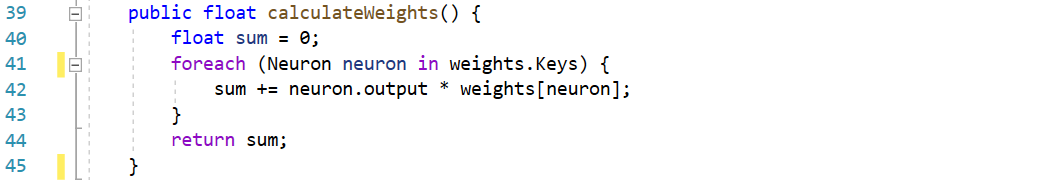
Implementacja zachowań sztucznej sieci neuronowej. Sieć w trakcie trwania symulacji powinna zwracać pewne wartości liczbowe, które osobnik będzie w stanie interpretować. Aby zapewnić losowość w generacji, sieć powinna też mutować. Dodatkowo w przypadku powielania osobników z dobrymi wynikami sieć powinna być zdolna do wyabstrahowania spoza obiektu Car. Wszystkie te czynności czyli umiejętność mutacji, przeliczanie wartości podanych do sieci oraz możliwość przenoszenia uznałem za zachowania, które powinna implementować sieć neuronowa a nie klasy zewnętrzne.

Poniższa metoda feedForward z klasy NeuronalNetwork realizuje przepływ wartości wewnątrz sztucznej sieci neuronowej. Klasa Car wewnątrz metody drive() najpierw dostarcza dane z sensorów do sieci za pomocą metody giveDataToNetwork(). Dane przekazane są pod postacią pięcioelementowej tablicy. Każda wartość zostaje przydzielona do innego neuronu w pierwszej warstwie. Następnie metoda drive wywołuje feedForward, a jej wynik zapisuje do tablicy której wartości zostaną użyte do ustawiania prędkości i wartości skrętu.



Kod 12 Zrzut ekranu metoda feedForward z klasy NeuralNetwork.cs

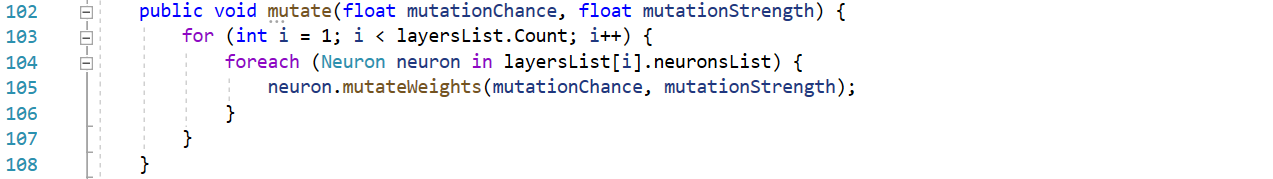
Wewnątrz metody feedForward znajdują się dwie pętle, które iterują po neuronach. Zgodnie z wcześniejszym założeniem, że wagi połączeń miedzy neuronem z warstwy x i warstwy x+1 zapisane są warstwie x+1, iteracja rozpoczyna się od drugiej warstwy sieci. Dla każdego neuronu wywoływana jest metoda calculateWeights. Obliczona wartość przekazywana jest jako argument do funkcji aktywacji, która zwraca wartość która zostaje zapisana jako wartość wyjściowa neuronu.



Kod 13 Zrzut ekranu metoda calculateWeights z klasy Neuron.cs

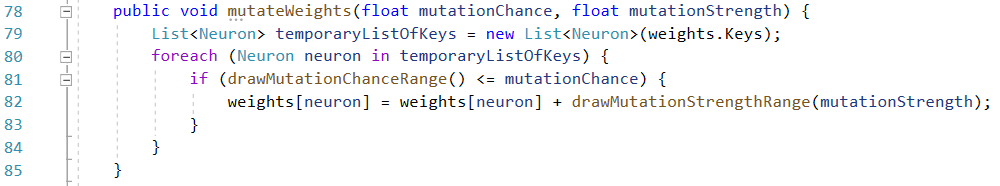
Powyżej prezentuje metodę calculateWeights, wykorzystuje ona pętle by dla kolekcji weights, przeiterować po neuronach. Dla każdego neuronu pobiera jego wartość wyjściową(w przypadku neuronów z pierwszej warstwy wartością wyjściową są wartości odczytane z sensorów) i mnoży ją przez wartość wagi połączenia. Metoda zwraca zsumowane wartość iloczynów.

Implementacja mechanizmu mutacji sieci neuronowej. O tym kiedy sieć powinna przejść operacje mutacji decyduje algorytm genetyczny, jednak wiedzę o tym jak przeprowadzić ten proces posiada sama sieć. Główną metodą rozpoczynającą mutacje jest mutate. Przyjmuje dwa parametry wejściowe: wartość określającą szanse na wystąpienie mutacji oraz wartość określającą siłę mutacji. Siła mutacji oznacza możliwość wylosowania większej liczby, która docelowo zmieni wartości wag.



Kod 14 Zrzut ekranu metoda mutate z klasy Neuron.cs

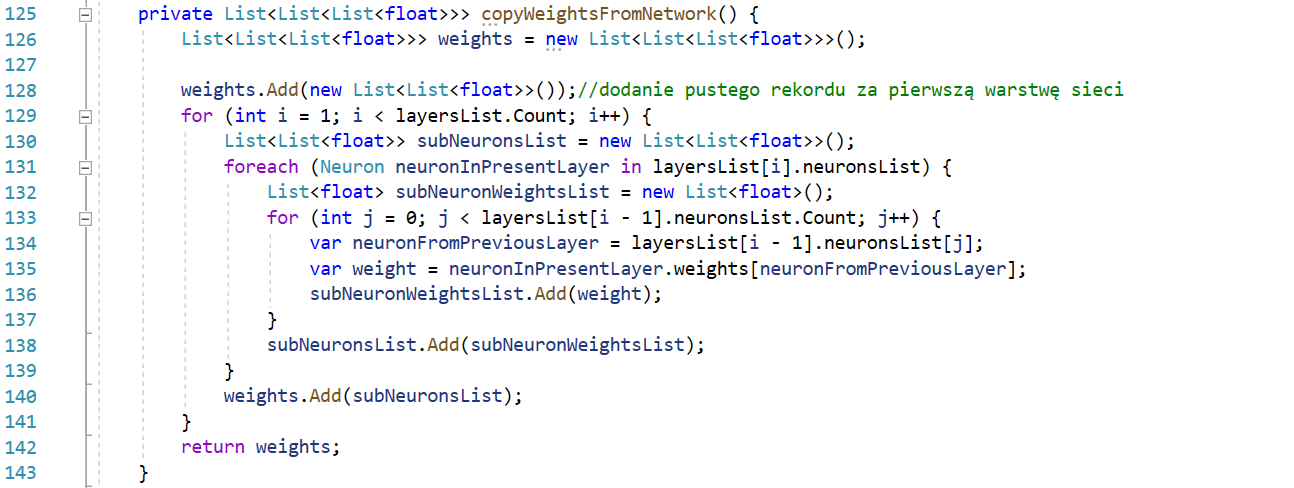
Metoda używa dwóch pętli by przejść po wszystkich neuronach w sieci, rozpoczynając od warstwy drugiej. Dla każdego neuronu wywoływa jest metoda mutateWeights, która odpowiada za mutacje wag.



Kod 15 Zrzut ekranu metoda mutateWeights z klasy Neuron.cs

Na początku tworzy zmienną pomocniczą która przechowuje listę kluczy z kolekcji weights. Następnie iteruje przez kolekcje i dla każdego neuronu sprawdza warunek odpowiedzialny za wystąpienie mutacji. Jeśli metoda drawMutationChanceRange() zwróci wartość mniejszą od zmiennej mutationChance, wtedy następuje mutacja. Metoda drawMutationStrengthRange otrzymuje w parametrze wartość określającą siłę mutacji i na jej podstawie zwraca liczbę z zakresu od -mutationStrenght do mutationStrenght(np. -0.4; 0.4). Wylosowana wartość dodawana jest do wagi połączenia i w ten sposób zmienia jego wartość.

Implementacja właściwości sieci jako wyabstrahowania nabytej wiedzy sieci. Jako nabytą wiedzę uznaje wartości wag. Właściwość ta jest wykorzystywana gdy określony osobnik powinien zostać powielony ze względu na wielkość jego wyniku wartości fitness. Do wyabstrahowania wiedzy z sieci służy metoda getNetworkData. Tworzy ona nowy obiekt o typie NeuralNetworkData. Jest to osobna klasa która ma za zadanie przechowywać w jednym miejscu wszystkie informacje które są przenoszone między sieciami. Klasa NeuralNetworkData przechowuje wagi w kolekcji o typie zagnieżdżonej listy List<List<List<float>>>. Obiekt kolekcji jest pobierany z sieci za pomocą metody copyWeightsFromNetwork. Jej cel to skopiowanie wartości wszystkich wag w sieci z zachowaniem położenia danej wagi w sieci. To zagnieżdżenie list wynika właśnie z imitacji struktury sieci. Pierwsza lista zawiera listę warstw, druga zawiera listę neuronów, a trzecia zawiera listę wartości wag dla każdego neuronu.

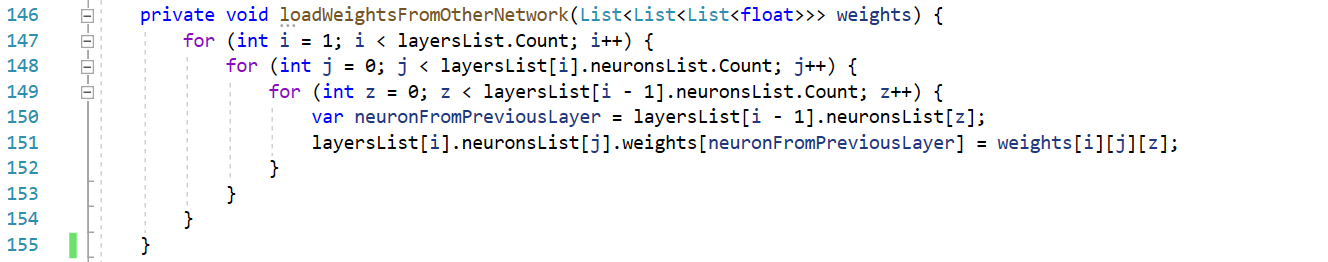


Kod 16 Zrzut ekranu metoda copyWeightsFromNetwork z klasy NeuralNetwork.cs

Dwie pierwsze pętle w linii 129 i 131 mają za zadanie przejść po wszystkich neuronach w sieci, zaczynając od warstwy drugiej. Pętla trzecia z linii 133 iteruje po neuronach warstwy poprzedniej względem warstwy o indeksie i. W każdym cyklu pętla wyznacza neuron z poprzedniej warstwy aby użyć go jako odniesienia do kolekcji weights w neuronie z warstwy o indeksie i. Wartość z kolekcji weights neurona jest dodawana do listy.

Gdy pobraliśmy już wiedzę sieci i zapisaliśmy do obiektu o typie NeuralNetworkData, zostało ją załadować do sieci innego obiektu. Służy do tego metoda loadNewNetworkData

Za pomocą gettera pobiera kolekcję przechowującą wagi i przesyła je do metody loadWeightsFromOtherNetwork. Analogicznie do metody pobierającej dane, używa ona trzech pętli. Dwie pierwsze mają za zadanie dostać się do każdego neuronu w sieci, trzecia natomiast ma za zadanie podmienić wszystkie wartości w kolekcji weights każdego neuronu.



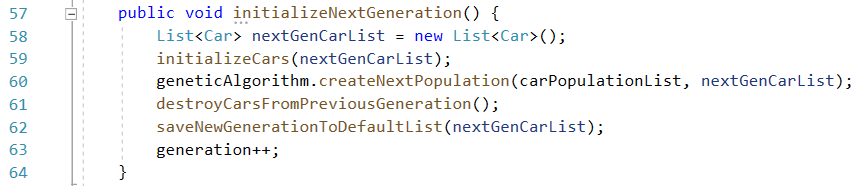
Kod 17 Zrzut ekranu metoda loadWeightsFromOtherNetwork z klasy NeuralNetwork.cs

Na pierwszy rzut oka wydawać się może że metody do pobierania i ładowania wiedzy sieci są nadmiarowe. Wywołują one jedynie po jednej metodzie nie zmieniając ich zachowania. Dlatego możliwe by było aby wywoływać te metody bezpośrednio z pominięciem getNetworkData i loadNewNetworkData. Postanowiłem jednak zastosować je razem z oddzielnym obiektem przechowującym wiedzę aby zadbać o łatwe rozszerzenie funkcjonalności i rozdzielenie logiki. Algorytm ewolucyjny opisany w kolejnym podrozdziale nie musi posiadać wiedzy o tym co konkretnie należy skopiować z jednej sieci do drugiej by skopiować obiekt. Interesuje go jedynie metoda kopiująca wiedzę sieci. W przypadku dodania parametru wpływającego na wartość funkcji przystosowania, należałoby w utworzyć dwie metody, jedna która pozyska ją z sieci i druga która ją do sieci wklei. Następnie wystarczy dodać wywołania tych metod do getNetworkData i loadNewNetworkData. Następnie klasa przechowujaca wiedzę powinna zostać rozszerzona o zmienną która ją przechowa oraz metody set i get. Z perspektywy reszty klas w programie, nic się nie zmienia.

4.6 Dodanie technik ewolucyjnych

Gdy wszystkie osobniki zostaną zdezaktywowane po uderzeniu w bariery lub wyczerpaniu czasu, program przystępuje do tworzenia następnej generacji. Jednak aby nowa generacja posiadała osobniki, które osiągną większe wartości funkcji przystosowania, należało zaimplementować algorytm, który będzie ukierunkowywać naukę całej populacji na osiągnięcie celu jakim jest wykształcenie osobnika który dojedzie do mety. Wykorzystałem do tego celu algorytm genetyczny, który zapisałem pod postacią osobnej klasy GeneticAlgorithm.cs. Jego głównym zadaniem jest zadbać by osobniki, które osiągnęły wyższe niż większość wartości funkcji przystosowania zostały powielone w następnej generacji, a następnie lekko zmodyfikować całą populacje poprzez losowe zmiany w wagach. Każde powtórzenie tego schematu doprowadzi do powielenia lepszych samochodów które dojechały trochę dalej niż reszta, co finalnie w generacji n przełoży się w na wylosowanie takich wartości wag, dla któregoś losowego osobnika, że będzie on w taki sposób intepretować otrzymywane przez sensory wartości że dojedzie do mety omijając przeszkody nie uderzając po drodze w żadną barierę.

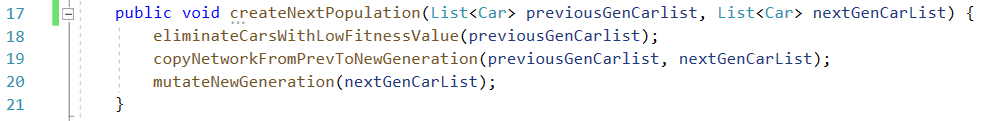
Inicjalizacja algorytmu następuje w centrum programu czyli w metodzie Start w klasie Manager. Metoda inicjalizująca tworzy nową instancje algorytmu i za pomocą seterów(czyli metod ustawiających wartość zmiennej lokalnej) ustawia mu wartości podane przez użytkownika (mutationChance i mutationStrength) przed uruchomieniem symulacji.



Kod 18 Zrzut ekranu metoda initializeNextgeneration z klasy Manager.cs

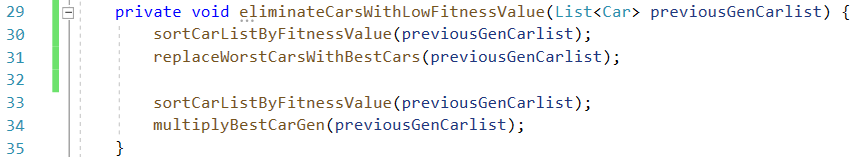
Główne użycie algorytmu następuje w czasie tworzenia nowej ale nie pierwszej generacji. Manager w metodzie update wykrywa że wszystkie osobniki są zdezaktywowane, sprawdza czy któryś samochód dojechał do mety i jeśli żaden nie osiągnął celu, przystępuje do inicjalizacji nowej populacji wywołując metodę initializeNextGeneration. Metoda inicjalizuje nową listę samochodów i zapisuje ją pod postacią listy nextGenCarList. Następnie odwołuje się do instancji algorytmu i wywołując jego publiczną metodę createNextPopulation przekazuje mu w parametrze dwie referencje, pierwsza do populacji która właśnie skończyła swoją próbę i druga do nowej generacji która za chwile będzie tę próbę podejmować. Zanim omówię co dokładnie dzieje się w czasie wywołania tej metody opiszę do końca działanie metody initializeNextGeneration. Po operacjach przeprowadzonych przez algorytm, manager usuwa obiekty o typie Car oraz wszystkie referencje zapisane w kolekcji carPopulationList. Następnie z kolekcji nextGenCarList kopiuje referencje do osobników z nowo utworzonej generacji i umieszcza je w kolekcji carPopulationList. Po tej operacji nowa generacja jest gotowa do zainicjalizowania, więc inkrementowy jest numer generacji.

Wracając do algorytmu genetycznego i metody createNextPopulation z klasy GeneticAlgorithm. Tworzy ona nową populacje w trzech krokach. Pierwszym krokiem jest wyeliminowanie słabych i powielenie mocnych osobników, drugi krok to przeniesienie referencji do osobników ze starej do nowej kolekcji. Trzecim i ostatnim krokiem jest wykonanie losowych mutacji na wszystkich obiektach w kolekcji. Poniżej prezentuje kod metody.



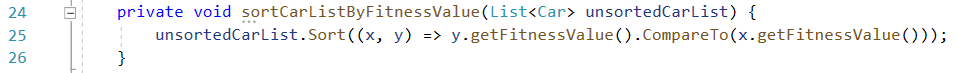
Kod 19 Zrzut ekranu metoda createNextPopulation z klasy GeneticAlgorithm.cs

Idąc zgodnie z kolejnością wywołań najpierw opiszę działanie metody eliminateCarsWithLowFitnessValue, która zawiera główną logikę algorytmu genetycznego. Jej działanie podzielone jest na dwa główne zadania, celem pierwszego jest wyeliminowanie najgorszych osobników z populacji, a celem drugiego jest powielenie sztucznej sieci neuronowej należącej do najlepszego osobnika.



Kod 20 Zrzut ekranu metoda eliminateCarsWithLowFitnessValue z klasy GeneticAlgorithm.cs

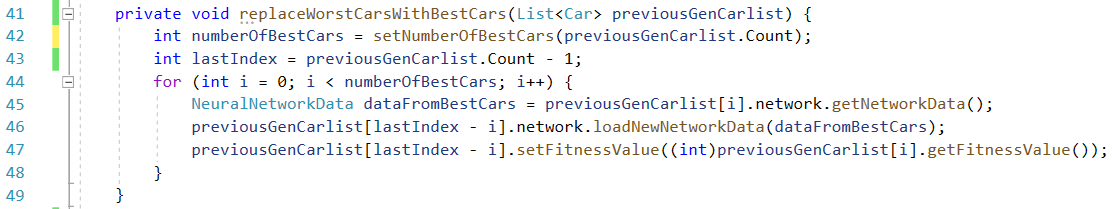
Implementacja obu zadań operuje na obiektach odnosząc się do nich poprzez indeksy. By metody poprawnie przeprowadziły proces powielania sztucznych sieci neuronowych pochodzących od najlepszych osobników najpierw kolekcje są sortowane malejąco według wartości funkcji przystosowania przez poniższą metodę.



Kod 21 Zrzut ekranu metoda sortCarListbyFitnessvalue z klasy GeneticAlgorithm.cs

Do sortowania wykorzystałem zaimplementowaną do kolekcji metodę sort, która jako parametr przyjmuje Comparison<T> comparison. Przy wywołaniu zadeklarowałem comparator pod postacią wyrażenia lambda. Przyjmuje on dwa parametry x i y które odpowiadają za obiekty z kolekcji czyli obiekty o typie Car. W bloku wyrażenia lambda odwołuje się najpierw do wartości fitness obiektu y ponieważ wymusza to posortowanie listy malejąco.

Poniżej prezentuje metodę replaceWorstCarsWithBestCars, która implementuje pierwsze zadanie.



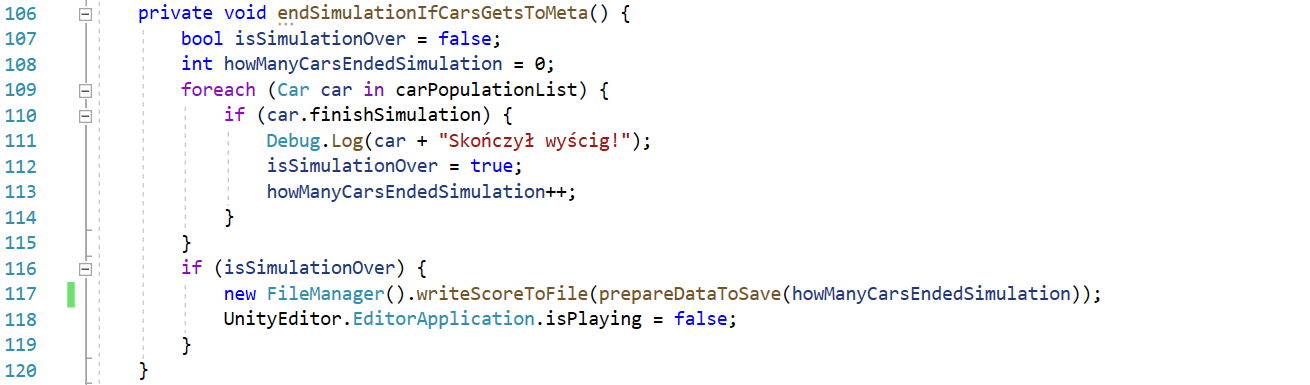
Kod 22 Zrzut ekranu metoda replaceWorstCarsWithBestCars z klasy GeneticAlgorithm.cs

Metoda otrzymuje w parametrze referencje do listy samochodów z poprzedniej generacji. Na początek metoda setNumberOfBestCars zwraca liczbę samochodów jaka zostanie powielona. Jej wynik determinuje zawarty w niej warunek, który w przypadku populacji zawierającej mniej niż 10 osobników zwraca wartość równą 2, a dla większych populacji zwraca liczbę równą 40% z ilości całej populacji. W linii 43 zadeklarowana jest zmienna pomocnicza lastIndex. Skutek działania metody byłby taki sam gdybym jej nie deklarował i używał w pętli odwołania bezpośrednio do wielkości listy jednak, jej użycie zwiększa czytelność kodu i pozwala w jasny sposób przekazać intencje jaką jest odwołania do ostatniego indeksu. Idąc dalej program wchodzi w pętle, która w każdym cyklu tworzy nową referencje do obiektu o typie NeuralNetworkData i przypisuje jej obiekt pobrany z sieci osobnika. Warto zwrócić uwagę, że pobranie networkData z previousGenCarList zależy od iteratora, który inicjowany jest z wartością równą zero. Jako że lista została wcześniej posortowana to odwołując się do pierwszych miejsc, pobierane są wartości najlepszych osobników. Następnie pętla odwołuje się do samochodów na ostatnich indeksach i podmienia ich networkData. Aby umożliwić ponowne posortowanie listy samochodów podmieniana jest również wartość fitness. Bez tego samochody które osiągnęły bardzo słaby wynik i zostały im przypisane sieci najlepszych samochodów, dalej znajdowały by się na końcu listy przez co mogły by znowu zostać zakwalifikowane do podmiany sieci w kolejnych krokach. Po metodzie replaceWorstCarsWithBestCars lista samochodów zostaje posortowana ponownie po czym metoda multiplyBestCar realizuje drugie zadanie. Jej działanie polega na pobraniu obiektu NeuralNetworkData od najlepszego samochodu, wybraniu 7% najsłabszych osobników z populacji i podmianie ich sieci na pobrany obiekt.

Przechodząc dalej program wraca do metody createNextPopulation i wywołuje copyNetworkFromPrevToNewGeneration by przenieść referencje do nowej kolekcji z której później klasa Manager zainicjuje nowe obiekty samochodów. By to osiągnąć metoda wykorzystuje wspomniane wcześniej metody Car.getNetworkData() i Car.loadNewNetworkData(). Na koniec otrzymujemy nową kolekcje, która zawiera w większości obiekty posiadające sztuczne sieci neuronowe które poradziły sobie najlepiej w poprzedniej generacji. Są to jednak kopie więc ponowne uruchomienie symulacji w tym momencie mogłoby skończyć się bardzo podobnymi wynikami. Dlatego w ostatnim kroku przeprowadzana jest mutacja, która ma zapewnić losowe zmiany w sieci neuronowych osobników. Algorytm nie posiada wiedzy jak przeprowadzić mutacje. W jego kompetencji jest jedynie wywołanie tego procesu w odpowiednim momencie z odpowiednimi parametrami. Metoda mutateNewGeneration wykorzystuje instrukcje foreach by dla każdego elementu kolekcji wywołać metodę mutate zdefiniowaną w klasie NeuralNetwork z parametrami które określają jak często mutacja będzie mieć szanse wystąpić oraz jak duży wpływ będzie mogła wywrzeć każda zmiana.

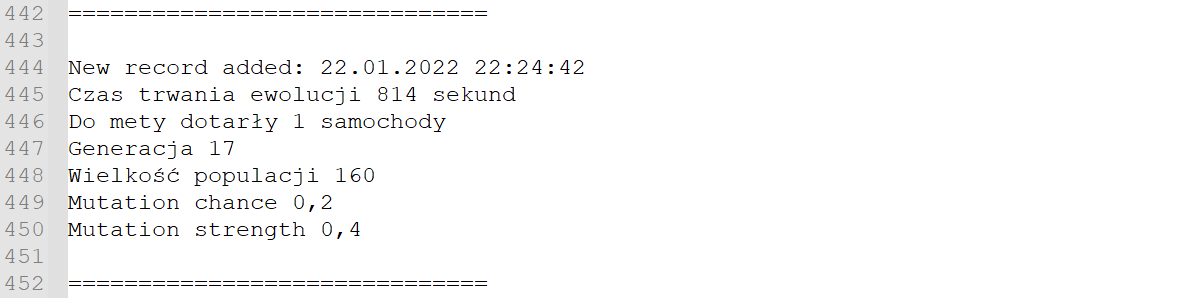
4.7 Implementacja mechanizmu zapisywania wyników

Symulacja kończy się gdy przynajmniej jeden osobnik dotrze do mety. W tej sytuacji program zapisuje wynik symulacji w pliku tekstowym oraz kończy działanie. Klasa sterująca symulacją – Manager, w metodzie update sprawdza czy wszystkie osobniki w populacji są jeszcze aktywne. W przypadku gdy wszystkie są zdezaktywowane Manager wywołuje metodę endSimulationIfCarsGetsToMeta.



Kod 23 Zrzut ekranu metoda endSimulationIfCarsGetsToMeta z klasy Manager.cs

Każdy obiekt o typie Car posiada zmienną finishSimulation o typie bool. Kiedy dotrze do mety (czyli checkpointu którego nazwa to „META”) wartość tej zmiennej ustawiana jest na true. Metoda przechodzi po całej populacji i sprawdza, czy któryś samochód skończył symulacje. Jeśli tak to ustawia wartość zmiennej isSimulationOver na true oraz inkrementuje licznik określający ile osobników dotarło do mety. Następnie w przypadku skończenia symulacji następuje zapisanie wyniku do pliku. Odpowiada za to oddzielna klasa o nazwie FileManager. Dane przeznaczone do zapisania w pliku zgromadzone są w klasie DataForTxt. Metoda z linii 117 prepareDataToSave używa klasy DatForTxtBuilder by utworzyć i zwrócić obiekt o typie DataForTxt. Metoda zapisująca dane do pliku posiada warunek który sprawdza czy plik o określonej nazwie (w tym przypadku „score.txt”) już istnieje. Jeśli nie to taki plik jest tworzony, w oddzielnym przypadku program znajduje plik o tej nazwie i dopisuje na jego końcu rekord. Poniżej prezentuje jak w notatniku wygląda pojedynczy rekord.



Ilustracja 6 Zrzut ekranu z programu notepad++ prezentujący wyniki zapisane po skończeniu symulacji

# **Rozdział 5 Prezentacja wyników i pomysłów na rozwinięcie projektu/wykorzystanie**

W tym rozdziale opisałem wyniki symulacji którą wielokrotnie uruchamiałem na swoim komputerze w trakcie pisania pracy, oraz pomysły na wykorzystanie projektu i związane z tym kilka propozycji rozwinięcia aplikacji.

5.1 Wyniki

Po skończeniu pracy nad aplikacją przystąpiłem do fazy testów. W jej trakcie wielokrotnie uruchamiałem symulacje by sprawdzić czy zaimplementowany algorytm jest w stanie wykształcić osobnika, który dojedzie do mety. Zarejestrowałem około 60 przejazdów[[33]](#footnote-33) testując różne konfiguracje parametrów symulacji (wielkość populacji, szansa na wystąpienie mutacji, siła mutacji) by wybrać te optymalne, czyli takie dla których jest największa szansa na powodzenie. Celowo używam tu słowa szansa, ponieważ zauważyłem, że możliwe jest dobranie parametrów w taki sposób by populacji nie udało się wykształcić zwycięskiego osobnika. Przykładem takiego przypadku jest ustawienie wielkości populacji na 10, szansy na mutacje na 0.1 oraz siły mutacji na 0.2. W tym przypadku populacja była zbyt mała i mutowała zbyt rzadko dlatego podczas kilkunastu godzinnej próby populacja nie wykształciła osobnika który dojechałby do mety. Dłuższych symulacji nie przeprowadzałem. W tym przypadku parametry były bardzo małe, jednak sprawdziłem też sytuacje w której parametry były by za duże by osiągnąć sukces. Dla parametrów: szansa na mutacje 0.5 i siła mutacji 0.5 osobniki zmieniały swoje parametry zbyt często i o zbyt duże wartości by cała populacji udało się osiągnąć sukces.

Optymalny zaobserwowany zakres parametrów, który wykorzystałem w dalszych testach to populacja licząca 100 osobników, szansa na wystąpienie mutacji w granicach od 0.15 do 0.35, siła mutacji w granicach od 0.3 do 0.5. Wybrałem wielkości parametrów dla 9 prób i dla każdej próby uruchomiłem symulacje 10 razy. Otrzymałem 90 rekordów[[34]](#footnote-34) które za pomocą samodzielnie napisanej aplikacji konsolowej w Visual Studio, przeniosłem do programu Excel[[35]](#footnote-35). Dla każdej próby obliczyłem średnie wartości, które prezentuje w tabelce poniżej.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Parametry próby** | **Średnia czasu przejazdu(w sekundach)** | **Średni numer generacji, która osiągnęła cel** |
| 1 | **szansa na mutacje 0.15; siła mutacji 0.3;  10 rekordów** | 918 | 18,7 |
| 2 | **szansa na mutacje 0.15; siła mutacji 0.4;  10 rekordów** | 799 | 15,8 |
| 3 | **szansa na mutacje 0.15; siła mutacji 0.5;  10 rekordów** | 586,3 | 12,1 |
| 4 | **szansa na mutacje 0.25; siła mutacji 0.3;  10 rekordów** | 661,5 | 13,5 |
| 5 | **szansa na mutacje 0.25; siła mutacji 0.4;  10 rekordów** | 636,4 | 12,7 |
| 6 | **szansa na mutacje 0.25; siła mutacji 0.5;  10 rekordów** | 686,5 | 13,5 |
| 7 | **szansa na mutacje 0.35; siła mutacji 0.3;  10 rekordów** | 684,5 | 13,9 |
| 8 | **szansa na mutacje 0.35; siła mutacji 0.4; 10 rekordów** | 635,5 | 12,9 |
| 9 | **szansa na mutacje 0.35; siła mutacji 0.5; 10 rekordów** | 513,9 | 11 |

Ilustracja 7 Tabela uśrednionych wyników z testu sprawdzającego wpływ szansy na mutacje i siły mutacje na wyniki populacji

Na podstawie tabeli utworzyłem dwa wykresy liniowe prezentujące otrzymane uśrednione wartości dla każdej z prób. Przerywana linia obrazuje trend.

W wyniku testu widać że dla 90 prób, symulacja trwała najdłużej gdy parametry: szansa na mutacje 0.15, siła mutacji 0.3 oraz najkrócej gdy szansa na mutacje wynosiła 0.35 a siła mutacji wynosiła 0.5. Oznacza to że populacja licząca 100 osobników przystosowuje się do środowiska szybciej jeśli częściej następują zmiany oraz zakres zmian jest większy.

Dla trzech pierwszych prób występuję największa różnica wyników, spośród prób z tą samą szansą na mutację. Na podstawie tych wyników widać, że populacja dostosowuje się o wiele szybciej gdy zakres zmian jest większy mimo tej samej małej szansy na mutacje.

Ilustracja 8 Wykres uśrednionego czasu przejazdu z testu pierwszego

Ilustracja 9 Wykres uśrednionego numeru generacji z testu pierwszego

W powyżej opisanym teście wielkość populacji była taka sama w każdej próbie. W celu zbadania wpływu tego parametru na tempo w jakim populacja osiąga cel przeprowadziłem kolejny test. Na podstawie wyników z poprzedniego testu wybrałem parametry z próby, dla której symulacja przystosowywała się najszybciej czyli z próby 9. Następnie wybrałem różne wielkości populacji i dla każdej próby uruchomiłem symulacje 10 razy[[36]](#footnote-36) (wyniki dla próby o id = 3 skopiowałem z poprzedniego testu). Wyniki uśredniłem i zapisałem do poniższej tabeli.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Parametry próby** | **Średnia czasu przejazdu(w sekundach)** | **Średni numer generacji** |
| **1** | **populacja 40** | **943,4** | **19,7** |
| **2** | **populacja 70** | **669,2** | **13,9** |
| **3** | **populacja 100** | **513,9** | **11** |
| **4** | **populacja 130** | **605** | **11,6** |
| **5** | **populacja 160** | **495,2** | **9,8** |

Ilustracja 10 Tabela uśrednionych wyników z testu sprawdzającego wpływ wielkości populacji na wyniki populacji

Ilustracja 11 Wykres uśrednionego czasu przejazdu z testu drugiego

Na podstawie tabeli utworzyłem wykresy liniowe prezentujący otrzymane uśrednione wartości dla każdej z prób. Na podstawie wykresów widać malejący trend oznaczony przerywaną linią, który wskazuje że im mniej osobników zawiera populacja tym wolniej jest ona w stanie przystosować się do środowiska.

Ilustracja 12 Wykres numeru generacji z testu drugiego

Po zarejestrowaniu 190 rekordów i powyższych testach moim głównym wnioskiem jest potwierdzenie, że zaimplementowany algorytm genetyczny, przy dobraniu optymalnych parametrów spełnia założony cel projektu.

Dodatkowo symulacja spełnia swoje zadanie mimo zrezygnowania z operatora krzyżowania i oparcia całego procesu nauki populacji na selekcji i operatorze mutacji.

Dzięki testom potwierdziłem, że wartości parametrów wpływają na szanse osiągnięcia sukcesu, więc mimo losowego występowania mutacji istnieje większa szansa na to, że populacja większa dostosuje się do środowiska szybciej niż populacja mniejsza, oraz na to że populacja na którą mutacja wywiera wpływ w niewielkim stopniu tzn. następuje rzadko i zmienia małe wartości dostosuje się do środowiska wolniej niż populacja, na którą mutacja oddziałuje mocniej. Dodatkowo apropo tego drugiego przypadku, empirycznie sprawdziłem też, że gdy mutacja oddziałuje na populacje zbyt mocno, nie będzie ona w stanie osiągnąć celu, dlatego ważne jest dobranie optymalnych parametrów.

5.2 Pomysły na rozwój i wykorzystanie projektu

Wymyśliłem dwa pomysły na różne zastosowania projektu. Pierwszym jest wykorzystanie symulacji w celu edukacyjnym. Na zajęciach dotyczących sztucznej inteligencji można by pokazywać studentom w jaki sposób działa algorytm genetyczny. Uczniowie mieliby szanse zobaczyć jak może wyglądać reprezentacja sztucznego neuronu oraz całej sztucznej sieci neuronowej pod postacią kodu w języku C#. Dzięki temu powinni móc lepiej zrozumieć jak sieć przetwarza sygnały oraz jak może wpływać na obiekty, które je implementują. Taka wizualna pomoc mogła by być dobrym wstępem do stworzenia własnej sztucznej sieci neuronowej. By poszerzyć ich perspektywę można by dodać do programu możliwość manualnej zmiany funkcji aktywacji z której korzysta sieć. By podkreślić to że symulacją steruję algorytm genetyczny można by rozszerzyć go o możliwość modyfikacji wielkości samochodu. Mechanizm ten musiałby losowo zmieniać szerokość i długość samochodu, a to w konsekwencji powinno przekładać się na zmiany w masie która skutkowałaby innymi wartościami przyśpieszenia oraz zwrotnością. Dzięki temu wyraźnie byłoby widać ewolucje samochodów.

Drugim sposobem jest wykorzystanie symulacji w rozrywce. Po zaimplementowaniu rozszerzonej fizyki prowadzenia pojazdu, dodaniu kilku torów wyścigowych symulacja mogła by pełnić rolę gry. Na każdym torze użytkownik wcielałby się w rolę gracza by ścigać się z osobnikiem wytrenowanym przez algorytm. Dzięki uwzględnieniu wpływu czasu przejazdu na wartość funkcji przystosowania, można by użyć konkretne generacje by zaimplementować stopniowanie modelu trudności. Po umieszczeniu takiego programu na stronie internetowej uzyskamy grę z kategorii wyścigowej.

**Podsumowanie**

Celem niniejszej pracy było stworzenie symulacji wyścigu samochodowego w którym poruszaniem samochodów sterowałyby sztuczne sieci neuronowe. Zaimplementowany algorytm genetyczny miał też zapewnić że w toku ewolucji wykształci się osobnik, który dojedzie do mety kończąc symulacje.

Symulacje sprawdzałem przynajmniej 110 razy, zmieniając jej parametry takie jak wielkość populacji, szansa na wystąpienie mutacji i moc mutacji. Jej wyniki zapisane po każdym sprawdzeniu dowodzą poprawnego działania algorytmu genetycznego, ponieważ w każdy zapisany rekord oznacza osiągnięty cel. Wyniki pokazują, że mimo pseudo losowości zastosowanej w nauce osobników poprzez mutacje, wyżej wspomniane parametry mają wpływ na prędkość z jaką populacja osiąga cel.

Zrealizowany projekt inżynierski podyktowany był osobistym zainteresowaniem nauki jaką jest szeroko rozumiana sztuczna inteligencja, a przede wszystkim zrealizowania projektu przedstawiającego program, który mimo początkowego braku odpowiednich parametrów, osiąga cel poprzez proces samodoskonalenia.

# Bibliografia

(2019, Marzec 25). Pobrano z lokalizacji https://o-m.pl/artykul/sztuczna-inteligencja-w-dzialaniach-marketingowych

*ai.c-labtech.net*. (brak daty). Pobrano z lokalizacji http://www.ai.c-labtech.net/sn/pod\_prakt.html

Brzezińska, A. N. (brak daty). Pobrano z lokalizacji https://docplayer.pl/13483611-Sieci-neuronowe-wprowadzenie-agnieszka-nowak-brzezinska.html

*computerworld.pl*. (2016, Czerwiec 19). Pobrano z lokalizacji https://www.computerworld.pl/news/Sztuczna-inteligencja-wkracza-do-samochodow,405655.html

*Dokumentacja Microsoftu*. (2021, Grudzień 28). Pobrano z lokalizacji https://docs.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/visual-studio-tools-for-unity?view=vs-2019

Gareth Woods, t. M. (2020, Październik 18). *redbull.com*. Pobrano z lokalizacji https://www.redbull.com/pl-pl/among-us-dlaczego-gra-jest-tak-popularna

Madej, K. (brak daty). *Encyklopedia Zarządzania*. Pobrano z lokalizacji https://mfiles.pl/pl/index.php/Test\_Turinga

*marketingibiznes.pl*. (2017, Czerwiec 13). Pobrano z lokalizacji https://marketingibiznes.pl/it/historii-badan-nad-ai/

*money.pl*. (2020, Wrzesień 3). Pobrano z lokalizacji https://www.money.pl/gospodarka/sztuczna-inteligencja-wkraczamy-w-nowa-faze-walki-z-pandemia-6550021129983617a.html

*Oficjalna strone parlamentu europejskiego*. (2021, Kwiecień 26). Pobrano z lokalizacji https://www.europarl.europa.eu/news/pl/headlines/society/20200827STO85804/sztuczna-inteligencja-co-to-jest-i-jakie-ma-zastosowania

Rutkowski, L. (2012). Metody i techniki sztucznej inteligencji. W L. Rutkowski. Warszawa: Wydawnictwo Naukowe PWN.

Sieja, B. (2015, marzec 12). *KomputerSwiat*. Pobrano z lokalizacji https://www.komputerswiat.pl/gamezilla/recenzje/recenzja-cities-skylines/2vwn82y

*spidersweb*. (2018, Kwiecień 12). Pobrano z lokalizacji https://spidersweb.pl/2018/04/facebook-sztuczna-inteligencja.html

*statfoft.pl*. (brak daty). Pobrano z lokalizacji https://www.statsoft.pl/textbook/glosfra\_stat.html?https%3A%2F%2Fwww.statsoft.pl%2Ftextbook%2Fglosf.html

*sztucznainteligencja*. (brak daty). Pobrano z lokalizacji https://www.sztucznainteligencja.org.pl/

Unity Technologies. (2022, Styczen 29). *Oficjalna dokumentacja Unity*. Pobrano z lokalizacji Unity Documentation: https://docs.unity3d.com/Manual/index.html

Unity Technologies. (brak daty). *Oficjalna platforma learningowa Unty*. Pobrano z lokalizacji https://learn.unity.com/

Unity Technologies. (brak daty). *Oficjalne forum Unity*. Pobrano z lokalizacji https://forum.unity.com/

WAYMO. (2021, Grudzień 30). *blog.waymo.com*. Pobrano z lokalizacji https://blog.waymo.com/2021/12/accelerating-mission-velocity.html

*Wikipedia*. (brak daty). Pobrano z lokalizacji https://wikipedia.org/

# Spis Ilustracji

[Ilustracja 1 Model neuronu 11](#_Toc95047525)

[Ilustracja 2 Model sieci neuronowej zaimplementowany w projekcie 12](#_Toc95047526)

[Ilustracja 3 Wykres funkcji Tangens hiperboliczny 13](#_Toc95047527)

[Ilustracja 4 Zrzut ekranu z edytora Unity podczas uruchomionej symulacji 17](#_Toc95047528)

[Ilustracja 5 Pojazd z 5 wiązkami czujników odległości 24](#_Toc95047529)

[Ilustracja 6 Zrzut ekranu z programu notepad++ prezentujący wyniki zapisane po skończeniu symulacji 39](#_Toc95047530)

[Ilustracja 7 Tabela uśrednionych wyników z testu sprawdzającego wpływ szansy na mutacje i siły mutacje na wyniki populacji 41](#_Toc95047531)

[Ilustracja 8 Wykres uśrednionego czasu przejazdu z testu pierwszego 42](#_Toc95047532)

[Ilustracja 9 Wykres uśrednionego numeru generacji z testu pierwszego 42](#_Toc95047533)

[Ilustracja 10 Tabela uśrednionych wyników z testu sprawdzającego wpływ wielkości populacji na wyniki populacji 43](#_Toc95047534)

[Ilustracja 11 Wykres uśrednionego czasu przejazdu z testu drugiego 43](#_Toc95047535)

[Ilustracja 12 Wykres numeru generacji z testu drugiego 44](#_Toc95047536)

# Spis kodów

[Kod 1 Zrzut ekranu metoda onTriggerEnter2D z klasy Car.cs 21](#_Toc94944407)

[Kod 2 Zrzut ekranu metoda drive z klasy Car.cs 23](#_Toc94944408)

[Kod 3 Zrzut ekranu metoda sensor z klasy Car.cs 24](#_Toc94944409)

[Kod 4 Zrzut ekranu metoda initNeuronalNetwork z klasy Manager.cs 26](#_Toc94944410)

[Kod 5 Zrzut ekranu metoda initializeCars z klasy Manager,cs 27](#_Toc94944411)

[Kod 6 Zrzut ekranu konstruktor klasy NeuralNetwork.cs 27](#_Toc94944412)

[Kod 7 Zrzut ekranu metoda initalizeLayers z klasy NeuralNetwork.cs 27](#_Toc94944413)

[Kod 8 Zrzut ekranu metoda initializeNeurons z klasy NnLayer.cs 28](#_Toc94944414)

[Kod 9 Zrzut ekranu metoda initializeWeights z klasy NeuralNetwork.cs 29](#_Toc94944415)

[Kod 10 Zrzut ekranu deklaracja zmiennej threadLocal w klasie StaticRandom.cs 29](#_Toc94944416)

[Kod 11 Zrzut ekranu metoda randomFloatNumberFromRange z klasy StaticRandom.cs 30](#_Toc94944417)

[Kod 12 Zrzut ekranu metoda feedForward z klasy NeuralNetwork.cs 31](#_Toc94944418)

[Kod 13 Zrzut ekranu metoda calculateWeights z klasy Neuron.cs 31](#_Toc94944419)

[Kod 14 Zrzut ekranu metoda mutate z klasy Neuron.cs 32](#_Toc94944420)

[Kod 15 Zrzut ekranu metoda mutateWeights z klasy Neuron.cs 32](#_Toc94944421)

[Kod 16 Zrzut ekranu metoda copyWeightsFromNetwork z klasy NeuralNetwork.cs 33](#_Toc94944422)

[Kod 17 Zrzut ekranu metoda loadWeightsFromOtherNetwork z klasy NeuralNetwork.cs 33](#_Toc94944423)

[Kod 18 Zrzut ekranu metoda initializeNextgeneration z klasy Manager.cs 35](#_Toc94944424)

[Kod 19 Zrzut ekranu metoda createNextPopulation z klasy GeneticAlgorithm.cs 35](#_Toc94944425)

[Kod 20 Zrzut ekranu metoda eliminateCarsWithLowFitnessValue z klasy GeneticAlgorithm.cs 36](#_Toc94944426)

[Kod 21 Zrzut ekranu metoda sortCarListbyFitnessvalue z klasy GeneticAlgorithm.cs 36](#_Toc94944427)

[Kod 22 Zrzut ekranu metoda replaceWorstCarsWithBestCars z klasy GeneticAlgorithm.cs 36](#_Toc94944428)

[Kod 23 Zrzut ekranu metoda endSimulationIfCarsGetsToMeta z klasy Manager.cs 38](#_Toc94944429)

1. https://spidersweb.pl/2018/04/facebook-sztuczna-inteligencja.html [↑](#footnote-ref-1)
2. https://www.money.pl/gospodarka/sztuczna-inteligencja-wkraczamy-w-nowa-faze-walki-z-pandemia-6550021129983617a.html [↑](#footnote-ref-2)
3. https://www.computerworld.pl/news/Sztuczna-inteligencja-wkracza-do-samochodow,405655.html [↑](#footnote-ref-3)
4. https://www.europarl.europa.eu/news/pl/headlines/society/20200827STO85804/sztuczna-inteligencja-co-to-jest-i-jakie-ma-zastosowania [↑](#footnote-ref-4)
5. https://marketingibiznes.pl/it/historii-badan-nad-ai/ [↑](#footnote-ref-5)
6. Sieci neuronowe wprowadzenie - Agnieszka Nowak Brzezińska [↑](#footnote-ref-6)
7. Rutkowski, L. (2012). Metody i techniki sztucznej inteligencji. W L. Rutkowski. Warszawa: Wydawnictwo Naukowe PWN. [↑](#footnote-ref-7)
8. https://mfiles.pl/pl/index.php/Test\_Turinga [↑](#footnote-ref-8)
9. https://www.sztucznainteligencja.org.pl/portfolio/1956-konferencja-w-dartmouth/ [↑](#footnote-ref-9)
10. https://en.wikipedia.org/wiki/ELIZA [↑](#footnote-ref-10)
11. https://www.sztucznainteligencja.org.pl/portfolio/1973-mycin-sztuczny-ekspert-medyczny/ [↑](#footnote-ref-11)
12. https://en.wikipedia.org/wiki/AlphaZero#Chess\_2 [↑](#footnote-ref-12)
13. https://www.sztucznainteligencja.org.pl/portfolio/2017-libratus-czyli-sztuka-blefu/ [↑](#footnote-ref-13)
14. https://blog.waymo.com/2021/12/accelerating-mission-velocity.html [↑](#footnote-ref-14)
15. https://pl.wikipedia.org/wiki/Perceptron\_wielowarstwowy [↑](#footnote-ref-15)
16. http://www.ai.c-labtech.net/sn/pod\_prakt.html [↑](#footnote-ref-16)
17. https://www.statsoft.pl/textbook/glosfra\_stat.html?https%3A%2F%2Fwww.statsoft.pl%2Ftextbook%2Fglosf.html pod hasłem „Perceptrony wielowarstwowe” [↑](#footnote-ref-17)
18. https://www.sztucznainteligencja.org.pl/kurs/sztuczna-inteligencja-dla-poczatkujacych/sztuczne-sieci-neuronowe/ [↑](#footnote-ref-18)
19. https://upload.wikimedia.org/wikipedia/commons/thumb/8/87/Hyperbolic\_Tangent.svg/800px-Hyperbolic\_Tangent.svg.png [↑](#footnote-ref-19)
20. PDF z dysku Ewa Figielska Algorytmu ewolucyjne i ich zastosowania [↑](#footnote-ref-20)
21. Leszek Rutkowski, *Metody i techniki sztucznej inteligencji*, warszawa 2012 str.242 [↑](#footnote-ref-21)
22. https://en.wikipedia.org/wiki/Unity\_(game\_engine) [↑](#footnote-ref-22)
23. https://www.redbull.com/pl-pl/among-us-dlaczego-gra-jest-tak-popularna [↑](#footnote-ref-23)
24. https://www.komputerswiat.pl/gamezilla/recenzje/recenzja-cities-skylines/2vwn82y [↑](#footnote-ref-24)
25. https://en.wikipedia.org/wiki/Hearthstone [↑](#footnote-ref-25)
26. https://docs.unity3d.com/Manual/UsingComponents.html [↑](#footnote-ref-26)
27. https://en.wikipedia.org/wiki/C\_Sharp\_(programming\_language) [↑](#footnote-ref-27)
28. https://docs.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/visual-studio-tools-for-unity?view=vs-2019 [↑](#footnote-ref-28)
29. https://learn.unity.com/tutorial/update-and-fixedupdate#5c8a4242edbc2a001f47cd63 [↑](#footnote-ref-29)
30. https://forum.unity.com/threads/what-is.138431/ [↑](#footnote-ref-30)
31. https://docs.microsoft.com/pl-pl/dotnet/api/system.random?view=net-6.0 [↑](#footnote-ref-31)
32. https://docs.microsoft.com/pl-pl/dotnet/api/system.threading.threadlocal-1?view=net-6.0 [↑](#footnote-ref-32)
33. Rekordy znajdują się w pliku rekordy próbne.txt [↑](#footnote-ref-33)
34. Rekordy znajdują się w pliku rekordy\_z\_pierwszego\_testu.txt [↑](#footnote-ref-34)
35. Plik wynikiTestów.xls [↑](#footnote-ref-35)
36. Rekordy znajdują się w pliku rekordy\_z\_drugiego\_testu.txt [↑](#footnote-ref-36)