

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-104376-92630

**NEUROEVOLÚCIA AUTONÓMNEHO VOZIDLA
DIPLOMOVÁ PRÁCA**

Bratislava 2023

Bc. Roman Ridzoň

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-104376-92630

NEUROEVOLÚCIA AUTONÓMNEHO VOZIDLA
DIPLOMOVÁ PRÁCA

Študijný program:	Robotika a kybernetika
Študijný odbor:	kybernetika
Školiace pracovisko:	Ústav robotiky a kybernetiky
Vedúci záverečnej práce/školiťel:	prof. Ing. Ivan Sekaj, PhD.

Bratislava 2023

Bc. Roman Ridzoň



ZADANIE DIPLOMOVEJ PRÁCE

Autor práce:	Bc. Roman Ridzoň
Študijný program:	robotika a kybernetika
Študijný odbor:	kybernetika
Evidenčné číslo:	FEI-104376-92630
ID študenta:	92630
Vedúci práce:	prof. Ing. Ivan Sekaj, PhD.
Vedúci pracoviska:	prof. Ing. Jarmila Pavlovičová, PhD.
Miesto vypracovania:	Ústav robotiky a kybernetiky
Názov práce:	Neuroevolúcia autonómneho vozidla
Jazyk, v ktorom sa práca vypracuje:	slovenský jazyk
Špecifikácia zadania:	<p>Neuroevolúcia je efektívny spôsob učenia umelých neurónových sietí, ktorý môže byť s výhodou použitý pre návrh riadenia dynamických systémov. Cieľom práce je návrh metodiky riadenia autonómnych vozidiel prostredníctvom neurónových sietí využitím evolučných algoritmov s cieľom dosiahnuť ich optimálne správanie sa v priestore z pohľadu zvolených kritérií.</p> <p>Úlohy:</p> <ol style="list-style-type: none">1. Naštudujte problematiku neuroevoúcie a autonómnych vozidiel (AV).2. Navrhните modely správania sa a metódy riadenia AV.3. Navrhните a naprogramujte algoritmy riadenia AV.4. Navrhnuté prístupy overte simulačne.5. Dosiahnuté výsledky vyhodnoťte.
Termín odovzdania práce:	12. 05. 2023
Dátum schválenia zadania práce:	20. 12. 2022
Zadanie práce schválil:	prof. Ing. Jarmila Pavlovičová, PhD. garantka študijného programu

Pod'akovanie

Ďakujem prof. Ing. Ivanovi Sekajovi, PhD. za odborné rady a nápady pre ďalšie rozšírenie pokrytia práce. Bez jeho podpory by táto práca bola len škrupinkou toho čo dosiahla.

ANOTÁCIA DIPLOMOVEJ PRÁCE

Slovenská technická univerzita v Bratislave
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný odbor: Kybernetika

Študijný program: Robotika a kybernetika

Autor: Bc. Roman Ridzoň

Diplomová práca: Neuroevolúcia Autonómneho Vozidla

Vedúci diplomovej práce: prof. Ing. Ivan Sekaj, PhD.

Mesiac, rok odovzdania: Máj, 2023

Kľúčové slová: Autonómne vozidlo, neuroevolúcia, UNITY, NEAT, evolučný algoritmus, genetický algoritmus, OpenDRIVE

Cieľom práce je vytvoriť na platforme unity Game engine prostredie pre simuláciu modelu autonómneho vozidla a jeho tréning pomocou NEAT algoritmu. Vozidlo úspešne dokončí zadané jazdné scenáre iba na základe senzorov, bez zásahu z vonku.

Riadenie vozidla bude riešené pomocou neurónovej siete s plne prepojenou štruktúrou, natrénovanou pomocou algoritmu neuroevolúcie. Vstupom budú dáta zo senzorov, akcelerometer, radar a kamera. Neurónová sieť bude mať jednu skrytú vrstvu. Výstup bude prírastok k cieľovej polohe natočenia volantu vozidla, hĺbky stlačenia plynového pedálu a hĺbky stlačenia brzdového pedálu.

Testovacie scenáre úspešnosti natrénovaného vozidla budú prebiehať v modeli postaveného vo formáte ASAM OpenDRIVE. Tento formát opisuje geometriu ciest, pruhov a objektov, ako sú značky na ceste a taktiež aj prvky pozdĺž ciest, ako sú signály. Ide o bezplatný štandard čo sa používa ako základ pre autonómne vozidlá.

MASTER THESIS ABSTRACT

Slovak University of Technology in Bratislava
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION
TECHNOLOGY

Branch of Study: Cybernetics

Study Programme: Robotics and Cybernetics

Author: Bc. Roman Ridzoň

Master Thesis: Neuroevolution of Autonomous Vehicle

Supervisor: prof. Ing. Ivan Sekaj, PhD.

Year, Month: 2023, May

Keywords: Autonomous car, neuroevolution, UNITY, NEAT, evolution
algorithm, genetic algorithm, OpenDRIVE

The aim of the thesis is to create an environment on the unity game engine platform for the simulation of an autonomous vehicle model and its training using the NEAT algorithm. The vehicle successfully completes defined driving scenarios based only on sensors, without outside intervention.

Vehicle control will be solved using a neural network with a fully connected structure, trained using a neuroevolution algorithm. The input will be data from sensors, accelerometer, radar and camera. The neural network will have one hidden layer. The output will be the increment to the vehicle's target steering wheel turning position, gas pedal depression depth, and brake pedal depression depth.

Test scenarios of the success of the trained vehicle will take place in models built in the OpenDRIVE format. Format describes the geometry of roads, lanes and objects such as road markings as well as features along roads such as signals. It is a free standard that is used as a basis for autonomous vehicles.

Obsah

Zoznam použitých skratiek	9
Úvod	10
1 Umelé neurónové siete	11
1.1 Neurón.....	11
1.2 Aktivačné funkcie	12
1.2.1 Sigmoid	12
1.2.2 Tanh.....	12
1.3 Štruktúra neurónovej siete.....	13
2 Neuroevolúcia	14
2.1 Genetický algoritmus	14
2.2 Neat	15
2.3 Augumentácia topológie	15
2.4 Fixná topológia.....	16
3 Simulátor	17
3.1 Unity 3D.....	17
3.2 Autonómne vozidlo	18
3.3 Senzory.....	19
3.3.1 Akcelerometre	19
3.3.2 Radar	19
3.3.3 Navigácia Globálna.....	20
3.3.4 Navigácia lokálna.....	21
3.4 UnitySharpNEAT.....	21
3.5 Optimalizácia simulátora.....	22
3.5.1 Fixná štruktúra	22
3.5.2 Jedna generácia	23
3.5.3 Ukladanie modelov	24
3.5.4 Použitie natrénovaných modelov	25
4 OpenDRIVE	26

4.1	RoadRunner.....	26
4.2	Scenario.....	27
4.3	Implementácia	28
5	Scenáre	29
5.1	Dávanie prednosti.....	30
5.1.1	Trénovanie1.....	31
5.2	Viac križovatiek	32
5.2.1	Trénovanie2.....	34
5.3	Porovnanie.....	36
	Záver	41
	Literatúra	43
	Použité súčasti	44

Zoznam použitých skratiek

NS – neurónová sieť

GA – genetický algoritmus

NEAT – neuroevolúcia pomocou augmentujúcej topológie

OpenDRIVE – otvorený štandard v automobilovom priemysle

Unity – prostredie pre 3D simuláciu

Path tool – špeciálny balík na prácu s krivkami v Unity

Úvod

V posledných rokoch si neuroevolúcia získala čoraz väčšiu pozornosť ako výkonná metóda na vývoj umelých neurónových sietí, ktoré dokážu vyriešiť zložité problémy v širokej škále domén. Základnou myšlienkou neuroevolúcie je použiť evolučné algoritmy na prehľadávanie priestoru možných architektúr a váh neurónových sietí s cieľom nájsť najefektívnejšiu sieť pre danú úlohu.

Tento prístup umožňuje vývoj vysoko flexibilných a adaptabilných systémov, ktoré možno trénovať na riešenie zložitých problémov. V tejto práci budeme skúmať potenciál neuroevolúcie pre vývoj inteligentných systémov so zameraním na autonómne vozidlá.

1 Umelé neurónové siete

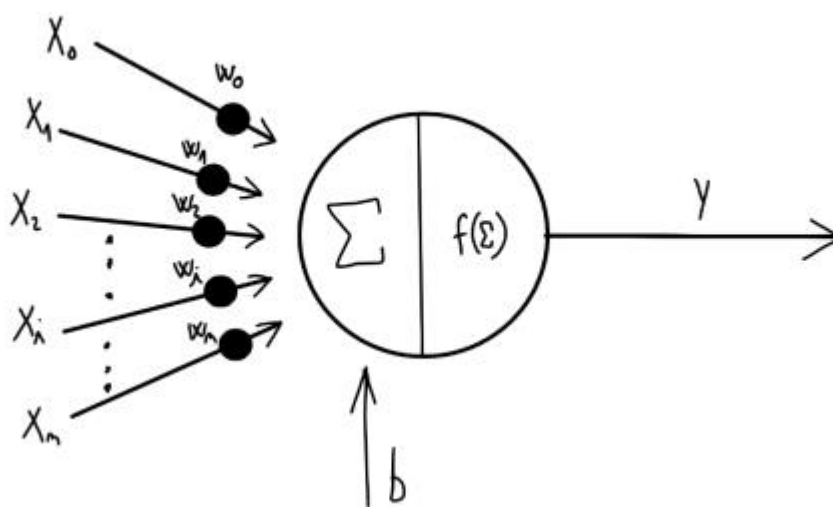
Neurónové siete sú súborom algoritmov inšpirovaných štruktúrou a funkciou ľudského mozgu. Používajú sa na rozpoznávanie vzorov a vytváranie predpovedí na základe údajov. Neurónové siete pozostávajú z vrstiev vzájomne prepojených uzlov, ktoré sa tiež nazývajú neuróny.

Existuje niekoľko typov neurónových sietí, vrátane dopredných neurónových sietí, rekurentných neurónových sietí, konvolučných neurónových sietí a iných. Každý typ má svoju vlastnú štruktúru a účel, ale všetky používajú rovnaké základné princípy spracovania vstupných údajov cez vrstvy neurónov na vytváranie výstupných predpovedí.

Neurónové siete sa úspešne používajú v širokej škále aplikácií, vrátane rozpoznávania obrazu a reči, spracovania prirodzeného jazyka, autonómnych vozidiel a mnohých ďalších.

1.1 Neurón

Každý neurón prijíma vstupné signály z iných neurónov alebo z externých údajov a spracováva ich prostredníctvom súboru váh a predpätí, aby vytvoril výstupný signál. Výstupný signál jedného neurónu môže byť prepojený so vstupom iného neurónu v ďalšej vrstve. Tento proces pokračuje, kým posledná vrstva neurónov nevytvorí výstup neurónovej siete.[1]



Obrázok 1: Neurón

1.2 Aktivačné funkcie

Aktivačné funkcie sú kľúčovým komponentom umelých neurónových sietí a používajú sa na zavedenie nelinearity do siete. Sú to matematické funkcie, ktoré sa aplikujú na výstupy každého neurónu v sieti. Aktivačná funkcia určuje, či neurón „vystrelí“ alebo „aktivuje“ na základe vstupu, ktorý dostane.

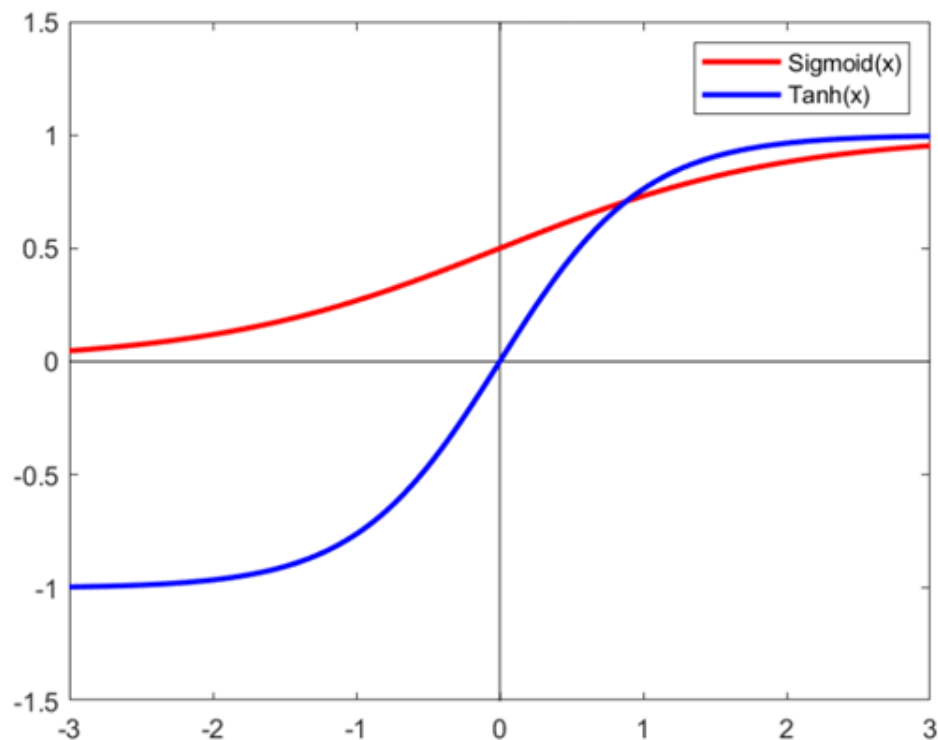
Výber aktivačnej funkcie závisí od daného problému a architektúry neurónovej siete. Nelinearita zavedená aktivačnými funkciami je dôležitá na to, aby sa sieť naučila zložité vzorce a vzťahy v údajoch.[1]

1.2.1 Sigmoid

Sigmoid je najjednoduchšie popísaná svojou funkciou, $y = 1 / (1 + e^{(-x)})$. Asymptoty tejto funkcie sa berú ako jej minimum a maximum a ležia v bodoch 0 a 1.

1.2.2 Tanh

Tanh (Hyperbolický tangens): Funkcia Tanh je podobná sigmoid funkcii, ale mapuje akúkoľvek vstupnú hodnotu na hodnotu medzi -1 a 1. Túto funkciu sme používali vo všetkých neurónoch čo sa v práci budú ďalej spomínať.

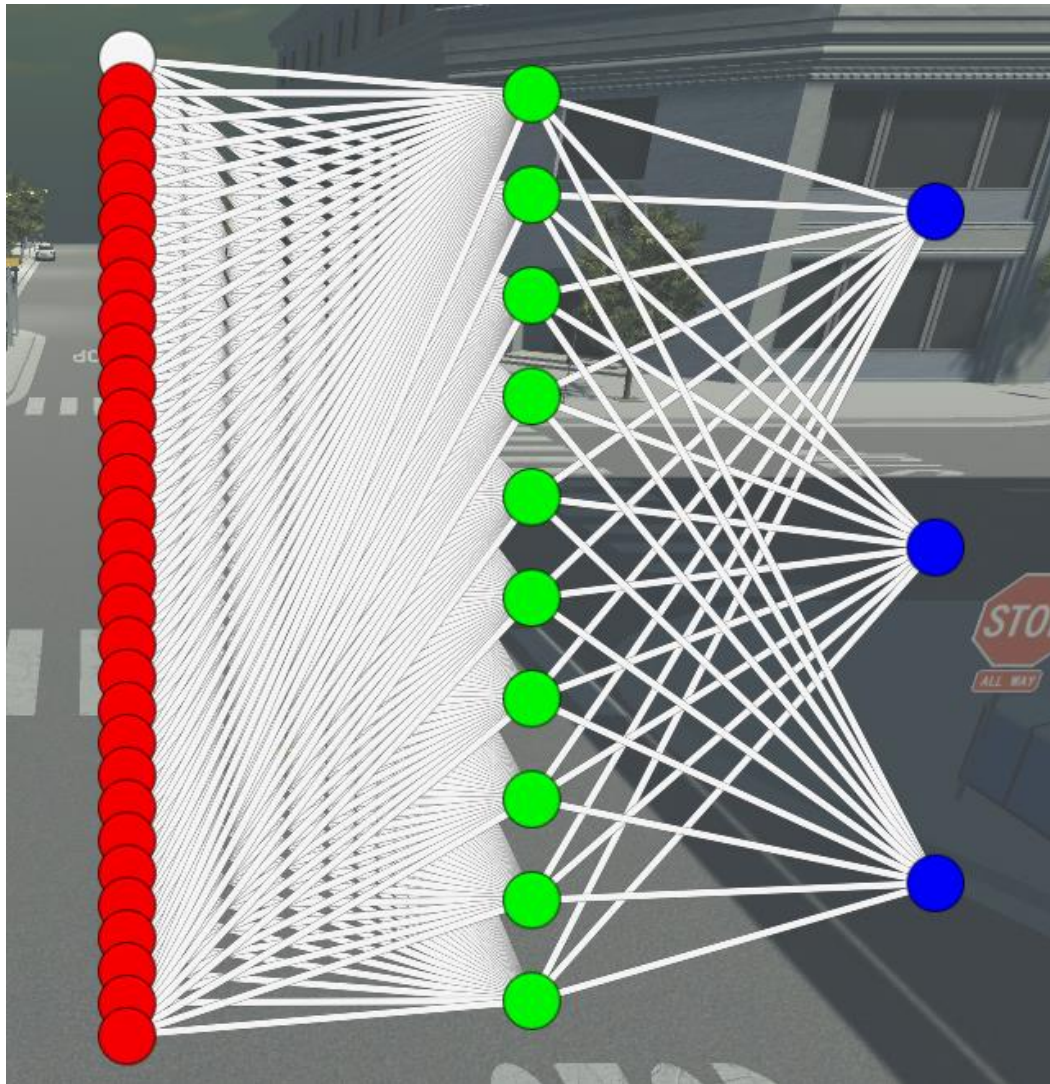


Obrázok 2: Krivky aktivačných funkcií

1.3 Štruktúra neurónovej siete

Klasická štruktúra neurónovej siete má tri vrstvy. Vstupnú vrstvu kde na vstup neurónov v tejto vrstve sú privedené externé údaje. Skrytú vrstvu kde vstupmi aj výstupmi sú napojenia na iné neuróny v neurónovej sieti. Výstupnú vrstvu kde vstupmi sú iné neuróny a výstupy sa posielajú vonku zo siete.[1]

V tejto práci bude primárne používaná jedna plne prepojená skrytá vrstva. Vstupnými externými údajmi budú senzory na autonómnom vozidle. Výstupné údaje sa budú používať na obsluhovanie natočenia volantu, plynového pedálu a brzdového pedálu.



Obrázok 3: Vizualizácia NS v Unity

2 Neuroevolúcia

Neuroevolúcia je podoblasť umelej inteligencie a strojového učenia, ktorá využíva evolučné algoritmy na tréning umelých neurónových sietí na riešenie zložitých problémov. Tento prístup zahŕňa použitie genetických algoritmov alebo iných evolučných techník na vývoj neurónových sietí prostredníctvom generácií selekcie, mutácie a rekombinácie.

Základnou myšlienkou neuroevolúcie je napodobniť proces prirodzenej evolúcie výberom najvýkonnejších neurónových sietí a ich použitím ako rodičov na produkciu potomkov s malými obmenami. Tieto variácie sa zavádzajú prostredníctvom mutácií, kríženia alebo iných genetických operátorov, ktoré modifikujú štruktúru a váhy neurónových sietí.

Postupom času proces výberu a variácie umožňuje populácii neurónových sietí prispôbiť sa problémovému priestoru a optimalizovať ich výkon pri špecifickej úlohe. Tento prístup ponúka niekoľko výhod oproti iným technikám strojového učenia, ako je hlboké učenie, ktoré si vyžaduje veľké množstvo údajov a je obmedzené architektúrou neurónovej siete.[2]

2.1 Genetický algoritmus

Genetický algoritmus je typ evolučného algoritmu, ktorý sa používa na riešenie optimalizačných problémov. Je inšpirovaný procesom prirodzeného výberu a genetiky. Napodobňuje proces evolúcie s cieľom nájsť najlepšie možné riešenie daného problému.

V genetickom algoritme sa populácia kandidátskych riešení inicializuje náhodne. Každé riešenie je reprezentované ako reťazec hodnôt, ktorý môže predstavovať parametre riešeného problému. Tieto hodnoty sa označujú ako gény a reťazec sa označuje ako genóm.

Počas procesu evolúcie genetický algoritmus vyhodnocuje spôsobilosť každého genómu na základe fitness funkcie, ktorá je mierou toho ako dobre rieši problém. S väčšou pravdepodobnosťou budú vybraní do ďalšej generácie zdatnejšie genómy.

Genetický algoritmus potom používa rôzne genetické operátory, ako je crossover a mutácia, na vytvorenie nových chromozómov pre ďalšiu generáciu. Crossover zahŕňa kombináciu dvoch rodičovských chromozómov na vytvorenie nového chromozómu potomstva, zatiaľ čo mutácia zahŕňa náhodnú zmenu jedného alebo viacerých génov v chromozóme.

Proces selekcie, kríženia a mutácie sa opakuje počas viacerých generácií, kým nie je splnené kritérium zastavenia. Kritériom zastavenia by mohol byť maximálny počet generácií, minimálna hodnota vhodnosti alebo určitá úroveň konvergenzie.

Konečným riešením je chromozóm s najvyššou fitness hodnotou v konečnej populácii. Genetický algoritmus môže nájsť riešenie na širokú škálu optimalizačných problémov, vrátane problémov s mnohými premennými alebo obmedzeniami, ktoré je ťažké vyriešiť pomocou tradičných optimalizačných metód.

Jednou z výhod genetických algoritmov je ich schopnosť efektívne prehľadávať veľké priestory riešení. Môžu však byť aj výpočtovo drahé a vyžadujú starostlivý výber parametrov, ako je veľkosť populácie, miera kríženia a rýchlosť mutácií, aby sa dosiahol dobrý výkon.[2]

2.2 Neat

NeuroEvolution of Augmenting Topologies je typ neuroevolučného algoritmu, ktorý sa používa na vývoj umelých neurónových sietí. NEAT bol vyvinutý Kennethom Stanleyom v roku 2002.

Algoritmus NEAT začína jednoduchou architektúrou neurónovej siete a vyvíja architektúru v priebehu niekoľkých generácií pomocou procesu nazývaného „speciation“. Speciácia zahŕňa rozdelenie populácie neurónových sietí do podskupín alebo „druhov“ na základe ich genetickej podobnosti. To umožňuje algoritmu udržiavať rozmanitosť v rámci populácie a zároveň podporovať výmenu užitočného genetického materiálu medzi druhmi.

NEAT tiež používa fitness funkciu na vyhodnotenie výkonu každej neurónovej siete. Funkcia fitness meria, ako dobre sieť funguje pri konkrétnej úlohe alebo probléme. Siete s vyšším skóre s väčšou pravdepodobnosťou prežijú a reprodukovujú sa.[3]

2.3 Augumentácia topológie

Počas procesu evolúcie môže NEAT pridávať alebo odstraňovať uzly a spojenia v neurónovej sieti, čím vytvára nové štruktúry, ktoré môžu lepšie vykonávať danú úlohu. To umožňuje NEAT vyvíjať v priebehu času zložitejšie architektúry neurónových sietí, pričom si stále zachováva svoju funkčnosť.

Algoritmus NEAT tiež používa metódu nazývanú "historické značky" na sledovanie toho, ako sa architektúra siete časom vyvíjala. Historické značenie umožňuje algoritmu sledovať, ktoré uzly a spojenia sú nové alebo boli upravené, čo uľahčuje sledovanie vývoja štruktúry neurónovej siete.[3]

2.4 Fixná topológia

NEAT taktiež dokáže fungovať s fixnou topológiou. Počas procesu evolúcie dochádza iba k crossover a mutácii váh medzi vopred navrhnutou štruktúrou neurónovej siete. Tento spôsob sa používa hlavne pri náročnejších problémoch kde sa riešenie môže objaviť až po určitej komplexnosti štruktúry. To umožňuje preskočiť niekoľko stoviek generácií než by sa pomocou augmentácie dosiahla dostatočná komplexnosť.[3]

V práci sme začali s jednou skrytou vrstvou zloženou z 10 neurónov, Obrázok 3. V neskorších častiach sme však museli zvýšiť komplexnosť na 20 neurónov nakoľko 10 nebolo dostatočných pre naše potreby.

3 Simulátor

Zatiaľ čo testovanie v reálnom svete je nevyhnutné na overenie výkonu systému, simulátory sú kritickým nástrojom v procese vývoja, ktorý umožňuje vývojárom testovať a hodnotiť systém v širokej škále scenárov a podmienok. Používanie simulátorov oproti skutočným experimentom na testovanie a hodnotenie automatizovaných vozidiel má niekoľko výhod:

Bezpečnosť: Simulátory poskytujú bezpečné prostredie na testovanie a hodnotenie automatizovaných vozidiel bez rizík spojených s testovaním v reálnom svete. Simulátory umožňujú vývojárom otestovať systém v kontrolovanom prostredí a zabezpečiť, aby bol systém bezpečný pred jeho nasadením na verejné cesty.

Nákladová efektívnosť: Simulátory sú nákladovo efektívne v porovnaní s testovaním v reálnom svete, čo môže byť drahé a časovo náročné. Simulátory umožňujú rýchle testovanie a vyhodnocovanie rôznych scenárov, čím sa znižuje čas a náklady spojené s testovaním v reálnom svete.

Reprodukovateľnosť: Simulátory umožňujú reprodukovateľnosť experimentov a umožňujú vývojárom opakovať experimenty za rovnakých podmienok na overenie výsledkov.

Flexibilita: Simulátory umožňujú modifikáciu parametrov a premenných, čo umožňuje vývojárom ľahko testovať a vyhodnocovať rôzne scenáre a podmienky. Táto flexibilita umožňuje vývojárom optimalizovať výkon systému a zlepšiť jeho funkčnosť.

3.1 Unity 3D

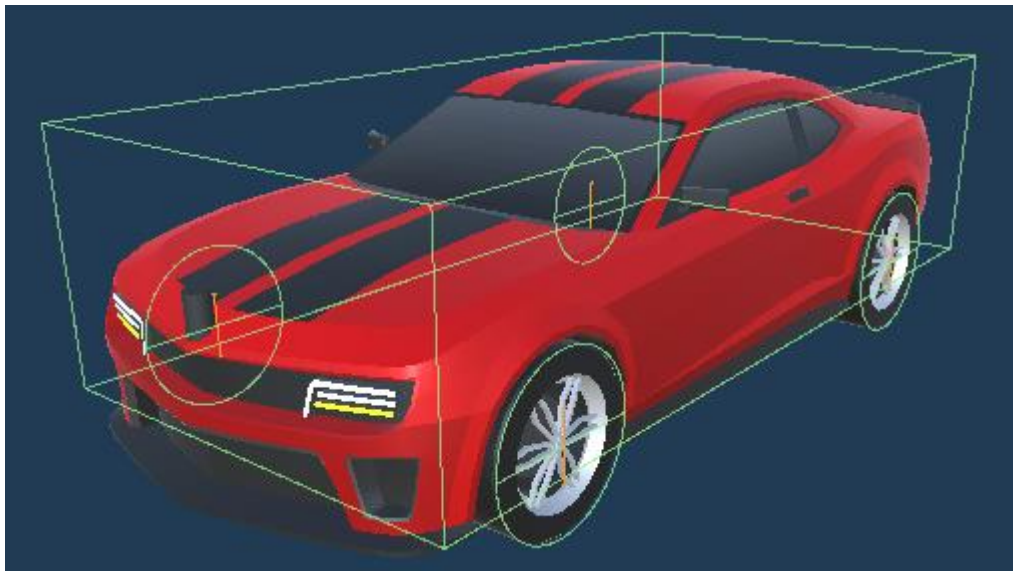
Unity je prostredie primárne používané na vytváranie hier. Exceluje v 3D vizualizácii v reálnom čase čo sa využíva aj v architektúre či pri výrobe animovaných filmov. Vďaka jeho mnohým možnostiam a dlhodobej podpore existuje okolo unity veľká komunita, ktorá vytvára mnoho nástrojov čo nie sú súčasťou základnej inštalácie.

Počas celej práce sme používali verziu Unity 2020.3.40f1, ktorá má dlhodobú podporu. Programovacím jazykom pre unity je C# a spolu s ním sme využili json formát na uskladnenie natrénovaných modelov.

Pre simuláciu tréningu autonómnych vozidiel v prostredí unity sme potrebovali vyplniť tri oblasti. NEAT algoritmus čo bude zabezpečovať tréning. Simulované vozidlo čo sa bude správať čo najbližšie k reálnemu. Senzory čo umožnia vozidlu vnímať okolie. Pre všetky tri oblasti sme čerpali z balíkov v open-source doméne. Z tohto dôvodu použité balíky vyžadovali značné úpravy alebo museli byť vytvorené.

3.2 Autonómne vozidlo

Model vozidla sme čerpali z tutoriálu na vytvorenie fyzicky simulovaného vozidla. Išlo o relatívne modulárny prístup, z ktorého sme použili iba hlavný skript pre transformáciu vstupu na pohyb kolies.



Obrázok 4: Vozidlo

Vstupnými údajmi sú natočenie volantu $\langle -1; +1 \rangle$, stlačenie plynového pedálu $\langle 0; +1 \rangle$ a stlačenie brzdového pedálu $\langle 0; +1 \rangle$. Vozidlo ma jeden box collider a štyri kolesové čo tvoria základ vozidla. Týchto päť objektov je fyzicky simulovaných na základe rigidbody komponentu prostredia unity. V rigidbody sme nastavili váhu vozidla na 2,8 ton.

Predné kolesá na vozidle sme nastavili ako riadenie a maximálny uhol ich vychýlenia je 35° . Zadné kolesá dostávajú v každom kroku simulácie krútiaci moment z motora na základe stlačenia plynového pedálu vynásobeného parametrom maximálnej sily motora čo sme nastavili na 5000. Brzdna sila je prenášaná na všetky kolesá na základe stlačenia brzdového pedálu vynásobeného parametrom maximálnej brzdnej sily nastaveného na 7000.

Posledný komponent sme pridali stabilizačné tyče medzi predné a zadné kolesá. Tento komponent tak ako v skutočných autách prevádza silu medzi kolesami na opačnej strane vozidla aby zabezpečil stabilitu, hlavne pri zákrutách. Keď sa jedno z kolies zatlačí nahor, tyč stabilizátora prenesie časť tejto kompresnej sily na druhé koleso aby sa zatlačilo aj jeho zavesení.

3.3 Senzory

Účelom senzorov na autonómnom vozidle je zhromažďovať a spracovávať informácie o okolí vozidla. Tieto údaje sú potom filtrované a predstavujú vstupy do neurónovej siete. V prostredí unity sú vstupy aktualizované vo funkcii `UpdateBlackBoxInputs()`, ktorá sa volá ako súčasť fyzikálnej slučky. Táto slučka sa vykoná podľa potreby viac násobne za každú snímku a odohráva sa v nej aj prepočet pohybu objektov s fyzikálnymi vlastnosťami.

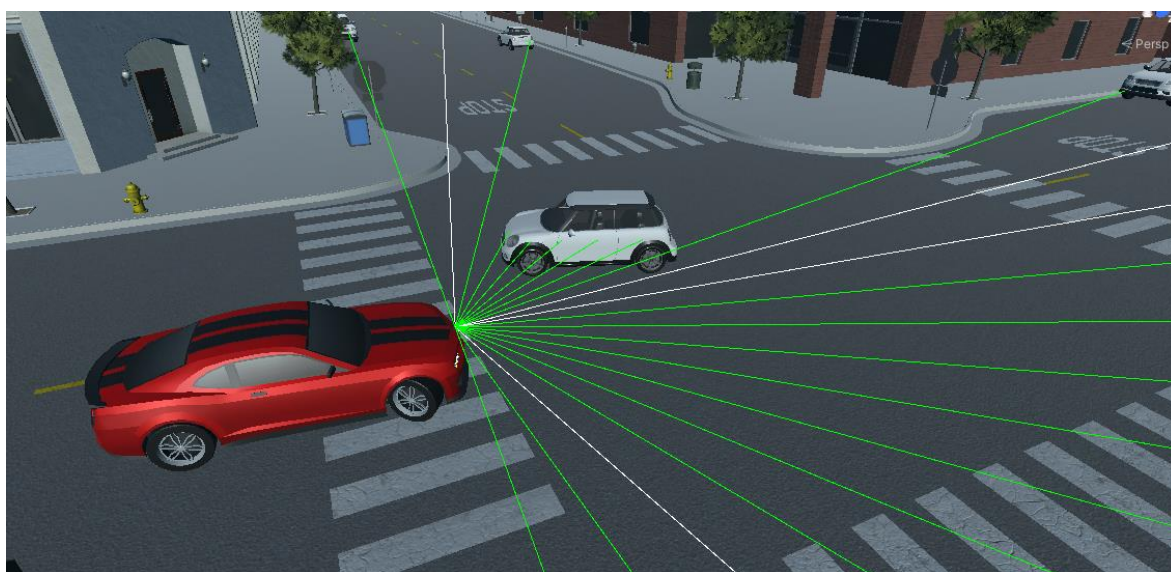
3.3.1 Akcelerometre

Komponent rigidbody je nutnosťou pre fyzikálnu simuláciu objektov v prostredí unity. Drží taktiež všetky potrebné informácie o pozícii a silách pôsobiacich na objekt. Z neho teda získavame informácie o rýchlosti vozidla v metroch za sekundu.

3.3.2 Radar

Simulovanie radaru sa odohráva pomocou vysielania lúčov do priestoru pred vozidlom. 21 lúčov s rozmedzím 9° stupňov pokrýva 180° stupňov pred vozidlom. Script radar má zadefinované parametre vrstvy objektov, s ktorými sa lúče môžu zraziť. V prípade celej práce to bola iba vrstva WALL, na ktorú sme umiestnili všetky objekty mapy a taktiež ostatné vozidlá v križovatke.

Druhým dôležitým parametrom je maximálna vzdialenosť lúčov čo bola nastavená na päťdesiat metrov. Formát údajov čo radar vracal mal podobu poľa, kam sa údaje normalizovali v podobe `nájdená/maximálna vzdialenosť`. V prípade ak lúč do ničoho nenarazil, vrátil hodnotu 1.



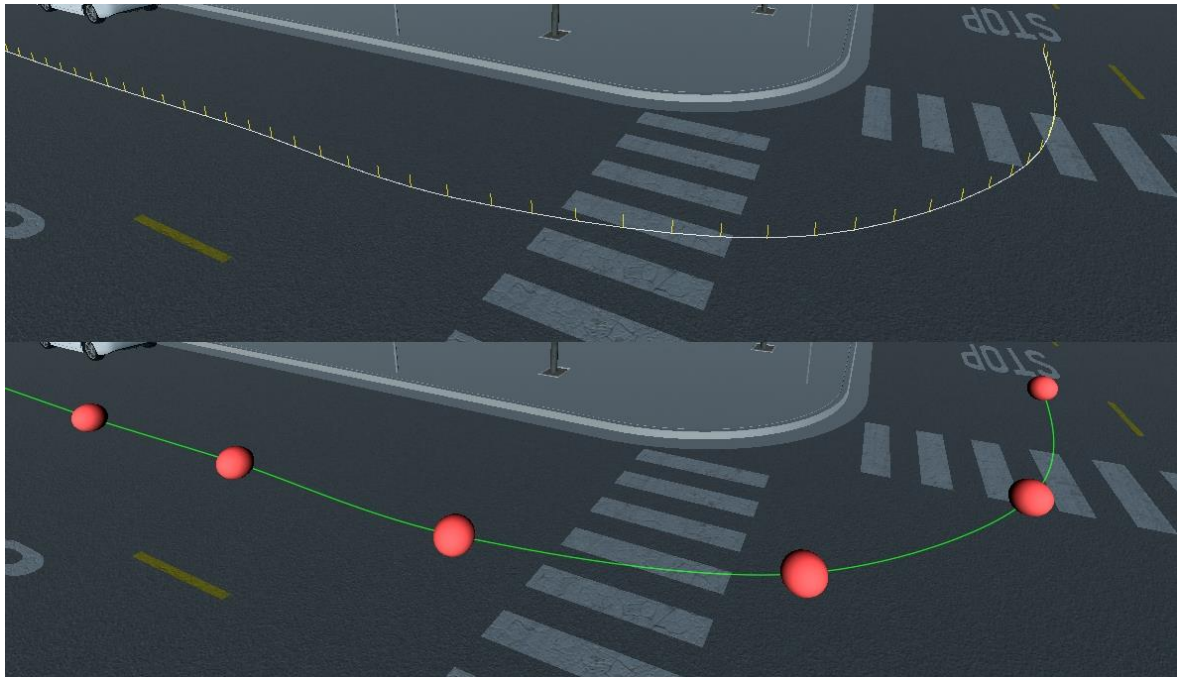
Obrázok 5: Radar

3.3.3 Navigácia Globálna

Globálna navigácia bola riešená formou bezierovej krivky. Balíček path tool má mechaniku rozdelenia trajektórie na sekciu bodov. Parametre pre toto rozdelenie je uhol zmeny smeru a minimálna vzdialenosť medzi bodmi. Tieto parametre sme nastavili na hodnoty 0° pre maximálnu zmenu uhla a 0,5 metrov na minimálnu vzdialenosť.

Tieto nastavenia dosiahnu aby kontrolné body mali absolútnu toleranciu k dodržiavaniu bezierovej krivky a ich rozptyl bude 0,5 metra. Ich vzdialenosť môže byť väčšia v priamych úsekoch, avšak nie viac ako 1 meter.

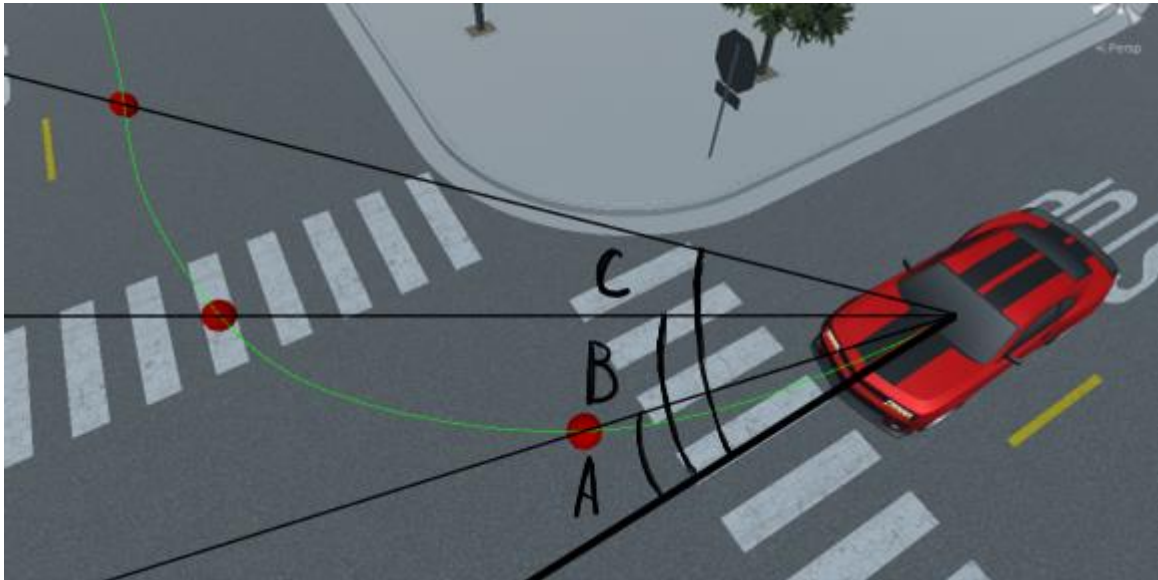
Na hornej časti Obrázok 6 môžeme vidieť vrcholy navigačnej trajektórie. V spodnej časti červené body označujú vrcholy bezierovej krivky. Počas simulácie, pre zrýchlenie výpočtov autonómne vozidlá pracujú iba s vrcholmi navigačnej trajektórie. Vrchol ku ktorému je vozidlo najbližšie označuje súčasnú pozíciu a v každom kroku sa pozeráme iba na jeden bod do oboch smerov. Ak vzdialenosť jedného z blízkych bodov je menšia ako k súčasnemu pozičnému vrcholu, ten sa stane novým pozičným vrcholom.



Obrázok 6: Path tool

3.3.4 Navigácia lokálna

Počas simulácie berieme do úvahy vrcholy na trajektórii 10, 20 a 30 krokov pred vozidlom. To nám v priemere dá body 5, 10 a 15 metrov pred vozidlom. S priamkou v smere vozidla ako počiatkom, uhly A, B a C používame ako vstupy do neurónovej siete. Tie sú normalizované tým že ich vydelíme 180° čo nám v akejkol'vek pozícii dá hodnoty $\langle -1;1 \rangle$.



Obrázok 7: Lokálna navigácia

V neurónovej sieti taktiež používame vzdialenosť k bodom ako druhý parameter pre každý z 3 vrcholov trajektórie. Tie sú vďaka parametrom lokálnej navigácie väčšinu času v rovnakej vzdialenosti od vozidla. Hrajú ale dôležitú rolu keď vozidlo dosiahne koniec trajektórie. V prípade že vrchol 30 krokov pred vozidlom neexistuje, uhol sa nastaví na 0° a vzdialenosť na -1, čo pomohlo neurónovej sieti spojiť vstupný parameter s brzdením.

3.4 UnitySharpNEAT

UnitySharpNEAT je open-source balík, ktorý integruje knižnicu SharpNEAT. SharpNEAT je knižnica s otvoreným zdrojovým kódom pre NEAT v jazyku C# a implementuje tento algoritmus. Balík UnitySharpNEAT rozširuje funkčnosť knižnice SharpNEAT tým, že poskytuje užívateľsky prívetivé rozhranie v rámci unity.

Balík poskytuje množstvo funkcií vrátane schopnosti trénovať a vyvíjať neurónové siete v reálnom čase, vizualizovať proces vývoja a exportovať výsledné siete na použitie v produkčných prostrediach. Balík tiež obsahuje príklad primitívneho vozidla, ktoré používateľom pomôžu začať používať SharpNEAT v Unity. To v teórii umožňuje vývojárom jednoducho integrovať evolučné algoritmy do svojich projektov unity.

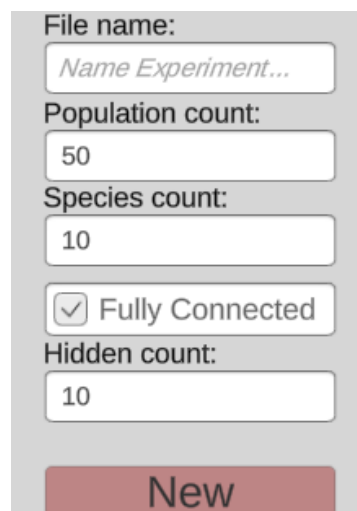
3.5 Optimalizácia simulátora

UnitySharpNEAT bol jediným NEAT orientovaným balíkom čo bolo možné nájsť pre unity. Zabezpečuje však slabú integráciu knižnice SharpNEAT. Tvorca integroval iba časti nutné na ukážku jeho príkladu. Parametre evolúcie boli všetky skryté hlboko v skriptoch a nebola tam možnosť viacerých možností pre rozdielne experimenty. Veľká časť kódu si teda vyžadovala prepisovanie, obzvlášť v oblasti rozhrania medzi knižnicou SharpNEAT a unity, čo mal balík zabezpečovať.

3.5.1 Fixná štruktúra

UnitySharpNEAT vo svojom základe funguje ako klasický NEAT algoritmus. Po vytvorení jedincov pre ďalšiu generáciu, časť z nich prejde mutáciou. Tieto mutácie sa vyberajú ruletovým výberom ich možnosti sú zmena hodnoty váhy, pridanie neurónu, pridanie váhy a vymazanie váhy. Nakoľko však v práci sme chceli použiť fixnú štruktúru, toto bolo nežiadúce.

Pre zachovanie funkčnosti aj normálneho NEAT algoritmu pre testovanie sme odhalili pravdepodobnosti na ruletovom výbere. Následne pri vytváraní novej neurónovej siete sme pridali možnosť či chceme používať klasický NEAT alebo fixnú plne prepojenú štruktúru. Podľa výberu sa potom nastaví pravdepodobnosti a vytvorí sa prvá generácia. V prípade normálneho NEAT sa použije na prvú generáciu pôvodný algoritmus. Pre plne prepojenú verziu sme pridali nový spôsob vytvorenia prvej generácie. V štruktúre sa vytvorí všetky skryté neuróny a ich prepojenia ešte pred začiatkom tréningu. Spolu so zablokovaním pravdepodobností mutácií čo môžu zmeniť štruktúru sme tak dosiahli aby zostala nemenná.



File name:

Population count:

Species count:

☒ Fully Connected

Hidden count:

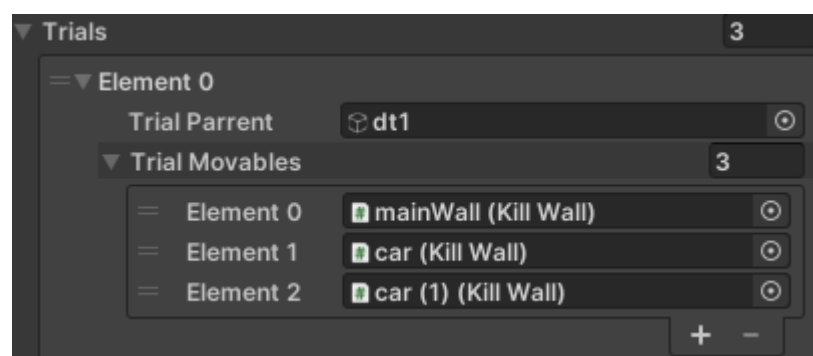
Obrázok 8: UI pre novú NS

3.5.2 Jedna generácia

V UnitySharpNEAT balíku, jedna generácia začala v bode počiatku scény a po stanovenom čase bola ukončená. Podľa fit funkcie zadefinovanej v hlavnom skripte autonómneho vozidla potom bola generácia vyhodnotená a najlepšimi jedincami sa stali tí čo dosiahli najvyššie hodnoty. Fit funkciu bolo možné jednoducho upravovať, jedinou podmienkou bolo vrátiť hodnotu na konci behu.

Balík mal zadefinovaný parameter trials čo mal spôsobiť že jedna generácia prejde scénou viackrát než bude vyhodnotená. Implementácia tejto funkcionality však končila pri odhalení tejto premennej. Ak bola nastavená na hodnotu vyššiu ako jedna, generácia bola jednoducho simulovaná po nastavený maximálny čas behu jednej generácie vynásobeným parametrom trials. Výsledná fit funkcia bola potom priemer z týchto rôznych časových úsekov. Keďže ale nedochádzalo k resetovaniu jedincov, toto bol nežiadúci efekt.

Prvou úpravou teda bolo implementovať toto resetovanie na pôvodnú pozíciu a vyčistenie zapamätaných hodnôt fit funkcie. Nakoľko k úpravám dochádzalo naprieč väčšie množstvo skriptov, súčasne sme pridali aj možnosť rôznych experimentov v jednej generácii. Autonómne vozidlo nasleduje path tool bezierovú krivku označenú tagom „lap“ teda funkcionality bola pridaná na prepínanie aktívnych prvkov pod objektom Trial Parent ako je možné vidieť na Obrázok 9. Pri dokončení jedného trial sa uloží fit hodnota pre každého jedinca a presunie sa na ďalší. Po dokončení všetkých trials sa vezmú všetky výsledné fit hodnoty a ich priemer sa stane výslednou hodnotou fit pre daného jedinca pre danú generáciu.



Obrázok 9: Príklad trial

Trial Movables, Obrázok 9, predstavuje skripty čo sedia na objektoch a pomenované sú príznačne Kill Wall. Myšlienka ako zrýchliť prechod každej generácie sa zrodila zo sledovania počiatku tréningu. Množstvo vozidiel iba sedelo na mieste kým nevypršal čas

určený ako maximálny čas tréovania. Ako motiváciu k pohybu pre neurónovú sieť sa teda zrodila smrtiaca stena, čo po stanovenom čase vyrazila z počiatku scény a zničila všetky vozidlá pri kontakte. Následne sme pridali funkcionality na sledovanie koľko vozidiel sa ešte stále pohybuje scénou a koľko ich bolo zničených. Po zničení všetkých vozidiel sme jednoducho zavolali ich fit funkcie ako keby vypršal čas pre ich trial.

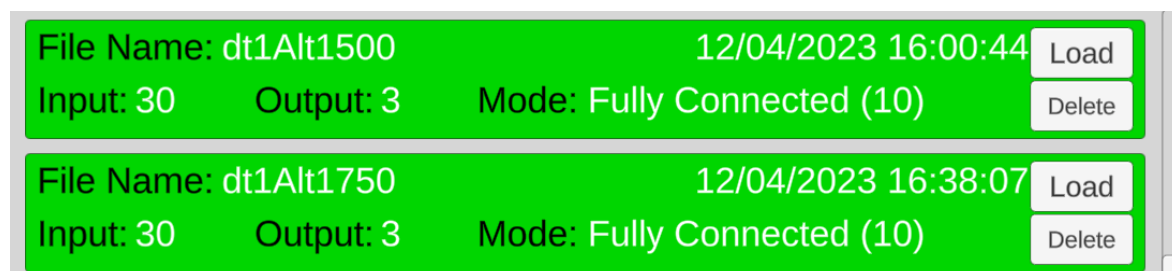
Každý Kill Wall skript môže zadať rýchlosť pohybu prepojeného objektu a ako dlho má čakať než svoj pohyb začne. Keďže taktiež fungovali podľa trajektórií path tool, nakoniec bol tento skript použitý aj na simulovanie ostatných vodičov v scéne. Umožnilo nám to tak zadať rozmanité križovatky aj keď sa scéna nezmenila.

3.5.3 Ukladanie modelov

Ukladanie v balíku UnitySharpNEAT bolo riešené pomocou štruktúrovaného xml súboru. Skrz tento systém bolo možné uložiť iba jeden experiment, nakoľko bol natvrdo naviazaný na jedinou cestu kam sa ukladalo a načítavalo. Pri ukladaní sa previedla celá súčasná generácia do xml formátu a taktiež sa uložil samostatne najlepší jedinec označený ako šampión.

Nakoľko ukladanie rôznych experimentov a ich načítavanie oddelene od cesty bola užitočná funkcionality, celý systém bol postavený od základov nanovo. Ukladanie je riešené pomocou json súborov a aj štruktúra uložených údajov bola minimalizovaná iba na nutné prvky. To značne zmenšilo veľkosť súborov a zvýšilo rýchlosť ukladania a načítavania. Nová štruktúra taktiež drží meta dáta o dátume uloženia, vstupoch, výstupoch a forme použitého NEAT algoritmu.

Všetky uložené modely sú zobrazené pri načítaní aby bolo možné pokračovať v tréovaní z akéhokoľvek bodu. Pridaný bol taktiež systém na priebežné ukladanie kde je možné definovať interval počtu generácií kde systém zastaví aby uložil všetky neurónové siete v generácii a taktiež šampióna. Obrázok 10 zobrazuje uložené verzie experimentu menom „dt1Alt“ uložené v generácii 1500 a 1750.

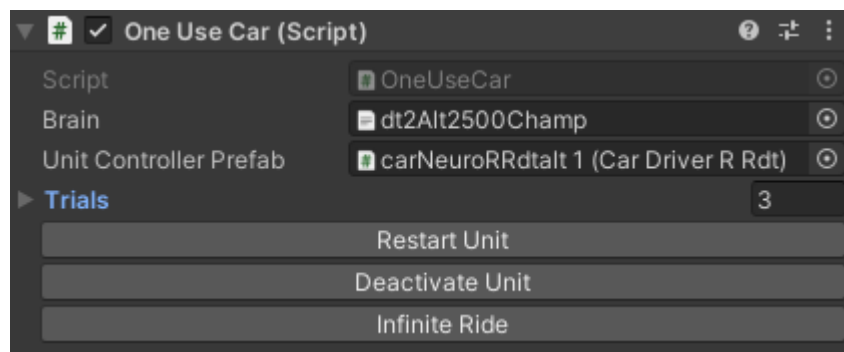


Obrázok 10: UI pre uložené modely

3.5.4 Použitie natrénovaných modelov

Použiť natrénovaný model bolo v UnitySharpNEAT limitované na spustenie šampióna v rovnakej podobe ako pri tréningu. Vyžadovalo si to aby uložený xml súbor bol na rovnakom mieste ako ho balík uložil, čo značne limitovalo testovanie. Pri prerábaní ukladania modelov bol teda aj tento systém upravený.

Uložené modely šampiónov je teraz možné priamo vložiť do prostredia unity a ako textový súbor ich vložiť do skriptu OneUseCar, do kolónky brain. Unit Controller Prefab predstavuje naše vozidlo čo sa používalo pri tréningu. Trials sa správa podobne ako pri tréningu a je tak možné preskakovať medzi rôznymi experimentami pomocou tlačidla Restart Unit. Nie je však nutné Trials naplniť, jedinou podmienkou pre spustenie použitých natrénovaných modelov je aby v scéne bola aspoň jedna path tool krivka s tagom „lap“. Tlačidlo Deactivate Unit deaktivuje vozidlo. Tlačidlo Infinite Ride spôsobí, že hneď po náraze bude autonómne vozidlo opäť aktivované v ďalšom Trials vstupe.



Obrázok 11: Použitie NN modelov

4 OpenDRIVE

ASAM OpenDRIVE je otvorený a štandardizovaný formát na reprezentáciu cestných sietí a súvisiacich prvkov, ako sú dopravné značky, signály a značenie jazdných pruhov. Formát je udržiavaný Asociáciou pre štandardizáciu automatizácie a meracích systémov a je široko používaný v automobilovom priemysle a priemysle autonómneho riadenia.

OpenDRIVE definuje hierarchickú štruktúru na reprezentáciu cestných sietí, ktorá zahŕňa nasledujúce úrovne: cesta, jazdný pruh, úsek jazdného pruhu a bočný profil. Na každej úrovni formát obsahuje atribúty, ktoré popisujú geometriu, topológiu a ďalšie charakteristiky zodpovedajúceho prvku cestnej siete.

Jednou z kľúčových vlastností OpenDRIVE je jeho schopnosť reprezentovať zložité cestné geometrie a topológie, ako sú križovatky, kruhové objazdy a rampy. Zahŕňa tiež podporu pre viacero typov jazdných pruhov, ako sú normálne jazdné pruhy, odbočovacie pruhy a núdzové pruhy. Taktiež aj rôzne cestné prvky, ako sú semaforey, značky zastavenia a obmedzenia rýchlosti.

Štandard OpenDRIVE bol široko prijatý v automobilovom priemysle a používajú ho rôzne spoločnosti a organizácie, vrátane výrobcov automobilov, dodávateľov a výskumných inštitúcií. Štandard sa naďalej vyvíja s novými funkciami a vylepšeniami, aby vyhovovali vyvíjajúcim sa potrebám odvetvia.

4.1 RoadRunner

OpenDRIVE je navrhnutý tak, aby sa dal ľahko integrovať s inými simulačnými a modelovacími nástrojmi, ako sú simulátory jazdy a softvér na simuláciu premávky. Obsahuje tiež podporu pre aplikácie v reálnom čase, vďaka čomu je vhodný na použitie v systémoch autonómneho riadenia.

Jedným z týchto prostredí je RoadRunner od spoločnosti Mathworks. Predstavuje interaktívne prostredie kde je možné intuitívne navrhovať mapy v súlade so štandardom OpenDRIVE. Podporuje taktiež funkcionality na ich exportovanie vo rôznych formátoch, vrátane fbx čo je potrebné pre unity.

V tomto prostredí sme sa inšpirovali príkladom mapy `down_town`, ktorá v sebe mala rozsiahlu reprezentáciu križovatky čo sme potrebovali. Jej komplexnosť bola až priam príliš vysoká, preto sme mapu zjednodušili. Zredukovanie budov iba na dva typy znížilo množstvo exportovaných textúr a zmenou mierky sme dosiahli aby stále vyzerali rozdielne.

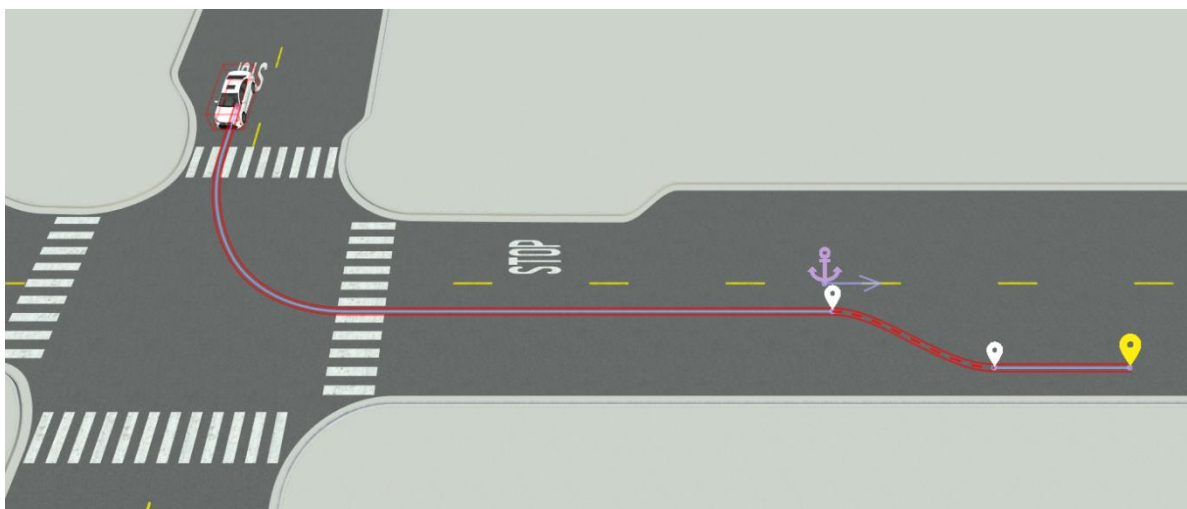


Obrázok 12: RoadRunner

4.2 Scenario

Súčasťou software RoadRunner je taktiež navrhovanie scenárov. Kliknutím do scény je možné vytvoriť vozidlo naviazané na kotvu. Následne je možné dodatočnými kliknutiami vytvoriť trajektóriu cesty. Na Obrázok 13 je možné vidieť príklad jednej takejto trajektórie. Z počiatočnej pozície vozidlo zabočí doľava skrz križovatku a následne zaparkuje vo vedľajšom pruhu pri parkovacom automate.

Pri exportovaní scenárov taktiež dochádza k exportovaniu mapy, preto sme spravili ešte jednoduchšiu verziu kde sme odstránili všetko okrem cesty. Následne sme vygenerovali súbory scenárov pre všetky možnosti prechodu križovatky. Zo všetkých smerov do všetkých smerov nám dalo dvanásť možných scenárov



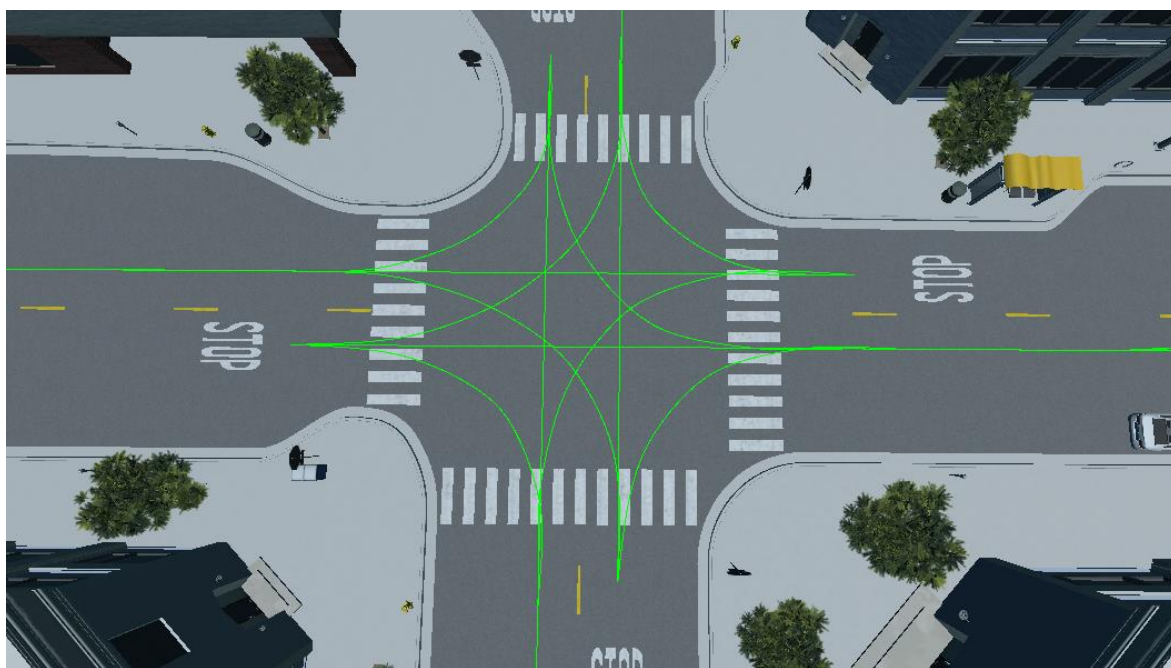
Obrázok 13: RoadRunner Scenario

4.3 Implementácia

Pri implementácii OpenDRIVE do prostredia unity nám pomohol balík Environment Simulator Minimalistic (esmini). Ide o prehrávač OpenSCENARIO súborov, ako ktoré sme vygenerovali v predchádzajúcej kapitole. Zameriava sa na čítanie týchto súborov a následné poskladanie minimalistickej simulácie kde je možné scenáre prehrať aj mimo prostredia RoadRunner čo je platené a teda nie všetci majú k nemu prístup.

ESMini má integráciu pre unity kde je možné v scéne prehrať OpenSCENARIO súbory. Toto sme využili na získanie bodov beziérovej krivky pre path tool. Skrip čo riešil prevod súradníc z OpenSCENARIO do systému súradníc v unity sme upravili aby na začiatku vytvoril path tool objekt pre každé vozidlo v scenári. Následne podľa parametru čo je možné upravovať sme určili časový interval ako často sa do path tool vložil nový vrchol beziérovej krivky. Čas čo sme experimentálne vybrali pre dostatočné výsledky bol 0,25 sekúnd.

Na koniec sme do unity importovali aj kompletnú mapu v fbx a trasy získané zo všetkých dvanástich scenárov sme vložili do jedinej mapy. Získali sme tak knižnicu všetkých možných prechodov križovatkou, čo sme použili k vybudovaniu scenárov na tréning a testovanie.

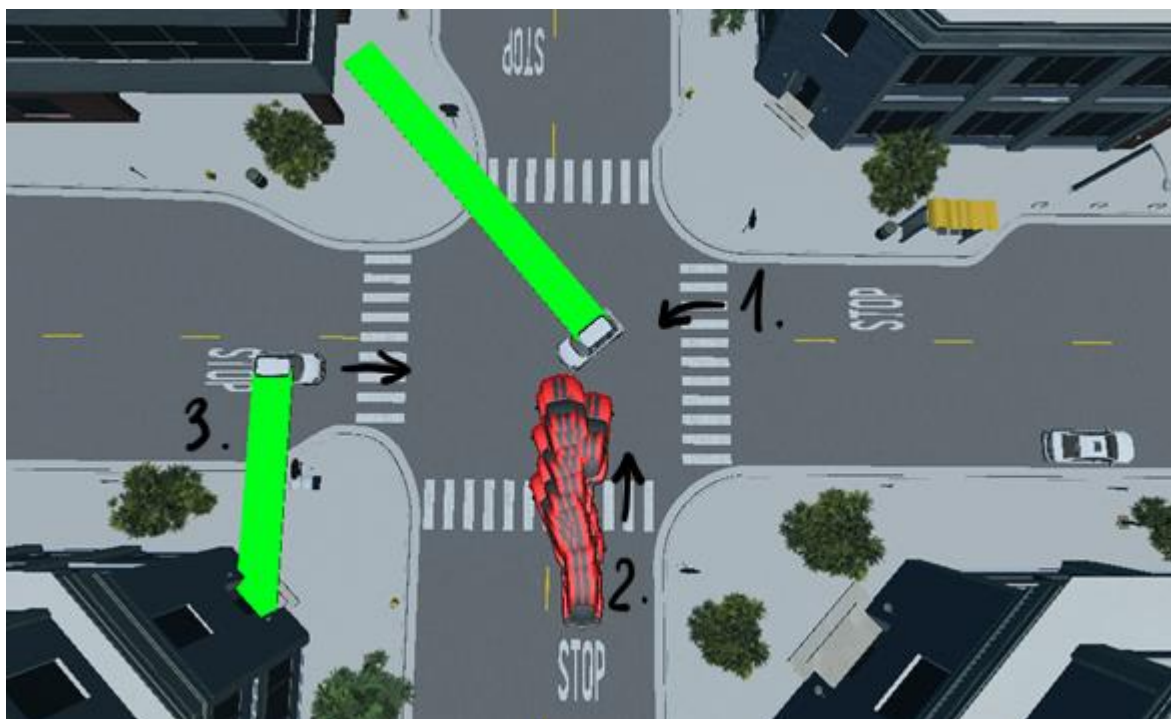


Obrázok 14: Path tool trasy

5 Scenáre

Pri navrhovaní scenárov pre tréning a testovanie sme sa inšpirovali testami z autoškoly. Ako cieľ sme si vybrali natrénovať autonómne vozidlo k prechodu križovatkou. Podmienkou je správne dať prednosť v jazde na križovatke bez svetelnej signalizácie. Všetky vozidlá začínajú zo zastavenia a tie mimo autonómneho dodržiavajú svoj postup v prechode podľa nastaveného času oneskorenia.

Pre zabezpečenie postupnosti prechodu križovatkou sme k vozidlám pridali bariéry čo vozidlo nedokáže vnímať avšak zničia ho v prípade keď sa pokúsil obísť scenár. Na Obrázok 15 je zelenou možné vidieť ich podobu. Podľa pravidiel premávky vozidlo číslo 1 má prejsť križovatkou ako prvé. V prípade že sa autonómne vozidlá označené číslom 2 pokúsia túto postupnosť obísť, tým že prejdú z ľavej strany vozidla číslo jedna, bariéra ich zničí na mieste kde sú. Podobne, vozidlo číslo 3 má bariéru na pravej strane aby zničilo autonómne vozidlá, čo by čakali kým bude celá križovatka prázdna než by sa pohli.



Obrázok 15: Bariéry prednosti

Vozidlá nedostávali žiadne dodatočné pokuty za to akým spôsobom boli zničené. Pokutované boli v prípade ak prekročili maximálnu povolenú rýchlosť 8,3 m/s (30 km/h). Pokuty dostali taktiež za koncovú vzdialenosť od cieľového bodu a súhrn odchýlky od navigačnej trajektórie. Pre zabránenie extrémnych údajov nakoľko odchýlka sa počítala

minimálne tridsaťkrát za každú snímku a množstvo snímok nebolo vždy konštantné, sme odchýlku brali do úvahy iba v miestach kde sa zmenil vrchol na navigačnej trajektórii. To zabezpečilo oddelenie týchto hodnôt od vplyvu zaťaženia simulátora, ktorý bol značne vyšší na začiatku každej generácie než jedinci začali umierať vplyvom nárazov do okolia.

Odmeňovanie jedincov bolo za základe koľkými vrcholmi prešli. Pri každom prechode vrcholov sa taktiež kontrolovalo či jedinec bol na ceste. To ale veľmi rýchlo prestalo hrať rolu nakoľko vozidlá nevychádzali na chodník stredom vozidla odkiaľ sa vysielal lúč na kontrolu čo bolo pod vozidlom (cesta alebo chodník). Nakoniec sa k výslednej fit hodnote pripočítalo 1000, vzhľadom na to že použitý balík UnitySharpNEAT prestal fungovať ak dostal negatívne hodnoty fit funkcie. Pripomíname tu že balík funguje s cieľom maximalizovať fit hodnoty.

Počas tréovania sme logovali fit hodnoty najlepšieho jedinca, avšak po dokončení tréovania sa ukázalo že všetky logované dáta boli zmenené na celé čísla. Stratila sa tak časť dát. Celočíselné údaje však zotrvali takže nie sme úplne v tme ako tréovanie prebiehalo.

5.1 Dávanie prednosti



Obrázok 16: Križovatka 1

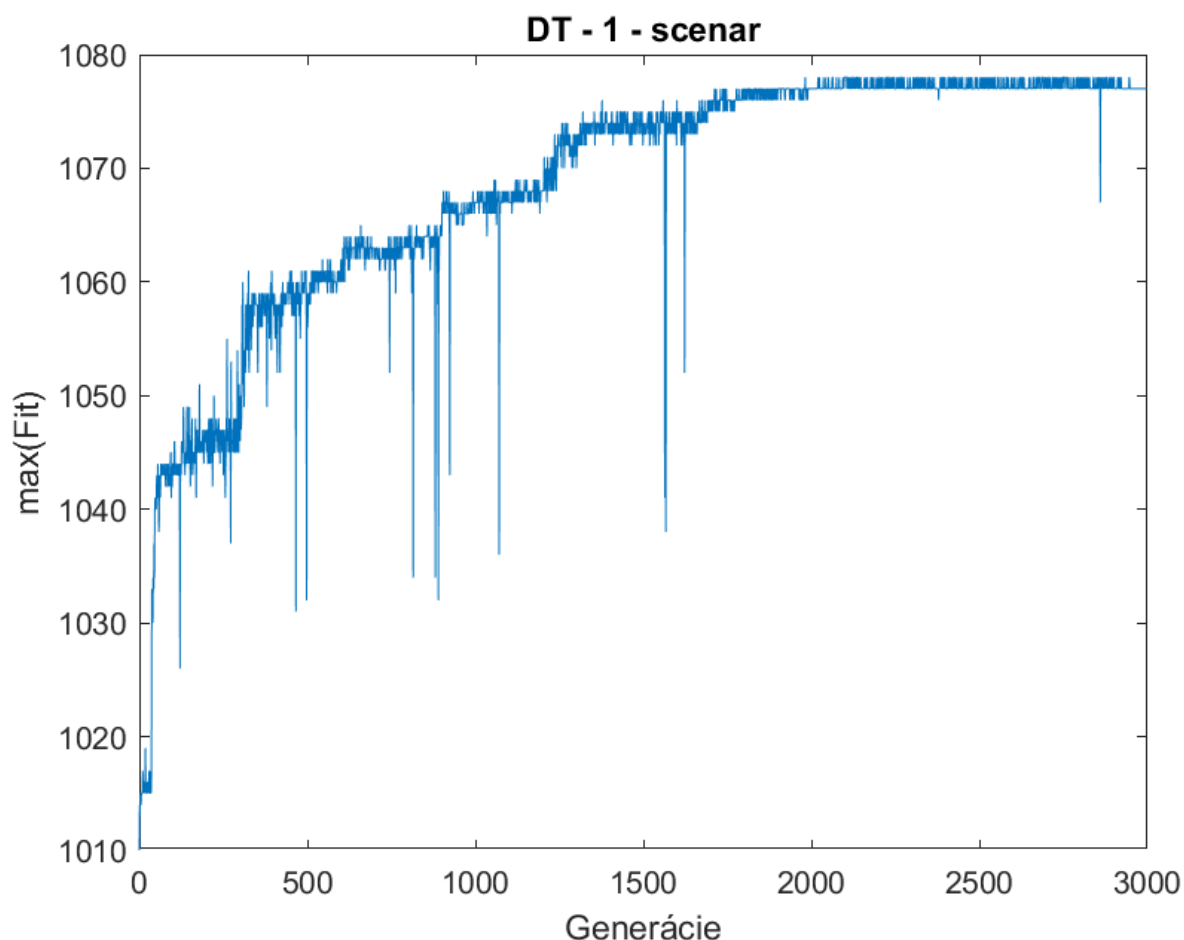
Pre prvý scenár sme pre autonómne vozidlo vybrali priamy prechod cez križovatku a zaparkovanie pri chodníku. Pohnúť sa malo ako druhé vozidlo v poradí. Autonómne

vozidlo musí dať prednosť sprava a potom rýchlo prejsť križovatkou pred tretím vozidlom. Zaručenie dávania prednosti sme dosiahli ako je možné vidieť na Obrázok 15.

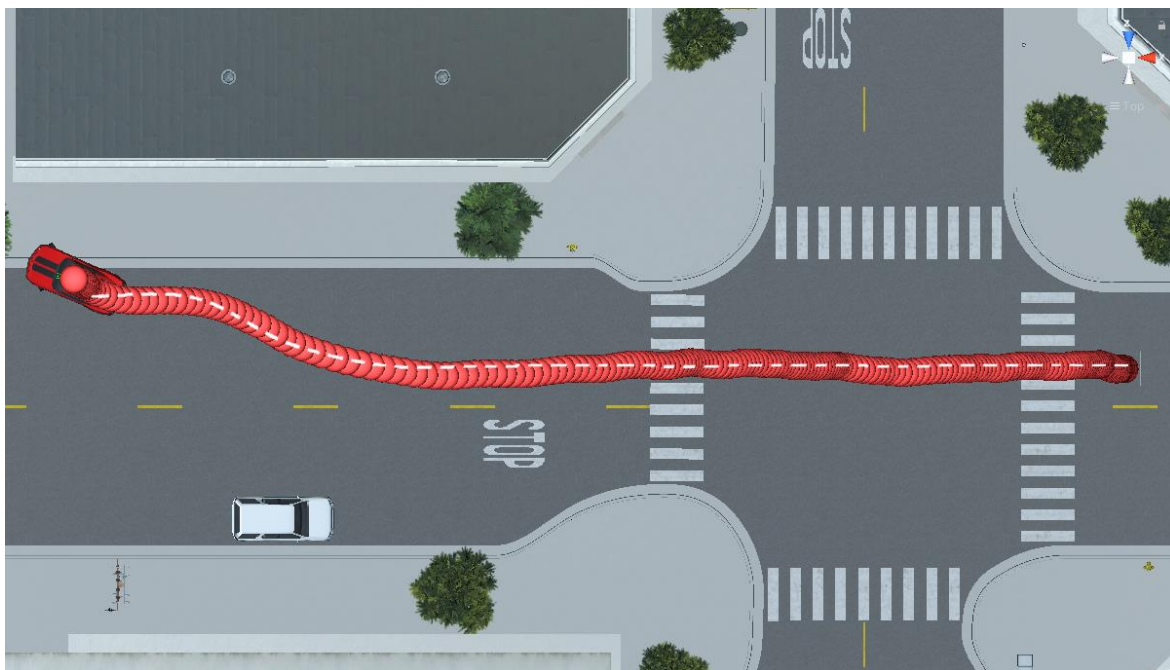
Vozidlo 1 sa pohlo s oneskorením 1 sekundu. To sme vybrali nakoľko skript čo pohyb ovládal nemal rozbiehanie a pohybovalo sa teda hneď od začiatku nastavenou rýchlosťou 8,3 m/s (30 km/h). Vozidlo 3 sa pohlo s oneskorením 5 sekúnd. Scenár bol použitý dvakrát pre každú generáciu a priemer úspešnosti jedinca medzi dvoma behmi križovatkou sa stal jeho výslednou fit hodnotou. Použitá neurónová sieť mala v skrytej vrstve 10 neurónov.

5.1.1 Trénovanie1

Celkový čas tréningu bol 24 hodín s 50 jedincami. Každých 500 generácií sme si uložili celú populáciu a jej šampióna. Pri generácii 3000 sa začali ukazovať znaky pretrénovania. I keď fit funkcia dosiahla o málo vyššie hodnoty, bolo to vďaka zneužitiu parametrov simulátora, menovite maximálneho času na jednu generáciu. Neurónová sieť sa naučila že ak pôjde pomalšie, simulácia sa ukončí v koncovom bode bez toho aby musela brzdiť. Preto pre záverečné testovanie použijeme šampióna z generácie 2500.



Obrázok 17: Fit tréningu križovatky 1



Obrázok 18: DT1-2500 v križovatke 1

5.2 Viac križovatiek

Nakoľko sa pri prvom teste ukázalo že neurónová sieť je schopná naučiť sa prejsť jednou križovatkou pre druhý test sme sa rozhodli otestovať jej schopnosť zovšeobecňovania. K prvému scenáru sme vybrali dva ďalšie tak sa mohla neurónová sieť trénovať na všetkých možnostiach v rámci križovatky s tromi autami. Počas prvého tréningu sa ukázalo že 10 skrytých neurónov nebolo dostatočných, preto sme ich počet zvýšili na 20.

Križovatka 2 ako je možné vidieť na Obrázok 19 vyžaduje od autonómneho vozidla aby prešlo križovatkou ako posledné. K tomu sme pridali zabočenie doľava aby vozidlo vedelo aj viac ako len ísť rovno. Vozidlá 1 a 2 mali na sebe bariéry na zabezpečenie prednosti v pravej strane aby zničili všetky vozidlá čo nečakali. Vozidlo 1 má oneskorenie než sa pohne 1 sekundu. Vozidlo 2 má oneskorenie než sa pohne 4 sekundy.



Obrázok 19: Križovatka 2

Križovatka 3 ako je možné vidieť na Obrázok 20 vyžaduje od autonómneho vozidla aby zabočilo doprava. V tomto prípade sa všetky vozidlá majú pohnúť súčasne. Nakoľko sa vozidlá neblokujú v jazde, neboli na nich umiestnené bariéry prednosti. Vozidlo 1 a 2 majú obe oneskorenie než sa pohnúť 1 sekundu.



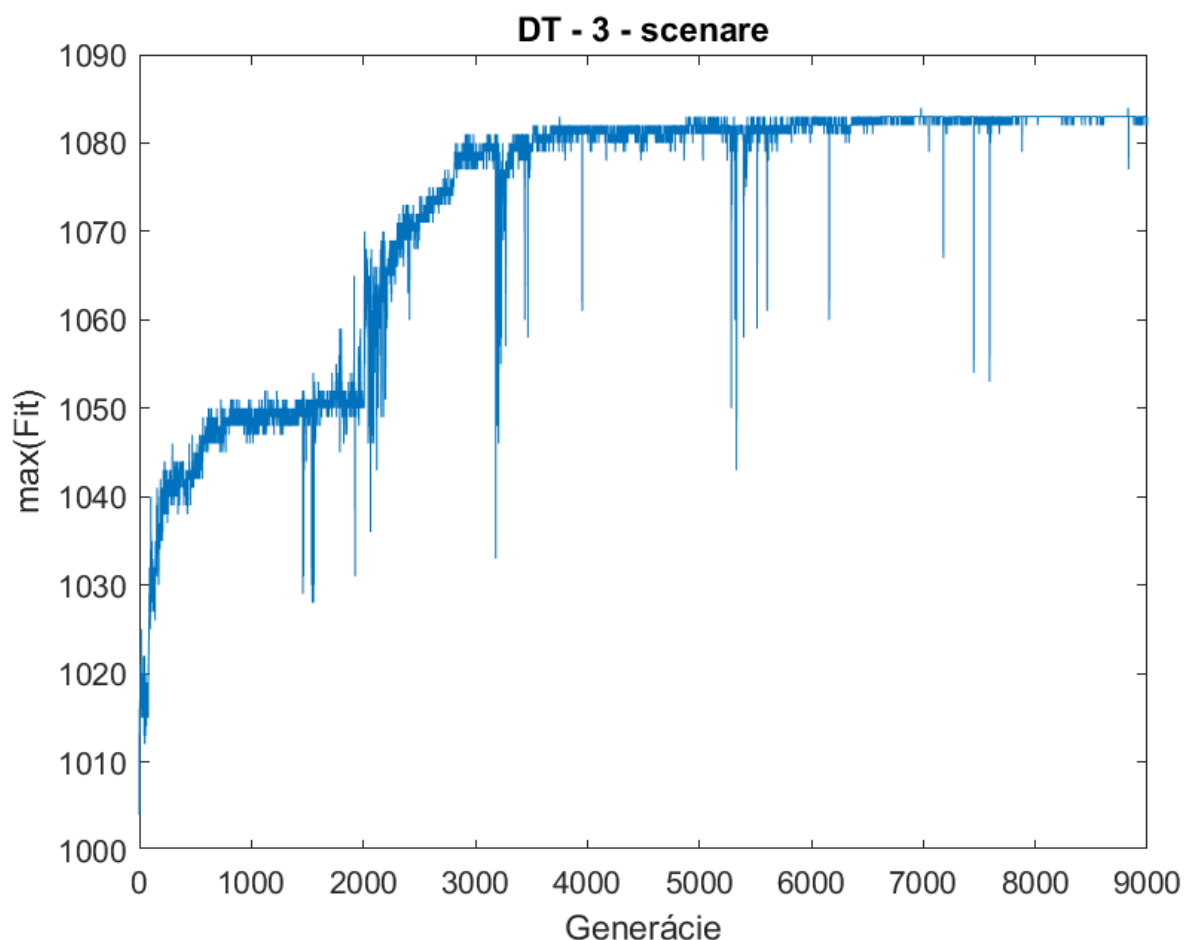
Obrázok 20: Križovatka 3

5.2.1 Trénovanie2

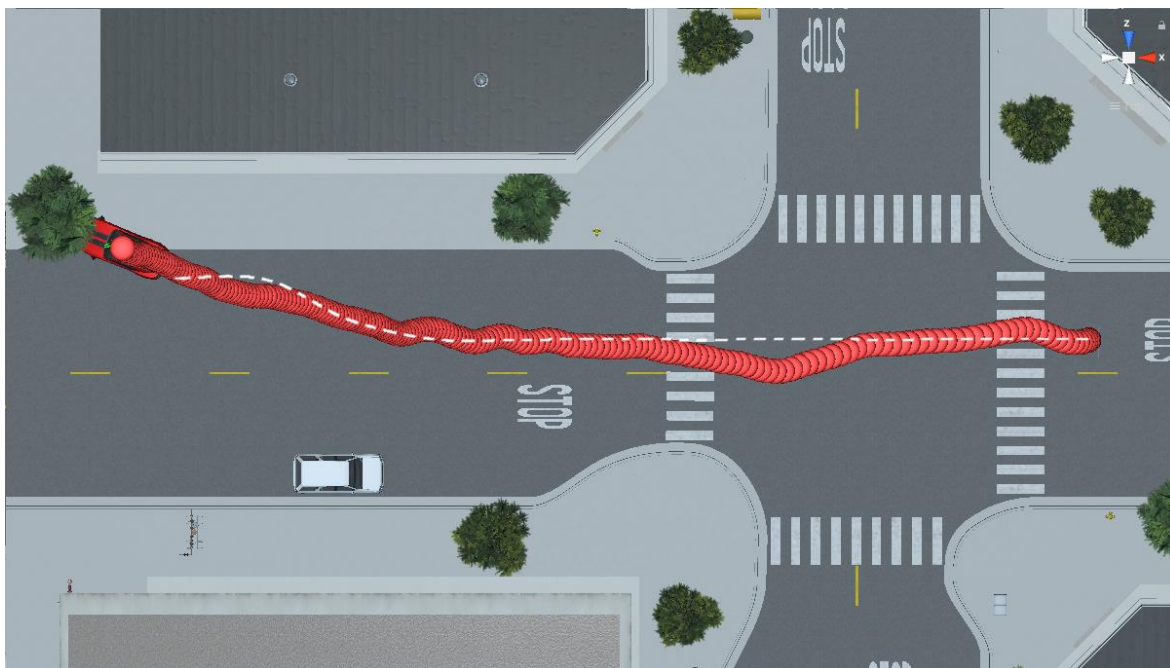
Opäť sme použili 50 jedincov. Prvý tréning iba s 10 skrytými neurónmi sme nechali bežať 48 hodín. Za tento čas prešlo 9000 generácií ale aj tak to nebolo dosť na pochopenie troch rozdielnych križovatiek. Neurónová sieť sa dokázala naučiť iba dve a tretiu sa posledných 3000 generácií už ani nepokúšala zlepšovať ako sa zasekla v lokálnom minime dvoch križovatiek čo relatívne dobre zvládala.

Pre verziu s 20 skrytými neurónmi sa simulácia spomalila približne o 50%. Celkový čas trénovania bol 72 hodín. Za tento čas sme taktiež prešli 9000 generácií s ukladáním populácie a šampióna každých 1000 generácií.

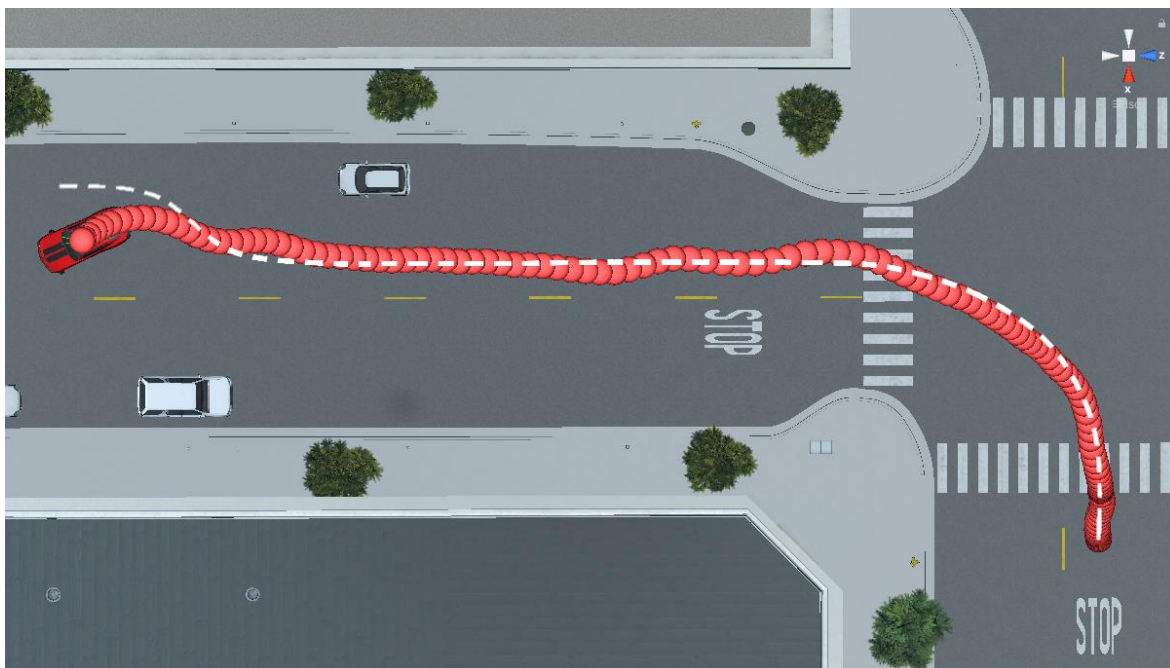
Opäť sa začalo ukazovať zneužívanie maximálneho času na jednu generáciu. Neurónová sieť opäť zneužívala časové ukončenie prechodu križovatkami pre zabezpečenie perfektnej pozície bez brzdenia. Problémom však bolo že nie na všetkých križovatkách sa to objavilo súčasne. Obzvlášť pri križovatke 3 sa tento problém ukázal najrýchlejšie. Preto pre záverečné testovanie použijeme šampióna z generácie 4000.



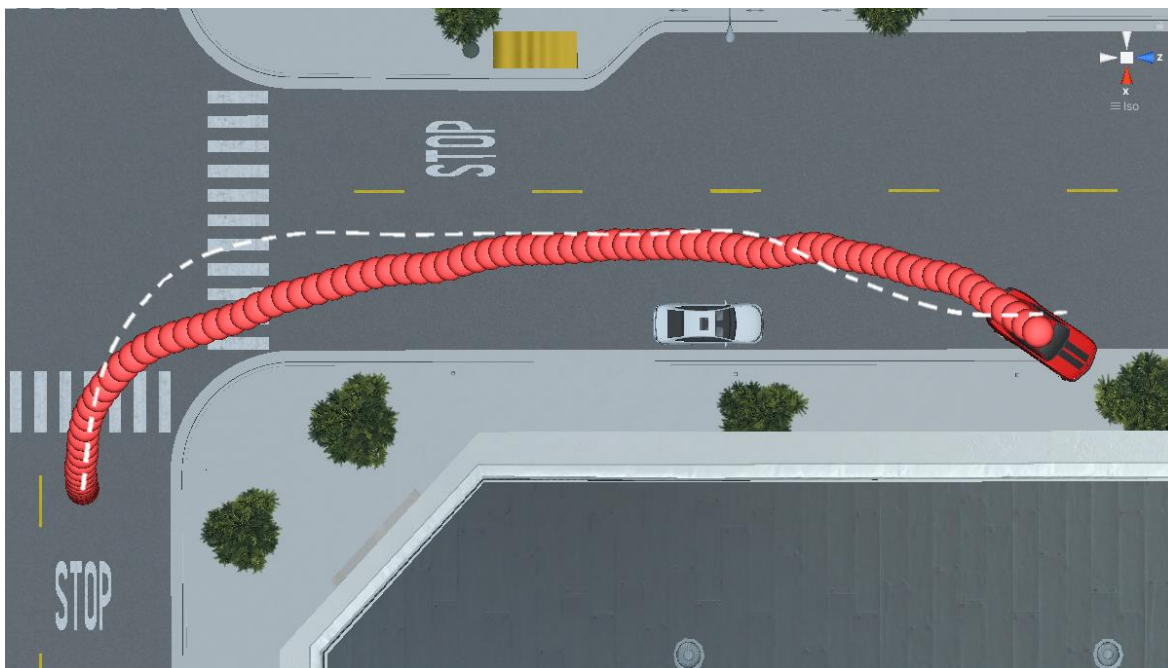
Obrázok 21: Fit tréningu 3 križovatiek



Obrázok 22: DT3-4000 v križovatke 1



Obrázok 23: DT3-4000 v križovatke 2



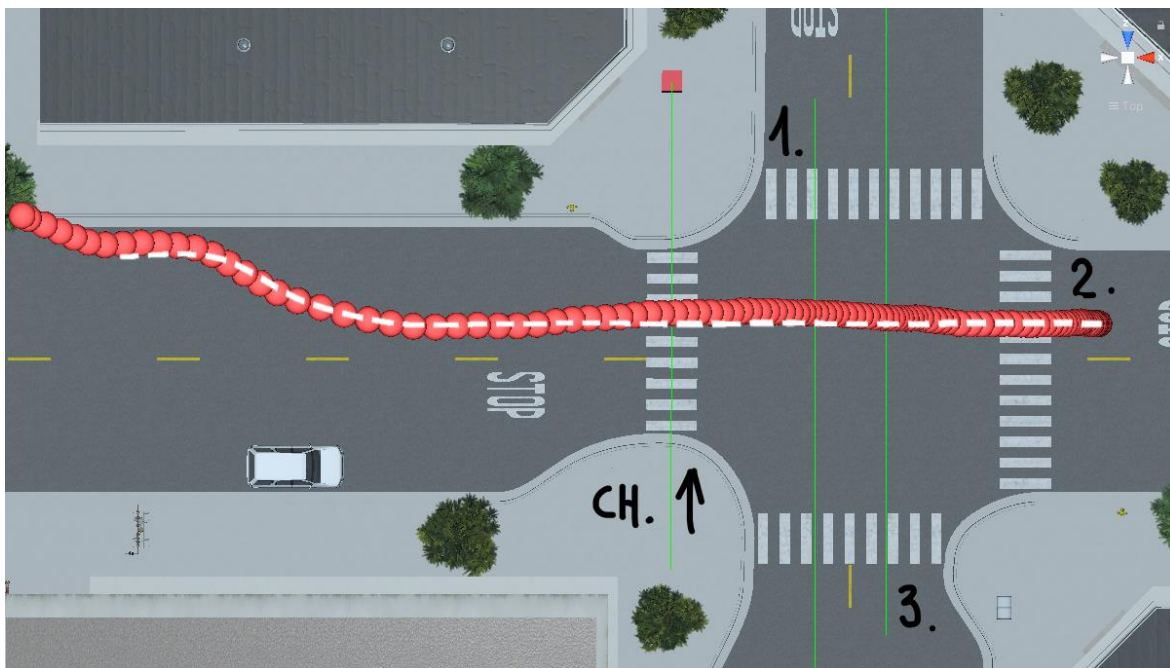
Obrázok 24: DT3-4000 v križovatke 3

5.3 Porovnanie

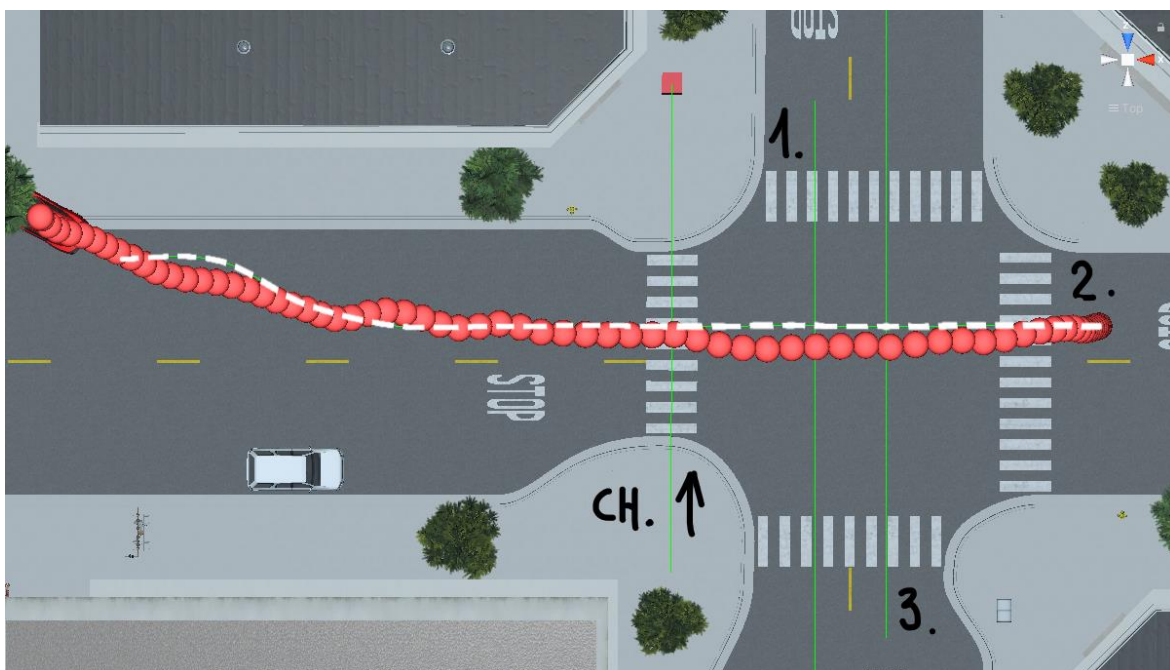
Pre porovnanie schopnosti zovšeobecňovania sme vytvorili tri scenáre rozdielne od tých kde sa autonómne vozidlá trénovali. Tu sme po prvý raz zaviedli do simulácie chodcov. Ich podoba bola iba box so stranou 1 meter, nastavený tak aby ho radar dokázal zachytiť. Podobne ako s vozidlami sme použili skript Kill Wall, trajektóriu chodca sme zadefinovali ručne pomocou path tool aby prešiel z jedného konca chodníka na druhý. Rýchlosť chodca sme nastavili na 3 m/s (10,8 km/h), čo sa môže považovať za ľahký beh. Ich oneskorenie bolo nastavené aby ich autonómne vozidlá ešte stretli na ceste.

Test 1 sme založili na križovatke 1 - Obrázok 16, zmenili sme kam smerovalo vozidlo 1. Miesto zabočenia doľava, blízko k autonómnemu vozidlu, pokračovalo priamo. Čas oneskorenia vozidla jedna sa nezmenil pretože opäť malo prejsť križovatkou ako prvé.

Vo výsledku obe vozidlá spomalili svoj prechod križovatkou pred chodcom. Tak sa im podarilo vyhnúť plánovanej zrážke. Ich trajektória bola dostatočne blízka želanej dráhe aby nezasiahli do premávky iných vozidiel a správne dali prednosť. Problémom bolo brzdenie kde v prípade Obrázok 25 môžeme vidieť že vozidlo narazilo do stromu.

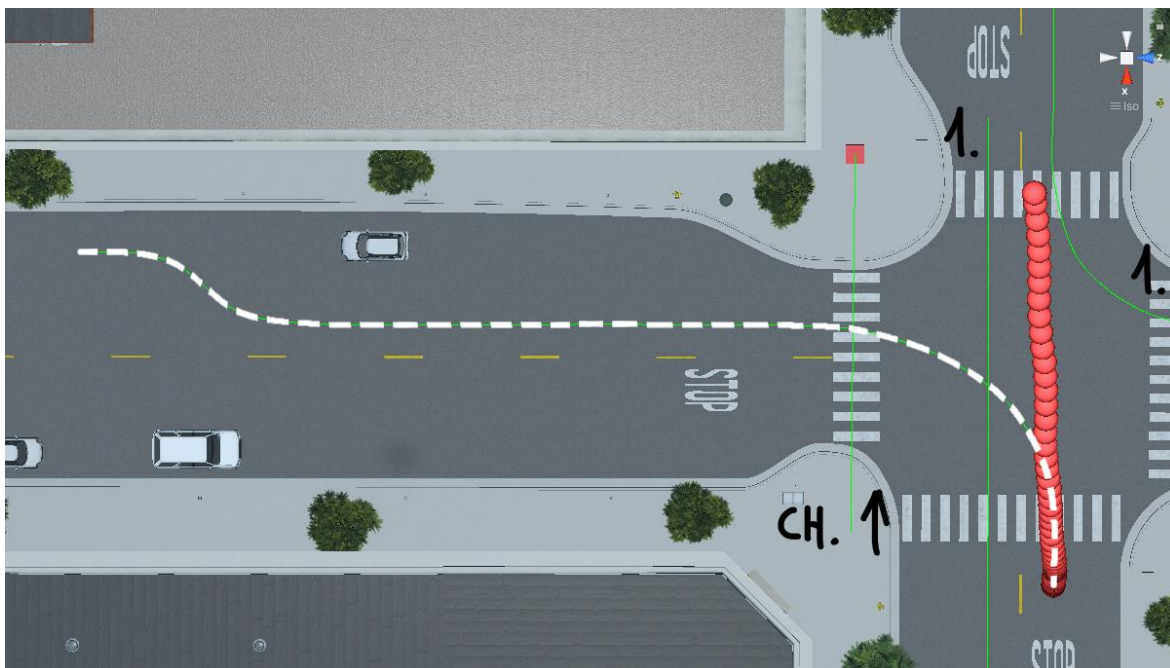


Obrázok 25: DT1-2500 v teste 1

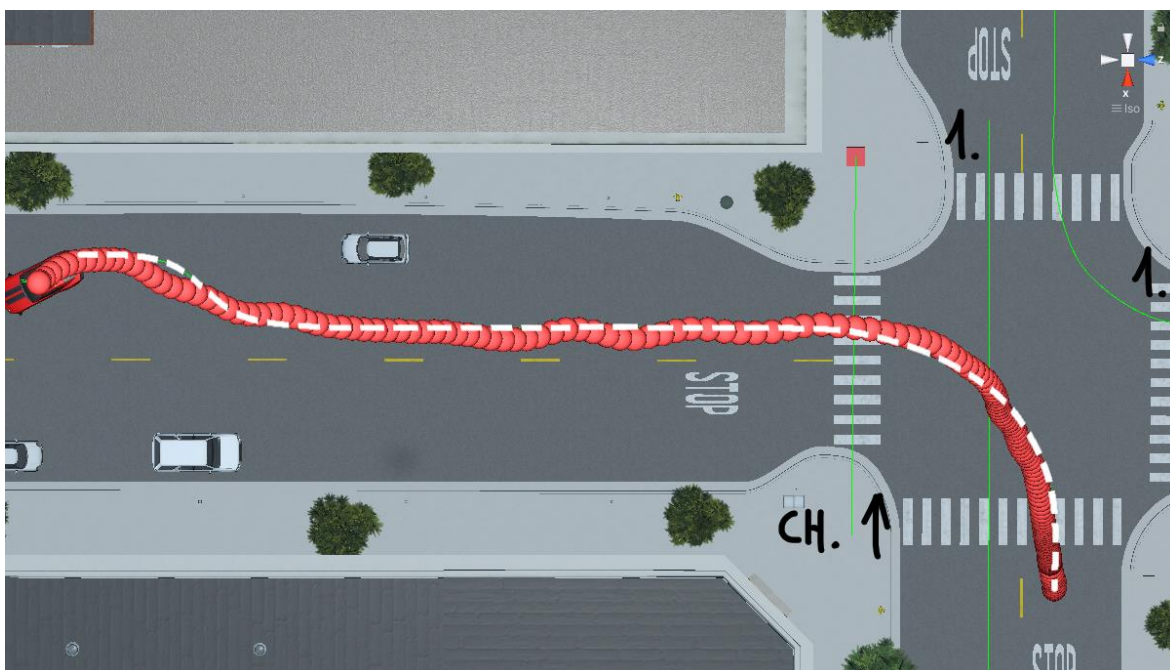


Obrázok 26: DT3-4000 v teste 1

Test 2 sme založili na križovatke 2 - Obrázok 19, zmenili sme kam smerovalo vozidlo 2. Miesto zabočenia doľava zabočilo doprava tak že nikoho neblokovalo a mohlo sa teda pohnúť súčasne s vozidlom 1. Autonómne vozidlo sa teda mohlo pohnúť ako druhé v poradí. Opäť do dráhy sme pripravili chodca.



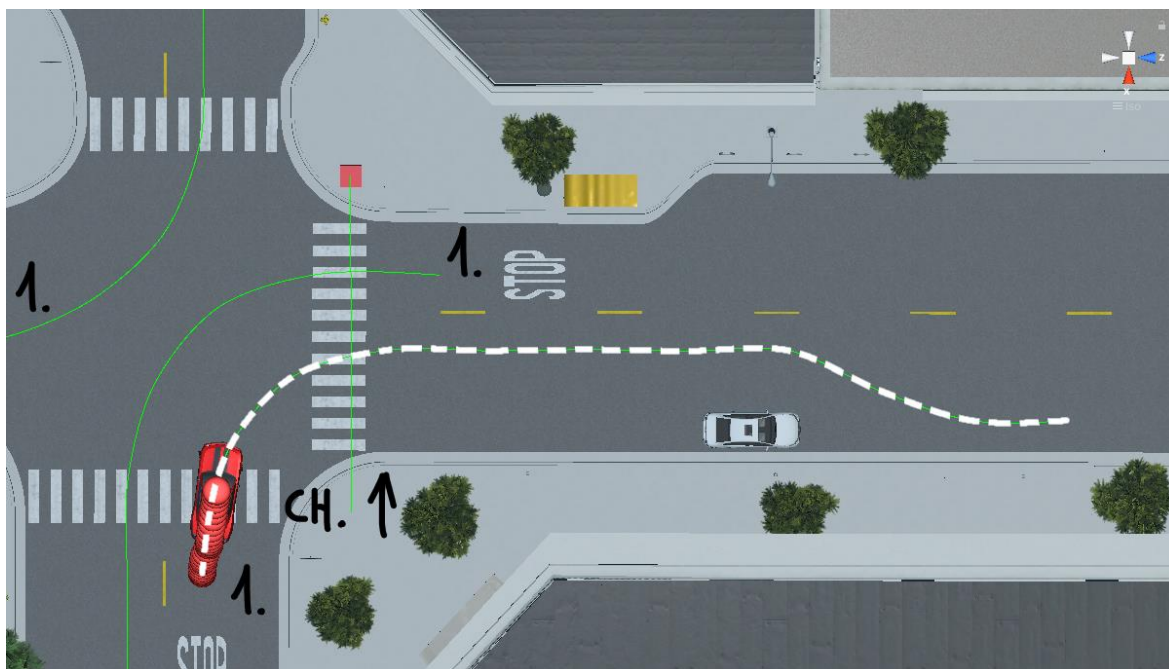
Obrázok 27: DT1-2500 v teste 2



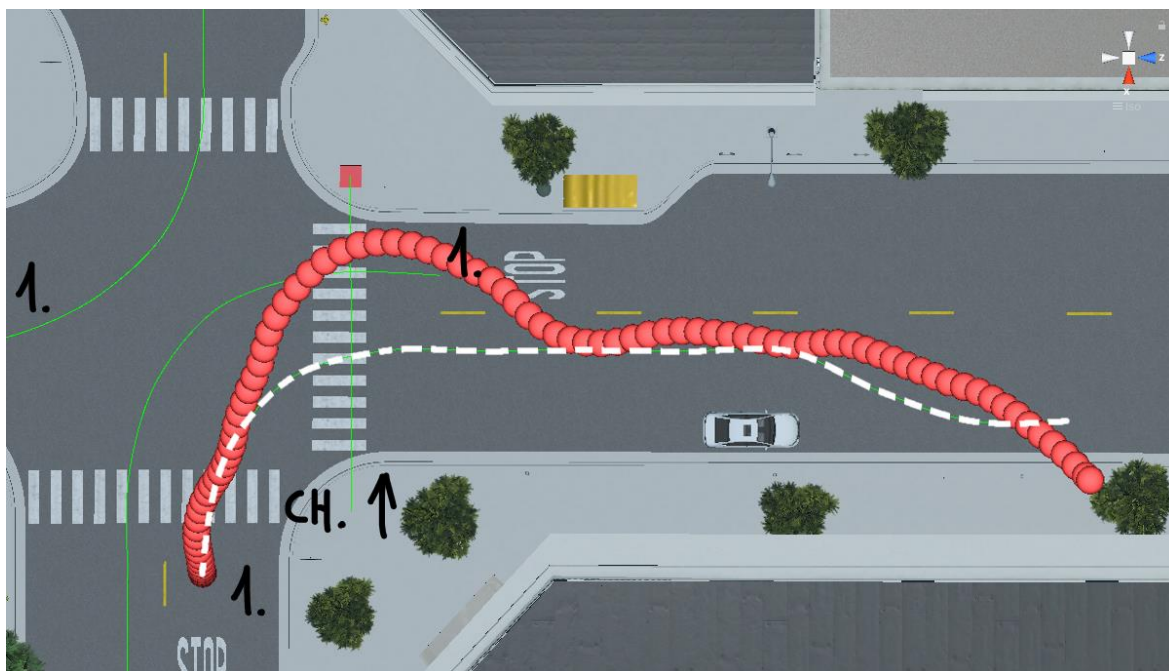
Obrázok 28: DT3-4000 v teste 2

Na Obrázok 27 môžeme vidieť že náš šampión prvého scenára nedokáže zabočiť doľava. Na koľko bol natrénovaný iba na trase dopredu, toto sa očakávalo. V prípade Obrázok 28 môžeme vidieť správanie kde vozidlo spomalilo aby sa vyhlo chodcovi. Následne ako v prípade tréningu na križovatke 2 zaparkovalo do cesty. Nakoľko však sledovalo trajektóriu a netrafilo chodca, berieme to ako prijateľný výsledok.

Test 3 sme založili na križovatke 3 - Obrázok 20, nezmenili sme k tomu žiadne vozidlá. V predchádzajúcich prípadoch by sa dalo povedať že chodci boli v práve. Vtkročili do križovatky keď sa zatiaľ žiadne vozidlo nepokúšalo prejsť ich cestou. Chodec v tomto scenári sa rozhodol že jednoducho ide, aj keď sprava sa k nemu blíži vozidlo.



Obrázok 29: DT1-2500 v teste 3



Obrázok 30: DT3-4000 v teste 3

V prípade šampióna prvého scenára sme nemali veľké očakávania ohľadom schopnosti odbočiť doprava. Obrázok 29 nám ukázal že naše očakávania boli správne. Vozidlo sa ani len nepokúsilo pohnúť, jedno ako dlho sme čakali. V okamihu ako chodec prešiel za polovicu chodníka, autonómne vozidlo jednoducho zastavilo. Vychádza to pravdepodobne z faktu že pri scenári kde sa trénovalo, z tej pozície vychádza vozidlo čo má prednosť.

V prípade šampióna troch križovatiek sme očakávali že ani tento scenár nebude problém. Ako sa však vozidlo a chodec pohli dopredu, vozidlo sa ho neustále pokúšalo obehnúť z ľavej strany miesto toho aby mu dal prednosť. Obrázok 30 ukazuje že tento pokus sa mu nakoniec podaril, i keď až v protismere. Odkiaľ sa však rýchlo vrátil späť do svojho pruhu. Chodec zostal nezranený, autonómne vozidlo ale na konci zaparkovalo do stromu.

Záver

Cieľom práce bolo simulovať autonómne vozidlo natrénované pomocou neuroevolučného algoritmu. V prvých dvoch kapitolách práce sme rozobrali teóriu neurónových sietí a neuroevolúcie so zameraním na NEAT algorimus. Toto sa pre nás stalo základom čo nás viedol k nášmu finálnemu cieľu.

S poznaním akú štruktúru neurónovej siete chceme použiť sme si vybrali prostredie, v ktorom chceme pracovať. Unity sa ukázalo ako dobrá voľba z komerčne dostupných možností. Napriek širokej podpore, existovalo iba malé množstvo projektov čo sa v unity pokúšali sprevádzkovať neuroevolúciu. Práca teda mala veľa čo objavovať, i keď sme stavali na našich predchodcoch. Balík UnitySharpNEAT pomohol k uskutočneniu práce, ale jeho obmedzenia si vyžadovali množstvo času na odstránenie. Je otázne či by bolo jednoduchšie celý NEAT naprogramovať do unity osobne, v čase odhalenia problémov balíka UnitySharpNEAT ale už na také myšlienky bolo neskoro.

To však nebolo jediným problémom čo sa v unity objavil. Hardware limitácie spôsobili pomalé tréningovanie nakoľko nami použitá verzia unity vedela používať iba jedno jadro procesora na svoje fungovanie. Vychádzalo to zo spôsobu ako sme projekt postavali od prvého dňa a opäť v čase keď sa problémy objavili bolo neskoro to zmeniť. Celé tréningovanie sa teda odohrávalo po dobu dní a požitý hardvér využíval iba osminu svojho potenciálu.

Pre potreby simulácie autonómnych vozidiel je nutné mať aj mapu a scenáre. ASAM OpenDrive je štandard pre túto oblasť. Univerzitná licencia od Mathworks nám umožnila použiť ich prostredie RoadRunner čo túto prácu zachránilo. OpenDrive sa totiž pýši tým že je to bezplatný štandard, prostredie na jeho používania ale nie sú. RoadRunner však ukázal prečo na ňom existuje cenová nálepka, toto prostredie malo podporu na všetky oblasti štandardu. Návrhu komplexných máp, návrh a simulovanie scenárov malo v sebe všetko potrebné. Jediné čo sa mu dá vytknúť je neintuitívne užívateľské rozhranie, existuje ale k nemu dostatok video návodov že to na konci práce nebol problém.

Pre scenáre na testovanie našich autonómnych vozidiel sme sa inšpirovali čímisi čo musí zvládnuť každý začínajúci vodič, test križovatiek. Vybrané boli z rady otázok pre autoškolu ohľadom problému, ktoré vozidlo má križovatkou prejsť prvé. Každý scenár bol podrobne rozobraný pri svojom uvedení. Nebyť pomalej rýchlosti tréningovania, použili by sme ich viac. Výsledky na testovacích scenároch dopadli lepšie ako sa predpokladalo. Napriek tomu že pri tréningu sa autonómne vozidlá nestretli s chodcami, nemali sme straty na životoch.

Šampión DT1-2500 podľa očakávaní nebol schopný odbočovať doľava a doprava. V teste 1 ale správne dal prednosť i keď vozidlo číslo jedna zmenilo svoju trasu a aj napriek chodcovi v ceste prešiel križovatkou pred vozidlom číslo tri.

Šampión DT3-4000 na druhej strane ukázal že križovatka pre neho nie je problém. S dvadsiatimi neurónmi v skrytej vrstve a trénovaní na troch križovatkách mu dalo dostatočnú robustnosť vyriešiť tri úplne odlišné scenáre. Riešenie v teste 3 nebolo najelegantnejšie avšak na jeho obranu, nevidel pred sebou v ceste žiadne vozidlo a hneď po obídení chodca sa zaradil späť do svojho pruhu.

V tejto práci sme použili limitovaný počet senzorov na vozidle pre riešenie problému tak komplexného že aj ľudia naň musia byť testovaní v oficiálnych zariadeniach. Neurónová sieť musela riadiť natočenie volantu, plynový pedál a brzdový pedál iba s jednou skrytou vrstvou. Bez použitia senzorov ako lidar či kamera sa autonómne vozidlo dokázalo naučiť dať prednosť v jazde.

Odporúčanie pre tých čo by chceli v budúcnosti pracovať s unity je vytvoriť si vlastný evolučný algoritmus a použiť nový balík unity DOTS pre vytvorenie projektu. Tento balík predstavuje budúcnosť kam unity smeruje a značne zvyšuje efektivitu prostredia.

Literatúra

- [1] ANDERSON, D. – McNIELL, G. 1992: Artificial neural networks technology, DACS State-of-the-Art Report, ELIN: A011, Rome Laboratory, RL/C3C Griffiss AFB, NY 13441-5700.
- [2] SEKAJ, I. 2005: Evolučné výpočty a ich využitie v praxi. Bratislava, IRIS, 2005. ISBN: 80-89018-87-4.
- [3] Stanley K. O. – Miikkulainen R. 2002: "*Evolving Neural Networks through Augmenting Topologies*," in *Evolutionary Computation*, vol. 10, no. 2, pp. 99-127, doi: 10.1162/106365602320169811.

Použité súčasti

Unity 2020.3.40f1	https://unity.com/releases/editor/qa/lts-releases
UnitySharpNEAT	https://github.com/flo-wolf/UnitySharpNEAT
PathTool	https://github.com/SebLague/Path-Creator
ESMini	https://github.com/esmini/esmini
RoadRunner	https://www.mathworks.com/products/roadrunner.html