SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKI TEHNOLOGIJA

Diplomski studij

**Razvoj autonomne vožnje pomoću evolucijskog algoritma**

Projektni zadatak iz kolegija

Raspoznavanje uzoraka i strojno učenje

Stefan Uglješić, DRD

Osijek, 2023

Sadržaj

[1. Uvod 2](#_Toc138602362)

[1.1. Evolucijski algoritam 2](#_Toc138602363)

[1.2. Zadatak 3](#_Toc138602364)

[2. Rješenje problema 4](#_Toc138602365)

[2.1. Treniranje neuronske mreže 6](#_Toc138602366)

[2.1.1. Pokretanje učenja neuronske mreže 12](#_Toc138602367)

[2.2. Testiranje modela 13](#_Toc138602368)

[3. Korisničko sučelje 14](#_Toc138602369)

[3.1. Početni ekran 14](#_Toc138602370)

[4. Staze 17](#_Toc138602371)

[5. Rezultati 18](#_Toc138602372)

[Literatura 19](#_Toc138602373)

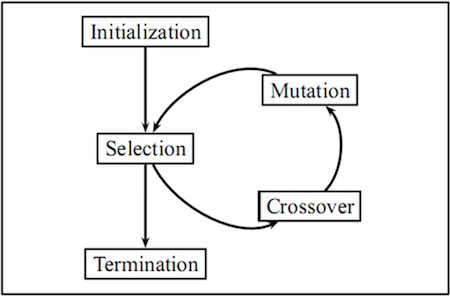
# Uvod

Strojno učenje je rastuća tehnologija koja računalima omogućuje automatsko učenje iz prošlih podataka. Strojno učenje koristi različite algoritme za izradu matematičkih modela i predviđanja pomoću povijesnih podataka ili informacija. Trenutno se koristi za različite zadatke kao što su prepoznavanje slika, prepoznavanje govora, filtriranje e-pošte, automatsko označavanje na Facebooku, sustav preporuka i još mnogo toga.

## Evolucijski algoritam

Evolucijski algoritmi heuristički su pristup rješavanju problema koji se ne mogu lako riješiti u polinomijalnom vremenu, kao što su klasični NP-teški problemi i bilo što drugo za što bi bilo potrebno predugo da se iscrpno obradi. Kada se koriste samostalno, obično se primjenjuju na kombinatorne probleme; međutim, genetski algoritmi se često koriste u tandemu s drugim metodama, djelujući kao brzi način da se pronađe donekle optimalno početno mjesto za rad drugog algoritma.

Premisa evolucijskog algoritma (koji će se dalje nazivati EA) prilično je jednostavna s obzirom da ste upoznati s procesom prirodne selekcije. EA sadrži četiri opća koraka: inicijalizaciju, odabir, genetske operatore i prekid. Svaki od ovih koraka odgovara, ugrubo, određenom aspektu prirodne selekcije i pruža jednostavne načine za modularizaciju implementacija ove kategorije algoritama.



**Slika 1.1.** Grafički prikaz evolucijskog algoritma

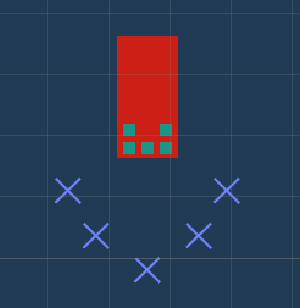
## Zadatak

Zadatak ovog seminarskog rada je izučiti neuronsku mrežu tako da upravljanjem automobila uspije završiti utrku bez da automobil dodirne „ogradu“ ili bilo kakvu prepreku. Treniranje je potrebno napraviti na jednoj vrsti staze a zatim testirati na drugoj mapi. Za izradu ovog projekta koristi se Unity Engine i C# jezik.

# Rješenje problema

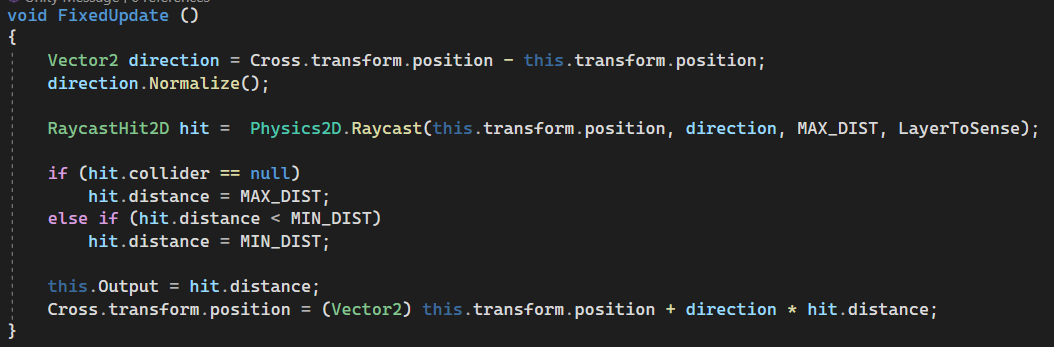
Osnovni dio simulacije je upravljanje automobilom koristeći neuronsku mrežu. „CarController.cs“ i „CarMovement.cs“. Skripte omogućuju neuronskoj mreži da pomoću varijabli: sila okretanja i brizna kretanja (a to je ujedno i izlaz neuronske mreže) upravljanje autom.

Automobil (**Slika 2.1.**) ima pet prednjih senzora koji mjere udaljenost do prepreka u određenom smjeru. Očitanja ovih senzora služe kao ulaz neuronske mreže automobila. Svaki senzor pokazuje u drugom smjeru, pokrivajući prednji dio od približno 90 stupnjeva. Maksimalni domet senzora je 10 jedinica a minimalni 0.1.



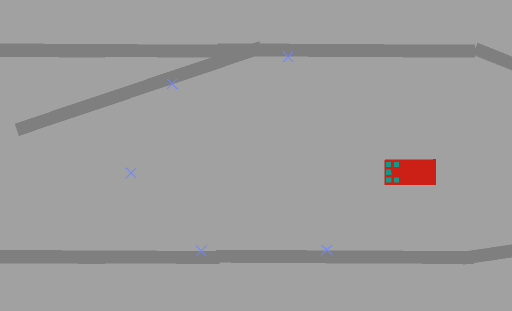
**Slika 2.1.** Izgled automobila

Svaki automobil je napravljen kao „prefab“ u Unity editoru i „parent“ (roditelj) je pet objekata koji kao komponentu sadrže skriptu „Sensor“. Skripta ima za cilj mjerenje udaljenosti između sebe i smjera u koji je uperena, a potom taj podatak šalje neuronskoj mreži. To se postiže pomoću ovog dijela koda (**Slika 2.2**).



**Slika 2.2.** Dio „Sensor“ koda odgovoran za mjerenje duljine

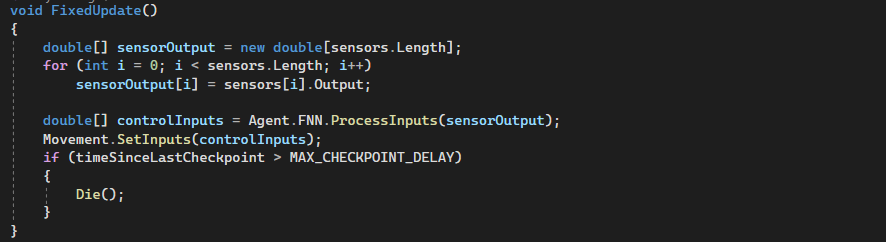
Iz **Slike 2.2.** vidimo korištenje „FixedUpdate“ metode, ona je Unity metoda koja se poziva svaki frame i pogodna je za fizičke proračune koje se u njoj pozivaju. Physics2D.Raycast je metoda koja se koristi za „pucanje“ zrake u određenom smjeru pomoću koje možemo odrediti udaljenost ili provjeru.



**Slika 2.3.** Reagiranje senzora automobila na okolinu

Iz **Slika 2.3** možemo vidjeti kako svaki senzor mjeri udaljenost između automobila i prepreke (u ovom slučaju ograde). Sa slike je vidljivo da na krajevima senzora postoji „X“, on nema nikakve svrhe za neuronsku mrežu ali koristan je za vizualizaciju rada i provjeru.

Osnovna svrha „CarController“ skripte je upravljati ponašanjem automobila na temelju ulaza dobivenih od genetskog agenta (Agent). Agent sadrži neuronsku mrežu (NeuralNetwork) koja se koristi za izračunavanje kontrolnih ulaza za kretanje automobila.



**Slika 2.4.** FixedUpdate funkcija u kojoj se upravlja automobilom

Ova metoda (**Slika 2.4**) koristi se za dobivanje ulaznih podataka od agenta te postavljanje kontrolnih ulaza za kretanje automobila. Također provjerava vrijeme proteklo od posljednjeg prolaska kroz kontrolnu točku i, ako je prekoračeno određeno vrijeme, automobil se proglašava "mrtvim".



**Slika 2.5.** FixedUpdate funkcija u kojoj se upravlja automobilom

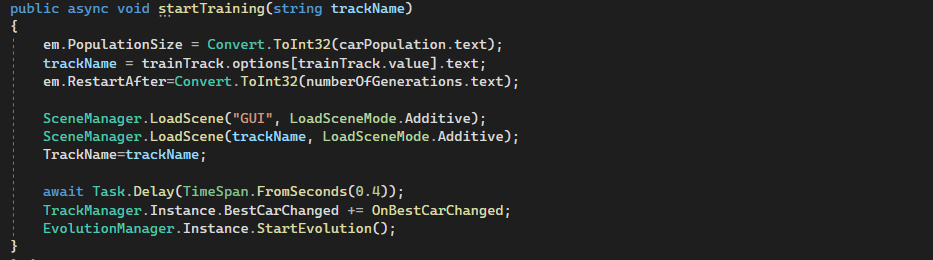
Ova privatna metoda (**Slika 2.5**) se koristi za prekid kretanja automobila, deaktiviranje senzora i "ubijanje" agenta.

## Treniranje neuronske mreže

Težine neuronske mreže treniraju se korištenjem evolucijskog algoritma poznatog kao genetski algoritam.

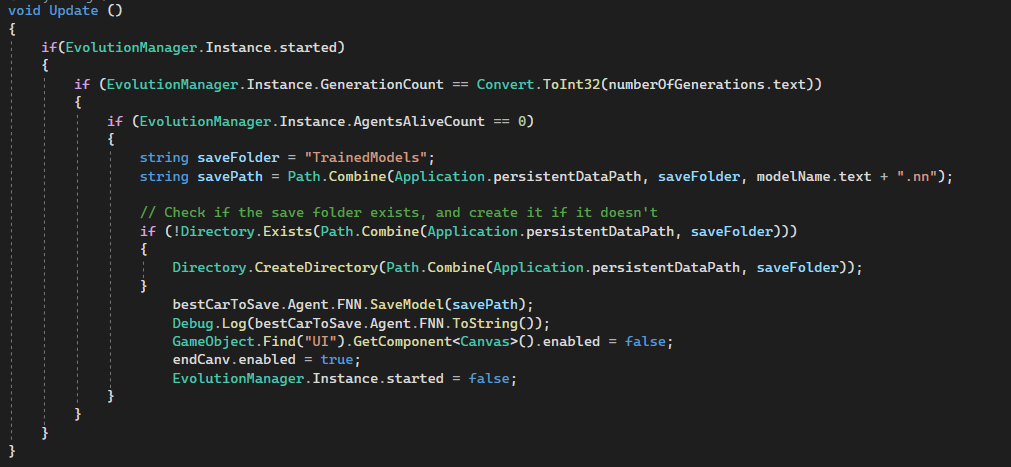
Postoje dvije bitne skripte a to su: „GameStateManager“ i „EvolutionManager“ koje su međusobno povezane kako bi omogućile treniranje i teniranje neuronske mreže. GameStateManager upravlja cjelokupnom simulacijom, uključujući interakciju s korisničkim sučeljem i manipulaciju kamerom.

EvolutionManager je odgovoran za konfiguriranje i pokretanje genetskog algoritma te evaluaciju agenata. Ove dvije klase rade zajedno kako bi omogućile ciklus treniranja, evaluacije i evolucije neuronske mreže za postizanje boljih rezultata u vožnji automobila.



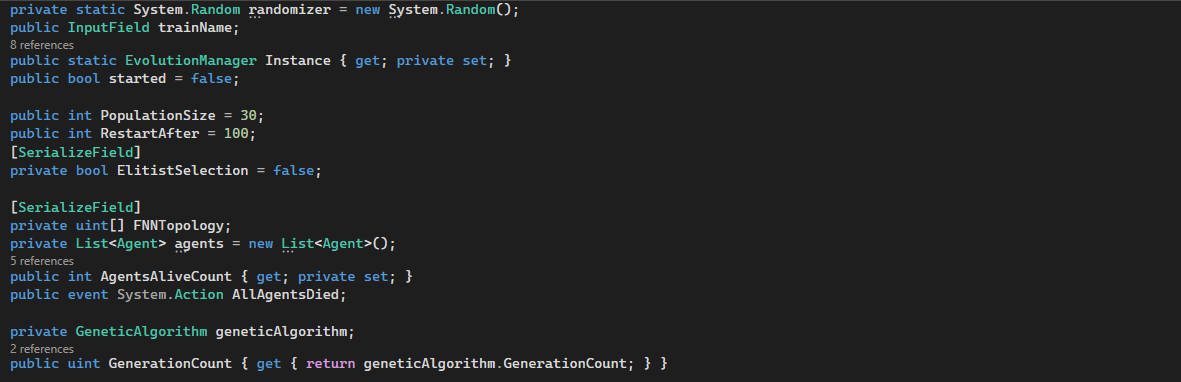
**Slika 2.6** Metoda startTraining

Metoda „startTraining“ u klasi „GameStateManager“ započinje trening neuronske mreže za vožnju automobila na odabranoj stazi. Ovaj dio skripte omogućuje pripremu i pokretanje treninga neuronske mreže na odabranoj stazi s konfiguriranim parametrima veličine populacije i broja generacija.



**Slika 2.7** Provjera završetka treniranja

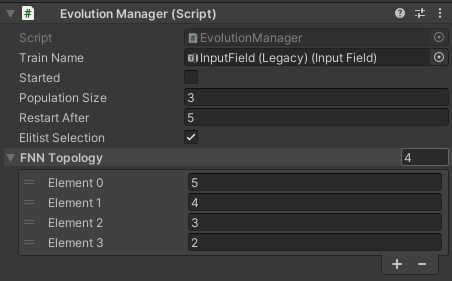
Metoda „Update“ (**Slika 2.7**) u klasi GameStateManager se izvršava svaki frame i provjerava određene uvjete kako bi završila treniranje neuronske mreže. Ovaj dio skripte provjerava je li postignut uvjet za završetak treninga (broj generacija i broj živih agenata) i ako je, sprema naučeni model neuronske mreže i mijenja prikaz korisničkog sučelja.



**Slika 2.8** Postavke EvolutionManager skripte

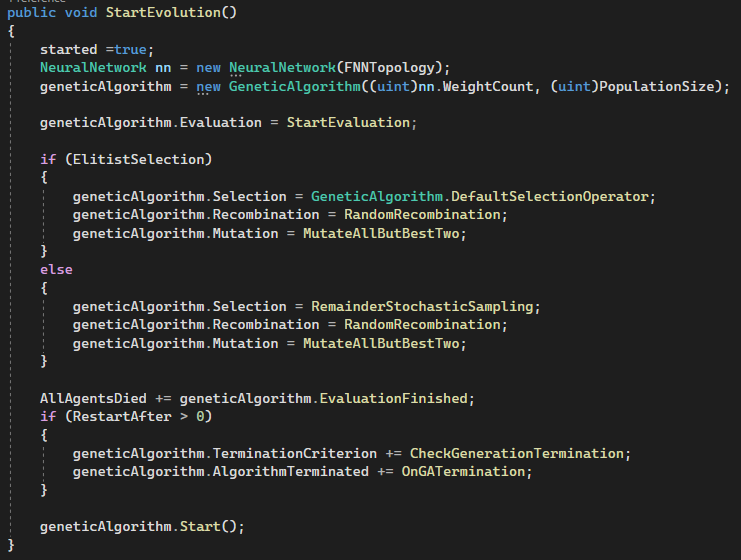
**Slika 2.8** prikazuje varijable koje se koriste za postavljanje evolution managera. Neke od tih postavki su:

* **public bool started = false;** Označava je li evolucijski proces započeo.
* **public int PopulationSize = 30;** Broj jedinki u populaciji, postavljen na 30 kao zadanu vrijednost.
* **public int RestartAfter = 100;** Nakon koliko generacija treba ponovno pokrenuti genetički algoritam. Ako je postavljeno na 0, genetički algoritam se neće ponovno pokrenuti.
* **[SerializeField] private bool ElitistSelection = false;** Označava koristi li se elitistička selekcija u genetičkom algoritmu. Ako je postavljeno na true, koristi se elitistička selekcija, inače se koristi ostatak stohastičkog uzorkovanja.
* **[SerializeField] private uint[] FNNTopology;** Topologija neuronske mreže koja se koristi za agenta u evolucijskom procesu. U ovom slučaju ova varijabla se postavlja iz Unity Editora (**Slika 2.9**)



**Slika 2.9** Prikaz „EvolutionManager“ postavki

Iz **Slike 2.9** možemo primjetiti kako izgleda neuronska mreža, dakle sadrži 5 ulaza (senzori), 2 sloja po 4 i 3 neurona i izlaz (brzina i rotacija).

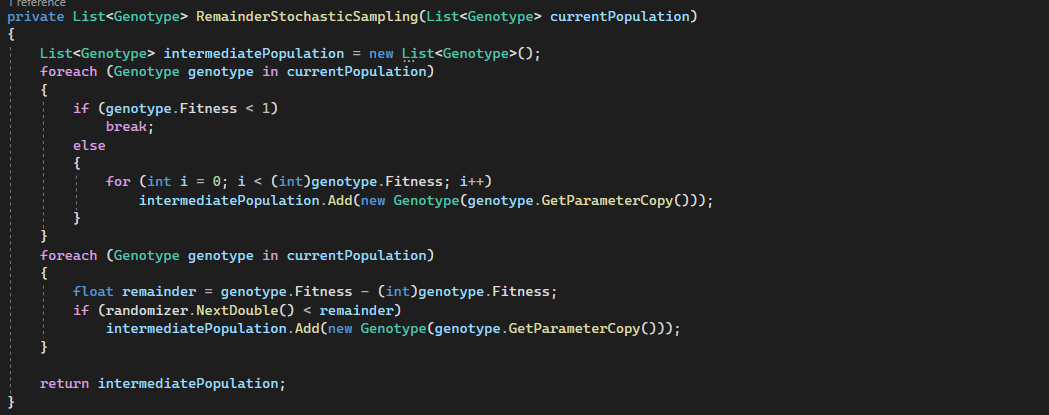


**Slika 2.10** Prikaz metode za pokretanje evolucijskog procesa

Ova metoda (**Slika 2.10**) započinje evolucijski proces inicijalizirajući neuronsku mrežu, genetički algoritam i povezujući događaje i operatore genetičkog algoritma. Nakon toga, pokreće se sam genetički algoritam.

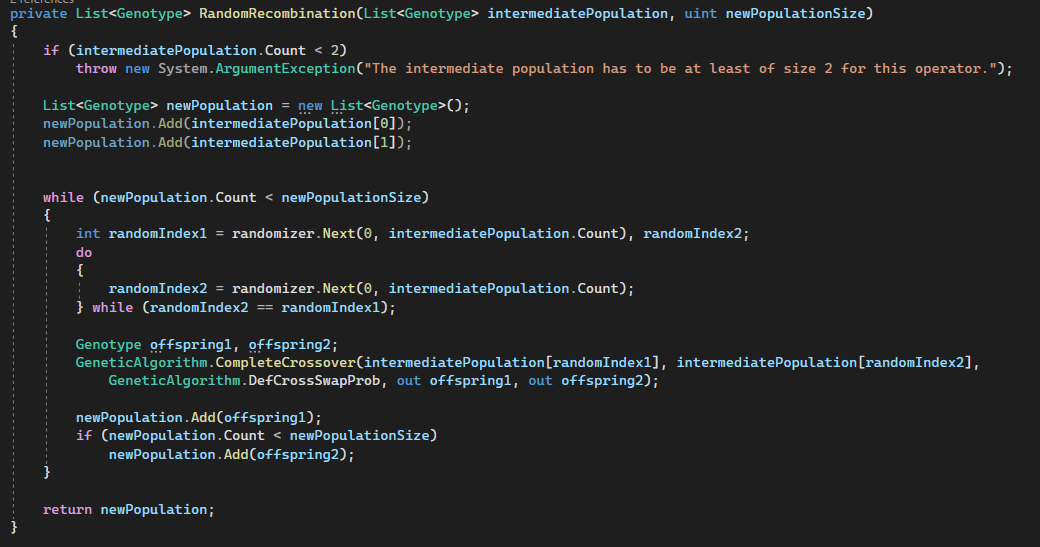
* **NeuralNetwork nn = new NeuralNetwork(FNNTopology)** Stvara novu instancu neuronske mreže NeuralNetwork s topologijom FNNTopology.
* **if (RestartAfter > 0) { ... }** Ako je postavljeno pravilo za ponovno pokretanje genetičkog algoritma (vrijednost RestartAfter veća od 0), dodjeljuje se kriterij za prekid generacije i događaj za završetak algoritma.

Klasa „GeneticAlgorithm“ (**Slika 2.10)** predstavlja implementaciju modificiranog genetičkog algoritma. DefInitParamMin, DefInitParamMax, DefCrossSwapProb, DefMutationProb, DefMutationAmount, DefMutationPercsu konstante koje definiraju zadane vrijednosti za inicijalizaciju populacije, vjerojatnost zamjene parametara tijekom križanja, vjerojatnost mutacije parametara i slično. Klasa definira razne delegatske tipove koji predstavljaju operatore genetičkog algoritma kao što su inicijalizacija populacije, evaluacija, izračunavanje prilagodljivosti, selekcija, rekombinacija i mutacija. Ove metode predstavlja implementaciju nekih operatora genetičkog algoritma koji se koriste u evolucijskom procesu u klasi GeneticAlgorithm.



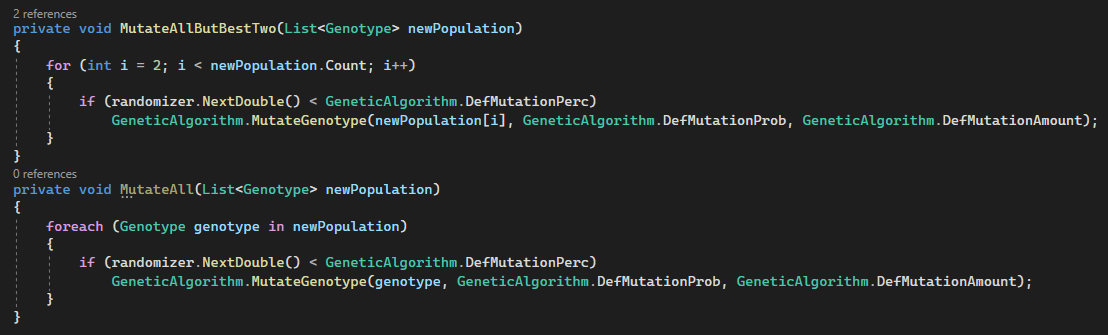
**Slika 2.11** Prikaz metode za „RemainderStochasticSampling“

Metoda (**Slika 2.11**) prima trenutnu populaciju genotipova i stvara novu populaciju primjenom operatora uzorkovanja s ostatkom. Operator uzorkovanja s ostatkom koristi vrijednost "fitness" genotipa kako bi odredio koliko puta će se taj genotip dodati u novu populaciju. Ako je "fitness" manji od 1, genotip se ne dodaje u novu populaciju. Ako je "fitness" veći ili jednak 1, genotip se dodaje "fitness" puta u novu populaciju. Zatim se za svaki genotip iz trenutne populacije provjerava ostatak vrijednosti "fitness". Ako je generirani slučajni broj manji od tog ostatka, genotip se dodaje u novu populaciju.



**Slika 2.12** Metoda „RandomRecombination“

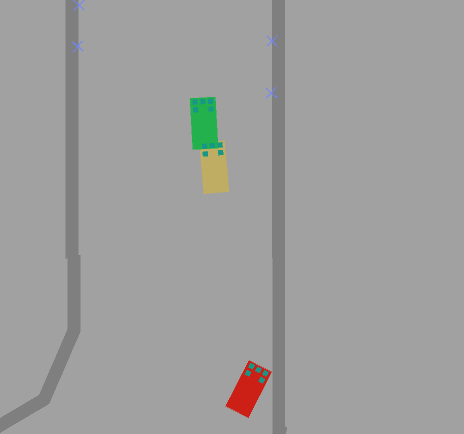
Metoda (**Slika 2.12**) primjenjuje operator rekombinacije na intermedijarnu populaciju genotipova kako bi generirala novu populaciju. Prva dva genotipa iz intermedijarne populacije dodaju se izravno u novu populaciju. Zatim se slučajno odabiru dva indeksa između 0 i veličine intermedijarne populacije kako bi se odabrala dva genotipa za rekombinaciju. Korištenjem metode CompleteCrossover, genotipi se rekombiniraju i dobiveni potomci se dodaju u novu populaciju. Postupak se ponavlja sve dok nova populacija ne dostigne zadani broj genotipova (newPopulationSize).



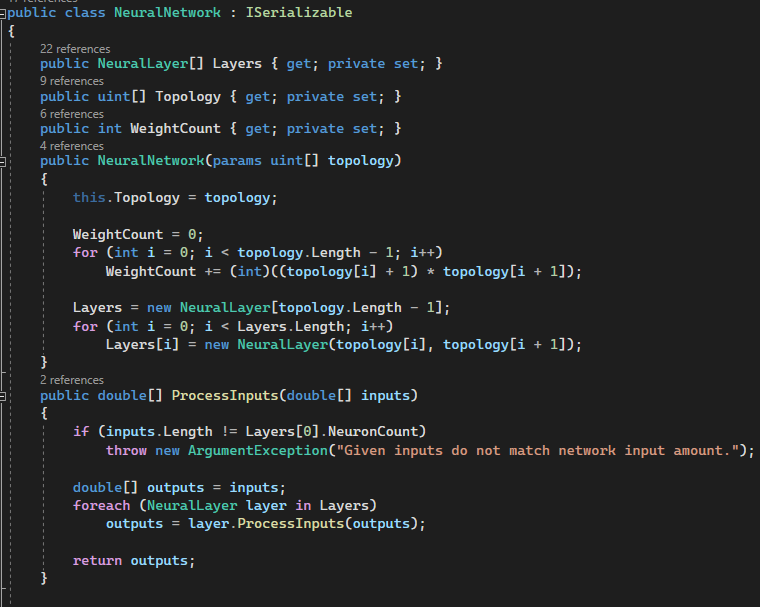
**Slika 2.13** Metode „MutateAllButBestTwo“ i „MutateAll“

Metoda „MutateAllButBestTwo“ (**Slika 2.13**) primjenjuje operator mutacije na sve genotipe u novoj populaciji, osim na prva dva genotipa. Za svaki genotip nakon prvog dva, generira se slučajni broj između 0 i 1. Ako je taj broj manji od vjerojatnosti mutacije (GeneticAlgorithm.DefMutationPerc), mutacija se primjenjuje na genotip korištenjem metode MutateGenotype.

Metoda „MutateAll“ (**Slika 2.13**) primjenjuje operator mutacije na sve genotipe u novoj populaciji. Za svaki genotip, generira se slučajni broj između 0 i 1. Ako je taj broj manji od vjerojatnosti mutacije (GeneticAlgorithm.DefMutationPerc), mutacija se primjenjuje na genotip korištenjem metode MutateGenotype.

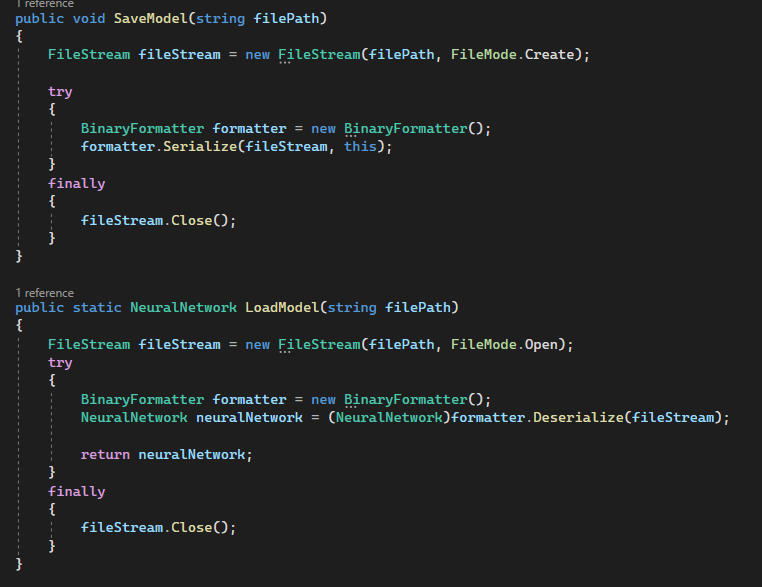


**Slika 2.14.** Prikaz selekcije 2 najbolja automobila



**Slika 2.15.** Klasa NeuralNetowork

Klasa „NeuralNetwork“ (**Slika 2.15**)predstavlja neuronsku mrežu i omogućava izvođenje obrade ulaznih podataka kroz slojeve mreže. Može se stvoriti neuronska mreža s određenom topologijom, postaviti slučajne težine, izračunati izlaze na temelju unesenih ulaza te spremiti i učitati model mreže (**Slika 2.16**). Također, klasa podržava duboku kopiju neuronske mreže i (de)serijalizaciju objekta pomoću binarnog formata.



**Slika 2.16.** Spremanje i Učitavanje objekta klasa NeuralNetowork

### Pokretanje učenja neuronske mreže

Prvo se pojavi N nasumično inicijaliziranih automobila. Najbolji automobili se tada odabiru kako bi se međusobno rekombinirali, stvarajući nove automobile "potomke". Ti automobili potomci zatim formiraju novu populaciju od N automobila i također su malo mutirani kako bi se u populaciju unijelo nešto više raznolikosti.

Novostvorena populacija automobila počinje proces ponovnog testiranja nakon čega se evaluacija, selekcija, rekombinacija i mutacija ponovno računa. Jedan kompletan ciklus od procjene jedne populacije do procjene sljedeće naziva se generacija.Nakon određenog broja generacija model se sprema.

## Testiranje modela

Skripta „CarTestController“ (**Slika 2.17**) omogućuje testiranje treniranih modela automobila na odabranim stazama. Koristi se neuronska mreža za izračunavanje upravljačkih ulaza na temelju izlaza senzora. Kada automobil udari u zid, testiranje se zaustavlja i prikazuje se kraj testiranja. Ova skripta koristi „TestCar“ kao prefab.



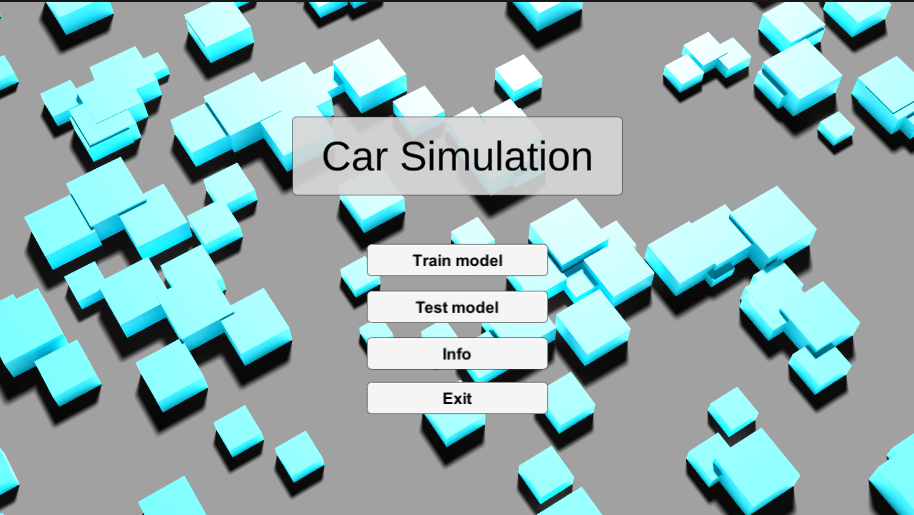
**Slika 2.17.** CarTestController skripta

* **Start()** Metoda koja se poziva pri pokretanju scene i koristi se za učitavanje treniranog modela, postavljanje teksta za prikaz naziva staze i modela te omogućavanje sučelja za testiranje modela.
* **void FixedUpdate()** Metoda koja se poziva u pravilnim intervalima i koristi se za obradu podataka senzora i upravljanje automobilom preko neuronske mreže.

# Korisničko sučelje

## Početni ekran

Grafičko sučelje za početni ekran ima četiri gumba: "Train model", "Test model", "Info" i "Exit". Gumb "Train" omogućuje pokretanje procesa treniranja. Gumb "Test" koristi se za provjeru performansi ili evaluaciju već treniranog modela. Gumb "Info" pruža dodatne informacije. Gumb "Exit" služi za izlazak iz aplikacije ili zatvaranje prozora grafičkog sučelja.



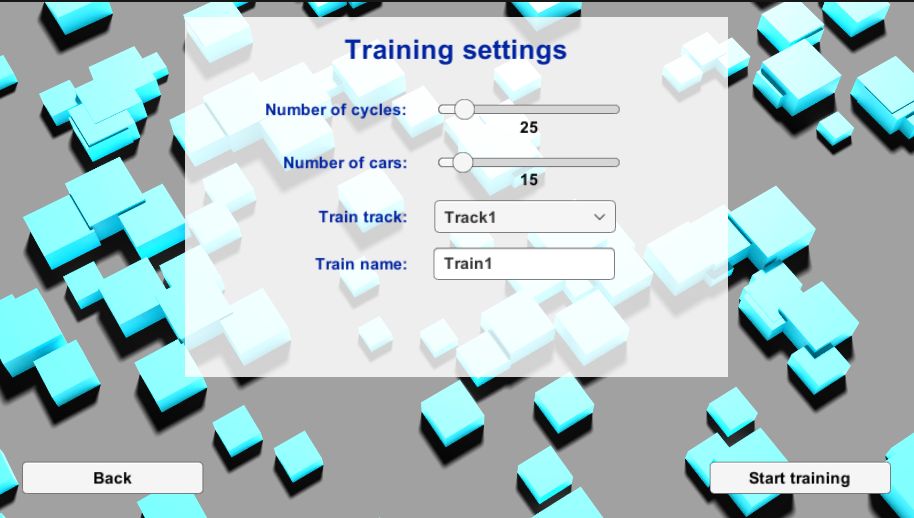
**Slika 3.1.** Početno sučelje programa

Kada se pritisne gumb "Train" u grafičkom sučelju, otvara se sučelje (**Slika 3.2**) za postavljanje parametara treninga modela. Korisnik može odabrati broj generacija, što predstavlja koliko puta će se algoritam evolucije izvršiti, odnosno koliko puta će se populacija gena prilagođavati. Također, korisnik može odabrati broj automobila koji će sudjelovati u simulaciji, što utječe na složenost treninga.

Također, sučelje omogućuje korisniku odabir staze na kojoj će se provoditi trening. To može biti predefinirana staza ili korisnik može odabrati vlastitu stazu za trening.

Korisnik također ima mogućnost unosa imena modela koji se trenira. Ovo ime će se koristiti za spremanje i identifikaciju modela nakon završetka treninga.

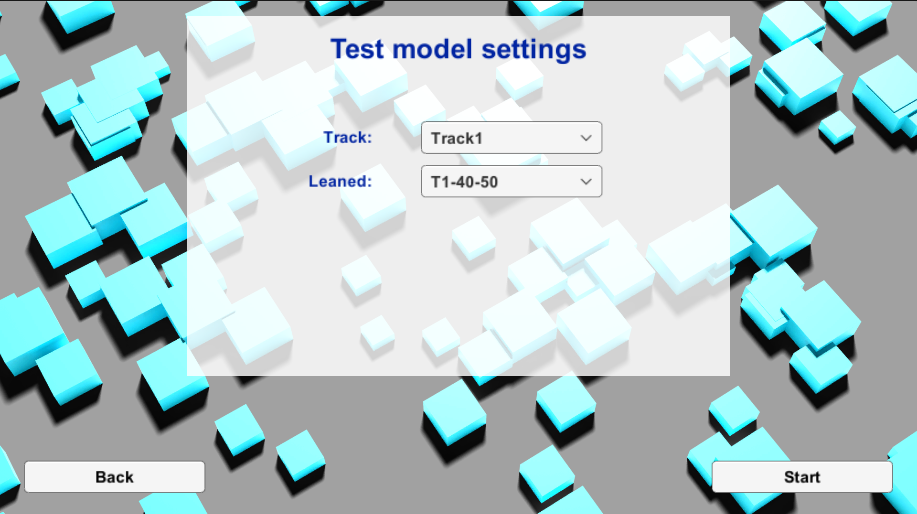
Nakon što su svi parametri postavljeni, korisnik može započeti trening pritiskom na gumb "Start" ili se može vratiti na početni ekran odabirom gumba "Back".



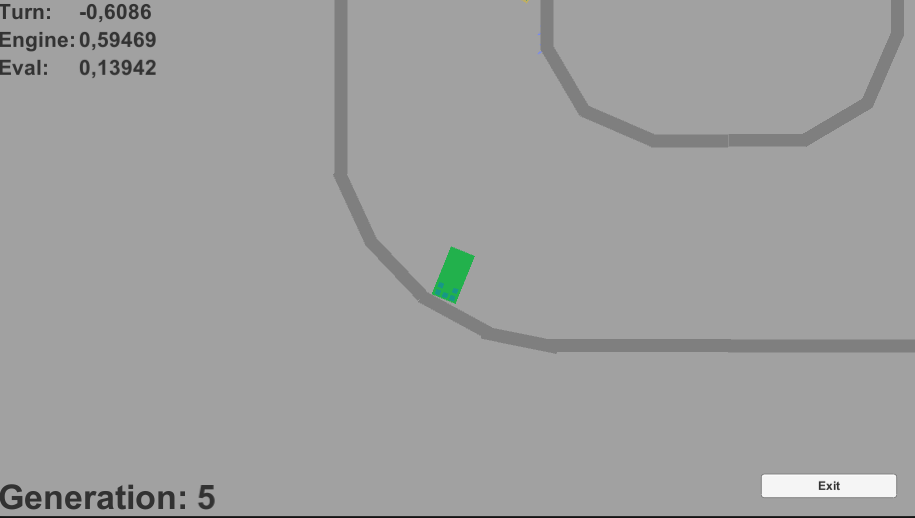
**Slika 3.2.** Sučelje postavki za treniranje modela

Kada se pritisne gumb "Test Model" (**Slika 3.1**) u grafičkom sučelju, otvara se sučelje za testiranje modela. Korisniku se pruža mogućnost odabira staze na kojoj će se model testirati. Moguće je odabrati predefiniranu stazu ili korisnik može odabrati vlastitu stazu za testiranje.

Također, korisnik može odabrati model koji je prethodno trenirao. Sučelje će prikazati dostupne modele koje je korisnik stvorio tijekom treninga. Nakon odabira staze i modela, korisnik može započeti testiranje pritiskom na gumb "Start" ili se može vratiti na početni ekran odabirom gumba "Back".



**Slika 3.3.** Sučelje postavki za testiranje modela



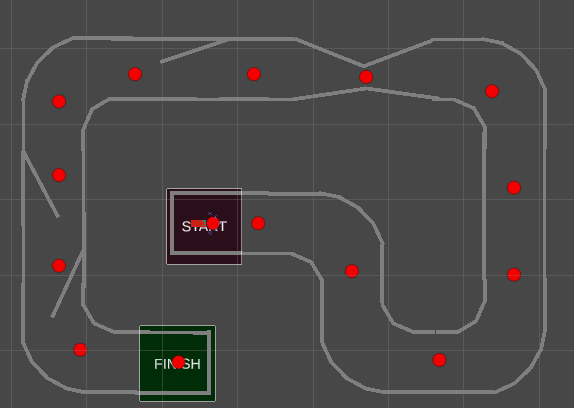
**Slika 3.4.** Sučelje postavki za testiranje modela

Korisničko sučelje (**Slika 3.3**) uvijek prikazuje podatke o trenutno najboljem automobilu. U gornjem lijevom kutu prikazan je izlaz neuronske mreže (brzina i okretanje). Odmah ispod izlaza prikazuje se vrijednost evaluacije (vrijednost evaluacije jednaka je postotku završenog tečaja). U donjem lijevom kutu prikazan je brojač generacija.

# Staze

Postoje četiri staze koje se razlikuju po težini, pri čemu je prva najlakša, a posljednja najteža. Ovdje je opis svake staze:

Staza 1 (**Slika 4.1**) je najlakša i istovremeno najšira među svim stazama. Karakterizira je prostran prostor i manje zahtjevni zavoji. Ova staza pruža veću slobodu kretanja automobila i manju vjerojatnost sudara s preprekama. Idealna je za početak i za testiranje osnovnih funkcionalnosti modela automobila.



**Slika 4.1.** Prikaz staze 1 i 2

Staza 2 (**Slika 4.1**) predstavlja srednje tešku stazu koja ima nešto uži prostor u usporedbi s prvom stazom. Na ovoj stazi postoje malo zahtjevniji zavoji. Model će morati razviti bolju preciznost u upravljanju kako bi se uspješno nosio s izazovima ove staze.

Staza 3 (**Slika 4.2.**) je izazovna i teža u usporedbi s prethodnim dvjema stazama. Karakterizira je uski prostor, oštri zavoji. Model će morati pokazati visoku razinu vještine u upravljanju kako bi se izbjegli sudari i uspješno prešla staza.

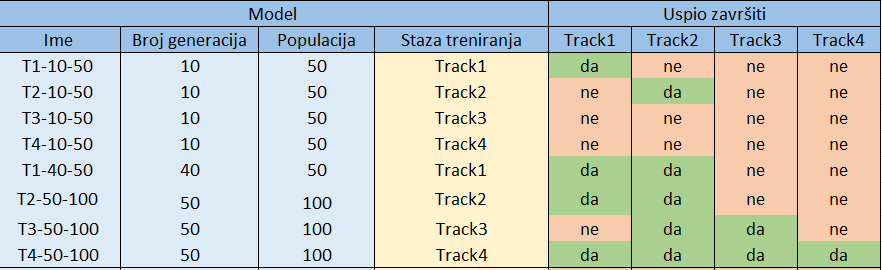


**Slika 4.1.** Prikaz staze 2 i 3

Staza 4 (**Slika 4.2.**) predstavlja najteži izazov od svih staza. Karakterizira je vrlo uski prostor, oštri zavoji. Model će morati pokazati vrhunsku vještinu i preciznost u upravljanju kako bi preživio i uspješno završio stazu.

# Rezultati

Iz **Tablice 5.1** vidimo da povećanje broja generacija može doprinijeti boljim rezultatima jer omogućuje dulje vrijeme za evoluciju populacije i pronalaženje boljih genetskih rješenja. Na primjer, modeli T1-40-50 i T4-50-100 postižu bolje rezultate u usporedbi s modelima koji su trenirani samo 10 generacija.



**Tablica 5.1.** Prikaz uspješnosti modela na određenim stazama

Kombinacija većeg broja generacija i veće populacije može dovesti do najboljih rezultata, također treniranje na težim stazama omogućuje modelu da završava lakše staze. To je vidljivo u modelu T4-50-100 koji postiže uspješnost na svim stazama. Povećanjem broja generacija i populacije omogućuje se veća eksploracija prostora rješenja i vjerojatnije je da će se pronaći optimalno rješenje.

Malo je vjerojatno da model treniran na prvoj stazi ima mogućnost za uspjeh na idućoj težoj stazi iz nekoliko razloga:

* Različite zahtjeve staza: Svaka staza ima različite karakteristike i zahtjeve koje model treba naučiti.
* Specifične informacije o stazi: Prva staza nema sve vrst prepreka sa kojima se automobil može susresti.
* Širina staze je puno veća u odnosu na druge staze i s obzirom na to automobil ima manju šansu da se sudari

S obzirom na ove faktore, model koji je treniran samo na prvoj stazi može imati malu šansu za uspjeh na idućoj težoj stazi. Isto vrijedi za model koji je treniran na drugoj stazi a testira se na trećoj itd.

# Literatura

[1] <https://unity.com/products/machine-learning-agents>

[2] <https://towardsdatascience.com/building-a-neural-network-framework-in-c-16ef56ce1fef>

[3]<https://medium.com/analytics-vidhya/introduction-to-genetic-algorithms-in-c-with-a-real-application-72c3fc42bdf9>

[4] https://visualstudiomagazine.com/articles/2022/12/12/simple-numerical-optimization.aspx