**Tehnička škola Čakovec**

**ELABORAT ZAVRŠNOG RADA**

**AI SIMULACIJA**

**Mentor: Učenik:**

**Krešimir Kočiš, prof. Toni Polanec, 4.RT**

**Čakovec, svibanj 2020.**

# Sadržaj

[1 Sadržaj 2](#_Toc36070296)

[2 Uvod 3](#_Toc36070297)

[3 Programiranje funkcionalnosti auta 4](#_Toc36070298)

[3.1 Kretanje 4](#_Toc36070299)

[4 Interakcija s okolinom 6](#_Toc36070300)

[4.1 Staza 6](#_Toc36070301)

[4.2 Vidokrug auta 8](#_Toc36070302)

[5 Programiranje inteligencije 9](#_Toc36070303)

[5.1 Perceptron 10](#_Toc36070304)

[5.2 Neuronska mreža 11](#_Toc36070305)

[5.3 Genetski algoritam (GA) 13](#_Toc36070306)

[5.3.1 Fitness 13](#_Toc36070307)

[6 Zaključak 14](#_Toc36070308)

[7 Bibliografija 15](#_Toc36070309)

[8 Popis slika 16](#_Toc36070310)

[9 Popis grafova 16](#_Toc36070311)

[10 Dodaci 17](#_Toc36070312)

# Uvod

U ovom radu ću pisati o primitivnoj vrsti AI (umjetne inteligencije). Zadatak rada je pokazati kako i na koji način je moguće implementirati algoritam strojnog učenja i genetske evolucije uz koji će subjekti, u ovom slučaju autići, kroz više generacija naučiti kako voziti stazom. Kad populacija dosegne nivo inteligencije kad u svaki generaciji dovoljno autića prođe stazom moguće ih je prebaciti na njima potpuno novu stazu i ukoliko je njihovo učenje bilo efektivno trebali bi moći iz prve proći novom stazom.

Rad će biti izrađen u dva najbitnija dijela. Programiranje autića da se mogu kretati i interakcija s prostorom. U to ulazi implementiranje osnovnih funkcionalnosti autića (ubrzavanje, kočenje, skretanje), pregled okoline iz njihove perspektive (5 senzora koji gledaju unaprijed) i na kraju njihova interakcija s stazom (dolazak do kontrolnih točaka, sudaranje s zidom, prolazak cilja).

Drugi dio je programiranje inteligencije. Svaki auto će se sam voziti. Mozak će biti implementiran sa neuronskom mrežom. Postupak svakog od autića biti će određen s onim što on vidi (senzori, tj. udaljenosti od zidova staze) i stanjem njegovih neurona u mozgu. Učenje ću implementirati genetskim algoritmom. Nakon svake generacije kroz razne algoritme određivat će se roditelji koji će svoje gene pretočiti u njihove potomke koji dalje nastavljaju spomenuto generacijsko učenje.

Umjetna inteligencija je svuda oko nas i ne možemo pobjeći od nje. U svakom mobitelu, u kamerama, u poznatim virtualnim asistentima poput Alexe ili Siri. Smatram da je umjetna inteligencija i njezino implementiranje budućnost tehnologije koja će uvelike pomoći razvoju čovječanstva.

# Programiranje funkcionalnosti auta

U ovom programu najbitniji objekt je auto. U sljedećih nekoliko ulomaka objasnit ću kako i na koji način sam implementirao potrebne funkcije auta za našu simulaciju. Podijeljeno je na 3 ulomaka: kretanje, vidokrug i stanja auta. Potrebno je objasniti ove dijelove vrlo dobro da bi kasnije lako shvatili kako ćemo u taj auto ugraditi neuronsku mrežu te kako će neuronska mreža uopće moči upravljati autom.

## Kretanje

Auto ima samo dvije najbitnije funkcije, a to su njegova brzina i skretanje. Kroz svaku iteraciju programa te dvije vrijednosti se mijenjaju. Kretanje sam implementirao koristeći troje vektora: *velocity* (brzina), *acceleration* (ubrzanje) i *location* (lokacija/pozicija). Svaku iteraciju location vektoru se pribraja velocity vektor, a velocity vektoru se pribraja acceleration vektor koji u glavnini upravlja autom. U funkciji koju sam nazvao update() se postavlja vrijednost acceleration vektora, tj. orijentacija koja upravlja skretanjem auta i vrijednost velocity vektora kojim upravljamo brzinom auta.

1. update(); //Brzina i skretanje se postavljaju.
3. vel.add(acc);
4. loc.add(vel);
5. **void** update(){
6. //…
7. // steeringSpeed[0] -> iznos skretanja, steeringSpeed[1] -> iznos brzine
9. //Utjecanje na skretanje auta. (-1 -> lijevo; 1 -> desno)
10. **float** steeringAngle = map((**float**)steeringSpeed[0], -1, 1, -0.05, 0.05);
11. acc.rotate(steeringAngle);
13. //Postavljanje limita na brzinu, tj. izravno utjecanje na brzinu auta.
14. **float** speed = map((**float**)steeringSpeed[1], -1, 1, 1, topSpeed);  //topSpeed = 4
15. vel.limit(speed);
16. //…
17. }

Neuronska mreža kao rezultat izbacuje dva broja **∈**[-1,1] koji upravljaju brzinom i skretanjem. Da bi ta dva broja auto mogao *razumjeti* oni se moraju mapirati (Funkcija koja pretvara broj iz jednog opsega u drugi: map(iznos, početak1, kraj1, početak2, kraj2)) u njemu razumljive vrijednosti. Kut skretanja je u opsegu [-0.05, 0.05] radijana što bi u stupnjevima bilo 2.86º. To znači da u jednoj iteraciji programa auto može najviše skrenuti za 2.86º. Do te vrijednosti sam došao isprobavanjem različitih vrijednosti, ako je vrijednost prevelika može se dogoditi da se auto vrti na mjestu, a ako je premala auto nije u mogućnosti izvoziti ni najblaži zavoj. Isto tako trebamo mapirati vrijednost brzine. Nju mapiramo ovisno o prije izabranom topSpeed-u (najveća brzina koju auto može postići), također odabran nakon isprobavanja različitih vrijednosti radi optimizacije učenja.

Ukratko:  
Location vektor sadrži x i y koordinatu i taj vektor koristimo da bi iscrtali auto na određenoj poziciji.

Velocity vektor koristimo da bi odredili i iscrtali smjer auta. Znači da ako vektor ima usmjerenje prema desno, auto se kroz određeno vrijeme kreće desno i crtamo tako da prednji dio auta gleda nadesno.

Acceleration vektorom upravljamo skretanjem auta.

# Interakcija s okolinom

Sam auto sam po sebi nema smisla, svaki trkaći auto mora imati i svoju stazu pa ću u sljedećim ulomcima govoriti o kreiranju spomenute staze i njezinim karakteristikama. Također ću objasniti svoju implementaciju načina na koji auto *vidi* stazu po kojoj vozi.

## Staza

Svaka staza se sastoji od 4 objekata: *starting point* (koordinate starta), *obstacles* (zidovi staze), *checkpoints* (kontrolne točke) i *finish line* (ciljna linija). Kod pokretanja programa prikažu se tri prije izrađene staze (kategorizirane po težini) i jedan upitnik s natpisom „Make your own“ kojeg ćemo se dotaknuti kasnije. Karekteristike staza se vuku iz vanjske datoteke, tj. svaka staza ima svoju mapu i svaki element ima svoju tekstualnu datoteku iz koje program iščitava i crta stazu. Svaki redak određenog objekta predstavlja jedan element. To znači da ako datoteka *obstacles.txt* ukupno ima 20 redaka ta staza ima 20 zidova.

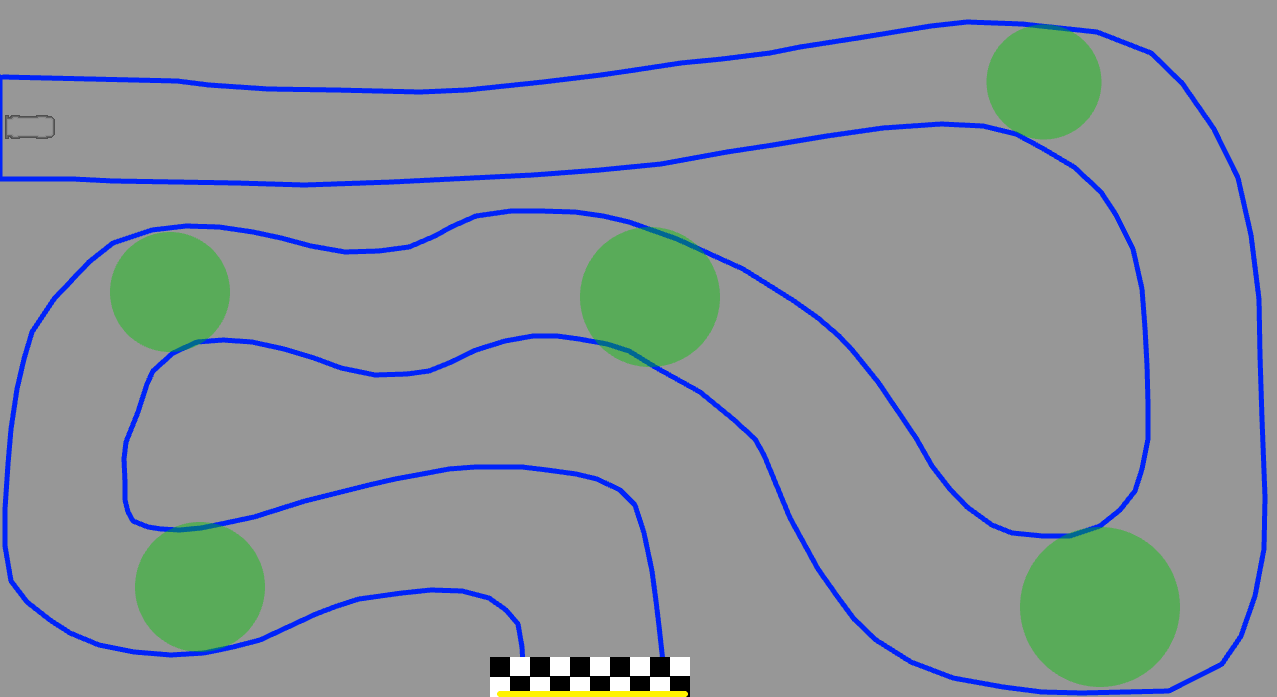
Objekt starta ima oblik točke te su u njegovoj datoteci zapisana samo dva brojeva u istom retku (x, y); x i y koordinata starta. Svi automobili kreću iz te jedne točke na početku generacije.

Objekt zida ima oblik obične linije. Datoteka *obstacles.txt* u sebi ima koordinate svakog zida određene staze. U svakom retku je ispisano 4 brojeva u ovom obliku:   
(x1, y1, x2, y2). Pomoću tih koordinata se postavljaju zidovi. Ako ima puno kratkih zidova moguće je postignuti izgled glatke zakrivljene crte.

Objekt kontrolne točke je nešto kompliciraniji. On nam služi za određivanje koji auto je došao do kojeg dijela staze te tu informaciju koristimo kod genetskog algoritma. Ima oblik kruga pa su osnovne karakteristike x i y koordinate i promjer. Osim spomenutih ima još jedan koji se naziva *fitnessMultiplier*. O njemu i njegovoj svrsi ćemo više govoriti kod objašnjavanja genetskog algoritma. Oblik svakog retka izgleda ovako: (x, y, R, fitnessMultiplier).

Cilj ima također oblik linije, ali u datoteci imamo dva retka. U prvom su koordinate cilja (x1, y1, x2, y2), a u drugom je specificirani ofset kod crtanja slike ciljne zastavice (koristi samo u vizualne svrhe).

Primjer staze:



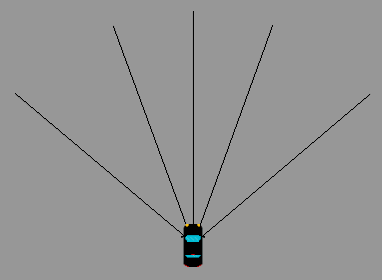
Slika 1 - Primjer staze

sivi obrub auta – start (starting point)  
plavo – zidovi (obstacles)  
zeleni krugovi – kontrolne točke (checkpoints)  
žuta linija – cilj (finish line)

## Vidokrug auta

Kod vožnje automobila ljudsko oko prima pregršt informacija iz različitih izvora. Signali dolaze kroz vjetrobransko staklo, bočnih i zadnjeg prozora kroz unutarnje ogledalo, retrovizora, kontrolne ploče, itd.. Naš mozak sve to ujedinjuje i odlučuje što nam je sljedeće činiti. Za razliku od našeg uma, um (neuronska mreža) auta u simulaciji nije toliko kompleksan. Za uspješno upravljanje dovoljno mu je samo pet ulaznih signala (input-a).

Svaki auto ima „ugrađena“ pet senzora. Senzori su postavljeni da *vide* 300 piksela ispred autića u 5 smjerova (prikazano na *Slici 2*). Od kojih je jedan postavljen da gleda ravno naprijed, dva lijevo i desno pod 20 stupnjeva (0.349 rad) i dva lijevo i desno pod 50 stupnjeva (0.873 rad) od središnjeg senzora.



Slika 2 - Prikaz senzora

# Programiranje inteligencije

Ova simulacija prikazuje učenje auta kako voziti stazom, da bi to postigli moraju imati neku razinu inteligencije. Moraju biti u mogućnosti odlučivati i birati svoje sljedeće postupke. Na primjer, moraju moći prepoznati zid te skrenuti od njega da se ne bi zaletjeli.

Inteligencija će se implementirati pomoću neuronskih mreža (NN), a učenje pomoću genetskog algoritma (GA).

Perceptron

Neuronska mreža se sastoji od mnoštva perceptrona. Perceptron u neuronskoj mreži je najlakše shvatiti kao neuron u ljudskom mozgu. On prima neke signale ili informaciju, obrađuje ih na neki način i šalje rezultat sljedećem perceptronu.

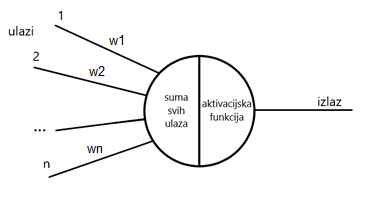
Perceptron može imati n broj ulaza. Svaki od ulaza množi se sa težinom spoja (*weight*), odnosno nekom vrijednošću koja je određene na tom spoju. Svi ulazi se zbrajaju i rezultat sume se ubacuje u aktivacijsku funkciju. Rezultat aktivacijske funkcije se smatra finalnim izlazom tog perceptrona.

Aktivacijske funkcije su matematičke jednadžbe koje određuju rezultat neuronske mreže. Aktivacijska funkcija je privržena svakom perceptronu mreže. Postoje više aktivacijskih funkcija koje su odabrane ovisno o vrsti problema i efikasnosti za pojedini problem. Najkorištenije su binarna, linearna, sigmoidna i hiperbolička funkcija. Za ovu implementaciju korištena je hiperbolička funkcija, točnije . Odabrana funkcija vraća vrijednost između -1 i 1 što je savršeno za ovu simulaciju.



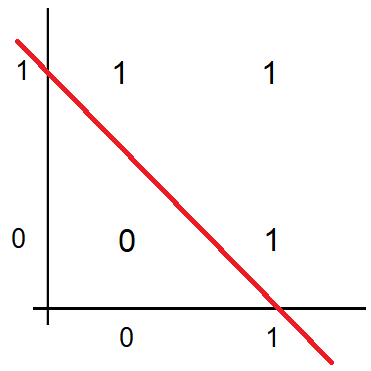
Graf 1 - hiperbolička funkcija

Kad sve to spojimo dobijemo element od kojeg je građena cijela neuronska mreža, perceptron.



Slika 3 - Perceptron

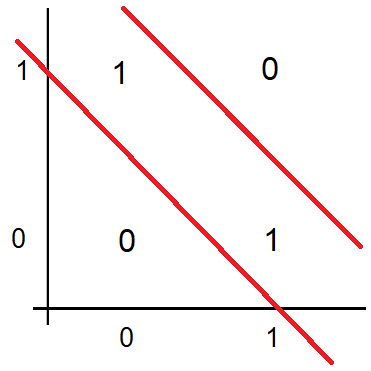
## Neuronska mreža

Neuronska mreža je osmišljena da rješava određeni tip problema. Ona rješava problem klasifikacije objekata koje nije moguće linearno podijeliti s jednim pravcem. Primjer linearno podjeljive klasifikacije su osnovna logička vrata. Uzmimo za primjer OR (ILI) vrata.

|  |  |  |
| --- | --- | --- |
| A | B | Y |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 0 | 1 |

Ako koristimo samo jedan perceptron već smo u mogućnosti rješavati kompleksnija logička vrata poput XOR vrata.

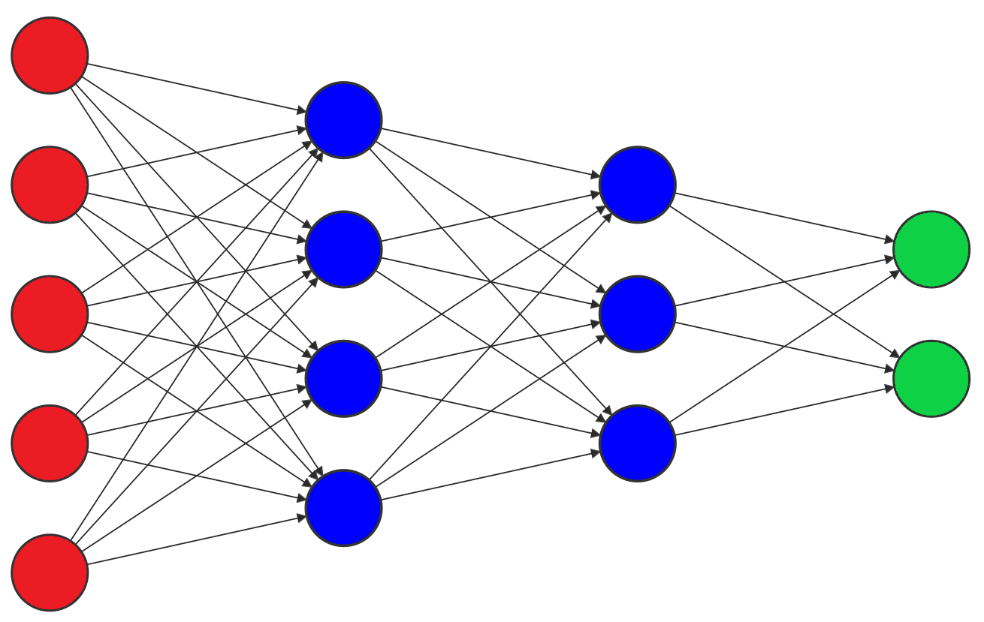
|  |  |  |
| --- | --- | --- |
| A | B | Y |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 0 | 0 |



Iz ovih jednostavnih primjera možemo zaključiti da je neuronska mreža vrlo moćan alat za klasifikaciju različitih objekata koje nije moguće podijeliti jednim pravcem već krivuljom ili više pravaca.

Neuronska mreža se sastoji od više slojeva podijeljenih u tri kategorije: ulazni sloj, skriveni slojevi i izlazni sloj. Svaki sloj se sastoji od proizvoljnog broja perceptrona. Skrivenih slojeva može biti bilo koji broj, ovisno o problemu kojeg neuronska mreža rješava, ali i mogućnosti sustava na kojem se učenje izvršava. Po broju skrivenih slojeva utvrđuje se vrsta neuronske mreže. Duboka neuronska mreža (Deep NN) ili takozvana *plitka*, tj. obična neuronska mreža. Nije točno definirano broj skrivenih slojeva koji razdjeljuje spomenute vrste neuronskih mreža, ali smatra se da svaka mreža sa više od jednog skrivenog sloja jest duboka neuronska mreža.

Postoji puno vrsta neuronskih mreža i s naglim napredovanjem strojnog učenja nove vrste se otkrivaju svaki dan. Za ovu simulaciju upotrjebljena je Potpuno povezana neuronska mreža, tj. svaki pereptron jednog sloja je povezan sa svakim perceptronom sljedećeg sloja, kao što je prikazano na slici 4. Crvenom bojom su obojani ulazni perceptroni, plavom skriveni i zelenom izlazni perceptroni.



Slika 4 - Neuronska mreža

Implementacija neuronske mreže u ovoj simulaciji odrađena je pomoću matrica.

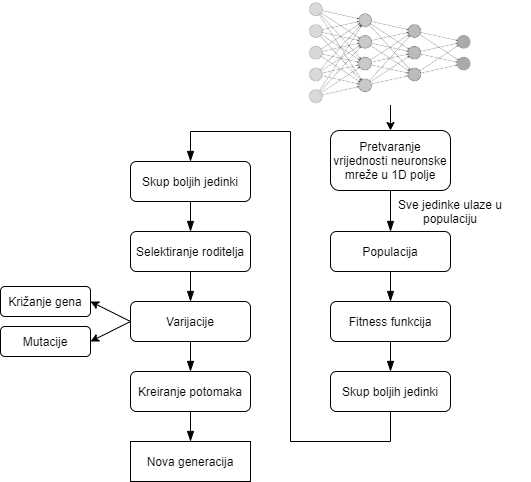
Svaku neuronska mreža čine tri matrice, svaka od njih sadrži vrijednosti težina između dva sloja NN-a. Ulazne podatke (udaljenost od zidova) moramo oblikovati u matricu oblika [5,1]. Da ostvarimo *razmišljanje* množimo ulaznu matricu s prvom matricom NN, svaki element dobivene matrice ubacujemo u aktivacijsku funkciju te opet množimo sa sljedećom matricom. Tako množimo i *aktiviramo* do kraja mreže dok ne dođemo do izlaznog sloja.

## Genetski algoritam (GA)

Kontinuirano učenje biti će implementirano pomoću genetskog algoritma. Genetski algoritam imitira evoluciju u prirodi. Kao i životinjske vrste kroz povijest, koje su se borile za opstanak tako će se i auti boriti za *opstanak* kroz više generacija. GA prati ideju „Opstanak najjačega“ i prirodne selekcije. Simulacija programa odvijat će se u vremenskim jedinicama nazvanima generacijama. Na početku svake generacije generirat će se n broj autića koji će pokušati voziti kroz stazu. Ovisno koliko daleko doguraju, na kraju generacije (kad svi autići ili završe stazu ili udare u zid), izračunati će im se *fitness score* (brojčana vrijednost koja pokazuje njihov uspjeh prelaženja staze). Autićev opstanak ovisi o njegovom uspjehu.

Genetski algoritam ima nekoliko koraka izvedbe prikazanih na Slici 5. Kad nema preostalih autića kreće genetski algoritam koji se odvija između generacija. Sve neuronske mreže se pretvaraju u 1D polje za lakše analiziranje i manipuliranje. Od svih autića bira se njih 2/3 ukupne populacije (skup boljih jedinki) koji će biti izabrani za roditelje. Preostalih 1/3 se eliminira jer nisu zadovoljili uvijete. Iz skupa boljih jedinki nasumično se izabiru po dva roditelja. Kada su svi roditelji smješteni u parove kreće proces varijacija. U tom procesu se kreira neuronska mreža potomaka. Imitirajući prirodu, potomak ima gene oba roditelja u sebi, tako će se i u simulaciji odvijati parenje. Prethodno smo svaku neuronsku mrežu roditelja pretvorili u 1D polje te ćemo to koristiti u ovom koraku. Neuronska mreža potomka se kreira uzimanjem jedne polovice NN jednog roditelja i druge polovice NN drugog roditelja te se spaja nazad u jedno polje. Ovim postupkom ostvarili smo križanje gena, ali nam još preostaju mutacije. Mutacije su ostvarene nasumičnim odabirom gena koji su postavljeni na nasumične vrijednosti. Mutacija je neophodna za genetski algoritam jer unosi određenu dozu nasumičnosti (bez mutacije nekad uopće ne dolazi do napretka). Novonastalo 1D polje nazad pretvaramo u matrice u oblik neuronske mreže. Sa kreiranom neuronskom mrežom kreiramo novi objekt Auto te ga ubacujemo u sljedeću generaciju. Taj postupak ponavljamo za svaki par roditelja.

Nova generacija se sastoji od 2/3 starih autića, tj. roditelja (iz prethodne generacije) i 1/3 novih autića, tj. potomaka.



Slika 5 - Grafički prikaz genetskog algoritma

### Fitnes

Fitness vrijednost je opisuje uspješnost prelaženja staze. Ako auto prođe dalje, veći će mu biti fitness score (fitnes vrijednost). Najlakši način izračunavanje fitnesa bi bio mjerenje jediničnih dužina (u ovom slučaju piksela) koje je auto prešao prije sudaranja. No, tu dolazimo do mogućeg problema koji se događa ovisno o širini staze. Pošto je svaki mozak nasumično konfiguriran na početku simulacije, moguće je da auto na temelju svoje konfiguracije uvijek skreće lijevo i pri tome ima malu brzinu. Takvi autići će jako dugo voziti i proći puno više od autića koji, npr. prođe pola staze. Efektivni put se ne može izjednačiti s prijeđenim putem. Rješenje za taj problem su kontrolne točke (checkpoint).

Kontrolne točke su implementirane kao kružnice koje detektiraju ako je auto ušao u kružnicu. Tu funkcionalnost možemo iskoristiti da riješimo prije navedeni problem. Ako pametno rasporedimo te kružnice po stazi možemo znati kada je auto došao do kojeg dijela staze. S tom informacijom možemo manipulirati fitnes i onom autiću koji je prošao pola staze dati veću vrijednost od onog koji se na početku vrti u krug. Simulacija se izvodi u određenim brojem sličica po sekundi (*fps* – *frame per second*), najčešće između 50 i 60 sličica u sekundi. Svakom sličicom se povećava fitnes autića (naravno samo ako je još *živ*). Fitnes se izračunava sljedećom formulom:

1. fitness = distTravelled \* fitnessMultiplier;

gdje je *distTravelled* udaljenost koju je auto prošao i *fitnessMultipier* fitnes multiplikator. FitnessMultiplier je vrijednost koja se povećava ako auto stigne do određene kontrolne točke. S tim algoritmom pridodajemo veću vrijednost prelasku staze, a manju vrijednost čistim prijeđenim putem. Riješili smo pola problema. Zbog nasumičnosti prirode simulacije moguće je da će se auto neodređeno dugo vrtjeti na mjestu. Rješenje je implementacija brojača koji *onesposobi* auto (postavlja ga u isto stanje kao da se je sudario) ako u određenom vremenu ne dođe do kontrolne točke. Također je implementirana tipka koja ručno onesposobi sve aute ako prijašnje rješenje nije prikladno.

# Zaključak

# Bibliografija

Bošnjak, M., 2011.. *Strojno učenje, PMF.* [Mrežno]   
Available at: https://web.math.pmf.unizg.hr/nastava/su/index.php  
[Pokušaj pristupa 14 Prosinac 2019.].

Thompson, J., 2019. *Collision detection.* [Mrežno]   
Available at: http://jeffreythompson.org/collision-detection/line-line.php  
[Pokušaj pristupa 28 Studeni 2019].

# Popis slika

[Slika 1 - Primjer staze 7](#_Toc36070118)

[Slika 2 - Prikaz senzora 8](#_Toc36070119)

[Slika 3 - Perceptron 10](#_Toc36070120)

[Slika 4 - Neuronska mreža 12](#_Toc36070121)

# Popis grafova

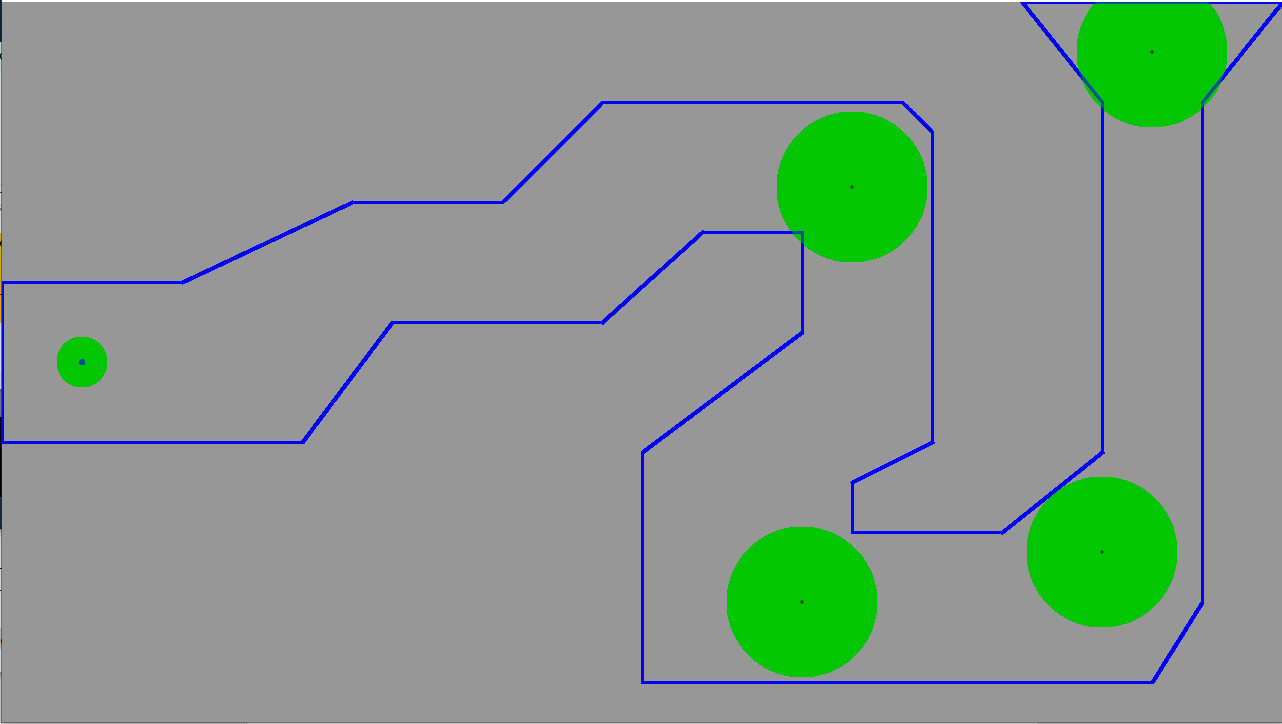
[Graf 1 - hiperbolička funkcija 10](file:///D:\Github\Neural-network-cars\Dokumentacija.docx#_Toc36070397)

# Dodaci

Prva faza – kreiranje mape

Zbog samog funkcioniranja programa Processing način kreiranje staze za autiće morao je biti manualan. Kod pokretanja Processing programa kreira se kanvas za crtanje, te svi entiteti koje vidimo o programu moraju biti „nacrtani“. Tako i granice staze.

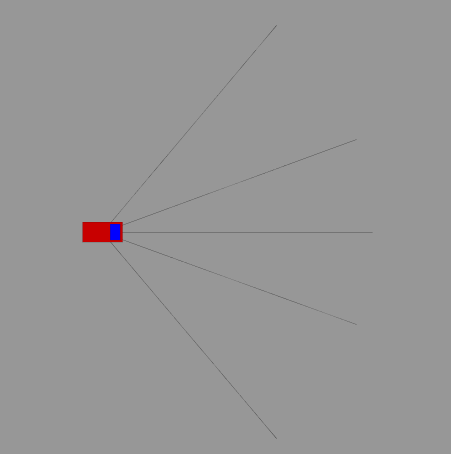
Jedna od bitnijih dijelova mape su takozvani „checkpointevi“ koji će nam služiti u genetskom algoritmu kroz koji ćemo proći kasnije. Oni su postavljeni na relativno bitne dijelove mape, nakon zavoja ili težih prepreka. Kad auto uđe u radijus checkpoint-a tada program prepozna da je auto stigao do određenog dijela staze.



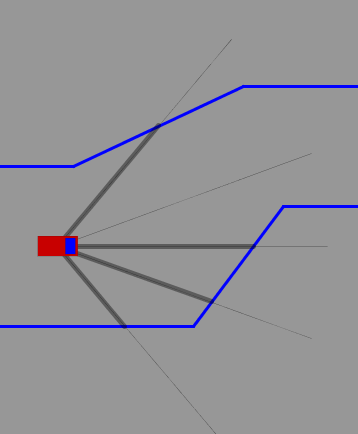
Druga faza – kreiranje autića

Tijelo auta je najobičniji pravokutnik te radi prepoznavanja koji je prednji i zadnji dio auta dodao sam prednje vjetrobransko staklo.

Najbitniji dio auta nije tijelo auta već senzori. Senzori nam prikazuju što i do koje udaljenost autić *vidi*. Senzori su postavljeni da *vide* 300 piksela ispred autića u 5 smjerova. Od kojih je jedan postavljen da gleda ravno naprijed, dva lijevo i desno pod 20 stupnjeva i dva lijevo i desno pod 50 stupnjeva od središnjeg senzora.



Da bi mi uočili i razumjeli što auto vidi, postavio sam uvjet ako senzor nešto prepozna ispred sebe, da se sama crta koja označuje senzor podeblja. S tim svojstvom nama je puno lakše vizualizirati autićevo vidno polje.



Problemi i rješenja

Problem 1: Auto ne detektira zid na vrijeme

Razlog: Funkcija isDead() gleda samo jedan senzor te po njegovoj vrijednosti postavlja dead na true ili false. Ako je taj određeni senzor okrenut lijevo, auto ne detektira zid s desne strane na vrijeme.

Rješenje: For petljom utvrđujem najmanju udaljenost zida svih senzora te istu šaljem isDead() funkciji.

Postupak generiranja nove populacije

1. Provjeravanje da li su svi autići završili u zidu
2. Izračunavanje ukupnog fitnessa svih autića
3. Postavljanje fitnessa svakog od autića po udjelu u ukupnom fitnessu

mojFitness = mojFitness / ukupniFitness;

1. Biranje 2/3 od ukupne populacije autiće koji će nastaviti putovanje pomoću Tournament selection.

Algoritam:

* + 1. Od svih autića biramo 2 autića nasumično.
    2. Od tih 2 uzmemo onaj s boljim fitnessom. Te ga zapisujemo u *winner* polje
    3. Ponavljamo korak 1 i 2 tako dugo dok ne dobijemo 2/3 od ukupnog broja autića u *winner* polje

1. Autići iz *winner* polja su odabrani za produciranje novih autića. (parent – child odnos). Child autić dobijemo tako da uzmemo 2 autića nasumično iz *winner* polja te radimo „crossover“. Pola neural network child dobije od prvog roditelja a drugu polovicu od drugog roditelja.
2. Mutation. Kao i u prirodi kod evolucije se događaju mutacije pa tako i u evolucijskome programu. Stopa mutacije je određena konstantom [0.0001, 0.1]. Stopa mutacije određuje koja je vjerojatnost da se određeni dio neural networka, tj. pojedini weight, promjeni.
3. Takvi novi autići prelaze u novu populaciju (1/3) zajedno sa svojim roditeljima (2/3).

Problem: Generacija 2 odmah mrtva, i neprestano generiranje novih generacija, također mrtvih. Debugiranje trajalo 4 sata. Rješenje: timer koji je mjerio vrijeme od početka programa ako auto ne prijeđe određeni checkpoint na vrijeme, umire se nakon završetka jedne populacije nije resetirao pa svaka sljedeća populacija i svi autići nisi bili u mogućnosti doći do checkpoint-a na vrijeme pa zato i svi bili mrtvi.