Tehnička škola Čakovec

ELABORAT ZAVRŠNOG RADA

AI SIMULACIJA

Mentor:

Krešimir Kočiš, prof.

Učenik:

Toni Polanec, 4.RT

Čakovec, svibanj 2020.

Tehnička škola Čakovec

Prosudbeni odbor za završni rad

Učenik: Toni Polanec

Razred: 4.RT

Školska godina: 2019./2020.

Obrazovno područje: računalstvo

Zanimanje: tehničar za računalstvo

Naziv zadatka: AI SIMULACIJA – UČENJE AUTIĆA DA VOZE KROZ STAZU

Opis zadatka:

2D simulacija u kojoj autići sami uče voziti kroz stazu koristeći neuronsku mrežu i evolucijske algoritme.

Učenik će se za konzultacije obratiti svojem mentoru.

Zadatak zadan:	**Rok predaje pisanog rada:**	**Predviđeni datum obrane:**
21. listopada 2019.	20. svibnja 2020.	

Mentor:

Krešimir Kočiš, dipl. ing. rač.

1 SADRŽAJ

2 UVOD

U ovom radu pisat ću 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 u određenoj generaciji dosegne neki nivo inteligencije i 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 stazu.

Rad će biti razrađen na dva najbitnija dijela. Programiranje kretanje autića 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 i prolazak cilja).

Drugi dio je programiranje inteligencije. Svaki auto će se sam voziti. Mozak će biti implementiran neuronskom mrežom. Postupak svakog od autića bit će određen s onim što on vidi (senzori, tj. udaljenosti od zidova staze) i stanjem njegovih neurona u mozgu. Učenje je implementirano genetskim algoritmom. Nakon svake generacije kroz razne algoritme određivat će se roditelji koji će svoje gene proslijediti potomcima 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 njezina primjena budućnost tehnologije koja će uvelike pomoći razvoju čovječanstva.

3 PROGRAMIRANJE FUNKCIONALNOSTI AUTA

U ovom programu je najbitniji objekt auto. U sljedećih nekoliko ulomaka bit će razloženo kako i na koji način su implementirane potrebne funkcije auta za simulaciju. Podijeljeno je na 3 dijelova: kretanje, vidokrug i stanja auta. Potrebno je objasniti ove dijelove za lakše shvaćanje neuronske mreže te kako neuronska mreža upravlja autom.

3.1 KRETANJE

Auto ima dvije najbitnije funkcije – njegova brzina i skretanje. Kretanje je realizirano pomoću tri vektora: *velocity* (brzina), *acceleration* (ubrzanje) i *location* (lokacija/pozicija). Svaku iteraciju simulacije *location* vektoru se pribraja *velocity* vektor, a *velocity* vektoru se pribraja *acceleration* vektor koji u glavnini upravlja autom. Zajedno, ta tri vektora omogućuju realno kontroliranje auta.

Neuronska mreža kao rezultat izbacuje dva broja $\in$[-1,1] koji upravljaju brzinom i skretanjem. Vrijednost -1 označuje najmanju brzinu ili potpuno skretanje ulijevo i vrijednost 1 koja označuje najveću brzinu i potpuno skretanje udesno. Kut skretanja je u opsegu [-0.05, 0.05] radijana što bi u stupnjevima bilo 2.86º. To znači da u jednoj iteraciji simulacije auto može najviše skrenuti za 2.86º. Ako je vrijednost maksimalnog skretanja prevelika može se dogoditi da se auto vrti na mjestu, a ako je premala auto nije u mogućnosti izvoziti ni najblaži zavoj. Maksimalna brzina je limitirana na neki proizvoljan broj.

Ukratko objašnjeno, *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. U primjeru da 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.

4 INTERAKCIJA S OKOLINOM

Svaki trkaći auto mora imati svoju stazu pa će u sljedećim ulomcima biti objašnjeno kreiranje staze i njezine karakteristike. Također će biti razložen način viđenja staze iz perspektive auta.

4.1 STAZA

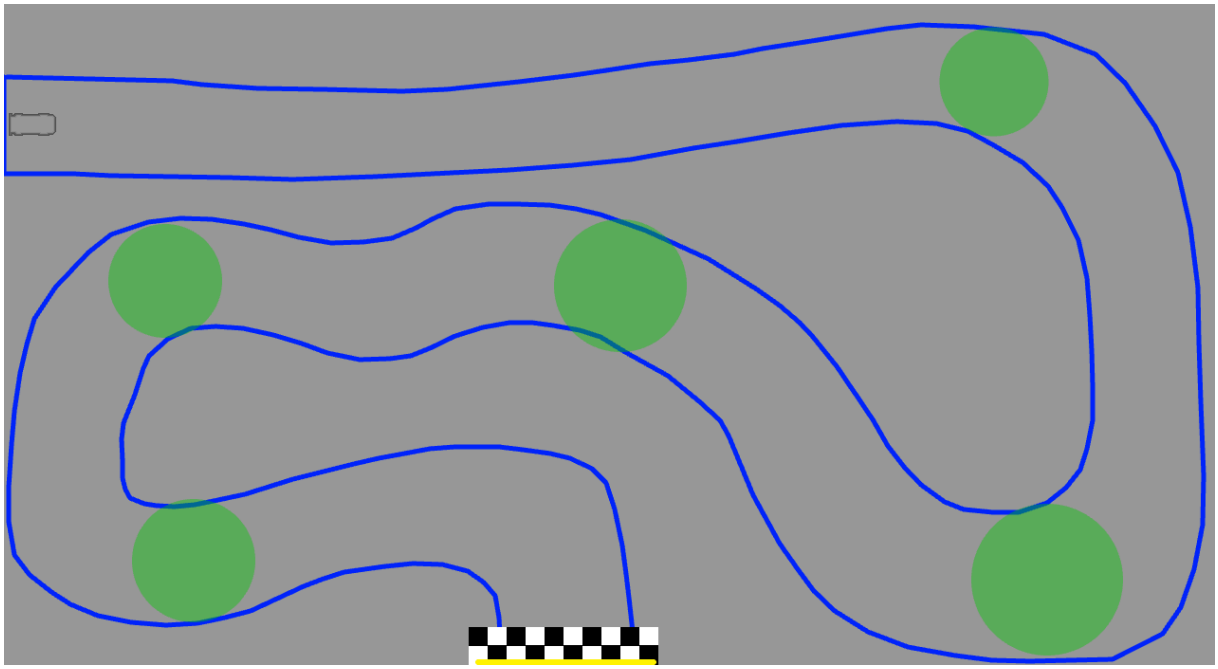
Svaka staza se sastoji od četiri objekata: *starting point* (koordinate starta), *obstacles* (zidovi staze), *checkpoints* (kontrolne točke) i *finish line* (ciljna linija). Primjer staze je prikazan na Slici 1. Kod pokretanja programa prikažu se tri izrađene staze (kategorizirane po težini) i jedan upitnik s natpisom „Make your own“ koji će biti spomenut kasnije. Karakteristike staza se uvoze 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, što 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 iste 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). X1 i y1 za početnu točku zida i x2 i y2 za krajnju točku zida. 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 također ima 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).



Slika 1 - Primjer staze

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

4.1.1 Kreator staza

Kod pokretanja simulacije, dočeka nas ekran sa mogućnošću odabira staze. Moguće je odabrati jednu od tri prije napravljene staze koje su rangirane od najlakše do najteže. Rangirane su po procjeni koliko autićima prosječno treba da se nauče voziti kroz nju. Moguće je odabrati i četvrtu varijantu, a to je kreator staza.

Kreator staza nam omogućuje da sami kreiramo stazu po kojoj se autići uče voziti. Ona može služiti u istraživačke svrhe. Možemo eksperimentirati i gledati na koji način će ova primitivna umjetna inteligencija pokušati riješiti problem koji je

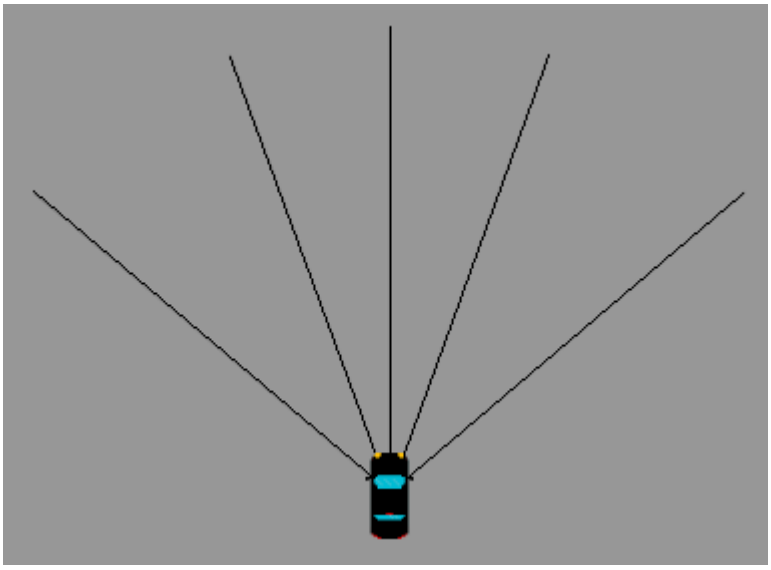
predstavljen ispred nje. Često to bude rješenje koje nas iznenadi i zainteresira za daljnje eksperimentiranje.

Da bi staza bila valjana, u kreatoru staza moramo postaviti sve esencijalne elemente: startna točka, zidovi, kontrolne točke (barem jednu) i ciljnu liniju. Bez svih potrebnih elemenata simulacija nije moguć. Na ekranu kreatora je moguće vidjeti sve kontrole i savjete kako napraviti stazu.

4.2 VIDOKRUG AUTA

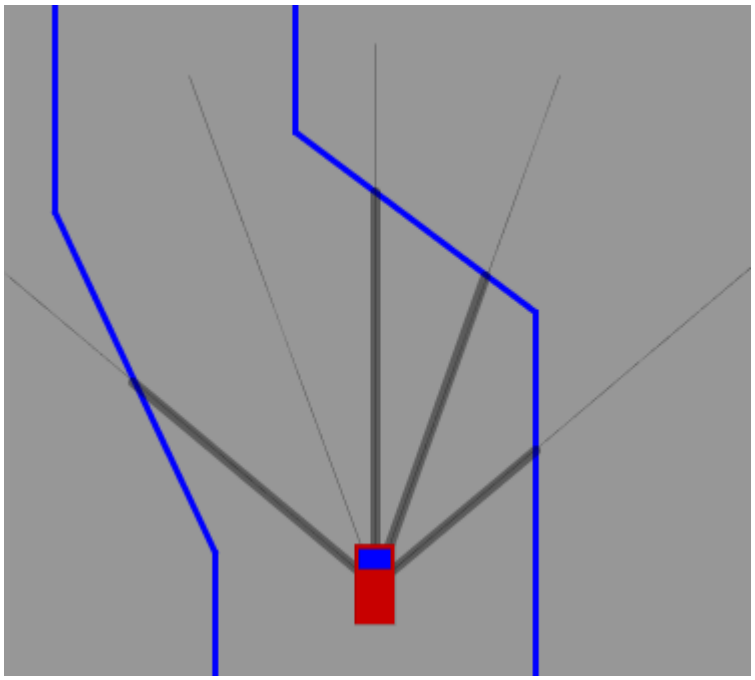
Kod vožnje automobila ljudsko oko prima pregršt informacija iz različitih izvora. Signali dolaze kroz vjetrobransko staklo, bočnih i zadnjeg prozora kroz unutarnje ogledalo, retrovizora, kontrolne ploče, itd.. Naš mozak sve to ujedinjuje i odlučuje što nam je sljedeće činiti. Za razliku od našeg uma, 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

Senzori detektiraju zidove i vraćaju vrijednost koja se odnosi na udaljenost tog zida od autića. Ako senzor ne doseže zid, vraća se maksimalna vrijednost senzora, ovisno o dužini senzora. Detekcija zida može se vidjeti na Slici 3 gdje su podebljani dijelovi senzora koji *vide* zid.



Slika 3 - Detekcija zidova

Za realizaciju takvog algoritma potrebno je razumijevanje geometrije. Svaki senzor kada je pozvana njegova funkcija *see()* (funkcija prikazana na slijedećoj stranici) provjerava svaki zid staze i traži točku sjecišta. Pošto se simulacija odvija u dvodimenzionalnom prostoru te mi na simulaciju gledamo *od gore* moguće je primijeniti relativnu laku formulu za izračunavanje sjecišta dvaju pravaca. Do problema je dolazilo ako bi dva zida bili blizu jedan iza drugoga, senzor bi prepoznao zid koji je se nalazi iza. Razlog tomu je prolazak kroz for petlju. Najčešće bi onaj zid koji je dalje od startne pozicije bio zapisan kasnije u polje zidova. Funkcija *see()* bi prepoznala bližnji zid, ali zbog potpunog prolaska kroz polje taj zid bi bio prebrisan sa slijedećim. Najjednostavnije rješenje tomu je svaki put kada se utvrdi zid koji ima točku sjecišta sa senzorom, provjerava se da li je baš taj zid najbliži autu. Ako nije najbliži niti se ne uzima u obzir.

```
1.  float see() {
2.      float tempUdaljenost = 0;
3.      float udaljenost = sensorStrength;
4.      float sjecisteX = loc.x, sjecisteY = loc.y;
5.
6.      for (Obstacle o : m.obstacles) {
7.        float x1 = o.x1;
8.        float y1 = o.y1;
9.        float x2 = o.x2;
10.       float y2 = o.y2;
11.
12.   // Matematicki izracun za dobivanje tocke do koje pojedini senzor vidi.
13.       float uA = ((x4-x3)*(y1-y3) - (y4-y3)*(x1-x3)) / ((y4-y3)*(x2-
    x1) - (x4-x3)*(y2-y1));
14.       float uB = ((x2-x1)*(y1-y3) - (y2-y1)*(x1-x3)) / ((y4-y3)*(x2-
    x1) - (x4-x3)*(y2-y1));
15.
16.       if (uA >= 0 && uA <= 1 && uB >= 0 && uB <= 1) {
17.         float tempX = x1 + (uA*(x2-x1));
18.         float tempY = y1 + (uA*(y2-y1));
19.         tempUdaljenost = dist(x3, y3, tempX, tempY);
20.
21.       // Pronalazenje tocke do koje senzor vidi (ne vidi dalje od zida).
22.         if (tempUdaljenost < udaljenost) {
23.           udaljenost = tempUdaljenost;
24.           sjecisteX = tempX;
25.           sjecisteY = tempY;
26.         }
27.       } else if (uA <= 0 && uA >= 1 && uB <= 0 && uB >= 1) {
28.         udaljenost = sensorStrength;
29.       }
30.     }
31.
32.     return udaljenost;
33.   }
```

5 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 realizirati pomoću neuronskih mreža (NN), a učenje pomoću genetskog algoritma (GA). Kroz generacije genetskim algoritmom birat će se najbolje jedinke koje će imati potomke. Svaka jedinka ima neuronsku mrežu koja je krucijalna za uspješno obavljanje zadatka.

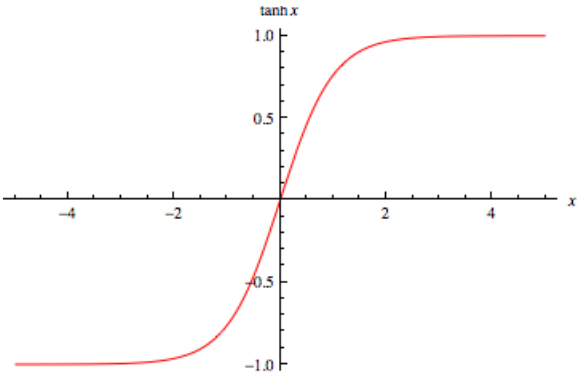
5.1 PERCEPTRON

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

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

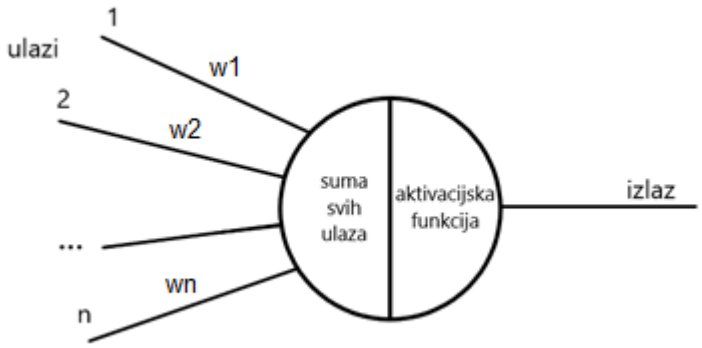
Aktivacijske funkcije su matematičke jednadžbe koje određuju rezultat neuronske mreže [1]. Aktivacijska funkcija je privržena svakom perceptronu. Postoje više aktivacijskih funkcija koje su odabrane ovisno o vrsti problema i efikasnosti za pojedini problem. Najkorištenije su binarna, linearna, sigmoidna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

Graf 1 - Hiperbolička funkcija



$y = \tanh x$. Odabrana funkcija vraća vrijednost između -1 i 1 što je savršeno za ovu simulaciju.

Kad sve to spojimo dobijemo element od kojeg je građena cijela neuronska mreža, perceptron (vizualiziran na Slici 4).



Slika 4 - Perceptron

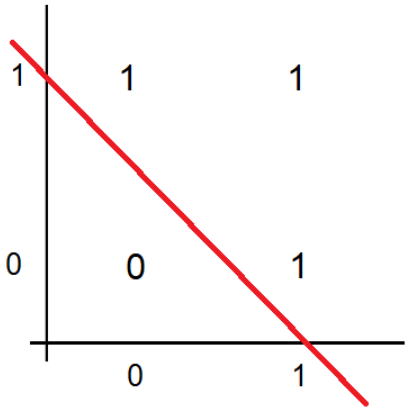
5.2 NEURONSKA MREŽA

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 neka od osnovnih logičkih vrata. Uzmimo za primjer OR (ILI) vrata.

A	B	Y
0	0	0
0	1	1
1	0	1
1	0	1

Tablica 1 - Tablica OR logičkih vrata

Graf 2 - Klasifikacija OR logičkih vrata

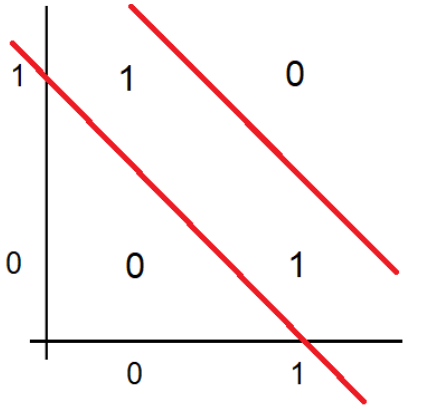


Ako koristimo samo jedan perceptron već smo u mogućnosti rješavati kompleksnija logička vrata poput XOR vrata.

A	B	Y
0	0	0
0	1	1
1	0	1
1	0	0

Tablica 2 - Tablica XOR logičkih vrata

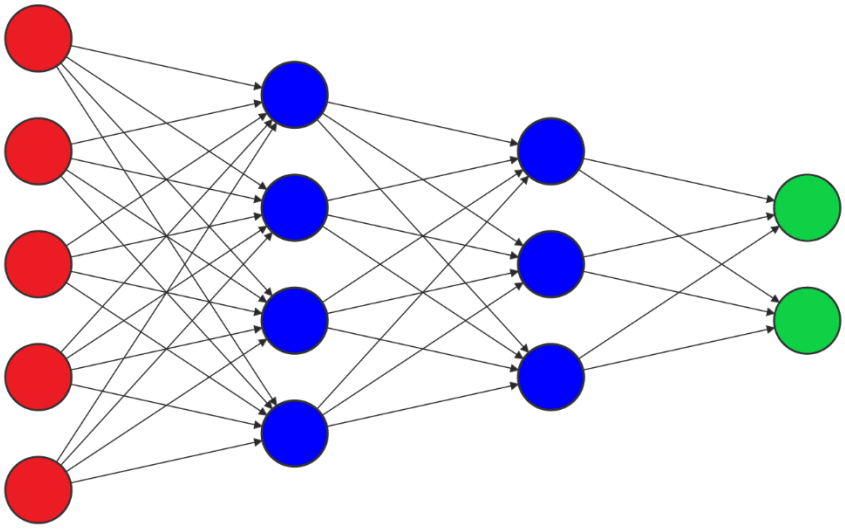
Graf 3 - Klasifikacija XOR logičkih vrata



Naravno, u neuronskoj mreži imamo puno veći broj perceptrona, pa se i moć rješavanja problema povećava. Iz ovih jednostavnih primjera možemo zaključiti da je neuronska mreža jedan 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 upotrijebljena je Potpuno povezana neuronska mreža [2], tj. svaki pereptron jednog sloja je povezan sa svakim perceptronom sljedećeg sloja, kao što je prikazano na Slici 5. Na Slici 5 crvenom su bojom obojani perceptroni ulaznog sloja, plavom skrivenog i zelenom izlaznog sloja.



Slika 5 - Neuronska mreža

Implementacija neuronske mreže u ovoj simulaciji odrađena je pomoću matrica.

Svaku neuronsku mrežu čine tri matrice, svaka od njih sadrži vrijednosti težina između dva sloja neuronske mreže. Ulazne podatke (udaljenost od zidova) moramo oblikovati u matricu oblika [5,1]. Da ostvarimo *razmišljanje* množimo ulaznu matricu s prvom matricom neuronske mreže, 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 [3].

U priloženom kodnom isječku ispod je prikazana implementacija aktivacijske funkcije. Svaki element matrice prolazi kroz aktivacijsku funkciju.

```
1.  Matrix activationF(Matrix a) {
2.      double[][] tempArrayOfMatrix = a.getArray();
3.      double[][] outArray = new double[tempArrayOfMatrix.length][tempArrayOfMa
    trix[0].length];
4.
5.      for (int i=0; i<tempArrayOfMatrix.length; i++) {
6.        for (int j=0; j<tempArrayOfMatrix[0].length; j++) {
7.
8.          // Aktivacijska funkcija f(x) = tanh(x)
9.          outArray[i][j] = Math.tanh(tempArrayOfMatrix[i][j]);
10.       }
11.     }
12.
13.     Matrix outMatrix = new Matrix(outArray);
14.     return outMatrix;
15.   }
```

Matrice su implementirane pomoću biblioteke Jama (Java Matrix Class) [4].

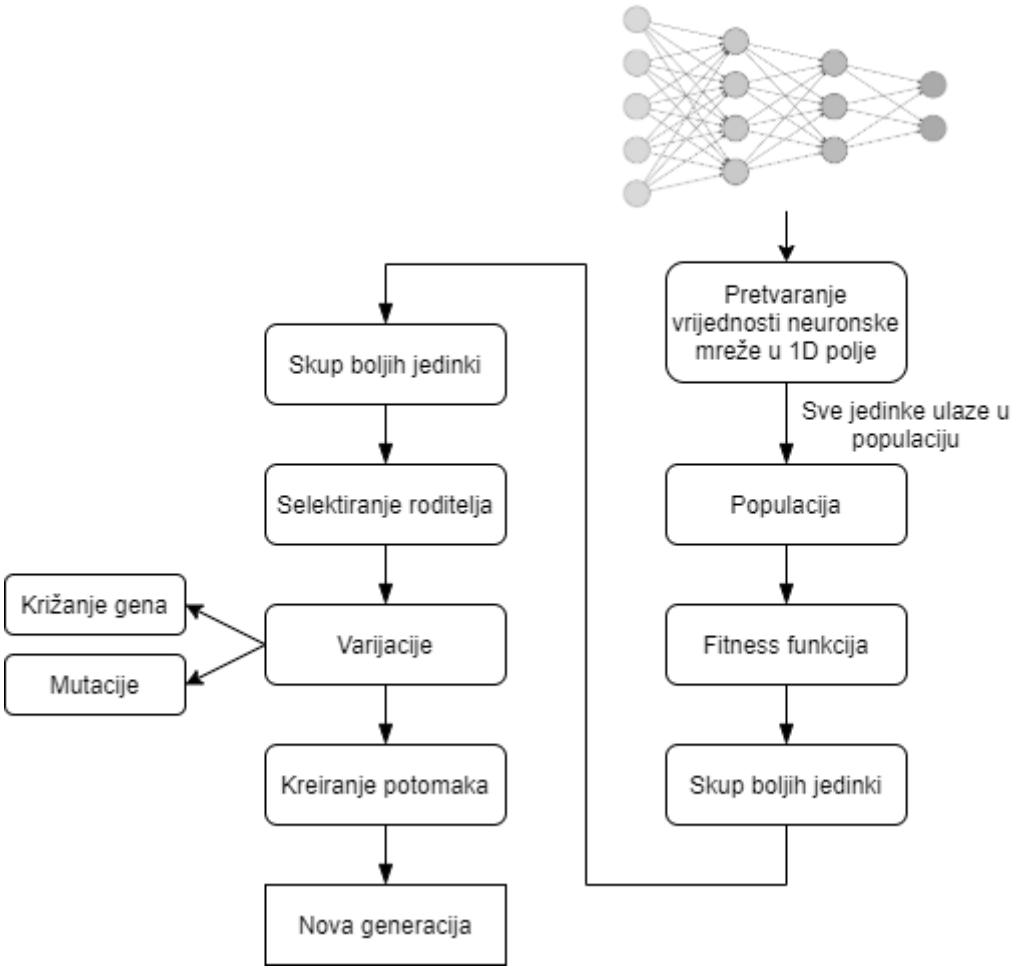
5.3 GENETSKI ALGORITAM (GA)

Kontinuirano učenje bit će realizirano 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 stazom. Ovisno koliko daleko doguraju, na kraju generacije (kad svi autići ili završe stazu ili udare u zid), izračunat će im se *fitness score* (brojčana vrijednost koja pokazuje njihov uspjeh prelaženja staze). Autićev opstanak ovisi o njegovom uspjehu prolaska staze.

Genetski algoritam ima nekoliko koraka izvedbe prikazanih na Slici 6 [5]. Kad nema preostalih autića kreće genetski algoritam koji se odvija između generacija. Sve neuronske mreže se pretvaraju u jednodimenzionalno polje za lakše analiziranje i manipuliranje. Od svih autića bira se 2/3 ukupne populacije (skup boljih jedinki) koji će biti izabrani za roditelje. Skup boljih jedinki se kreira pomoću algoritma koji se naziva Metoda turnira (*Tournament Selection method*) [6]. Od cijele populacije biramo postepeno po dvije jedinke. Između njih dvije odabiremo „bolju“ (veća fitnes vrijednost). Tako nastavljamo dok ne eliminiramo 1/3 populacije. Iz skupa boljih jedinki nasumično se izabiru po dva roditelja. Kada su svi roditelji smješteni u parove kreće proces varijacije. 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 jednodimenzionalno polje te ćemo takav oblik koristiti u ovom koraku. Neuronska mreža potomka kreira se uzimanjem jedne polovice neuronske mreže jednog roditelja i druge polovice neuronske mreže drugog roditelja te se spaja u jedno polje. Ovim postupkom ostvarili smo križanje gena, ali nam još preostaju mutacije. Mutacije su ostvarene nasumičnim odabirom gena koji su, kod mutiranja, postavljeni na nasumične vrijednosti. Mutacija je neophodna za uspješnost genetskog algoritma jer unosi određenu dozu nasumičnosti (bez mutacije često uopće ne dolazi do napretka). Novonastalo jednodimenzionalno polje vraćamo u oblik matrica kreirajući novu neuronsku mrežu. Sa kreiranom neuronskom mrežom

kreiramo novi objekt Auta 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 6 - Grafički prikaz genetskog algoritma

Kao i na dijagramu na Slici 6 genetski algoritam je tako i realiziran u kodu:

```
1.  void populationIsDead() {
2.      println("Generacija " + populationNumber + " je gotova.");
3.
4.      takeFitnesses();
5.      choosingWinnerCars();
6.      parentSelection();
7.      makingBabies();
8.      goingToNextGeneration();
9.    }
```

Priložena funkcija se poziva kada svi objekti iz populacije ili završe stazu ili se zalete u zid. Funkcija *populationIsDead()* pokreće sve prije spomenute procese.

5.3.1 Fitnes

Fitnes vrijednost je opisuje uspješnost prelaženja staze. Ako auto pređe više, 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 i konfiguraciji neuronske mreže. Pošto je svaki mozak nasumično konfiguriran na početku simulacije, moguće je da auto, na temelju svoje konfiguracije uvijek skreće ulijevo i pri tome ima malu brzinu. Ako je staza dovoljno široka moguće je da će se takav auto beskonačno dugo vrtjeti u krug bez da se sudari sa zidom. 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 (*checkpoints*).

Kontrolne točke su implementirane kao kružnice koje detektiraju ako je auto ušao u tu 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 vrijednost i onom autiću koji je prošao pola staze dati veću vrijednost od onog koji se na startu vrti u krug. Simulacija se izvodi u određenim brojem sličica (iteracija) po sekundi (*fps – frame per second*), najčešće između 50 i 60 sličica u sekundi. Svakom iteracijom se povećava fitnes autića (naravno 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 količini prijeđenog puta. Riješili smo pola problema. Zbog nasumičnosti prirode simulacije moguće je da će se auto neodređeno dugo vrtjeti po stazi (proći će pola staze pa će se okrenuti i voziti prema startu). 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.

6 ZAKLJUČAK

Kada autići prođu kroz par generacija, njihov napredak je sve vidljiviji. Svakom generacijom autići dosegnu veću udaljenost, a kada dođu do cilja, broj autića koji završe stazu se povećava. Da bi dokazali njihovu „inteligenciju“ moramo pokazati da se nisu samo naučili voziti kroz tu određenu stazu, već da su se općenito naučili voziti. Bez obzira na kakvu stazu bi ih stavili, oni bi morali lagano svladavati zavoje i možda, iz prve, proći cijelom, prije nepoznatom, stazom. To napravimo tako da između generacija promijenimo stazu. U simulaciji su omogućene tri (plus jedna) staza, njih možemo birati tijekom izvođenja simulacije da provjerimo njihovu inteligenciju. Testiranjem je utvrđeno da se oni nauče voziti, a ne da samo nauče stazu napamet odvoziti.

Veoma je zanimljivo kako se autići postepeno kroz generacije poboljšavaju. Meni osobno promatranje tog napretka ima neku vrstu terapeutskog učinka. Nerijetko osoba može izgubiti pojam o vremenu igrajući se sa simulacijom i eksperimentiranjem različitih staza. Najzanimljivije je kreirati raznorazne prepreke te promatrati kako će autići riješiti te probleme.

Implementacija genetskog algoritma sa neuronskom mrežom je vrlo dobar način ulaska u svijet umjetne inteligencije. Iako sa suvremenim tehnologijama i sustavima, ovakva simulacija izgleda vrlo primitivno i jednostavno. No, za početnike i osobe koje zanima umjetna inteligencija to je vrlo dobar uvod u to područje računarstva. Nakon završetka ovog rada imam puno šire znanje o programiranju neuronskih mreža. Vrsta znanja koju je nemoguće naučiti iz knjiga, videozapisa i sličnih izvora. Puno bolje razumijem pozadinu koja se krije iza zastrašujućeg imena „umjetne inteligencije“. Također sada puno češće i lakše prepoznajem raznorazne primjene te tehnologije u svakodnevnom svijetu.

Smatram da je rad poput ovoga jako dobra odskočna daska za daljnje, kompliciranije projekte strojnog učenja i umjetne inteligencije. Neke od kojih već imam u glavi i planiram ostvariti u čim kraćem roku.

7 BIBLIOGRAFIJA

1. Activation Function, veljača 2014. URL: https://en.wikipedia.org/wiki/Activation_function (2019-12-14)
2. Popčević, Jelena, Varga, Ines, Žuvela, Petar. Seminarski rad iz kolegija Uvod u matematičke metode u inženjerstvu, srpanj 2012. URL: http://matematika.fkit.hr/novo/izborni/referati/Popcevic_Varga_Zuvela_Neuronske_mreze.pdf (2019-12-14)
3. Fortuner, Brendan, Chaudhary, Puru. Forward propagation, 4. prosinca 2019. URL: https://ml-cheatsheet.readthedocs.io/en/latest/forwardpropagation.html (2020-01-15)
4. Jama Class, 2012. URL: https://math.nist.gov/javanumerics/jama/doc/Jama/Matrix.html (2020-01-11)
5. Gad, Ahmed. Artificial Neural Networks Optimization using Genetic Algorithm with Python, 7. ožujka 2019. URL: https://towardsdatascience.com/artificial-neural-networks-optimization-using-genetic-algorithm-with-python-1fe8ed17733e (2020-01-13)
6. Sil, Pritam. Tournament Selection (GA), rujan 2018. URL: https://www.geeksforgeeks.org/tournament-selection-ga (2020-01-22)

8 POPIS SLIKA

9 POPIS GRAFOVA

10 POPIS TABLICA

11 DODACI

TEHNIČKA ŠKOLA ČAKOVEC

EVIDENCIJSKI LIST KONZULTACIJA

IZRADE ZAVRŠNOG RADA

Ime i prezime učenika **Toni Polanec**

Razred **4.RT**

Program – zanimanje **Tehničar za računalstvo**

Mentor **Krešimir Kočiš, dipl. ing. el.**

Redni broj	DATUM KONZULTACIJE	SADRŽAJ RADA	POTPIS MENTORA
1.	11.01.2020.	Prikaz rada jednog auta te prelazak na rad populacije autića	Kočiš
2.	01.02.2020.	Napravljena populacija i kroz generacije autići počeli učiti	Kočiš
3.	15.02.2020.	Prikaz rada simulacije. Savjeti za pisanje dokumentacije	Kočiš
4.			
5.			

Prijedlog ocjene i potpis mentora