Napredne tehnike pronalaženja puteva primjenom genetskog algoritma u Unityu

Muženjak, Nikola

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Napredne tehnike pronalaženja puteva primjenom genetskog algoritma u Unityu

Završni rad

Nikola Muženjak

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 14.07.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Nikola Muženjak
Studij, smjer:	Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4543, 27.07.2020.
OIB Pristupnika:	78589559495
Mentor:	izv. prof. dr. sc. Časlav Livada
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Napredne tehnike pronalaženja puteva primjenom genetskog algoritma u Unityu
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak završnog rad:	U radu je potrebno opisati temeljne principe neuronskih mreža i genetskog algoritma pronalaženja puta u zatvorenim prostorima. Praktični dio rada potrebno je napraviti u Unityu. Tema rezervirana za: Nikola Muženjak
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	14.07.2023.
Datum potvrde ocjene od strane Odbora:	26.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2023.

Ime i prezime studenta:	Nikola Muženjak
Studij:	Programsko inženjerstvo
Mat. br. studenta, godina upisa:	R4543, 27.07.2020.
Turnitin podudaranje [%]:	14

Ovom izjavom izjavljujem da je rad pod nazivom**: Napredne tehnike pronalaženja puteva primjenom genetskog algoritma u Unityu**

izrađen pod vodstvom mentora izv. prof. dr. sc. Časlav Livada

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

U današnje vrijeme kada je umjetna inteligencija postala stvar svakodnevice te podaci koji pristižu sa svih strana postaju sve veći i kompleksniji, dolazi do potrebe za njihovom obradom. Kako bi se ti problemi riješili, koriste se koncepti koji su stekli popularnost tek u posljednjih deset godina, a to su neuronske mreže. Neuronske mreže, čiji je nagli razvoj tek „nedavno“ započeo, nalaze vrlo visoku razinu uspješnosti u područjima predviđanja i klasifikacije te zbog toga nalaze primjenu u strojnome učenju. U ovom će radu biti opisana korelacija između umjetne i biološke neuronske mreže. Također, biti će objašnjeni svi dijelovi neuronske mreže te kako oni rade i na koji su način međusobno povezani. Razvoj neuronskih mreža omogućuje njihovo povezivanje s raznim drugim konceptima i algoritmima, a jedni od njih su genetski algoritmi. Kako bi se mogla izvršiti implementacija takvoga algoritma, postojala je potreba objašnjenja genetskoga algoritma kroz ovaj rad. Nadalje, ovaj će rad objasniti princip rada genetskoga algoritma te kasnije izdvojiti svaki korak kojeg detaljno opisuje i objasniti koje se sve metode mogu koristiti. Također, objasniti će podjelu genetskoga algoritma i ukratko opisati izgled svake vrste tog algoritma te međusobne razlike. Međusobno povezivanje neuronske mreže i genetskoga algoritma omogućava rješavanje raznih novih problema. Jedan od problema za koji je odrađena programska implementacija te detaljno opisan način korištenja je problem pronalaženja puteva.

1.1. Zadatak završnog rada

Detaljno analizirati i opisati neuronsku mrežu i genetski algoritam. Napraviti programsku implementaciju neuronske mreže s genetskim algoritmom te ga objasniti i napisati upute za njegovo korištenje. Dokazati kako se spajanjem neuronske mreže i genetskoga algoritma može dobiti poseban algoritam pomoću kojega se mogu pronalaziti putevi.

2. PREGLED PODRUČJA RADA

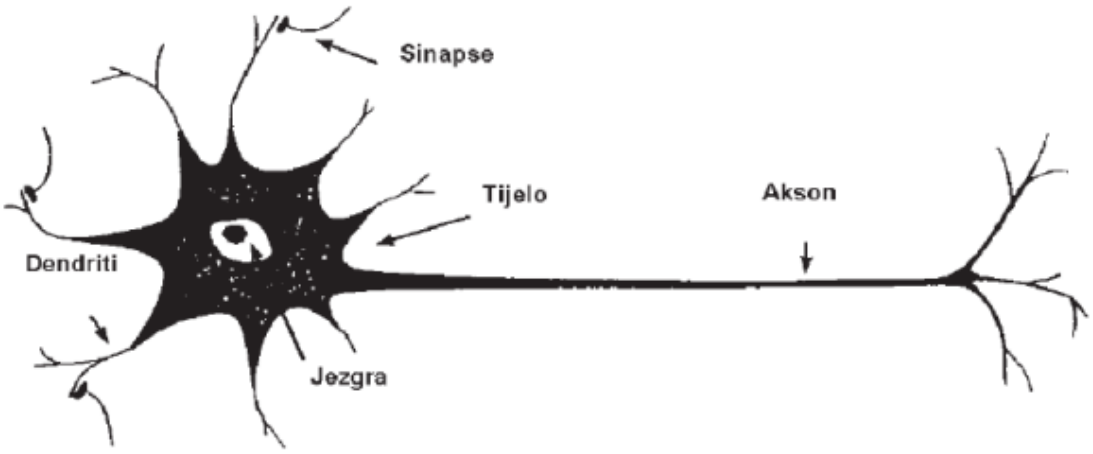
Prema [1], jedan od najvećih izazova u dizajniranju realističnih umjetnih inteligencija u računalnim igrama je kretanje agenata. U današnje vrijeme moderne računalne igre postaju sve dinamičnije te se za njihov razvoj koriste jezgre igara (*engl. game engines*) koje u sebi sadrže već prije implementiranu fiziku. Problem nastaje u tome što se metode pronalaska puteva često rješavaju starijim, dobro definiranim algoritmima za pronalazak puteva koji svoj rad temelje na nepromjenjivim reprezentacijama virtualnoga svijeta unutar računalnih igara. Prema[2], najpoznatiji tradicionalni algoritmi za pronalazak puteva su:

- pretraživanje po dubini (*engl. Depth-first search*)
- pretraživanje po širini (*engl. Breadth-first search*)
- Dijkstrin algoritam
- A*

Tradicionalno su takvi algoritmi za pronalazak puteva bili dovoljni, ali u današnje vrijeme kada se za razvoj koristi jezgra igre, programerima je omogućilo da provedu puno više vremena u stvaranju upečatljivijih računalnih igara s dinamičnim scenama u stvarnom vremenu. Prema [1], zbog tehnološkog razvoja, tradicionalni algoritmi za pronalazak puteva ograničavaju razvoj igre na način da ograničavaju broj dinamičkih objekata koji se mogu koristiti, što znači da će s povećanjem dinamičkih objekata biti potrebna dodatna podešavanja koja sa sobom povlače posljedicu povećanoga vremena razvoja računalne igre. Kako bi se agent mogao učinkovito kretati u dinamičnom virtualnom svijetu, mora imati svijest o okolini koja ga okružuje i to u stvarnome vremenu. Kako bi se to postiglo, tradicionalni algoritmi trebaju se pokretati poslije svakog koraka, odnosno svake promijene na virtualnome svijetu što stvara dodatna opterećenja na procesoru jer zahtjeva puno dodatnog računanja i još za sobom povlači potrebu za većom količinom memorije. Zbog toga u današnje vrijeme dolazi do sve većeg razvoja novih algoritama za pronalaženje puteva, a jedan od njih je kombinacija neuronske mreže i genetskoga algoritma. Prema [2], takav algoritam ima sposobnost generaliziranja što uvelike pomaže kod dinamičkih scena te agenti tog algoritma imaju senzore koji predstavljaju svijest o okolini koja je potrebna za njihovo učinkovito kretanje.

3. NEURONSKA MREŽA

Pojam umjetne neuronske mreže dolazi iz područja umjetne inteligencije te se temeljni na biološkoj neuronskoj mreži. Prema [3], umjetne neuronske mreže su apstraktne simulacije realnog neuronskog sustava i alternativni pristup klasičnom von Neumannovom računalu. Glavni smisao neuronske mreže je oponašanje rada bioloških neurona u mozgu te na taj način razvijanje primjerene strategije analize podataka. Neuron u mozgu je povezan sa 10^4 drugih neurona i to putem sinapsi. Kako bi biološki neuroni mogli međusobno komunicirati odnosno prenositi informacije, koriste dendrite koji primaju ulazne signale od susjednih neurona. Nakon što primi ulazni signal, biološki neuron ih modificira te ih prenosi na druge neurone putem aksona koji su sa svojim završecima povezani s dendritima drugih neurona. Sinapsa predstavlja razmak između završetka aksona prethodnog neurona i dendrita. Prikaz pojednostavljene građe biološkog neurona (Slika 3.1.)



Sl 3.1. Biološka neuronska mreža[3, str.457]

Prema [4], neuroni sa svojom jednostavnom građom (zanemarujući način realizacije navedenih funkcija na biološkoj razini) i činjenicom da se vrijeme generiranja i prijema impulsa mjeri u mikrosekundama, ljudski mozak kao organizirana jedinstvena mreža neurona postiže nevjerojatnu brzinu i sposobnost realiziranja raznovrsnih zadataka zahvaljujući paralelnom radu neurona. Prema tab 3.1. vidljivo je na koji način biološki neuroni predstavljaju alternativni pristup von Neumannovoj arhitekturi računala te prednosti i mane takvoga.

Tab 3.1. Usporedba von Neumannovog računala i biološkog neuronskog sustava

	Von Neumannovo računalo	Biološki neuronski sustav
Procesor	složen velika brzina jedan ili nekoliko	jednostavan mala brzina veliki broj
Memorija	odvojeno od procesora lokalizirana	integrirana u procesoru distribuirana
Računanje	centralizirano sekvencijalno pohranjeni programi	distribuirano paralelno sposobnost učenja
Pouzdanost	vrlo osjetljiva	velika
Ekspertiza	numeričke i simboličke manipulacije	problemi percepcije
Radna okolina	dobro određena	slabo određena

Iako je model neuronske mreže izrazito star te potječe još iz 50-ih godina prošloga stoljeća, tek je u posljednjem desetljeću stekao popularnost. Do popularnosti dolazi zbog nekoliko čimbenika, a to su:

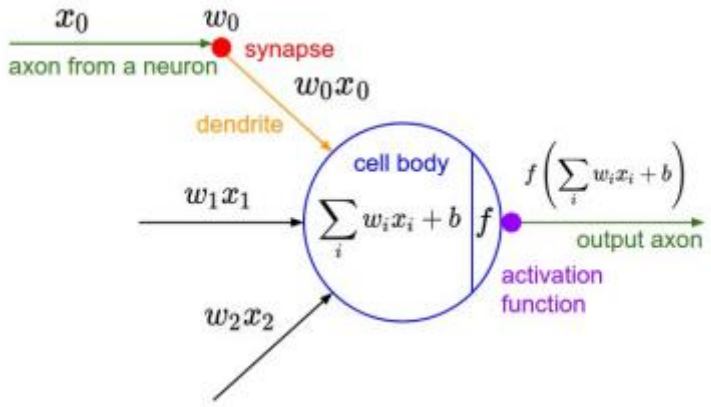
- veliko otkriće 1986. g. - uvođenje algoritma s povratnim rasprostiranjem pogreške (engl. *backpropagation algorithm*)
- razvoj grafičkih kartica koje omogućuju brzi i kompleksan rad s matricama koje se najviše koriste u neuronskim mrežama
- pristup velikoj količini podataka potrebnih za treniranje neuronske mreže.

Danas neuronske mreže nalaze široku primjenu u tehničkim i društvenim znanostima. „Uspjesi koje su neuronske mreže postigle u modeliranju različitih oblika ljudskog ponašanja navele su neke autore na pomisao da su ti modeli svemoćni, tj. da predstavljaju sveobuhvatnu teoriju o kognitivnom funkcioniranju“ [5].

3.1. Princip rada neuronske mreže

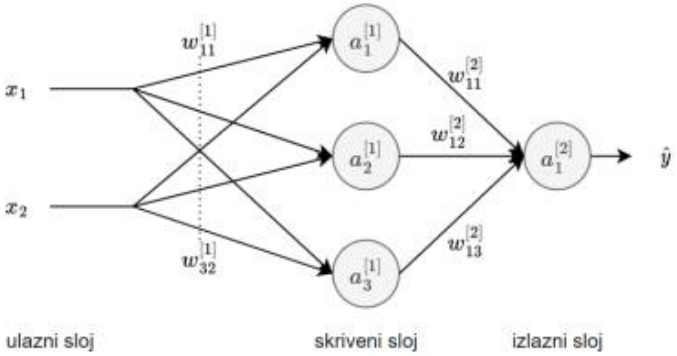
Za shvaćanje principa rada neuronske mreže, prvo se treba proučiti kako radi jedan umjetni neuron. Prema slici 3.2. vidi se korelacija između umjetnog i biološkog neurona. Iz predviđenoga se vidi

kako signali koji putuju aksonima (npr. x_0) dolaze u interakciju s dendritima drugog neurona na način da se množe s težinama sinapse. Težine sinapsi w_i predstavljaju parametre modela koje mreža može sama naučiti. Svaki neuron računa sumu umnožaka svih ulaznih signala x_i s odgovarajućim težinama w_i, uzimajući u obzir pomak b. Na ovu se vrijednost primjenjuje aktivacijska funkcija koja ovisno o vrsti aktivacijske funkcije dodatno modificira ulazne vrijednosti te prosljeđuje dobivenu vrijednost k slijedećem neuronu.



Sl 3.2. Matematički model umjetnog neurona [6, str.9]

Više neurona povezuju se u neuronsku mrežu. Prema [7], jedna od osnovnih umjetnih neuronskih mreža je unaprijedna višeslojna neuronska mreža (engl. *feedforward multilayer neural network*) gdje se propagacija signala u mreži odvija u jednom smjeru, od ulaza prema izlazu, bez povratnih veza. Slojevi takve mreže nižu se jedna iza druge pri čemu je svaki neuron sloja povezan sa svim neuronima prethodnoga sloja. Takva mreža se naziva potpuno povezana neuronska mreža. Na slici 3.3 prikazana je jedna takva potpuno povezana mreža s dvije ulazne veličine, jednim skrivenim slojem i jednim izlaznim slojem.

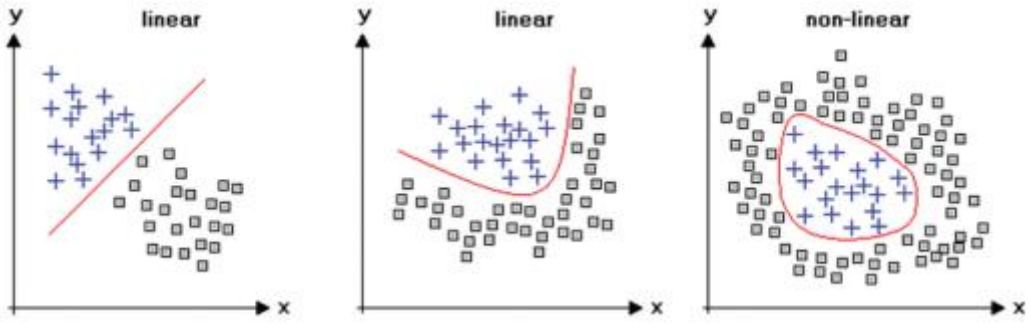


Sl 3.3. Unaprijedna višeslojna neuronska mreža [7, str.69]

Ova mreža u prvom skrivenom sloju ima 3 neurona i 1 neuron u izlaznom sloju. Kako bi se neuronska mreža mogla zvati dubokom, mora sadržavati barem dva skrivena sloja. Ulazne veličine se propuštaju do prvoga skrivenoga sloja gdje se izvode prije objašnjene transformacije i prosljeđuju se u izlazni sloj koji onda još jednom izvodi te transformacije i daje konačnu vrijednost. Raspon izlaznih vrijednosti ovisi o tome koja se aktivacijska funkcija koristila u izlaznome sloju. Kako bi mreža mogla „učiti“ (podešavati parametre poput težina i pomaka), uvodi se ciljna funkcija (engl. *objective function*). Ciljne funkcije koje se temelje na gubitku odnosno smanjivanju toga gubitka zovu se funkcije gubitka (engl. *loss function*) te se takve funkcije i najčešće koriste kod neuronskih mreža. Kako bi neuronska mreža radila, potreban je i optimizator odnosno funkcija koja pomiče parametre prema optimalnima. Bitno je razlikovati ciljne funkcije od optimizatora, ciljne funkcije daju brojčanu vrijednost ili „ocjenu“ koliko su određeni parametri dobri, a prema tim vrijednostima optimizator pokušava dobiti takve vrijednosti parametara koje će kvalitetnije i točnije rješavati zadane probleme. Brzina učenja ovisi o još jednom parametru, a to je stopa učenja α (engl. *learning rate*) koja opisuje koliko će brzo kriterijska funkcija konvergirati optimalnom rješenju. Kako bi mreža kvalitetno učila, stopa učenja ne smije biti premala jer će onda jako sporo konvergirati, a također ne smije biti prevelika jer postoji mogućnost preskakanja optimalnih rješenja. Uobičajene vrijednosti za stopu učenja su 0.01, 0.001 i sl.

3.2. Aktivacijske funkcije

Aktivacijske funkcije su funkcije koje se izvršavaju nad sumom umnožaka svih ulaznih signala s odgovarajućim težinama uzimajući u obzir pomak. Aktivacijske funkcije se koriste kako bi neuronska mreža bila nelinearna. Prema [8], kada bi postojali isključivo linearni izlazi iz svakoga sloja neuronske mreže, svi slojevi bi se mogli zamijeniti jednim slojem zbog nedostatka nelinearnosti. Na slici 3.4. vidi se primjer linearnosti i nelinearnosti statističkih modela.

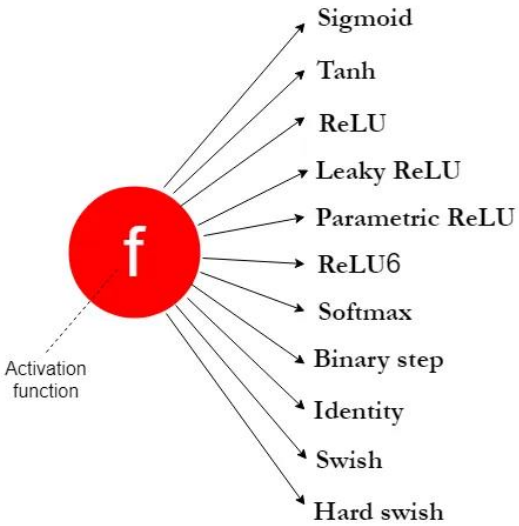


Sl 3.4. Linearni i nelinearni modeli [8, str.26]

Prema [8], bez nelinearnosti nemoguće je napraviti kompleksniji model klasifikacije te bi se neuronska mreža zapravo mogla svesti na logističku regresiju. Aktivacijske funkcije se međusobno razlikuju po matematičkim operacijama koje izvršavaju nad datim podacima. Najčešće korištene aktivacijske funkcije su:

- Sigmoid
- Tanh
- ReLu
- Leaky ReLu.

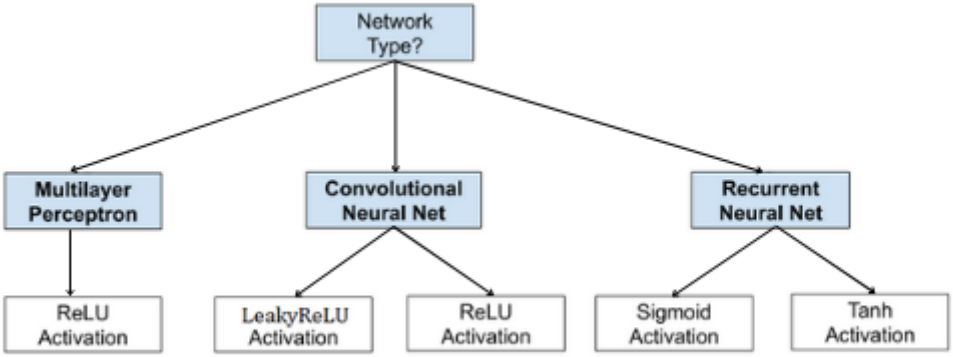
Na slici 3.5. može se vidjeti niz drugih aktivacijskih funkcija osim onih najčešće korištenih.



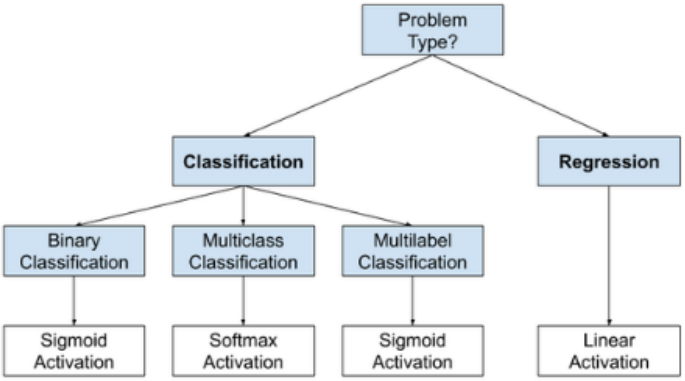
Sl 3.5. Različite vrste aktivacijskih funkcija [9]

Kako su već prije navedene, aktivacijske funkcije uvode nelinearnost u mrežu što znači da bi skriveni slojevi trebali koristiti nelinearne aktivacijske funkcije. Za razliku od skrivenih slojeva, izlazni sloj može koristiti i linearne aktivacijske funkcije ovisno o zadanom problemu. Linearne aktivacijske funkcije se koriste ako postoji regresijski problem. Na slici 3.6. vidi se opći prikaz

odabira aktivacijske funkcije u skrivenom sloju, a na slici 3.7. opći prikaz odabira aktivacijske funkcije u izlaznom sloju.

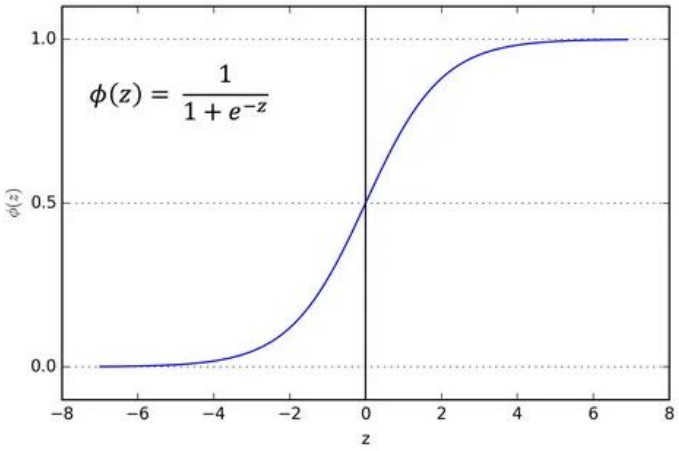


Sl 3.6. Aktivacijske funkcije u skrivenome sloju [10]



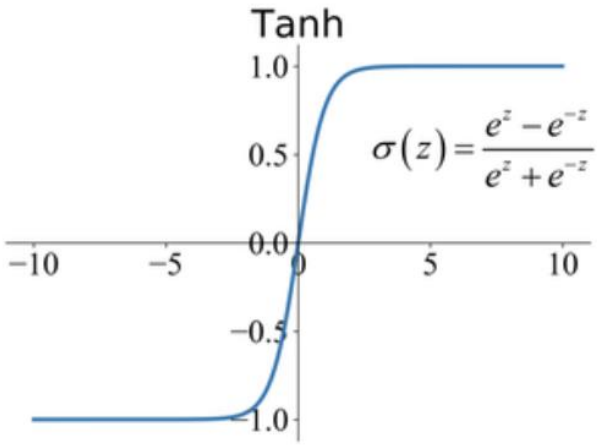
Sl 3.7. Aktivacijske funkcije u izlaznome sloju [10]

Sigmoid aktivacijska funkcija pruža svojstvo nelinearnosti te joj je raspon vrijednosti od 0 do 1. (Slika 3.8.)



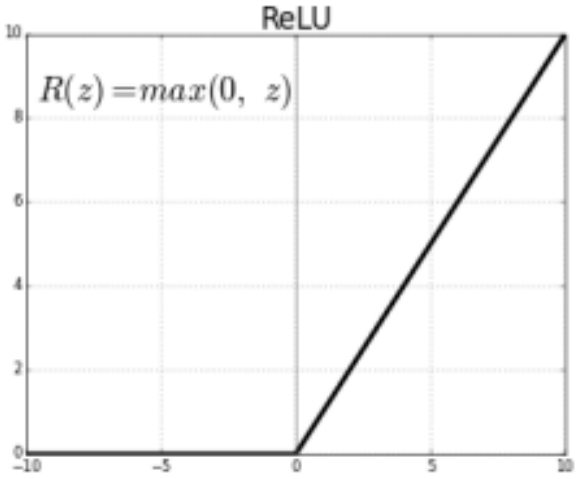
Sl 3.8. Sigmoid funkcija [11, str.7]

Tanh aktivacijska funkcija je izgledom vrlo slična Sigmoid funkciji, ali s rasponom od -1 do 1. (Slika 3.9.)



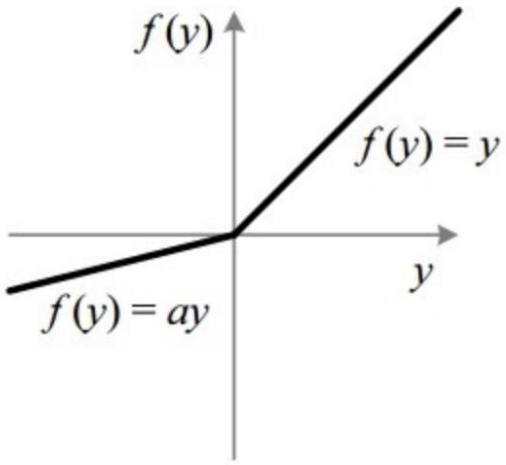
Sl 3.9. Tanh funkcija [12, str.3]

ReLu aktivacijska funkcija je naizgled vrlo jednostavna, ali se pokazala jako korisnom u praksi zbog svoje nelinearnosti i gradijenta 1 ili 0. (Slika 3.10.)



Sl 3.10. ReLu funkcija [13, str.30]

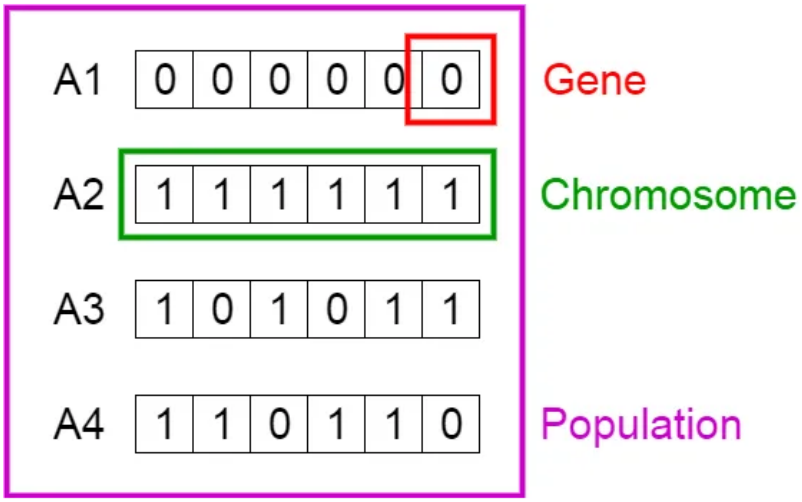
Leaky ReLu funkcija nastaje zbog problema umirućih ReLu neurona. Ako dođe negativna vrijednost do ReLu aktivacijske funkcije, dobiva se gradijent 0 što znači da se težina i pomak neurona neće promijeniti te takvi neuroni mogu biti beskonačno u tome stanju.



Sl 3.11. Leaky ReLu funkcija [14]

4. GENETSKI ALGORITAM

Genetski algoritam je jedna od najpopularnijih implementacija evolucijskih algoritama koja je inspirirana teorijom prirodne evolucije Charlesa Darwina. Ovaj algoritam se temelji na procesu prirodne selekcije gdje se odabiru najspremniji pojedinci za reprodukciju kako bi proizveli potomstvo sljedeće generacije. Princip rada je sljedeći. Postoji populacija jedinki. Svaka jedinka predstavlja jedno moguće rješenje zadanog problema. Jedinke se nazivaju i kromosomima (engl. *chromosome*) (Slika 4.1.). Kromosom je skup parametara koji je nazvan gen (engl. *gene*) (Slika 4.1.). Skup kromosoma zove se populacija (engl. *population*) (Slika 4.1.).



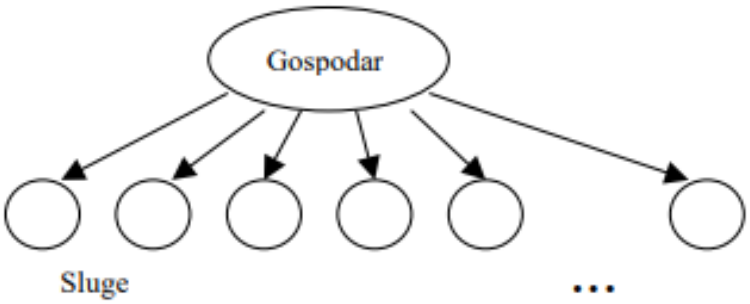
Sl 4.1. Primjer populacije, kromosoma i gena [15]

Svakoj jedinki se može odrediti njezina dobrota (engl. *fitness*). Dobrota jedinke određuje koliko je ta jedinka u „formi“ (sposobnost jedinke da se natječe s drugim jedinkama). Operatorom selekcije (engl. *selection*) biraju se jedinke iz populacije koje postaju roditelji (engl. *parents*). Roditelji pomoću operatora križanja (engl. *crossover*) stvaraju djecu (engl. *offsprings*), a taj proces simulira izmjenu genetskoga materijala. Nad djecom potom djeluje operator mutacije (engl. *mutation*). Na kraju, operatorom zamjene (engl. *reinsertion*) djeca ulaze u populaciju rješenja, čime se zatvara ciklus rada algoritma. Algoritam se ponavlja sve dok se ne pronađe jedinka s najboljom „formom“ ili dok se ne stigne do zadovoljavajućeg rješenja.

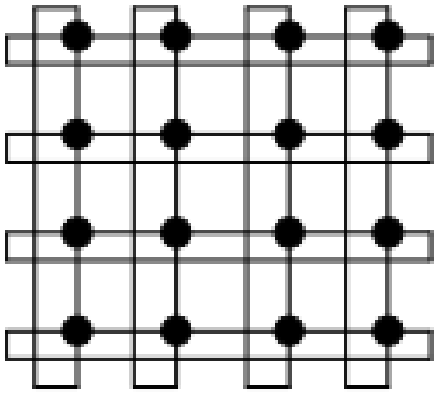
4.1. Vrste genetskoga algoritma

Genetski algoritmi dijele se na paralelne i sekvencijske. Paralelni genetski algoritmi koriste se pristupom „podijeli pa vladaj“. Neke metode paralelizacije koriste jednu populaciju, dok drugi dijele populaciju u nekoliko relativno izoliranih manjih populacija (engl. *subpopulations*). Neke metode mogu iskoristiti masovne paralelno računalne arhitekture, dok su druge prikladnije za više računala s manje snažnijih elemenata za obradu. Prema [16], postoje tri glavna tipa paralelnih genetskih algoritama, a to su:

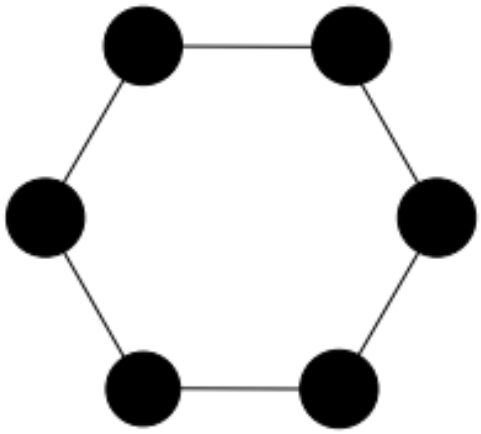
- globalni paralelni genetski algoritam (engl. *master-slave genetic algorithm*) – Postoji jedna slučajna populacija, ali je procjena dobrote raspoređena između nekoliko procesora odnosno sluga koji izvršavaju procjenu dobrote na jedinkama (Slika 4.2.).
- sitnozrnati genetski algoritam (engl. *fine-grained genetic algorithm*) – Prema [17], prikladan za masivna paralelna računala koja se sastoje od jedne prostorno strukturirane populacije. Odabir i reprodukcija ograničeni su na malo susjedstvo, ali susjedstva se preklapaju dopuštajući određenu interakciju među svim jedinkama (Slika 4.3.).
- višepopulacijski genetski algoritam (engl. *multiple-deme parallel genetic algorithm*) – Sofisticiraniji jer se sastoji od nekoliko manjih populacija koje povremeno razmjenjuju jedinke (Slika 4.4.). Ovakva razmjena jedinki naziva se migracija i kontrolira se pomoću nekoliko parametara.



Sl 4.2. Globalni paralelni genetski algoritam [16, str.17]



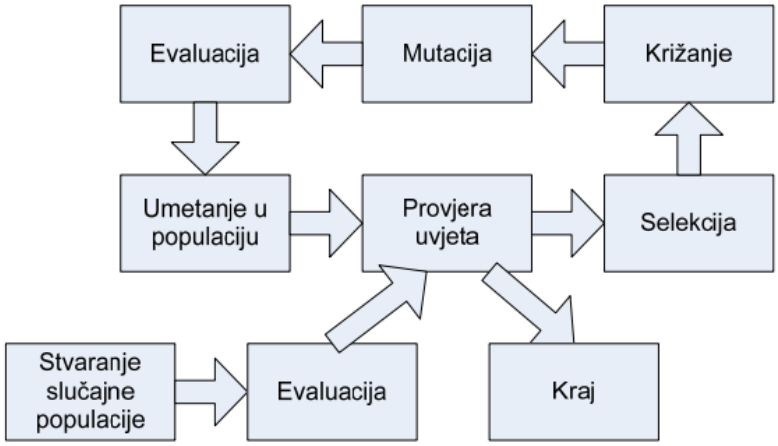
Sl 4.3. Sitnozrnati genetski algoritam [16, str.16]



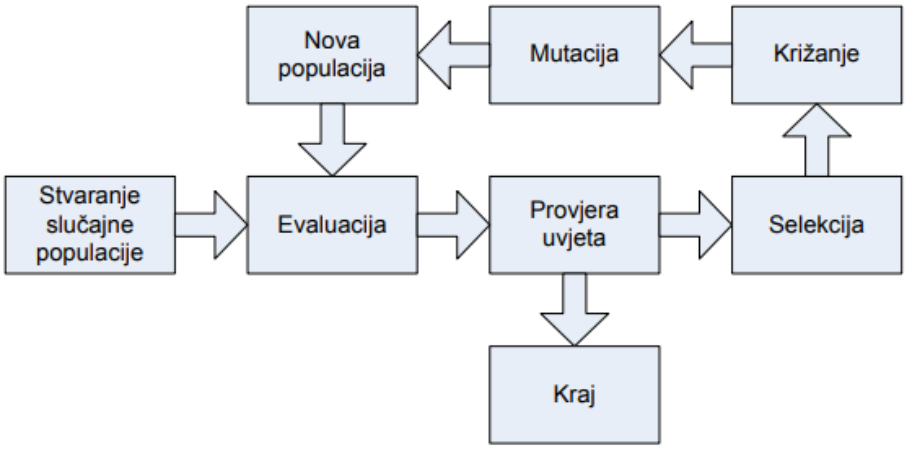
Sl 4.4. Višepopulacijski genetski algoritam[17]

Prema [18], sekvencijski genetski algoritam može se podijeliti na dvije tipične izvedbe:

- steady-state genetski algoritam - U svakoj generaciji iz čitave se populacije odabiru dva roditelja nad kojima se izvodi križanje, čime nastaje novo dijete. Dijete potom mutira i ubacuje se u populaciju. Kako veličina populacije mora biti stalna, ovo ubacivanje se izvodi tako da dijete zamijeni neku jedinku iz populacije (Slika 4.5.).
- generacijski genetski algoritam – Algoritam koji iz generacije u generaciju iz populacije roditelja stvara novu populaciju djece koja potom postaju roditelji s time da odmah nakon toga, stara generacija roditelja izumire (Slika 4.6.).



Sl 4.5. Steady-state genetski algoritam [18, str.8]

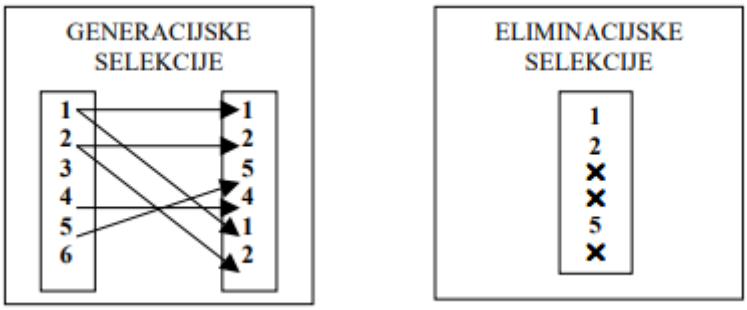


Sl 4.6. Generacijski genetski algoritam [18, str.9]

4.2. Operator selekcije

Operator selekcije je mehanizam za odabir jedinki koje će sudjelovati u reprodukciji. Selekcijom se omogućava prijenos boljeg genetskog materijala (kromosoma) iz generacije u generaciju. Za odabir pravilne selekcije prvo treba odabrati vrstu selekcije prema tome kako će se prenositi genetski materijal. To znači da se prema načinu prijenosa genetskoga materijala jedinki u buduće generacije dijele na (Slika 4.7.):

- generacijske selekcije
- eliminacijske selekcije.



Sl 4.7. Podjela selekcije prema načinu prenošenja genetskog materijala [16, str.22]

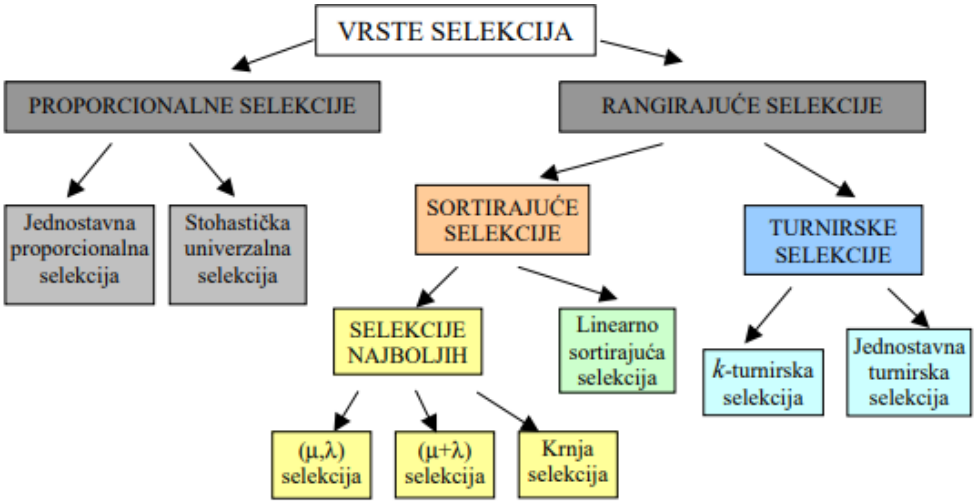
Prema [16], generacijskom selekcijom odabiru se bolje jedinke koje će sudjelovati u reprodukciji. Odabiru se bolje jedinke iz populacije prethodne generacije te se kopiraju u novu populaciju. Ta nova populacija služi za reprodukciju te se naziva međupopulacijom. Pomoću međupopulacije dobiva se populacija koja predstavlja novu generaciju. Glavni problemi kod ovakve selekcije su:

- postojanje dviju populacija jedinki u radnoj memoriji što kod jako velikih populacija može stvarati memorijske probleme
- broj jedinki koje prežive selekciju manji je od veličine populacije te se ta razlika mora nadoknaditi reprodukcijom odnosno korištenjem jednog od operatora križanja koji će biti objašnjeni u sljedećem potpoglavlju. Odabirom lošeg operatora križanja može dovesti do dupliciranih jedinki što ne doprinosi kvaliteti dobivenoga rješenja, već samo usporava evolucijski proces.

Prema [16], kod eliminacijske selekcije bolje jedinke preživljavaju mnoge generacije za razliku od generacijske selekcije gdje jedinke žive samo jednu generaciju osim ako se ne primjeni elitizam (tehnika u kojoj se najbolje jedinke preslikavaju u novu populaciju, a ostatak se dobiva već prije objašnjenim načinom). Zbog duljeg životnoga vijeka jedinki , u eliminacijskoj selekciji ne postoji stroga granica između generacija. Eliminacijska selekcija briše M jedinki te ih nadomještava reprodukcijom. Za razliku od generacijske selekcije, eliminacijska selekcija izbjegne problem postojanja dvije populacije, ali postoji i problem odabira lošeg operatora križanja što dovodi do prije objašnjenoga problema. Prema [16], nakon što je odabrana vrsta selekcije prema načinu prijenosa genetskoga materijala, potrebno je odabrati selekciju prema određivanju vrijednosti vjerojatnosti odabira određene jedinke, a to su (Slika 4.8.):

- proporcionalne selekcije – Prema [16], odabiru jedinke s vjerojatnošću koja je proporcionalna dobroti jedinke. To znači da vjerojatnost selekcije ovisi o kvocijentu dobrote jedinke i prosječne dobrote populacije. Prema tab 4.1. vidljivo je kako se vjerojatnost selekcije proporcionalno povećava s vrijednošću njihove dobrote.

- rangirajuće selekcije - Prema [16], odabiru jedinke s vjerojatnošću koja ovisi o položaju jedinke u poretku jedinki sortiranih po dobroti. Prednost ove selekcije je ta što ukoliko postoji više jedinki podjednakih vrijednosti dobrote, dolazi se do gubitka selekcijskog pritiska prema prikladnijim pojedincima. Ova selekcija rangira jedinke ovisno o njihovoj dobroti te za svaki rang daje konstantnu vrijednost neovisno o njihovom međusobnom odnosu dobrote. Prema tab 4.2. vidi se kako je vjerojatnost selekcije konstantna te je neovisna o odnosima dobrote od ostalih jedinki.



Sl 4.8. Podjela selekcija prema određivanju vrijednosti vjerojatnosti odabira određene jedinke [16, str.23]

Tab 4.1. Primjer proporcionalne selekcije

Jedinka	1	2	3	4	5	6	7	8
Dobrota jedinke	6	4	3.6	0.4	0.8	1.2	2.4	1.6
Vjerojatnost odabira	0.3	0.2	0.18	0.02	0.04	0.06	0.12	0.08

Tab 4.2. Primjer rangirajuće selekcije

Jedinka	1	2	3	4	5	6
Dobrota jedinke	6	5.8	4.9	6.1	5.75	6.43
Rang	3	4	6	2	5	1
Vjerojatnost odabira	0.2	0.15	0.05	0.2	0.1	0.3

4.3. Operator križanja

Prema [19], operator križanja je mehanizam kojim se omogućuje prenošenje svojstva jedinke s roditelja na djecu. Križanje je postupak izmjene gena između roditelja. Temelji se na tome da ako su roditelji dobri (prošli su proces selekcije), tada će najvjerojatnije i dijete biti dobro, ako ne i bolje od svojih roditelja. Križanjem nastaje jedno ili dvoje djece. Operator križanja dijeli se na (Slika 4.9.):

- uniformno – Uniformno križanje se može smatrati kao križanje s k-1 točkom prekida (gdje je k broj bitova gena). Kromosom svakoga roditelja će se podijeliti na gene te svako dijete bira koji će gen od roditelja uzeti. Odabir gena je najčešće slučajan odnosno vjerojatnost da uzme gen prvog ili drugog roditelja je jednaka, a iznosi 50%. Dijete se dobiva prema aritmetičkoj operaciji:

$$Dijete = A * B + R * (A\ XOR\ B) \tag{4-1}$$

  Gdje je:
  - A – geni prvog roditelja
  - B – geni drugog roditelja
  - R – nasumično generirani vektor
  - Dijete – geni djeteta

- s n-točaka prekida – Križanje s n-točaka prekida je križanje kako samo ime kaže koje ima n prekida (n je proizvoljan cijeli broj koji je manji od broja gena u kromosomu). Slučajnim se odabirom uzima n točaka gdje se kidaju oba kromosoma te se ti dijelovi kromosoma unakrsno povezuju kako bi se dobila djeca (Slika 4.10.).
- aritmetičko – Prema [20], aritmetičko križanje se koristi kod prikaza rješenja u obliku realnih brojeva. Djeca se stvaraju prema sljedećim izrazima:

$$Dijete_1 = a * roditelj_1 + (1 - a) * roditelj_2 \tag{4-2}$$

$$Dijete_2 = (1 - a) * roditelj_1 + a * roditelj_2 \tag{4-3}$$

  Gdje je:
  - $Dijete_i$ – geni i–tog djeteta
  - $roditelj_i$ – geni i-tog roditelja
  - a – slučajni težinski koeficijent iz [0, 1]

- heurističko – Slično kao i kod aritmetičkog križanja, heurističko križanje se koristi pri prikazu rješenja u obliku realnih brojeva. Prema [20], kod heurističkog križanja koristi se vrijednost funkcije dobrote dvaju roditelja kako bi se generiralo dijete. Djeca se generiraju temeljem sljedećih izraza:

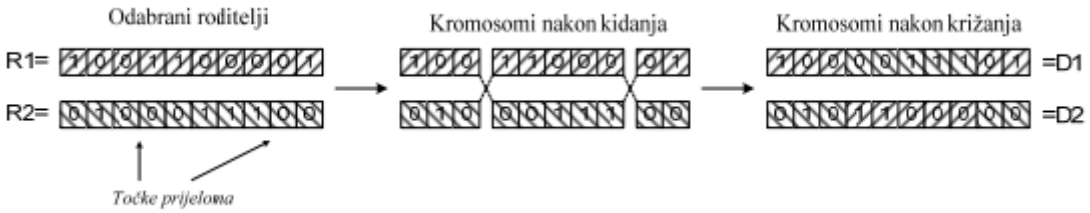
$$Dijete_1 = bolji\ roditelj + r * (bolji\ roditelj - lošiji\ roditelj) \quad (4\text{-}4)$$

$$Dijete_2 = bolji\ roditelj \quad (4\text{-}5)$$

Gdje je:

- $bolji\ roditelj$ – geni roditelja s većom funkcijom dobrote
- $lošiji\ roditelj$ – geni roditelja s manjom funkcijom dobrote
- r – slučajni težinski koeficijent iz [0, 1]

Sl 4.9. Vrste križanja [20, str.35]



Sl 4.10. Primjer križanja s 2 točke prekida [18, str.13]

4.4. Operator mutacije

Operator mutacije utječe na promjenu gena u kromosomu. Mutacija djeluje nad jednom jedinkom te rezultat mutacije je promijenjena jedinka. Parametar koji određuje vjerojatnost mutacije je P_m te on predstavlja vjerojatnost da pojedini gen mutira. Prema [19], svrha mutacije je bolje istražiti prostor rješenja te time izbjeći lokalne minimume odnosno maksimume. Koristi se zato što

genetski algoritam često nailazi na problem prerane konvergencije, što znači da ako se reprodukcijom kreiraju jedinke koje su slične prijašnjim, vrlo brzo cijela populacija može zapeti u lokalnom optimumu te se nikako ne bi moglo izaći iz njega. Mutacija sprječava jedinke da postanu previše slične te se na taj način dobivaju jedinke koje zaobilaze lokalne optimume. Vrste mutacija su:

- jednostavna – Jednostavna mutacija je oblik mutacije pri kojem se svaki gen mijenja s jednakom vjerojatnošću.
- potpuna – Prema [19], potpuna mutacija je poseban oblik miješajuće mutacije pri kojoj su gornja i donja granica intervala rubovi kromosoma što uzrokuje miješanje svih gena u kromosomu. U ovom slučaju broj nula i jedinica ostaju nepromijenjeni.
- potpuna miješajuća – Potpuna miješajuća mutacija je oblik miješajuće mutacije pri kojoj se svi bitovi unutar zadanoga intervala slučajno generiraju.
- invertirajuća miješajuća – Invertirajuća miješajuća mutacija je oblik miješajuće mutacije u kojoj se invertiraju svi bitovi u zadanome intervalu.

Prema tab 4.3. vidi se primjer različitih vrsta mutacija nad kromosomom: 110110001011 pri korištenju slučajno generirane maske (maska ukazuje nad kojim će se bitovima izvršiti mutacija) te maske nad svim genima 111111111111.

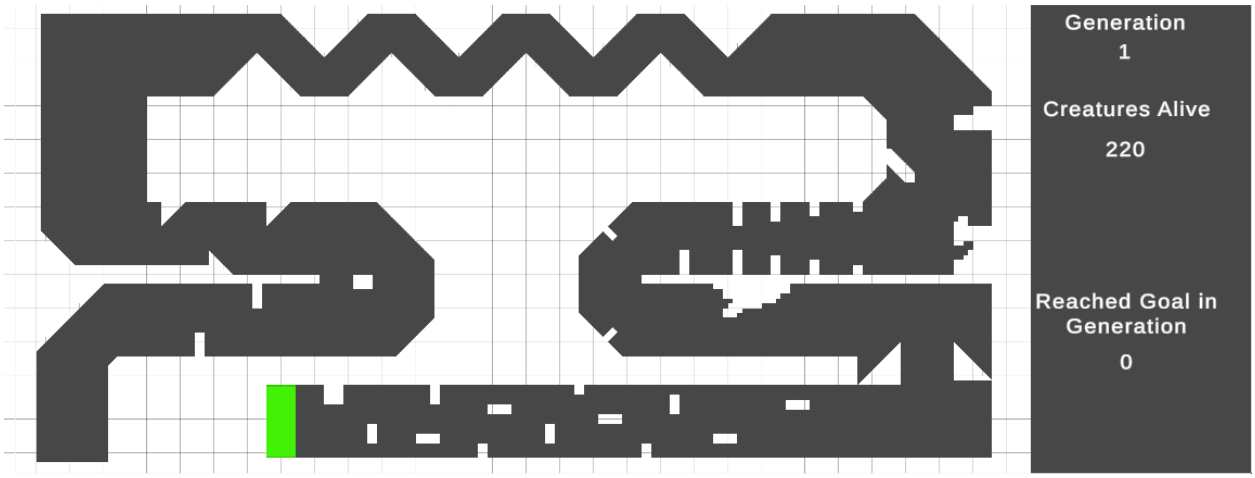
Tab 4.3. Primjer različitih vrsta mutacija

vrsta mutacije	slučajno odabran jedan gen npr. peti	slučajno generirana maska: 001001111000	svi geni
jednostavna mutacija	1101**0**0001011	111**1**11**110**011	**001001110100** (potpuna inverzija)
miješajuća mutacija	-	11**0**11**0100**011	**011110100101** (broj jedinica i nula ostaje isti)
potpuna miješajuća mutacija	-	111**1**1**0011**011	**011101011010** (potpuno slučajni kromosom)
invertirajuća miješajuća mutacija	-	111**1**11**110**011	**001001110100** (potpuna inverzija)

5. PROGRAMSKA IMPLEMENTACIJA ALGORITMA

5.1. Opis programa

Kao dio ovoga završnoga rada izrađen je program zvan Neural network with genetic algorithm. Ovaj algoritam objedinjuje neuronsku mrežu i genetski algoritam u poseban algoritam koji se koristi za pronalazak puteva. Program je razvijen u Unityu koji koristi programski jezik C# te je u obliku 2D igre koja pruža cjeloviti pregled događanja prilikom učenja neuronske mreže (Slika 5.1.). Program odnosno sami algoritam za pronalazak puteva radi na raznim 2D stazama ali zbog jednostavnosti prilikom implementacije i zbog samoga shvaćanja principa rada koristi se samo jedna staza (Slika 5.1.).



Sl 5.1. Grafičko sučelje pokrenutoga programa

Sučelje programa sastoji se od trake za učenje na kojoj se u stvarnom vremenu mogu pratiti trenutna stanja stvorenja ili agenata (jedinke koje predstavljaju populaciju) i informacijske trake na kojoj se nalaze neke od osnovnih informacija kako bi se olakšalo praćenje učenja algoritma. Traka za učenje sadrži:

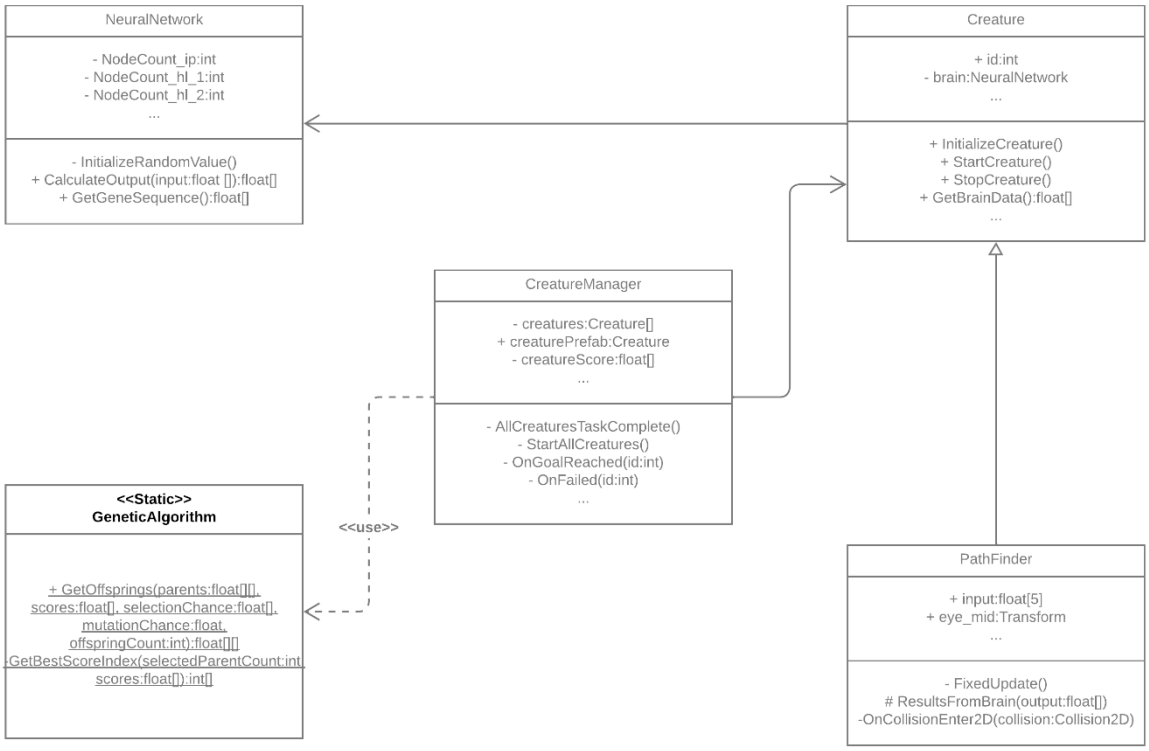
- agente
- stazu na kojoj mreža uči put
- cilj (mjesto do kojega agenti uče doći).

Informacijska traka sadrži:

- broj trenutne generacije koja se izvodi
- broj aktivnih agenata
- broj generacije kada je prvi puta dostignut cilj.

5.2. Opis programskoga koda

Programski kod algoritma za pronalaženje puteva se sastoji od 5 klasa koje su predstavljene pomoću zasebnih skripti (svaka klasa je predstavljena odnosno implementirana u svojoj skripti). Struktura programa može se popratiti na slici 5.2.



Sl 5.2. Struktura programa

Osnova ovakvog algoritma za pronalaženje puteva je neuronska mreža, a klasa koja ju simulira mora obavljati niz zadataka a neki od najvažnijih su:

- InitializeRandomValues – Metoda koja se poziva prilikom kreiranja objekta. Dodjeljuje slučajne vrijednosti težinama i pomacima u neuronskoj mreži (Slika 5.3.).
- CalculateOutput – Metoda koja uz pomoć niza brojeva koji predstavljaju udaljenost od prepreke a ujedno i ulaz u neuronsku mrežu daje broj snage kojom bi se trebao okrenuti kako bi izbjegao prepreku što predstavlja izlaz iz neuronske mreže (Slika 5.4.).
- GetGeneSequence – Metoda koja vraća sve težine između svih slojeva i pomake na svakom neuronu(Slika 5.5.).

```
private void InitializeRandomValues()
  {
    for(int i = 0; i < nodeCount_hl_1; i++)
    {
      bias_hl_1[i] = UnityEngine.Random.Range(-1f,1f);
    }

    for (int i = 0; i < nodeCount_hl_2; i++)
    {
      bias_hl_2[i] = UnityEngine.Random.Range(-1f, 1f);
    }

    for (int i = 0; i < nodeCount_op; i++)
    {
      bias_op[i] = UnityEngine.Random.Range(-1f, 1f);
    }

    for(int i = 0; i < nodeCount_ip; i++)
    {
      for(int j=0;j< nodeCount_hl_1; j++)
      {
        weight_ip_hl_1[i][j]= UnityEngine.Random.Range(-1f, 1f);
      }
    }

    for (int i = 0; i < nodeCount_hl_1; i++)
    {
      for (int j = 0; j < nodeCount_hl_2; j++)
      {
        weight_hl_1_hl_2[i][j] = UnityEngine.Random.Range(-1f, 1f);
      }
    }

    for (int i = 0; i < nodeCount_hl_2; i++)
    {
      for (int j = 0; j < nodeCount_op; j++)
      {
        weight_hl_2_op[i][j] = UnityEngine.Random.Range(-1f, 1f);
      }
    }
  }
```

Sl 5.3. Metoda InitializeRandomValues

```
public float[] CalculateOutput(float[] input)
  {
    if (input.Length != nodeCount_ip)
      return null;

    float[] firstHiddenLayerValues = new float[nodeCount_hl_1];
    float[] secondHiddenLayerValues = new float[nodeCount_hl_2];
    float[] outputLayer = new float[nodeCount_op];

    for(int i = 0; i < nodeCount_hl_1; i++)
    {
      float summation = 0;
      for(int j = 0; j < nodeCount_ip; j++)
      {
        summation += weight_ip_hl_1[j][i] * input[j];
      }

      summation += bias_hl_1[i];

      firstHiddenLayerValues[i]=ReluActivationFunction(summation);
    }
```

```
        for (int i = 0; i < nodeCount_hl_2; i++)
        {
            float summation = 0;
            for (int j = 0; j < nodeCount_hl_1; j++)
            {
                summation += weight_hl_1_hl_2[j][i] * firstHiddenLayerValues[j];
            }

            summation += bias_hl_2[i];

            secondHiddenLayerValues[i] = ReluActivationFunction(summation);
        }

        for (int i = 0; i < nodeCount_op; i++)
        {
            float summation = 0;
            for (int j = 0; j < nodeCount_hl_2; j++)
            {
                summation += weight_hl_2_op[j][i] * secondHiddenLayerValues[j];
            }

            summation += bias_op[i];

            outputLayer[i] = SigmoidActivationFunction(summation);
        }
        return outputLayer;
    }
```

Sl 5.4. Metoda CalculateOutput

```
public float[] GetGeneSequence()
    {
        float[] geneSequence = new float[nodeCount_ip*nodeCount_hl_1 + bias_hl_1.Length + nodeCount_hl_1*nodeCount_hl_2 +
bias_hl_2.Length + nodeCount_hl_2*nodeCount_op + bias_op.Length];
        int currentIndex = 0;

        for(int i=0;i< nodeCount_ip; i++)
        {
            for(int j = 0; j < nodeCount_hl_1; j++)
            {
                geneSequence[currentIndex] = weight_ip_hl_1[i][j];
                currentIndex++;
            }
        }

        for (int i = 0; i < nodeCount_hl_1; i++)
        {
            geneSequence[currentIndex] = bias_hl_1[i];
            currentIndex++;
        }

        for (int i = 0; i < nodeCount_hl_1; i++)
        {
            for (int j = 0; j < nodeCount_hl_2; j++)
            {
                geneSequence[currentIndex] = weight_hl_1_hl_2[i][j];
                currentIndex++;
            }
        }

        for (int i = 0; i < nodeCount_hl_2; i++)
        {
            geneSequence[currentIndex] = bias_hl_2[i];
            currentIndex++;
```

```
    }

    for (int i = 0; i < nodeCount_hl_2; i++)
    {
        for (int j = 0; j < nodeCount_op; j++)
        {
            geneSequence[currentIndex] = weight_hl_2_op[i][j];
            currentIndex++;
        }
    }

    for (int i = 0; i < nodeCount_op; i++)
    {
        geneSequence[currentIndex] = bias_op[i];
        currentIndex++;
    }
    return geneSequence; }
```

Sl. 5.5. Metoda GetGeneSequence

Kako i samo ime govori, algoritam za pronalazak puteva primjenom genetskoga algoritma treba sadržavati klasu genetski algoritam u kojemu su najvažnije metode:

- GetOffsprings – Metoda koja kreira djecu tako da se slučajnim odabirom uzimaju dva roditelja iz liste najboljih roditelja (Slika 5.6.).
- GetBestScoreIndex – Metoda koja vraća indekse onih jedinki iz populacije koje su imale najveću dobrotu (Slika 5.7.).

//

```
public static float[][] GetOffsprings(float[][] parents, float[] scores, float[] selectionChance, float mutationChance, int offspringCount)
  {
    int selectedParentCount = selectionChance.Length;

    int[] bestParentIndexes = GetBestScoreIndex(selectedParentCount, scores);
    float[][] bestParents = new float[selectedParentCount][];

    // Selection of best parents for offspring creation
    for(int i=0;i<selectedParentCount;i++)
    {
        bestParents[i] = parents[bestParentIndexes[i]];
    }

    // Summation of selectionChances
    float total = 0;
    for (int i = 0; i < selectedParentCount; i++)
    {
        total += selectionChance[i];
    }

    float[][] offsprings = new float[offspringCount][];
    int counter;

    // Add best parents into next generation
    for(counter=0;counter<selectedParentCount; counter++)
    {
        offsprings[counter] = bestParents[counter];
    }

    // Create offsprings
    for(int i= counter; i<offspringCount;i++)
```

```
    {
        //rand is used for randomly choosing parents
        float rand = UnityEngine.Random.Range(0f, total);
        float selectionPoint = selectionChance[0];

        float[] parent1 = bestParents[0];
        int parent1Index = 0;

        //find first parent
        for(int j=0;j<selectedParentCount;j++)
        {
            //If you found parent or he is a last parent
            if (rand <= selectionPoint || j == selectedParentCount - 1)
            {
                parent1 = bestParents[j];
                parent1Index = j;
                break;
            }
            else
                selectionPoint += selectionChance[j + 1];
        }

        //find second parent
        rand = UnityEngine.Random.Range(0f, total);
        selectionPoint = selectionChance[0];
        float[] parent2 = bestParents[0];

        for (int j = 0; j < selectedParentCount; j++)
        {
            if (j == parent1Index && j != selectedParentCount - 1)
                continue;
            else if (rand <= selectionPoint || j == selectedParentCount - 1)
            {
                parent2 = bestParents[j];
                break;
            }
            else
                selectionPoint += selectionChance[j + 1];
        }

        //create chromosome for offspring
        float[] offspring = new float[parent1.Length];
        for(int j = 0; j < offspring.Length; j++)
        {
            // Select gene from parent 1 or 2
            offspring[j] = UnityEngine.Random.Range(0f, 1f) > 0.5f ? parent1[j] : parent2[j];

            // Possible mutation
            if (UnityEngine.Random.Range(0f, 1f) < mutationChance)
            {
                offspring[j] = UnityEngine.Random.Range(-1f, 1f);
            }
        }
        offsprings[i] = offspring;
    }
    return offsprings;
}
```

Sl. 5.6. Metoda GetOffsprings

```
private static int[] GetBestScoreIndex(int selectedParentCount, float[] scores)
  {
    int[] indices = new int[selectedParentCount];
```

```
    int[] sortedIndices = Enumerable.Range(0, scores.Length)
                          .OrderByDescending(i => scores[i])
                          .ToArray();
    // copies first selectedParentCount elements from sortedIndices to indices
    Array.Copy(sortedIndices, indices, selectedParentCount);
    return indices;
}
```

Sl 5.7. Metoda GetBestScoreIndex

Za rad algoritma odnosno učenje neuronske mreže potrebni su agenti koji sadrže niz funkcionalnosti koje omogućavaju ispravan rad. Zbog velikoga broja funkcionalnosti potrebnih za rad agenata koriste se dvije klase Creature i PathFinder. Klasa Creature nema kompleksnih funkcionalnosti ali zato ima niz jednostavnih poput aktivacije i deaktivacije samog agenta i prosljeđivanje podataka neuronskoj mreži. Klasa PathFinder proširuje funkcionalnosti klase Creature sa metodama za prikupljanje ulaznih podataka u neuronsku mrežu, kretanje agenta i detekcija kolizije (Slika 5.8.).

```
private void FixedUpdate()
{
    if (isRunning)
    {
        input[0] = Physics2D.Raycast(eye_left.position, eye_left.transform.up, maxRange, layerMask: mask).distance;
        input[0] = input[0] == 0 ? maxRange : input[0];

        input[1] = Physics2D.Raycast(eye_midLeft.position, eye_midLeft.transform.up, maxRange, layerMask: mask).distance;
        input[1] = input[1] == 0 ? maxRange : input[1];

        input[2] = Physics2D.Raycast(eye_mid.position, eye_mid.transform.up, maxRange, layerMask: mask).distance;
        input[2] = input[2] == 0 ? maxRange : input[2];

        input[3] = Physics2D.Raycast(eye_midRight.position, eye_midRight.transform.up, maxRange, layerMask: mask).distance;
        input[3] = input[3] == 0 ? maxRange : input[3];

        input[4] = Physics2D.Raycast(eye_right.position, eye_right.transform.up, maxRange, layerMask: mask).distance;
        input[4] = input[4] == 0 ? maxRange : input[4];

        SendInputToBrain(input);
    }
}


protected override void ResultsFromBrain(float[] output)
{
    float turn = output[0] * 2 - 1;
    transform.Rotate(Vector3.forward * turn * steerSpeed * Time.deltaTime);
}


private void OnCollisionEnter2D(Collision2D collision)
{
    if (!isRunning) return;
```

```
    var collidedObjectLayerValue = collision.collider.gameObject.layer;
    if (collidedObjectLayerValue == LayerMask.NameToLayer("Obstacle"))
    {
        StopCreature();
        OnFailed();
        return;
    }

    if (collidedObjectLayerValue == LayerMask.NameToLayer("Goal"))
    {
        StopCreature();
        GoalReached();
    }
}
```

Sl 5.8. Metode za prikupljanje podataka, kretanje agenta i detekcija kolizije

Kako bi mogle sve ranije navedene klase međusobno surađivat, potrebna je dodatna klasa koja predstavlja ulogu voditelja (engl. manager). U ovom radu klasa CreatureManager predstavlja voditelja te sadrži niz važnih metoda poput:

- AllCreaturesTaskComplete – Metoda koja se poziva kada više nema aktivnih agenata. Radi tako što dohvaća kromosome svih jedinki te ih prosljeđuje genetskome algoritmu kako bi se dobile jedinke za sljedeću generaciju (Slika 5.9.).
- StartAllCreatures – Metoda koja postavlja sve agente na početnu poziciju te ih aktivira (Slika 5.10.).
- OnGoalReached – Metoda koja se poziva prilikom dolaska agenta do cilja a služi za dodjelu dobrote agenta (Slika 5.11.).
- OnFailed – Slična metodi OnGoalReached s time da se na nešto drugačiji način dodjeljuje dobrota agentu (Slika 5.11.).

```
private void AllCreaturesTaskComplete()
{
    counter = 0;
    _isRunning = false;
    float[][] parents = new float[numberOfCreatures][];

    if (generation%5==0)
    {
        Debug.Log($"{generation} Generation time passed: {DateTime.Now - start}");
    }

    for(int i=0;i<parents.Length;i++)
    {
        parents[i] = creatures[i].GetBrainData();
    }

    float[][] offsprings = GeneticAlgorithm.GetOffsprings(parents, creatureScore, selectionChance, mutationChance,
numberOfCreatures);

    for (int i=0;i<numberOfCreatures;i++)
    {
```

```
            creatures[i].SetBrainData(offsprings[i]);
        }
        generation++;
        generation_text.text = generation.ToString();
        StartAllCreatures();
    }
```

Sl 5.9. Metoda za dohvaćanje novih agenata

```
private void StartAllCreatures()
    {
        for(int i = 0; i < creatureScore.Length; i++)
        {
            creatureScore[i] = 0;
        }

        foreach(Creature creature in creatures)
        {
            creature.transform.SetPositionAndRotation(startPoint.position, startPoint.rotation);
            creature.StartCreature();
        }

        _isRunning = true;
        numberOfActiveCreatures = creatures.Length;
    }
```

Sl 5.10. Metoda za aktivaciju svih agenata

```
private void OnGoalReached(int id)
    {
        if (!isFinished)
        {
            finish_generation_text.text = generation.ToString();
            Debug.Log($"On finished time passed: {DateTime.Now - start}");
        }

        isFinished = true;
        numberOfActiveCreatures--;
        creatureScore[id] = 5000 - counter;

        if(numberOfActiveCreatures <= 0)
        {
            AllCreaturesTaskComplete();
        }
    }

    private void OnFailed(int id)
    {
        numberOfActiveCreatures--;
        creatureScore[id] = counter;

        if (numberOfActiveCreatures <= 0)
        {
```

```
            AllCreaturesTaskComplete();
        }
    }
```

Sl 5.11. Metode za dostizanje ili ne dostizanje cilja

5.3. Upute za korištenje

Algoritam za pronalaženje puteva je kompleksan algoritam koji se sastoji od niza komponenti, ali od njih se mogu izdvojiti 3 najvažnije komponente, a to su:

- agent – inicijalni agent pomoću kojega se dobivaju ostali agenti koji služe za učenje neuronske mreže
- staza – put koji mreža treba naučiti
- manager – objekt koji se brine o cjelokupnom radu algoritma.

Inicijalni agent predstavlja shemu prema kojoj se kloniraju svi ostali agenti populacije. Kloniranje znači da se svi parametri prenose s originala na kopiju, ali s jednom znatnom razlikom, a to je u njihovim genima. Zbog toga je pravilno postavljanje parametara inicijalnog agenta jako važno jer s loše postavljenim parametrima dolazi do toga da mreža jako slabo uči ili čak uopće ne uči. Prema slici 5.12., vide se najosnovniji parametri koje treba podesiti, a to su brzina kretanja agenata te njihova brzina okretanja.

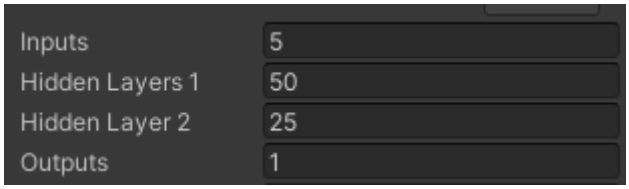
Sl 5.12. Parametri brzine kretanja i okretanje agenata

Problemi koji mogu nastati s lošim postavljanjem ovih parametara su:

- sporo kretanje agenata – Dolazi do toga da se agenti kreću previše sporo te na taj način povećavaju vrijeme između generacija odnosno povećava se vrijeme učenja neuronske mreže. Ovaj se problem prvobitno ne čini toliko velikim jer s povećanjem vremena trajanja jedne generacije za par sekundi se čini malim, ali kroz niz generacija poput 500, 600 ili više, brzo te sekunde postanu minute pa čak i kod nekih kompleksnih staza i sati. Drugi razlog zašto ovaj problem prvobitno nije toliko očit je taj što na početku učenja neuronske mreže razlika između brže i sporije gibajućih agenata samo je sekunda ili dvije, ali s povećanjem naučenosti mreže, agenti će dalje dostizati na stazi te će razlika između sporijih i bržih agenata postajati sve veća.

- brzo kretanje agenata – Dolazi do problema da računalo odnosno procesor neće stići izvršiti svo računanje te mreža neće stići naučiti stazu odnosno njezine prepreke. Problem je lako uočljiv jer će agenti početi zapinjati te se naglo pomicati.
- sporo okretanje agenata – Problem u kojemu agenti zbog previše male brzine okretanja ne stignu zaobići prepreke. Ovakva vrsta problema pruža velike poteškoće jer se ne može odmah zaključiti kako se radi o problemu sporog okretanja, nego se može pomisliti kako mreža još nije naučila zadanu prepreku. Ovisno o tome koliko je brzina okretanja blizu potrebnoj brzini, može doći do toga da će biti potrebno veći broj generacija kako bi se otkrio ovaj problem.
- brzo okretanje agenata – Vrsta problema koja se vrlo često odmah prepozna, a to je zato što se kod ovoga problema agenti počnu okretati u mjestu te na taj način ne pridonose učenju nego ga usporavaju pa čak i zaustavljaju (ovisni o tome kako je implementirano zaustavljanje trenutne generacije te započinjanje sljedeće).

Nakon određivanja brzine kretanja i brzine okretanja agenata, sljedeći parametri koje treba odrediti su parametri neuronske mreže, odnosno koliko će neurona svaki sloj neuronske mreže imati (Slika 5.13.)



Sl 5.13. Parametri neuronske mreže

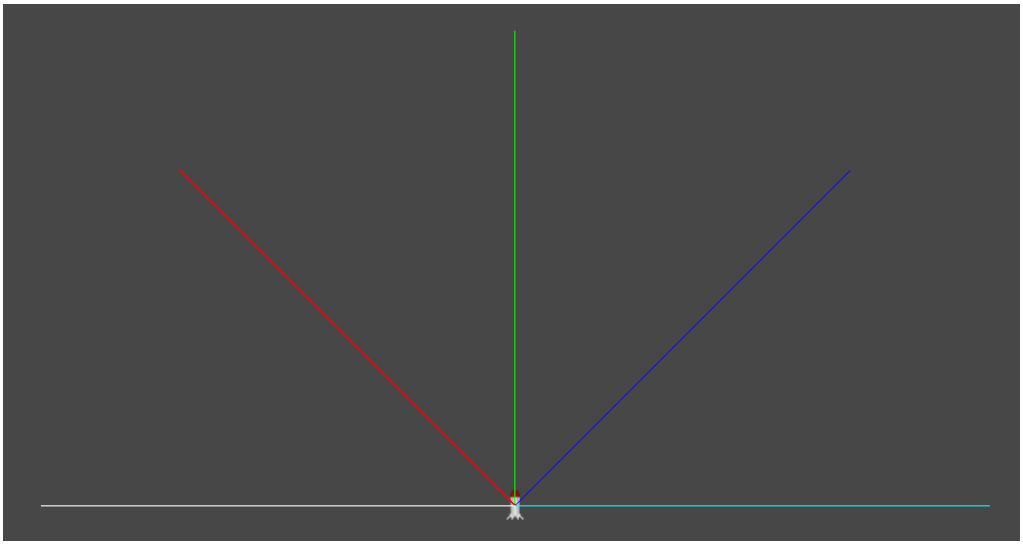
Ovaj algoritam se može koristiti u niz drugih staza pa čak i u 3D scenama (potrebne prilagodbe) što znači da će ovisno o vrsti problema i načinu implementacije ulazni i izlazni sloj imati točno određen broj neurona. Zbog toga su jedini stvarni parametri broj neurona u skrivenim slojevima. Broj neurona u skrivenim slojevima je veoma važan jer s povećanjem broja neurona povećava se mogućnost različitih prilagodbi neuronske mreže odnosno povećava se mogućnost učenja mreže. Kako povećanje neurona ima svoje prednosti, tako ima i mane kao što su:

- povećano zauzimanje memorije – Kod rasta broja neurona potrebna je veća količina prostora za pohranu podataka koji su vezani za broj neurona poput kromosoma svakog agenta (kromosomi u sebi sadrže vrijednosti težina između neurona susjednih slojeva i pomak svakoga neurona).
- više računanja – S povećanjem broja neurona povećava se broj težina i pomaka neurona, izvodi se više računanja aktivacijskih funkcija, suma svih ulaza u određeni neuron je

zahtjevnija, kreiranje djeteta pomoću roditelja je duže (broj gena u kromosomu je veći što znači da će se križanje dulje izvoditi).

Zbog prije navedenih prednosti i mana povezanih s brojem neurona u mreži, potrebno je naći takve brojeve neurona da mreža može naučiti sve zadane prepreke, a s time da nije suviše kompleksna. Jedan od najboljih načina dobivanja takvih parametara je ručno testiranje različitih brojeva neurona od manjih prema većim. Iako je ručna optimizacija parametara neuronske mreže kvalitetna, ne izvodi se toliko često jer je vremenski jako zahtjevna.

Kako bi agenti mogli izbjegavati prepreke odnosno naučiti mrežu, potrebni su određeni podaci iz okoline agenata. Podaci koji se najčešće koriste su podaci o udaljenosti agenta od prepreke te kako je i ranije objašnjeno ovi podaci predstavljaju ulaz u neuronsku mrežu. Udaljenost agenta od prepreke dobiva se pomoću raycast-a. Raycast se može zamisliti kao virtualno svijetlo koje putuje iz predmeta u određenom smjeru te detektira objekte s kojima se sudara i može izračunati udaljenost od svoga ishodišta do objekta s kojim se sudara. Agenti za rad koriste 5 raycastova koji prate udaljenosti te periodički šalju te informacije u neuronsku mrežu (Slika 5.14.).



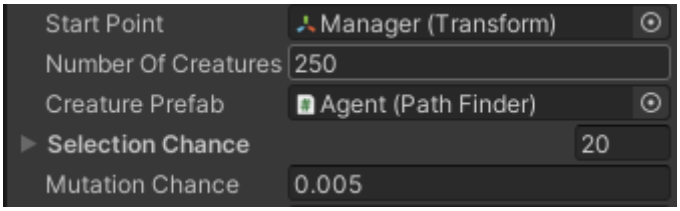
Sl 5.14. Raycastovi agenta

Duljina raycastova još je jedan parametar koji treba podesiti, ali za razliku od ostalih parametara nije toliko rigorozan u smislu da vrijednosti duljina ne moraju biti toliko precizne poput nekih drugih vrijednosti. Bitno je izabrati toliku duljinu raycastova kako bi mreža mogla primijetiti razliku između prepreke koja je daleko i prepreke koja je odmah pored, a opet ne tako kratku duljinu da agent nema vremena reagirati na nadolazeću prepreku. Staza predstavlja problem koji neuronska mreža treba naučiti, a kako bi mreža mogla kvalitetno učiti potrebno je izraditi ili generirati takvu stazu koja ima lakši početak te postupni rast u težini, ali s time da su svi dijelovi staze mogući za naučiti. Kvalitetnom stazom za ovaj algoritam mogla bi se zvati ona staza koja

na početku ima lagana skretanja pa nagla i brza skretanja te nasumične prepreke. Najveći problem neuronskim mrežama s genetskim algoritmom izazivaju slijepi ili pogrešni putevi (putevi koji nemaju nastavak odnosno izlazak) zato što mreža nema moć predikcije te ne može predvidjeti kako određeni put ne vodi do cilja. Veliku ulogu u nastanku toga problema igra vrsta korištene funkcije dobrote. Ako staza ima takav problem te algoritam pretpostavi kako pogrešan put vodi do cilja, dolazi do pauze u učenju jer će mreža pokušati naučiti problem za koji nema rješenja. Iz toga se problema izlazi na način čekanja takve mutacije agenata koja će cjelokupno izbjeći taj put te otići onim pravim. Čekanje na takve mutacije vremenski je dugo, a ovisno o količini i duljini takvih puteva, može se drastično produljiti vrijeme učenja.

Manager je prazan objekt koji u sebi jedino sadržava skriptu za upravljanjem cjelokupnoga algoritma. Ta skripta stvara agente, dodjeljuje dobrotu koja je predstavljena kao preživjelo vrijeme agentu koji se zaustavio, započinje nove generacije te prati i obnavlja podatke u informacijskoj traci. Najvažniji parametri koje treba podesiti su (Slika 5.15.):

- startna pozicija – Lokacija odnosno koordinatne vrijednosti gdje da se stvore svi agenti na početku svake generacije.
- broj agenata – Broj koji govori koliko će se agenata stvoriti u svakoj generaciji. Ovaj parametar bi trebao biti što veći jer se na taj način može istovremeno pokriti veći prostor mogućih rješenja. Fizička ograničenja sustava sprječavaju rast broja agenata.
- inicijalni agent – Pokazatelj na objekt inicijalnog agenta.
- operator selekcije – Parametar koji se sastoji od broja agenata koji će se uzeti za reprodukciju te niza vrijednosti koje predstavljaju vjerojatnost uzimanja određenog roditelja (Slika 5.16.). Za odabir roditelja iz populacije, koristi se rangirajuća selekcija. Broj roditelja koji se uzimaju iz populacije ovisi o tome koliko je velika cjelokupna populacija te koliko se brzo želi konvergiranje rješenja. Nakon odabira broja roditelja, treba odrediti vjerojatnost uzimanja svakog roditelja ovisno o njihovom ranku. Suma svih vjerojatnosti ne mora biti 1, ali se preporučuje zbog lakšeg razumijevanja.
- mutacija – Parametar koji predstavlja vjerojatnost da dođe do mutacije gena. Ovaj parametar treba pravilno izabrati jer ako se postavi na previše malu vrijednost, neće se dobro istražiti prostor rješenja odnosno agenti će postati međusobno sličniji te se na taj način dobiva učinak manjeg broja agenata nego što ih stvarno ima. Ukoliko je vjerojatnost za mutaciju prevelika onda algoritam neće konvergirati te će dolaziti do čestoga gubljenja znanja (proces u kojemu jedna generacija nauči određeni dio problema, ali se u budućim generacijama to znanje izgubi).



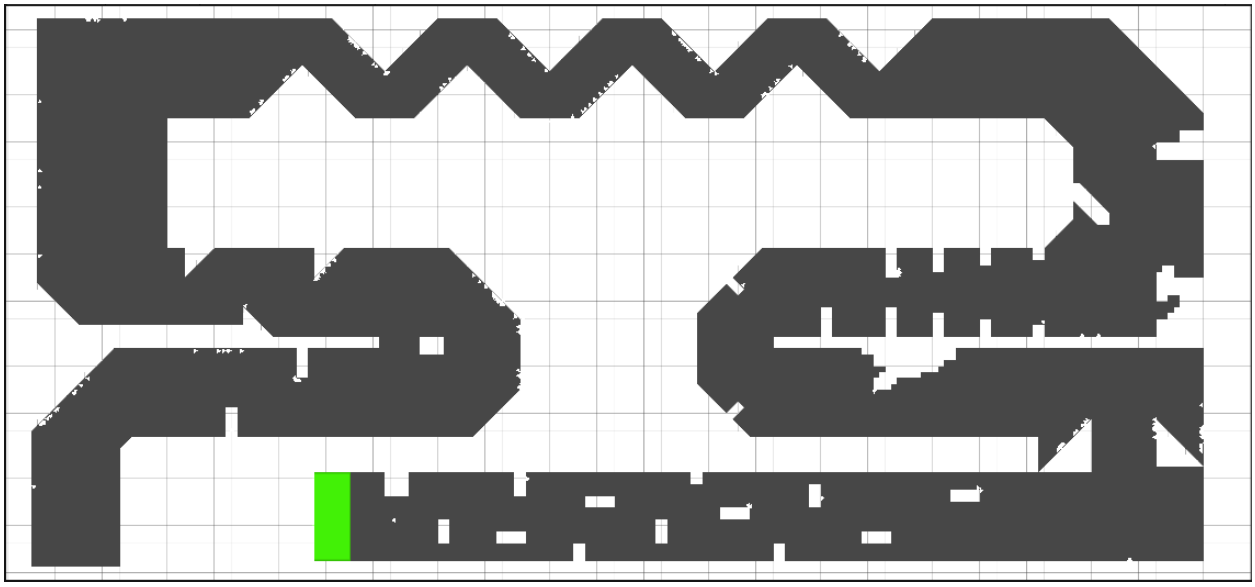
Sl 5.15. Parametri manager-a

Sl 5.16. Parametri operatora selekcije

5.4. Rezultati programa

Prije objašnjenja o rezultatima programa odnosno samog algoritma, treba se uzeti u obzir kako rezultati jako variraju odnosno algoritam u nekim slučajevima može brzo pronaći rješenje. Ponekad je pak potrebna velika količina vremena što se događa zbog toga jer je za neke dijelove staze potrebna pravilna mutacija kako bi se napredovalo u učenju. Problem koji se pojavio je taj što u jednom mjerenju algoritam pronađe put do cilja u 30. generaciji, a u drugom čak ni nakon 150 generacija ne uspijeva pronaći put. Zbog takvih varijacija u eksperimentima, za buduća objašnjenja uzet će se jedno od boljih mjerenja tj. mjerenje u kojemu je algoritam pronašao put do cilja relativno brzo. Rezultat algoritma za pronalaženje puteva dokaz je kako takav algoritam može

pronaći put do cilja te prikaz samoga puta. Algoritam je uspio pronaći put do cilja te se prikaz toga puta može pratiti u programu gdje je taj algoritam implementiran, a cilj je dostignut u 40. generaciji (Slika 5.17.).



Sl 5.17. Prikaz agenta koji dostigne cilj

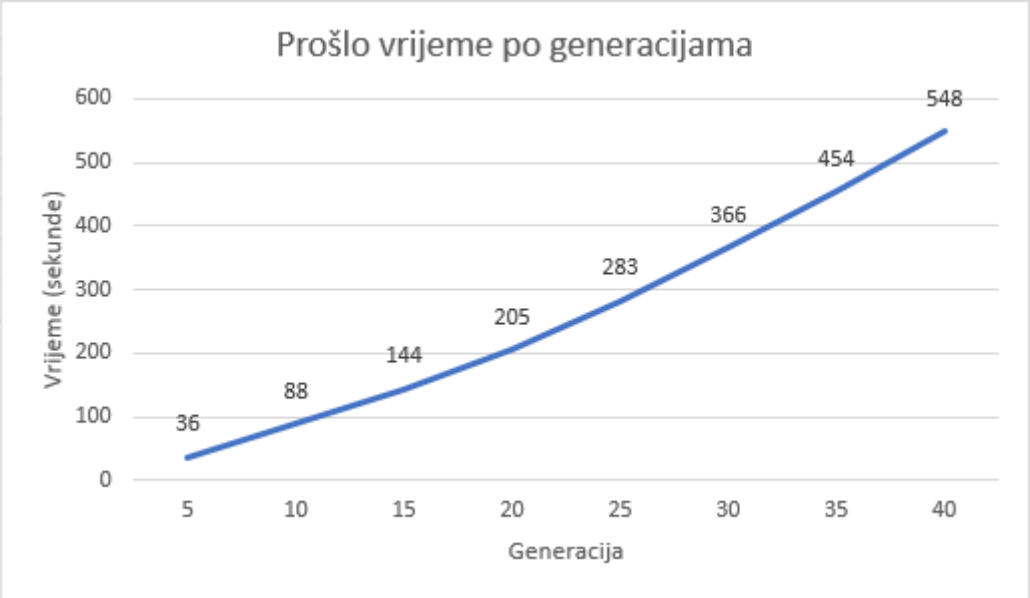
Osim dokaza kako algoritam može pronaći put do cilja te prikaz samoga puta, postoji još jedna vrijednost koja se smatra rezultatom algoritma, a to je vrijeme. Vrijeme služi kao pokazatelj koliko je algoritam prikladan za dati problem te se može koristiti za usporedbu s drugim algoritmima koji pronalaze puteve. Razlika ovoga algoritma od većine ostalih je u tome da osim što pronalazi put do cilja, on taj put pokušava optimizirati. Točnije, nakon što pronađe put, algoritam ne mora nužno stati te se često i dalje ostavlja u radu kako bi našao put koji je bolji (bolji put predstavlja onaj put gdje će agent postići bolje rezultate za funkciju dobrote). Stoga, kada se u daljnjem tekstu bude govorilo o vremenu, mislit će se samo na potrebno vrijeme do pronalaska puta koji vodi do cilja, a ne i daljnje vrijeme provedeno optimizirajući pronađeni put. Potrebno vrijeme da algoritam pronađe put je 548 sekundi. Iako vrijeme daje uvid u kvalitetu samoga algoritma, ono se često proširuje s vremenima generacija kako bi se mogla odraditi daljnja analiza poput proučavanja na kojim dijelovima staze je algoritmu potrebno više vremena da ga nauči (Tab 5.1.).

Tab 5.1. Prikaz potrebnog vremena za generacije

Generacija	Vrijeme (sekunde)
5	36
10	88
15	144
20	205
25	283

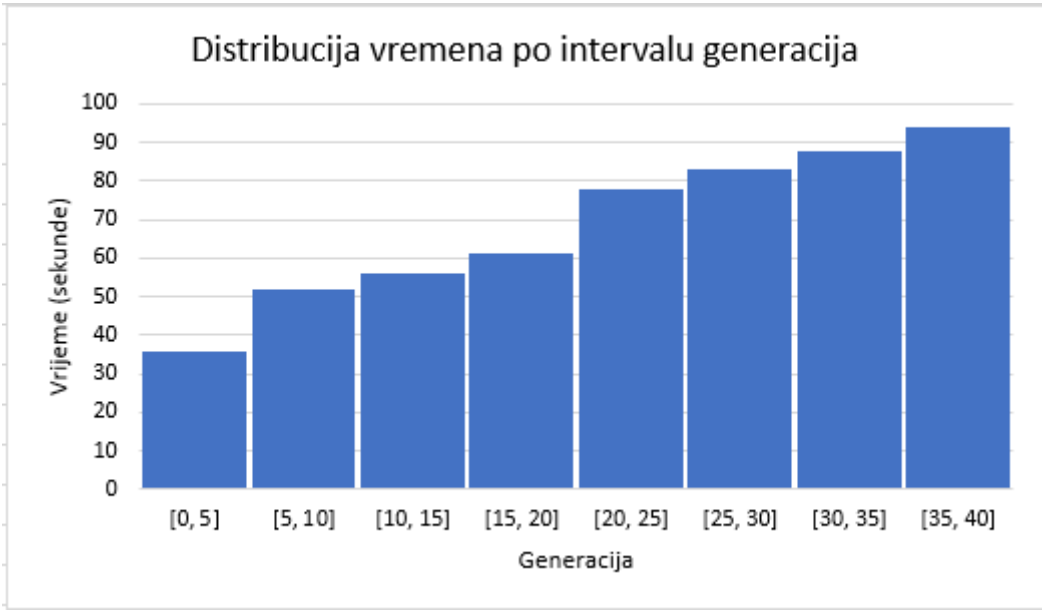
30	366
35	454
40	548

Na slici 5.18. prikazan je linijski dijagram prije spomenutih tabličnih podataka koji iako pruža više informacija nego vrijeme pronalaska puta, on i dalje ne daje uvid u kojim je dijelovima staze algoritmu potrebno više vremena.



Sl 5.18. Linijski dijagram prošlog vremena po generacijama

Daljnjom obradom dobivaju se podaci koji daju informacije o odnosu algoritma i staze (Slika 5.19.).



Sl 5.19. Distribucija vremena po intervalu generacija

Prema slici 5.19., dobiva se prikaz potrebnog vremena po intervalima. Ranije u tekstu je objašnjeno da kako algoritam uči stazu tako mu i raste potrebno vrijeme za svaku generaciju. To se na grafu očituje na način da ako je algoritam naučio određeni dio staze, doći će do skoka u potrebnom vremenu (Slika 5.19.). Ako se u nekim intervalima generacija potrebno vrijeme ne promijeni ili slabo poraste, to znači da je algoritmu potreban veći broj generacija kako bi naučio taj dio staze. Prema slici 5.19., postoje dva skoka u vremenu što znači kako je algoritam dva puta naučio dio staze za koji mu je bilo potrebno više generacija.

6. ZAKLJUČAK

Zadatak ovoga rada bio je proučiti, analizirati i softverski konstruirati algoritam za pronalaženje puteva koristeći potpuno povezanu duboku neuronsku mrežu s genetskim algoritmom koji optimizira duboku neuronsku mrežu. Bilo je potrebno obaviti detaljnu analizu teorijske pozadine neuronske mreže kako bi se mogla razumjeti, koristiti i efikasno implementirati u algoritam za pronalaženje puteva. Isto je tako bilo potrebno obaviti istraživanje i analizu genetskoga algoritma radi razumijevanja svih procesa koje taj algoritam izvodi te shvaćanje svakoga od parametara od kojih se sastoji, odabira najbolje vrste algoritma za zadani problem. U konstrukciji programa trebalo je pravilno implementirati neuronsku mrežu i genetski algoritam sa svim svojim funkcionalnostima te ih znati međusobno povezati. Za testiranje toga algoritma za pronalaženje puteva kreirana je 2D staza u Unityu te prototip stvorenja pomoću kojega je kreirana početna populacija. Izvedeno je nekoliko testiranja kako bi se namjestili neki od važnijih parametara, ali za kvalitetniji odabir parametara potrebna su daljnja testiranja te vizualizacija podataka kako bi bilo lakše prepoznati koje vrijednosti parametara treba postaviti. Kako bi algoritam bolje radio te izbjegao neke moguće greške, trebala bi se zamijeniti funkcija dobrote.

LITERATURA

[1] R. Graham, H. McCabe i S. Sheridan: Neural Networks for Real-time Pathfinding in Computer Games, The ITB Journal, Vol. 5, Iss. 1, Article 21, 2004. https://arrow.tudublin.ie/cgi/viewcontent.cgi?article=1204&context=itbj (srpanj 2023.)

[2] R. Graham, H. McCabe i S. Sheridan: Pathfinding in Computer Games, The ITB Journal, Vol. 4, Iss. 2, Article 6, 2003. https://arrow.tudublin.ie/cgi/viewcontent.cgi?article=1063&context=itbj (srpanj 2023.)

[3] N. Bolf i I. Jerbić: Primjena umjetnih neuronskih mreža, Kemija u industriji. 55 (11), str. 457-468. 2006. https://hrcak.srce.hr/file/10369 (lipanj 2023.)

[4] M. Baotić: Identifikacija i upravljanje nelinearnim vremenski promjenjivim procesima primjenom neuronskih mreža, Magistarski rad, Sveučilište u Zagrebu, 2000.

[5] D. Domijan: Uvod u neuronske mreže, Metodički ogledi, str. 101-127, 2000.

[6] E. Ariawan, M. Zefanya: Predictive Analysis of Employee Loyalty: A Comparative Study Using Logistic Regression Model and Artificial Neural Network, The 7th International Conference on Mathematics and Natural Sciences, 2018. https://www.researchgate.net/publication/329799984_Predictive_Analysis_of_Employee_Loyalty_A_Comparative_Study_Using_Logistic_Regression_Model_and_Artificial_Neural_Network (lipanj 2023.)

[7] R. Grbić i P. Pejić: 8. Laboratorijska vježba: Umjetne neuronske mreže, Ak. god. 2022./2023. https://moodle.srce.hr/2022-2023/pluginfile.php/8032542/mod_folder/content/0/LV8.pdf?forcedownload=1 (lipanj 2023.)

[8] T. Krambeger, B. Nožica, I. Dodig, D. Cafuta: Pregled tehnologija u neuronskim mrežama, POLYTECHNIC & DESIGN, Vol. 7, No.1, 2019.

[9] R. Pramoditha: How to Choose the Right Activation Function for Neural Networks, Towards Dana Science, 2022. https://towardsdatascience.com/how-to-choose-the-right-activation-function-for-neural-networks-3941ff0e6f9c (lipanj 2023.)

[10] J. Brownlee: How to Choose an Activation Function for Deep Learning, Deep Learning, 2021. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/ (lipanj 2023.)

[11] P. K. Singh: MRFGRO: a hybrid meta-heuristic feature selection method for screening COVID-19 using deep features, Scientific reports, 2021. https://www.researchgate.net/publication/357068475_MRFGRO_a_hybrid_meta-heuristic_feature_selection_method_for_screening_COVID-19_using_deep_features (lipanj 2023.)

[12] J. Feng: Reconstruction of porous media from extremely limited informationusing conditional generative adversarial networks, PHYSICAL REVIEW E 100, 2019. https://www.researchgate.net/publication/335845675_Reconstruction_of_porous_media_from_extremely_limited_information_using_conditional_generative_adversarial_networks (lipanj 2023.)

[13] A. T. da Silva: DenseNet-DC: Optimizing DenseNet Parameters Through Feature Map Generation Control, Revista de Informática Teórica e Aplicada, 2020. https://www.researchgate.net/publication/342369912_DenseNet-DC_Optimizing_DenseNet_Parameters_Through_Feature_Map_Generation_Control (lipanj 2023.)

[14] A. Upadhyay: Activation Function & It's Type In Neural Network (Part 2), 2020. https://uanurag6212.medium.com/activation-function-its-type-in-neural-network-part-2-6d9efffea62 (lipanj 2023.)

[15] V. Mallawaarachchi: Introduction to Genetic Algorithms — Including Example Code, Towards Data Science, 2017. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (lipanj 2023.)

[16] M. Golub: Genetski algoritam, drugi dio, V 2.2, 2004. http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf (lipanj 2023.)

[17] E. Cantú-Paz: A Survey of Parallel Genetic Algorithms, Department of Computer Science and Illinois Genetic Algorithms Laboratory University of Illinois at Urbana-Champaign https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a13fcba4fab4713d5acb2d970eddc6b148d2596d (lipanj 2023.)

[18] M. Čupić: Prirodom inspirirani optimizacijski algoritmi, FER, V. 1.0.7, 2010. https://www.fer.unizg.hr/_download/repository/Cupic2009-PrirodomInspiriraniOptimizacijskiAlgoritmi.pdf (lipanj 2023.)

[19] M. Golub: Genetski algoritam, prvi dio, V2.3 ,2004. http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf (lipanj 2023.)

[20] E. Ivanjko: Genetski algoritam, Umjetna inteligencija, ZITS, Ak. God. 2017. https://www.fpz.unizg.hr/eivanjko/files/UI/UIP07GeneteskiAlgoritam.pdf (lipanj 2023.)

SAŽETAK

Neuronske mreže u današnjici predstavljaju jednu od popularnijih grana umjetne inteligencije. One su skup osnovnih jedinica koje su slične biološkim neuronima jer su po principu rada i njihovim dijelovima veoma slične. Neuronske mreže služe kao poveznica između neurona u mozgu i umjetne inteligencije te na taj način pomažu pri boljem shvaćanju ljudskoga mozga, nude mogućnost obrade velike količine podataka u stvarnom vremenu i služe za rješavanje niza kompleksnih problema koji su prije bili teško rješivi. Tehnika koja nalazi sve veću uporabu te je najzastupljenija grana evolucijskih algoritama, zove se genetski algoritam. Genetski algoritam je posebna tehnika istraživanja koja se temelji na evoluciji te se pomoću njezinih simulacija rješava niz problema koji se bave pretragom. Genetski algoritam se koristi u mnogo različitih disciplina a pogotovo u strojnome učenju. Njegova uloga u strojnome učenju je optimizacija algoritama jer s vremenom pruža takve modele koji točnije opisuju i rješavaju zadani problem. Spoj neuronske mreže i genetskoga algoritma daje mogućnost kreiranja posebne vrste umjetne inteligencije koja se može koristiti za otkrivanje puteva i izbjegavanje prepreka čak i kod kompleksnih ili promjenjivih staza.

Ključne riječi: genetski algoritam, neuronske mreže, pronalaženje puteva, Unity

ABSTRACT

Neural networks represent one of the most popular branches of artificial intelligence today. They are a set of basic units that are similar to biological neurons because they are very similar in terms of the principle of operation and their parts. Neural networks serve as a link between neurons in the brain and artificial intelligence and thus help us to better understand the human brain, offer us the possibility of processing large amounts of data in real time and serve to solve a number of complex problems that were previously difficult to solve. The technique that finds increasing use and is the most represented branch of evolutionary algorithms is called the genetic algorithm. The genetic algorithm is a special research technique based on evolution, and with the help of its simulation, it solves a number of search related problems. Genetic algorithm is used in many different disciplines and especially in machine learning. Its role in machine learning is the optimization of algorithms because over time it provides us with such models that describe and solve the given problem more accurately. The combination of a neural network with genetic algorithm gives us the possibility of creating a special type of artificial intelligence that can be used to discover paths and avoid obstacles even on complex or changeable paths.

Keywords: genetic algorithm**,** neural network, pathfinding, Unity