Primjena NEAT algoritma za navigaciju mobilnog robota

Gajdek, Vita

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Vita Gajdek

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Sudent:

izv. prof. dr. sc. Petar Ćurković, dipl. ing. Vita Gajdek

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se obitelji na strpljenju i podršci tijekom obrazovanja i pisanja rada, te svim kolegama i prijateljima na pruženoj pomoći.
Posebice zahvaljujem Tiboru Vraniću na motivaciji i prijateljstvu tijekom studija.

Također se zahvaljujem mentoru prof. dr. sc. Petru Ćurkoviću na ukazanom povjerenju, vodstvu i stručnosti koju ste dijelili sa mnom tokom izrade rada i preddiplomskog studija.

Vita Gajdek

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum 20 -09- 2024	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –** 334	

ZAVRŠNI ZADATAK

Student: **Vita Gajdek** JMBAG: **0035228426**

Naslov rada na hrvatskom jeziku: **Primjena NEAT algoritma za navigaciju mobilnog robota**

Naslov rada na engleskom jeziku: **Application of the NEAT algorithm for mobile robot navigation**

Opis zadatka:

NEAT algoritam (Neuro Evolution of Augmenting Topologies) je algoritam strojnog učenja temeljen na kombinaciji evolucijskog algoritma i umjetnih neuronskih mreža. Ovim pristupom omogućuje se da pri treniranju mreže objekt treniranja nisu samo težine sinapsi, već i topologija i struktura mreže. Stoga pri treniranju dolazi do kompleksifikacije mreže proporcionalno s težinom problema. NEAT algoritam uspješno se primjenjuje u mobilnoj robotici, gdje omogućuje konstantnu prilagodbu ponašanja robota u odnosu na kontinuirano promjenjivu okolinu.

U radu je potrebno napraviti sljedeće:

- upoznati se s NEAT algoritmom, opisati ga i upoznati se s primjerima primjene s naglaskom na mobilnu robotiku,
- u programskom paketu Python implementirati NEAT algoritam koji će omogućiti mobilnom robotu kretanje u okolini od početne do odredišne točke,
- analizirati parametre NEAT algoritma, dati preporuke za optimalan izbor parametara,
- kritički se osvrnuti na dobivene rezultate i dati preporuke za budući rad.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2023.	**1. rok:** 22. i 23. 2. 2024. **2. rok (izvanredni):** 11. 7. 2024. **3. rok:** 19. i 20. 9. 2024.	**1. rok:** 26. 2. – 1. 3. 2024. **2. rok (izvanredni):** 15. 7. 2024. **3. rok:** 23. 9. – 27. 9. 2024.

Zadatak zadao:
Izv. prof. dr. sc. Petar Ćurković

Predsjednik Povjerenstva:
Prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

b_k	- bias neurona k
c_i	- koeficijenti jednadžbe kompatibilnosti
D	- broj neuparenih gena
d	- udaljenost koju prelazi jedinka u jednoj sličici
E	- broj suvišnih gena
F	- fitnes funkcija
g	- genom g
k	- neuron k
N	- broj gena u većem genomu
t	- trajanje simulacije izraženo u broju sličica
$\bar{W}$	- broj podudarnih gena
w_{ki}	- težine neurona k
x_i	- ulazni signal
y_k	- izlazni signal
φ	- aktivacijska funkcija
δ	- mjera kompatibilnosti genoma
δt	- prag kompatibilnosti

SAŽETAK

Tehnološki napredak u posljednjih nekoliko desetljeća uvelike se oslanja na postignuća ostvarena u strojnom učenju. Danas je gotovo nemoguće zamisliti modernu tehnologiju koja ne koristi umjetne neuronske mreže. Težnja za optimizacijom inteligentnih sustava potaknula je i razvoj evolucijskog računarstva inspiriranog principima prirodne evolucije.

U ovom će se radu objasniti mehanizmi evolucijskih algoritama čijom se primjenom nastoji poboljšati put do rješenja kompleksnih zadataka. Cilj rada je analizirati NEAT (engl. *NeuroEvolution of Augmenting Topologies)* algoritam i osvrnuti se na njegovu implementaciju u području mobilne robotike.

Korištenjem biblioteke *neat-python* u programskom jeziku *python* implementirat će se NEAT algoritam u svrhu navigacije mobilnih robota. Analizirat će se rezultati simulacije te će se dati prijedlozi za daljnji rad.

Ključne riječi: evolucijski algoritmi, neuroevolucija, NEAT, mobilna robotika, python

SUMMARY

Advancements in technology in the last few decades stem from the achievements in the field of machine learning. It is almost impossible to imagine modern day technology that isn't using artificial neural networks. Evolutionary computation has gained significant interest due to the constant strive for optimization of intelligent systems.

This study elaborates on the concepts of evolutionary algorithms that are aimed to improve problem solving processes of more complex tasks. The goal is to analize NEAT (*NeuroEvolution of Augmenting Topologies)* algorithm in the context of mobile robotics.

Furthermore, the NEAT algorithm will be implemented in *Python* using *neat-python* library in order to simulate a controller of a mobile robot. The simulation results shall be analized for multiple cases.

Key words: evolutionary algorithms, neuroevolution, NEAT, mobile robotics, python

1. UVOD

Priroda je oduvijek služila znanstvenicima i inženjerima kao izvor inspiracije za stvaranje inovacija. Robusnost prirodnih sustava poželjna je karakteristika koju se oduvijek nastojalo pretočiti u one umjetne. Međutim, činjenica da su se mnogi prirodni sustavi razvijali kroz milijune godina otkriva da se radi o vrlo sofisticiranim i kompleksnim sustavima. Otkrića u poljima prirodnih znanosti omogućila su bolje razumijevanje njihovih koncepata, međutim matematički ih opisati na što jednostavniji često predstavlja velik izazov. Mehanizmi kojima se koristi evolucija i ljudski mozak privukli su velik interes u svijetu računalnih znanosti i inženjerstva. Razvoj tehnologije omogućio je da se upravo ti mehanizmi prenesu u digitalni svijet i pruže nove metode rješavanja zamršenih funkcija.

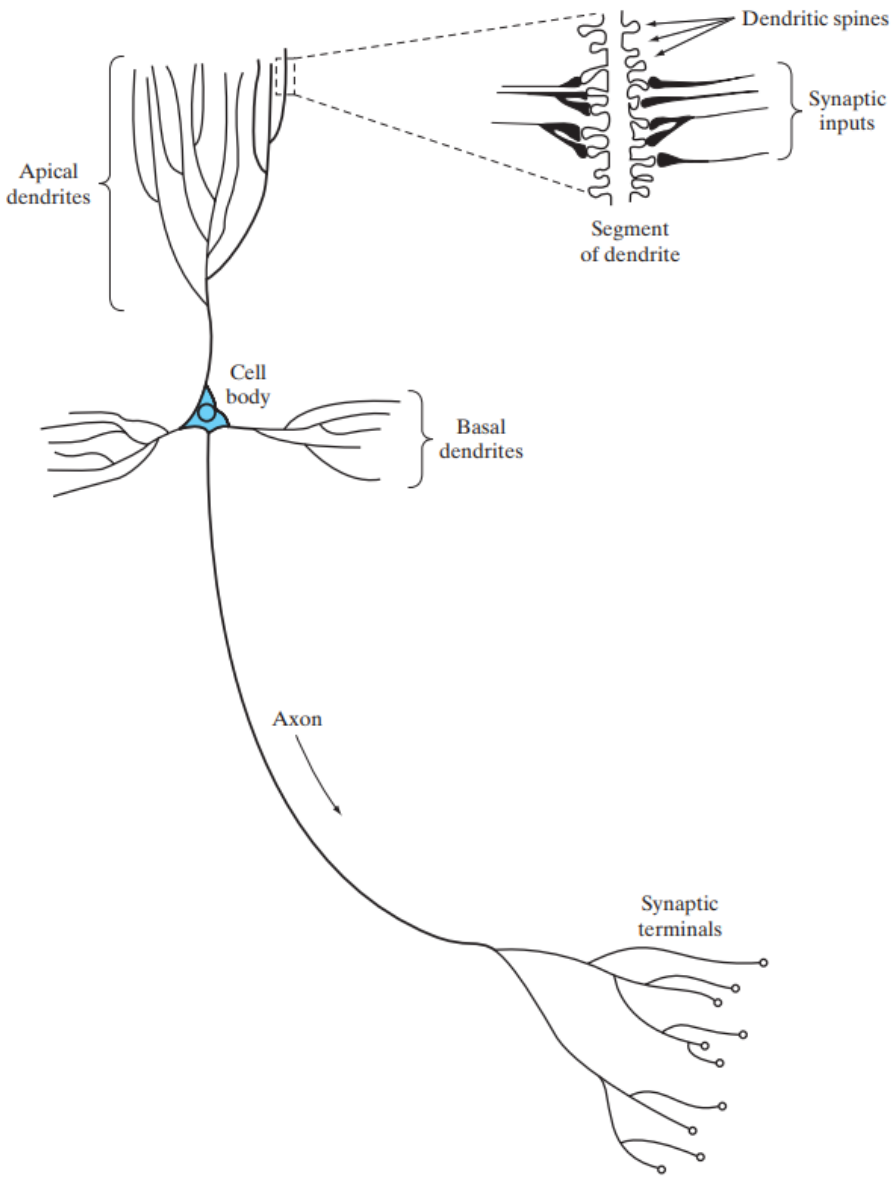
U nastavku će se dataljnije opisivati koncepti koji se zasnivaju na prirodnim procesima. Nastojat će se detaljno obrazložiti ključni mehanizmi neuroevolucije, te način na koji se oni mogu primjeniti pomoću NEAT algoritma. Implementacija algoritma prikazat će se kroz simulaciju kontrolera mobilnog robota unutar nekoliko različitih okolina. Analiza rješenja dat će uvid o uspješnosti primjene takve metode za potrebe mobilne robotike.

2. NEURONSKE MREŽE

Ljudski mozak vrlo je kompleksan, nelinearan i paralelan procesor informacija koji ima mogućnost organizirati svoje gradivne jedinice (neurone) za obavljanje raznih funkcija kao što su: prepoznavanje uzoraka, percepcija, motoričke vještine [1]. Već pri rođenju mozak posjeduje poprilično složenu strukturu te mogućnost razvijanja i prilagodbe okolini kroz iskustvo.
Isto tako, u području umjetne inteligencije, neuronske su mreže načinjene od umjetnih neurona. Neuronska mreža je mehanizam koji je izrađen kako bi imitirao način na koji mozak obavlja određeni zadatak ili funkciju. Mreža se obično implementira pomoću elektroničkih komponenti ili se simulira u softveru na digitalnom računalu. One koriste međupovezanost neurona, odnosno procesnih jedinica, nastalih kroz proces učenja. Algoritam učenja modificira sinaptičke težine mreže kako bi se dostiglo željeno rješenje za određeni problem. Prednost njihove paralelno distribuirane strukture jest računalna snaga, a također se reflektira kroz dobivanje dobrih rezultata za ulaze s kojima se mreža nije susrela u procesu učenja. Takva karakteristična obrada informacija omogućava neuronskim mrežama dobru aproksimaciju rješenja kompleksnih problema.

2.1. Biološki neuron

Početkom 20. stoljeća Ramón y Cajál predstavio je ideju neurona kao strukturne i funkcionalne jedinice mozga. Procjenjuje se da se samo u moždanoj kori nalazi preko 10 milijardi neurona te 60 trilijuna sinapsi [1]. Takva struktura očekivano je vrlo efikasna, posebice kada se uzme u obzir njezina energetska učinkovitost.
Biološki neuron građen je od dendrita, tijela, aksona i sinaptičkih završetaka (Slika 1). Sinapse su elementarne funkcionalne jedinice koje posreduju u interakciji neurona. One uzrokuju ekscitaciju ili inhibiciju receptivnog neurona koji prima informacije putem dendrita. Većina neurona kodira svoje izlaze u obliku kratkih naponskih impulsa, a ukoliko uzbuda prelazi prag osjetljivosti receptivnog neurona, on šalje impuls kroz akson.
Mozak ima mogućnost razvijati se, prilagođavati i učiti što ga čini „plastičnim“, a to je posljedica stvaranja novih sinaptičke veza ili modifikacije postojećih sinapsi.

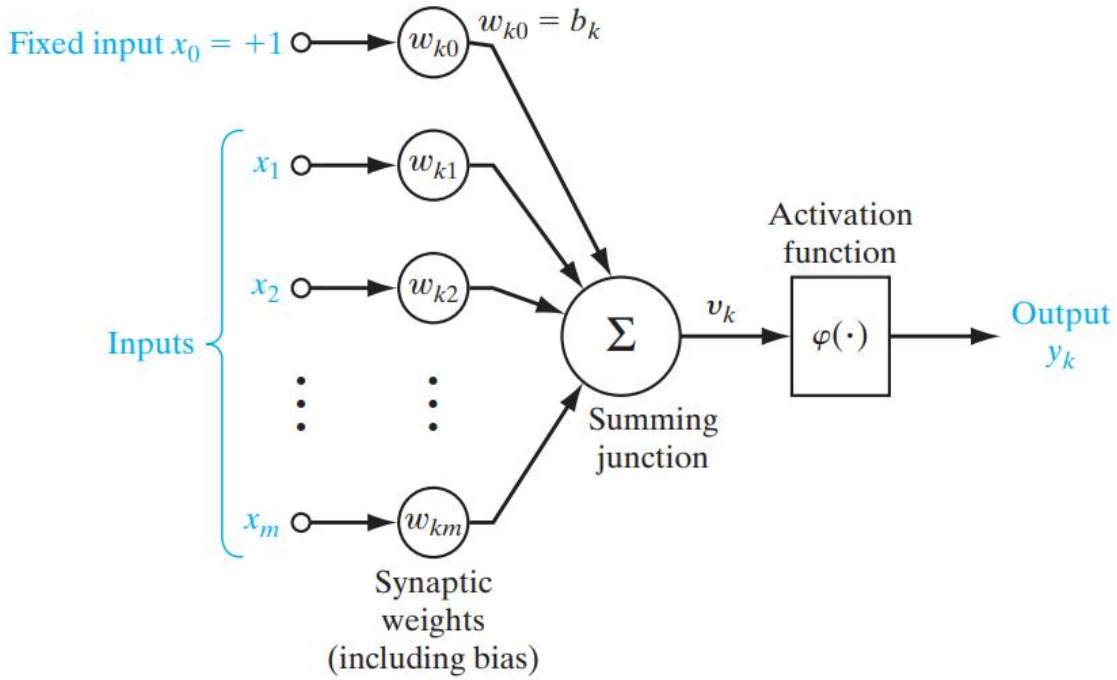


Slika 1 Biološki neuron [1]

Važno je napomenuti da je prethodno opisana struktura jedinstvena karakteristika mozga te su umjetni neuroni koji se koriste za izgradnju umjetne neuronske mreže primitivni u usporedbi s onima koje nalazimo u mozgu.

2.2. Umjetni neuron

Umjetni neuroni predstavljaju temeljnu jedinku procesiranja u neuronskoj mreži. Blok dijagram na slici 2 prikazuje osnovni model za projektiranje mreža s tri osnovna elementa.



Slika 2 Osnovni model umjetnog neurona [1]

Svaki model neurona sastoji se od:

- Sinapse
- Sumatora
- Aktivacijske funkije [1].

Sinapse povezuju neurone s ulazima, izlazima te neuronima iz drugih slojeva. Svaka sinapsa karakterizirana je težinom koja može poprimiti vrijednost u rasponu [-1,1]. Drugim riječima, svaki ulazni signal x_i koji je povezan s neuronom k množi se iznosom težine w_{ki}. Zatim se u sumatoru zbrajaju svi ulazni signali koji su prethodno ponderirani odgovarajućim težinama. Na izlaz sumatora primjenjuje se aktivacijska funkcija koja ograničava amplitudu izlaznog signala neurona na neku konačnu vrijednost. Na model se također primjenjuje bias b_k koji pomiče (povećava ili smanjuje) ulaz aktivacijske funkcije. Primarna uloga biasa je omogućiti modelu aktivaciju čak i onda kada su svi ulazi jednaki nuli. Na taj način neuron može bolje aproksimirati razne funkcije što povećava sposobnost mreže za učenjem složenijih uzoraka.

Ovaj se model (slika 1.2) može i matematički zapisati na sljedeći način:

$$v_k = \sum_{i=0}^{m} w_{ki} x_i, \tag{2.1}$$

$$y_k = \varphi(v_k). \tag{2.2}$$

Bitno je napomenuti da formule vrijede kada je:

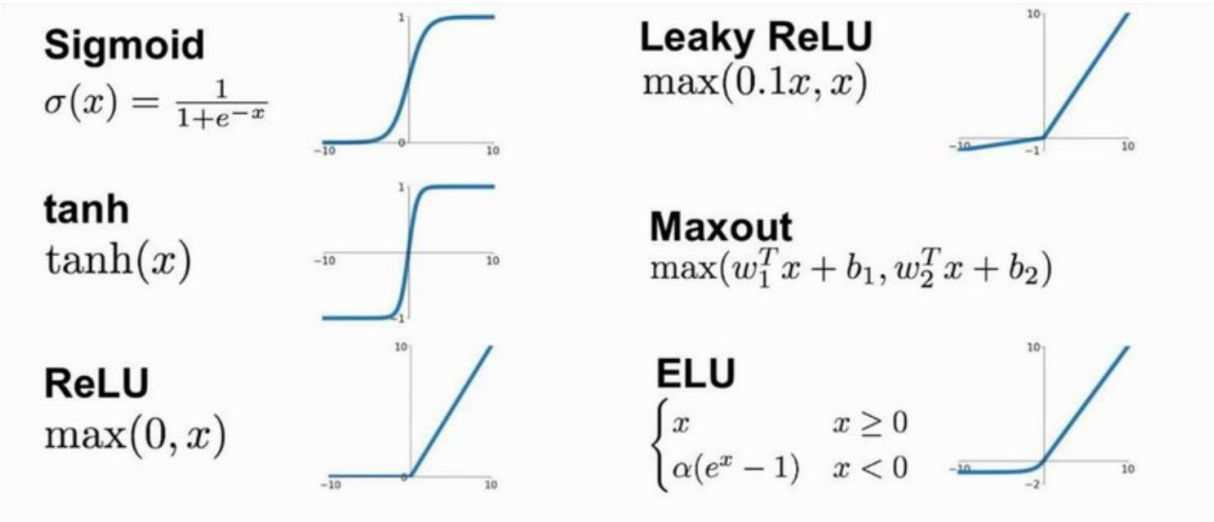
$$x_0 = +1, \tag{2.3}$$

$$w_{k0} = b_k. \tag{2.4}$$

Aktivacijska funkcija φ(*v*) definira izlaz iz neurona, odnosno sumu ulaza koji su prethodno otežani. Bez nje bi neuronska mreža bila samo linearna kombinacija ulaza i težina, što bi ograničilo njenu sposobnost učenja složenijih odnosa među podacima. Ako bi primjerice uzeli tangens hiperbolni na modelu, funkcija bi tada glasila:

$$\varphi(v) = \tanh(v). \tag{2.6}$$

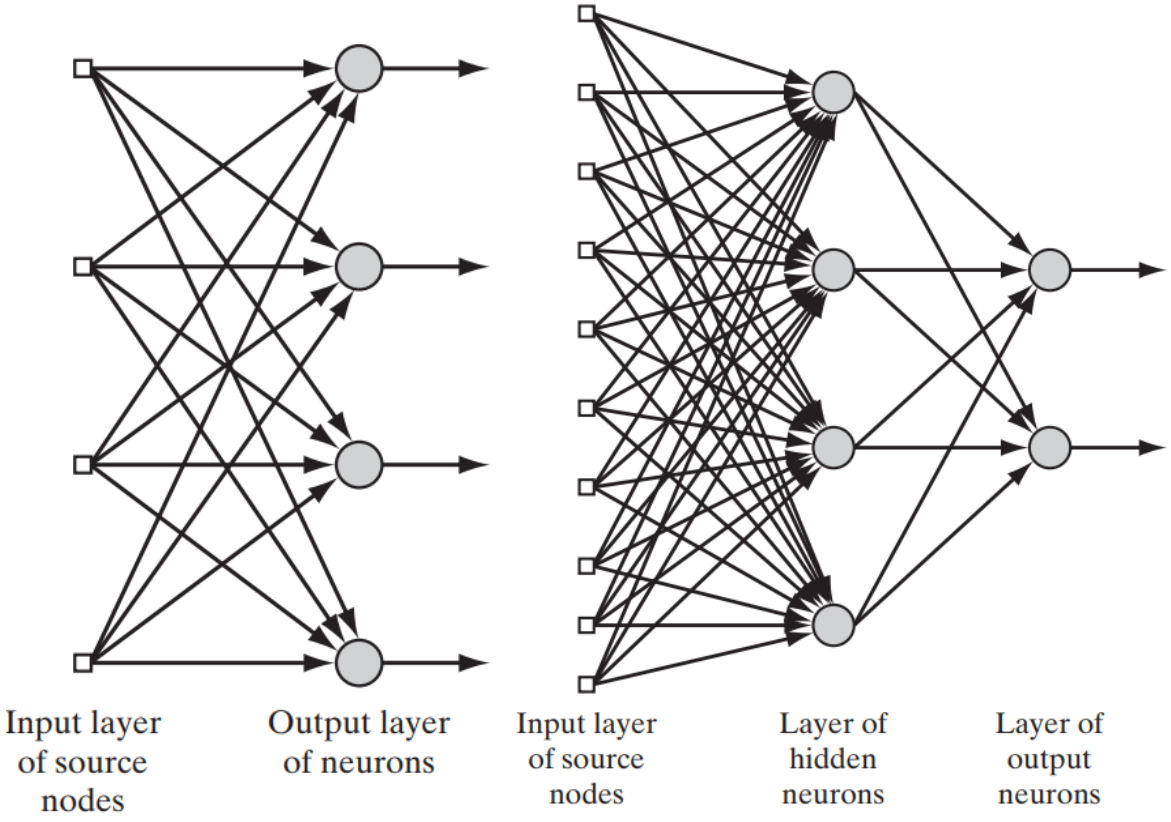
Postoji još nekolicina aktivacijskih funkcija koje se mogu koristiti i prikazane su na slici 3.



Slika 3 Primjeri aktivacijskih funkcija [2]

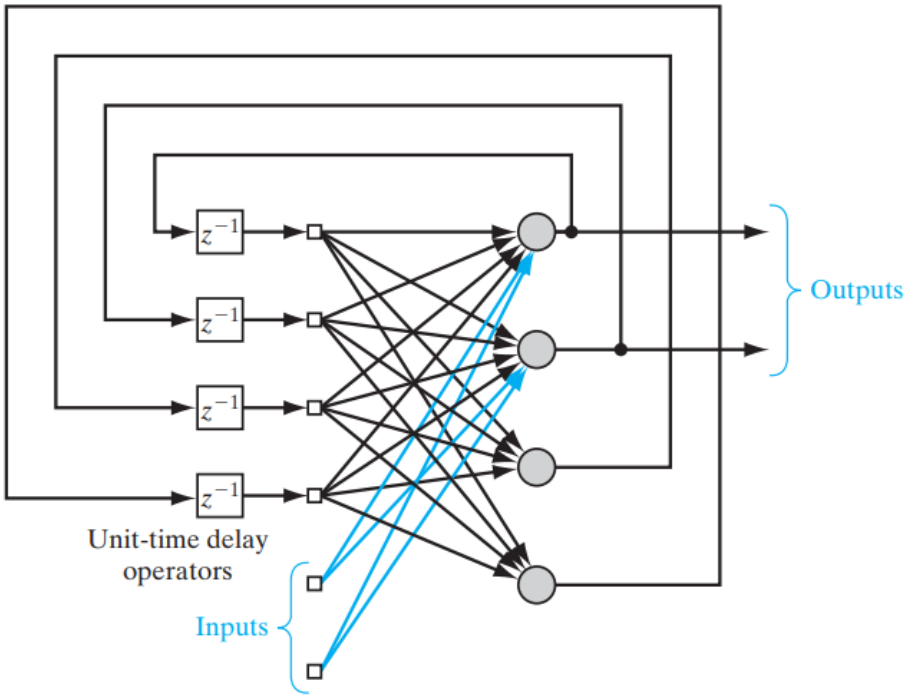
2.3. Umjetne neuronske mreže

Umjetna neuronska mreža je paralelna distribuirana informacijska struktura koja se sastoji od elemenata procesiranja (neurona) koji su povezani u slojeve jednosmjernim vezama [3]. Postoji više podjela mreža – one mogu biti statičke ili dinamičke, imati jedan ili više slojeva itd. Kod statičke mreže (engl. *Feedforward*) informacije se prenose u jednom smjeru – od ulaza do izlaza. Primjer takve mreže je jednoslojni perceptron (slika 4). Naime, Neuroni su unutar mreže raspoređeni u nekoliko slojeva. Sloj se može definirati kao skup neurona koji međusobno nisu povezani, već su umreženi s neuronima iz drugih slojeva. Također postoje i višeslojne statičke mreže koje uz ulazni i izlazni sloj sadrže i skriveni sloj. U njemu se nalaze skriveni neuroni od kojih su, kako im ime kaže, ulazi i izlazi mreže skriveni, a njihova je uloga omogućiti mreži učenje složenijih zadatkaka/obrazaca.



Slika 4 Primjeri jednoslojne i višeslojne statičke neuronske mreže [1]

Postoje i dinamičke mreže (slika 5) koje sadrže povratne veze. Za razliku od statičkih mreža koje nemaju memoriju o prethodnim stanjima, dinamičke mreže omogućuju održavanje informacija o prethodnim stanjima kroz vrijeme te iskoristiti ih za generaliziranje izlaza.



Slika 5 Primjer dinamičke neuronske mreže [1]

Struktura neuronske mreže usko je povezana s algoritmom učenja koji se koristi za treniranje mreže. Algoritam učenja ključna je stavka za uspješan rad neuronske mreže. Treniranjem mreže optimiziraju se njezine težine što rezultira sve točnijim i učinkovitijim modelom. Modifikacija težina moguća je uz povratno prostiranje pogreške (engl. *backpropagation*). Potrebno je izračunati funkciju gubitaka, odnosno usporediti trenutne rezultate mreže s željenim rezultatima. Ona se može računati na više načina, a cilj je minimizirati ju:

$$\text{Loss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}\log(\widehat{y_{ij}}) \tag{2.7}$$

Ovo je samo jedan od načina na koji neuronske mreže uče i optimiziraju svoje parametre. U nastavku će se detaljnije objasniti evolucijski mehanizmi koji nastoje unaprijediti način na koji neuronske mreže mogu učiti.

3. EVOLUCIJSKI ALGORITMI

Evolucijsko računarstvo je istraživačko područje koje spada pod računalne znanosti. Kao što se može razaznati iz naziva, inspiracija proizlazi iz prirodnog procesa evolucije. Poveznica između prirodne i umjetne evolucije može se obrazložiti na sljedeći način: okruženje sadrži populaciju jedinki koje teže za preživljavanjem i reprodukcijom. Kvalitetu ovih jedinki određuje okolina te reflektira koliko su uspješne u ostvarenju svojih ciljeva. Drugim riječima, predstavlja njihove šanse za opstanak i razmnožavanje. U kontekstu stohastičkog procesa rješavanja zadataka metodom pokušaja i promašaja, postoji skup kandidata (rješenja) čija kvaliteta odražava koliko dobro rješavaju problem te određuje vjerojatnost da budu zadržane i iskorištene za stvaranje daljnjih kandidata (rješenja) [4].

Za analizu takvih algoritama potrebno je prvo iznijeti neka temeljna zapažanja iz genetike. Naime, svaka jedinka posjeduje fenotip i genotip. Svojstva fenotipa svakog pojedinca reprezentirana su na razini genotipa, drugim riječima, genotip kodira karakteristike fenotipa pomoću gena.

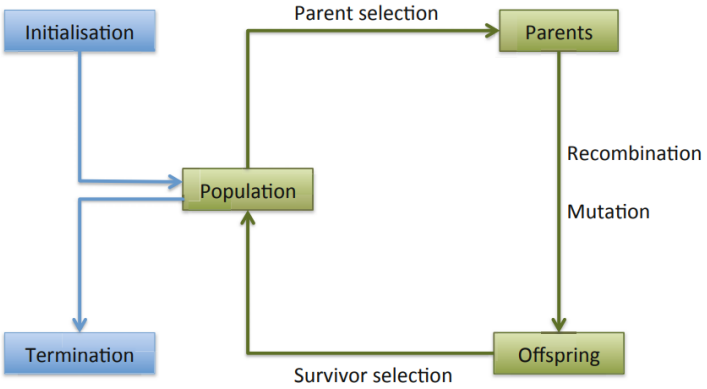
Svi evolucijski algoritmi dijele temeljnu ideju umjetne evolucije, a ono što ih međusobno razlikuje jesu strukture podataka koje koriste za kodiranje jedinki. U nastavku će se govoriti o genetskim algoritmima čija reprezentacija rješenja obično podrazumijeva korištenje nizova ili bit-stringova.

3.1. Genetski algoritmi

Robusnost potrebna za preživljavanje adaptivnih sustava u raznolikim okruženjima ponukala je znanstvenike da njihove mehanizme iskoriste pri stvaranju Genetskih algoritama (GA). Prednost takvih sustava čini njihova fleksibilnost uz računalno jednostavne, ali moćne algoritme. Genetski algoritmi su alat pomoću kojeg se nastoje optimizirati procesi ili funkcije, odnosno poboljšati tradicionalne metode.

Glavna ideja GA jest da je populacija jedinki u nekoj okolini s ograničenim resursima izložena prirodnoj selekciji te da će u natjecanju za resurse opstati najbolje jedinke. Iz tog će razloga sposobnost na razini populacije rasti. Pomoću zadane funkcije kvalitete koju se nastoji maksimizirati, nasumično se stvori skup kandidata (rješenja). Zatim funkciju kvalitete primjenjujemo na kandidate kao mjeru fitnesa (dobrote). S obzirom na fitnes kandidata izabiru se oni s najvećom ocjenom te ih se koristi za stvaranje novih generacija. Kako bi u tome uspjeli, primjenjuju se varijacijski operatori rekombinacije i/ili mutacije. Novonastalo potomstvo se

međusobno natječe za opstanak u sljedeću generaciju sve dok se ne dostigne zadovoljavajuće rješenje (fitnes) ili se dosegne ograničenje.




```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

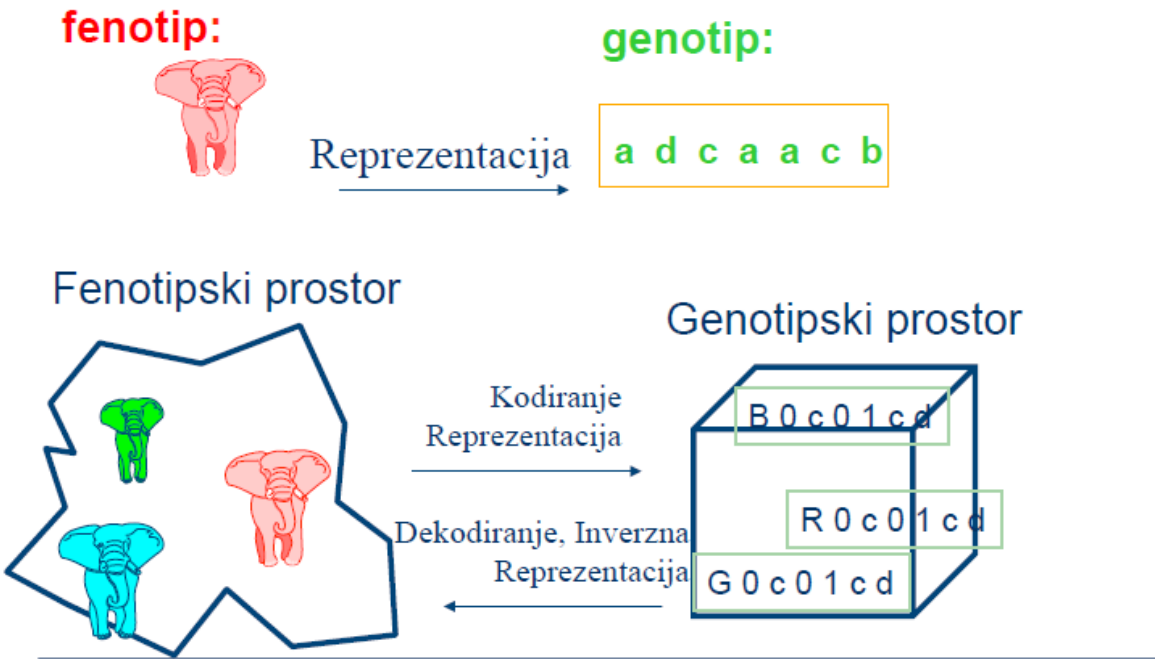
Slika 6 Pseudo-kod genetskog algoritma [6]

Iz pseudo-koda (slika 6) mogu se izdvojiti komponente, procedure i operatori koji su ključni za primjenu genetskih algoritama:

- Reprezentacija (definiranje jedinki),
- Fitnes funkcija (funkcija dobrote),
- Populacija,
- Mehanizam izbora roditelja,
- Varijacijski operatori; rekombinacija i mutacija,
- Mehanizam izbora opstalih (replacement) [5].

3.1.1. *Reprezentacija*

Kako bi se genetski algoritam mogao inicijalizirati prvo je potrebno napraviti reprezentaciju. Taj korak podrazumijeva razlučiti što predstavlja genotip, a što fenotip te definirati proces mapiranja fenotipova na skup genotipova. Važno je pritom spomenuti da se pretraživanje prostora odvija u genotipu.



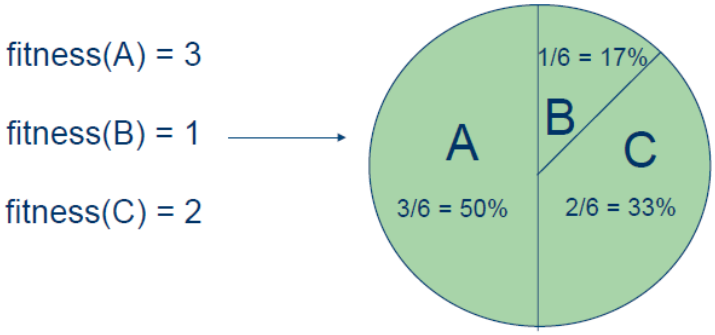
Slika 7 Fenotip i genotip [5]

Skup genotipova čini populaciju. Često je veličina populacije konstantna i ne mijenja se tijekom procesa evolucije, a njezina raznolikost reflektira se brojem različitih rješenja (jedinki) koji ju čine.

3.1.2. *Selekcija roditelja*

Velik udio komponenata evolucije su stohastičke, primjerice, tokom selekcije se najbolje jedinke ne biraju deterministički. Stoga, čak i slabije jedinke dobiju priliku preživjeti i križati se. Mehanizam izbora roditelja može se implementirati na više načina, a onaj najjednostavniji je možda pomoću ruletnog pravila. Naime, svaka jedinka u populaciji zauzima postotak na ruletnom kolu proporcionalan ocijeni fitnesa. Iz slike 8 može se zaključiti da će jedinke s boljom ocjenom fitnesa imati veću vjerojatnost postati roditelj, međutim uvijek postoji mogućnost da i jedinke sa slabijim rezultatima budu izabrane. Stohastički mehanizam selekcije sprječava zapinjanje algoritma u lokalnim optimumima.

Uvođenjem elitizma može pomoći u očuvanju jedinki s najboljim rezultatima. Time se može osigurati da kvaliteta rješenja ne pada tokom procesa evolucije.

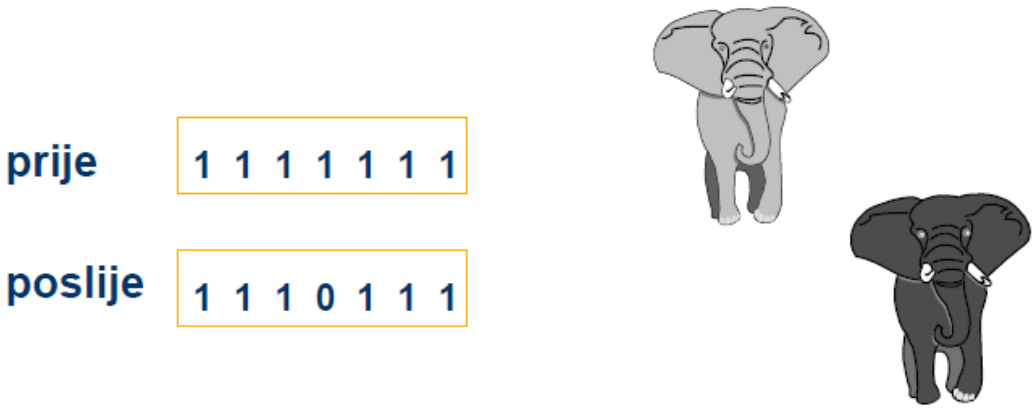


Slika 8 Ruletno pravilo [5]

3.1.3. *Varijacijski operatori*

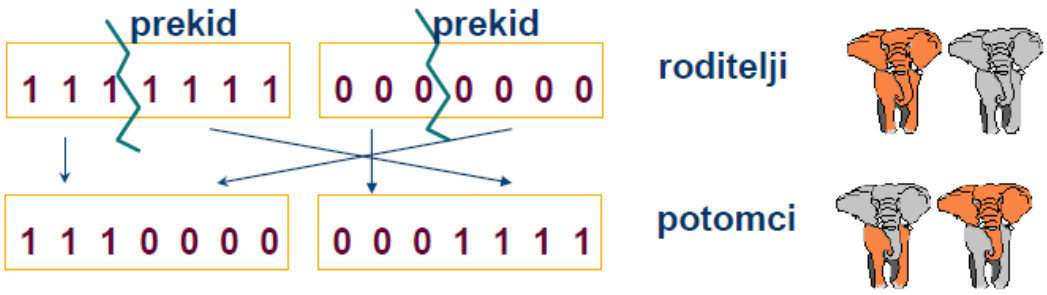
Jedinke koje su izabrane za roditelja podlijeći će križanju i mutaciji u svrhu održavanja raznolikosti unutar populacije.

Mutacija je operator koji se primjenjuje na jednoj jedinci u svrhu male promjene njezinog genotipa, a rezultira nastajanjem jedne nove jedinke. Također je stohastički proces što znači da su nastale promjene sasvim slučajne, a ne uzrokovane analizom danog genotipa u svrhu njegova poboljšanja.



Slika 9 Mutacija [5]

Rekombinacija je operator koji se primjenjuje na dva ili više odabranih kandidata (roditelja) te se kao rezultat dobiva jedan kandidat. Križanje roditelja je također stohastički proces u kojem se nasumično biraju dijelovi genoma koji se kopiraju u potomka.

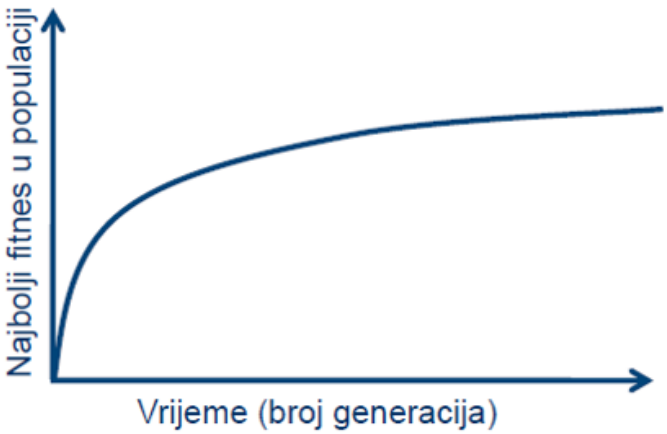


Slika 10 Rekombinacija [5]

Selekcija, mutacija i rekombinacija pokazale su se vrlo efikasnim metodama za optimizaciju uz vrlo jednostavnu računalnu obradu, a njihovo kombiniranje generalno vodi ka poboljšanju fitnesa kroz generacije.

3.1.4. *Funkcija fitnesa*

Fitnes funkcija pruža heurističku procjenu kvalitete rješenja. Ona čini osnovu za selekciju i time olakšava proces poboljšanja. Zapravo je to funkcija ili postupak kojim se dodjeljuje mjera kvalitete genotipovima. Na proces evolucije se može gledati kao na optimizaciju ili barem aproksimaciju funkcije fitnesa jer se tokom vremena sve više približava optimalnim vrijednostima (Sika 11).



Slika 11 Fitnes kroz evoluciju [5]

3.2 Neuroevolucija

Neuroevolucija (NE) je umjetna evolucija neuronskih mreža koja se provodi primjenom genetskih algoritama. NE analizira prostor ponašanja s ciljem pronalaska mreže koja uspješno rješava zadatak. To ju čini bržom i efikasnijom metodom od tradicionalnih kod problema s kontinuiranim i višedimenzionalnim prostorima stanja.

Kod tradicionalnih NE pristupa topologija koja će se evoluirati bira se prije početka samog eksperimenta. Najčešće je to arhitektura s jednim skrivenim slojem u kojem je svaki neuron povezan sa svakim ulaznim i svakim izlaznim neuronom. Evolucija pretražuje prostor težina u potpuno povezanim mrežama tako da dopušta razmnožavanje mreža koje pokazuju dobre rezultate. Taj se prostor težina istražuje križanjem vektora težina i mutacijom težina unutar pojedinačne mreže. Evidentan cilj NE mreža s fiksnom topologijom jest optimizirati težine koje utječu na funkcionalnost mreže.

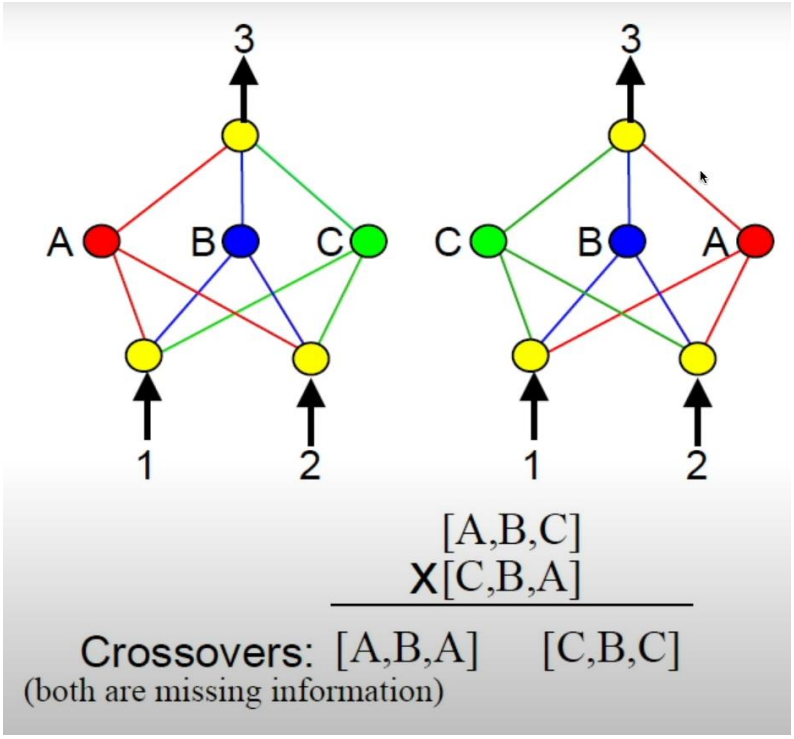
Međutim, težine nisu jedini aspekt neuronskih mreža koji doprinosi njihovu ponašanju. Topologija, odnosno struktura, također utječe na njihovu funkcionalnost. NE metode moraju se osvrnuti na sljedeće pitanje: Može li se evolucijom topologije mreže, uz postojeću evoluciju težina, ostvariti prednost pored evolucije težina mreža fiksne topologije? Gruau et al. (1996) tvrdi da evolucija strukture mreže štedi vrijeme koje bi čovjek potrošio na prosudbi najprikladnije topologije mreže za pojedinačne NE probleme [7]. Iako gotovo svaki sustav NE s fiksnom topologijom koristi potpuno umreženi skriveni sloj, odluka o količini potrebnih neurona u skrivenom sloju je proces pokušaja i promašaja.

Inkrementalna evolucija strukture nailazi na nekoliko tehničkih prepreka. Treba utvrditi postoji li genetska reprezentacija koja dopušta da se različite topologije križaju na smislen način. Također treba pronaći način kako da se topološka inovacija kojoj je potrebno nekoliko generacija za optimizaciju zaštiti, tako da ne nestane iz populacije preuranjeno. Nadalje, poželjno je minimizirati topologiju bez uvođenja zasebne funkcije dobrote koja mjeri složenost topologije.

U devedesetim godinama prošlog stoljeća razvila se nekolicina sustava koji su namijenjeni za evoluciju arhitekture neuronske mreže i težina. Te metode obuhvaćaju niz ideja o tome kako bi TWEEAN (engl. *Topology and Weight Evolving Artificial Neural Networks*) trebale biti implementirane.

3.2.1. *Problem permutacija*

Možda i najveći problem za koji TWEEAN metode moraju dokučiti rješenje jest problem permutacija (engl. *Competing coventions problem*). Na ovu prepreku nailazi se u trenutku križanja dviju jedinki. Čak ako se križaju jedinke iste topologije, ne može se sa sigurnošću tvrditi gdje se koja funkcija odvija u neuronskoj mreži. Svaka neuronska mreža može evoluirati tako da se određene funkcije odvijaju na drugačijem mjestu u odnosu na druge mreže. Drugim riječima, postoji više od jednog načina na koji se može izraziti rješenje optimizacije neuronske mreže. Kada genomi koji prikazuju isto rješenje nisu istovjetno kodirani, križanjem bi dobili nepotpune odnosno nepoželjne potomke. Na slici 12 prikazan je jednostavan primjer mreže s tri skrivena neurona. Unatoč tome što obje mreže izvode istu funkciju, njihovi skriveni čvorovi pojavljuju se u različitom redoslijedu što ih čini nekompatibilnima za proces križanja. Prikazane mreže samo su dvije od šest mogućih (3! = 6) permutacija redoslijeda skrivenih čvorova.



Slika 12 Problem permutacije [7]

Kad se jedna od permutacija križa s drugom, postoji mogućnost da se izgubi dio informacija. Slika 12 prikazuje da se križanjem [A, B, C] i [C, B, A] izgubila trećina informacija koju su posjedovali roditelji (dobiva se [A, B, A] odnosno [C, B, C]).

Problem se može dodatno zakomplicirati tako što TWEANN mreže mogu izvoditi vrlo slična rješenja koristeći potpuno različite topologije (čak i genome potpuno različite veličine).

3.2.2. *Specijacija*

Inovacije se u TWEANN metodama uvode dodavanjem novih struktura postojećim mrežama pomoću mutacije. Često se dodavanjem nove strukture isprva narušava fitnes mreže. Primjerice, dodavanjem nove sinapse može se smanjiti dobrota prije nego što je težina uopće dobila priliku optimizirati se. Postoji vrlo mala vjerojatnost da će novo uvedeni čvor ili sinapsa doprinijeti korisnom funkcijom u trenutku njihove implementacije. Nažalost, zbog inicijalnog gubitka dobrote inovativne strukture nemaju velike izglede za preživljavanje unutar populacije dovoljno dugo da bi dobile priliku za optimizaciju. Stoga je potrebno na neki način zaštititi mreže sa strukturnim inovacijama kako bi ju imale priliku iskoristiti. Kada bi ih se izoliralo i kada bi se stvorila zasebna vrsta, mogle bi dobiti šansu optimizirati iste prije nego što bi se morale natjecati na razini cijele populacije.

Specijacija podrazumijeva uvođenje funkcije kompatibilnosti koja bi uspoređivala genotipove i odlučivala može li ih se smjestiti unutar iste vrste. Teško je formulirati takvu funkciju kompatibilnosti među mrežama raznovrsnih topologija, što je i razlog zašto se specijacija izbjegavala u TWEANN metodama. Problem permutacija čini mjerenje kompatibilnosti još težim za rješavanje s obzirom da mreže koje izvode istu funkciju mogu biti potpuno različite strukture.

3.2.3. *Inicijalna populacija i topološka inovacija*

U mnogim TWEANN sustavima inicijalna populacija zapravo je kolekcija nasumično odabranih topologija čime se osigurava raznovrsnost topologija od samog početka. Međutim, takav pristup nailazi na nekolicinu problema. Naime, ukoliko se početna populacija sastoji od nasumično izabranih topologija, velika je vjerojatnost da one već sadrže mnoge nepotrebne čvorove i težine. Time se povećava i broj parametara koje je potrebno pretraživati, međutim, teži se upravo suprotnom. Poželjno je minimizirati rješenja, a time i broj parametara. Nastoji se izbjeći implementacija nove fitnes funkcije koja bi kažnjavala jedinke s većom topologijom.

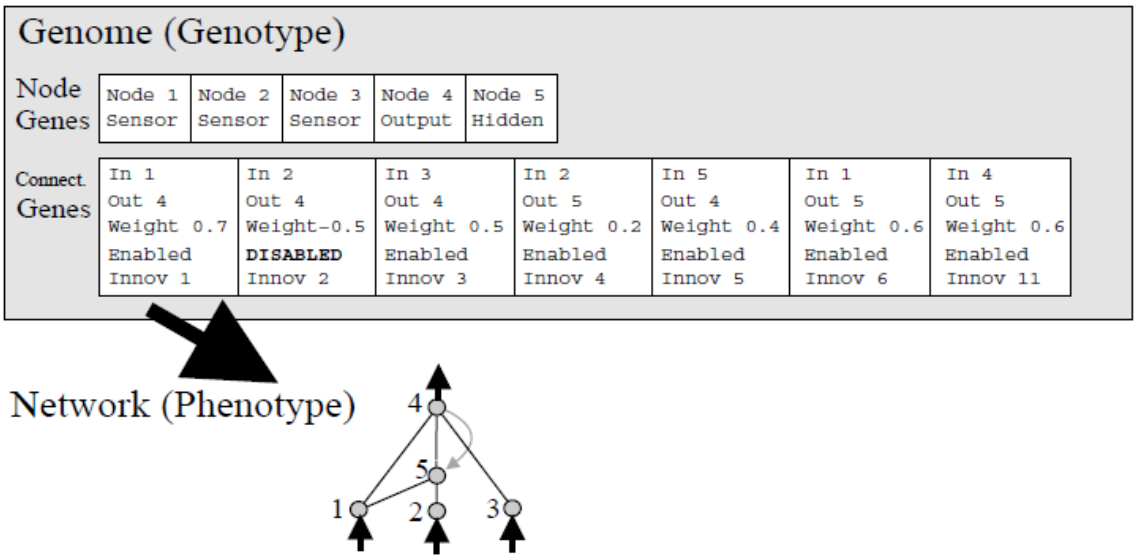
4. NEUROEVOLUCIJA PROMJENJIVIH TOPOLOGIJA (NEAT)

Kenneth O. Stanley predstavio je metodu pod akronimom NEAT (engl. *NeuroEvolution of Augmenting Topologies*) u svom radu „Evolving Neural Networks through Augmenting Topologies“ 2002. godine. NEAT se ističe u usporedbi s ostalim metodama NE zbog mogućnosti križanja različitih topologija, zaštite strukturnih inovacija korištenjem speciacije te inkrementalnog proširenja strukture počevši od onih minimalnih. Zbog smanjenja dimenzija prostora težina, NEAT algoritam postiže značajne rezultate u brzini učenja.

4.1. Kodiranje genoma

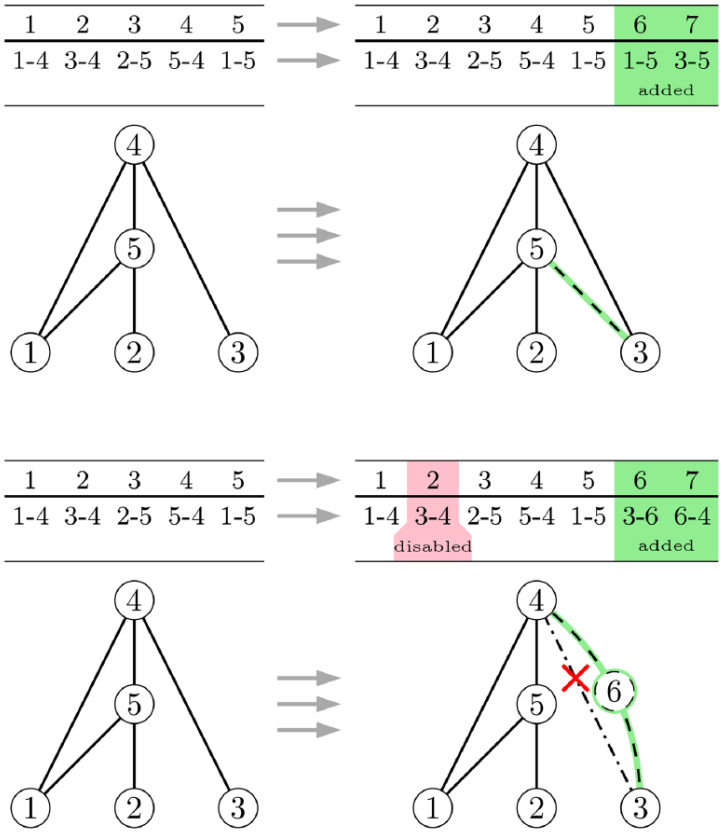
Intuicija koja stoji iza NEAT algoritma povezana je s izazovom predstavljanja različitih struktura. Njihove reprezentacije se neće nužno podudarati. Ponekad su genotipovi različitih veličina, a ponekad geni koji se nalaze na istim pozicijama, a različitim kromosomima, mogu imati potpuno različita svojstva. Osim toga, geni koji sadrže ista svojstva mogu se pojaviti na različitim pozicijama, na drugačijem kromosomu. Kad bi se novi genetski materijal nasumično pozicionirao unutar postojećeg genoma bez ikakve naznake o kojem se genu radi, život kakav znamo vjerojatno ne bi uspio zbog problema permutacije. Dakle, trebao se održati nekakav red tokom križanja. Kod umjetnih neuronskih mreža ne može se lako utvrditi podudarnost izravnom strukturnom analizom. NEAT algoritam koristi porijeklo koje dva gena dijele kao dokaz njihove podudarnosti.

Genetsko kodiranje unutar NEAT-a osmišljeno je na način da se odgovarajući genetski materijali lako križaju. Pri tome genomi linearno reprezentiraju mrežnu povezanost. Svaki genom jedinke neuronske mreže sadrži dvije liste (Slika 13). Prva lista sadrži i opisuje ulazne, skrivene i izlazne čvorove, dok druga sadrži informacije o sinapsama. Sinapse su definirane ulaznim i izlaznim čvora koje spajaju, težinom, aktivnosti gena te brojem inovacije.



Slika 13 Genotip NEAT jedinke [7]

Mutacija može utjecati na promjenu i težine i strukture mreže. Dok težine mutiraju kao i u svakom drugom sustavu NE, mutacije strukture mreže mogu se manifestirati na više načina. Svakom se mutacijom proširuje veličina genoma dodavanjem gena. Kada nastupi mutacija dodavanja sinapse, ona se u genom upisuje kao novi gen koji povezuje dva prethodno nepovezana čvora s nasumično odabranom težinom. Kada nastupi mutacija dodavanja novog čvora, jedna od postojećih sinapsi se dijeli, a novi čvor dolazi na njezino mjesto. Dakle, kao što se može vidjeti na slici 14, postojeća sinapsa se deaktivira, ali ostaje sadržana u genomu, a genom se proširuje s dvije nove sinapse nastale iz postojeće. Težina nove sinapse koja se dovodi novom čvoru poprima vrijednost 1, dok izlazna sinapsa poprima težinu deaktivirane sinapse. Ova metoda nastala je u svrhu minimiziranja inicijalnog utjecaja mutacije na dobrotu mreže. Također, uz specijaciju, na ovaj se način osigurava dovoljno vremena za optimizaciju mreže kako bi se nova struktura mogla iskoristiti.

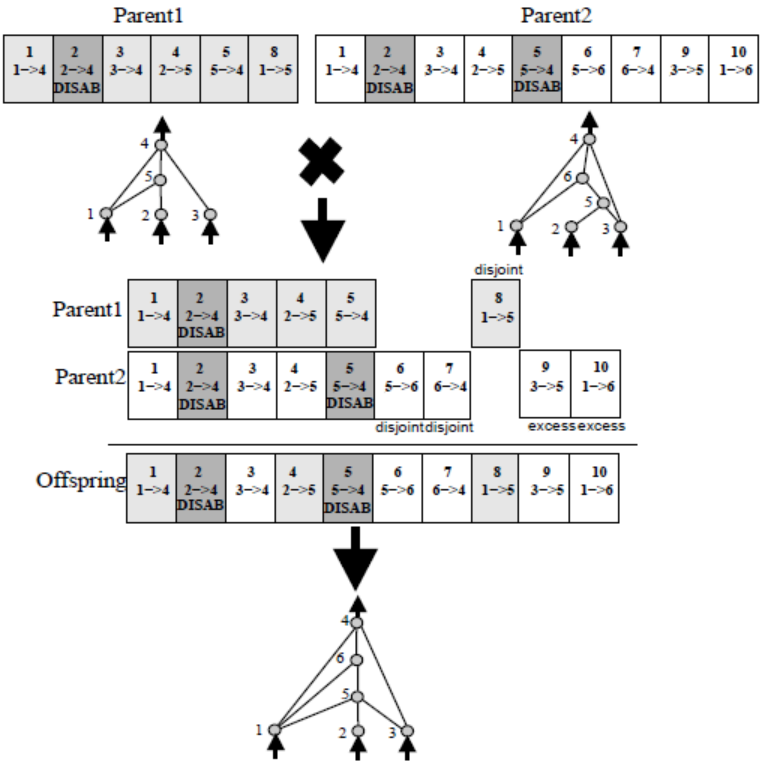


Slika 14 Prikaz mutacije u NEAT jedinci [8]

4.1.1. *Uvođenje broja inovacije*

Prethodno spomenuti brojevi inovacije su oznake gena koje imaju važnu ulogu i čine NEAT vrlo uspješnom NE metodom. Naime, geni koji dijele porijeklo moraju imati istu strukturu (vrlo vjerojatno s različitim vrijednostima težina) s obzirom na to da su izvedeni iz zajedničkog pretka u prošlosti. Dakle, sustav treba zapamtiti porijeklo svakog gena u sustavu što zahtjeva vrlo malo računalne obrade. Svaki put kada se uslijed mutacije pojavi novi gen, uveća se globalni brojač inovacije i dodjeljuje se upravo tom genu (kronološki prikazuje svaki gen u sustavu). Važno je napomenuti da se naslijeđeni geni kopiraju u potomka s nepromijenjenim brojem inovacije, upravo kako bi se moglo pratiti njihovo porijeklo. Naime, prilikom križanja evidentno je postoje li geni s istim brojem inovacije. Algoritam ih tada može poredati kao na slici 15. Ukoliko algoritam pri križanju naiđe na gen koji postoji u jednom, ali izostaje u drugom genomu radit će se o neuparenom (engl. *disjoint*) ili suvišnom (engl. *excess*) genu, ovisno o njegovoj poziciji unutar roditeljskog genoma. Oni predstavljaju strukturu koja izostaje u drugom genomu. Kod kopiranja gena u potomka, nasumično se odabire jedan od podudarnih

gena roditelja koji će se kopirati, dok se ostali genetski materijal uzima od roditelja s većim fitnesom.



Slika 15 Prikaz križanja NEAT jedinki [8]

4.1.2. *Zaštita inovativnih struktura pomoću specijacije*

Populacija ne može samostalno održati strukturne inovacije jer se manje strukture optimiziraju brže od onih većih te radi inicijalnog pada fitnesa mreže pri implementaciji novih čvorova i sinapsa. No, specijacijom populacije mogu se zaštiti. S obzirom na to da NEAT koristi informacije o porijeklu gena, specijacija se može relativno lako provesti nad populacijom.

Kako bi se mreže sortiralo prema kriteriju sličnosti topologije, potrebno je na neki način odrediti poklapanje njihovih genoma. Broj gena koji su višak ili se ne podudaraju određuju kompatibilnost dvaju genoma. Što se oni manje podudaraju, to su manje kompatibilni. Postoji jednadžba prema kojoj se mjeri kompatibilnost δ kako slijedi:

$$\delta = \frac{c_1}{N}E + \frac{c_2}{N}D + c_3\overline{W} \tag{4.1}$$

Dakle, možemo mjeriti kompatibilnost kao linearnu kombinaciju broja suvišnih gena E i neuparenih gena D te kao prosjek razlike težina podudarnih gena $\overline{W}$ uključujući neaktivne gene. Koeficijenti c_1, c_2 i c_3 omogućuju prilagodbu jednadžbe ovisno o važnosti navedena tri faktora.

Faktor N u jednadžbi 4.1 predstavlja broj gena sadržanih unutar većeg genoma te se koristi u svrhu normalizacije veličine genoma (može biti 1 za relativno male genome npr. do 20 gena). Mjera kompatibilnosti δ omogućuje provođenje specijacije uz pomoć prag kompatibilnosti δt. Održava se uređeni popis vrsta tako što se u svakoj generaciji genomi sekvencijalno raspoređuju u vrste. Predstavnik svake vrste jest nasumično odabran genom iz prethodne generacije te vrste. Dakle, novonastali genom g u trenutnoj generaciji sortira se u vrstu čiji je reprezentativni genom kompatibilan s njime ($\delta_g < \delta_t$). Ukoliko se genom g ne poklapa niti s jednom postojećom vrstom ($\delta_g > \delta_t$), stvara se nova vrsta s g kao reprezentativnim genomom.

NEAT jedinke unutar vrste koriste eksplicitno dijeljenje fitnesa, populacija unutar jedne vrste ne može si priuštiti velik broj jedinki, unatoč tome što većina njih ima dobre rezultate. Time se sprječava dominacija jedne vrste u odnosu na populaciju u potpunosti što je temeljni zadatak specijacije. Prilagođeni fitnes jedinke računa se u odnosu na δ od svake jedinke unutar populacije.

4.1.3. *Inkrementalan rast strukture kroz evoluciju*

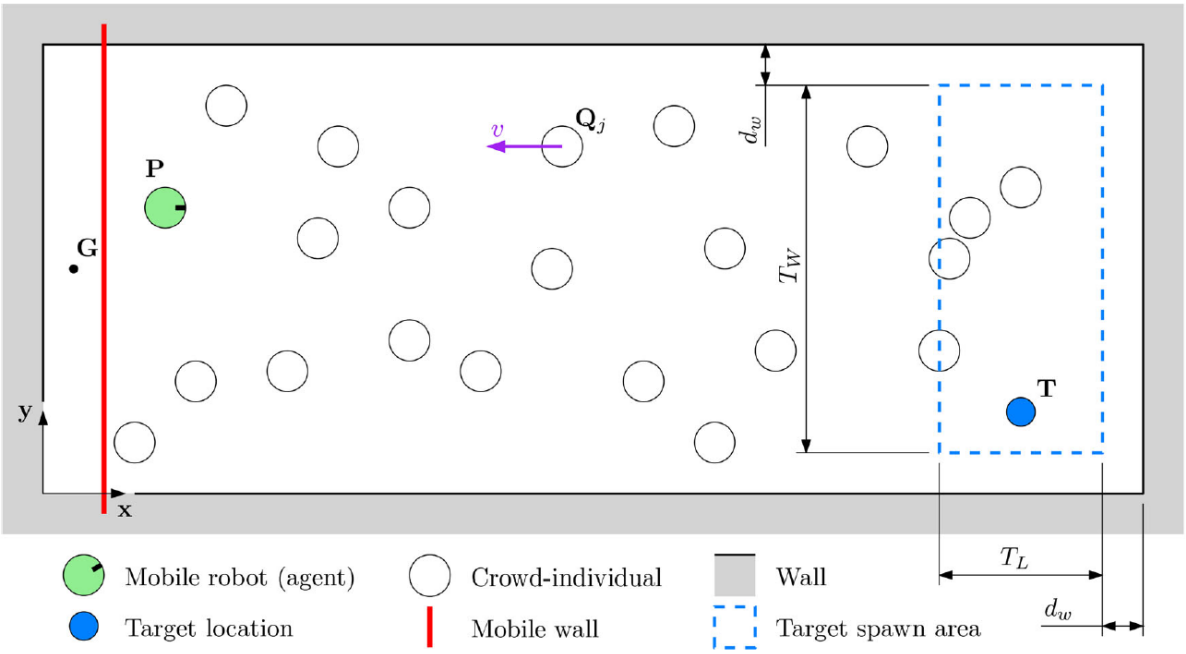
Inicijalna NEAT populacija mreža vrlo je jednostavna jer ne sadrži skrivene čvorove, već se struktura razvija kroz evoluciju u smjeru boljih rezultata. Time se postiže pretraživanje unutar najniže moguće dimenzije težina (minimizira se prostor pretraživanja) tijekom svih generacija. Ovaj korak je ključan u ostvarivanju drastičnih poboljšanja u performansama mreža. Upravo specijacija, odnosno zaštita i održavanje inovativnih mreža unutar populacije, NEAT algoritmu omogućava takav pristup za razliku od ostalih TWEANN metoda.

Algoritam započinje pretragu s ujednačenom populacijom neuronskih mreža bez skrivenih neurona (svi su ulazni neuroni direktno povezani s izlaznim). Kako nastupaju mutacije strukture, tako se uvode nove koje preživljavaju isključivo ako se pokažu korisnima putem ocjene fitnesa.

4.2. Primjer implementacije u mobilnoj robotici

Da je NEAT prikladna metoda za optimizaciju navigacije mobilnih robota dokazuju i neka istraživanja. Tema koja privlači sve više interesa jest navigacija autonomnih sustava u okruženju s mnogo zapreka, kao što su primjerice okruženja s prolaznicima. Navigacija kroz gomile ljudi je iz sigurnosnih razloga težak zadatak za mobilne robote. Primjenom NEAT metode na kontroleru mobilnog robota u simuliranom okruženju uspjelo se dokazati da

inteligentan kontroler može parirati ljudskom navigiranju robota u takvim okruženjima [8]. U navedenom istraživanju hipotetsko autonomno vozilo opremljeno je 2D laserskim skenerom (LiDAR). Kognitivni model robota bazira se na neuronskim mrežama koje služe za navigaciju kroz koridor ispunjen ljudima u pokretu (Slika 16). Percepcija okoline visoke razlučivosti nužno zahtjeva velik broj zraka što direktno utječe i na složenost neuronskih mreža. Naime, broj ulaznih neurona mreže povezan je s brojem zraka, stoga je primjena NEAT algoritma u ovom slučaju vrlo povoljna. Optimalna topologija mreže pronalazi se kroz evoluciju, što korisniku olakšava zadaću pronalaska idealne topologije unaprijed.



Slika 16 Primjer NEAT primjene [8]

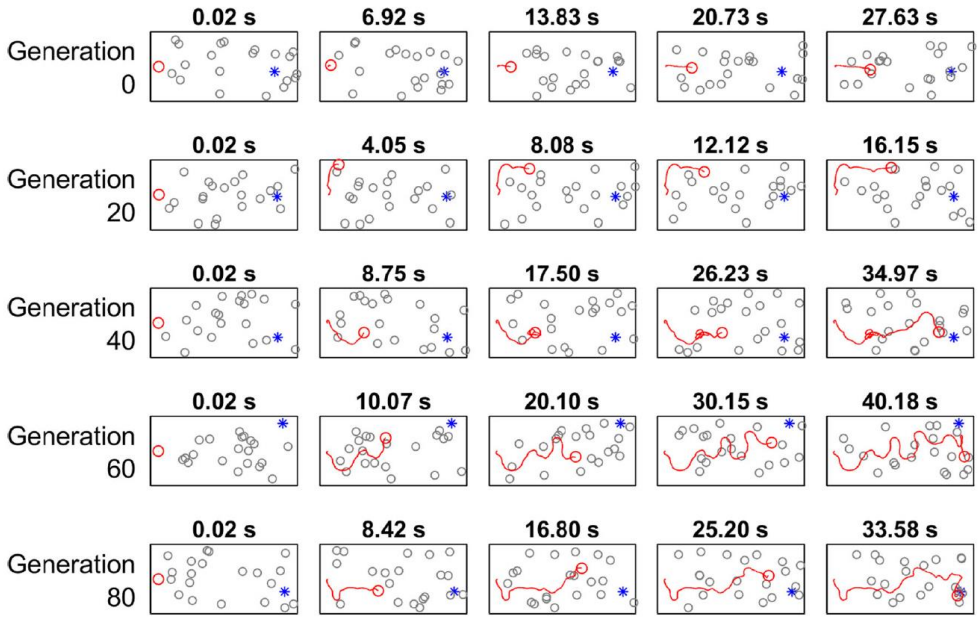
Evolucija se u ovom slučaju provodi na NEAT populaciji od 600 jedinki. Raznolikost u populaciji ključna za postizanje značajnih rezultata budući da NEAT metoda implementira i specijaciju i elitizam. Broj generacije seže od 100 do 1000. Preliminarna testiranja uključivala su velik broj generacija (1000) zbog kompleksnosti zadatka i mnogih drugih nepoznanica kao što je konvergencija evolucije. Pokrenuto je 13 evolucijskih procesa koji su sadržavali fiksne i promjenjive parametre iz tablice na slici 17.

Table 1 Parameters space

Parameter	Unit	Values
Number of rays n		15, **30**, 60, 90
Lookback steps s_{lb}		0, 2, 4, 6, 8, **10**
Crowd mean speed $\bar{u}$	ms^{-1}	0.25, **0.50**, 0.75, 1.00, 1.20, 1.50
Crowd size m		1, 5, 10, **15**, 20, 25
v_{wall}	ms^{-1}	0.15
L	m	18
W	m	8
R	m	0.6
R_R	m	0.2
R_{range}	m	3
n_{gen}		100
n_{pop}		600

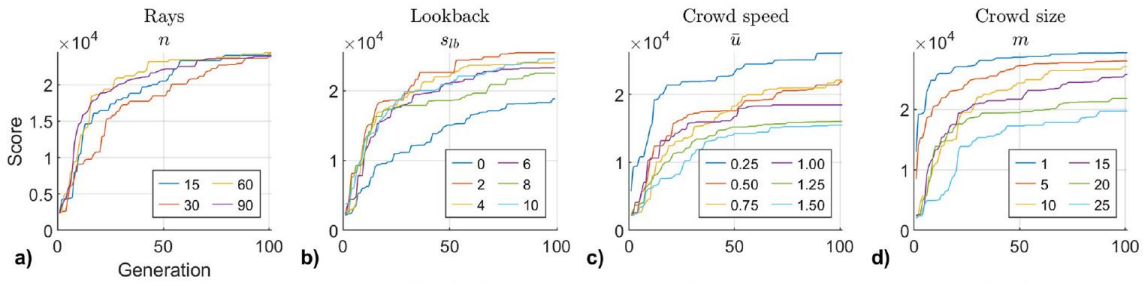
Slika 17 Parametri primjera [8]

Nakon pokretanja simulacija ustanovljeno je da se proces evolucije ubrzano pojačava kroz prvih 100 generacija, te do oko 500-te generacije doseže svoj maksimum. Na slici 18 vidi se da robot dolazi vrlo blizu svom cilju u 40. generaciji, 20 generacija kasnije ne uspijeva, te u 80-oj generaciji dolazi do svog cilja.



Slika 18 Rezultati primjera po generacijama [8]

Parametri koji su varirani kroz različite simulacije s namjerom ispitivanja izvedbe NEAT algoritma (broj zraka, brzina hodanja, količina ljudi…) imaju velik utjecaj na brzinu konvergencije i sveukupnu ocjenu fitnesa, no kroz prihvatljiv broj generacija postigli su se zadovoljavajući rezultati (Slika 19).



Slika 19 Ponašanje fitnesa kroz promjenu parametara [8]

5. IMPLEMENTACIJA NEAT ALGORITMA

5.1. Python

Python je objektno orijentirani programski jezik visoke razine koji se može koristiti za strojno učenje, duboko učenje i implementaciju neuronskih mreža. Zbog njegove fleksibilnosti i jednostavnosti sintakse dostupni su mogi resursi, biblioteke, korisnički priručnici itd. Pogodan je za početnike, ali i za programere raznih iskustvenih razina.

Za potrebe ovog rada korišten je *Anaconda* softverski paket koji sadrži nekolicinu integriranih okruženja koji koriste *Python* 3 jezik. Za implementaciju NEAT algoritma u *PyCharm* okruženju bilo je potrebno instalirati i nekoliko bibilioteka, kao što su *neat-python* i *pygame*.

5.1.1. *Neat-python*

Ključna biblioteka koja omogućuje implementaciju NEAT algoritma je *neat-python*. Pomoću ovog paketa moguće je primijeniti evoluciju topologije mreža, odnosno, tijekom generacija je moguće razvijati strukture neuronske mreže novim čvorovima i vezama. Definiranjem populacije, uvođenjem konfiguracijske datoteke te primjenom već navedenih mehanizama evolucije može se implementirati i analizirati učinak ovog algoritma. [9]

5.1.2. *PyGame*

Simulacija je pokrenuta pomoću pygame paketa. Ova se biblioteka obično koristi za razvijanje 2D video igara ili multimedijalnih aplikacija, ali zbog jednostavnog rada s grafikom i mogućnosti detekcije kolizije između objekata idealan je izbor za implementaciju simulacije mobilnog robota. [10]

5.2. Simulacija navigacije mobilnog robota pomoću NEAT algoritma

Za analizu NEAT algoritma u kontekstu mobilne robotike simulirat će se navigacija kontrolera robota. Prije same simulacije, grafički primjer jedinke i okoliš u kojem one uče upravljati kreirane su u programu *Inkscape*.

Jedinke mobilnog robota veličine 60 x 60 piksela kreirane su kao objekti klase *Robot*. Važno je napomenuti da svaka jedinka mobilnog robota predstavlja jednu neuronsku mrežu koja njome

upravlja. Broj ulaza i izlaza jednak je za svaku jedinku u populaciji. Ulazi u mrežu su podaci koje prikupljaju senzori. U ovom slučaju implementirano je 5 senzora, s razmakom od 45 stupnjeva (-90º, -45 º, 0 º, 45 º i 90 º) koji mjere udaljenost robota od ruba staze. Točnije, oni su programirani da detektiraju udaljenost od boje ruba. Radar senzora je 300 piksela.

Izlazni neuroni davat će robotu naredbe kretanja na temelju podataka koje dobivaju iz senzora. Izlazi će davati onoliko dobre instrukcije za kretanje robota koliko je neuronska mreža povoljno povezana.

```
# Izlazi mreže
for i, robot in enumerate(robots):
    output = nets[i].activate(robot.get_data())
    choice = output.index(max(output))
    if choice == 0:
        robot.angle += 10 # skretanje lijevo
    elif choice == 1:
        robot.angle -= 10 # skretanje desno
    elif choice == 2:
        if(robot.speed - 2 >= 12):
            robot.speed -= 2 # kočenje
    else:
        robot.speed += 2 # ubrzavanje
```

Slika 20 Izlazi iz mreže

Zadaća NEAT-a je takva da kroz generacije unaprjeđuje topologiju sve dok jedinka ne uspije doći do svog cilja u što bržem vremenskom roku. Algoritam će birati najbolje jedinke za roditelje u sljedećoj generaciji prema ocjeni fitnesa. Fitnes funkcija jedinke F raste proporcionalno udaljenosti koju ona prijeđe bez sudaranja. Točnije, povećava se s porastom daljine d koju robot prijeđe tijekom trajanja jedne sličice t (60 *fps*):

$$F = \frac{d}{t/60}, \tag{5.1}$$

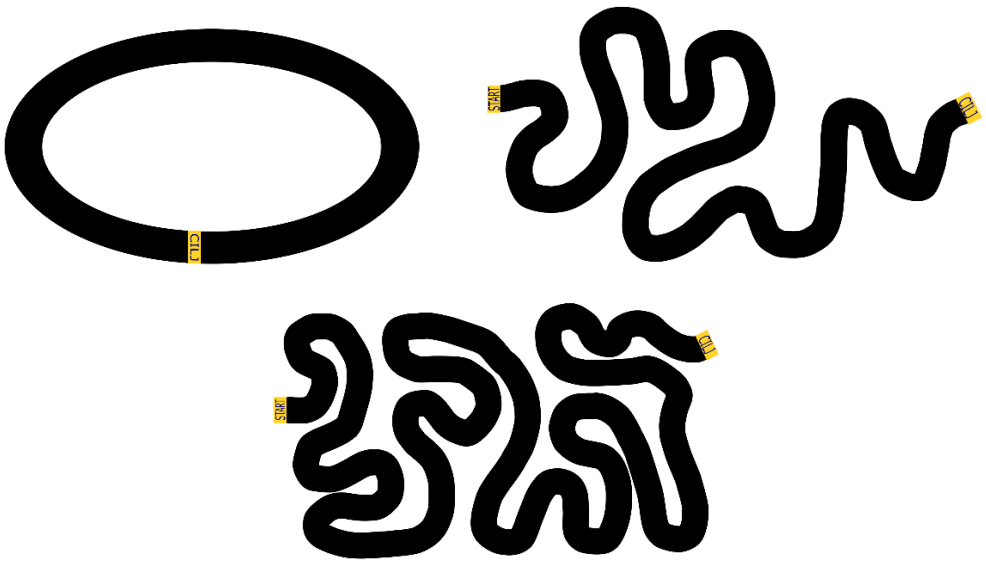
Ako neuronska mreža robota nauči povećavati svoju brzinu u prikladnom trenutku, doći će do boljih rezultata. Unutar klase robota izračunava se nagrada temeljena na udaljenosti koju je jedinka prešla, dok se tokom simulacije funkcija povećava proporcionalno nagradi koju robot dobiva (Slika 21).

```python
class Robot:  1 usage
    def get_reward(self):  1 usage
        if self.time > 0:
            return self.distance / (self.time / 60)
        else:
            return 0
```

```python
for i, robot in enumerate(robots):
    if robot.is_alive():
        still_alive += 1
        robot.update(game_map)
        genomes[i][1].fitness += robot.get_reward()
```

Slika 21 Fitnes funkcija u pythonu

Simulacija započinje učitavanjem mape koja sadrži put koji mobilni robot mora prijeći od starta do cilja. Za potrebe ovog rada kreirane su tri mape različite težine (Slika 22). Veličine su 1920 x 1080 piksela.



Slika 22 Prikaz mapa za simulaciju

Simulacija NEAT populacije kreće od nulte generacije, odnosno, simulacija započinje s populacijom potpuno povezanih neuronskih mreža bez skrivenih slojeva. Na taj način demonstrirat će se kakve rezultate NEAT postiže ovisno o težini mape.

```
if __name__ == "__main__":
    # Load Config
    config_path = "./config.txt"
    config = neat.config.Config(neat.DefaultGenome,
                                neat.DefaultReproduction,
                                neat.DefaultSpeciesSet,
                                neat.DefaultStagnation,
                                config_path)

    # Create Population And Add Reporters
    population = neat.Population(config)
    population.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    population.add_reporter(stats)
```

Slika 23 Pokretanje NEAT algoritma [11]

Broj generacija važan je podatak za analizu algoritma u odnosu na njegove parametre, osobito u kontekstu brzine konvergencije. Trajanje jedne generacije ovisi o uspješnosti populacije na zadanoj mapi – ona će trajati sve dok se posljednja jedinka ne sudari ili prođe kroz cilj, a u svrhu optimizacije evolucije te stabilnosti rezultata, ograničena je na trajanje do maksimalno 20 sekundi. Pritom se sprječava pojava „beskonačne“ simulacije te se jedinke potiče na stvaranje učinkovitijih strategija za kretanje kroz mapu.

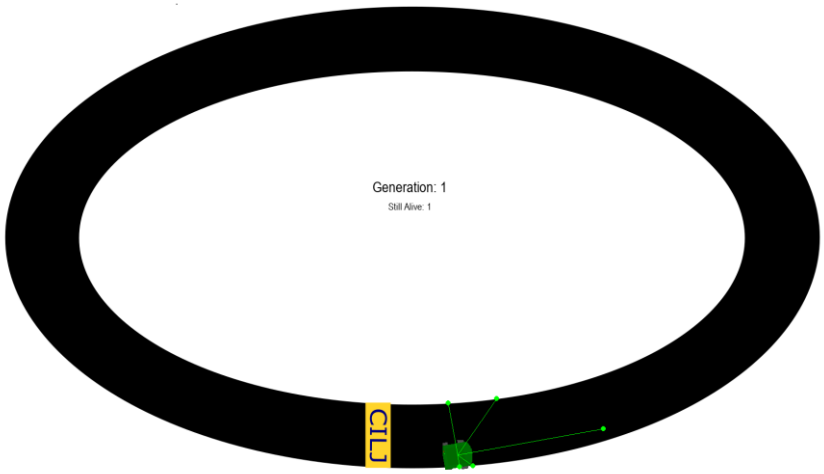
6. REZULTATI

Za analizu i grafički prikaz rezultata simulacija korišten je *visualize* [12]. Ova datoteka posebno je prilagođena za vizualizaciju *neat-python* algoritma. Nakon završetka simulacije omogućuje pregled evolucije populacije, specijacije te uz *graphviz* [13] i prikaz topologije neuronskih mreža. *Visualize* definira funkciju *plot_stats* koja daje grafički prikaz prosječnog i najboljeg fitnesa kroz generacije. Zatim, sadrži funkciju *plot_species* kojom uvodi praćenje vrsta te funkciju *draw_net* koja iscrtava neuronsku mrežu genoma.

Simulacijom će se prikazati treniranje populacije od 30 jedinki kontrolera mobilnog robota u tri različita okruženja. Naime, inkrementalnim otežavanjem mape koju robot mora proći dobiva se uvid u ponašanje NEAT algoritma ovisno o težini zadatka. Očekivano je da će konvergencija evolucije odgovarati težini staze, a kako parametri utječu na nju vidjet će se u nastavku.

6.1. Prva mapa

Prije pokretanja prve simulacije definiraju su neki od parametara u konfiguraciji NEAT algoritma. Primjerice, elitizam je podešen za 3 najbolje jedinke, a maksimalna stagnacija vrste ograničena je na 20 generacija.
Kod jednostavnijih zadataka, kao što je navigiranje robota ovom kružnom mapom, u početnoj populaciji od 30 jedinki može se dogoditi da već postoji model neuronske mreže koji će već u prvoj generaciji doći do cilja (Slika 24).



Slika 24 Prolzak robota kroz cilj u prvoj generaciji

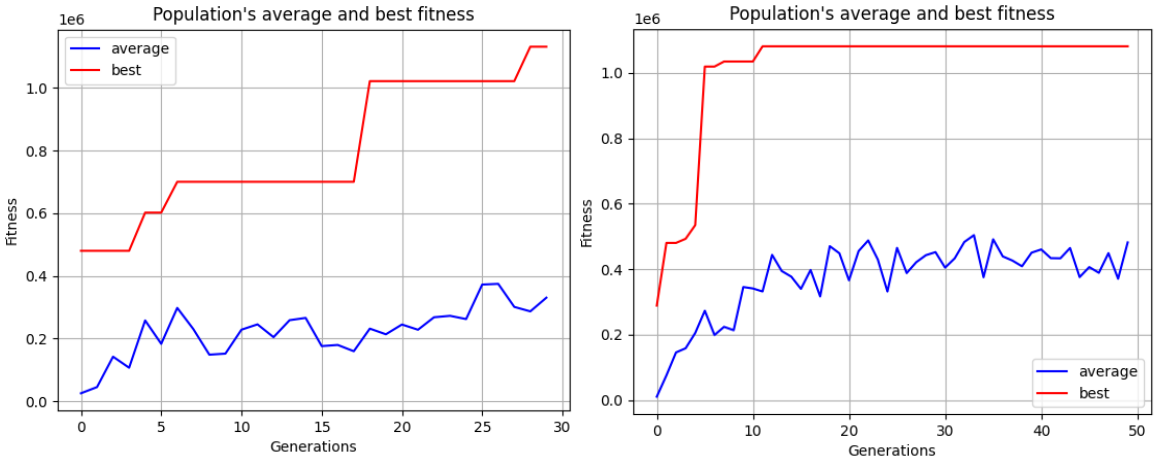
Do kraja simulacije je gotovo pola jedinki preživjelo cijelu generaciju, međutim, grafička analiza pokazala je zanimljive podatke. Na slici 25 a) može se vidjeti da kroz 30 generacija postoje stagnacije fitnesa koje nakon nekog vremena uslijede naglim skokom. Takvo što se može dogoditi kada se populacija previše homogenizira. S prethodno definiranim parametrima populacija je svedena na jednu vrstu (slika 26 a), stoga se za sljedeću simulaciju maksimalna stagnacija smanjila na 10 generacija te se uveo parametar *min_species_size* i postavio na vrijednost 1. Promjena konfiguracije je učinjena jer je raznolikost ključna za inovaciju i istraživanje novih rješenja izvan lokalnog optimuma. I u drugoj je simulaciji prva jedinka prošla kroz cilj u prvoj generaciji, a do stagnacije fitnesa došlo je već oko desete generacije i to na vrijednosti koja se u prvoj simulaciji dosegla tek prije tridesete generacije (Slika 25 b). Također je važno napomenuti da je nastala još jedna vrsta rješenja u trenutku dostizanja maksimalnog fitnesa (Slika 26 b).

Za treću simulaciju elitizam je smanjen na dvije jedinke kako bi se testirali utjecaji još nekih parametra mreže. Zanimljivo je usporediti topologiju najbolje jedinke iz prethodne dvije simulacije s topologijom iz posljednje simulacije (Slika 27). Neuronska mreža trećeg 'pobjednika' sadrži jedan skriveni čvor više, dok njezin fitnes stagnira na manjoj vrijednosti od prethodne dvije simulacije. Kroz simulaciju su se pojavile tri vrste od kojih je jedna izumrla. Na ovom relativno laganom zadatku vidi se da promjena i jednog parametra može rezultirati različitim rješenjima, ali isto tako se u svim slučajevima dolazi do cilja u prvoj generaciji.

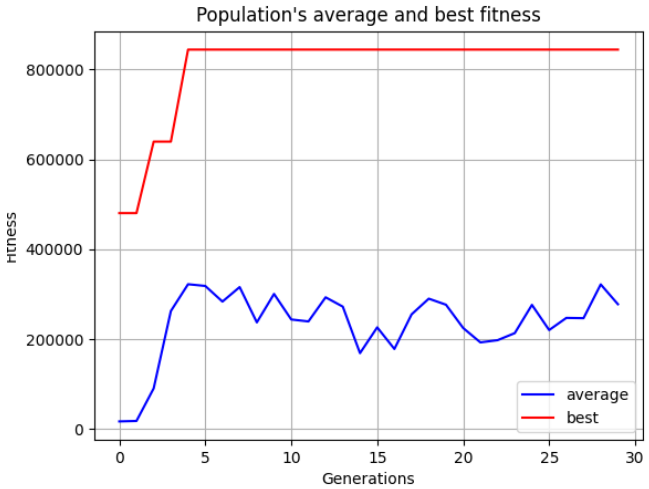
Tablica 1 Rezultati simulacija na mapi 1

	Elitizam	Maks. stagnacija vrste	Min. veličina vrste	Dolazak do cilja u generaciji	Stagnacija fitnesa u generaciji	Maks. fitnes	Broj nastalih vrsta
a	3	20	Nedefinirana	1	27	1136582	1
b	3	10	1	1	11	1138456	2
c	2	10	1	1	4	832278	3

.

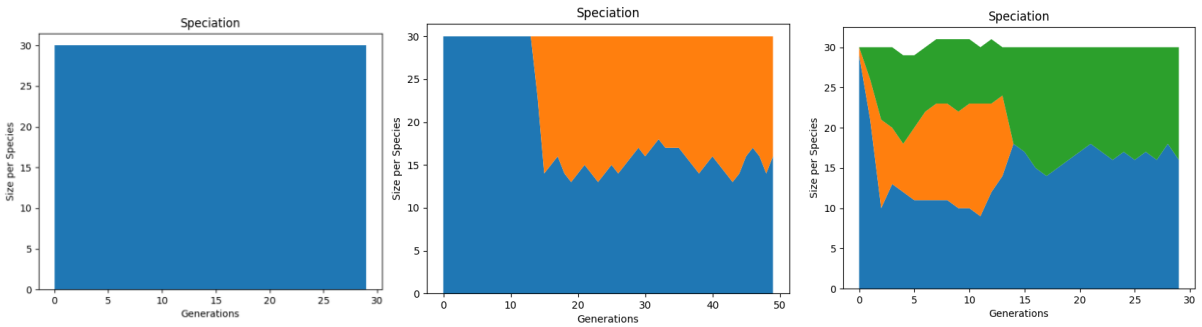


a) **b)**



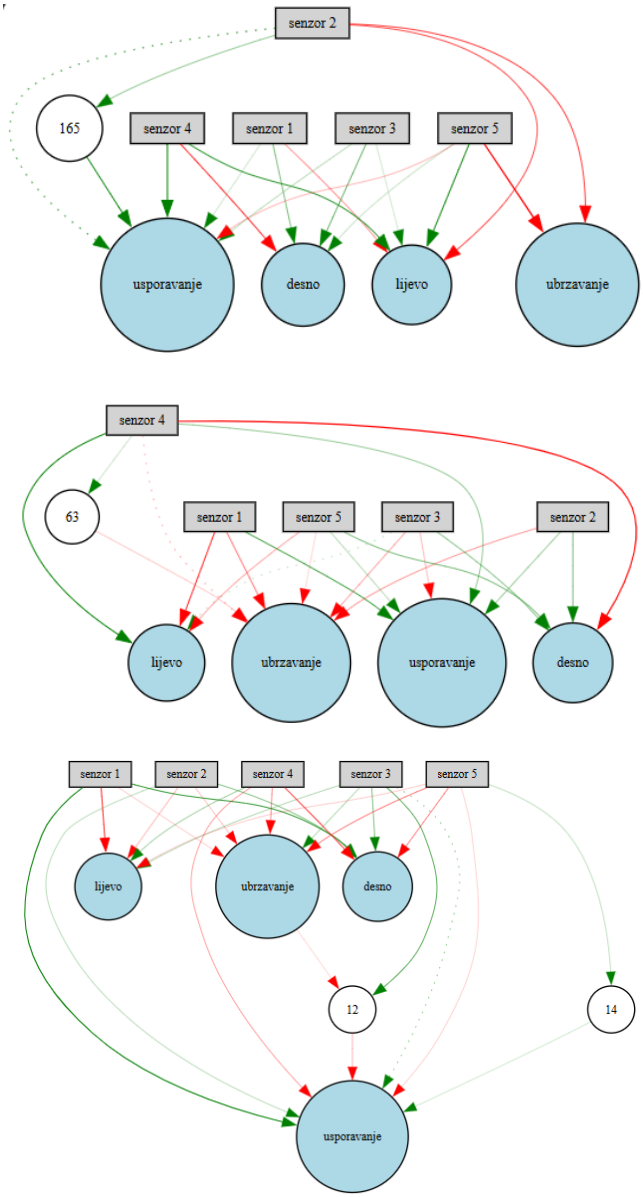
c)

Slika 25 Fitnes funkcije simulacija na mapi 1



a) **b)** **c)**

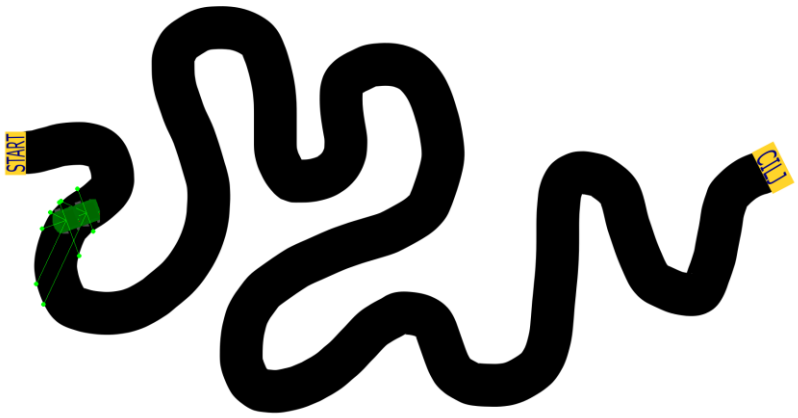
Slika 26 Specijacija za mapu 1



Slika 27 Topologije jedinki sa mape 1

6.2. Druga mapa

Za razliku od prve mape, ne može se očekivati da će u prvih nekoliko generacija robot doći do cilja, naprotiv, moguće je da robot i nakon nekoliko generacija ne prođe ni prvi zavoj (Slika 28).



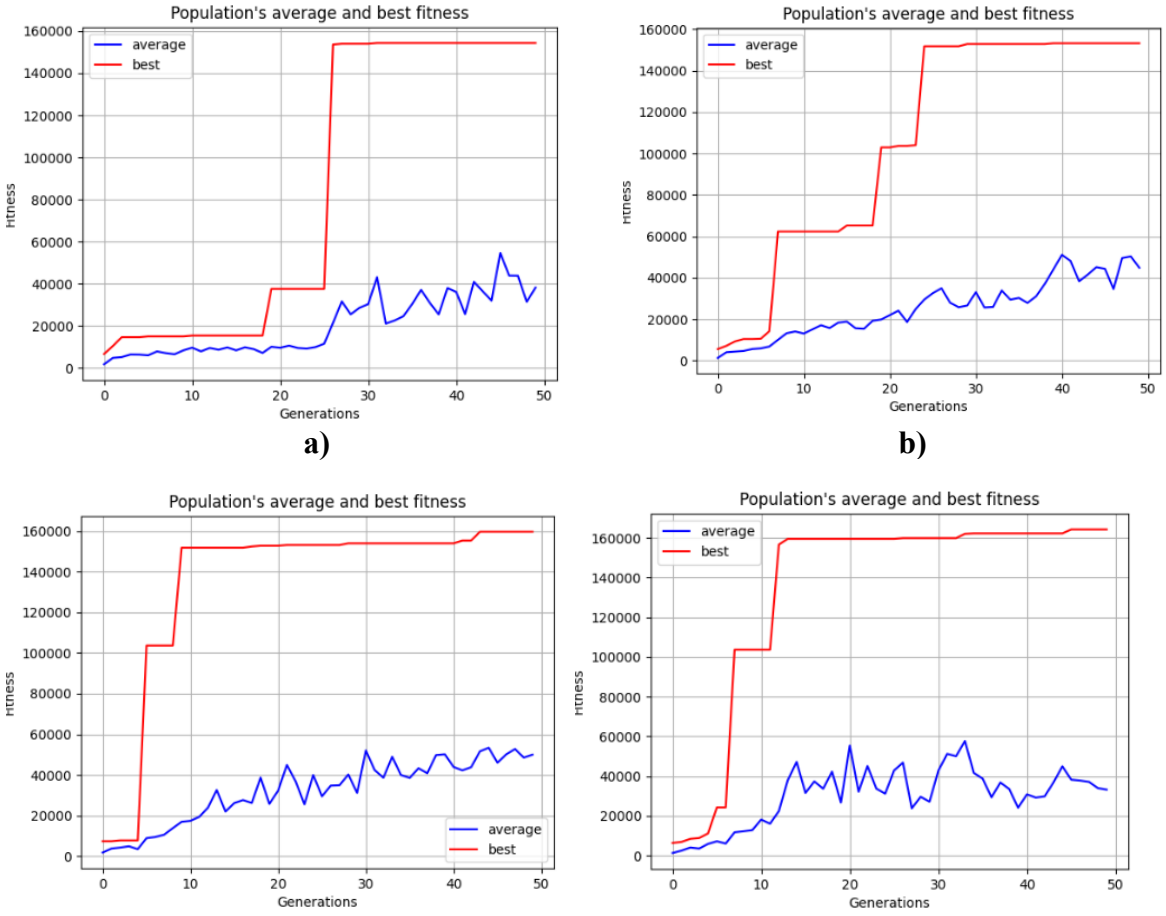
Slika 28 Preostale dvije jedinke koje će se izumrijeti u 34. generaciji

Nakon što je konfiguracija vraćena na istu kao i u prvoj simulaciji kružne mape, na zahtjevnijoj mapi robot je stigao do cilja u 26. generaciji. Međutim, tokom više generacija samo je nekoliko jedinki došlo do cilja. Fitnes je u prvoj simulaciji imao puno manje skokova nego što je to u a) slučaju za prvu mapu, a počeo stagnirati nakon 31. generacije (Slika 29 a). Kod ponovnog pokretanja, prva jedinka prošla je kroz cilj u 24. generaciji, a se fitnes stabilizirao na sličnom iznosu kao i u prethodnom pokušaju (Slika 29 b). U drugoj simulaciji došlo je i do nastanka nove vrste (Slika 30 b).

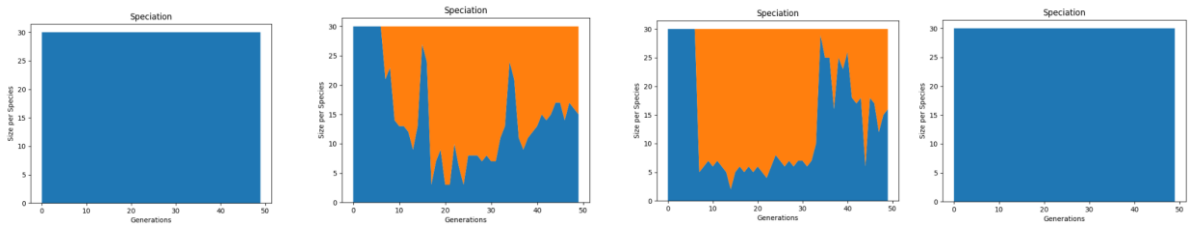
Zatim je ponovno uveden *min_species_size* te je smanjena maksimalna stagnacija. U prosjeku su simulacije s takvom konfiguracijom dolazile do cilja u ranijim generacijama. Slika 29 c) prikazuje primjer gdje se je cilj dostignut već u 9. generaciji, a do kraja simulacije su se postigli i bolji rezultati fitnesa nego u prvom slučaju. Fitnes je u tom slučaju sličan rezultatima prije uvođenja vrsne raznolikosti. U jednom od pokretanja nastala je samo jedna vrsta unatoč konfiguraciji koja potiče raznolikost (Slika 30 d), međutim, dosegao se veći iznos fitnesa nego u svim prijašnjim simulacijama. Razlog je taj što kompleksnost zadatka može utjecati na ponašanje NEAT algoritma. Također je važno imati na umu da NEAT sadrži mnoge stohastičke mehanizme koji mogu utjecati na pojavu različitih rezultata svakim pokretanjem nove simulacije. Iz tog razloga važno je napraviti dovoljan broj simulacija i analizirati ponašanje algoritma uslijed promjene parametara.

Tablica 2 Rezultati simulacija na mapi 2

	Elitizam	Maks. stagnacija vrste	Min. veličina vrste	Dolazak do cilja u generaciji	Stagnacija fitnesa u generaciji	Maks. fitnes	Broj nastalih vrsta
a	3	20	Nedefinirana	26	31	154339	1
b	3	20	Nedefinirana	24	39	153255	2
c	3	10	1	9	43	159760	2
d	3	10	1	12	45	164229	1



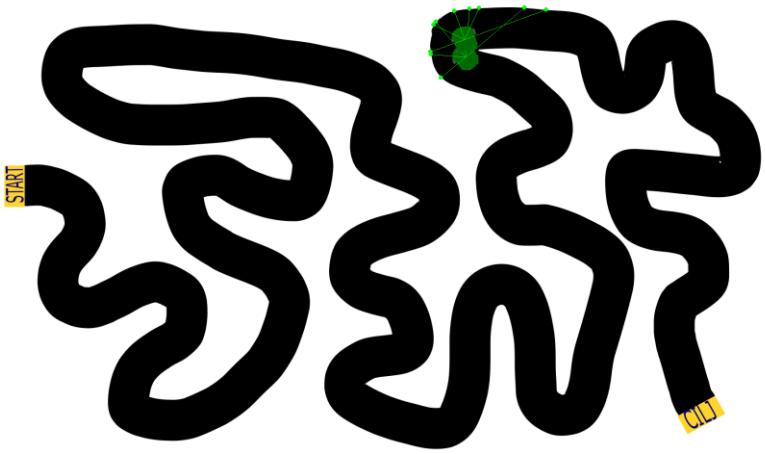
Slika 29 Fitnes funkcije simulacija na 2. Mapi



Slika 30 Specijacija za mapu 2

6.3. Treća mapa

U simulaciji najteže mape, neuronske mreže su se nastojale križati i mutirati, no zbog njezine geometrije potomci izumiru vrlo brzo nakon starta. Manje promjene u mreži ne dovode do značajnijeg napretka u prvih nekoliko generacija. U 14. generaciji simulacije jedna se jedinka izdvojila, međutim tek u 28. generaciji više jedinki prolazi kroz krajnju točku. Ponekad se broj uspješnih robota kroz generacije smanjuje, a to je posljedica elitizma koji osigurava da samo najbolji prežive i razmnože se. Unatoč većem fitnesu bržih robota, vjerojatnost preživljavanja mutirane verzije takve jedinke često je manja no što je vjerojatnost preživljavanja mutirane verzije sporijeg robota. Takvo što se može promijeniti s generacijama. Favoriziranje bržih jedinki kod kompleksnijih mapa može utjecati na brzinu konvergencije pa se u nekim simulacijama može dogoditi da populacija dođe blizu, ali ne prolazi kroz cilj unutar zadanog broja generacija (Slika 32 b). Kada bi se nastavilo otežavanje mape koje robot mora savladati, trebala bi se, osim promjene parametara, razmotriti i važnost brzine robota u odnosu na prijeđen put. Smanjenjem utjecaja brzine bi se osiguralo da robot dostigne cilj u manjem broju generacija.



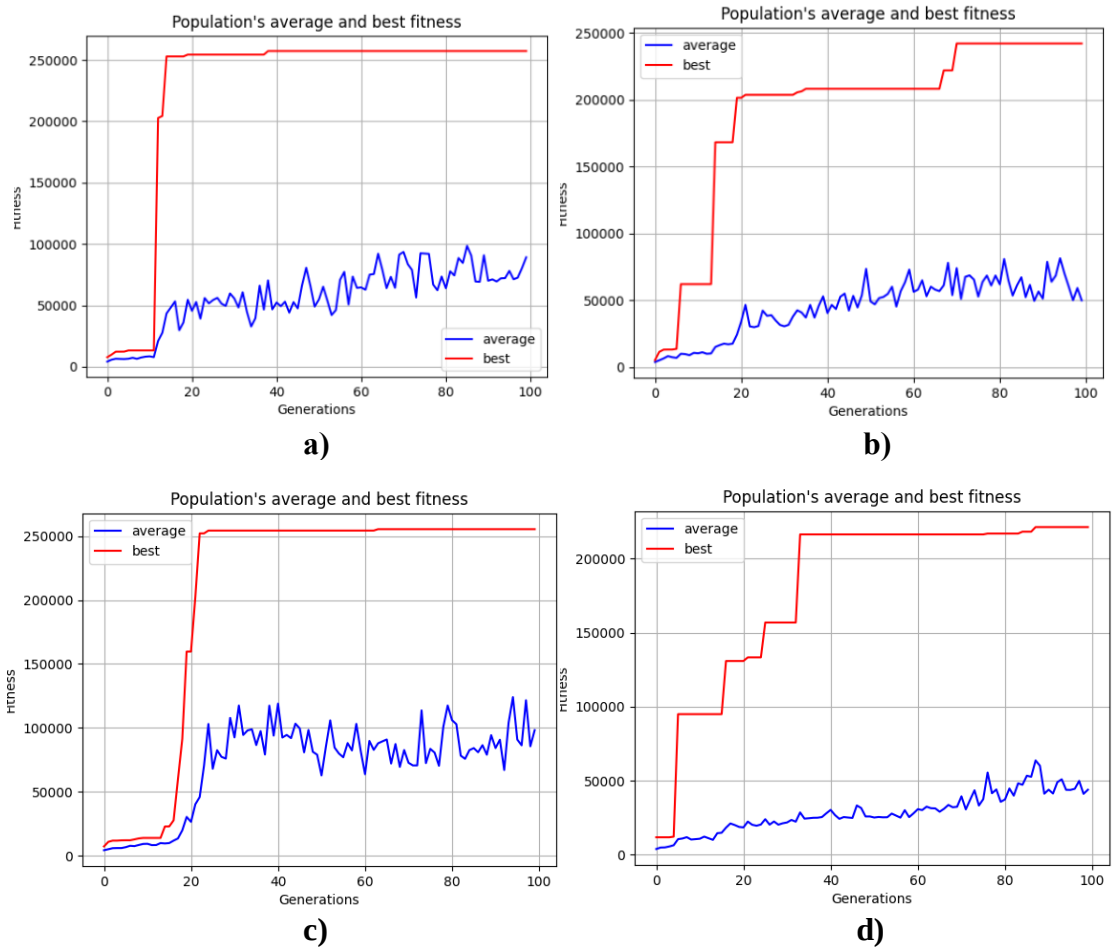
Slika 31 Mapa 3

Nakon što se konfiguracija parametara promijenjena radi povećanja raznolikosti populacije, jedinka je došla do cilja u 22. generaciji, a fitnes je počeo stagnirati tek u 70. generaciji i to na nešto manjoj vrijednosti nego u prethodnom slučaju (Slika 32 c). I sa promijenjenim parametrima ponekad se događa da populacija unutar 100 generacija ne uspijeva doći do cilja. Ono što je iznenađujuće jest da broj vrsta nije imao veliki utjecaj na konvergenciju. Točnije, u slučaju a) konvergiralo se sa svega dvije vrste, dok se u slučaju b) nije došlo do cilja unatoč velikoj vrsnoj raznolikosti. U slučaju c) nastalo je 4 vrste tokom simulacije koja dolazi do

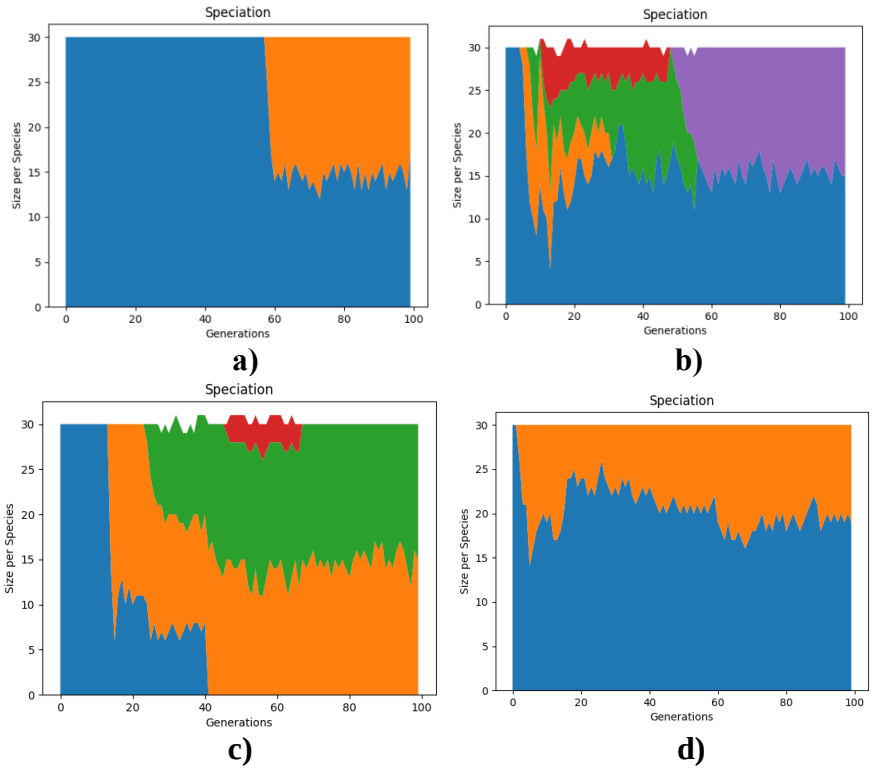
rješenja, dok u slučaju d) dvije vrste nije bilo dovoljno za doći do cilja (Slika 33). Slika 34 otkriva da ono što uspješne simulacije dijele jest kompleksnija topologija. Za razliku od neuspješnih slučajeva b) i d), razvile su topologiju koja sadrži jedan ili više skrivenih slojeva (neurona).

Tablica 3 Rezultati simulacija na mapi 3

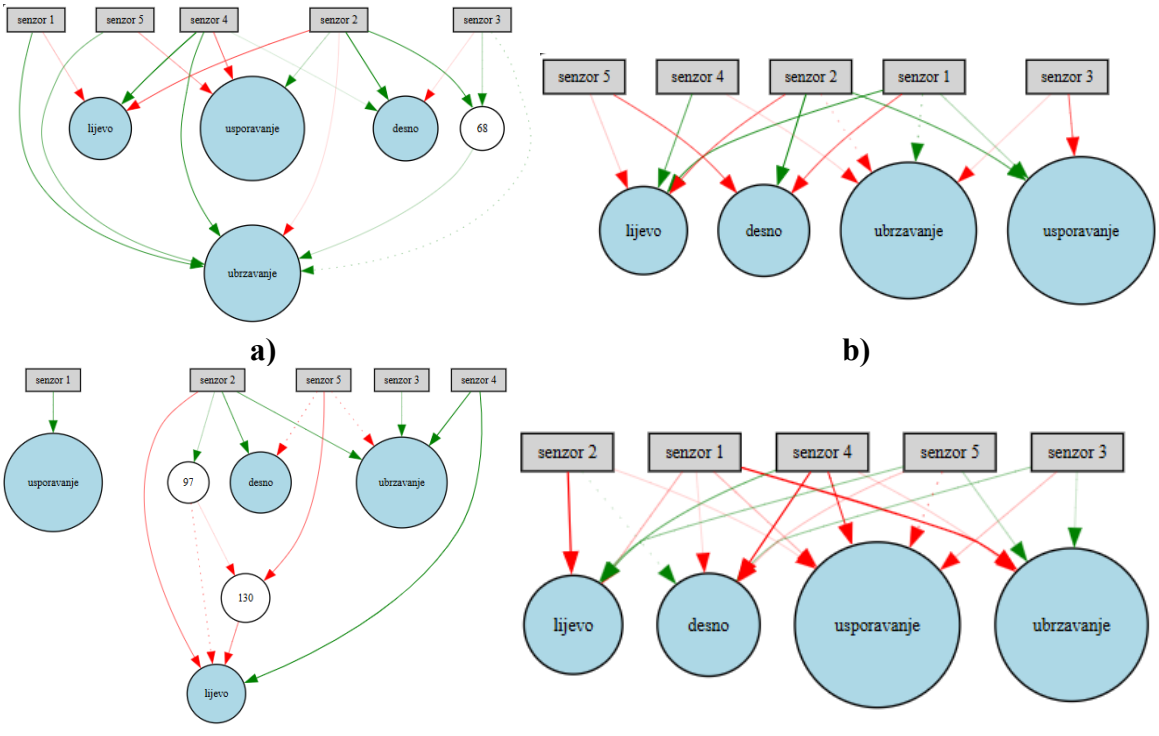
	Elitizam	Maks. stagnacija vrste	Min. veličina vrste	Dolazak do cilja u generaciji	Stagnacija fitnesa u generaciji	Maks. fitnes	Broj nastalih vrsta
a	3	20	Nedefinirana	14	38	257352	1
b	3	20	Nedefinirana	Ne dolazi	70	242040	5
c	3	10	1	22	63	255530	4
d	3	10	1	Ne dolazi	87	221377	2



Slika 32 Usporedba fitnesa uspješne i neuspješne simulacije



Slika 33 Usporedba raznolikosti vrsta za mapu 3



Slika 34 Usporedba topologije uspješne i neuspješne jedinke

7. ZAKLJUČAK

Završni rad osvrnuo se na koncepte u prirodi koji su inspirirali napredak u svijetu računarstva. Neuronske mreže i evolucijski algoritmi moćni su alati kojima se može optimizirati put do rješenja kompleksnih zadataka.

Cilj rada bio je analizirati koncept NEAT algoritma i rezultate njegove primjene kroz implementaciju navigacije mobilnog robota. Evolucijom težina i topologije neuronskih mreža uspješno se upravljalo kontrolerom mobilnog robota čija je zadaća bila slijediti put do cilja bez sudara. Uz težinu zadatka, i parametri algoritma imaju velik utjecaj na učinak njegove implementacije, stoga je bitno višestruko simulirati zadatke i analizirati ponašanje rezultata. Parametri koji funkcioniraju za jednu vrstu zadatka neće nužno biti idealni za neku drugu primjenu. No, uz dovoljnu raznolikost populacije i dovoljno dobar fitnes, mogu se naći zadovoljavajuća inovativna rješenja.

S obzirom na potencijal mehanizama NEAT-a, ovaj algoritam može optimizirati puno zamršenije zadatke. Primjer takvog zadatka je navigacija mobilnog robota kroz gomile ljudi koji je kratko opisan u ovom radu. Pouzdanost algoritma mogla bi se dalje testirati uvođenjem dodatnih prepreka u okruženje robota, a postavljanjem zahtjeva (npr. pronalaženja najkraćeg puta) ispitala bi se njegova efikasnost kod složenijih zadataka.

LITERATURA

[1] Haykin, S.: Neural Networks and Learning Machines Third Edition, McMaster University, Hamilton, Ontario Canada, 2009.

[2] Rahuk Jayawardana J. K., Sameera Bandaranayake, T.: Analysis of Optimizing Neural Networks and Artificial Intelligent Models for Guidance, Control and Navigation, International Research Journal of Modernization in Engineering Technology and Science,2021.

[3] Majetić, D.: Podloge za nastavu iz kolegija Neuronske mreže, FSB, 2022.

[4] Eiben A. E., Simth, J.E.: Introduction to Evolutionary Computing Second Edition, Springer, 2015.

[5] Ćurković, P.: Podloge za nastavu iz kolegija Umjetna inteligencija, FSB, 2022.

[6] Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning, The University of Alabama, 1989.

[7] Stanley, K. O., Miikkulainen, R.: Evolving Neural Networks through Augmenting Topologies, The University of Texas at Austin, The MIT Press Journals, 2002.

[8] Seriani, S., Msrcini, L., Caruso, M., Gallina, P., Medvet, E.: Crowded Environment Navigation with NEAT: Impact of Perception Resolution on Controller Optimization, Journal of Intelligent & Robotic Systems, 2021.

[9] https://neat-python.readthedocs.io/en/latest/, 7.9.2024.

[10] https://github.com/pygame/pygame, 28.8.2024.

[11] https://github.com/NeuralNine, 26.8.2024.

[12] https://github.com/CodeReclaimers/neat-python/blob/master/examples/xor/visualize.py, 5.9.2024.

[13] https://graphviz.org/download/, 5.9.2024.

PRILOG 1: PYTHON KOD ZA SIMULACIJU MOBILNIH ROBOTA

```
import math
import sys
import neat
import pygame
from visualize import plot_stats, plot_species, draw_net

# Dimenzije ekrana
WIDTH = 1920
HEIGHT = 1080

# Dimenzije robota
ROBOT_SIZE_X = 60
ROBOT_SIZE_Y = 60

BORDER_COLOR = (255, 255, 255, 255)  # Bijela boja definira rub staze

current_generation = 0  # Broj generacija

class Robot:
    def __init__(self):
        # Učitavanje slike robota
        self.sprite = pygame.image.load('robot.png').convert_alpha()
        self.sprite = pygame.transform.scale(self.sprite, (ROBOT_SIZE_X,
ROBOT_SIZE_Y))
        self.rotated_sprite = self.sprite # Rotacija slike robota

        # self.position = [80, 450] # Početna pozicija mape 2 i 3
        self.position = [870, 955]  # Početna pozicjia mape 1
        self.angle = 0
        self.speed = 0
        self.speed_set = False

        # Izračun središta robota na temelju trenutne pozicije
        self.center = [self.position[0]+ROBOT_SIZE_X/2,
self.position[1]+ROBOT_SIZE_Y/2]

        self.radars = []  # Lista senzora
        self.drawing_radars = []  # Crtanje radara

        self.alive = True  # Provjera je li robot i dalje na stazi
        self.distance = 0  # Prijeđen put
        self.time = 0  # Vrijeme

    def draw(self, screen):
        screen.blit(self.rotated_sprite, self.position)  # Prikaz robota u
trenutnoj poziciji
        self.draw_radar(screen)
```

```
    def draw_radar(self, screen):
        # Prikaz radara u simulaciji
        for radar in self.radars:
            position = radar[0]
            pygame.draw.line(screen, (0, 255, 0), self.center, position, 1)
            pygame.draw.circle(screen, (0, 255, 0), position, 5)

    def check_collision(self, game_map):
        # Provjera kolizije
        self.alive = True
        for point in self.corners:
            # Ako rub robota dotiče rub pretpostaviti sudar
            if game_map.get_at((int(point[0]), int(point[1]))) == BORDER_COLOR:
                self.alive = False
                break

    def check_radar(self, degree, game_map):
        # Provjera udaljenosti od prepreke
        length = 0
        x = int(self.center[0] + math.cos(math.radians(360 - (self.angle +
degree))) * length)
        y = int(self.center[1] + math.sin(math.radians(360 - (self.angle +
degree))) * length)

        # Produljenje radara sve do ruba staze ili duljine od 300 piksela
        while not game_map.get_at((x, y)) == BORDER_COLOR and length < 300:
            length = length + 1
            x = int(self.center[0] + math.cos(math.radians(360 - (self.angle +
degree))) * length)
            y = int(self.center[1] + math.sin(math.radians(360 - (self.angle +
degree))) * length)

        # Računanje udaljenosti od ruba i appendanje
        dist = int(math.sqrt(math.pow(x - self.center[0], 2) + math.pow(y -
self.center[1], 2)))
        self.radars.append([(x, y), dist])

    def update(self, game_map):
        # Brzina postavljena na 20 piksela po sličici
        self.time +=1 # za računanje fitnesa
        if not self.speed_set:
            self.speed = 20
            self.speed_set = True

        # Rotacija robota & pomicanje u smjeru osi-x
        self.rotated_sprite = self.rotate_center(self.sprite, self.angle)
        self.position[0] += math.cos(math.radians(360 - self.angle)) * self.speed
        # Ograničenje udaljenosti od lijevog i desnog ruba ekrana
        self.position[0] = max(self.position[0], 20)
        self.position[0] = min(self.position[0], WIDTH - 120)
```

```
        # ažuriranje ukupne prijeđene udaljenosti i vremena (ciklusa)
        self.distance += self.speed
        self.time += 1

        # Pomicanje robota u smjeru osi-y
        self.position[1] += math.sin(math.radians(360 - self.angle)) * self.speed
        # Ograničenje udaljenosti od gornjeg i donjeg ruba ekrana
        self.position[1] = max(self.position[1], 20)
        self.position[1] = min(self.position[1], WIDTH - 120)

        # Računanje novog središta robota
        self.center = [int(self.position[0]) + ROBOT_SIZE_X / 2,
int(self.position[1]) + ROBOT_SIZE_Y / 2]

        # Izračun koordinata kutova robota na temelju njegove trenutne poziciije i
orijentacije za provjeru kolizije
        length = 0.5 * ROBOT_SIZE_X
        left_top = [self.center[0] + math.cos(math.radians(360 - (self.angle +
30))) * length,
                    self.center[1] + math.sin(math.radians(360 - (self.angle +
30))) * length]
        right_top = [self.center[0] + math.cos(math.radians(360 - (self.angle +
150))) * length,
                     self.center[1] + math.sin(math.radians(360 - (self.angle +
150))) * length]
        left_bottom = [self.center[0] + math.cos(math.radians(360 - (self.angle +
210))) * length,
                       self.center[1] + math.sin(math.radians(360 - (self.angle +
210))) * length]
        right_bottom = [self.center[0] + math.cos(math.radians(360 - (self.angle +
330))) * length,
                        self.center[1] + math.sin(math.radians(360 - (self.angle +
330))) * length]
        self.corners = [left_top, right_top, left_bottom, right_bottom]

        # Provjera kolizije i clearanje radara
        self.check_collision(game_map)
        self.radars.clear()

        # Senzori na pozicijama -90, -45, 0, 45, 90 stupnjeva
        for d in range(-90, 120, 45):
            self.check_radar(d, game_map)

    def get_data(self):
        # Dobivanje udaljenosti od ruba staze
        radars = self.radars
        return_values = [0, 0, 0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)

        return return_values
```

```
    def is_alive(self):
        # Vraća je li robot još na stazi
        return self.alive

    def get_reward(self):
        # Fitnes funkcija
        if self.time > 0:
            return self.distance / (self.time / 60)
        else:
            return 0

    def rotate_center(self, image, angle):
        # Rotacija slike robota
        rectangle = image.get_rect()
        rotated_image = pygame.transform.rotate(image, angle)
        rotated_rectangle = rectangle.copy()
        rotated_rectangle.center = rotated_image.get_rect().center
        rotated_image = rotated_image.subsurface(rotated_rectangle).copy()
        return rotated_image

# Pokretanje simualcije za NEAT populaciju robota
def run_simulation(genomes, config):
    # Inicializacija lista za pohranu mreža i robota
    nets = []
    robots = []

    # Inicijalizacija PyGame biblioteke i prozora za prikaz simulacije
    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))

    for i, g in genomes:
        # Kreiranje mreža za svaki genotip
        net = neat.nn.FeedForwardNetwork.create(g, config)
        nets.append(net)
        g.fitness = 0 # Postavljanje Fitnesa

        robots.append(Robot()) # Dodavanje kreiranih robota u listu

    # Upravljanje fps
    clock = pygame.time.Clock()

    # Prikaz podataka o trenutnoj generaciji i broju živih jedinki
    generation_font = pygame.font.SysFont("Arial", 40)
    alive_font = pygame.font.SysFont("Arial", 30)

    # Učitavanje slike mape
    game_map = pygame.image.load('mapa.png').convert()

    global current_generation
    current_generation += 1

    counter = 0
```

```
    while True:
        # Omogućuje izlaz iz simulacije
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit(0)

        # Izlazi mreže
        for i, robot in enumerate(robots):
            output = nets[i].activate(robot.get_data())
            choice = output.index(max(output))
            if choice == 0:
                robot.angle += 10  # skretanje lijevo
            elif choice == 1:
                robot.angle -= 10  # skretanje desno
            elif choice == 2:
                if (robot.speed - 2 >= 12):
                    robot.speed -= 2  # kočenje
            else:
                robot.speed += 2  # ubrzavanje

        # Provjera je li jedinka još na na stazi
        still_alive = 0
        for i, robot in enumerate(robots):
            # Ako je robot na stazi uvećava mu se fitnes
            if robot.is_alive():
                still_alive += 1
                robot.update(game_map)
                genomes[i][1].fitness += robot.get_reward()
       # Kraj generacije ako je broj preživjelih jedinki jednak 0
        if still_alive == 0:
            break

        counter += 1
        if counter == 30 * 40:  # vremensko graničenje trajanja jedne generacije
            break

        # Prikaz isključivo preživjelih robota na mapi
        screen.blit(game_map, (0, 0))
        for robot in robots:
            if robot.is_alive():
                robot.draw(screen)

        # Prikaz podataka o trenutnoj generaciji
        text = generation_font.render("Generation: " + str(current_generation),
True, (0, 0, 0))
        text_rect = text.get_rect()
        text_rect.center = (900, 450)
        screen.blit(text, text_rect)
```

```
        # Prikaz podataka o trenutnom broju aktivnih jedinki
        text = alive_font.render("Still Alive: " + str(still_alive), True, (0, 0,
0))
        text_rect = text.get_rect()
        text_rect.center = (900, 490)
        screen.blit(text, text_rect)

        pygame.display.flip() # Osvježavanje ekrana
        clock.tick(60)  # brzina izvođenja simulacije (60 FPS)


if __name__ == "__main__":
    # Učitavanje NEAT konfiguracije
    config_path = "./config.txt"
    config = neat.config.Config(neat.DefaultGenome,
                                neat.DefaultReproduction,
                                neat.DefaultSpeciesSet,
                                neat.DefaultStagnation,
                                config_path)

    # Stvaranje populacije genoma
    population = neat.Population(config)
    # Prikupljanje podataka / statistika za evoluciju
    population.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    population.add_reporter(stats)

    # Pokretanje siumulacije s maksimalnim brojem generacija
    winner = population.run(run_simulation, 100)

# Vizualizacija
print('\nBest genome:\n{!s}'.format(winner))
node_names = {-1: 'senzor 1', -2: 'senzor 2', -3: 'senzor 3', -4: 'senzor 4', -5:
'senzor 5', 0: 'lijevo', 1: 'desno',
              2: 'ubrzavanje', 3: 'usporavanje'}
plot_stats(stats, view=True) # Grafički prikaz fitnesa
plot_species(stats, view=True) # Grafički prikaz specijacije
draw_net(config, winner, view=True, node_names=node_names) # Crtanje Topologije
najboljeg genoma
```

PRILOG 2: KONFIGURACIJA NEAT ALGORITMA

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 100000000
pop_size              = 30
reset_on_extinction   = True

[DefaultGenome]
# Opcije aktivacije čvorova
activation_default      = tanh
activation_mutate_rate  = 0.01
activation_options      = tanh

# Opcije agregacije čvorova
aggregation_default     = sum
aggregation_mutate_rate = 0.01
aggregation_options     = sum

# Opcije biasa
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# Opcije kompatibilnosti gena
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# Stope dodavanja/uklanjanja veza
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# Opcije omogućavanja veza
enabled_default         = True
enabled_mutate_rate     = 0.01

feed_forward            = True
initial_connection      = full

# Stope dodavanja/uklanjanja čvorova
node_add_prob           = 0.2
node_delete_prob        = 0.2

# Parametri mreže
num_hidden              = 0
num_inputs              = 5
num_outputs             = 4
```

```
# Opcije odziva čvora
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# Opcije težina
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 2.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
# max_stagnation     = 10
species_elitism      = 2

[DefaultReproduction]
elitism            = 3
# elitism          = 2
survival_threshold = 0.2
# min_species_size = 1
```