SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 831

RAZVOJ AGENTA ZA IGRU ZMIJA

Hana Smoković

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 831

RAZVOJ AGENTA ZA IGRU ZMIJA

Hana Smoković

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 831

Pristupnica:	**Hana Smoković (0036531489)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Marko Đurasević

Zadatak: **Razvoj agenta za igru zmija**

Opis zadatka:

Proučiti pravila igre zmija te pronaći ili izraditi jednostavnu implementaciju igre koja omogućuje igranje od strane čovjeka ili računalnog agenta. Istražiti postojeću literaturu koja se bavi razvojem agenata za igru zmija. Proučiti mogućnost primjene neuronskih mreža za dizajn agenta za igru zmija. Implementirati sustav za automatski razvoj agenata za igru zmija baziranu na neuronskim mrežama učenim evolucijskim algoritmima. Ocijeniti učinkovitost razvijenih agenata i predložiti moguća poboljšanja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 9. lipnja 2023.

SADRŽAJ

1. Uvod

Razvoj umjetne inteligencije započeo je idejom da se računalni sustavi mogu programirati da simuliraju inteligentno ponašanje. Iako rani korijeni umjetne inteligencije sežu u sredinu 20. stoljeća, tek posljednjih desetljeća dolazi do impresivnih postignuća. Zahvaljujući širokoj primjeni i vidljivosti u svakodnevnom životu, umjetna inteligencija vrlo je popularna u općoj populaciji. Neki od posebno zanimljivih softvera baziranih na umjetnoj inteligenciji su: chatbotovi, virtualni osobni asistenti, programi za preporuku sadržaja i alati za automatsko prevođenje.

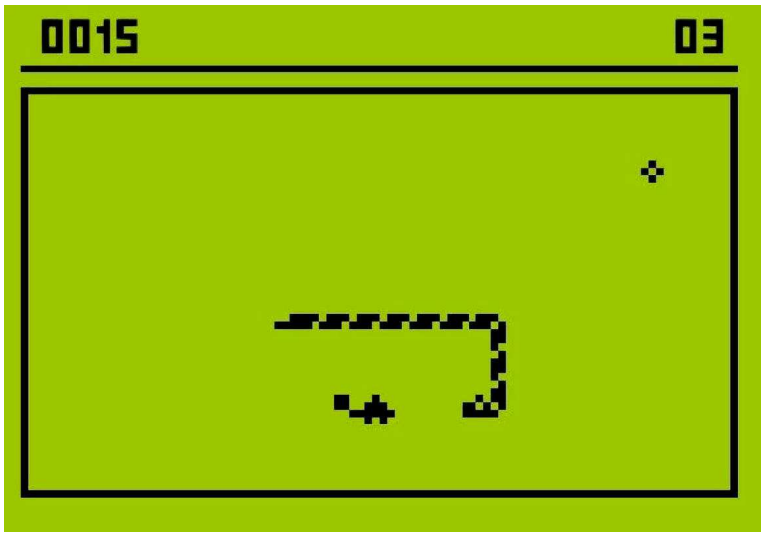
Ljude je oduvijek interesirao pojam inteligencije, a proučavanjem inteligentnog ponašanja u prirodi (npr. mravlje kolonije, jata ptica) pokušavali su otkriti kako nastaje inteligencija te je li ju moguće izgraditi. Ni danas nije jasno mogu li strojevi misliti, ali znamo da je moguće programirati računala tako da upravljaju složenim procesima, rješavaju složene probleme, donose odluke, zaključuju, prepoznaju vizualne objekte... Iako računala takve kompleksne zadatke mogu obavljati bez problema, na problem nailaze kod zadataka jednostavnih čovjeku, npr. zdravorazumsko zaključivanje, kretanje i navigacija, kreativnost i razumijevanje prirodnog jezika.

Zahvaljujući jednostavnim pravilima igre, razvoj agenta i implementacija igre Zmija pripadaju u zadatke s kojima računala nemaju problem. U ovom radu računalni agent razvijen je pomoću neuronskih mreža i genetskih algoritama. U idućem poglavlju nalaze se najbitnije informacije o igri i njenim pravilima. U trećem poglavlju napravljen je uvod u umjetne neuronske mreže, a u poglavlju nakon uvod u genetske algoritme uz prikaz njegove implementacije. Poglavlje pet govori o grafičkom korisničkom sučelju nakon čega slijedi prikaz rezultata istraživanja, komentar o metodi i validacija rezultata. Zaključak rada nalazi se u posljednjem poglavlju.

2. Igra Zmija

2.1. Povijest

Priča igre Zmija započinje puno prije globalnog uspjeha na kultnoj Nokiji. Koncept igre Zmije stvoren je davne 1976. godine pod imenom Blockage. Bila je to arkadna igra za dva igrača čiji su likovi pri kretanju ostavljali trag, a promjenu smjera su omogućavale tipke sa strelicama. Pobjednik bi bio igrač koji bi dulje izdržao bez da pogodi tragove ili zidove. Igra Zmija kakvu danas poznajemo premijerno se na tržištu pojavila 1997. godine na mobilnom telefonu Nokia 6110 [6]. Kako je takva igra izgledala prikazuje slika 2.1.



Slika 2.1: Prikaz igre Zmija na mobilnom telefonu Nokia [4]

2.2. Pravila igre

Postoje razne verzije igre Zmija: s različitim preprekama, s okolnim zidom, bez njega, s više vrsta hrane od kojih su neke vrjednije od drugih. Inačica kojom se ovaj radi bavi

je među jednostavnijima.

Cilj igre je postići što veći rezultat, odnosno pojesti što više hrane pritom izbjegavajući zabijanje u zidove ili sudaranje s vlastitim tijelom. Glava zmije neprestano se pomiče prema naprijed, ni u jednom trenutku ne može se zaustaviti niti kretati unazad, ali može mijenjati smjer kretanja: ravno, lijevo ili desno. U trenutku kad zmija pojede hranu, njezino tijelo produlji se za jedan segment.

Koordinate na kojima se pojavljuje hrana generiraju se slučajno i svaka koordinata ima jednaku vjerojatnost biti izabrana. Zmija na početku kreće iz središta prozora igre. Računalni agent za igru ima podatke na kojim koordinatama se nalazi tijelo zmije te koordinate na kojima je hrana trenutačno.

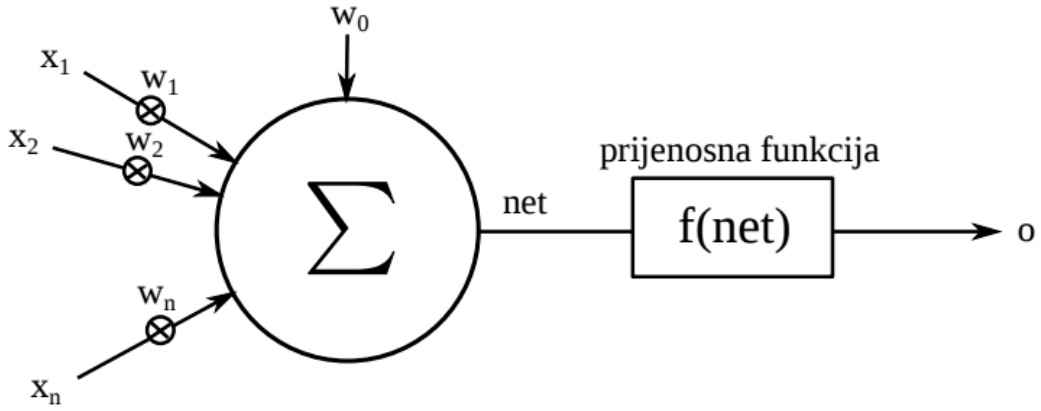
3. Umjetne neuronske mreže

Istražujući ljudski mozak, neurofiziologija i kognitivna znanost otkrile su da se on sastoji od jako velikog broja neurona (živčanih stanica) koji paralelno obrađuju podatke. Ta spoznaja potaknula je razvoj konektivističkog pristupa izgradnji sustava s arhitekturama sličnim arhitekturi mozga koji uče samostalno na temelju iskustava. Upravo tako umjetnoj neuronskoj mreži ne definira se način na koji treba obrađivati podatke, već ona sama uči složene obrasce i donosi odluke na temelju ulaznih podataka [10].

3.1. Umjetni neuron

Proučavajući strukturu biološkog neurona, znanstvenici Warren McCulloch i Walter Pitts 1943. godine prvi su definirali model umjetnog neurona. Svaki umjetni neuron prima ulazne podatke, obrađuje ih te proizvodi jedan izlaz koji prosljeđuje na ulaze više drugih neurona.

Umjetni neuron prima ulaze x_1, x_2,..., x_n koji mogu biti izlazi prethodnih neurona ili podaci iz vanjskog svijeta. Svaki ulaz x_i povezan je s težinom w_i koja govori u kojoj mjeri signal utječe na neuron. To njegovo djelovanje određeno je umnoškom $x_i \cdot w_i$.



Slika 3.1: Model umjetnog neurona [10]

Zadaća tijela stanice umjetnog neurona je objediniti primljene podatke u jednu vrijednost potencijala stanice. Kao na slici 3.1, tijelo stanice na svojem izlazu generira sumu umnožaka ulaza i težina koja je označena kao net, a računa se izrazom:

$net = \sum_{i=1}^{n} w_i \cdot x_i + w_0.$

Svaki neuron ima još jedan ulaz koji označavamo s x_0 i čija vrijednost je uvijek 1. Uzevši to u obzir, formulu možemo jednostavnije zapisati. net sada postaje skalarni produkt vektora ulaza i vektora težina:

$net = \sum_{i=0}^{n} w_i \cdot x_i = \vec{w} \cdot \vec{x}.$

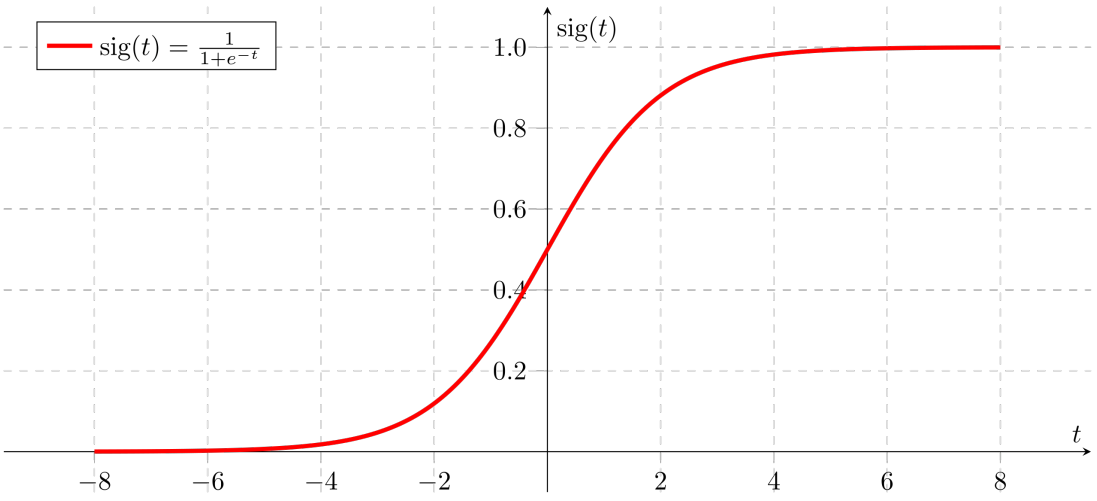
3.2. Prijenosna funkcija

Skalarni produkt net dolazi na ulaz prijenosne funkcije neurona čija je zadaća modelirati prijenos signala i odrediti konačni izlaz neurona prema izrazu:

$o = f(net).$

Neke od najčešće upotrebljavanih prijenosnih funkcija su: sigmoidalna funkcija, funkcija identiteta, tangens hiperbolni, funkcija skoka i ReLu. Samo nelinearne prijenosne funkcije omogućavaju rješavanje kompleksnih problema koristeći mali broj čvorova. U ovom radu korištena je sigmoidalna prijenosna funkcija čiji graf se nalazi na slici 3.2.

3.2.1. Sigmoidalna funkcija



Slika 3.2: Grafički prikaz sigmoidalne funkcije [8]

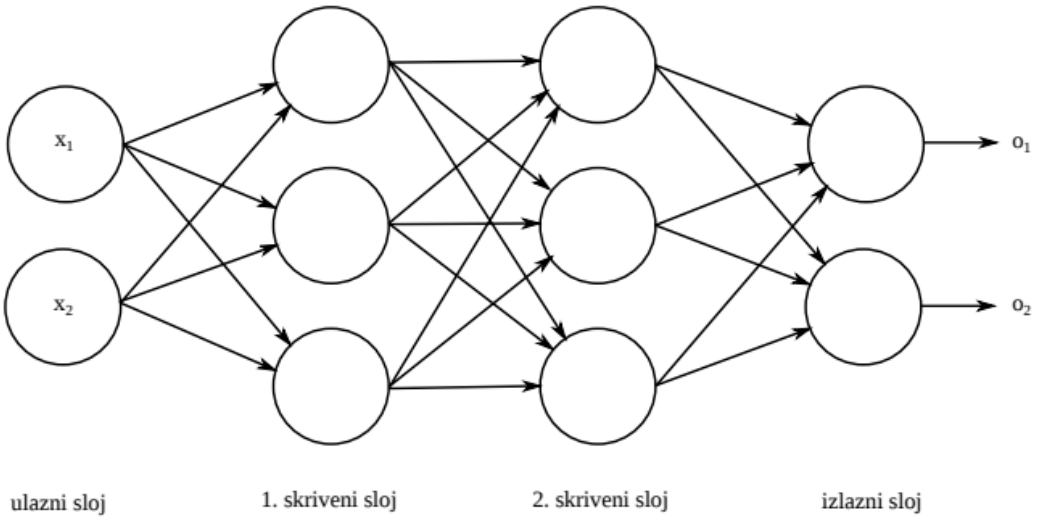
Sigmoidalna funkcija poprima bilo koju stvarnu vrijednost kao ulaz i daje vrijednosti u rasponu od 0 do 1 na izlazu. Za veće ulazne vrijednosti izlazna vrijednost bit će bliža 1 i obrnuto, za manje ulaze izlazi će biti bliži 0. Matematička formula sigmoidalne funkcije je:

$sigm(net) = \frac{1}{1+e^{-net}}$.

Ova je funkcija poopćena verzija funkcije skoka, koja je također definirana na istom intervalu. Sigmoida je dobra za modele u kojima moramo predvidjeti vjerojatnost kao izlaz jer vraća vrijednosti iz intervala između 0 i 1. Drugi razlog zašto je često korištena je jer pruža glatki gradijent čime sprječava skokove u izlaznim vrijednostima.

3.3. Neuronska mreža

Umjetne neuronske mreže sastoje se od velikog broja međusobno povezanih umjetnih neurona. Postoje različite vrste neuronskih mreža, ali u ovom radu bavimo se *potpuno povezanim unaprijednim slojevitim umjetnim neuronskim mrežama* (engl. *Fully-connected Feed-forward Multi-layered*). Karakteristike ove vrste mreže su: izostanak ciklusa te podjela neurona u slojeve, gdje neuroni sloja *i* dobivaju podatke samo iz sloja *i-1*, i to od svakog neurona iz tog sloja.



Slika 3.3: Slojevita neuronska mreža s četiri sloja [9]

Slika 3.3 prikazuje ovu vrstu neuronske mreže, a ona se sastoji od četiri sloja. Ulazni sloj s dva neurona mreži omogućuje primitak ulaznih podataka. Drugi sloj, točnije prvi skriveni sloj, ima ulogu obrade podataka na način opisan u potpoglavlju 3.1. Neuroni računaju težinsku sumu ulaznih podataka, prosljeđuju ih prijenosnoj funkciji,

a njihov izlaz predstavlja prijenosne funkcije. Prvi skriveni sloj mreže sa slike 3.3 ima tri neurona, od koji svaki ima po tri težine: dvije koje određuju utjecaj ulaznog sloja neurona i jedna koja određuje pristranost (engl. *bias*). Time prvi skriveni sloj ukupno ima $3 \cdot 3 = 9$ težina. Za treći sloj, drugi skriveni sloj, vrijedi sve kao i za prvi skriveni sloj, osim što tri neurona mreže sad imaju svaki po četiri težina (tri za utjecaj prvog skrivenog sloja i pristranost). Tako drugi skriveni sloj ukupno ima $3 \cdot 4 = 12$ težina. Četvrti, odnosno izlazni sloj sastoji se od dva neurona koja imaju po četiri težine što rezultira s ukupno $2 \cdot 4 = 8$ težina. Ovu neuronsku mrežu arhitekture $2 \times 3 \times 3 \times 2$ čini 29 težina, tj. 29 parametara je moguće mijenjati tijekom treniranja mreže.

U tijeku izrade ovog rada korišteno je nekoliko različitih arhitektura, a pri treniranju najbolja se pokazala $6 \times 20 \times 3$. To znači da je neuronska mreža građena od 3 sloja: ulazni sloj sa šest neurona, prvi skriveni sloj s dvadeset neurona i izlazni sloj s tri neurona. Na slici 3.4 prikazana je implementacija funkcije unaprijedne propagacije koja kao argument prima ulazni vektor.

```
# funkcija koja obavlja prolaz kroz mrežu
def forwad_propagation(self, network_input):
    output = network_input
    # iteriranje po slojevima neuronske mreže
    for i in range(len(self.weights)):
        weights = self.weights[i]
        bias = self.bias[i]
        # output sloja je sigmoidalna funkcija od zbroja biasa te umnoška ulaza sloja i težina
        output = self.sigmoid(np.dot(output, weights.T) + bias)
    # traži se indeks najvećeg elementa izlaza
    ind = max_index(output)
    # ovisno koji element je najveći odabire se sljedeći potez
    action = actions[ind]
    return action
```

Slika 3.4: Implementacija unaprijedne propagacije kroz mrežu

Mreža prima ulazni vektor duljine šest koji se generira ovisno o trenutnoj poziciji zmije i hrane na ploči. Značenje elemenata ulaznog vektora je sljedeće:

- postoji li opasnost lijevo od zmije
- postoji li opasnost desno od zmije
- postoji li opasnost ravno ispred zmije
- udaljenost glave zmije od hrane u slučaju da skrene lijevo
- udaljenost glave zmije od hrane u slučaju da skrene desno
- udaljenost glave zmije od hrane u slučaju da nastavi ravno

Ulazni sloj omogućuje primitak ovog ulaznog vektora, dok skriveni sloj obrađuje podatke. Izlazni sloj određuje idući potez zmije. S obzirom da podaci prije izlaza prolaze kroz sigmoidalnu funkciju, elementi izlaznog vektora duljine tri bit će iz intervala [0, 1]. Najveći element postavlja se na 1, a ostala dva elementa na 0. Time dobivamo tri moguća izlaza od kojih svaki odgovara jednom potezu:

- zmija nastavlja kretanje ravno za izlaz $[1, 0, 0]$
- zmija skreće lijevo za izlaz $[0, 0, 1]$
- zmija skreće desno za izlaz $[0, 1, 0]$

4. Genetski algoritmi

4.1. Općenito

Evolucijsko računarstvo grana je umjetne inteligencije, a glavna joj je svrha riješiti optimizacijske probleme. Evolucijsko se računarstvo može podijeliti u tri skupine: algoritme rojeva, evolucijske algoritme i ostale algoritme. U ovom radu bavimo se genetskim algoritmima, podvrstom evolucijskih algoritama.

H. J. Holland je 1975. godine stvorio genetske algoritme. Nadahnuće za razvoj svih evolucijskih algoritama, pa tako i genetskih algoritama bila je Darwinova teorija o postanku vrsta. Iz Darwinove teorije slijedi da će u svakoj populaciji postojati borba za preživljavanje ako potomaka ima više no što je potrebno, a količina hrane je ograničena [9]. Tada bolje i jače jedinke imaju veću šansu preživjeti i stvarati potomke. Potomci su u velikoj mjeri određeni genetskim materijalom roditelja, ali postoji određeno odstupanje.

4.2. Generacijski genetski algoritam

Prva vrsta genetskih algoritama je eliminacijski genetski algoritam. U svakom koraku tog algoritma radi se izmjena jedne jedinke u populaciji. Druga vrsta je generacijski genetski algoritam koji je primijenjen u ovom radu.
Pseudokod algoritma:

```
broj_generacije = 1
dok je veličina populacije manja od zadane veličine generacije:
   generiraj jedinku (neuronsku mrežu)
   jedinku ubaci u populaciju
dok je broj_generacije < 500:
```

```
evaluacija populacije
u novu populaciju ubaci elitne jedinke
dok nova populacije ne bude veličine kao stara:
    selekcijom izaberi dva roditelja iz stare
    križanje
    mutacija djeteta
    ubaci dijete u novu populaciju
nova populacija postaje stara
broj_generacije += 1
```

U prvoj iteraciji algoritma generira se slučajna roditeljska populacija jedinki koja se zatim evaluira. U svakoj iteraciji stvara se nova populacija jedinki jednake veličine kao stara. Odabran broj elitnih jedinki nepromijenjene se kopiraju u novu populaciju. Do popunjenja populacije ponavlja se: selekcijom se biraju dva roditelja koja se zatim križaju, a dobiveno dijete nakon mutacije sprema se u novu populaciju. Tada završava smjena generacija i nova populacija postaje roditeljska u sljedećoj iteraciji algoritma.

4.3. Funkcija dobrote

Funkcija dobrote (engl. *fitness-function*) koristi se za procjenu kvalitete svake jedinke u populaciji. Funkcija dobrote jedinki dodjeljuje brojčanu vrijednost ovisno koliko jedinka dobro rješava problem. Cilj ovog postupka je sortirati jedinke po kvaliteti. Rangirane jedinke kasnije su bitne kod elitizma kad 5 najboljih jedinki automatski prebacujemo u novu populaciju. Također, bitno je i za troturnirsku selekciju gdje od tri slučajno odabrane jedinke najbolja postaje roditelj u križanju. U ovom radu korišten je model gdje svaka jedinka iz populacije tri puta odigra igru, a njena funkcija dobrote je prosjek pojedene hrane po igri. Cilj takvog izračuna je smanjenje slučajnosti, da se ne bi događalo da dobra jedinka slučajno odigra loše te ne bude prenesena u sljedeću generaciju.

4.4. Elitizam

Elitizam je karakteristika algoritma da ne može izgubiti najbolje pronađeno rješenje. To se postiže prijenosom određenog broja najboljih nepromijenjenih jedinki iz stare u novu populaciju. Konkretno, u ovom radu prenosi se najboljih 5 od 100 iz roditeljske populacije. Na slici 4.1 nalazi se način provedbe elitizma u kodu.

```python
# funckija koja vraća listu najboljih jedinki
# koje se nepromijenjene prenose u iduću generaciju
def elitism(population):
    return population[0:neural['elitism']]
```

Slika 4.1: Implementacija elitizma

4.5. Selekcija

Selekcija predstavlja odabir jedinki za križanje na temelju njihove kvalitete. Jedinke s većom vrijednosti funkcije dobrote imaju veću vjerojatnost da budu odabrane u procesu selekcije. Selekcijski pritisak posljedica je načina na koji dobrota jedinke utječe na vjerojatnost njezinog razmnožavanja i preživljavanja [9].

4.5.1. Turnirska selekcija

U ovom radu korištena je k-turnirska selekcija koja omogućava kontroliranje selekcijskog pritiska podešavanjem parametra k. U turnir ulazi k jedinki (k je manji ili jednak veličini populacije), dok je izlaz najbolja jedinka. k jedinki koje ulaze u turnir odabiremo slučajnim postupkom uz jednaku distribuciju vjerojatnosti iz još neodabranih jedinki. Implementacija ovakvog načina selekcije predočena je na slici 4.2. Najpoželjnija jedinka bit će ona koja od jedinki u turniru ima najveću dobrotu. Selekcijski postupak provodimo dva puta čime biramo dva roditelja iz kojih u križanju stvaramo dijete.

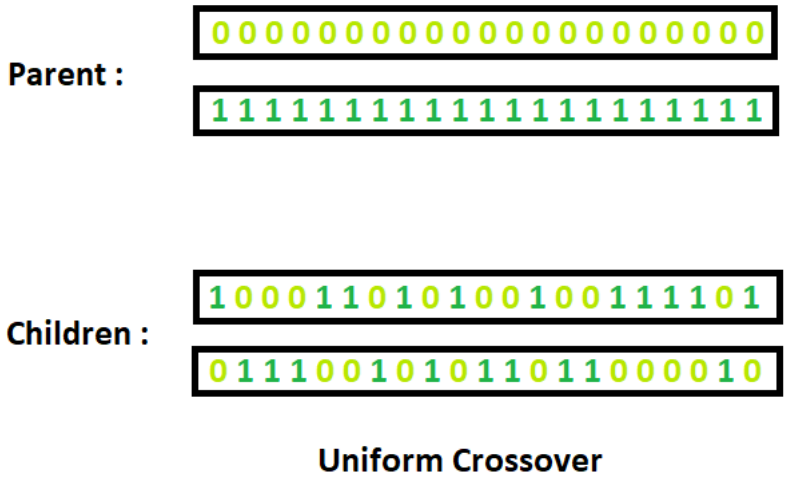
```python
# troturnirska selekcija, od 3 jedinke vraća se ona s najvećim fitnessom
def selection(population):
    # generira se lista 3 nasumična indeka jedinki u populaciji
    selected = random.sample(range(0, neural['generation_size']), 3)
    # kao trenutno najbolja izabire se prva
    best = population[selected[0]]
    # iteriranje po listi nasumičnih indeksa
    for ind in selected:
        # ako je neka od ostalih jedinki iz turnira bolja od najbolje, ona postaje najbolja
        if population[ind].fitness > best.fitness:
            best = population[ind]
    return best
```

Slika 4.2: Implementacija turnirske selekcije

U ovom radu korišten je $k = 3$ što omogućuje natjecanje jedinki u odabiru najbolje. Korištenjem većeg parametra k, odabrane jedinke bit će birane iz malog skupa najboljih jedinki populacije. Npr. ako u turnir uzimamo sve jedinke iz populacije, selekcijom će se uvijek izabrati najbolja jedinka, zbog čega će algoritam zaglaviti u lokalnom optimumu. Suprotno od toga, ako za turnir jednolikom distribucijom biramo samo jednu jedinku, ona će uvijek biti najpoželjnija. To dovodi do problema jer se dobrota jedinki uopće ne uzima u obzir.

4.6. Križanje

Križanje je operator kojim se nove jedinke stvaraju kombiniranjem genetskog materijala roditelja tako da dijete dio gena preuzme od prvog roditelja, a dio od drugog roditelja. Postoji nekoliko načina na koji se križanje može ostvariti, najčešći primjeri su uniformno križanje, križanje s jednom točkom i s dvije točke prijeloma. U križanju s jednom točkom prijeloma pronalazi se pozicija na kojoj se cijepa oba roditelja. Dijete od prvog roditelja dobiva gene do te točke, a od drugog gene nakon točke [9]. Suprotno od takvog križanja, kod uniformnog križanja kromosom ne dijelimo na segmente, već tretiramo svaki gen zasebno. U ovom radu koristi se uniformno križanje, a njegova implementacija nalazi se na slici 4.4.



Slika 4.3: Uniformno križanje [2]

4.6.1. Uniformno križanje

Kod uniformnog križanja svaki gen se uzima od jednog roditelja pri čemu oba roditelja imaju jednaku vjerojatnost biti izabrana. Moguće je koristiti neki drugi omjer vjerojatnosti odabira pojedinog roditelja, a posljedica tog su potomci koji nasljeđuju više genetskih informacija od jednog roditelja nego od drugog. Odabir dobiva li dijete gen prvog ili gen drugog roditelja može se usporediti s bacanjem ispravnog novčića. Novčić je ispravan kad je vjerojatnost pojavljivanja oba događaja jednak [3].

```
# funkcija uniformnog križanja kojoj predajemo 2 roditelja kao argumente
def crossover(parent1, parent2):
    w, b = [], []
    # određivanje težina djeteta
    for i in range(len(parent1.weights)):
        parent1_w = parent1.weights[i]
        parent2_w = parent2.weights[i]
        shape = parent1_w.shape
        # inicijalizacija matrice težina na nule
        child_w = np.zeros(shape)
        for j in range(shape[0]):
            for k in range(shape[1]):
                # ako je random manje od 0.5, dijete preuzima težinu parenta1
                if np.random.uniform(0, 1) < 0.5:
                    child_w[j, k] = parent1_w[j, k]
                # inače dobiva težinu parenta2
                else:
                    child_w[j, k] = parent2_w[j, k]
        w.append(child_w)

    # određivanje biasa djeteta
    for i in range(len(parent1.bias)):
        parent1_b = parent1.bias[i]
        parent2_b = parent2.bias[i]
        shape = parent1_b.shape
        # inicijalizacija bias vektora na nule
        child_vector = np.zeros(shape)
        for j in range(shape[0]):
            # ako je random manje od 0.5, dijete preuzima bias parenta1
            if np.random.uniform(0, 1) < 0.5:
                child_vector[j] = parent1_b[j]
            # inače dobiva bias parenta2
            else:
                child_vector[j] = parent2_b[j]
        b.append(child_vector)

    return NeuralNetwork(w, b)
```

Slika 4.4: Implementacija uniformnog križanja

Primjena uniformnog križanja je dobra u situacijama kad su svi geni važni i kad nije poznato koji su geni bolji od drugih. Budući da djeca nastaju kombiniranjem genetskog materijala roditelja ona su im genetski slična. Ovakav princip vidljiv je na slici 4.3. Zato kažemo da operator križanja radi detaljnu pretrage prostora rješenja u kojem se nalaze roditelji. Kada bismo koristili isključivo križanje naišli bismo na problem, svaka generacija bila bi sve zgusnutija, volumen populacije bi se smanjivao i gubio bi se genetski materijal. Problem rješavamo primjenom operatora mutacije objašnjenog u idućem potpoglavlju.

4.7. Mutacija

Mutacija je operator koji obavlja slučajnu promjenu genetskog materijala jedinki stvorenih križanjem. Svrha mutacije su očuvanje raznovrsnosti jedinki u populaciji, otkrivanje novih kombinacija gena koje nije moguće dobiti križanjem i izbacivanje populacije iz lokalnog optimuma.

```
# funkcija koja izvršava mutaciju jedinke
def mutate(self):
    # mutacija težina
    for i in range(len(self.weights)):
        weight = self.weights[i]
        for j in range(weight.shape[0]):
            for k in range(weight.shape[1]):
                # ako je random vr. manja od zadane vjerojatnosti mutation_prob -> mutacija
                if np.random.uniform(0, 1) < neural['mutation_prob']:
                    # težinama se pribraja Gaussov šum
                    self.weights[i][j][k] += np.random.normal(0, neural['K'])

    # mutacija biasa
    for i in range(len(self.bias)):
        bias = self.bias[i]
        for j in range(bias.shape[0]):
            # ako je random vr. manja od zadane vjerojatnosti mutation_prob -> mutacija
            if np.random.uniform(0, 1) < neural['mutation_prob']:
                # biasima se pribraja Gaussov šum
                self.bias[i][j] += np.random.normal(0, neural['K'])
```

Slika 4.5: Implementacija mutacije jedinke

Mutaciju možemo raditi na više načina, ali u radu je korištena sljedeća. Mutacija se primjenjuje na svaki gen zasebno, stoga ćemo definirati vjerojatnost mutacije koja predstavlja vjerojatnost da će se neki određeni gen mutirati. Ta vjerojatnost određuje koliko će se često primijeniti mutacija za svaki gen. Uobičajeno je koristiti niže vje-

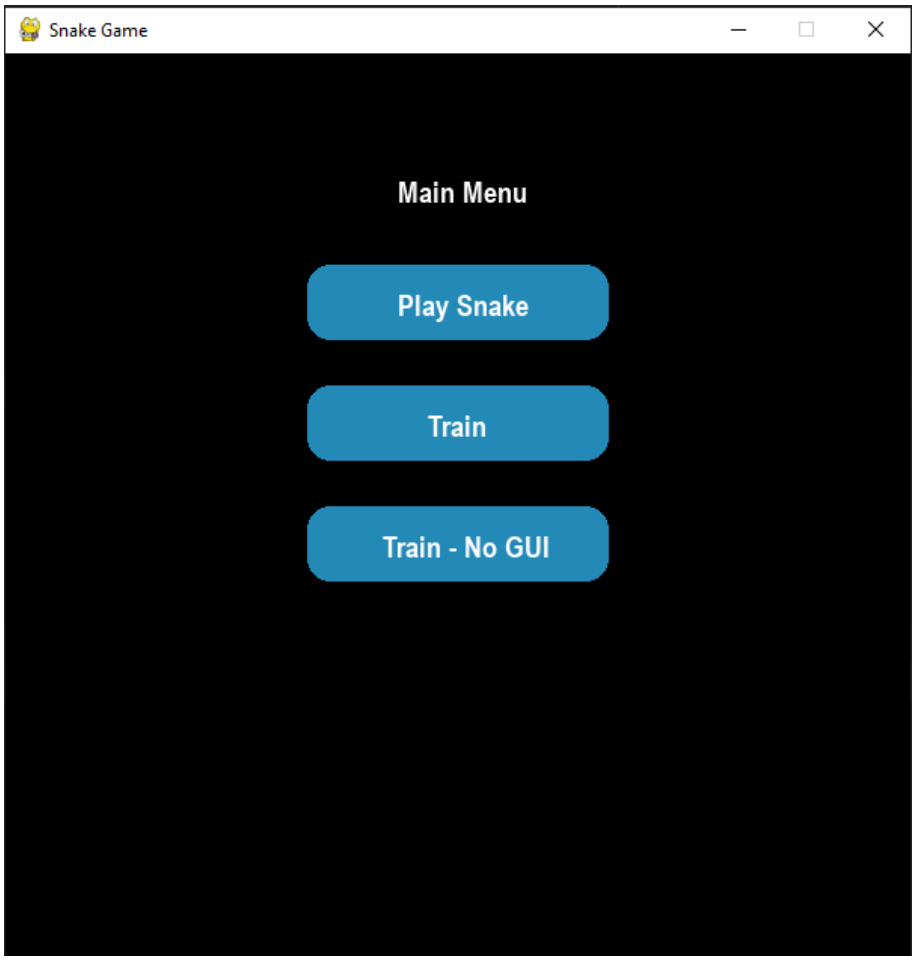
rojatnosti mutacije kako bi se održala stabilnost populacije i izbjegao gubitak dobrih rješenja. U implementaciji je korištena vjerojatnost 0.1 što znači da svaki gen ima 10% šanse biti mutiran.

Kao što je prikazano na slici 4.5, iteriramo kroz sve gene jedinke, za svaki gen uzimamo vjerojatnost iz uniformne razdiobe koja nam govori trebamo li ga mutirati ili ne. Ako je ta vjerojatnost manja od vjerojatnosti mutacije 0.1 izvršavamo mutaciju tog gena. Operator mutacije implementiran je kao Gaussov šum, što znači da se genu pribroji vektor uzorkovan iz normalne razdiobe centrirane oko 0 sa standardnom devijacijom 0.2.

5. Implementacija

Implementacija igre Zmija napravljena je u programskom jeziku Python. U njemu je izrađeno i grafičko korisničko sučelje kojim je omogućeno igranje igre od strane čovjeka ili računalnog agenta.

5.1. Početni izbornik



Slika 5.1: Prikaz početnog izbornika igre

Početni izbornik grafičkog korisničkog sučelja nalazi se na slici 5.1. Gumb *Play Snake* pokreće igru za čovjeka i zmija se automatski počinje kretati. Promjena smjera moguća je strelicama gore, dolje, lijevo i desno.

Gumb *Train* započinje proces treniranja neuronske mreže. Igre se igraju automatski, pa se odmah nakon zabijanja zmije u zid ili svoj rep započinje nova igra. Isti proces treniranja aktivira gumb *Train - No GUI*, osim što se u ovom slučaju GUI gasi, pa je proces treniranja brži oko sto puta.

5.2. Sučelje igre

Sučelje igre može se vidjeti na slici 5.2. Na zaslonu je prikazan trenutni rezultat korisnika ili agenta te najbolji rezultat od pokretanja igre.

Slika 5.2: Prikaz implementacije igre Zmija

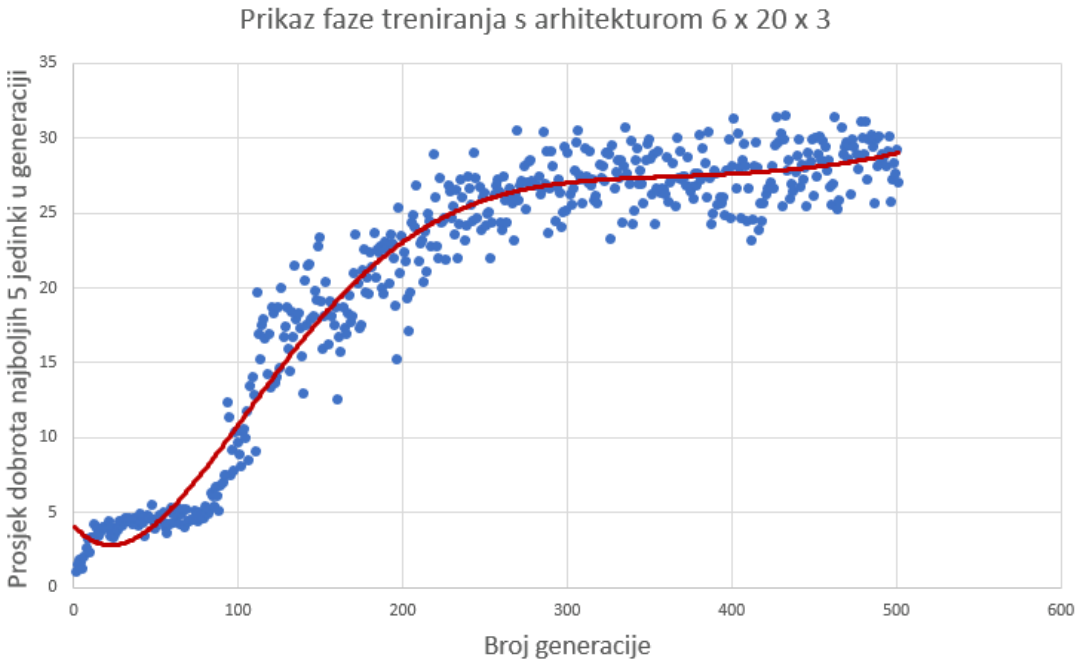
6. Rezultati

Veličina populacije koja je prošla kroz proces treniranja iznosila je 100 jedinki, a svaka jedinka iz generacije igrala je Zmiju 3 puta kako bi dobila izračun funkcije dobrote. Proces treniranja izvršen je na 500 generacija što je trajalo otprilike 2 sata. Svi parametri umjetne neuronske mreže i genetskog algoritma prikazani su u tablici 6.1.

Tablica 6.1: Tablični prikaz parametara neuronske mreže i genetskog algoritma

Parametri neuronske mreže	**Parametri genetskog algoritma**
arhitektura = [6, 20, 3]	broj generacija = 500
težine	veličina generacije = 100
pristranosti	vjerojatnost mutacije = 0.1
	broj elitnih jedinki = 5
	broj jedinki koji ulazi u turnir k = 3
	standardna devijacija Gaussova šuma K = 0.2

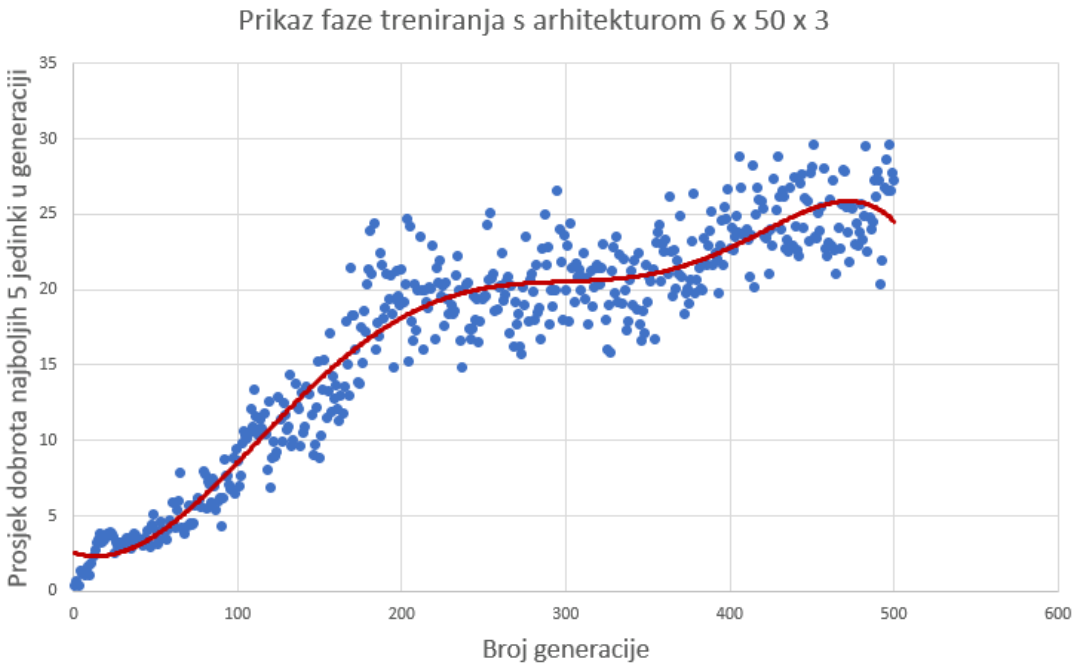
Na slici 6.1 nalazi se grafički prikaz podataka za vrijeme treniranja s arhitekturom $6 \times 20 \times 3$. Za svaku generaciju odabrano je 5 najboljih jedinki te je izračunat prosjek njihovih dobrota. Svaka točka na grafu predstavlja prosjek dobrota 5 najboljih jedinki u pojedinoj generaciji. Na slici 6.1 može se primijetiti kako jedinkama u početku treba vremena da bi krenule brže napredovati. To može upućivati na lošu inicijalizaciju parametara neuronske mreže. Bez obzira na to kakva je inicijalizacija, genetski algoritam trebao bi uspjeti optimizirati težine umjetne neuronske mreže, samo bi mu trebalo više vremena. Nakon sporijeg početka slijedi brži napredak, pa potom opet period stagnacije.



Slika 6.1: Grafički prikaz prosječne dobrote najboljih jedinki tijekom faze treniranja s arhitekturom $6 \times 20 \times 3$

Na slici 6.2 nalazi se grafički prikaz podataka za vrijeme treniranja s arhitekturom $6 \times 50 \times 3$. Kao i za prošli graf, iz svake generacije odabrano je 5 najboljih jedinki te je izračunat prosjek njihovih dobrota. Svaka točka na grafu predstavlja prosjek dobrota 5 najboljih jedinki u pojedinoj generaciji. I na slici 6.2 vidljiv je sporiji napredak jedinki u početnim generacijama, nakon čega slijedi faza bržeg poboljšanja.

Uspoređujući rezultate treniranja sa slika 6.1 i 6.2 arhitektura $6 \times 20 \times 3$ pokazala se boljom od arhitekture $6 \times 50 \times 3$. Arhitektura $6 \times 20 \times 3$ nakon 500 generacija dolazi do prosječne dobrote od otprilike 30, dok arhitektura $6 \times 50 \times 3$ staje na oko 25. Također, faza bržeg napretka kod arhitekture $6 \times 20 \times 3$ kreće ranije te je izraženija.



Slika 6.2: Grafički prikaz prosječne dobrote najboljih jedinki tijekom faze treniranja s arhitekturom $6 \times 50 \times 3$

Nakon faze treniranja s arhitekturom $6 \times 20 \times 3$ uslijedilo je testiranje najbolje jedinke iz 500. generacije, koja je zatim odigrala 200 igara. Podaci prikupljeni pri testiranju prikazani su na slici 6.3. Prosječni rezultat na ovom uzorku je oko 23 pojedene hrane po igri, uz nekoliko odličnih izuzetaka koji idu do 60.

Uspoređujući graf sa slike 6.1 i graf sa slike 6.3 vidi se da 5 najboljih jedinki na kraju faze treniranja ima bolje performanse nego najbolja jedinka u fazi testiranja. Odabir jedne najbolje jedinke nije najbolji način evaluacije rezultata zbog prevelikog faktora slučajnosti. Npr. moglo se nekoj lošijoj jedinki posrećiti igranje 3 igre u 500. generaciji, dok neka bolja jedinka nije imala toliko sreće pa nije odabrana za testiranje. Taj faktor slučajnosti smanjen je korištenjem prosjeka pojedene hrane u 3 igre, ali on još uvijek postoji.

Slika 6.3: Grafički prikaz broja pojedene hrane u fazi testiranja

7. Zaključak

U ovom radu istražena je mogućnost primjene genetskih algoritama prilikom razvoja agenta za igru Zmija. Pravila igre objašnjena su na početku rada, a budući da su ona vrlo jednostavna, igra Zmija je pogodna za ovakav zadatak. U idućem poglavlju pojašnjena je teorijska pozadina neuronskih mreža, arhitektura implementirane mreže te način na koji mreža odlučuje koji potez se obavlja sljedeći. Poglavlje nakon opisuje koncept genetskih algoritama i način njihove primjene. Uz opise korištenih modela selekcije, križanja i mutacije dani su isječci implementacije istih. S obzirom na to da je u sklopu rada izrađeno grafičko korisničko sučelje koje omogućuje igranje od strane čovjeka ili računalnog agenta, u idućem poglavlju pokazan je njegov izgled i funkcionalnosti. Rezultati istraživanja zatim su grafički prikazani. Uspoređene su faze treniranja s arhitekturama $6 \times 20 \times 3$ i $6 \times 50 \times 3$ te je zaključeno da arhitektura $6 \times 20 \times 3$ pokazuje bolje performanse.

Pametniji odabir inicijalnih težina i pristranosti jedinki doveo bi do mogućih poboljšanja modela. Mjesta za poboljšanje ima i kod selekcije, križanja i mutacije, pa bi odabir drugačijih i boljih operatora možda poboljšao performanse agenta. Kvalitetniji ulazni podaci za neuronsku mrežu s povećanim brojem parametara te dodatni slojevi i veći broj neurona u skrivenim slojevima još jedno su moguće poboljšanje.

LITERATURA

[1] Activation function. URL https://en.wikipedia.org/wiki/Activation_function. pristupljeno 7. lipnja 2023.

[2] Crossover in genetic algorithm, . URL https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/. pristupljeno 6. lipnja 2023.

[3] Crossover (genetic algorithm), . URL https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm). pristupljeno 6. lipnja 2023.

[4] Snake (video game genre), . URL https://wallpapers.com/wallpapers/classic-snake-game-design-jw1osayb8qr3c3e4.html). pristupljeno 1. lipnja 2023.

[5] Snake wallpaper, . URL https://en.wikipedia.org/wiki/Snake_(video_game_genre). pristupljeno 1. lipnja 2023.

[6] Ayla Angelos. The history of snake. URL https://www.itsnicethat.com/features/taneli-armanto-the-history-of-snake-design-legacies-230221. pristupljeno 1. lipnja 2023.

[7] B.D. Bašić i M. Čupić i J. Šnajder. Uvod u umjetnu inteligenciju, 2023.

[8] Hemalatha Raghavan. Activation function: Sigmoid. URL https://cloud2data.com/activation-function-sigmoid/. pristupljeno 7. lipnja 2023.

[9] Marko Čupić. *Evolucijsko računarstvo*. Fakultet elektrotehinke i računarstva, Zagreb, 2016. Prvo izdanje.

[10] Marko Čupić. *Umjetne neuronske mreže*. Fakultet elektrotehinke i računarstva, Zagreb, 2016. Prvo izdanje.

Razvoj agenta za igru zmija

Sažetak

U ovom radu opisan je razvoj računalnog agenta za igru zmija, čija pravila su objašnjena na početku rada. Agent je razvijen uz pomoć neuronskih mreža i genetskih algoritama, a arhitektura mreže i način odvijanja selekcije, križanja i mutacije opisan je u odgovarajućim poglavljima. Izrađeno je grafičko korisničko sučelje koje omogućuje igranje od strane čovjeka ili praćenje procesa treniranja agenta. Grafički su prikazani dobiveni rezultati u procesu treniranja i testiranja.

Ključne riječi: igra Zmija, umjetni neuron, umjetne neuronske mreže, genetski algoritmi

Development of an agent for the snake game

Abstract

This paper describes the development of a computer agent for the snake game, the rules of which are explained at the beginning of the paper. The agent is developed from neural networks and genetic algorithms, the architecture of the network and the methods of selection, crossover and mutation are described in the corresponding chapters. A graphical user interface has been created that allows playing by a human or monitoring the process of training an agent. The results obtained in the process of training and testing are shown graphically.

Keywords: Snake game, artificial neuron, artificial neural networks, genetic algorithms