SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1668

Simulacija bioma postupcima strojnog učenja

Vedran Pintarić

Zagreb, lipanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 15. ožujka 2018.

DIPLOMSKI ZADATAK br. 1668

Pristupnik: **Vedran Pintarić (0036476793)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Simulacija bioma postupcima strojnog učenja**

Opis zadatka:

Opisati pojam bioma te pravila po kojima se ravnaju navedeni biološki sustavi. Navesti postojeće pristupe simulaciji bioma i korištene računalne modele. Ostvariti programski sustav za simulaciju određene vrste bioma na temelju pravila interakcije između jedinki i okoliša. Ostvariti model ponašanja i programski sustav za optimizaciju ponašanja jedinki u simuliranoj okolini. Ispitati učinkovitost različitih modela ponašanja i algoritama učenja na temelju zadanih ograničenja i pravila okoline. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 29. lipnja 2018.

Mentor:
Prof. dr. sc. Domagoj Jakobović

Predsjednik odbora za diplomski rad profila:
Prof. dr. sc. Siniša Srbljić

Dielovođa:
Doc. dr. sc. Tomislav Hrkać

Zahvaljujem mentoru prof. dr. sc. Domagoju Jakoboviću na mentorstvu i pomoći u izradi rada. Zahvaljujem roditeljima, Mariju i Mirjani, na velikoj potpori tijekom ovih 17 godina školovanja.

SADRŽAJ

1. Uvod

Postoji širok spektar problema gdje se koriste optimizacijski algoritmi. Jedan od najfleksibilnijih optimizacijskih algoritama je genetski algoritam čiji je uzor priroda i teorija evolucije. Genetski algoritam probleme koje optimizira gleda kao crne kutije pa ne ovisi o vrsti problema. Cilj ovog rada je iskoristiti genetski algoritam za simuliranje evolucije iz stvarnosti te optimizirati ponašanje bića koje se pojavljuju unutar simuliranog svijeta.

Za stvaranje jednog takvog simuliranog svijeta potrebno je definirati sva pravila interakcije među objektima koji se nalaze u simulaciji. Za realističnu simulacija potrebna je i simulacija fizike gdje se iskorištava jedan od mnogih postupaka numeričke integracije. Osim stvaranja simulacije potrebno je istu i prikazati korisniku. Prikaz se ostvaruje pomoću SDL2 biblioteke i OpenGL API-a.

Uz sve navedeno potrebno je odabrati modele koji će služiti kao modeli po kojima se bića koja evoluiraju ponašaju. Nameće se nekoliko opcija a za implementaciju u ovome radu odabrane su neuronske mreže i stabala odluke. Izvršeno je uspoređivanje performansi ta dva modela nakon evoluiranja pod istim uvjetima.

2. Motivacija

2.1. Evolucija života

Od vremena prvih jednostaničnih organizama na Zemlji život se sastoji od genotipa i fenotipa. Genotip (Taylor i Lewontin, 2017) je skup gena koje organizam prenosi na svoje potomstvo. Pod fenotip (Taylor i Lewontin, 2017) spadaju svi atributi organizma - vanjski izgled, razvoj organizma, ponašanje, morfologija i ostalo.

Evolucija se u prirodi provodi kroz nekoliko mehanizama od kojih su bitniji mutacija, seksualna rekombinacija i prirodna selekcija.

Mutacija je spontana promjena genotipa organizma kojom se uvodi varijacija u cjelokupnu populaciju.

Seksualna rekombinacija je mehanizam koji omogućava novim organizmima nasljeđivanje roditeljskih genotipa. Kod aseksualnih organizama djeca nasljeđuju cijeli roditeljski genotip dok kod seksualnih organizama dijete nasljeđuje kombinaciju genotipa svojih roditelja.

Prirodna selekcija je mehanizam koji omogućava korisnim genima da budu nasljeđeni u novije generacije. Organizmi koji posjeduju genotip koji im omogućuje dulje preživljavanje u trenutnoj okolini će poživjeti dulje te će imati više prilika za reprodukciju i stvaranje potomstva koje će naslijediti taj genotip te ga nastaviti prenositi iz generacije u generaciju.

Pomoću simuliranja biološke evolucije jednostavnih organizama još jednom se pokazuje moć evolucije kao svojevrsnog optimizacijskog algoritma.

2.2. Genetski algoritam

Genetski algoritam je algoritam optimizacije koji izravno preuzima tehnike iz prirode opisane u 2.1.

Pri pokretanju algoritam inicijalizira nekoliko, najčešće nasumičnih, rješenja za

```
population = initializePopulation()
for i in range(0, numberOfGenerations):
        evaluate(population)
        newPopulation = crossover(population)
        mutate(newPopulation)
        population = newPopulation
```

Slika 2.1: Pseudokod genetskog algoritma

definirani problem. Skup rješenja nad kojima genetski algoritam provodi optimizaciju naziva se populacija rješenja. Nakon inicijalizacije rješenja algoritam kroz svaku iteraciju provodi evolucijske mehanizme nad populacijom s ciljem poboljšanja dobrote populacije. Pseudokod primjera jednog genetskog algoritma dan je na 2.1.

2.3. Postojeći radovi

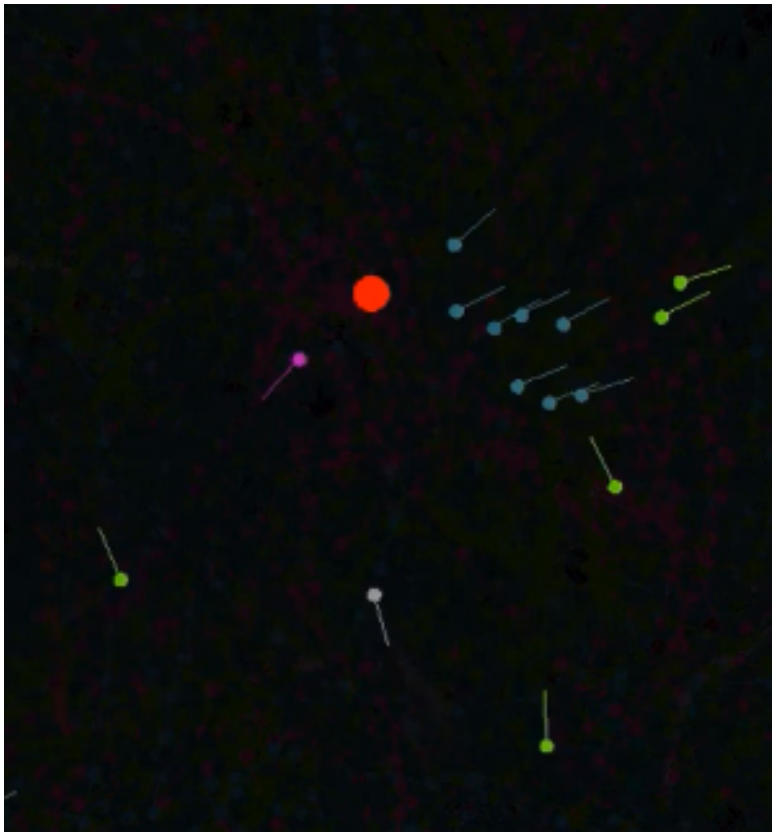
Već postoje implementirani projekti slične tematike koji obuhvaćaju simulaciju svijeta, igre ili slično te evoluiranje agenata unutar takve simulacije.

2.3.1. Genetski algoritam uz neuronske mreže

Projekti ove tematike koriste neuronske mreže kao modele inteligencije agenata unutar svoje simulacije. Za optimizaciju neuronskih mreža koriste genetski algoritam. Cilj agenata u ovim projektima je, najčešće, potraga za hranom i izbjegavanje opasnosti. Moguće je pronaći veliki broj primjera ovakvih projekata kratkim pretraživanjem Interneta.

Jedan od primjera je evoluiranje jednostavnih životinja tako da uspješno pronalaze hranu (Boxwell, 2016). Životinje u simulaciji (krugovi s "repom" vidljivi na slici 2.2) trebaju pronaći hranu (crveni krugovi) da bi stekli energiju za život i reprodukciju. Reprodukcija se vrši mitozom što znači da djeca imaju samo jednog roditelja te da su identična roditelju. Iz tog razloga varijacija u genotip populacije se vrši jedino od strane nasumičnih mutacija.

Još jedan od primjera je također evolucija životinja u 2D prostoru koje imaju cilj preživljavanja, međutim, ova simulacija je kompleksnija te postoji i međusobna direktna interakcija među životinjama (u projektu se zovu *Guppies*) (Oliver, 2012). Također su uvedeni i novi objekti u simulaciju - veliki *zapperi* plave boje koji nanesu štetu

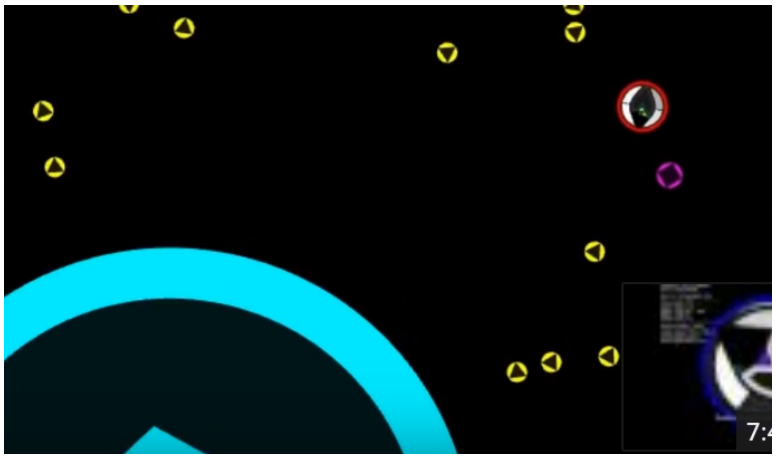


Slika 2.2: Simulacija evoluiranja jednostavnih 2D životinja

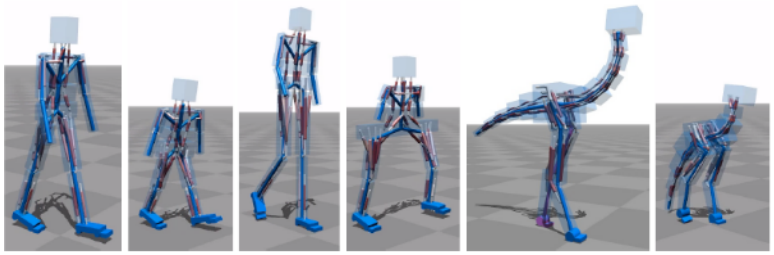
bićima pri doticaju, hranjive bobice žute boje te hranjivi ljubičasti ostaci mrtvih *Guppiesa*. Na isječku simulacije 2.3 moguće je vidjeti navedene elemente simulacije. Osim toga, implementirana je i svojevrsna "borba" između *Guppiesa* po uzoru na kamen-škare-papir igru. Prilikom sudara dviju *Guppyja* pobjednik se određuje pomoću trenutnih boja *Guppyja* te uzima energiju od gubitnika.

2.3.2. Genetski algoritam za optimizaciju kretanja

U projektu "Fleksibilno kretanje pomoću mišića za dvonožna bića" (Geijtenbeek et al., 2013) genetski algoritam je korišten kao optimizacijska metoda za učenje hodanja različitih 3D tijela čiji su primjeri prikazani na slici 2.4. Implementirana je simulacija interakcija među mišićima i kosturima tijela te simulacija fizike stvarnog svijeta. Modeli inteligencije imaju mogućnost upravljanja nad kontrakcijama mišića tijela te je cilj genetskog algoritma pronaći model koji će efikasno hodati po ravnoj podlozi, nagibu, zaobilaziti prepreke i slično. U radu je istaknuto da, nakon učenja, modeli uspješno prolaze kroz različite vrste problema i terena.



Slika 2.3: Evoluiranje *Guppiesa*



Slika 2.4: Razni modeli 3D tijela

3. Pravila simulirane okoline

Simulirana okolina je napravljena po uzoru na projektu *"Evolving Guppies"* koji je prikazan na slici 2.3. Okolina je dvodimenzionalna te ograničena kružnom granicom crvene boje. Unutar granica se može pojaviti nekoliko vrsta objekata koji imaju različito definirane međusobne interakcije.

3.1. Fizikalne zakonitosti

Unutar simulacije vrijede pravila fizike stvarnog svijeta u dvodimenzionalnom prostoru za neke od vrsta objekata. Objekti nad kojima vrijede pravila fizike imaju nekoliko atributa koji određuju njihovo trenutno stanje u simulatoru fizike:

- trenutna pozicija i kut rotacije
- trenutna translacijska i rotacijska brzina
- trenutna sila koja djeluje na objekt
- veličina (radijus)
- masa i moment tromosti

Pomoću navedenih vrijednosti koje definiraju trenutno stanje objekta moguće je izračunati stanje objekta u sljedećem okviru (eng. *frame*) simulacije.

3.1.1. Sudar objekata

Svi objekti su kružnog oblika što olakšava implementaciju detekcije sudar među objektima te smanjuje opterećenje na performanse računala. Sudar objekata se simulira kao realistični neelastični sudar kojim objekti u sudaru gube dio kinetičke energije. Osim kinetičkih energija objekata tijekom sudara uzimaju se u obzir i mase objekata za krajnju rezoluciju sudara.

3.1.2. Otpor zraka

Osim već navedenog u simulaciju fizike je uveden i otpor zraka. Otpor zraka se simulira tako da se suprotstavlja smjeru kretanja i rotacije objekata. Amplituda otpora se računa po sljedećim izrazima:

- za brzinu: $\frac{k_1 * v^2}{m}$
- za rotaciju: $\frac{k_2 * w^2}{I}$

Gdje su v translacijska brzina, w rotacijska brzina, m masa objekta, I moment tromosti objekta, k_1 i k_2 pozitivni realni koeficijenti. Razlog implementiranja otpora zraka je da onemogući brzini da dostigne visoke vrijednosti bez ulaganja velike količine energije.

3.2. Vrste objekata

Unutar granice okoline pojavljuju se četiri različite vrste objekata:

- *Minions* (životinje)
- Tijela
- Hranjive bobice
- Otrovne bobice

3.2.1. *Minions*

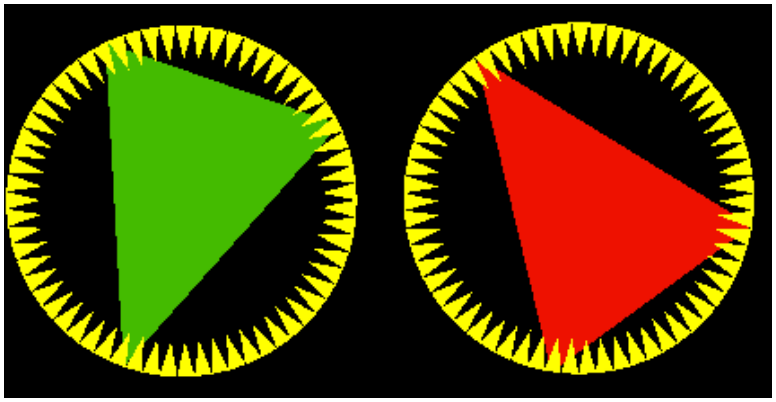
Minion se sastoji od tri komponente: objekta, osjetila i kontrolera.

Objekt

Pod objekt *miniona* podrazumijeva se fizičko "tijelo" koje se prikazuje na grafičkom sučelju simulacije i nad kojim se vrši detekcija sudara i simuliranje fizike. Objekt pamti trenutne atribute *miniona* unutar simulacije kao što su: brzina, akceleracija, sila koja djeluje na *miniona*, trenutno zdravlje, prijeđeni put tijekom života, duljina života, količina obnovljenog zdravlja, količina nanesene štete i drugi. Kako zdravlje *miniona* pada tako će tijelo objekta prelaziti iz zelene boje u crvenu boju kao što je vidljivo na slici 3.1. Kada zdravlje padne ispod nule, objekt se pretvara u mrtvo tijelo.

Objekt gubi zdravlje na nekoliko načina:

- kroz vrijeme
- sudarom s granicom simulacije



Slika 3.1: Grafički prikaz objekta *miniona* te promjene boje kako pada zdravlje

– sudar s drugim *minionom*

– sudar s otrovnom bobicom

– korištenjem sile za kretanje

Osjetila

Osjetila prikupljaju podatke iz okoline u svakom koraku simulacije te prosljeđuju te informacije kontroleru u obliku niza realnih brojeva. Unutar simulacije postoji mogućnost grafičkog prikaza osjetila. Osjetila se prikazuju kao siva kružnica s dužinama na prednjoj strani *miniona*.

Svaka linija osjetila radi kao zraka koja je ispaljena iz središta *miniona* te detektira prvi objekt s kojim se sudari kao što je prikazano na 3.2. Tako svaka linija daje informaciju koju vrstu objekta je detektirala i na kojoj udaljenosti od *miniona*. Linija može vratiti šest različitih vrijednosti ovisno o vrsti objekta kojeg je detektirala:

– ništa, u slučaju da se zraka nije s ničim sudarila do maksimalne granice osjetila

– živ *minion*

– mrtav *minion* (tijelo)

– hranjiva bobica

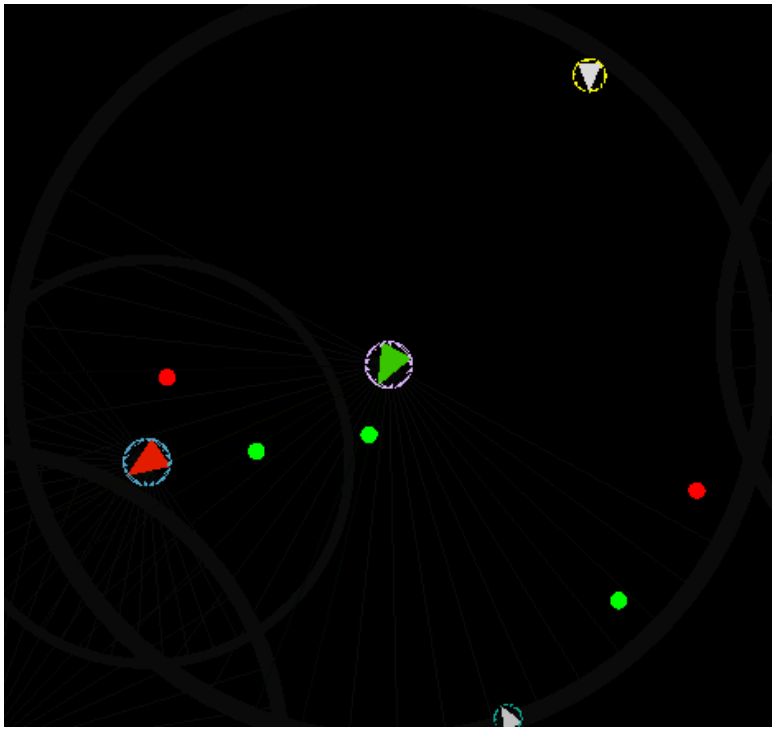
– otrovna bobica

– granica

Osim detekcije objekata ispred *miniona* osjetila vraćaju podatke o:

– udaljenosti od najbližeg objekta sa stražnje strane

– trenutnom zdravlju



Slika 3.2: Grafički prikaz osjetila unutar simulacije

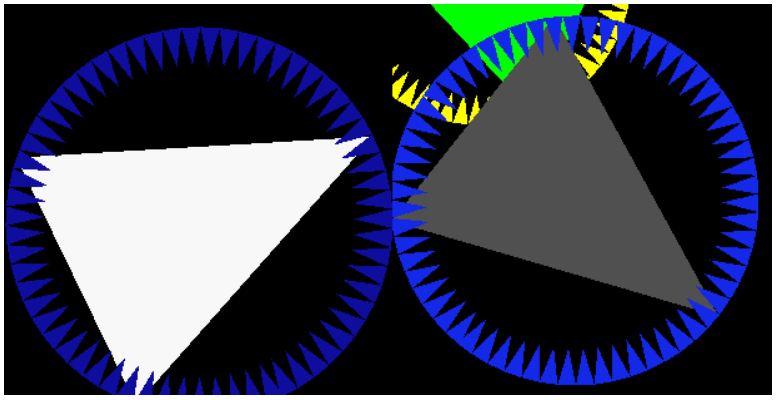
- trenutnoj brzini
- trenutnom kutu između vektora brzine miniona i vektora koji pokazuje u smjeru "naprijed"
- trenutnoj kutnoj brzini

Kontroler

Kontroler dobiva informacije o okolini te nekim atributima *minona* od senzora te na temelju tih informacija upravlja objektom *miniona*. Konkretno, kontroler na ulaz dobiva niz realnih brojeva od senzora a na izlaz vraća dva realna broja. Prvi broj na izlazu određuje silu u smjeru "naprijed" (ili "unazad" u slučaju da je broj negativan). Drugi broj određuje kutnu silu kojom se upravlja rotacija *miniona*.

Kontroler je glavna komponenta *miniona* koja se optimizira. U sklopu ovog projekta kontroler je implementiran na tri načina:

- ručno isprogramirano ponašanje (eng. *hardcoded*)
- neuronska mreža
- stablo odluke



Slika 3.3: Grafički prikaz tijela unutar simulacije te promjene boje tijekom raspadanja

3.2.2. Tijela

Tijela su ostaci mrtvih *miniona* te su kod njih komponente senzora i kontrolera deaktivirane te se kreću jedino inercijom i u slučaju sudara s nekim drugim objektom unutar simulacije. Tijela se prikazuju na isti način kao i živi *minioni* s razlikom da unutrašnji trokut modela ima nijansu sive boje. Boja trokuta prelazi iz bijele (svježe tijelo) prema crnoj (raspadnuto tijelo), kao što je prikazano na slici 3.3, te nakon toga tijelo nestaje iz simulacije.

Minioni mogu "jesti" tijela te tako obnoviti svoje zdravlje. "Jedenje" je simulirano tako da živi *minion* dođe u koliziju s tijelom.

Tijelo se raspada na tri načina:

- kroz vrijeme
- doticajem s granicom simulacije
- kada služi kao hrana nekom *minionu*

3.2.3. Bobice

Bobice su statički objekti koji se pojavljuju na nasumičnim mjestima unutar granice simulacije. Na isti način kao što *minioni* jedu tijela mogu jesti i bobice. Svaka bobica ima atribut koji pamti preostalu energiju te bobice. Nakon što energija bobice padne ispod nule bobica se reinicijalizira s punom energijom na neko drugo nasumično mjesto.

Postoje dvije vrste bobica - hranjive i otrovne. Jedenjem hranjivih bobica *minion* dobiva zdravlje dok jedenjem otrovnih bobica gubi zdravlje. Hranjive bobice su grafički prikazane kao puni zeleni krugovi dok su otrovne puni crveni krugovi.

4. Implementacija

Projekt je implementiran uz pomoć dodatnih biblioteka za grafički prikaz a prevođenje se izvodi uz pomoć *cmake* alata. Upute za instalaciju dane su u dodatku A.

4.1. Simulacija fizike

4.1.1. Numerička integracija

U svakom koraku simulacije ažuriraju se vrijednosti položaja, brzine i akceleracije objekata. Za ažuriranje je potrebna neka metoda numeričke integracije. Najjednostavnija metoda numeričke integracije, eksplicitna Eulerova metoda (slika 4.1 Dawkins (2018), je previše nestabilna. Zbog nestabilnosti numeričke integracije u svakom koraku (slika 4.3) se uvodi greška koja vodi k tome da se u sustav unosi dodatna energija. Konstantnim unosom energije u sustav dolazi do "eksplozije" simulacije zbog preljeva u *floating point* varijablama.

Problem nestabilnosti se djelomično može zaobići tako da se svaki korak simulacije metoda provodi u više koraka. To znači da se jedan vremenski korak simulacije podijeli na više manjih koraka te da se numerička integracija obavlja u tim manjim vremenskim koracima. Ovaj pristup poboljšava stabilnost ali drastično usporava brzinu izvođenja simulacije.

Umjesto eksplicitnog Eulera u simulaciji se koristi Runge Kutta metoda četvrtog reda (slika 4.2). Uz RK-4 metodu nije potrebno više izvođenja numeričke integracije za jedan korak simulacije. Metoda je mnogo stabilnija od eksplicitnog Eulera te njezina evaluacija drastično ne usporava simulaciju.

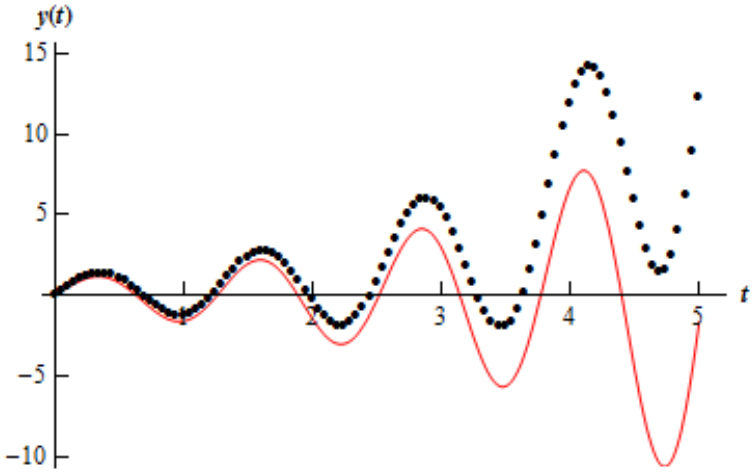
Numerička integracija i ažuriranje svojstava objekata izvode se u klasi *Object* koju

$$y_{n+1} = y_n + h * f(t_n, y_n)$$
$$t_{n+1} = t_n + h$$

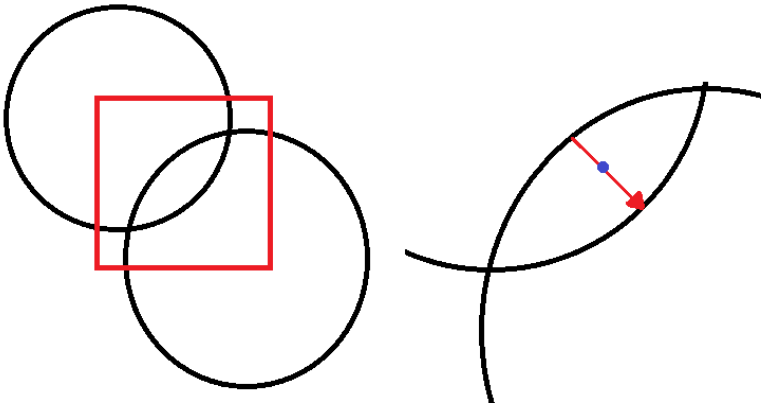
Slika 4.1: Metoda numeričke integracije eksplicitni Euler

$$
\begin{aligned}
k_1 &= h * f(t_n, y_n), \\
k_2 &= h * f(t_n + h/2, y_n + k_1/2), \\
k_3 &= h * f(t_n + h/2, y_n + k_2/2), \\
k_4 &= h * f(t_n + h, y_n + k_3), \\
y_{n+1} &= y_n + (1/6) * (k_1 + 2 * k_2 + 2 * k_3 + k_4), \\
t_{n+1} &= t_n + h
\end{aligned}
$$

Slika 4.2: Metoda numeričke integracije Runge-Kutta četvrtog reda



Slika 4.3: Primjer pogreške koju eksplicitna Eulerova metoda unosi kroz vrijeme



Slika 4.4: S lijeve strane je prikazan sudar dvaju krugova, na desnoj strani je crvenom bojom prikazan vektor u smjeru normale sudara i duljine za koju je potrebno odmaknuti objekte tako da nisu u sudaru a plavom bojom je prikazana točka sudara

nasljeđuju svi objekti nad kojima je potrebno izvoditi simulaciju sudara.

4.1.2. Detekcija sudara

Kako su svi objekti nad kojima je potrebno detektirati sudar kružnog oblika detekcija je jednostavna. *Minioni*, tijela i bobice su krugovi dok je granica simulacije kružni vijenac. Iz detekcije sudara potrebno je otkriti sljedeće vrijednosti:

- Koordinate točke sudara
- Normala na točku sudara (u slučaju sudara dva kruga to je vektor usmjeren od središta jednog kruga do središta drugog kruga)
- Vektor u smjeru normale čija duljina predstavlja udaljenost za koju je potrebno "odmaknuti" objekte tako da više nisu u sudaru

Na slici 4.4 je prikazan sudar dvaju krugova.

Klasa koja obavlja detekciju sudara te vraća potrebne podatke o sudaru je *CollisionDetection*.

4.1.3. Rezolucija sudara

Pod rezoluciju sudara se podrazumijevaju ažuriranja položaja i brzina objekata nakon sudara. Prvotno je potrebno pomaknuti objekte u smjeru normale sudara za udaljenost koju definira duljina vektora koji je prikazan crvenom bojom na slici 4.4. Kako se takvim trenutnim pomakom objekata unosi energija u sustav potrebno je minimizirati unesenu energiju. Način na koji se to postiže je da je udaljenost za koju se pomiče

objekt obrnuto proporcionalna masi objekta. Tako se čuva i realističnost simulacije u prikazivanju stvarnog svijeta.

Nakon pomaka objekata računa se stanje brzina objekata nakon sudara pomoću izraza klasične mehanike Smid (2018). Sudar je implementiran kao generalizirani neelastični sudar u kojem se gubi 99% kinetičke energije (koeficijent restitucije je 0.1). Klasa koja provodi navedene proračune te ažurira objekte sukladno tome je CollisionResponse.

4.2. Grafički prikaz

Za grafički prikaz simulacije koristi SDL2 (*Simple DirectMedia Layer*) biblioteka uz OpenGL API verzije 3.0. SDL2 se koristi kao jednostavno *cross-platform* rješenje za stvaranje prozora za grafički prikaz te, osim toga, služi za dohvaćanje pritisaka tipki s tipkovnice od strane korisnika. OpenGL koristi se za renderiranje trenutnog stanja simulacije u svakom koraku na prozor stvoren od strane SDL-a.

Klase koje implementiraju pomoćne metoda i strukture podataka za potrebe grafičkog prikaza su Camera, ColorModel, Model i Renderer. Model i ColorModel su klase koje sadrže podatke za renderiranje pojedinih modela koji su korišteni u simulaciji (*minioni*, bobice...). Camera sadrži podatke o trenutnom položaju kamere u svijetu simulacije te sadržava metode koje upravljaju pomicanjem i zumiranjem kamere. Renderer klasa sadrži podatke o matricama transformacije te generalne pomoćne metode potrebne pri renderiranju objekata.

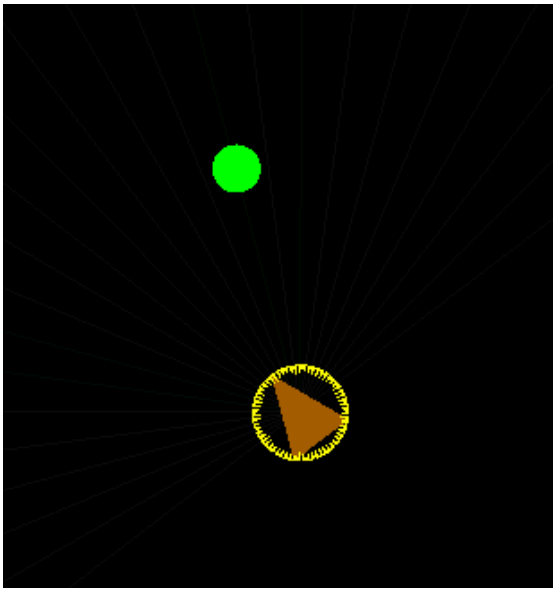
4.3. Implementirani modeli kontrolera

4.3.1. textitMinioni fiksnog ponašanja

Ova vrsta *miniona* služi samo kao dodatni element pri učenju druga dva modela kontrolera. Ponašanje ovih *miniona* direktno je "upečeno" u kod simulacije. Ovi *minioni* se mogu prepoznati po njihovoj žutoj boji kože (vanjskog zupčastog kruga).

Minioni ove vrste reagiraju samo na objekt koji im je najbliže te će pokušati pobjeći ili uloviti objekt ovisno o kontekstu u kojem se nalaze. Pravila ponašanja ovisno o tipu objekta kojeg vidi i koji mu je najbliži su sljedeća:

- granica simulacije ili otrovna bobica - nastoj izbjeći
- tijelo ili hranjiva bobica - nastoj se približiti umjerenom brzinom



Slika 4.5: *Minion* fiksnog ponašanja uspješno detektira hranjivu bobicu te se okreće prema njoj

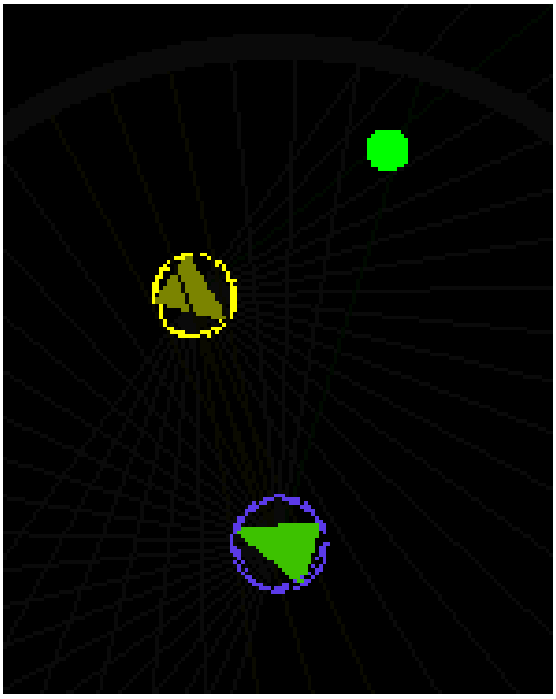
- živi *minion* - u slučaju da imaš više od pola zdravlja nastoj se zaletjeti direktno i čim brže a u slučaju da imaš manje od pola zdravlja nastoj izbjeći
- ništa - u slučaju da osjećaš nešto iza sebe okreni se i detektiraj tip objekta u protivnom nastoj zaustaviti rotaciju i kreći se sporom brzinom ravno

Na slikama 4.5 i 4.6 prikazan je *minion* fiksnog ponašanja unutar simulacije te je opisano njegovo ponašanje u tim situacijama.

4.3.2. Neuronska mreža

Jedan od modela ponašanja *miniona* u simulaciji je unaprijedna potpuno povezana neuronska mreža, skica primjera jedne takve mreže prikazana je na slici 4.7. Svaki *minion* koji koristi ovaj model ponašanja ima dvije neuronske mreže koje upravljaju njegovim ponašanjem. Jedna neuronska mreža za izlaz daje vrijednost za silu koja upravlja akceleracijom *miniona* dok druga za izlaz daje rotacijski moment koji upravlja kutnom akceleracijom *miniona*.

Obje mreže na ulaz dobivaju cijeli niz normaliziranih realnih brojeva dobivenih od osjetila. Veličina niza podataka na ulazu mreže je 55, od toga 50 je od osjetila koja detektiraju objekte ispred *miniona* a 5 od raznih podataka o samom *minionu* opisanih u 3.2.1. Podaci dobiveni od osjetila se normaliziraju u klasi osjetila na vrijednosti u rasponu od -1 do 1. Razlog za normalizaciju je potencijalno brža optimizacija mreža jer se eliminiraju velike razlike u rasponima podataka - npr. udaljenost nekog objekta



Slika 4.6: *Minion* fiksnog ponašanja detektira konkurentnog *miniona* te ga izbjegava, nakon toga detektira hranjivu bobicu te se okreće prema njoj

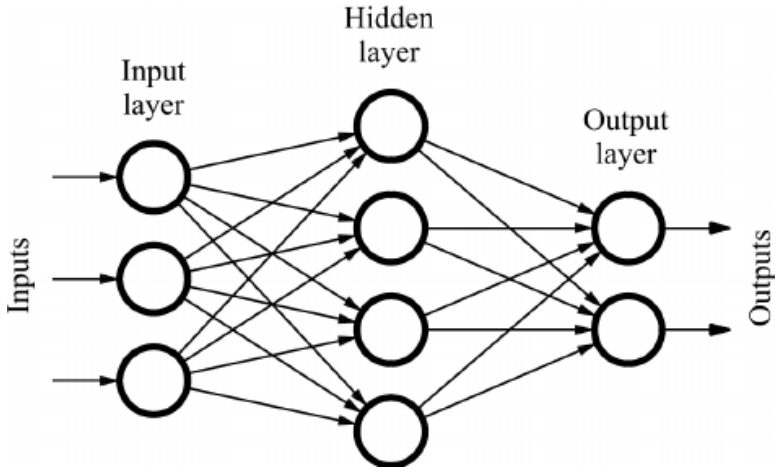
se može kretati u rasponu od 0 do 1000 dok se zdravlje *miniona* kreće u rasponu 0 do 100 što može uzrokovati poteškoće pri optimizaciji težina neuronske mreže. Zbog regularizacije podataka težine i *bias* neuronske mreže su ograničeni na isti raspon od -1 od 1.

4.3.3. Stablo odluke

Drugi implementirani model ponašanja u simulaciji je stablo odluke. Skica primjera jednostavnog stabla odluke prikazana je na slici 4.8. Svaki *minion* koji koristi ovaj model ponašanja ima dva stabla odluke u svojem kontroleru. Isto kao i s neuronskim mrežama jedno stablo odluke upravlja silom a jedno kutnim momentom.

Kontroler sa stablima odluke radi tako da na ulaz dobije niz realnih brojeva dobivenih od osjetila te ih pretprocesira te nakon toga provodi podatke kroz stabla odluke da dobije krajnje vrijednosti za upravljanje *miniona*. Pretprocesiranje služi tome da se dobiju činjenice (jednostavne *boolean* vrijednosti) koje stablo odluke može ispitivati u svojim čvorovima grananja. Činjenice koje kontroler može dobiti iz pretprocesiranja su:

- detekcija konkurentnog miniona - *FACT_MINION_LEFT*, *FACT_MINION_FRONT* i *FACT_MINION_RIGHT*

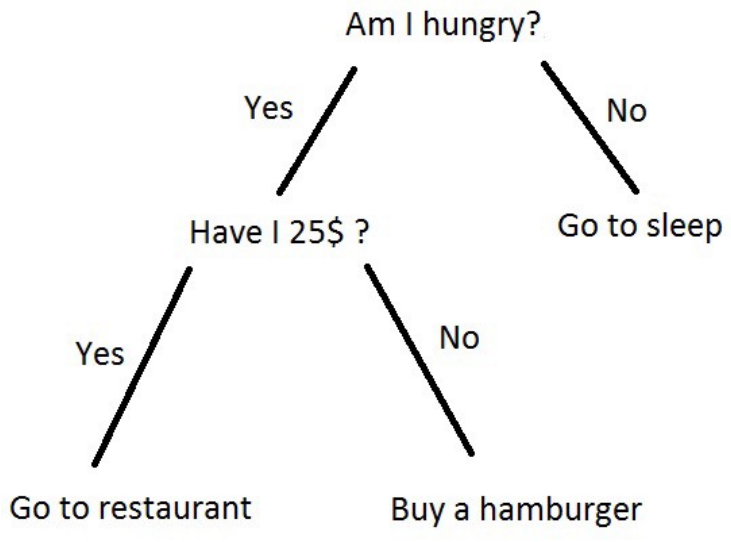


Slika 4.7: Primjer unaprijedne potpuno povezane neuronske mreže s jednim skrivenim slojem veličine 4

- detekcija tijela - *FACT_BODY_LEFT*, *FACT_BODY_FRONT* i *FACT_BODY_RIGHT*
- detekcija granice - *FACT_BOUNDARY_LEFT*, *FACT_BOUNDARY_FRONT* i *FACT_BOUNDARY_RIGHT*
- detekcija hranjive bobice - *FACT_FOOD_LEFT*, *FACT_FOOD_FRONT* i *FACT_FOOD_RIGHT*
- detekcija otrovne bobice - *FACT_POISON_LEFT*, *FACT_POISON_FRONT* i *FACT_POISON_RIGHT*
- detekcija nekog objekta sa stražnje strane - *FACT_SOMETHING_FAR_BEHIND* i *FACT_SOMETHING_CLOSE_BEHIND*
- činjenice o trenutnom zdravlju - *FACT_HEALTHY* i *FACT_UNHEALTHY*
- činjenice o trenutnoj translacijskoj brzini - *FACT_MOVING_FAST* i *FACT_MOVING_SLOW*
- činjenica je li *minion* poravnan sa smjerom u kojem se kreće ili ne - *FACT_FRONT_ALIGNED* i *FACT_FRONT_MISALIGNED*
- činjenica o trenutnoj rotacijskoj brzini - *FACT_ROTATING_CLOCKWISE* i *FACT_ROTATING_C*

Svaki čvor grananja ispituje jednu od tih činjenica te ovisno je li ta činjenica istinita ili ne bira put grananja na sljedeći čvor dijete. Svaki od terminalnih čvorova (listova stabla) sadrži jednu od diskretnih vrijednosti. Kada evaluacija danih činjenica kroz stablo dođe do nekog terminalnog čvora stablo kao izlaz daje vrijednost u tom terminalnom čvoru.

Ovisno radi li se o stablu za akceleraciju ili rotaciju stablo može vratiti sljedeće diskretne vrijednosti - ako se radi o akceleraciji:



Slika 4.8: Primjer stabla odluke dubine 3

- različite razine pozitivnog ubrzanja: *ACC_FAST_FORWARD*, *ACC_FORWARD* i *ACC_SLOW_FORWARD*
- akceleracija je nula: *ACC_ZERO*
- različite razine negativnog ubrzanja: *ACC_SLOW_BACKWARD*, *ACC_BACKWARD* i *ACC_FAST_BACKWARD*,

, ako se radi o rotaciji:

- različite razine rotacije u smjeru kazaljke na satu: *ROT_FAST_CLOCK*, *ROT_CLOCK* i *ROT_SLOW_CLOCK*
- rotacija je nula: *ROT_ZERO*
- različite razine rotacija u smjeru suprotnom od kazaljke na satu: *ROT_SLOW_CCLOCK*, *ROT_CCLOCK* i *ROT_FAST_CCLOCK*

4.4. Dobrota *miniona*

4.4.1. Prvi način izračunavanja

U prvom načinu izračunavanja dobrote *miniona* za dobrotu *miniona* izravno se uzima duljina života *miniona*. U ovakvom izračunavanju dobrote *minioni* nemaju poticaja za istraživanjem širokog spektra ponašanja pa da postepeno dolaze do kompleksnijih ponašanja koja im omogućuju dugotrajno preživljavanje. Umjesto postepenih poboljšanja

minioni upadaju u lokalni optimum gdje svi nauče ne raditi ništa kako bi štedjeli energiju. Kako ovakvo ponašanje nije zanimljivo za proučavanje bilo je potrebno pronaći bolji način izračunavanja dobrote koji će poticati *minione* na aktivnija ponašanja.

4.4.2. Drugi način izračunavanja

U ovom načinu dobrota *miniona* je definirana zbrojem četiri različite komponente skalirane koeficijentima koji se mogu definirati u konfiguraciji. Te četiri komponente su:

- Prijeđena udaljenost - u određenim intervalima se zapisuje trenutni položaj *miniona* te se na ovu komponentu pribraja udaljenost između prijašnjeg zapamćenog položaja i trenutnog
- Duljina života - simulacijsko vrijeme koje određuje koliko dugo je *minion* živio
- Nanesena šteta - količina štete na zdravlje konkurentnih *miniona* koje je počinio ovaj *minion*
- Obnovljeno zdravlje - količina zdravlja koje je ovaj *minion* obnovio tako da je jeo tijela i/ili hranjive bobice

Iako su *minioni* u mogućnosti proizvesti ponašanja koja ih tjeraju da požive čim dulje samo uz komponentu "duljina života", ideja s ostalim komponentama je da se olakša i ubrza evolucija takvog ponašanja. Komponenta "prijeđena udaljenost" bi trebala tjerati *minione* na istraživanje svoje okoline dok bi ih komponente "nanesena šteta" i "obnovljeno zdravlje" trebale tjerati na aktivnije lovljenje hrane i borbu s konkurentima.

Osim optimiziranja ponašanja *miniona* optimizira se i njihov radijus. Optimizacijski algoritam direktno utječe na radijus *miniona* ali time indirektno utječe na početnu količinu zdravlja i masu *miniona*. Početno zdravlje i masa proporcionalni su radijusu. Ukratko, to znači da što je *minion* veći to je tromiji i sporije reagira kretanjem ali je masivniji te ga je teže oštetiti dok on radi više štete konkurentima.

4.5. Genetski algoritam

U simulaciji je implementiran generacijski genetski algoritam. Generacijski genetski algoritmi rade uz populacije rješenja te jedna generacija algoritma stvara cijelu jednu novu populaciju iz stare populacije.

Konkretno, implementirani algoritam prvo stavlja neki broj najboljih jedinki u sljedeću populaciju što se naziva elitizam. Nakon toga radi petlju selekcija-križanje-mutacija te tako stvara nove jedinke kojima popunjava sljedeću populaciju. Nakon što je popunjena nova populacija jedinke u toj populaciji se evaluiraju tako da se izračuna njihova dobrota. Time završava jedna generacija algoritma te započinje sljedeća.

4.5.1. Selekcija roditelja

Vjerojatnost da je neki *minion* u generaciji odabran za roditelja u operaciji križanja ovisi o njegovoj evaluiranoj dobroti - *minioni* veće dobrote imaju veću vjerojatnost da budu izabrani. Na ovaj način selekcija uzima samo dobar genetski materijal za sljedeću generaciju, gdje definicija dobroga ovisi o definiciji dobrote *miniona* koja se koristi.

Evaluacija *miniona*

Kako na prosječnom računalu simulacija radi prihvatljive brzine za oko 30 *miniona* odjednom nemoguće je genetski algoritam pokrenuti s velikim brojem jedinki u populaciji (npr. 100). Iz tog razloga ne evaluiraju se sve jedinke odjednom već se populacija dijeli na manje particije koje je moguće efikasno evaluirati u simulaciji. Kada se sve particije populacije evaluiraju, populacija se smatra evaluiranom te svaki *minion* ima svoju vrijednost dobrote.

4.5.2. Operatori križanja

Operatori križanja u genetskim algoritmima služe za stvaranje novog djeteta (novog rješenja) od, najčešće, dva roditelja (Padmavathi Kora, 2017). Cilj operatora je u prostoru svih rješenja dijete postaviti u prostor između dvaju roditelja. Ako bi trenutnu populaciju rješenja gledali kao višedimenzionalno tijelo korištenje operatora križanja bi sužavalo to tijelo što znači da križanje, uz selekciju roditelja na temelju dobrote, usmjerava pretragu prema boljim rješenjima. Zato križanje možemo gledati kao metodu intenzifikacije kod metaheurističkih optimizacijskih algoritama (Yang, 2011).

Križanje se može implementirati na različite načine ovisno o kakvoj strukturi podataka se radi.

Radijus

Uzima se aritmetička sredina roditelja.

Neuronske mreže

Za križanje neuronskih mreža implementirana su dva operatora: aritmetička sredina s jednom točkom križanja i kopiranje jednog sloja. Uvjet za ove operatore je da su mreže jednakih arhitektura. Oba operatora prvotno provjeravaju koji od roditelja ima veću odnosno manju dobrotu.

Prvi operator križanja prvotno nasumično odabire neku točku između dva sloja u mrežama. Mreža dijete se stvara tako da se do odabrane točke kopiraju slojevi roditelja s većom dobrotom. Nakon odabrane točke slojevi djeteta se stvaraju tako da se radi aritmetička sredina koeficijenata težina i pristranosti slojeva roditelja. Grafički prikaz rada operatora na jednostavnim mrežama prikazan je na slici 4.10.

Drugi operator križanja prvotno nasumično odabire neki od slojeva. Mreža dijete se stvara tako da se odabrani sloj kopira od roditelja s manjom dobrotom dok se svi ostali slojevi kopiraju od roditelja s većom dobrotom. Grafički prikaz rada operatora prikazan je na slici 4.9.

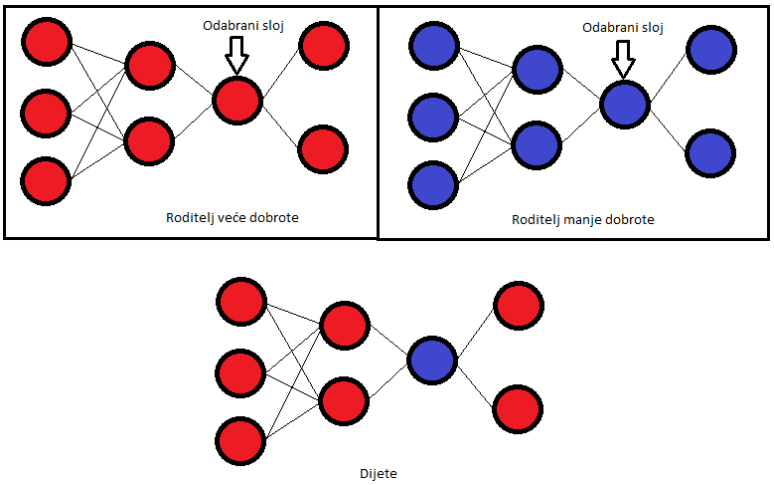
Stabla odluke

Implementirani operatori križanja i mutacije za stabla odluke koriste pomoćnu operaciju podrezivanja stabla u slučaju da stablo koje su generirali ima dubinu veću od maksimalne dopuštene dubine određene u konfiguraciji. Podrezivanje radi tako da se od podstabla koje prodire ispod maksimalne dubine nasumično uzima neki list tog podstabla te se cijelo podstablo zamijeni tim listom. Podrezivanje je prikazano na slici 4.11.

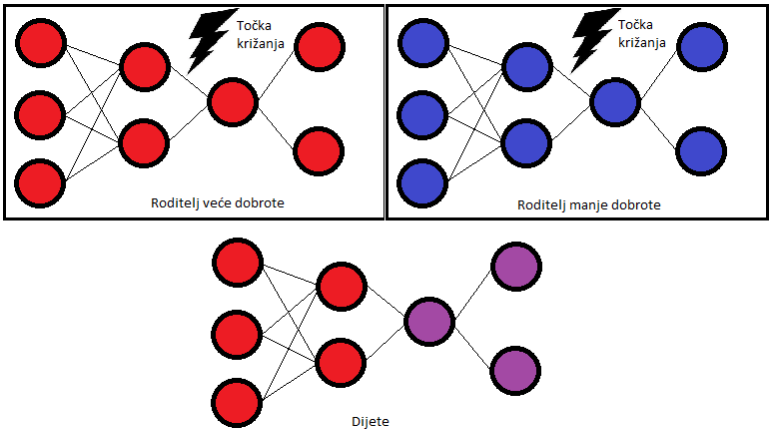
Implementirani operator križanja za stabla odluke prvotno nasumično odabire jednog od roditelja. Nakon toga odabire nasumični čvor odabranog roditelja gdje veću vjerojatnost odabira imaju čvorovi na većoj dubini. Podstablo čiji je korijen odabrani čvor zamjenjuje s drugim roditeljem. Kako nakon ove operacija postoji mogućnost da je stablo veće dubine od maksimalne dubine obavlja se operacija podrezivanja stabla. Grafički prikaz rada opisanog operatora prikazan je na slici 4.12.

4.5.3. Operatori mutacije

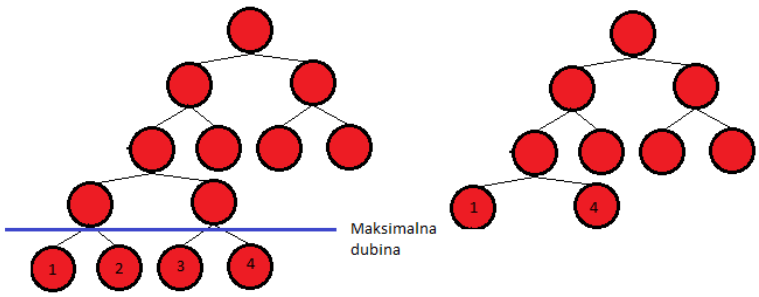
Operatori mutacije, u kontrastu s operatorima križanja, svojim djelovanjem proširuju veličinu višedimenzionalnog tijela kojeg tvori trenutna populacija rješenja u prostoru rješenja. Kod metaheurističkih optimizacijskih algoritama to se naziva diverzifikacija pretrage. Cilj toga jest izbjegavanje lokalnih optimuma pri pretraživanju te tjeranje



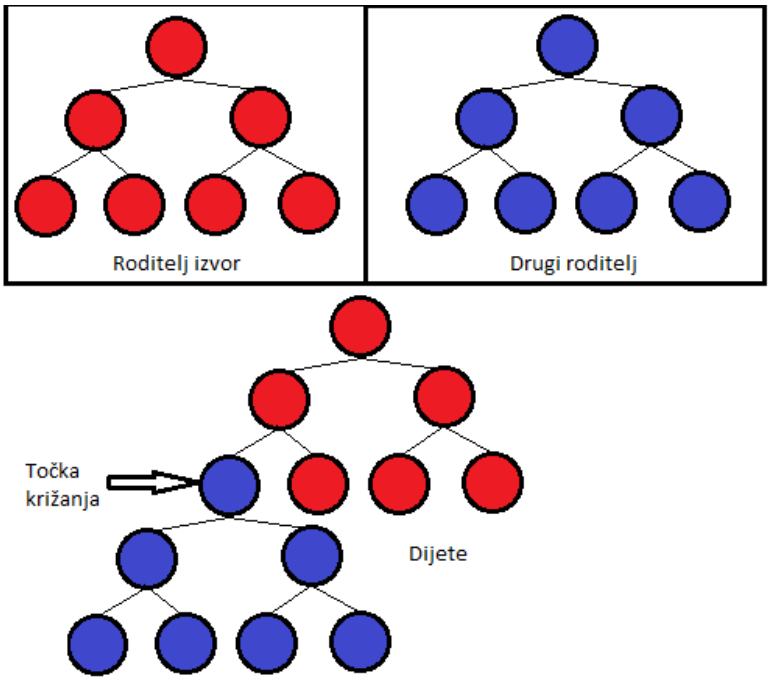
Slika 4.9: Grafički prikaz operatora križanja koji kopira jedan sloj iz lošijeg roditelja



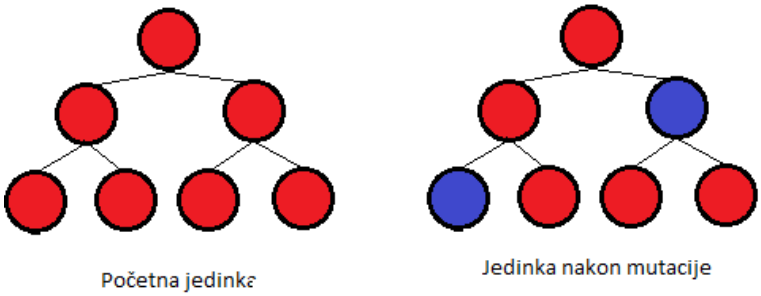
Slika 4.10: Grafički prikaz operatora križanja koji za dio slojeva radi aritmetičku sredinu



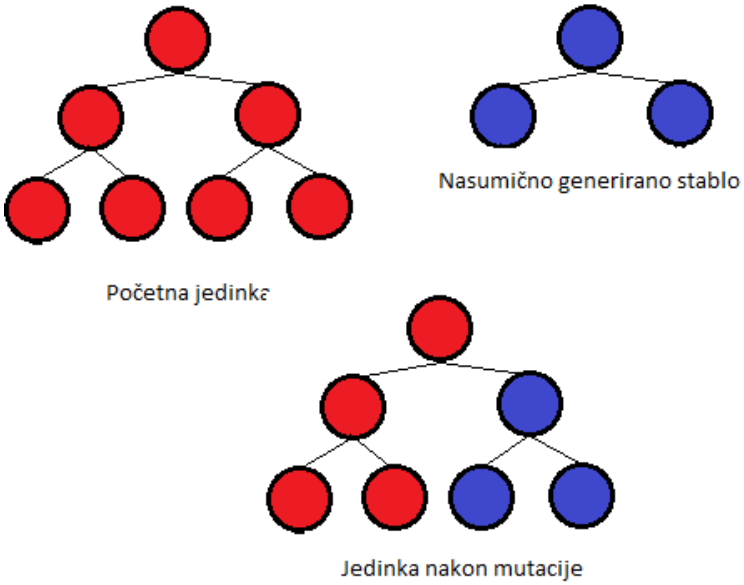
Slika 4.11: Grafički prikaz operacije podrezivanja stabla



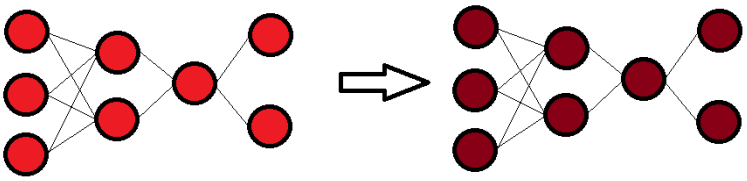
Slika 4.12: Grafički prikaz operatora križanja koji zamjenjuje podstablo jednog roditelja s cijelim stablom drugog roditelja



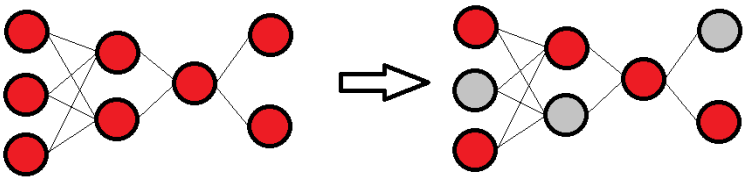
Slika 4.13: Grafički prikaz nasumične mutacije pojedinih čvorova stabla



Slika 4.14: Grafički prikaz zamjene podstabla nasumično generiranim stablom



Slika 4.15: Grafički prikaz mutacije svih neurona



Slika 4.16: Grafički prikaz mutacije koja resetira neke neurone

pretrage da pronađe bolja rješenja umjesto stagniranja.

Radijus

Dodaje se realni broj uzorkovan iz normalne distribucije čiji se parametri postavljaju u konfiguraciji.

Neuronske mreže

Za neuronske mreže implementirana su dva operatora mutacije.

Prvi operator mutacije na svaku težinu i pristranost neuronske mreže dodaje realni broj iz Gaussove distribucije čiji su parametri dio konfiguracije. Nasumično generirani brojevi trebali bi biti dovoljno maleni da ne uzrokuju prevelike perturbacije u rješenju a, na drugu ruku, dovoljno veliki da unesu primjetne promjene. Grafički prikaz se nalazi na slici 4.15.

Drugi implementirani operator nasumično odabire težine i/ili pristranosti te ih nanovo postavlja. Postavljanje se odvija tako da se uzorkuje realni broj iz Gaussove distribucije čiji su parametri navedeni u konfiguraciji. Grafički prikaz se nalazi na slici 4.16.

Stabla odluke

Kao i za neuronske mreže za stabla odluke implementirana su dva operatora mutacije.

Prvi operator generira privremeno nasumično stablo. Nakon toga odabire nasumični čvor u stablu kojeg mutira s time da čvorovi na većoj dubini imaju veću vjerojatnost biti odabrani. Tada podstablo čiji je korijen odabrani čvor zamjenjuje s nasumično generiranom stablom. Moguće je da krajnje stablo ima dubinu veću od maksimalne dopuštene te se još dodatno radi i podrezivanje. Grafički prikaz operatora nalazi se na slici 4.14.

Drugi implementirani operator nasumično mutira čvorove stabla. Čvorovi grananja imaju vjerojatnost da im se nasumično promjeni vrijednost koju ispituju pri grananju dok čvorovi listovi imaju vjerojatno da im se nasumično promjeni vrijednost koju vraćaju. Grafički prikaz operatora nalazi se na slici 4.13.

5. Rezultati

5.1. Opis ispitivanja

Ispitana je evolucija *miniona* s neuronskom mrežom odnosno stablima odluke kao model ponašanja. Svaka evolucija je odrađena zasebno kroz 900 generacija (što je, uz 5 particija po generaciji, 4500 izvođenja simulacije s ciljem evaluiranja dobrote) te je svakih 300 generacija proučeno kakvo je ponašanje evoluiralo. Krajnji cilj je usporedba performansi neuronskih mreža i stabala odluke nakon evolucije.

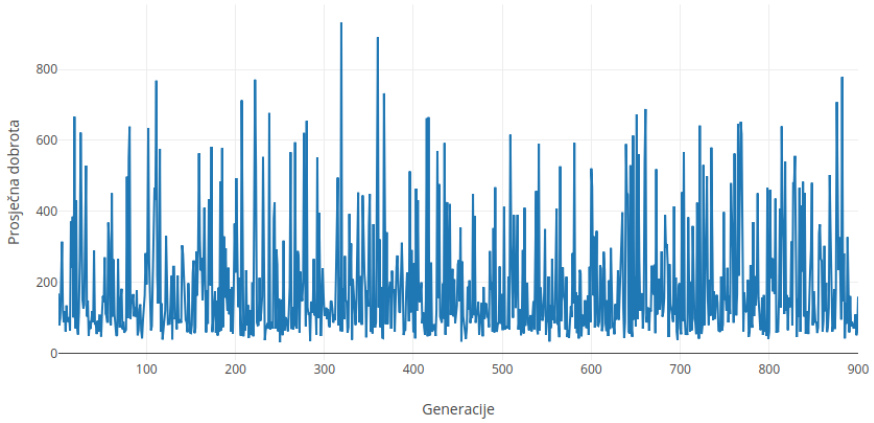
Osim *miniona* s modelima ponašanja koji evoluiraju unutar simulacije će se pojavljivati i *minioni* s fiksnim modelom ponašanja koji su opisani u 4.3.1. *Minioni* fiksnog ponašanja dodani su u simulaciju da bi uveli konkurentnost u sustav te da bi kažnjavali gene koji ne pridonose preživljavanju u takvoj okolini.

5.2. Zajednička korištena konfiguracija

Za ispitivanje oba modela dio korištene konfiguracije je jednak. Oba modela koriste istu funkciju izračunavanja dobrote s istim parametrima. Korištena funkcija dobrote opisana je u 4.4.2 a korišteni koeficijenti za pojedine komponente su:

- 1.0 - duljina života
- 5.0 - napravljena šteta
- 7.5 - oporavljeno zdravlje
- 0.001 - prijeđena udaljenost

Radijus granice je 1500 jedinica. Broj *miniona* u populaciji je 100 te su podijeljeni na 5 particija po generaciji te se svaka particija evaluira zasebno. U simulaciji se uvijek nalaze po 10 hranjivih bobica i 5 otrovnih bobica. Osim *miniona* koji evoluiraju ponašanje pri inicijalizaciji svake particije prije njene evaluacije inicijalizira se i 10 *miniona* s fiksnim ponašanjem opisanih u 4.3.1. Elitizam genetskog algoritma postavljen



Slika 5.1: Prosječna dobrota *miniona* s modelom neuronskih mreža kroz 900 generacija evolucije

je na 20 jedinki što znači da će 20 najboljih jedinki u generaciji biti direktno prekopirane u sljedeću generaciju.

Detaljan pregled svih mogućnosti u konfiguracijskoj datoteci dan je u dodatku B.

5.3. Neuronske mreže

5.3.1. Konfiguracija

Za arhitekturu neuronskih mreža koriste se dodatna 2 skrivena sloja s 80 odnosno 30 neurona.

Operatori križanja koji se koriste opisani su u 4.5.2. Operator križanja koji radi aritmetičku sredinu između roditelja na dijelu mreže izvršava se s 90% vjerojatnosti dok se operator koji kopira jedan sloj od lošijeg roditelja izvršava s 10% vjerojatnosti.

Operatori mutacije koji se koriste opisani su u 4.5.3. Operator koji dodaje Gaussov šum na težine i pristranosti mreže koristi se s 75% vjerojatnosti i uzorkuje šum iz normalne distribucije varijance 0.1 i srednje vrijednosti 0. Drugi operator koji ponovno postavlja težine i pristranosti koristi se s 25% vjerojatnosti. Operator za svaki koeficijent težine i pristranosti ima 5% vjerojatnosti da će ponovno postaviti taj koeficijent. Vrijednost na koju će postaviti koeficijente uzorkuje iz normalne distribucije varijance 0.2 i srednje vrijednosti 0.

5.3.2. Prosječna dobrota

Graf prosječne dobrote *miniona* kroz 900 generacija prikazan je na slici 5.1. Može se vidjeti da se prosjek dobrote ponaša vrlo kaotično kroz generacije. Prosječna dobrota ovisi o uvjetima (položaji *miniona*, hrane i otrova) u kojima se *minioni* nađu na početku evaluacije. Ti uvjeti se nasumično iniciraju te se *minion* koji je naučio neko specifično ponašanje koje je uspješno za jednu situaciju neće nužno dobro ponašati u nekoj drugoj situaciji. Drugi razlog kaotičnosti grafa je konkurentnost *miniona* pri evaluaciji. Tijekom evaluacije jedne particije generacije *minioni* se bore jedni protiv drugih te nužno postoje gubitnici i pobijednici. Kako dobrota individualnog *miniona* ovisi o ponašanju svih *miniona* u particiji i početnim uvjetima simulacije ovakav graf prosječne dobrote je očekivan. Ovakvo kretanje prosječne dobrote nužno ne znači da *minioni* ne razvijaju korisna ponašanja koja im pomažu preživjeti unutar simulacije. Osim navedene prosječne dobrote *miniona* tijekom evolucije najviša dostignuta dobrota je 6826.23 u generaciji 416.

5.3.3. Uočena ponašanja nakon 300 generacija

Minioni fiksnog ponašanja u većini slučajeva nadživljavaju *minione* koji za model ponašanja imaju neuronske mreže, međutim, neuronske mreže su napredovale te su vidljiva neka istaknuta ponašanja. Evoluirana ponašanja imaju smisla iz perspektive pravila senzora i općenitih pravila simulacije te pružaju dobru osnovu za daljnje nadogradnje na ta ponašanja.

Rotiranje

Ponašanje koje su usvojili svi *minioni* je konstantno rotiranje oko vlastite osi. Iako konstantnim rotiranjem *minioni* troše više energije svi su usvojili to ponašanje. Mogući razlog tome je ograničenost senzora da detektiraju objekte samo na prednjem dijelu *miniona* te rotiranjem *minion* konstantno provjerava u cijelom krugu oko svojeg položaja. U kombinaciji s ostalim usvojenim ponašanjima rotacija se ispostavlja vrlo korisnom.

Kretanje u smjeru drugih objekata

Većina *miniona* je usvojila kretanje u smjeru drugih objekata koje detektiraju ali ne koriste informaciju koja vrsta objekta je detektirana te postoji mogućnost da se *minion* približava nečemu štetnome kao što je granica ili otrovna bobica.

Zabijanje

Neki od *miniona* koji su usvojili prijašnje opisano ponašanje usvojili su i zabijanje u druge objekte. Kada detektiraju da im je neki objekt vrlo blizu i izravno ispred njih daju jako visoku akceleraciju prema naprijed te se efektivno zabijaju u detektirani objekt. Kako zabijanjem većom brzinom u konkurentske *minione* radi više štete ovo je pozitivno ponašanje.

Neodlučnost

Veliki udio *miniona* je neodlučno kada detektiraju nekoliko objekata odjednom na različitim položajima. Pokušavaju se približiti svim objektima odjednom te se na kraju ne pomaknu sa svojeg početnog položaja.

5.3.4. Uočena ponašanja nakon 600 generacija

Minioni fiksnog ponašanja i dalje konzistentno nadživljavaju *minione* s neuronskim mrežama. I dalje je jedno najistaknutijih ponašanja rotiranje. Najistaknutija promjena u odnosu na provjeru na 300. generaciji je manja agresivnost te većina *miniona* pokušava izbjegavati detektirane objekte.

Manja agresivnost

U kontrastu na 300. generaciju ova generacija *miniona* je manje agresivna te umjesto da se kreću prema detektiranim objektima većinom se kreću od njih.

Izbjegavanje opasnosti

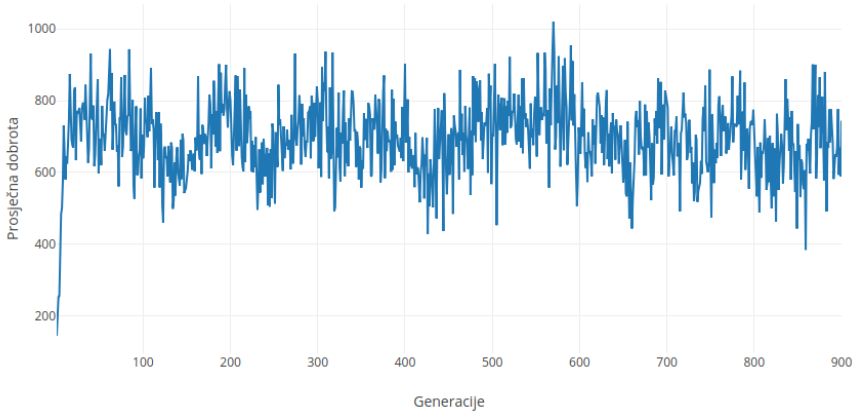
Jedna od zanimljivijih ponašanja je da je većina *miniona* naučila konzistentno izbjegavati granicu simulacije i otrovne bobice dok su u vezi ostalih objekata neodlučni.

5.3.5. Uočena ponašanja nakon 900 generacija

I dalje konzistentno pobjeđuju *minioni* fiksnog ponašanja.

Rotiranje

Kao što je i prije istaknuto *minioni* i dalje rade konstantno rotiranje. Međutim, u ovoj generaciji rotiraju se puno većom brzinom nego ranije. Kako im je brzina vrtnje visoka



Slika 5.2: Prosječna dobrota *miniona* s modelom stabala odluke kroz 900 generacija evolucije

teško rade translacijske pomake te teže dolaze do hrane za preživljavanje u odnosu na ranije generacije.

Agresivnost

Manji dio *miniona* nazad poprima agresivnost iz ranijih generacija te se zabijaju u konkurente koji se nalaze u blizini.

5.4. Stabla odluke

5.4.1. Konfiguracija

Maksimalna dubina stabala je postavljena na 8 čvorova u dubinu. Koristi se operator križanja koji je opisan u 4.5.2 te je minimalna dubina čvora koji će se odabrati kao korijen podstabla koje će se zamijeniti postavljena na 1 a maksimalna dubina na 7. Korišteni operatori mutacije opisani su u 4.5.3. Operator koji zamjenjuje cijelo podstablo se koristi s 25% vjerojatnosti a minimalna i maksimalna dubina čvora čije će se podstablo zamijeniti je 1 odnosno 7. Maksimalna dubina nasumično generiranog podstabla je 3. Drugi operator mutacije koji mutira individualne čvorove koristi se s 25% vjerojatnosti. Vjerojatnost da se mutira neki čvor je 0.1%.

5.4.2. Prosječna dobrota

Kretanje prosječne dobrote *miniona* kroz generacije može se vidjeti na slici 5.2. Kao i kod *miniona* s modelom neuronskih mreža na grafu se vidi kretanje prosječne dobrote u određenom rasponu. Za ovaj model taj raspon je uži te je pomaknut prema većim

dobrotama. Na temelju toga se zaključuje da je model stabala odluke konzistentniji i bolji u ponašanju od modela neuronskih mreža. Najviša dostignuta dobrota je 7864.57 u generaciji 254.

5.4.3. Uočena ponašanja nakon 300 generacija

Minioni sa stablima odluke kao modelom ponašanja u većini slučajeva uspješno nadživljavaju *minione* fiksnog ponašanja. Uspijevaju samostalno skenirati okolinu, pronaći hranu te loviti konkurentske *minione*.

Kretanje u smjeru drugih objekata

Većina *miniona* reagira na druge objekte tako da se krene sporijom brzinom kretati u smjeru detektiranih objekata. Čini se da najbolje reagiraju na tijela te im takvo ponašanje omogućuje dugotrajan život.

Zabijanje u konkurentske *minione*

Kada *minion* ispred sebe detektira konkurentskog živog *miniona* pokušat će se velikom brzinom zaletjeti u njega s ciljem da mu napravi štetu te da sebi stvori hranu.

Rotacija

Neki od *miniona* su osvojili ponašanje gdje se konstantno sporijom brzinom rotiraju oko svoje osi. Ovo im omogućava skeniranje cijele okoline u punom krugu oko svoje pozicije te bolje reakcije na opasnosti i prilike u okolini. Ovo ponašanje se ističe isto kao i kod *miniona* koji za model ponašanja koriste neuronske mreže. Međutim vrtnja kod *miniona* sa stablima odluke je sporija te time troše manje energije.

Blokada

U nekim slučajevima *minion* zablokira te se uopće ne kreće. Uočeno je da se ovo najčešće događa kada ispred sebe detektira granicu a ne detektira ništa drugo na što bi mogao reagirati.

5.4.4. Uočena ponašanja nakon 600 generacija

Minioni još češće nadživljavaju *minione* fiksnog ponašanja. Neka od ponašanja iz prijašnjeg poglavlja su skupljena u zanimljive stereotipe poput strvinara i lovca.

Rotacija

Kao i prije uočeno je konstantno rotiranje *miniona* oko svoje osi. Međutim, veći dio populacije sada se rotira većom brzinom.

Nezainteresiranost za bobice

Skoro svi *minioni* u populaciji nimalo ne reagiraju na hranjive i otrovne bobice. Ponašaju se jednako bez obzira detektiraju li bobice ili ne.

Strvinari

Jedan od uočenih stereotipa ponašanja su strvinari. *Minioni* ovog stereotipa konstantno se rotiraju te skeniraju okolinu. U potpunosti uvijek izbjegavaju žive konkurentske *minione* ali jako ih privlače tijela mrtvih *miniona*. Sve u svemu, ovakvo defenzivno ponašanje im omogućuje da su, u većini slučajeva, među zadnjim preživjelima.

Lovci

Ovaj stereotip ponašanja je po svemu sličan strvinarima. Jedina razlika je što lovci umjesto da bježe od živih konkurenata love ih te se zabijaju u njih. Ovakav agresivniji način ponašanja ih ponekad dovodi do smrti zbog nedovoljne brzine pri lovu gdje ne uspijevaju uloviti svoj plijen.

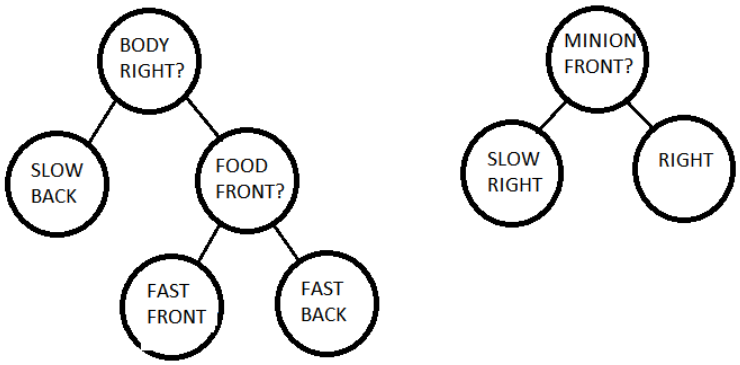
5.4.5. Uočena ponašanja nakon 900 generacija

Sva prijašnje navedena ponašanja su i dalje prisutna te *minioni* sa stablima odluke u većini slučajeva nadživljavaju *minione* fiksnog ponašanja. Razlika u odnosu na prijašnju provjeru na 600. generaciji je da su sada *minioni* precizniji i odlučniji u svojim odlukama.

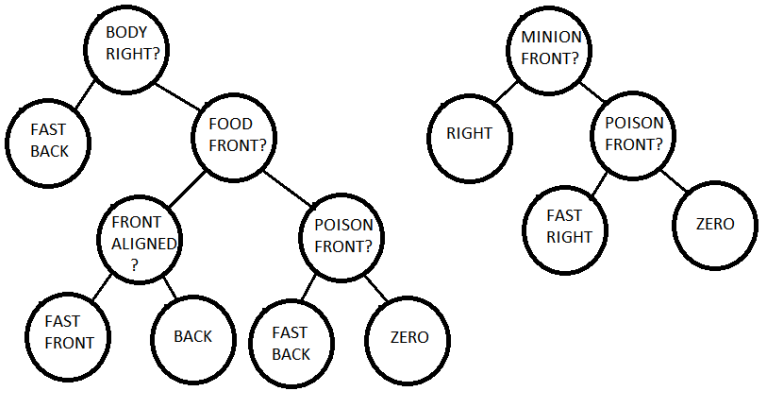
5.5. Rezultati evolucije plitkih stabala

5.5.1. Opis ispitivanja

Evolucija se provodi na isti način kao i u prijašnjem ispitivanju s istom konfiguracijom opisanom u 5.2. Postoje dvije razlike u konfiguraciji - maksimalna dubina stabala odluke i maksimalan broj generacija evolucije. Maksimalna dubina postavljena je na 4 a broj generacija kroz koje se provodi evolucija je 300.



Slika 5.3: Jedan od evoluiranih modela ponašanja - stablo koje upravlja akceleracijom (lijevo) i stablo koje upravlja rotacijom (desno)



Slika 5.4: Jedan od evoluiranih modela ponašanja - stablo koje upravlja akceleracijom (lijevo) i stablo koje upravlja rotacijom (desno)

Cilj ovog ispitivanja je prikazati evoluirana stabla odluke koja upravljaju ponašanje *miniona*.

5.5.2. Rezultati

Na slikama 5.3 i 5.4 prikazani su grafički prikazi dvoje evoluiranih modela ponašanja tijekom ispitivanja. Evaluacija stabla započinje s korijenskim čvorom na vrhu te se ispitivanjem činjenice u čvorovima grananja evaluira lijevo (u slučaju da je odgovor na upit istinit) ili desno (u slučaju da je odgovor neistinit) podstablo. Rezultat evaluacije stabla se nalazi u nekom od listova stabla koji vraća neki izraz za potrebnu akceleraciju ili rotaciju.

Model prvog *miniona* prikazan na slici 5.3 je naučio brzo kretati prema hrani kada ju osjeti ispred sebe. To ponašanje je vidljivo na stablu zaduženom za akceleraciju u lancu izvođenja *"BODY RIGHT? -> NE -> FOOD FRONT? -> DA -> FAST FRONT"*.

Osim tog ponašanja uočava se uobičajeno konstantno rotiranje u stablu zaduženom za rotaciju gdje oba lista stabla vraćaju vrtnju u smjeru kazaljke na satu.

Model drugog *miniona* prikazan je na slici 5.4. Ovaj model je kompleksniji od prvoga ali se ponaša na sličan način. Dodatno zanimljivo ponašanje u ovom modelu u odnosu na prijašnji je izbjegavanje otrovnih bobica. Ovo izbjegavanje otrova je vidljivo u lancu evaluacije stabla za akceleraciju *"BODY RIGHT? -> NE -> FOOD FRONT? -> NE -> POISON FRONT? -> DA -> FAST BACK"*. Osim tog lanca evaluacije vidljiv je i lanac evaluacije u stablu za rotaciju koji je, čini se, zadužen za isti zadatak izbjegavanja otrova *"MINION FRONT? -> NE -> POISON FRONT? -> DA -> FAST RIGHT"*. Kombinacijom tih dvoje lanaca kada *minion* ispred sebe uoči otrov on će naglo usporiti te skrenuti desno kako bi izbjegao taj otrov.

5.6. Statistika najboljih *miniona*

5.6.1. Opis ispitivanja

Cilj ovog ispitivanja je analizirati kakvo je statističko ponašanje najboljih dostignutih dobrota *miniona* u više pokretanja evolucije. Koristi se slična konfiguracija kao i u 5.2 sa sljedećim izmjenama:

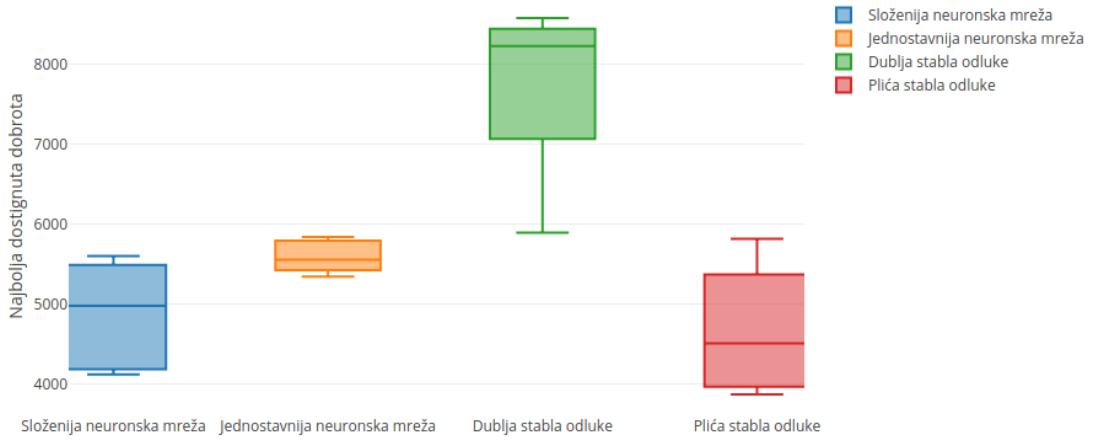
- broj *miniona* po generaciji smanjen je na 20
- postoji samo jedna particija po generaciji
- elitizam je smanjen na 4
- broj generacija kroz koje se provodi evolucija smanjen je na 100

Ispituju se sljedeće konfiguracije:

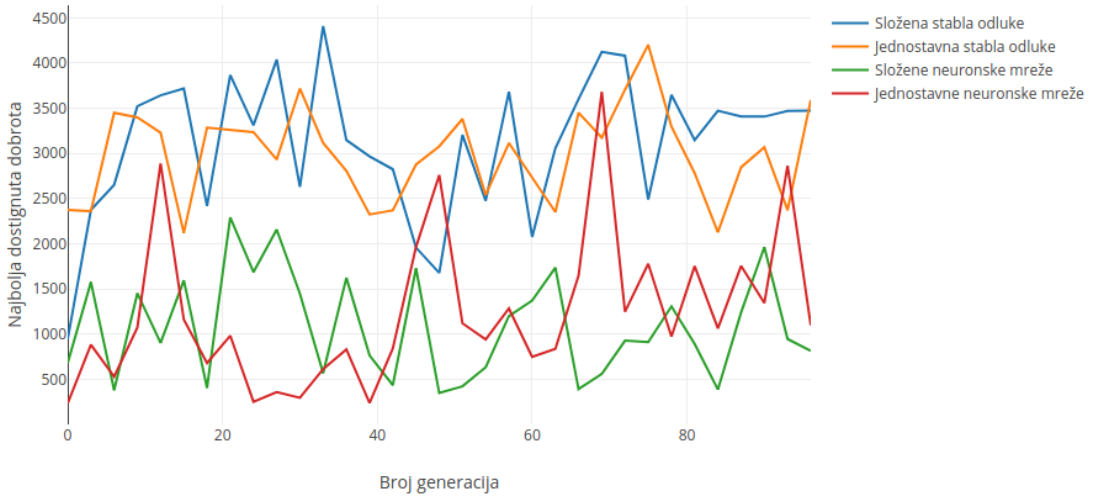
- model neuronskih mreža s arhitekturom koja ima jedan skriveni sloj veličine 50
- model neuronskih mreža s arhitekturom koja ima dva skrivena sloja veličine 80 odnosno 30
- model stabala odluke s maksimalnom dubinom stabla 3
- model stabala odluke s maksimalnom dubinom stabla 8

5.6.2. Rezultati

Na slici 5.5 prikazan je *box-plot* koji prikazuje odnose najboljih postignutih dobrota po prijašnje navedenim konfiguracijama.



Slika 5.5: *Box-plot* najbolje dostignutih dobrota po različitim konfiguracijama



Slika 5.6: Graf koji prikazuje dobrotu najboljeg *miniona* kroz 100 generacija evolucije za različite modele ponašanja

Najbolja dobrota po generacijama

Na slici 5.6 nalazi se graf koji prikazuje dobrotu najboljeg *miniona* u generaciji kroz generacije evolucije. Graf je vrlo kaotičan jer, kao što je i prije istaknuto, dobrota najbolje jedinke ovisi o nasumičnim faktorima kao što je inicijalni položaj svih objekata u simulaciji. Međutim, iz grafa se može vidjeti da oba modela stabala odluke daju konzistentno bolje rezultate od oba modela neuronskih mreža. Osim toga, zanimljivo je istaknuti da složeniji model neuronskih mreža daje bolje rezultate od jednostavnijeg sve do, otprilike, generacije 60 nakon koje jednostavniji model postaje bolji.

Najbolja dobivena dobrota tijekom cijele evolucije

Najbolje dobrote dobivaju se kompleksnijim stablima odluke dok se najgore dobrote dobivaju kompleksnim neuronskim mrežama.

Zanimljivo je usporediti kompleksniji i jednostavniji model neuronskih mreža. Kompleksniji model ima velik broj parametara pa genetski algoritam teško pronalazi bolja rješenja s nasumičnim podešavanjem parametara. Posljedica toga je nekonzistentnost u ponašanju modela. Nasuprot tome - jednostavniji model nudi relativno bolja rješenja te puno konzistentnije jer manji broj parametara olakšava optimizaciju ali je mreža još uvijek ima dovoljnu ekspresivnost da napravi dobre rezultate.

Kod modela stabala odluke čini se da maksimalna dubina stabla utječe na kvalitetu rješenja ali ne i na konzistentnost.

6. Zaključak

Implementacija svijeta s konzistentnim pravilima među objektima je uspješno provedena u sklopu rada. Većina pravila su implementirana po uzoru na neke od postojećih radova na sličnu temu. Pod pravilima svijeta podrazumijeva se simulacija realistične fizike u 2D okruženju, posljedice sudara između objekata različitih i istih vrsta te način na koji radi organizam *miniona*.

Za optimizaciju ponašanja i atributa *miniona* nameće se genetski algoritam kao očiti izbor s obzirom na to da se radi o simulaciji svijeta pa se optimizacija evolucijom čini prikladnom. Implementirana su dva modela ponašanja *miniona* čije se performanse uspoređuju ispitivanjima. Prvi model su neuronske mreže koje na ulaz dobivaju čiste podatke koje šalju senzori pa time imaju veću slobodu. Drugi model su stabla odluke koje na ulaz dobiju ograničen broj *true* ili *false* činjenica koje se dobiju pretprocesiranjem podataka iz senzora. Stabla odluke su tako ograničenija u broju mogućih ponašanja od neuronskih mreža. Osim ova dva modela implementirani su i *minioni* fiksnog ponašanja koji služe kao referentna točka u ispitivanjima i kao konkurencija koja tjera druga dva modela da optimiraju svoje ponašanje.

Iz rezultata ispitivanja zaključeno je da stabala odluke evoluiraju brže i konzistentnije od neuronskih mreža. Postoji nekoliko sličnosti u ponašanjima između ova dva modela kao što je konstanta rotacija kako bi se skeniralo cijelo okolno područje oko vlastite pozicije. Neuronske mreže često evoluiraju nekonzistentna ponašanja te ponašanja koja daju preveliki naglasak na neke odluke. Primjer toga je kako se rotiranje ubrzavalo što dulje je trajala evolucija što je pridonijelo bržem trošenju energije i ranijoj smrti *miniona*. Stabla odluke evoluiraju konzistentno te evoluiraju ponašanja koja su konzistentna i odlučna. Međutim, kod stabala odluke se ističe nedostatak mogućnosti kod optimizacije tako da se u specifičnim slučajevima događa blokada *miniona*. Pri nekim dobivenim podacima *minion* ne radi ništa te samo stoji na mjestu.

Sve u svemu, *minioni* fiksnog ponašanja, kao referentna točka, često nadživljavaju model neuronskih mreža dok vrlo rijetko nadžive model stabala odluka. Po tome se može zaključiti da su za ovu vrstu problema stabala odluke bolji izbor. Međutim,

zbog ograničenja stabala odluka navedenih ranije postoji mogućnost da, uz dovoljno generacija evoluiranja, neuronske mreže postanu bolje jer raspolažu s većom količinom informacija.

LITERATURA

Paul Matthew Boxwell. Natural selection neural network simulation. http://exploreai.blogspot.hr/, 2016.

Paul Dawkins. Euler's method. http://tutorial.math.lamar.edu/Classes/DE/EulersMethod.aspx, 2018.

Thomas Geijtenbeek, Michiel van de Panne, i A. Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32 (6), 2013.

Paul T. Oliver. Evolving guppies. https://www.youtube.com/watch?v=tCPzYM7B338, 2012.

Priyanka Yadlapalli Padmavathi Kora. Crossover operators in genetic algorithms: A review. https://pdfs.semanticscholar.org/708d/79c888955fe92b84df74d8bc058134503307.pdf, 2017.

Thomas Smid. Elastic and inelastic collision in two dimensions. https://www.plasmaphysics.org.uk/collision2d.htm, 2018.

Peter Taylor i Richard Lewontin. The genotype/phenotype distinction. U Edward N. Zalta, urednik, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2017 izdanju, 2017.

Xin-She Yang. Metaheuristic optimization. http://www.scholarpedia.org/article/Metaheuristic_Optimization, 2011.

Dodatak A
Upute za instalaciju

Simulacija se instalira prevođenjem izvornog koda simulacije.

A.1. Dodatne biblioteke

Kako bi se izvorni kod uspješno preveo potrebno je preuzeti i instalirati SDL2 (eng. *Simple DirectMedia Layer*) biblioteku i GLEW (eng. *The OpenGL Extension Wrangler Library*).

Za instalaciju SDL2 biblioteke potrebno je pristupiti stranici `https://wiki.libsdl.org/Installation`https://wiki.libsdl.org/Installation te pratiti upute za instalaciju ovisno o platformi.

Za instalaciju GLEW biblioteke potrebno je pristupiti stranici `http://glew.sourceforge.net/install.html`http://glew.sourceforge.net/install.html te pratiti upute za instalaciju.

Sve ostale dodatne biblioteke se nalaze uz sam izvorni kod simulacije te ih nije potrebno dodatno instalirati.

A.2. Prevođenje i pokretanje simulacije

Projekt koristi *cmake* alat kao *build* alat te ga je potrebno instalirati. Za instalaciju *cmake-a* potrebno je pristupiti stranici `https://cmake.org/install/`https://cmake.org/install/ te pratiti upute za instalaciju.

Daljnje korištenje *cmake-a* u prevođenju izvornog koda projekta ovisi o platformi za koju se prevodi. Na Unix/Linux sustavima prevođenje se svodi na izvršavanje sljedećih naredbi u terminalu:

```
cmake .
make
```

Nakon uspješnog prevođenja projekt se može pokrenuti pomoću naredbe:

```
./Biome_Simulation
```

Dodatak B
Konfiguracijska datoteka

Konfiguracijska datoteka se piše u JSON formatu te se mora nalaziti u direktoriju iz kojeg se pokreće simulacija i mora imati ime "config.txt". Svaki ključ u ključ-vrijednosti parovima u JSON datoteci je tipa *string* te predstavlja ime neke od konfigurabilnih vrijednosti simulacije.

Korijenski objekt JSON datoteke prepoznaje sljedeće vrijednosti kao ključeve za konfiguraciju:

- "boundary_radius" - realni broj koji određuje radijus granice simulacije
- "number_generation_partitions" - cijeli broj koji određuje broj particija populacije jedinki koje se neovisno evaluiraju
- "number_minions_per_partition" - cijeli broj koji određuje broj *miniona* koji se evoluiraju po particiji generacije
- "number_default_minions" - cijeli broj koji određuje broj *miniona* fiksnog ponašanja koji se ubacuje na početku evaluacije particije generacije
- "food_pellets" - cijeli broj koji određuje broj hranjivih bobica
- "poison_pellets" - cijeli broj koji određuje broj otrovnih bobica
- "elitism" - cijeli broj koji određuje broj najboljih *miniona* koji se automatski kopiraju u sljedeću generaciju
- "number_max_generations" - cijeli broj koji određuje koliko generacija genetskog algoritma se izvodi prije nego simulacija završi
- "load_directory" - put do direktorija iz kojeg se učitava prva generacija *miniona*
- "save_directory" - put do direktorija u koji će se spremati generacije *miniona*
- "print_info_every_second" - cijeli broj koji predstavlja u kojim intervalima se ispisuju relevantni podaci za simulaciju

- "persist_minions_every_seconds" - cijeli broj koji predstavlja u kojim intervalima se *minioni* spremaju na disk
- "minion_generator" - string koji određuje koji model ponašanja se koristi ("decision_tree" ili "neural_net")
- "minion_generator_config" - objekt koji određuje konfiguraciju modela ponašanja
- "fitness" - string koji određuje koji model dobrote se koristi ("activity" ili "time_alive")
- "fitness_config" - objekt koji određuje konfiguraciju modela dobrote
- "crossover_operator" - string koji određuje koji operator križanja se koristi ("pick_better", "pick_random" ili "custom")
- "crossover_operator_config" - objekt koji određuje konfiguraciju operatora križanja
- "mutation_operator" - string koji određuje koji operator mutacije se koristi ("do_nothing" ili "custom")
- "mutation_operator_config" - objekt koji određuje konfiguraciju operatora mutacije

Objekt "minion_generator_config":

- "max_tree_depth" - cijeli broj koji predstavlja maksimalnu dubinu stabla odluke (koristi se samo ako je "minion_generator" bio postavljen na "decision_tree")
- "architecture_hidden" - niz cijelih brojeva koji predstavlja arhitekturu skrivenih slojeva u neuronskoj mreži (koriste se samo ako je "minion_generator" bio postavljen na "neural_net")

Objekt "fitness_config" se čita samo ako je "fitness" bio postavljen na "activity":

- "time_lived_coeff" - koeficijent koji množi vrijeme života *miniona* te se taj umnožak pridodaje ukupnoj dobroti *miniona*
- "damage_dealt_coeff" - koeficijent koji množi koliko je štete *minion* načinio konkurentima te se taj umnožak pridodaje ukupnoj dobroti *miniona*
- "health_recovered_coeff" - koeficijent koji množi koliko je zdravlja *minion* obnovio te se taj umnožak pridodaje ukupnoj dobroti *miniona*
- "distance_traveled_coeff" - koeficijent koji množi koliki je put *minion* prošao te se taj umnožak pridodaje ukupnoj dobroti *miniona*

Objekt "crossover_operator_config" se učitava samo ako je "crossover_operator" postavljen na "custom". Objekt se sastoji od 3 ključa "object_crossovers", "senses_crossovers"

i "controller_crossovers". Svaki od tih ključa ima niz objekata za pripadnu vrijednost. Svaki objekt u nizu ima 3 ključa "name" (ime operatora), "probability" (vjerojatnost da se taj operator koristi u nekom trenutku) i "config" (dodatni objekt koji konfigurira taj operator). Za "object_crossovers" i "senses_crossovers" postoje 2 mogućnosti "fitness_weighted_average" i "arithmetic_average". Za "controller_crossovers" postoje 3 mogućnosti "decision_tree_subst_subtree", "neural_net_layer_weights" i "neural_net_single_layer". Od tih 3 operatora samo "decision_tree_subst_subtree" prima konfiguraciju:

- "min_depth_cross_point" - minimalna dubina čvora čije podstablo će biti zamijenjeno
- "max_depth_cross_point" - maksimalna dubina čvora čije podstablo će biti zamijenjeno

Kao i za operatore križanja operatori mutacije (u slučaju da se radi o "custom" mutaciji) su podijeljeni na "object_mutations", "senses_mutations" i "controller_mutations". "object_mutations" i "senses_mutations" podržavaju samo "gauss_noise" mutaciju koja u konfiguraciji prima samo "variance" parametar koji određuje varijancu normalne distribucije. "controller_mutations" prepoznaje sljedeće operatore ovisno o korištenom modelu ponašanja "decision_tree_rand_subtree", "decision_tree_mutate_nodes", "gauss_noise_weights" i "gauss_sparse_reset".

Objekt "decision_tree_rand_subtree":

- "min_depth" - minimalna dubina čvora čije podstablo će biti zamijenjeno
- "max_depth" - maksimalna dubina čvora čije podstablo će biti zamijenjeno
- "subtree_max_depth" - maksimalna dubina nasumično generiranog podstabla

Objekt "decision_tree_mutate_nodes":

- "min_depth" - minimalna dubina čvorova koji se mogu mutirati
- "max_depth" - maksimalna dubina čvorova koji se mogu mutirati
- "probability_mutate_node" - vjerojatnost mutiranja pojedinog čvora

Objekt "gauss_noise_weights":

- "variance" - varijanca normalne distribucije iz koje se uzorkuje šum

Objekt "gauss_sparse_reset":

- "weight_reset_probability" - vjerojatnost resetiranja pojedine težine
- "bias_reset_probability" - vjerojatnost resetiranja pojedine pristranosti
- "variance" - varijanca normalne distribucije iz koje se uzorkuje šum

Simulacija bioma postupcima strojnog učenja

Sažetak

Rad uspoređuje performanse neuronskih mreža i stabala odluke kao modela ponašanja za jedinke koje se nalaze u simuliranom 2D svijetu. Cijeli svijet i njegova pravila interakcije među objektima unutar njega implementirana su i opisana u sklopu ovog rada. Modeli ponašanja optimiziraju se korištenjem generacijskog genetskog algoritma kojim se nastoji čim više približiti stvarnoj evoluciji iz prirode.

Ključne riječi: neuronske mreže, stabla odluke, genetski algoritam, strojno učenje, simulacija, biom, računalna grafika, evolucija, simulacija fizike

Biome simulation with machine learning

Abstract

Thesis compares the performances of neural networks and decision trees as behavior models for individuals which exist in a simulated 2D world. Whole world and its rules of interaction between the objects within it are implemented and described as a part of this thesis. Behavior models are optimized by using generational genetic algorithm which tries to mimic the real evolution found in nature.

Keywords: neural networks, decision trees, genetic algorithm, machine learning, simulation, biome, computer graphics, evolution, physics simulation