SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1118

KOOPERATIVNO, NATJECATELJSKO I MJEŠOVITO KOOPERATIVNO-NATJECATELJSKO PODRŽANO UČENJE ZA IGRU ZMIJA S VIŠE AGENATA

Eva Kaštelan

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1118

KOOPERATIVNO, NATJECATELJSKO I MJEŠOVITO KOOPERATIVNO-NATJECATELJSKO PODRŽANO UČENJE ZA IGRU ZMIJA S VIŠE AGENATA

Eva Kaštelan

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 1118

Pristupnica:	**Eva Kaštelan (0036529863)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Domagoj Jakobović

Zadatak: **Kooperativno, natjecateljsko i mješovito kooperativno-natjecateljsko podržano učenje za igru Zmija s više agenata**

Opis zadatka:

Opisati problem pronalaženja i optimizacije strategija igranja u igrama s više igrača (agenta). Na nekoliko inačica igre oblikovati problem pronalaska strategije igranja kao problem optimizacije ili strojnog učenja. Posebnu pažnju posvetiti mogućnosti primjene koevolucije u evolucijskim algoritmima za optimizaciju strategije igranja. Ostvariti model učenja strategije i programski sustav za učenje temeljen na evolucijskom algoritmu. Eksperimentalno utvrditi učinkovitost različitih inačica algoritma optimizacije strategije ovisno o parametrima okoline učenja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 9. lipnja 2023.

SADRŽAJ

1. Uvod **1**

2. Evolucijski algoritam **2**

2.1. Uvod 2

2.2. Rad evolucijskog algoritma 2

2.3. Genetsko programiranje 4

2.3.1. Genetsko programiranje temeljeno na stablima 5

3. Igra Zmija **6**

3.1. Opis motivacije 6

3.2. Opis okoline i agenta 6

4. Kooperativna inačica koevolucije **8**

4.1. Opis željenog ponašanja 8

4.2. Razvoj i metode 8

4.3. Joint-action learning (JAL) 9

5. Natjecateljska inačica koevolucije **13**

5.1. Opis željenog ponašanja 13

5.2. Razvoj i metode 13

5.3. Self-play 13

6. Mješovito kooperativno-natjecateljska inačica koevolucije **15**

6.1. Opis željenog ponašanja 15

6.2. Razvoj i metode 15

7. Implementacija **17**

7.1. Implementacijski okvir 17

7.1.1. Okvir za natjecateljsku koevoluciju 17

7.1.2. Okvir za koevoluciju agenta u timovima 18
7.2. Model genetskog programiranja 19

8. Eksperimenti **21**
8.1. Kooperativna inačica 21
8.1.1. JAL s 1 jabukom 22
8.1.2. JAL s 2 jabuke 24
8.1.3. Usporedba 26
8.2. Natjecateljska inačica 27
8.2.1. Natjecateljska okolina bez self-play tehnike 27
8.2.2. Self-play tehnika 29
8.2.3. Usporedba 31
8.3. Mješovito kooperativno-natjecateljska inačica 31
8.3.1. Mješovita inačica 32
8.4. Usporedba inačica 35

9. Zaključak **37**

Literatura **38**

1. Uvod

Umjetna inteligencija (UI) je područje studija i istraživanja koje ima za cilj razviti inteligentne sustave sposobne obavljati zadatke koji obično zahtijevaju ljudsku inteligenciju. Tijekom posljednjih nekoliko desetljeća, UI je doživjela izvanredan napredak, transformirajući različite aspekte naših života. Jedno fascinantno područje unutar UI-a su evolucijski algoritmi, koji, za rješavanje složenih problema, crpe inspiraciju iz biološke evolucije. Konkretno, genetsko programiranje, podskup evolucijskih algoritama, usredotočuje se na stvaranje računalnih programa kroz proces prirodne selekcije i genetske rekombinacije. Evolucijski algoritmi se temelje na konceptu preživljavanja najprilagodljivijih. Ovi algoritmi simuliraju dobro poznat prirodan evolucijski proces tako da iterativno generiraju nove generacije iz početne populacije potencijalnih rješenja i evaluiraju njihovu prilagodljivost, dobrotu i iterativno poboljšavaju jedinke generacija. Opetovanom primjenom genetskih operatora poput križanja i mutacije, evolucijski algoritmi istražuju prostor rješenja i konvergiraju prema optimalnim ili gotovo optimalnim rješenjima. Genetsko programiranje, kao specifičan oblik evolucijskih algoritama, proširuje ovaj pristup na evoluciju računalnih programa. Umjesto optimiziranja fiksnog skupa parametara, genetsko programiranje evoluira strukturu i ponašanje samih programa.

U ovom radu opisana je podloga i osnova evolucijskih algoritama te neke njegove podvrste tj. tehnike kojim se mogu modelirati agenti podržanog učenja. Uvodi se opis i pojam igre Zmija te su definirane 3 različite inačice igre. Svaka inačica igre pretpostavlja drukčije željeno ponašanje agenta u višeagentskim sustavima te tako svaka verzija predstavlja zaseban problem pronalaženja i optimizacije strategije. Za svaku inačicu opisan i implementiran je zaseban programski sustav za učenje koji se svi temelje na evolucijskom algoritmu. Sustavi su potom ispitani te su komentirani dobiveni rezultati.

2. Evolucijski algoritam

2.1. Uvod

Evolucijski algoritam (EA) je algoritam čija temeljna ideja pronalazi svoj korijen i osnovu u biološkoj evoluciji. Spada u podskup evolucijskog računarstva i bio-informatike te se najčešće primjenjuje pri rješavanju netraktabilnih problema koji nisu jednostavno rješivi u polinomijalnom vremenu. Evolucijski algoritmi su inspirirani Darwinovom evolucijom što se očituje u uporabi mehanizama i funkcija algoritama koje oponašaju razvitak vrsta. Neke od tehnika poput: selekcije, križanja, mutacije ... Navedeni mehanizmi se primjenjuju na populaciju čija svaka pojedinačna jedinka predstavlja jedno moguće rješenje problema optimizacije. Evolucijski algoritmi izvrsni su u optimiziranju rješenja. No, važno je napomenuti da iako učinkovito optimiziraju, ne pronalaze nužno optimalno rješenje. Umjesto toga, neprestano pronalaze trenutna radna rješenja i međusobno uspoređuju njihovu uspješnost, što može ili ne mora pronaći apsolutno najbolje moguće rješenje. Mana evolucijskih algoritama je njihov zahtijev za relativno visoke računalne resurse. No, najveća prednost evolucijskih algoritama je činjenica da spadaju u skup metaheuristika tj. da je algoritamski koncept koji se koristi za definiciju heurističkih metoda primjenjivi na širokom skupu problema - u sažetku nisu problemski specifični.

2.2. Rad evolucijskog algoritma

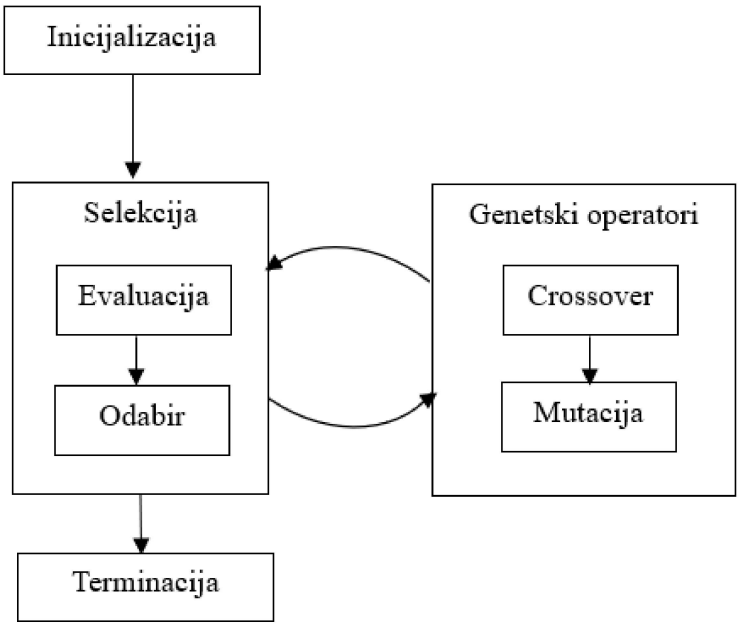
Temeljni koncept evolucijskog algoritma je, kroz više generacija, evoluirati neku stvorenu populaciju jedinki tj. potencijalnih rješenja problema. Algoritam iterativno nad populacijom primjenjuje različite genetske operatore kako bi u idućoj generaciji stvorio nove primjere rješenja odnosno nove jedinke. EA, u osnovi, sadrži 4 temeljna koraka: inicijalizacija, selekcija, genetski operatori te terminacija.

1. Inicijalizacija
   - u početku algoritma stvara se inicijalna populacija potencijalnih rješenja koja ujedino predstavlja i genetski bazen naših jedinki. Uobičajeno je da su u početku jedinke nasumično generirane sa slučajnim „kromosomom“ tj. genetskim materijalom

2. Selekcija
   - drugi korak selekcija simulira proces prirodne selekcije te se intuitivno dijeli na dva dijela: evaluacija i odabir jedinki. U evaluaciji svakoj jedinci populacije pridajemo neku vrijednost njezine dobrote (eng. fitness). Funkcija dobrote našeg algoritma na temelju nekih karakteristika i postignuća pojedine jedinke, njoj pridaje neku numeričku vrijednost. Točan odabir funkcije dobrote se smatra jednom od glavnih problematika evolucijskog algoritma jer je iznimno specifična za svaki pojedini problem i od velike važnosti radu algoritma. Nakon evaluacije, zasebna su rješenja potom, u koraku odabira jedinki, na temelju vrijednosti dobrote izabrana za daljnju evoluciju ili reprodukciju.

3. Genetski operatori
   - ovaj korak oponaša proces reprodukcije vrste u prirodi te se stoga dijeli u dva potkoraka: Križanje (Rekombinacija) i Mutacija.
     - Križanje – nakon odabira najboljih članova generacije (uobičajeno u praksi 2 člana) oni se sada koriste za stvaranje sljedeće generacije u algoritmu. Miješajući karakteristike tj. genetski materijal odabranih roditelja, stvaraju se novi potomci koji predstavljaju jedinke iduće generacije. Ovaj proces kombinira genetski materijal roditelja, time omogućujući istraživanje različitih kombinacija genetskih informacija tijekom evolucijskog procesa.
     - Mutacija – je također jedan od ključnih dijelova algoritma jer je ona ta koja uvodi kompletno novi genetski materijal u genetski bazen populacije. Uvođenjem novih gena, koji nisu savršena kopija svojih roditelja, sprječava se zapinjanje algoritma u lokalnom ekstremu problema. Uobičajeno se primjenjuje nad kromosomom sa malom vjerojatnošću.

4. Terminacija
   - kao posljednji korak dolazi terminacija koja zaustavlja rad algoritma i daljnju evoluciju populacije. Dva su uobičajena načina kojim se postiže kraj: algoritam dostiže neki zadani kraj u vremenu ili broju generacije, ili je trenutna najbolja jedinka zadovoljila neki prag performansa tj. postigla zadovoljavajuću vrijednost funkcije dobrote



Slika 2.1: Dijagram evolucijskog algoritma

Sažimajući cijeli postupak, ove vrste primjene evolucijskog algoritma, problem se može definirati kao pronalazak najbolje kombinacije elementa genetskog materijala ili kromosoma jedinki koja maksimizira neku zadanu funkciju dobrote, a algoritam se provodi sve dok nismo dosegli maksimalni broj iteracija ili pronašli prihvatljivo rješenje tj. neki prag funkcije dobrote.

2.3. Genetsko programiranje

Genetsko programiranje (GP) definira se kao podvrsta ili jedna od tehnika u evolucijskim algoritmima. GP također kao cilj ima evoluiranje populacije na početku nepodobnih jedinki u zadovoljavajuće jedinke, a njegova evolucija, kao i kod evolucijskih

algoritama, prati i oponaša prirodnu evoluciju te koristi analogne metode i funkcije pri njezinom provođenju. Glavna razlika, odnosno ono što izdvaja genetsko programiranje od njegovog nadpojma, je specifični fokus na evoluciju evoluirajućih računalnih programa. Genetsko programiranje općenito se koristi za simboličku regresiju, automatsko programiranje i slične zadatke gdje je nužno razvijati programe.

2.3.1. Genetsko programiranje temeljeno na stablima

Najstarija kao i prva razvijena metoda i uporaba genetskog programiranja je ona temeljena na stablima u memoriji. Tako je u populaciji jedinki svaki genetski materijal pojedine jedinke sadržan kao zasebno stablo. Uobičajeno je da svaki čvor stabla ima neku operatorsku funkciju (funkcija poput množenja, zbrajanja ...), a svaki list stabla sadrži operand tih funkcija. Kako stablo predstavlja jedinku tako se nad njim provode genetski operatori i tehnike predstavljene u prošlom potpoglavlju.

3. Igra Zmija

3.1. Opis motivacije

Igra Zmija, inspiracija okoline agenata, vrsta je videoigre u kojoj igrač manevrira glavom trake, obično prikazane i tematizirane kao zmija, te se igraču kao zadatak postavlja sprječavanje zmije da se sudari s preprekama – zidom, samom sobom ili drugom zmijom istog okruženja. Taj cilj otežava se tijekom igre kako zmija svakom ulovljenom jabukom „raste“ tj. izdužuje svoje tijelo.

3.2. Opis okoline i agenta

Okolina je predstavljena kao mapa pozicija dimenzija 20 x 20 koju „nastanjuje“ više zmija (od 1 do 4). Agent je jedinka zmije čiji „mozak“, tj. koja, uči na temelju izgeneriranog GP-stabla. Takvo stablo predstavlja model tj. strategiju koja se kroz evoluciju poboljšava i mijenja. Svaki agent dodatno sadrži podatke o položaju svih ćelija u svom tijelu, položaju jabuke i trenutni rezultat koji predstavlja dužinu zmije. Agent zmija kreće se kroz okolinu i kao ulazne podatke prima informaciju o udaljenosti i tipu objekta u svakom od sedam smjerova, od svoje glave duž mape, počevši od onog zarotiranog za 45° u smjeru kazaljke na satu od zmijinog vrata te svaki idući zarotiran za 45° od prethodnog. Ako je tip objekta u određenom smjeru jabuka, zmija kao ulazni parametar tog smjera dobiva uređen par (1, d) gdje d označava udaljenost objekta od zmijine glave. U svim ostalim slučajevima, kada u smjeru nije jabuka, za udaljenosti objekata veće ili jednake od 2 zmija dobiva uređen par (0, 0) dok za udaljenosti objekata d manje od 2 prima par (-1, d) . Takvi uređeni parovi spremaju se potom u vektor veličine 14 (uređeni parovi za svih 7 smjerova) koje onda model, implementiran GP stablom, koristi za račun izlaza tj. za donošenje odluke o sljedećoj akciji zmije. Odluka modela zmije može biti pomaknuti tijelo zmije u jedan od 3 smjera: lijevo, desno ili ravno. Kada zmija pojede jabuku, povećava se dobitak, kao i tijelo zmije za 1.

Stablo modela u svakom svom listu i čvoru sadrži cijeli broj. Terminali sadrže vrijednosti spomenutog ulaznog vektora koji predstavljaju agentov pogled u okolinu. U slučajevima stabla koji imaju veći broj listova od veličine 14 ulaznog vektora, terminali sadrže nasumično odabranu cjelobrojnu konstantu 3. Čvorovi stabla tj. njegove funkcije mogu sadržati cjelobrojnu vrijednost iz intervala [0,5]. Svaki broj predstavlja jednu funkciju čiji su operandi vrijednosti djece čvora (Tablica 3.1). Brojevi funkcije nasumično su odabrani iz uniformne razdiobe. Izlaz iz modela izračunati je realan broj koji se preslikava u jednu od 3 akcija.

Tablica 3.1: Funkcije implementiranog GP stabla

Broj funkcije	Izraz funkcije
0	lijevo dijete + desno dijete + 1
1	lijevo dijete - desno dijete
2	lijevo dijete * (desno dijete + 1)
3	(lijevo dijete + 1) / (desno dijete + 1)
4	min(lijevo dijete, desno dijete)
5	max(lijevo dijete, desno dijete)

Agenti tj. zmije uče se različitim inačicama suživota pa se tako u istoj okolini nalazi više agenata. U generaciji prati se više okolina svaka predstavljena klasom Simulator. U samom simulatoru se nalaze ili dvije zmije ili dva tima (svaki tim od po 2 zmije) koji se onda uče suživotu.

4. Kooperativna inačica koevolucije

4.1. Opis željenog ponašanja

Prije početka razvoja strože je definirano željeno kooperativno ponašanje modela učenja strategije kojeg zmije predstavljaju. Suradnja se definirala kao suživot dviju zmiju u istoj okolini s ciljem da obje zmije maksimiziraju svoj životni vijek (odnosno broj koraka) i istovremeno obje skupljaju podjednaki broj jabuka čiji se zbroj također maksimizira.

4.2. Razvoj i metode

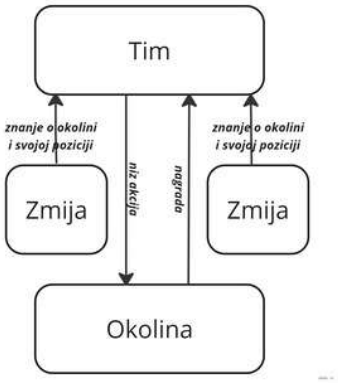
Razvoj stvaranja programskog sustava za kooperativno učenje može se podijeliti u 3 faze razvoja svaka temeljena na drugom opisu problema pronalaženja i optimizacije modela u okolini s više agenata. U prvoj fazi pokušaj je bio pronaći ili preinačiti funkciju dobrote zmije koja se kao pojedinačni agent kreće i živi u okolini s više pojedinačnih agenta. Prva implementirana ideja je bila u funkciju dobrote svake zasebne zmije uračunati i ukupni dobitak okoline i ukupni broj mrtvih i živih zmija okoline u kojoj se kreće pojedinačni agent. Tako bi primjeri nekih isprobanih novih funkcija dobrote bili: ukupni broj skupljenih jabuka* (br. živih/ sve zmije) po okolini, ukupni broj skupljenih jabuka * (br.mrtvih / sve zmije) po okolini i slične. No, ti pokušaji su navodili na kompetitivnije ponašanje ili sputavali uopće bilo kakvo učenje pojedinačnih zmija zamuteći im usmjerenost funkcije dobrote prema nekom ponašanju. Daljnja ideja za izmjenu funkcije bilo je uvođenje većih kazni za zabijanje u prepreke (zid ili drugu zmiju). Dva istaknuta primjera javljaju se onda u obliku: prvi oblik u kojem zbrajamo dobitak svih živih zmiji, ali ih i dodatno kažnjavamo ako su umrle oduzimajući im pola ukupno zbrojenog dobitka okoline, te drugi oblik u kojem razlikujemo zasebne zmije na mapi i uvedemo brojač za svaku zmiju kada se nađe blizu druge zmije odnosno kažnjavamo ih proporcionalno tome koliko puta joj je glava bila korak

od druge zmije te se dodatno još kažnjava zabijanje u drugu zmiju. Prvi od navedenih pokušaja iznimno je ubrzao učenje zmija koje su već u puno ranijim generacijama pokazivale inteligentnije ponašanje skupljajući više jabuka i bolje izbjegavajući zidove okoline, ali pokušaj nije pomogao u razvitku suradnje kod zmija. Drugi pokušaj je bio korak bliži kooperaciji gdje su zmije postale iznimno oprezne oko svojih sustanara zajedničkog okruženja te su prestale loviti jabuke i uglavnom naučile vrtjeti se podalje jedna od druge. Posljednji pokušaj u prvoj fazi razvoja potaknut preduvjetima za učenje s više agenata u radu[2] bio je dodavanje više jabuka u okruženje kao više ciljeva za agente koji bi smanjili natjecateljsku narav agenata. Inicijalno dodavanje više jabuka u okolinu od prve generacije nadalje iznimno je „zbunjivalo“ agente u početku tj. iznimno usporilo njihovo učenje. Pretpostavljam da se izgubila jasna usmjerenost zbog pojma jednog cilja za nasumično izgeneriranog agenta u početku evolucije koju je imao agent u okolini s jednom jabukom. Prirodna evolucija te ideje bila je uvođenje više jabuka nakon nekoliko generacija. Inkrementiran je broj generacija nakon kojih se uvodi više jabuka u koracima od 5 te je eksperiment u kojem je nakon 15. generacije uvedeno više jabuka pokazao interesantno, iako dalje nekooperativno, ponašanje. Zmije razvijaju iznimno zanimljivu strategiju skupljanja gdje veći klaster tj. nakupinu više jabuka ignoriraju te pribjegavaju skupljanju jabuka na rubu mape i rubu tih nakupina, tako sprječavaju opkoliti se vlastitim tijelom i u početku zabiti se u druge zmije. Druga faza razvoja dijeli problem dodavanja više agenta u jednu okolinu i njihovog učenja na suradnji u 2 zahtjeva. Ideja je inspirirana radom [3]. 2 zahtjeva na koja se dijele: 1. Učenje single-agent goal attainment i 2. Multi-agent cooperation. Izvodi se tako da se prvih par generacija svakom agentu ostavi stara funkcija dobrote temeljena na skupljanju sto većeg broja jabuka zasebno, a nakon određenog broja (implementirani pokušaj 5, 10 i 20 generacija) koristi jedna od navedenih funkcija dobrote u prvom pokušaju prve faze razvoja. Zmije ni u drugoj fazi razvoja nisu pokazivale suradnju. Treća i konačna faza razvoja uvodi simulaciju tehnike Joint-action learning (JAL) tj. tehniku učenja združenih akcija.

4.3. Joint-action learning (JAL)

Joint-action learning uključuje suradnju više agenata ili entiteta kako bi se postigao zajednički cilj. Agenti uče kako koordinirati akcije s drugima kako bi postigli zajedničke ciljeve. U fokusu su algoritmi i tehnike koje omogućuju agentima učenje i prilagodbu putem interakcije i suradnje na primjer uključujući učenje politika, protokole komunikacije i modeliranje ponašanja drugih agenata. Jedna od najjednostavnijih modifikacija

JAL-a je inačica u kojoj se svi agenti modeliraju kao jedan agent koji na temelju dijeljenog unosa svih agenta jedne okoline generira skupno niz akcija - jedna za svakog od agenata. U našem okruženju to se odražava tako da se uvodi novi pojam tima. Tim je taj „novi agent“ koji prati dobitke (skupljene jabuke), okolinu kao i životni vijek zmija u toj dijeljenoj okolini. Ključna izmjena dosadašnjeg algoritma je funkcija donošenja odluke o sljedećem koraku zmije. Sada tu odluku ne donosi zmija posebice sama za sebe već svaka prosljeđuje svoje znanje o svojoj okolini i poziciji timu, a tim onda za sve zmije u dijeljenoj okolini radi odluku o sljedećem koraku. Glavna mana i nedostatak ove tehnike združenog učenja je eksponencijalni rast broja odluka u proporciji s brojem agenata. Za razliku od prije gdje je svaki agent samostalno birao između 3 akcije, tim se za svaku pojedinu zmiju mora odlučiti između 3 akcije. U implementaciji taj problem je riješen tako da je broj zmija limitiran na 2 po timu, a tim ima 9 različitih izbora (3 izbora za 2 zmije).



Slika 4.1: Pojednostavljeni dijagram suradnje komponenti implementirajući joint-action learning tehniku učenja

Sada se opet javlja problem pronalaska funkcije dobrote, no, kako se zapravo „trenira tim“ tj. njegov model za donošenja združenih odluka, puno je jednostavniji i utjecajniji na kooperativno ponašanje zmija. Funkcija dobrote se ponovno pronalazila eksperimentalno. Dobivena funkcija dobrote prikazana je u kodu na slici 3.2 te ispisano poprima izgled formule:

$$dobrota = 500*(1-\frac{\Delta dobitak}{200})+300*(1-\frac{\Delta koraci}{200})+\sum_{zmija}^{tim} koraci+30*\sum_{zmija}^{tim} dobitak \tag{4.1}$$

Varijable Δdobitak i Δkoraci predstavljaju razliku između dobitka (broja skupljenih jabuka) dviju zmija u timu i isto tako razliku između njihovih brojeva koraka. Također

se u obzir uzimaju i sume koraka i dobitka svih zmija u pojedinom timu. Formula funkcije u suštini je suma 4, ponašanju bitna, rezultata odabranih akcija kretanja zmija u timu. Svaki taj rezultat ima svoj koeficijent (500, 300, 1 i 30) koji skaliraju važnost njegovog ostvarenja. Prvi i „najcjenjeniji“ rezultat tj. prvi pribrojnik funkcije s koeficijentom 500 nagrađuje malu razliku u broju koraka zmija u timu. Time se potiče opisani i željeni kooperativni karakter ponašanja zmija. Isto potiče i idući pribrojnik (koeficijenta 300) koji nagrađuje malu razliku u koracima između zmija tima. Iduća dva pribrojnika pomažu pri profiliranju najboljih pojedinaca u skupini surađujućih zmija tako da se još u dosadašnju vrijednost funkcije pribraja suma koraka i ukupna suma skupljenih jabuka tima. Suma jabuka ima postavljen koeficijent 30 kako bi više utjecala na iznos funkcije vrijednosti.

```
int Team::getTeamScore()
{
    //return this->teamScore;
    std::vector<int> scores = allSnakeScores();
    std::vector<int> steps = allSnakeSteps();
    float deltaScores = (float)(std::abs(scores[0] - scores[1]));
    float deltaSteps = (float)(std::abs(steps[0] - steps[1]));
    if (deltaScores == 0) deltaScores = 1;
    if (deltaSteps == 0) deltaSteps = 1;
    int reward = 0;
    if (steps[0] != 101 && steps[1] != 101) {
        if (scores[0] != 4 && scores[1] != 4) {
            reward += 500 * (1 - deltaScores / 200);
        }
        reward += 300*(1- deltaSteps/200) + (steps[0] + steps[1]);
        reward += 30*(scores[0] + scores[1]);
    }

    return reward;

}
```

Slika 4.2: Prikaz koda funkcije dobrote tima u implementaciji

Ograničenja i filtriranje zmija koje zapinju u ciklusima kretanja i time skupljaju veliki broj koraka, što je dobro ocijenjeno funkcijom dobrote, u kodu su vidljiva kao predikati koji preskaču dodjeljivanja bodova u sumu dobrote ako je ijedna zmija skupila minimalni broj jabuka ili prešla broj koraka veći od 100 (vrlo ne razuman broj koraka za dvije žive zmije u mapi dimenzija 20 x20).

Manipulirajući funkcijom dobrote definirala se zadana vrsta suradnje. Nagrađuje se dugi životni vijek odnosno veliki broj koraka za obje zmije u timu kao i veliki i ujednačen dobitak i broj koraka za obje zmije. Tijekom evolucije jasno je rastao broj koraka, rastao broj dobitaka i smanjivala se razlika između zmija, no unatoč tome uvijek je jedna zmija značajno dominirala. Presudna odluka za konačno postignuće željenog

ponašanja agenata bila je dodavanje druge jabuke tj. dodavanje postavke da se u svakom trenutku u okolini nalaze točno dvije jabuke. Odluka inspirirana već spomenutim radom [2] dodala je dodatni cilj u okolinu za koji svaka zmija u svakom trenutku ima odvojenu metu.

5. Natjecateljska inačica koevolucije

5.1. Opis željenog ponašanja

Kompetitivno tj. natjecateljsko ponašanje agenta definirano je kao takvo ponašanje agenta gdje za cilj ima skupiti što veći broj jabuka i time biti najbolji.

5.2. Razvoj i metode

U prošlom poglavlju kroz razne inačice i pokušaje kao glavna prepreka ostvarenju učenja modela strategije surađivanja konstantno se pojavljivala kompetitivna narav agenata. Jedna od glavnih tehnika koja je dovela do konačnog kooperativnog ponašanja zmijica bila je dodavanje jednakog broja ciljeva (jabuka) u okolinu s više agenata. Iz toga se da izvući zaključak da bilo koja koevolucija u okolini u kojoj se kreće veći broj agenta, a sadrži jedan posao, zadatak ili cilj agentu, automatski agenta uči natjecateljskom ponašanju. Tome naravno pridonosi i funkcija dobrote koja u ovoj inačici, za zadatak probiranja najboljih agenta, ubraja isključivo dobitak pojedine zmije. Tako da u ovom poglavlju fokus zadatka nije bio postići natjecateljsko ponašanje kod agenta već proučiti drukčiju metodu sličnog utjecaja na ponašanje zmija. Promatrala se simulirana varijacija tehnike Self-play.

5.3. Self-play

U ovoj tehnici agent koristi ranije kopije sebe tj. svoje strategije kao protivnika. Tako agent, boreći se protiv agenta razine slične njegovoj, sebi postavlja ne prevelik izazov kojim ima priliku postupno poboljšavati svoju strategiju. Tehnika je realizirana tako da se u prvoj generaciji, kao i do sada, generiraju okoline s 2 zmije. No, samo jedna od tih zmija je nasumično generirani model GP-a. Druga zmija zauzima ulogu „zmije izazivačice“. U prvoj generaciji to je tvrdo kodirana zmija koja Hamiltonovim ciklu-

som prolazi mapu okoline. Ona ima dokazano uspješnu strategiju u okolini s jednim agentom, no u okolini s više agenata nije ni približno savršena. Kako joj je put čvrsto određen i prije pokretanja, okolina s više zmija koje joj bilo kada mogu prepriječiti put nije optimalna za takvu zmiju. Točna takva strategija savršena je za početni izazov našoj nasumično izgeneriranoj populaciji zmija. Nasumično izgenerirane zmije i dalje evolucijskim algoritmom unaprjeđuju strategiju igranja sve do trenutka kada i jedna jedina jedinka ne pobjedi „zmiju izazivačicu“ tj. ne skupi veći broj jabuka. Tada ulogu izazivača preuzima agent pobjednik i dalje se nastavlja evolucija, ali ovaj put agent igra protiv agenta sličnih ili čak istih sebi. Kroz ponovljene iteracije igranja igre protiv sebe, agent može učiti iz vlastitih iskustava i postupno poboljšavati donošenje odluka i strategiju.

6. Mješovito kooperativno-natjecateljska inačica koevolucije

6.1. Opis željenog ponašanja

Inačica s mješovito kooperativno-natjecateljskim ponašanjem zmija u okolini uvodi pojam timova zmija. Zmije iste okoline dijeli na timove, a korištena definicija kooperativno-natjecateljskog ponašanja navodi da zmije istog tima moraju pokazivati vrline suradnje tj. zmije istog tima maksimiziraju svoj životni vijek i istovremeno obje skupljaju podjednaki broj jabuka čiji se zbroj također maksimizira, dok timovi zmija međusobno moraju pokazivati kompetitivno ponašanje u kojem se svaki tim bori za veću funkciju dobrote i poboljšanje strategije.

6.2. Razvoj i metode

Ova inačica igre tj. sustava za učenje koristi obje tehnike Joint-action learning i Self-play opisane u prethodnim poglavljima. Prvo se radni okvir sustava prilagodio tako da podržava 2 tima u istoj okolini. Svaki tim sadrži svoje dvije zmije čijim podacima o okolini i položaju donosi odluku o njihovom kretanju po mapi. Zbrojivši sve, svaka mapa početkom života tada sadrži 4 zmije. Opisani postupak je realizacija Joint-action tehnike, a funkcija dobrote za pojedine timove ostaje ista kao razrađena funkcija u trećem poglavlju (izraz 3.1). Tehnika Self-play u ovoj verziji igre implementirana je tako što se na početku generira nasumični model „izazivač“ kojeg istog u svakoj okolini dijeli jedan tim. Ostali timovi u okolinama koriste nasumično generirane modele. Jednom kada model tj. tim „izazivač“ pobjedi njega zamjenjuje pobjednički tim i nastavlja se iterativno ciklus. Timovi tako postupno poboljšavaju strategiju kooperativno-

kompetitivnog ponašanja.

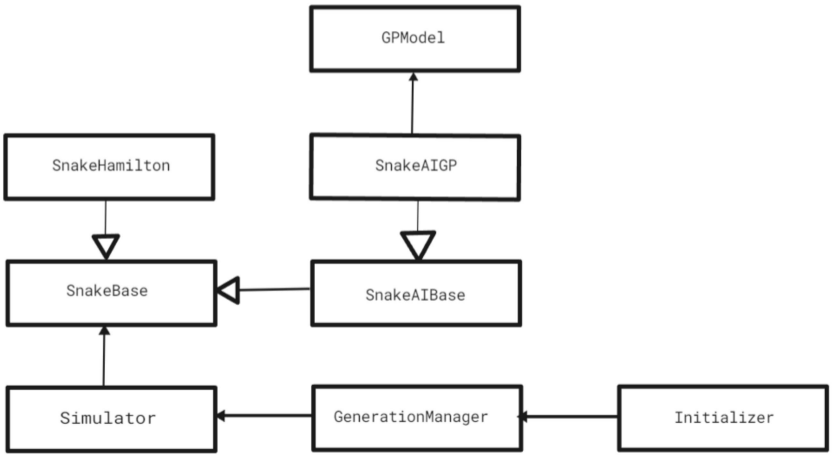
7. Implementacija

7.1. Implementacijski okvir

Radni okvir za pronalaženje i optimazicaju strategije nadogradnja je okvira prvotno implementiran za kolegij Projekt R[3]. Okvir je napisan u programskom jeziku C++, te se za vizualni prikaz (u ovom radu jedino omogućen za natjecateljsku inačicu učenja) koristi i više platformska multimedijalna biblioteka „Simple and Fast Multimedia Library“ (SFML). Implementiran okvir odvaja 2 glavna dijela: dio za natjecateljsku koevoluciju te dio za koevoluciju timova agenata koji se pokazao potrebnim za implementaciju joint-action učenja razrađenog u trećem poglavlju.

7.1.1. Okvir za natjecateljsku koevoluciju

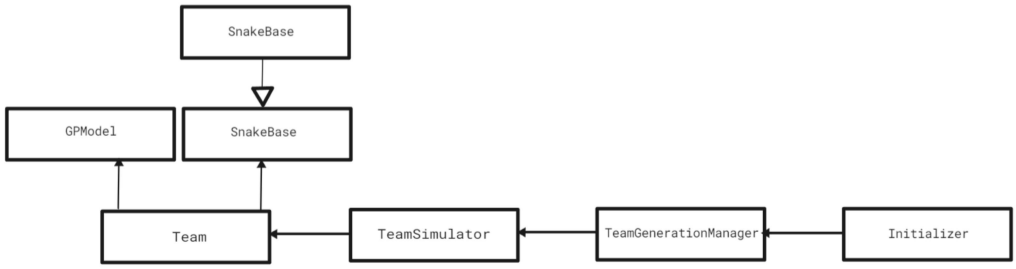
Razred GenerationManager ima zadaću inicijalizirati sve instance zmija te postaviti po dvije u simulatore. Za svaku novu generaciju radi selekciju, generira novu populaciju potomaka i razmješta ih opet dvije po dvije u simulatore. Simulator sadrži polje jedne igre tj. okolinu i mapu jednog životnog vijeka dviju zmiju te prati životni vijek (status jesu li žive ili mrtve te njihovo kretanje po mapi) zmija koje sadržava. Snake Base je bazna klasa koja sadrži osnovne podatke o položaju zmije i njezinom trenutnom uspjehu (broju skupljenih jabuka). Klasu nasljeđuju SnakeHamiltnon razred, koji predstavlja implementaciju tvrdo kodirane zmije koja prateći Hamiltonov ciklus na mapi izbjegava zidove i time nesmetano skuplja jabuke, te SnakeAIBase temeljna apstraktna klasa koja definira metode i algoritme strojnog učenja za pronalaženje najboljeg puta. Izvedena klasa iz SnakeAIBase je klasa SnakeAIGP. SnakeAIGP sadrži kao člansku varijablu model tipa GPModel. Modelu SnakeAIGP prenosi sve potrebne unosne vrijednosti okoline kojim onda model može donijet zaključak o svojoj idućoj akciji. Klasa modela također definira i sve genetske operatore nad sobom (mutaciju i križanje). Razred Initializer rukuje grafikom te pokreće rad GenerationManager klase.



Slika 7.1: Dijagram komponenti okvira za natjecateljsku koevoluciju

7.1.2. Okvir za koevoluciju agenta u timovima

Drukčiji dijagram komponenti u ovom radnom okviru je rezultat uvođenja timova zmija u evoluciju. Razredi Initializer i TeamGenerationManager zadržavaju iste odgovornosti kao i njihovi analogni razredi u okviru za natjecateljsku koevoluciju. Glavna razlika dvaju dijelova okvira očituje se ponajviše u klasi TeamSimulator. Klasa TeamSimulator umjesto više agenata zmija sadrži do 2 tima zmija te rukuje mapom koja predstavlja okolinu jedne igre i prati životni vijek svih zmija svih timova koje sadrži. Dodan je razred Team koji sadrži dvije zmije sa ciljem suradnji u parom. Team također sadrži i GPModel objekt kojim sada simulirajući JAL metodu (objašnjena u poglavlju 3) prima ulazne vrijednosti svojih zmija i donosi odluke za iduću akciju svake zmije.



Slika 7.2: Dijagram komponenti okvira za koevoluciju agenta u timovima

7.2. Model genetskog programiranja

GPModel predstavlja „mozak“ zmije tj. agenta učenja. Realiziran je kao struktura binarnog stabla čiji čvorovi sadrže operatorske funkcije dok se u listovima čuvaju numeričke konstante i vrijednosti unosa koje agent prima iz okoline. Klasa je odgovorna za sve dijelove i mehanizme genetskog programiranja (inicijalizaciju jedinke, crossing i mutacija operatore) kao i za donošenje odluke o idućem koraku zmije temeljem unosnih podataka iz okoline. Inicijalizacija jedne jedinke izvedena je kao funkcija generiranja binarnog stabla. Rekurzivno stvara strukturu stabla tako da nasumično generira čvorove pazeći na ograničenje dubine 6 koje smanjuje i obuzdava složenost modela, istovremeno dozvoljavajući dovoljno listova za sve potrebne unose. Ovaj proces generira raznoliki skup struktura stabala, omogućavajući istraživanje različitih kombinacija funkcija i ulaza tijekom evolucijskog procesa tj. čini raznoliki genetski bazen. Genetski operator križanja realiziran je kao funkcija koja izvodi križanje između dvije instance GPModela. Zamjenom čvorova na nasumično odabranim mjestima svakog pojedinog modela stvara dva nova potencijalna potomka te nasumičnim izborom odabire jedan kao novog agenta iduće generacije u nastavku evolucije. Implementiran genetski operator mutacije nasumično odabire čvor u stablu GP-a i zamjenjuje njegovu vrijednost novim nasumično generiranim podstablom manje dubine. Time se uvodi varijacija i potencijalna evolucija u strukturi stabla GP-a što omogućuje istraživanje novih rješenja tijekom evolucijskog procesa. Jednostavnom šetnjom stablom i izvodeći operacije od listova prema korijenu dobiva se krajnja vrijednost koju potom u mapToAction funkciji klasa agenta preslikava u odluku zmije. Spomenuta funkcija još je jedna od razlika između dva cilja ponašanja i učenja zmija tj. razlika dva dijela okvira razrađena u prošlim potpoglavljima. Kako funkcija preslikavanja rezultata evaluacije GP stabla u akciju za tim preslikava u jednu od mogućih 9 permutacija triju akcija dok za pojedinu zmiju preslikava u 3, mapToAction funkcija se mora razlikovati. Odabrane su nove funkcije čvorova stabla koja se odmiču od nule i daju veći domet rezultata evaluacije stabla i time raznolikije intervale za funkciju preslikavanja u korake. Jednadžbe 7.1 i 7.2 prikazuju intervale funkcije preslikavanja za opisana dva slučaja.

$$Preslikavanje(x) = \begin{cases} lijevo, & x < 0.5 \\ desno, & x > 0.5 \\ ravno, & x = 0.5 \end{cases} \tag{7.1}$$

$$
Preslikavanje(x) = \begin{cases}
(lijevo, lijevo), & x < -15 \\
(lijevo, desno), & -15 \leq x < -10 \\
(desno, lijevo), & -10 \leq x < -5 \\
(desno, ravno), & -5 \leq x < -1 \\
(ravno, desno), & -1 \leq x < 0 \\
(ravno, lijevo), & 0 \leq x < 1 \\
(lijevo, desno), & 1 \leq x < 5 \\
(desno, desno), & 5 \leq x < 10 \\
(ravno, ravno), & x \geq 10
\end{cases} \tag{7.2}
$$

8. Eksperimenti

U ovom poglavlju prikazani su rezultati eksperimenata učenja svih 3 različitih inačica igre. Kako je u pitanju izvođenje 3 različita algoritma na velikom broju generacija prikazane su najznačajnije inačice svake verzije na dijelovima gdje je neka značajna promjena iznimno utjecala na pospješivanje algoritma u dobivanju i učenju željenog karaktera tj. ponašanja zmija. Prikazana su učenja svih inačica, uspoređene inačice algoritama od značaja. Sva izvođenja provedena su s konfiguracijom prikazanoj u tablici 8.1. Svi prikazani eksperimenti izvedeni su 10 puta. Za sve algoritme i njihove inačice prikazan je graf konvergencije funkcije dobrote kroz generacije. U grafu se prikazuju maksimum, minimum, srednja vrijednost i medija dobrote najbolje jedinke za sva pokretanja u nekom trenutku. Dodatno se još prikazuje aproksimirana krivulja vrijednosti dobrote svih najboljih jedinki kroz generacije.

Tablica 8.1: Temeljna konfiguracija svih eksperimenata

Konfiguracija	Vrijednost
veličina mape	20x20 polje
broj zmija koje postaju roditelji u sljedećoj gen	5
vjerojatnost mutacije kromosoma (cijelog modela/potomka)	0.009
maksimalna dubina GP stabla	6
broj različitih funkcija čvorova GP stabla	4

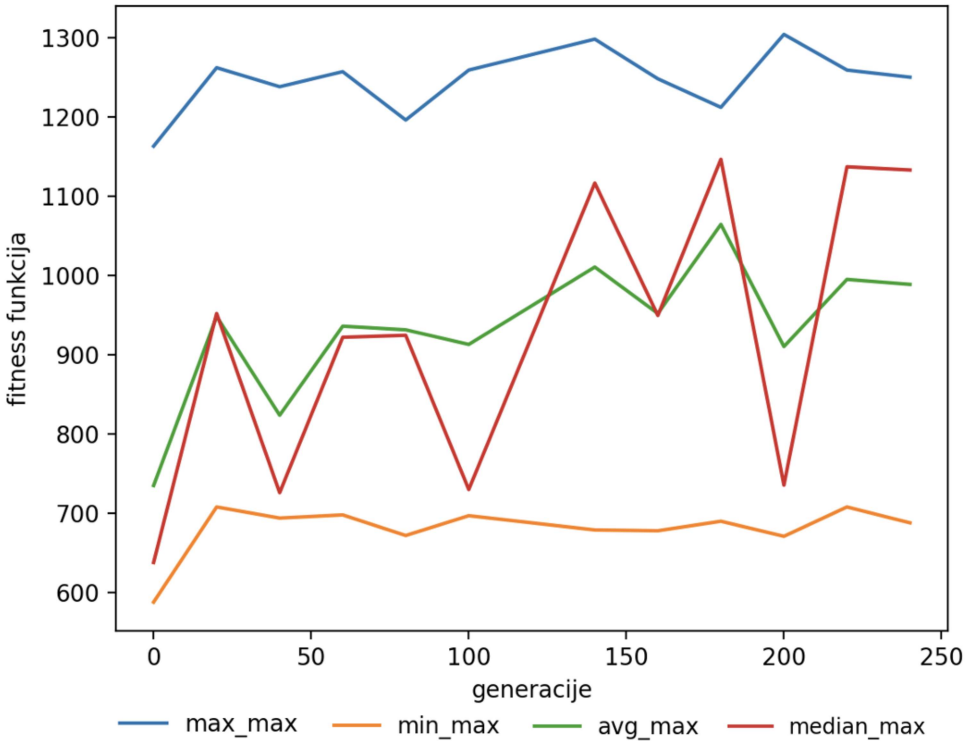
8.1. Kooperativna inačica

Za inačicu koja je agente iste okoline učila suradnji, u razradi u prijašnjim poglavljima, najučinkovitiji algoritam se ispostavio biti implementacija tehnike JAL. Dodatno, pravi preokret u ponašanju zmijica pokazao se nakon uvođenja dviju jabuka za 2 agenta unutar iste okoline. Tako je prikazan rad algoritma JAL tehnika s 1 jabukom i JAL tehnika s 2 jabuke te njihova usporedba.

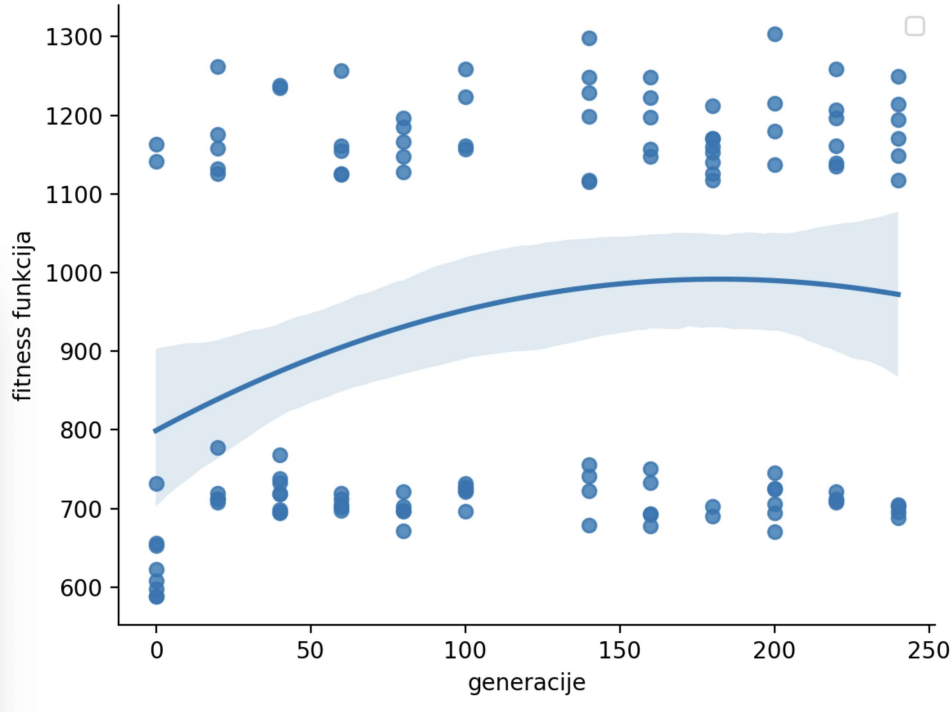
Tablica 8.2: Konfiguracija kooperativne inačice

Konfiguracija	Vrijednost
broj okolina (simulatora)	100
broj agenata (timova)	1
broj zmija u timu	2

8.1.1. JAL s 1 jabukom

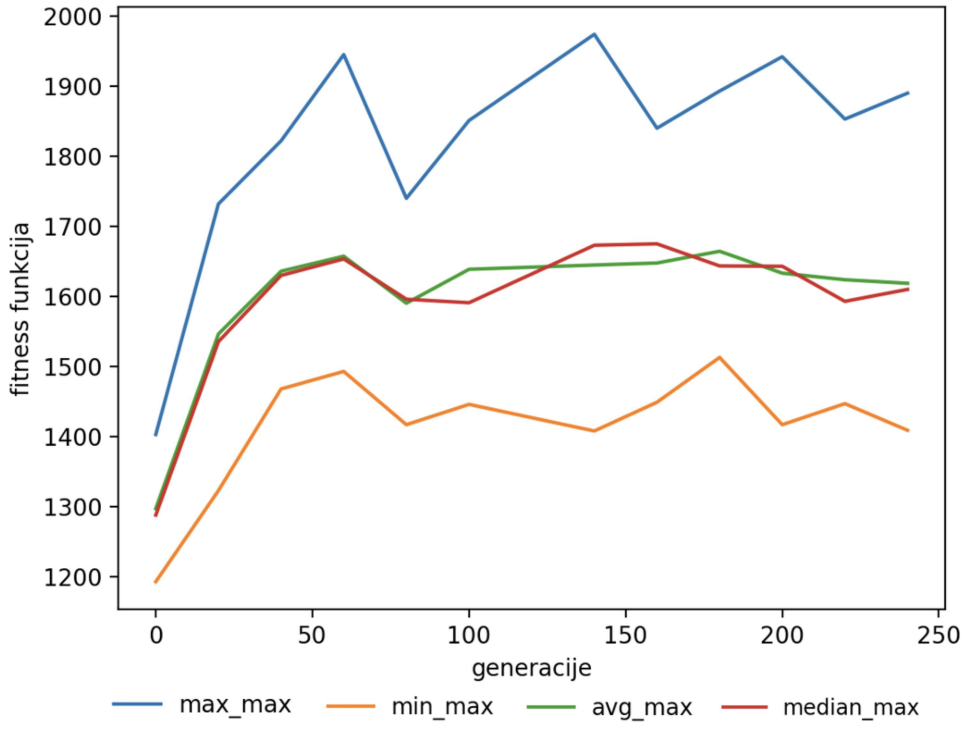


Slika 8.1: Konvergencija JAL algoritma s 1 jabukom kroz generacije

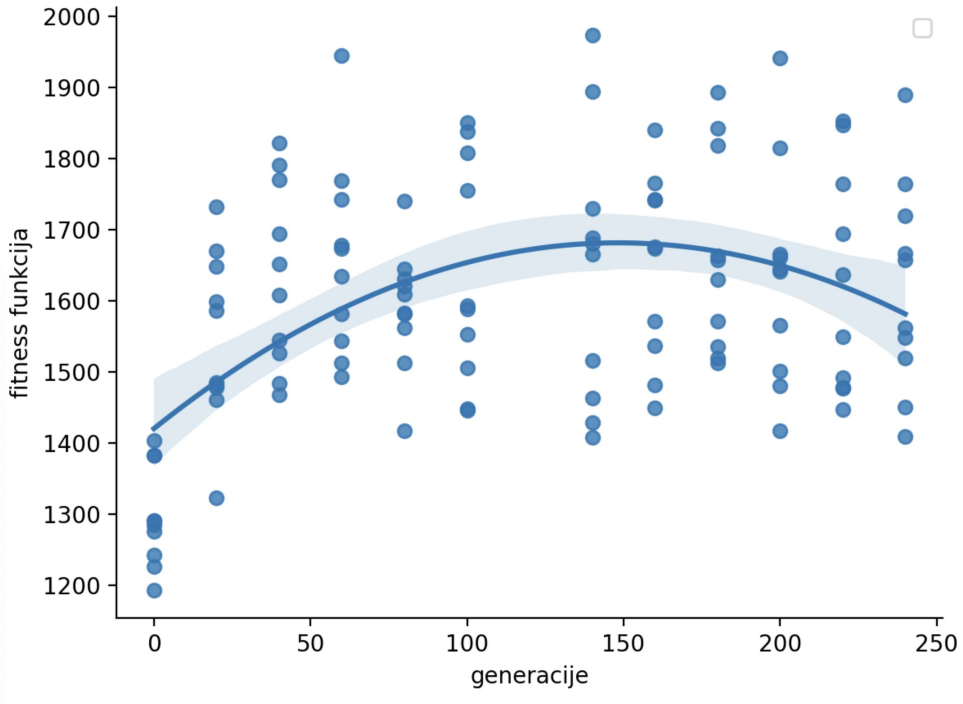


Slika 8.2: Aproksimirana krivulja vrijednosti funkcija dobrote najboljih jedinki kroz generacije

8.1.2. JAL s 2 jabuke

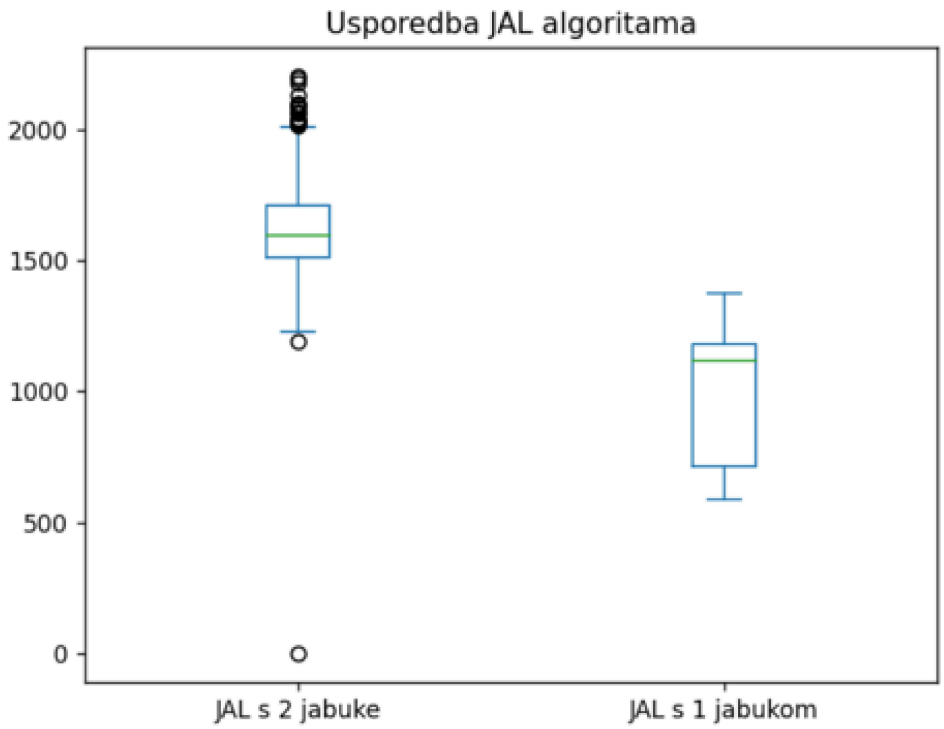


Slika 8.3: Konvergencija JAL algoritma s 2 jabukom kroz generacije



Slika 8.4: Aproksimirana krivulja vrijednosti funkcija dobrote svih najboljih jedinki kroz generacije

8.1.3. Usporedba



Slika 8.5: Box plot usporedba JAL algoritama

Dijagram usporedno prikazuje boxplotove dvaju JAL algoritama - lijevo je JAL s dvije jabuke, desno je JAL s jednom jabukom. Metodama deskriptivne statistike lako je iščitati da je JAL s dvije jabuke uspješniji u odnosu na svog suparnika. Uzimajući medijan kao glavnu mjeru sredine vrijednosti performansi algoritama, zaključuje se da je JAL s dvije jabuke puno uspješniji model (medijan veći za 25 posto). Također je od značaja primijetiti da prvi prikazani algoritam ima znatno manji IQR, time ukazujući na gušće „nakupljanje“ vrijednosti oko medijana. Zanimljivo je za primijetiti da je velik broj stršećih vrijednosti (eng. outlier) u slučaju JAL algoritma s 2 jabuke, odnosno da je varijanca u njegovim performansama primjetno veća te bi mogla biti i dio objašnjenja za neuobičajen pad aproksimirane krivulje vrijednosti funkcije dobrote algoritma u većim generacijama. Uzrok veće varijance mogla bi biti neobična i unutar 250 generacija ne dovršena konvergencija ili jednostavno mali broj uzoraka kojim se aproksimirala krivulja.

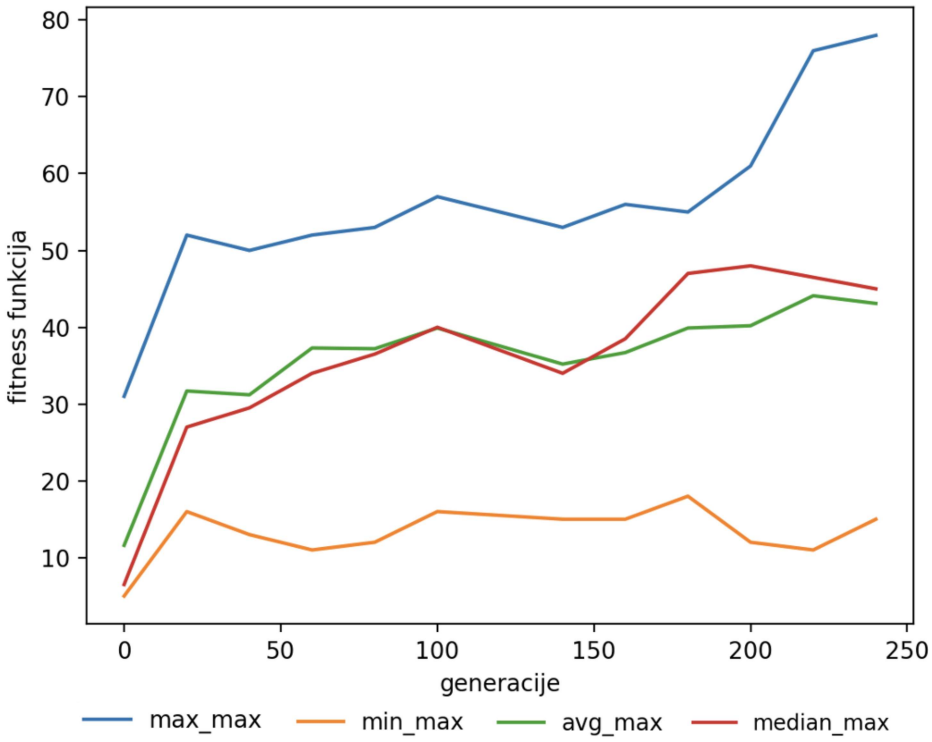
8.2. Natjecateljska inačica

Natjecateljska verzija sustava za učenje modela imala je znatnu prednost prilikom razvoja, kako agenti u zajedničkoj okolini svako s jednim "suprotstavljajućim" ciljem naginju kompetitivnom ponašanju. Implementirana tehnika self-play uvedena je kao algoritam za pospješivanje učenja tog karaktera. Tako se u ovom potpoglavlju prikazuje inačica natjecanja agenata bez tehnike i s navedenom tehnikom te njihova usporedba.

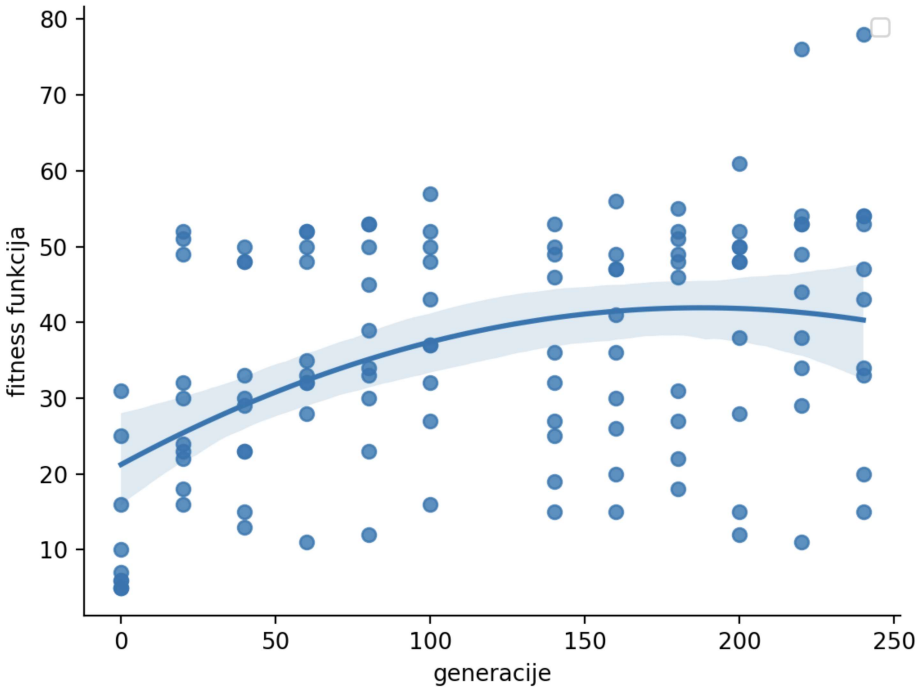
Tablica 8.3: Konfiguracija natjecateljske inačice

Konfiguracija	Vrijednost
broj okolina (simulatora)	100
broj agenata u okolini	2
veličina populacije u generaciji	200

8.2.1. Natjecateljska okolina bez self-play tehnike

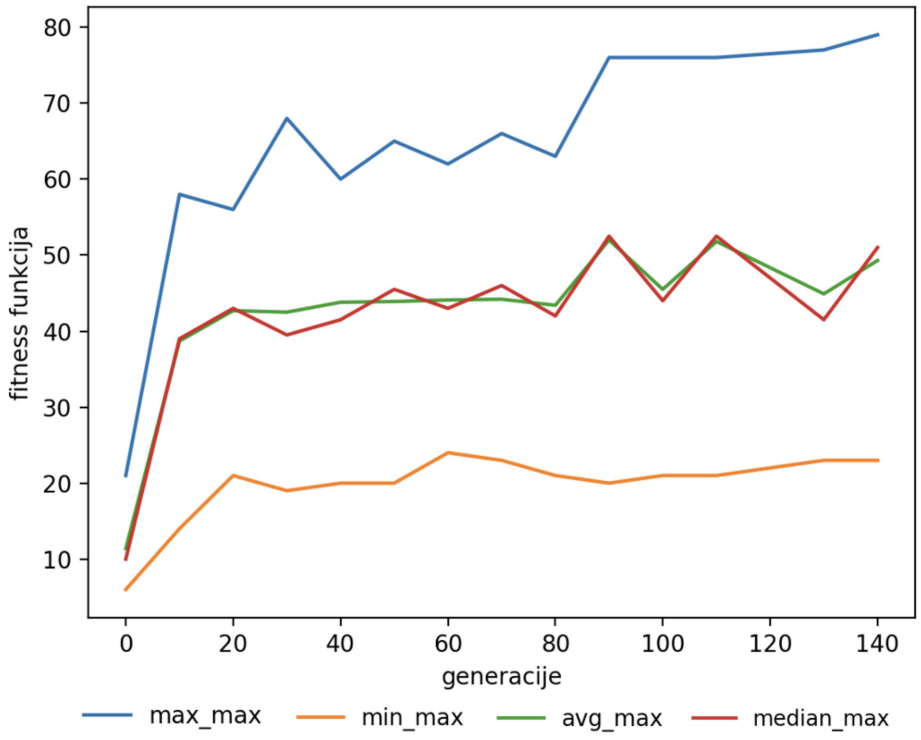


Slika 8.6: Konvergencija algoritma natjecateljske okoline kroz generacije

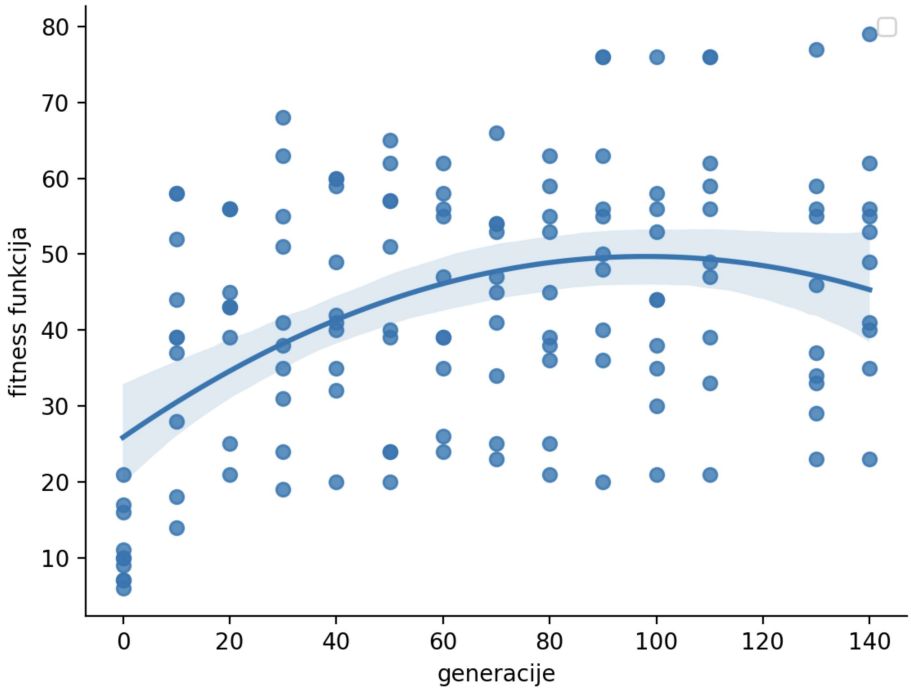


Slika 8.7: Aproksimirana krivulja vrijednosti funkcija dobrote svih najboljih jedinki kroz generacije

8.2.2. Self-play tehnika

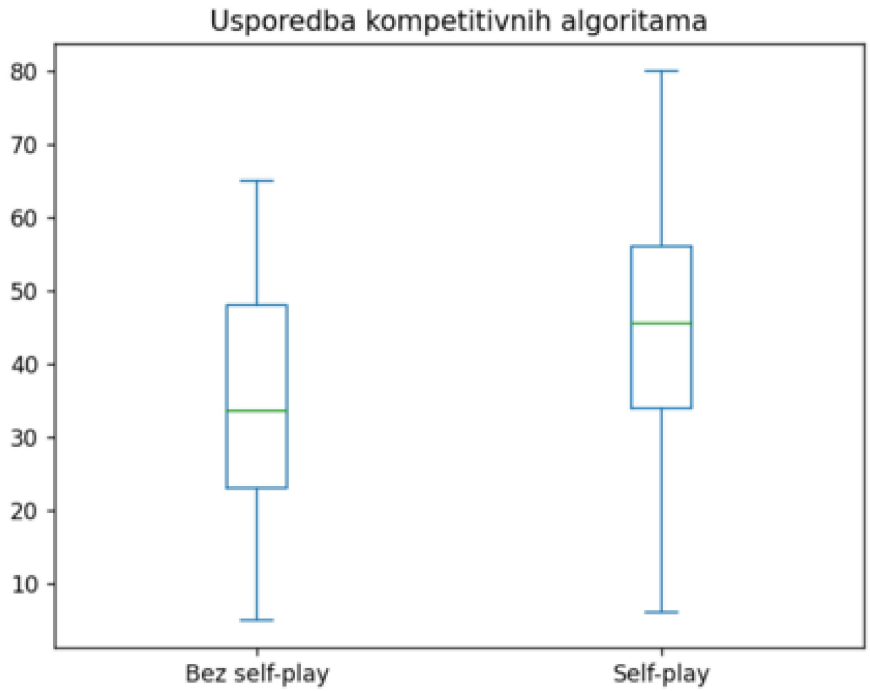


Slika 8.8: Konvergencija Self-play algoritma kroz generacije



Slika 8.9: Aproksimirana krivulja vrijednosti funkcija dobrote svih najboljih jedinki kroz generacije

8.2.3. Usporedba



Slika 8.10: Box plot usporedba natjecateljskih algoritama

Dijagram usporedno dva boxplota svaki predstavljajući jednu inačicu kompetitivnog algoritma. Algoritam Self-play tehnike ostvaruje osjetno veće vrijednosti funkcije dobrote za sva tri kvartila i oko 30 posto veći medijan, a u prilog zaključku da je uspješniji od algoritma s implementiranim Self-play tehnikom ide i činjenica da je IQR nešto manji u 1. algoritmu. Iščitavanjem ovog dijagrama zaključak je da je model Self-play uspješniji.

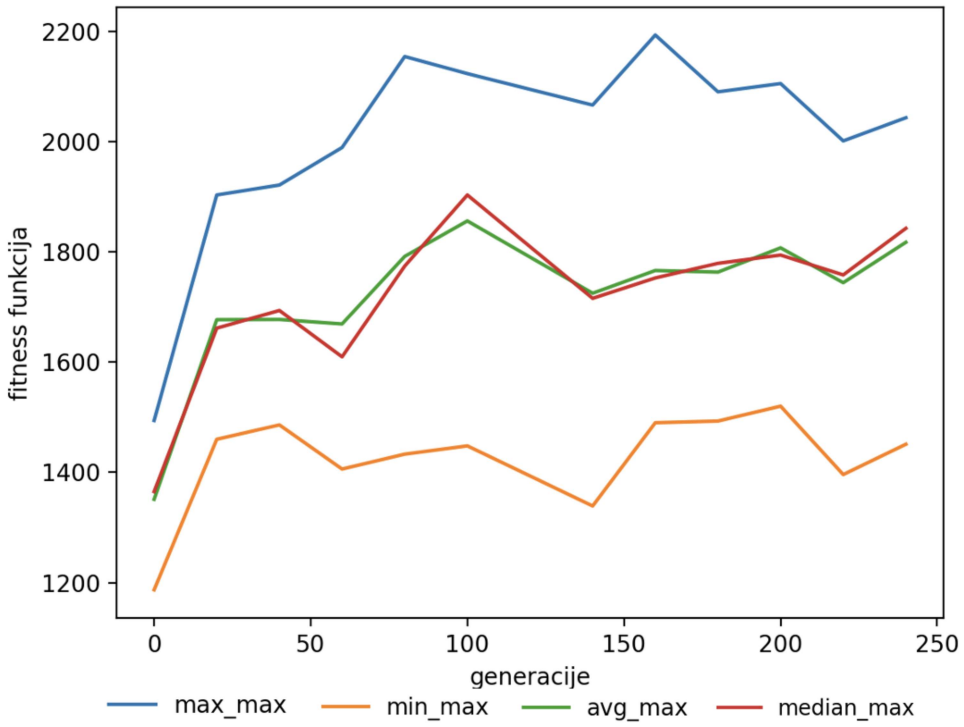
8.3. Mješovito kooperativno-natjecateljska inačica

Mješovita inačica implementirana je kao kombinacija dviju tehnika: Joint-action learning i Self-play. Za nju je isto prikazan rad kroz 250 generacija te dodatno eksperimenti za optimiziranje parametra broja simulatora tj. broja okoline u kojima se natječu 2 tima.

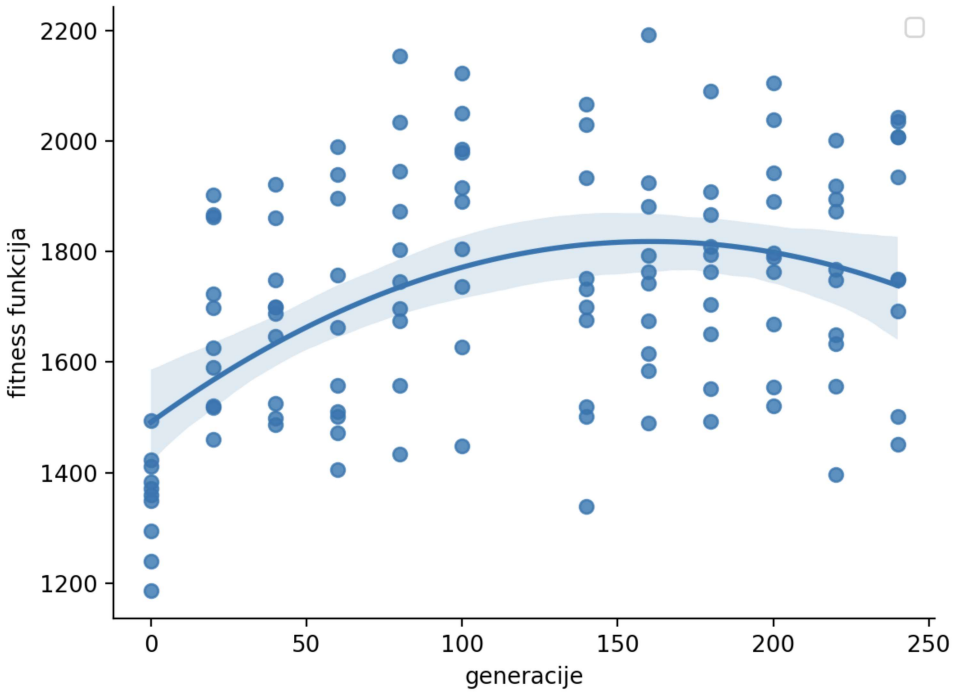
Tablica 8.4: Konfiguracija mještovite inačice

Konfiguracija	Vrijednost
broj okolina (simulatora)	100
broj agenata (timova)	2
broj zmija u timu	2
velicina populacije u generaciji	200

8.3.1. Mješovita inačica



Slika 8.11: Konvergencija algoritma mješovitog ponašanja kroz generacije



Slika 8.12: Aproksimirana krivulja vrijednosti funkcija dobrote svih najboljih jedinki kroz generacije

Iz grafova na slikama 8.11 i 8.12 vidimo da se kriterij zaustavljanja javlja oko 170. generacije. U idućem tesktu prikazano je prvih 5 najboljih timova u generaciji jednog pokretanja algoritma. Za svaki tim prikazane su obje zmije tima i njihovi rezultati. Uočljivo je da jedinke uče tj. napreduju u smjeru željenog ponašanja. Zmijama unutar tima raste ukupni broj koraka i broj skupljenih jabuka te je istovremeno očuvana i smanjena razlika između dobitka i napravljenih koraka pojedinih zmija članica tima. Prikazan je napredak u 1., 28. i 83. generaciji pokretanja.

Generation 1

1.team with a team score of: 1195

- individual snake score -> Snake 1: 5; Snake 2: 7
- individual snake step -> Snake 1: 25; Snake 2: 45

2.team with a team score of: 1134

- individual snake score -> Snake 1: 5; Snake 2: 5
- individual snake step -> Snake 1: 19; Snake 2: 20

3.team with a team score of: 1124

- individual snake score -> Snake 1: 5; Snake 2: 6
- individual snake step -> Snake 1: 5; Snake 2: 30

4.team with a team score of: 1120

- individual snake score -> Snake 1: 5; Snake 2: 5
- individual snake step -> Snake 1: 13; Snake 2: 19

5.team with a team score of: 1107

- individual snake score -> Snake 1: 5; Snake 2: 5
- individual snake step-> Snake 1: 7; Snake 2: 15

Generation 28

1. team with a team score of: 1854

- individual snake score -> Snake 1: 13; Snake 2: 17
- individual snake step -> Snake 1: 83; Snake 2: 83

2. team with a team score of: 1591

- individual snake score -> Snake 1: 10; Snake 2: 10
- individual snake step -> Snake 1: 98; Snake 2: 98

3. team with a team score of: 1534

- individual snake score -> Snake 1: 11; Snake 2: 8
- individual snake step -> Snake 1: 87; Snake 2: 91

4. team with a team score of: 1526

- individual snake score -> Snake 1: 13; Snake 2: 6
- individual snake step -> Snake 1: 88; Snake 2: 91

5. team with a team score of: 1508

- individual snake score -> Snake 1: 13; Snake 2: 5
- individual snake step -> Snake 1: 95; Snake 2: 98

```
Generation 83
1. team with a team score of: 1651
- individual snake score -> Snake 1: 8; Snake 2: 17
- individual snake step -> Snake 1: 69; Snake 2: 97
2. team with a team score of: 1592
- individual snake score -> Snake 1: 13; Snake 2: 8
- individual snake step -> Snake 1: 90; Snake 2: 99
3. team with a team score of: 1553
- individual snake score -> Snake 1: 10; Snake 2: 10
- individual snake step -> Snake 1: 79; Snake 2: 79
4. team with a team score of: 1544
- individual snake score -> Snake 1: 16; Snake 2: 5
- individual snake step -> Snake 1: 72; Snake 2: 75
5. team with a team score of: 1514
- individual snake score -> Snake 1: 12; Snake 2: 8
- individual snake step -> Snake 1: 63; Snake 2: 67
```

8.4. Usporedba inačica

Dosad odrađeni eksperimenti su pokazivali maksimiziranje funkcije dobrote te su pomoću funkcije izraženi rezultati svakog algoritma. Tablice 8.5 i 8.6 na jednoj provedbi najuspješnijeg algoritma svake od inačica prikazuju rezultate u kontekstu tj. prirodi problema u kojem je lakše uočiti naučena ponašanja. Kao predstavnik kooperativne inačice provodio se JAL algoritam s 2 jabuke, a za natjecateljsku, self-play algoritam.

Tablica 8.5: Usporedba broja skupljenih jabuka po inačicama

Inačica	1.gen		50. gen		120. gen		260. gen	
	Zmija 1	Zmija 2	Zmija 1	Zmija 2	Zmija 1	Zmija 2	Zmija 1	Zmija 2
Mješovita	1	4	4	14	2	18	8	5
Kooperativna	3	4	2	18	2	9	9	6
Natjecateljska	2		45		50		59	

Prikazani su brojevi skupljenih jabuka najboljih jedinki za sve inačice koevolucija. Brojevi koraka su prikazani samo za mješovitu i kooperativnu inačicu gdje se i jedino smatraju relevantnim za definirano ponašanje. Natjecateljska inačica se također razlikuje po prikazu rezultata samo jedne najbolje zmije dok za druga dva algoritma, koji

Tablica 8.6: Usporedba broja koraka po inačicama

Inačica	1.gen		50. gen		120. gen		260. gen	
	Zmija 1	Zmija 2	Zmija 1	Zmija 2	Zmija 1	Zmija 2	Zmija 1	Zmija 2
Mješovita	39	64	69	99	32	98	77	68
Kooperativna	37	68	51	100	64	73	85	77

rade s timovima zmija, prikazujemo rezultate jedne i druge zmije tima. Sva 3 algoritma postižu učenje željenog ponašanja. Kod natjecateljske inačice vidljivo je kako najbolja jedinka skuplja daleko najviše jabuka te je izraženo rano brzo konvergiranje broja jabuka. Interesantno učenje opaža se i kod mješovite i kod kooperativne inačice, gdje se u ranijim generacijama vidi porast u ukupnom broju koraka i jabuka tima, no u timu je uvijek blago izražena jedna "dominantnija" tj. uspješnija zmija. No, pred kraj, u 260. prikazanoj generaciji vidi se smanjenje razlika između zmija u timu i sve većoj kooperaciji.

9. Zaključak

Ovaj rad proučava problem učenja više agenata, modeliranih genetskim programiranjem temeljenom na stablu, u istoj okolini pomoću evolucijskog algoritma. Problem na kojem temeljimo naše agente, njihovu okolinu i željeno ponašanje je inspiriran poznatom video igrom Zmija. U početku rad iznosi osnove i pokriva bazu evolucijskih algoritama, genetskog programiranja i koevolucije te predstavlja agenta i okolinu. Razvijeno je nekoliko inačica igre koje se razlikuju ovisno o cilju više agenata za igranje. Kroz svaki od kojih se modelira optimizacija. Kooperativni način rada gdje je cilj, za više zmija koje igraju, da zajedno ostvare najviše bodova dok istovremeno izbjegavaju "ubijanje“ jedni drugih u njihovom zajedničkom okruženju. Natjecateljski način rada u kojem je cilj svake zmije postići što je moguće veći rezultat bez obzira na druge agente te konačno, Mješoviti kooperativno-natjecateljski način gdje podijeljeni u timove od po dva agenta u istom timu surađuju dok se timovi natječu za najbolji rezultat. Razvojem programskog sustava koji omogućava učenje zmija dolazimo i do najkonkurentnijih optimizacijskih algoritama, oni razrađeni i s implementiranim modifikacijskim tehnikama algoritama učenja – Joint-action učenje i Self-play. Za kooperativnu verziju prikazani su rezultati i usporedba dviju inačica algoritma: JAL algoritam s 1 jabukom te JAL algoritam s 2 jabuke. Taj parametar postigao je konačno učenje željene suradnje među zmijama te se iz rezultata jasno vidi prednosti druge inačice u maksimiziranju funkcije dobrote prilagođenoj kooperaciji. Za natjecateljsku verziju također su prikazana dva različita algoritma učenja: obična koevolucija više agenata i implementirana self-play tehnika. Iz rezultata jasno se vidi poboljšanje učenja i pojačanja natjecateljske naravi algoritma implementirane Self-play tehnike. Na posljetku još je prikazana i konvergencija i rad mješovite inačice.

LITERATURA

[1] Mislav Jelić, Eva Kaštelan, Luka Nestić, Hana Ujčić, i Joško Vrsalović. Zmijica. *za kolegij Projekt R na Fakultetu elektrotrhnike i Računarstva, Sveučilište u Zagrebu*, 2023.

[2] Fumito Uwano i Keiki Takadama. Utilizing observed information for no-communication multi-agent reinforcement learning toward cooperation in dynamic environment. *SICE Journal of Control, Measurement, and System Integration*, 12(5):199–208, 2019.

[3] Jiachen Yang, Alireza Nakhaei, David Isele, Kikuo Fujimura, i Hongyuan Zha. Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning. *arXiv preprint arXiv:1809.05188*, 2018.

[3] [2] [1]

Kooperativno, natjecateljsko i mješovito kooperativno-natjecateljsko podržano učenje za igru Zmija s više agenata

Sažetak

Zadatak i cilj rada je opisati problem pronalaženja i optimizacije strategije igranja u igrama s više igrača (agenata) te realizirati model učenja strategije i sustav za učenje temeljen na evolucijskim algoritmima. Razvijeno je nekoliko verzija igre "Zmija" koje se razlikuju ovisno o cilju više agenata za igranje. Kroz svaki od kojih se modelira optimizacija. Optimizacija je modelirana kao koevolucija u evolucijskom algoritmu dok su agenti genetski programi temeljeni na stablu. Eksperimentalno se kroz rad utvrđuje učinkovitost nekoliko optimizacijskih algoritama ovisnih o parametrima okruženja za učenje. Kao glavni konkurenti nekoliko proučavanih optimizacijskih algoritama, razmatraju se modifikacijske tehnike implementiranog algoritma učenja – Joint-action učenje i Self-play. U radu je prikazan izvorni kod, dobiveni rezultati uz njihovo objašnjenje, razradu i korištenu literaturu.

Ključne riječi: Umjetna inteligencija, evolucijski algoritam, evolucijsko računarstvo, Genetski algoritam, Genetsko programiranje, Multi-agent učenje, Optimizacija

Cooperative, Competitive and Mixed cooperative-competitive multi-agent reinforcement learning for the game Snake

Abstract

The task and goal of the dissertation is to describe the problem of finding and optimizing a playing strategy in games with multiple players (agents) and to realize a strategy learning model and system for learning based on evolutionary algorithms. Several versions of the game "Snake" are developed that vary by the goal of the multiple playing agents through each of which optimization is modeled. The optimization is modeled as a coevolution in an evolutionary algorithm while the agents are tree-based genetic programs. Experimentally, throughout the dissertation, the effectiveness of several optimization algorithms dependent on the learning environment parameters is determined. As top contenders of the several studied optimization algorithms, the modification techniques of the implemented learning algorithm joint-action learning and self-play are considered. The dissertation presents the source code, the obtained results alongside their explanation, elaboration, and utilized literature.

Keywords: Artificial intelligence, Evolutionary algorithm, Evolutionary computation, Genetic algorithm, Genetic programming, Multi-agent learning, Optimization