



UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U
NOVOM SADU




Goran Rajn

Upotreba evolutivnih algoritama za pronalaženje rešenja u igrama dekodiranja

MASTER RAD
- Master akademske studije -

Novi Sad, 2016

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	ЗАДАТАК ЗА ИЗРАДУ ЗАВРШНОГ (MASTER) РАДА	Лист/Листова:
		3/40

(Податке уноси предметни наставник - ментор)

Врста студија:	а) Основне академске студије б) Основне струковне студије
Студијски програм:	Рачунарство и аутоматика
Руководилац студијског програма:	Др Никола Јорговановић, ред. проф.

Студент:	Стефан Анђелић	Број индекса:	E2 57 / 2013
Област:	Софтверско инжењерство		
Ментор:	Др Ђорђе Обрадовић, доцент		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ ЗАВРШНОГ (МАСТЕР) РАДА:

Употреба еволутивних алгоритама за проналажење решења у играма декодирања

ТЕКСТ ЗАДАТКА:

- Извршити дизајн и спецификацију софтверског система за решавање игре декодирања применом еволутивних алгоритама.
- Описати основне појмове и дефиниције из области еволутивног рачунарства.
- Извршити спецификацију и имплементацију софтверског система користећи програмски језик C#.
- Верификацију имплементираног система извршити на примеру игре Мастерминд и измерити његове перформансе.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Ментора

Sadržaj

1.	UVOD	1
2.	OSNOVNI POJMOVI I DEFINICIJE	3
2.1.	Evolutivno računarstvo	3
2.2.	Evolutivni ciklus	4
2.3.	Biološki aspekt.....	5
2.3.1.	Geni	6
2.3.2.	Reprodukcija	7
2.4.	Genetski algoritmi.....	8
2.4.1.	Prostor pretrage	9
2.4.2.	Genetski operatori	11
2.4.3.	Selekcija	11
2.4.4.	Ukrštanje	13
2.4.5.	Mutacija	14
2.4.6.	Evaluacija rešenja	15
2.4.7.	Koraci algoritma	16
2.4.8.	Igra Mastermind	17
3.	SPECIFIKACIJA I IMPLEMENTACIJA SOFTVERA	19
3.1.	Arhitektonski šablon	20
3.2.	Model.....	21
3.2.1.	Klasa Individual	22
3.2.2.	Klasa Population	23
3.2.3.	Klasa Generation.....	23
3.2.4.	Klasa Utility	24
3.3.	Dijagram aktivnosti aplikacije	25
3.4.	Korisnički interfejs	26
3.4.1.	Komponenta TablePanel	27
3.4.2.	Komponenta CombinationPanel	28
3.4.3.	Komponenta FeedbackPanel	28
3.4.4.	Komponenta FitnessPanel	28
3.4.5.	Dijalog sa opcijama.....	29
3.5.	Implementacija genetskog algoritma	30
4.	VERIFIKACIJA REŠENJA	33
5.	ZAKLJUČAK	35
6.	LITERATURA	36
7.	BIOGRAFIJA	37
	KLJUČNA DOKUMENTACIJSKA INFORMACIJA	39
	KEY WORDS DOCUMENTATION	40

1. UVOD

Svet koji vidimo danas, sadržan je od velikog broja različitih vrsta organizama koji su izuzetno dobro prilagođeni u okruženju u kojem borave. To je rezultat evolucije, tj. procesa koji se u prirodi odvija već tri milijarde godina. Složenost i visoka prilagođenost današnjih životnih formi postignuta je rafinisanjem i kombinacijom genetskog materijala u tom vremenskom periodu.[1]

Neki od prvih naučnika na polju računarstva, kao što su Alan Turing, John von Neuman i Norbert Wiener su u velikoj meri bili motivisani idejama o spoju računarskih programa i inteligencije koja bi donela prirodne sposobnosti učenja i adaptacije prema okruženju.

Ovi začetnici računarskih nauka su bili u istoj meri zainteresovani za biologiju i psihologiju, koliko i za ostale segmente računarstva, te su koristili prirodne sisteme kao metafore u ostvarenju svojih vizija. Iz tog razloga, ne bi trebalo da čudi što su računari od prvih dana bili korišteni, ne samo u vojne svrhe, već i za modelovanje mozga, oponašanja ljudskog učenja i simuliranja biološke evolucije. [2]

Danas, genetski algoritmi imaju široko područje primene, pre svega, zato što efikasno nalaze zadovoljavajuća rešenja na velikim prostorima pretrage, za šta bi tradicionalnim metodama trebalo mnogo više vremena. Neke od oblasti u kojima se koriste su : Auto-moto industrija, robotika, evolucija hardvera, pravljenje rasporeda, izračunavanje putnih ruta i regulisanje saobraćaja, industrija igara, bankarstvo, razvoj strategija investiranja, marketing, itd.

Zadatak ovog rada je implementacija igre Mastermind (skočko), gde bi se pored klasičnog načina korišćenja (igrač pogađa rešenje), koristio i genetski algoritam za pronalaženje pravog rešenja u konačnom broju pokušaja. Rešenje predstavlja kombinaciju nekoliko znakova, gde mogu postojati i duplikati, iz konačnog skupa mogućih znakova. Od veličine tražene kombinacije zavisi i veličina prostora pretrage.

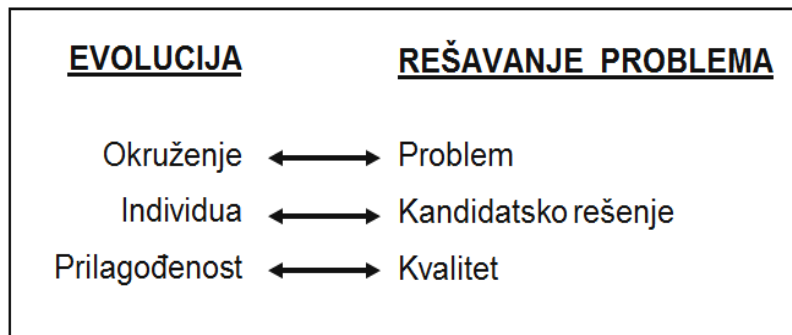
Dok bi standardne metode pretraživanja bile sporije i zahtevale više informacija, korišteni algoritmi efikasno i u zatom broju pokušaja dolaze do zadovoljavajućih rezultata zbog svoje sposobnosti da usmeravaju pretragu u dobrom pravcu na osnovu istorijskih podataka, što je ujedno bio i razlog za njihovo korišćenje u ovom radu.

2. OSNOVNI POJMOVI I DEFINICIJE

2.1. Evolutivno računarstvo

Ovaj deo računarske inteligencije svoju inspiraciju pronalazi u procesu prirodne evolucije i očiglednih dokaza o njegovoj moći putem mnoštva različitih vrsta organizama koji su izuzetno dobro prilagođeni za život u svojoj sredini. Osnovna metafora evolutivnog računarstva mogla bi biti prikazana ovako:

- 1) Svako okruženje je ispunjeno jedinkama koje se međusobno bore za opstanak i reprodukciju.
- 2) Prilagođenost individua je određena okruženjem i odnosi se na sposobnost organizma da preživi i da se razmnožava u tom prostoru.
- 3) Selekcija jedinki je neizbežna zbog ograničenosti resursa okruženja[3].



Slika 2.1 Osnovna metafora evolutivnog računarstva koja povezuje prirodnu evoluciju sa rešavanjem problema

Polje evolutivnog računarstva obuhvata mnoštvo tehnika i metoda bazirane na visoko apstraktnim biološkim modelima. Osnovni pravci ove oblasti su:

- 1) Genetsko programiranje
- 2) Evolutivno programiranje
- 3) Evolutivne strategije
- 4) Genetski algoritmi

2.2. Evolutivni ciklus

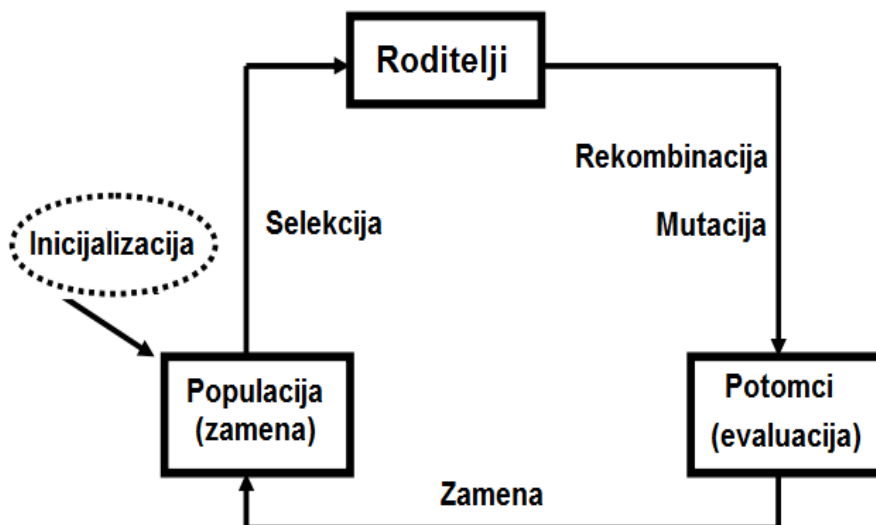
Slika 2.2 prikazuje uopšteni ciklus evolucije koji se koristi u genetskim algoritmima prilikom traženja potencijalnih rešenja. Prvi korak predstavlja kreiranje inicijalne populacije kandidata rešenja. Oni mogu biti potpuno nasumični, kreirani na osnovu ranijeg rešenja ili pak korišćenjem heuristika, tj. prilaza problemu koji pruža dovoljno dobre rezultate, kako bi se osigurale poželjne osobine.

Svakom članu populacije se dodeljuje mera prilagođenosti, što se obično predstavlja kao brojčana vrednost. Ovaj postupak obuhvata dekodiranje genetske predstave ili genotipa u problemsko rešenje – fenotip i provere njegove prilagođenosti kako bi se utvrdilo u kojoj meri odgovara konačnom rešenju.

Nakon toga selektuju se roditelji, pri čemu prednost imaju prilagođeniji članovi populacije, kako bi se dobili potomci korišćenjem genetskih operatora poput rekombinacije i mutacije.

Naslednici se potom evaluiraju kako bi se odredila njihova mera prilagođenosti i određeni broj njih će zauzeti mesto prethodnih članova populacije na osnovu šeme zamene (uobičajno se zamenjuju najmanje prilagođeni članovi ili čitave prethodne generacije).

Ciklus se ponavlja sve dok se ne ispuni neki od kriterijuma zaustavljanja kao što je pronalazak dovoljno kvalitetnog rešenja ili dostizanje određenog broja generacija. Najveći deo evolutivnog ciklusa uključuje slučajne ili stohastičke procese koji su neophodni za uspeh ovog metoda [4].



Slika 2.2 Prikaz uopštenog ciklusa evolucije

2.3. Biološki aspekt

Sudeći po Darvinovoj teoriji prirodne evolucije, evolutivne promene dolaze zbog različitih osobina koje životne forme imaju, a koje se takođe mogu i naslediti u budućim generacijama. Individue koje prežive zahvaljujući dobroj kombinaciji naslednih osobina, prenose ih u sledeću generaciju. Drugim rečima, najprilagođeniji preživljavaju kako bi preneli karakteristike koje su ih učinile visoko adaptiranim na svoje potomke.

Darvin u svoje vreme nije znao mnogo o razlogu i načinu nastajanja tih varijacija osobina, te je moderna genetika dala odgovor na pitanje kako prirodna selekcija deluje na stvaranje različitosti unutar populacije i kakvu ulogu u svemu tome imaju geni, kao osnovne jedinice nasleđivanja.

2.3.1. Geni

Gen predstavlja fizičku i funkcionalnu jedinicu nasleđivanja koja prenosi naslednu poruku iz generacije u generaciju.

Svako ljudsko biće se sastoji od nekoliko milijardi sitnih ćelija koje zajedno formiraju tkiva organizma. Svaka ćelija ima jezgro u kome se, po gruboj proceni, nalazi oko 30.000 različitih gena koji kontrolišu većinu osobina i mehanizama u organizmu. Primera radi, geni mogu određivati boju očiju, visinu, bujnost kose, itd. Sa druge strane, oni mogu biti odgovorni za varenje određene hrane i za funkcionisanje nervnih ćelija u mozgu. Svi geni, kao celina, formiraju jedinstveni individualni otisak svake jedinke.

Neki od značajnih termina vezani za oblast genetike su:

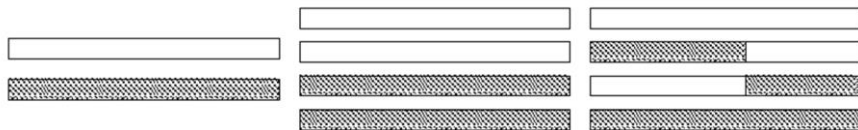
- 1) **Hromozomi** – skup gena (i još po nečega kao što su regulatorni elemenati i nukleotidne sekvence). Predstavljaju niti DNK u kojima su geni enkodirani.
- 2) **Lokus** – fiksna pozicija na hromozomu na kojoj može da se nađe jedan ili više gena (u slučaju jednostavnog genetskog algoritma to je jedan gen).
- 3) **Genom** – čini ga skup svih gena koji definišu vrstu i predstavlja kompoziciju više hromozoma.
- 4) **Genotip** – informacija o načinu izgradnje organizma, tj. njegova genetska konstitucija. U toku reprodukcije, on biva modifikovan od strane genetskih operatora, kao što su mutacija i rekombinacija.
- 5) **Fenotip** – bilo koja karakteristika organizma koju je moguće posmatrati (razvoj ili ponašanje) ili vrednost te karakteristike. Rezultat je genotipa i uticaja okruženja ili interakcije genotipa i okruženja.

Prirodna selekcija po principu “preživljavanja najboljih” odvija se nad fenotipom. Fenotipske osobine organizma (u koje spadaju ponašanje i fizičke razlike koje utiču na reakciju jedinke na okruženje), delom su određene nasleđem, delom faktorima u toku razvoja, a delimično i kao rezultat slučajnih promena koje su jedinstvene za svaku jedinku. Iz tog razloga, centralni stav koji važi u molekularnoj genetici je da genotip određuje fenotip, ali obrnuto ne važi. Razlog tome je činjenica da se stečene osobine ne mogu naslediti.

2.3.2. Reprodukcijska

Ljudska DNK je organizovana u hromosome. Sve ćelije ljudskog tela, osim reproduktivnih, sadrže 23 para hromozoma koji definišu fizičke attribute pojedinca. Gamete su specijalne reproduktivne ćelije koje nastaju posebnom vrstom ćelijske deobe – mejoze.

Dok normalne ćelije sadrže 46 hromozoma ili 23 para, dok gamete sadrže samo 23 pojedinačna hromozoma. Stvaranje potomka podrazumeva sjedinjavanje dve gamete. U toku spajanja dve ćelije ove vrste u jednu ćeliju potomka, javlja se genetsko ukrštanje pri čemu rezultujuća ćelija dobija 23 hromozoma oba roditelja, što ukupno čini 23 para. Hromozomi roditelja se ujedinjuju i dupliraju. Unutrašnji parovi se povezuju i međusobno razmenjuju delove, kao na slici 2.3. Proizvod ovoga procesa su jedna kopija majčinskog i očevog hromozoma i dve potpuno nove kombinacije nastale ukrštanjem [5].



Slika 2.3 Proces ukrštanja parova hromozoma u toku mejoze

Novonastala jedinka, koja se naziva i zigot, ubrzano se razvija formirajući mnoštvo ćelija sa istim genetskim sadržajem.

Ipak, s vremena na vreme se nešto od genetskog materijala malo izmeni u toku ovog procesa. Ta pojava se naziva mutacija i predstavlja grešku replikacije. To znači da potomci mogu da imaju genetski materijal koji nije nasleđen od roditelja, a to može da bude:

- 1) Katastrofalno – potomak nije sposoban za život.
- 2) Neutralno – nova osobina ne utiče na prilagođenost jedinke.
- 3) Prednost – nova osobina poboljšava prilagođenost u okruženju.

U kontekstu genetskih algoritama, mutacija i rekombinacija predstavljaju genetske operatore koji su zaduženi da vode populaciju u pravcu rešenja problema. U saradnji sa selekcijom, ovi operatori se koriste za odabir kvalitetnijih članova populacije, kombinaciju njihovih sadržaja i unošenje genetske raznolikosti.

2.4. Genetski algoritmi

Ideja o pretragama u skupu kandidata, kako bi se pronašlo željeno rešenje, je toliko česta u računarskim naukama da je dobila i svoje ime: pretraga u „prostoru pretraživanja“. Ovde se taj termin odnosi na kolekciju potencijalnih rešenja problema pri čemu postoji određena „udaljenost“ između kandidata.

Genetski algoritmi simuliraju preživljavanje prilagođenijih jedinki u tom prostoru u toku više generacija sve dok se ne postigne zadovoljavajući rezultat ili se ispuni terminacioni uslov (broj proteklih generacija ili nedostatak napretka posle određenog broja iteracija). Takođe, u kontekstu genetskih algoritama važi:

- 1) Okruženje definiše funkciju prilagođenosti individua u populaciji, koja se zamenjuje nakon svake generacije.
- 2) Problem se predstavlja kao string (klasično kao binarni ali postoje i druge vrste reprezentacija) i to je jedinka, odnosno hromozom.
- 3) Hromozom predstavlja genotip ili genetsku konstituciju jedinke.
- 4) Svaka pozicija na hromozomu je lokus ili gen.

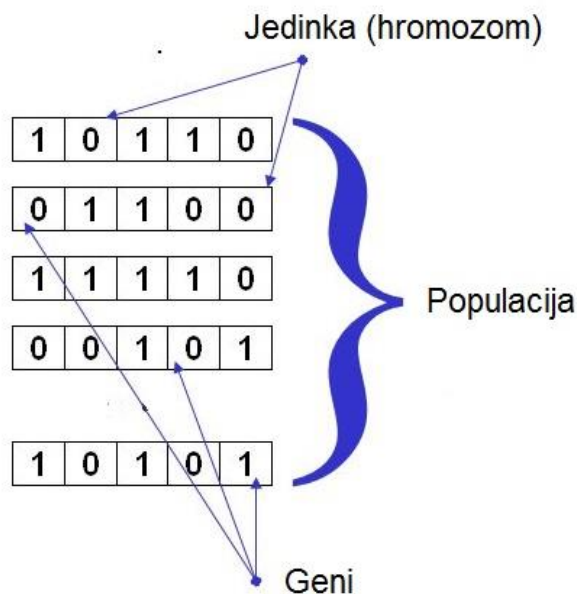
Genetski algoritam	Priroda
Optimizacioni problem	Okruženje
Dopustiva rešenja	Individue koje žive u okruženju
Funkcija prilagođenosti	Stepen adaptacije individue na okruženje
Skup dopustivih rešenja	Populacija organizama
Stohastički genetski operatori	Selekcija, ukrštanje i mutacija u procesu prirodne evolucije
Iterativna primena skupa stohastičkih operatora na skup dopustivih rešenja	Evolucija populacije sa ciljem prilagođavanja okruženju

Tabela 2.1 Metafora genetskog algoritma i prirode

Kvalitet nekog rešenja određuje njegov budući uticaj na celokupan proces pretrage. Dobra rešenja se koriste za generisanje novih, koja bi takođe trebala biti dobra, ako ne i bolja od prethodnih. Prema tome, populacija u svakom trenutku čuva sve što je naučeno o rešenju u svakoj tački.

2.4.1. Prostor pretrage

Kod genetskih algoritama populacija individua se održava unutar prostora pretrage, pri čemu svaka od njih predstavlja potencijalno rešenje problema. Jedinke su kodirane kao konačan vektor bita, realnih brojeva, komponenti, promenljivih, ili nekih drugih struktura podataka. U svetlu analogije sa genetikom, te životne forme su uporedive sa hromozomima, dok su strukture podataka korištene u kodiranju jednake genima. Ukratko, hromozom (potencijalno rešenje) je sačinjen od nekolicine gena (struktura podataka).



Slika 2.4 Odnos gena i hromozoma u kontekstu genetskih algoritama

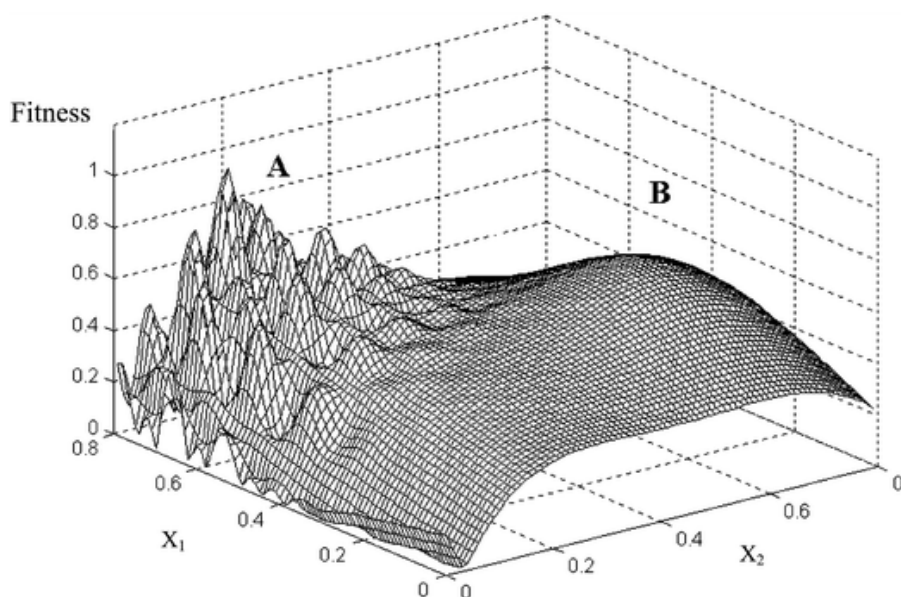
Svaka tačka u prostoru pretraživanja je jedno moguće rešenje, koje se može označiti na osnovu njegove mere prilagođenosti (fitness). Imajući to u

vidu, potraga za rešenjem znači potragu za nekim ekstremom (minimumom ili maksimumom) toga područja.

Genetski algoritam održava populaciju konstantnog broja jedinki sa dodeljenim vrednostima prilagođenosti. Smena generacija sa sobom donosi i nova moguća rešenja, koja u proseku sadrže više dobrih gena od tipične jedinke iz skupa roditelja.

Svaka naredna generacija će sadržati više dobrih “parcijalnih rešenja” od prethodne. Na kraju, kada čitava populacija konvergira ka određenom rezultatu i ne dolazi do proizvodnje potomaka primetno većeg kvaliteta od prethodnika, tada se može reći da je dovoljno kvalitetno rešenje pronađeno i proces pretrage prestaje.

Ukoliko bi se populacija individua prikazala kao “oblak tačaka” u prostoru, on bi se kretao njime sa svakom novom iteracijom, vođen procesom evolucije, u pravcu rešenja. Na slici 2.5 prikazana je jedna generacija kandidata. Vertikalna osa označava vrednost prilagođenosti, a ostale dve predstavljaju vrednosti drugih osobina jedinki.



Slika 2.5 Prstor pretraživanja u kojem moguća rešenja poseduju dve osobine

Prostor pretrage (ili adaptivni pejzaž) može biti potpuno poznat u vreme rešavanja problema, ali to obično nije slučaj. U većem broju situacija poznate su samo neke tačke u prostoru, dok se ostale generišu naknadno, kako se proces traganja nastavlja.

Problem kod “adaptivnih pejzaža” je u tome što potraga u njemu može biti veoma komplikovana pošto se ne zna gde početi i u kom pravcu pretraživati.

Izuzev genetskih algoritama, metode koje se još primenjuju u tim slučajevima su: Penjanje uzbrdo (hill climbing), tabu pretraga i simulirana preklapanja (simulated annealing). Rešenja nađena ovim tehnikama se smatraju dobrim iz razloga što je često nemoguće dokazati šta je stvarni optimum.

2.4.2. Genetski operatori

Varijacije osobina životnih formi su neophodne u procesu evolucije. Genetski operatori korišteni u ovim algoritmima omogućavaju te promene i analogni su onima u prirodi:

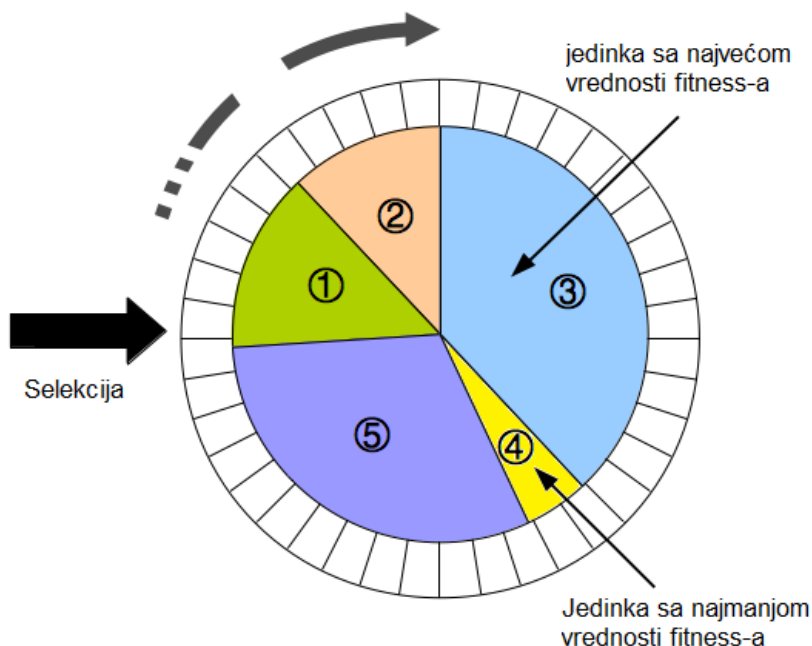
- 1) Selekcija ili preživljavanje prilagođenijih
- 2) Reprodukција ili ukrštanje
- 3) Mutacija

Ove tri vrste operatora moraju da rade u međusobnoj sprezi kako bi algoritam bio uspešan. Mutacija obezbeđuje raznovrsnost u populaciji slučajnim promenama gena unutar hromozoma, dok se selekcijom i ukrštanjem postojećih rešenja omogućava vođenje pretrage u novim pravcima.

2.4.3. Selekcija

Ovim postupkom se odabiru hromozomi za dalju reprodukciju, preferirajući pri tome prilagođenije pojedince. Na taj način algoritam se usmerava ka perspektivnim regionima prostora pretraživanja. Ukoliko je vrednost fitness-a veća, veće su i šanse da više puta jedinka bude odabrana u procesu ukrštanja.

Selekcija može biti deterministički proces, ali u većini slučajeva ona se implementira tako da sadrži i elemente slučajnosti. Jedan od takvih primera je selekcija ruletom.



Slika 2.6 Primer selekcije hromozoma metodom ruleta

Površina oblasti na ruletu koja se dodeljuje svakom hromozomu proporcionalna je vrednosti funkcije prilagođenosti u tački koju on reprezentuje.

Postupak vrtenja se ponavlja onoliko puta koliko je potrebno da bi se dobila roditeljska populacije iste veličine kao i prethodna. Verovatnoća izbora individue i u tom slučaju je:

$$P(i, \text{odabrano}) = \frac{f(i)}{\sum f(i)}$$

Dok je očekivani broj kopija jedinke i u toku selekcije:

$$E(n_i) = \mu * \frac{f(i)}{\langle f \rangle}$$

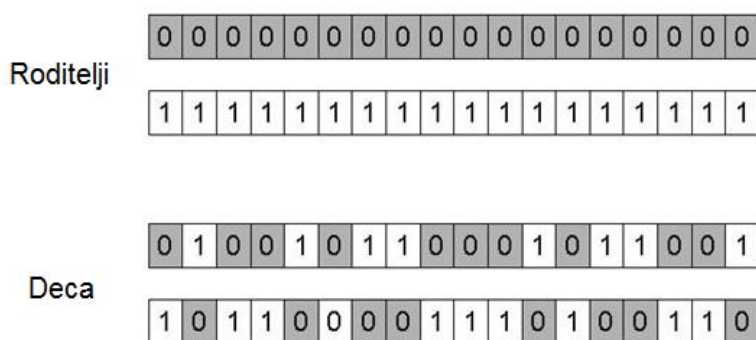
Pri tome, μ označava veličinu populacije, $f(i)$ prilagođenost individue i , a $\langle f \rangle$ srednju prilagođenost populacije.

Ova vrsta selekcije ne garantuje da će se prilagođeniji hromozomi uvek odabrati, već da imaju veću šansu da budu selektovani u odnosu na slabije konkurente, što ostavlja šansu i njima (iako manju) da se nađu u procesu rekombinacije, baš kao i u prirodi.

2.4.4. Ukrštanje

Nakon odabira dva potencijalna rešenja korišćenjem neke od metoda selekcije, dolazi do rekombinacije genetskog materijala između dva roditelja. Ta raspodela se obično dešava tako što se dva hromozoma presecaju na slučajnim mestima, a zatim spajaju, pri čemu se takođe na slučajan način određuje koje će gene dete naslediti od oca, a koje od majke.

Iako postoji nekoliko vrsta implementacija rekombinacije u okviru genetskih algoritama (uniformno, tačkasto, presecanje, itd.) ovde će biti razmotreno uniformno ukrštanje s obzirom da je ono korišteno u radu.



Slika 2.7 Rezultat rekombinacije usled uniformnog ukrštanja

Za svaku poziciju unutar hromozoma na slučajan način se određuje da li će elementi biti zamenjeni. To znači da se uparivanje izvodi pomoću slučajno generisane maske koja govori koje gene će potomak preuzeti od prvog, a koje od drugog roditelja. Drugo dete će u tom slučaju biti invertovana kopija prvog deteta.

Da li će doći do nastanka potomaka zavisi i od verovatnoće ukrštanja. Ukoliko je vrednost ovog parametra maksimalna, selektovi hromozomi će uvek proizvoditi naslednike sa zajedničkim genetskim sadržajem za sledeću

generaciju. Sa druge strane, manja verovatnoća ukrštanja ostavlja mogućnost da se rekombinacija ne desi, već se u narednu generaciju prenose identične kopije roditelja.

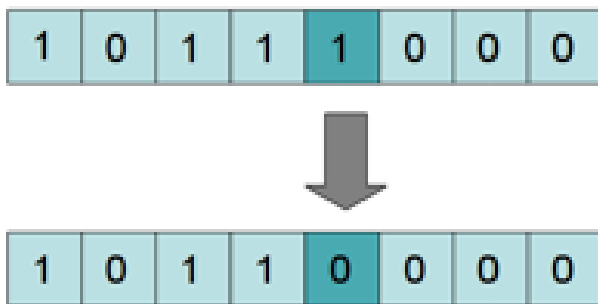
2.4.5. Mutacija

Cilj ovog genetskog operatora je da simulira efekte grešaka koji se dešavaju (sa malom verovatnoćom) u toku reprodukcije. Primenjuje se na naslednike, odmah nakon završetka procesa ukrštanja, a pre određivanja njihove prilagođenosti. Rezultati mutiranja, pored kretanja kroz prostor pretraživanja, mogu se videti i u restauraciji izgubljenih osobina populacije.

Jednostavna mutacija podrazumeva da svaki gen jedinke ima istu verovatnoću da bude mutiran. Ta verovatnoća se predstavlja kao parametar mutacije p_m . Njegova vrednost se obično kreće između recipročne vrednosti veličine populacije i recipročne vrednosti dužine hromozoma.

$$p_m \in \left[\frac{1}{vel_pop}, \frac{1}{vel_hrom} \right]$$

Selektovani gen tada menja vrednost u zavisnosti od načina na koji se vršilo kodiranje. Ukoliko je individua kodirana kao string bita, vrednost elementa na toj poziciji se menja sa 0 na 1 i obrnuto, dok se kod ostalih načina vrednost postavlja slučajnim izborom u opsegu dopustivosti rešenja.



Slika 2.8 Jednostavna mutacija gena binarno kodirane jedinke

Ovim promenama završavaju se izmene na hromozomima, što znači da je sledeći korak procena prilagođenosti upotrebom funkcije prilagođenosti.

2.4.6. Evaluacija rešenja

Funkcija prilagođenosti zavisi od problema koji se posmatra, ali u svakom slučaju, to je funkcija koja kao ulaz uzima jedinku a kao izlaz vraća realan broj [6].

Računanje vrednosti prilagođenosti članova populacije se ponavlja često u toku izvršavanja algoritma tako da bi ono trebalo da bude što brže. Sporo izračunavanje fitness-a bi moglo negativno da se odrazi na genetski algoritam i učini ga nedopustivo sporim.

U većini slučajeva funkcija prilagođenosti i funkcija koja je predmet zadatka su iste, zbog toga što je zadatak određivanje maksimuma ili minimuma zadate funkcije. Ipak, za kompleksnije probleme koji sadrže više ograničenja ili predmeta rešavanja nekada je potrebno uzeti drugu funkciju za ocenjivanje adaptivnosti potencijalnih rešenja. Ona pri tome treba da ispunjava dva osnovna uslova:

- 1) Njeno izračunavanje bi trebalo biti dovoljno brzo.
- 2) Mora kvantitativno da izrazi koliko su data rešenja kvalitetna.

U nekim slučajevima može doći do nemogućnosti računanja evaluaciona funkcije direkto zbog nasleđenih složenosti problema koji se posmatra. U tim situacijama, radi se aproksimacija prilagođenosti kako bi se odgovorilo zahtevima problema.

Mnogo istraživanja je sprovedeno poslednjih godina na temu klasifikovanja vrsta funkcija koje bi trebale biti lakše za genetske algoritme pri optimizaciji i onih koje bi bile teže za rad sa njima. John Holand i Melanie Mitchell su definisali takozvani „kraljevski put” funkcija koje su definisane na takav način da su parametri problema x_1, x_2, \dots, x_n susedno kodirani i evaluaciona funkcija f je suma od n broja funkcija za svaki parametar, što znači:

$$f(x_1, x_2, \dots, x_n) = f_1(x_1) + f_2(x_2) + \dots + f_n(x_n)$$

Sa druge strane, funkcije koje „sakrivaju“ optimum od genetskog algoritma, David Goldberg i ostali autori nazivaju obmanjivačkim. One navode algoritam na lažne tragove, što u tim slučajevima znači da se pronalazak optimuma može postići samo uz mnogo sreće [7].

Rešenje problema je dobro onoliko koliko je dobra i funkcija prilagođenosti. Izbor dobre evaluacione funkcije često predstavlja najteži zadatak, pre svega zbog toga što je izuzetno zavisna od problema koji se rešava i ne postoji generalno pravilo prilikom njenog izbora.

2.4.7. Koraci algoritma

Nakon razumevanja problema, definisanja funkcije prilagođenosti i odabira načina na koji će potencijalna rešenja biti predstavljena, jednostavni genetski algoritam se implementira sledećim redosledom operacija:

- 1) Kreiranje inicijalne populacije kandidata. Veličina populacije je unapred određena, kao i dužina hromozoma kojima su jedinke predstavljene.
- 2) Određuje se prilagođenost svakog hromozoma uz pomoć funkcije prilagođenosti.
- 3) Sledeći koraci se ponavljaju sve dok se ne stvori onoliko potomaka kolika je veličina početne populacije:
 - a) Odabiru se dva roditeljska hromozoma iz trenutne generacije nekom od metoda selekcije, pri čemu iste jedinke mogu biti selektovane više puta, ali ne istovremeno u procesu ukrštanja.
 - b) Izvodi se ukrštanje sa određenom verovatnoćom (šansa da se uparivanje dogodi) nekom od metoda rekombinacije, proizvodeći pri tome dva naslednika. U slučaju da se ukrštanje ne dogodi, kreiraju se dva potomka identičnog genetskog sadržaja kao roditelji (kopije).
 - c) Mutiraju se oba deteta na svakom lokusu sa određenom verovatnoćom mutacije i stavljaju se na odgovarajuća mesta u novoj populaciji. U

slučajevima kada je veličina populacije neparan broj, jedan od potomaka se nasumičnim odabirom briše.

- d) Zamenjuje se trenutna populacija sa novonastalom.
- e) Ponovo se određuje prilagođenost svake jedinke.

Svako ponavljanje ovog procesa se naziva iteracija. U praksi, genetski algoritmi se iteriraju u opsegu između 50 i 500 ili više ponavljanja. Na kraju svih iteracija obično se pronalazi jedan ili više visoko prilagođenih hromozoma.

Dobrom praksom se smatra ponavljanje celokupnog postupka pronalaženja rešenja nekoliko puta pre donošenja konačnog zaključka. Razlog tome je što slučajnost igra znatnu ulogu u svim iteracijama, tako da različite početne generacije mogu da dovedu do drugačijeg ponašanja algoritma i generalno, u manjoj ili većoj meri, drugačijih konačnih rešenja.

2.4.8. Igra Mastermind

Mastermind ili skočko, je igra dekodiranja namenjena za dva igrača. Današnju verziju igre je patentirano 1970. godine Mordecai Meiowitz, Izraelski ekspert za telekomunikacije.

Nakon što je bio odbijen od strane vodećih kompanija za proizvodnju igara, privukao je pažnju kompanije Invicta Plastics, koja je restilizovala i preimenovala igru. Puštajući je u masovnu prodaju 1971. godine, prodana je u više od 50 miliona primeraka u 80 zemalja, učinivši je najprodavanijom novom igrom 70-ih godina prošlog veka. Pravila igre su sledeća:

Jedan igrač je zadužen za kodiranje a drugi za dekodiranje rešenja koje se traži. Kod se sastoji od četiri ili više uzastopnih znakova, pri čemu mogu postojati duplikati. Kodirano rešenje je sakriveno od igrača koji ga pokušava pronaći, a vidljivo igraču koji ga je kreirao.

Cilj je pronalazak šablona, po tipu znaka i redosledu, u ograničenom broju pokušaja. Igrač koji pokušava postavlja kombinaciju za koju misli da je tačna na tablu, a drugi igrač mu daje povratnu informaciju o tome koliko je blizu konačnog rešenja, time što pored odigrane kombinacije stavlja male krugove. Crveni krug označava da je neki od simbola tačan i nalazi se na pravom mestu. Žuti pak predstavlja pravi znak na pogrešnom mestu, dok je beli namenjen za pogrešne znakove. Na slici 2.9 prikazana je originalna tabla igre.

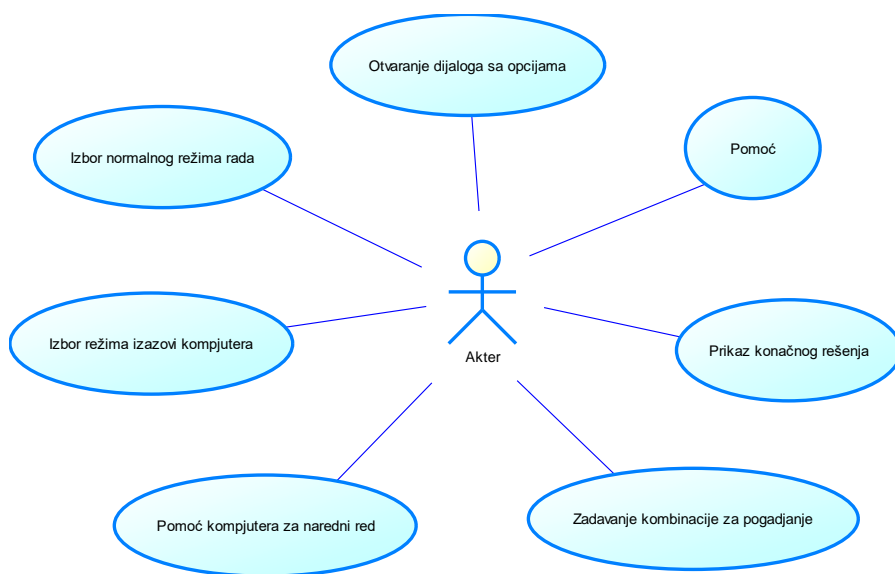
Posle povratne informacije, sledi novi pokušaj, sve dok se kodirano rešenje ne pronade ili se dostigne maksimalan broj pokušaja. Igrač koji je zadao kombinaciju dobija poen za svaki red koji je protivniku bio potreban na putu traženja. Na kraju, pobednik je igrač sa više osvojenih poena.



Slika 2.9 Originalna tabla za igru Mastermind

3. SPECIFIKACIJA I IMPLEMENTACIJA SOFTVERA

Arhitektura softvera predstavlja opis podsistema i komponenti softverskog sistema zajedno sa njihovim međusobnim vezama. Podsystemi i komponente mogu biti specificirane iz više uglova sa ciljem ilustriranja funkcionalnih i nefunkcionalnih osobina softverskog sistema. Softverska arhitektura sistema predstavlja proizvod koji je posledica aktivnosti projektovanja softvera [9].



Slika 3.1 Dijagram slučajeva korišćenja aplikacije

U ovom poglavlju je opisana arhitektura same aplikacije, tehnologije izrade, kao i procesa koji se odvijaju u toku izvršenja. Namena ovog rešenja je da demonstrira upotrebu genetskih algoritama u što bržem rešavanju igre Mastermind, kao i da prikaže napredak u procesu pretrage kroz svaki korak. Na slici 3.1 je predstavljen dijagram slučajeva korišćenja sa osnovnim funkcionalnostima, o kojima će biti nešto više reči nastavku

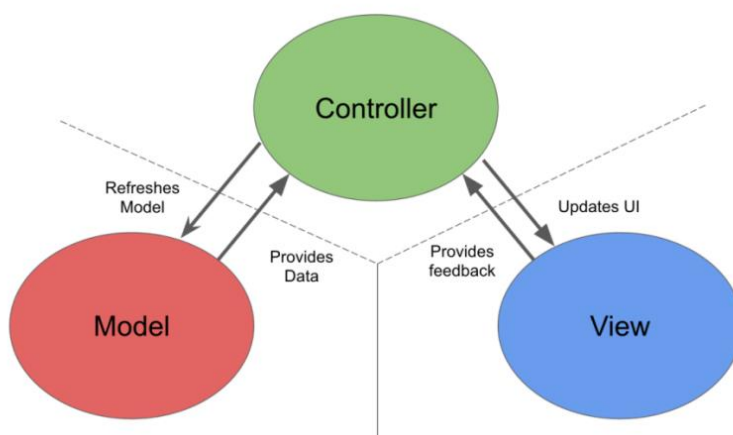
3.1. Arhitektonski šablon

Prezentovano softversko rešenje je implementirano u programskom jeziku C# korišćenjem .Net radnog okvira, baziranog na Windows Forms platformi sa osloncem na MVC (Model-View-Controller) šablon.

Windows forms kao platforma donosi čist, objektno orijantisan set klasa koje omogućavaju bogat razvoj windows aplikacija. Takođe, može da se ponaša kao lokani korisnički interfejs u rešenjima koja su sadržana u više nivoa. Povezivanje podataka u ovoj platformi omogućava prezentovanje i izmenu informacija sa njihovog izvora putem kontrola koje se nalaze na formi. Na ovaj način moguće je rukovati i povezati, kako tradicionalne izvore podataka, tako i bilo koju drugu strukturu koja sadrži podatke.

Model-View-Controller šablon je arhitektonski šablon koji razdvaja model podataka aplikacije i korisnički interfejs u odvojene komponente.

Model podataka je grupa struktura podataka koja čini osnovu za poslovnu logiku softverskog rešenja. Kod tipičnih objektno-orijentisanih aplikacija model podataka je sačinjen od klasa, kolekcija, njihovih međusobnih odnosa, kao i odnosa prema osobinama stvarnog sveta.



Slika 3.2 Odnos komponent MVC šablona

View ili pogled predstavlja element korisničkog interfejsa koji prezentuje podatke i dozvoljava korisniku da ih izmeni. Drugim rečima, pogled je vizuelna predstava svog modela i neretko se dešava da je nekoliko pogleda aktivno u isto vreme.

Kontroler je poveznik između dve prethodno navedene komponente. Preuzima zahteve, koji uglavnom dolaze od pogleda i vraća odgovor na te zahteve. Odgovoran je za kontrolisanje toka izvršenja aplikacije i sadrži jednu ili više akcija koje mogu da vrate različite tipove rezultata dotičnom zahtevu.

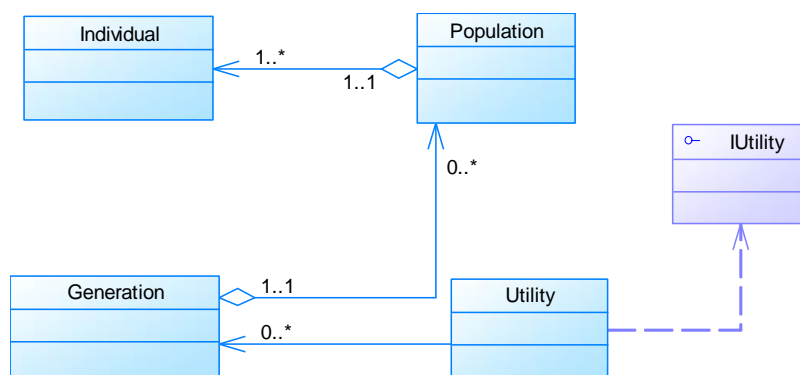
Ovaj arhitektonski šablon je primenjen u prezentovanom rešenju kako bi se povećala modularnost, olakšalo održavanje i razdvojila logika pogleda od poslovne logike aplikacije.

3.2. Model

Model predstavlja prezentaciju ili apstrakciju realnosti. On je najčešće pojednostavljeni prikaz realnosti, s obzirom da je ona suviše kompleksna da bi se tačno opisala. Pri rešavanju određenog problema, veći deo kompleksnosti u suštini je irelevantan. Pri izradi modela, stepen pojednostavljena i apstrakcije zavisiće od ugla iz koga se realnost posmatra i potrebnog nivoa detaljnosti [9].

Prilikom dizajniranja sistema korišten je UML jezik za modelovanje, čiji grafički elementi omogućavaju da se na razumljiv način opišu: klase, komponente, čvorevi, aktivnosti, stanja, kao i da se modeluju relacije između tih elemenata.

Elementi modela predstavljeni su klasama *Individual*, *Population*, *Generation* i *Utility*. Na slici 3.3 prikazan je dijagram klasa koje predstavljaju elemente modela.



Slika 3.3 Dijagram klasa modela podataka.

3.2.1. Klasa Individual

Ova klasa predstavlja jednog člana populacije, odnosno potencijalno rešenje problema. Sadrži informacije o individui kao i metode za njihovo podešavanje i inicijalizaciju.

Individual			
- genes	: Array<int>		
- fitness	: Decimal		
- numberOfSignsInCombination	: int	= 4	
+ Individual ()	: Individual		
+ InitializeIndividual ()	: void		
+ Fitness ()	: Decimal		
+ NumberOfSignsInCombination ()	: int		

Slika 3.4 Klasa Individual

Kodiranje jedinke je implementirano kao niz celih realnih brojeva, pri čemu svaki broj predstavlja tačno jedan znak kao što je prikazano na slici 3.5



Slika 3.5 Kodiranje znakova iz igre upotrebom brojeva

Ovaj način kodiranja individua je odabran kako bi se očuvala dopustivost rešenja nakon primene genetskih operatora nad populacijom, kao i da bi njena veličina ostala nepromenjena kroz svaku generaciju.



Slika 3.6 Primer kodirane jedinke

3.2.2. Klasa Population

Ovaj element modela opisuje populaciju kandidata. Sadži informacije o opštoj prilagođenosti, kao i o prilagođenosti najboljeg pripadnika. Ovde su sadržani genetski operatori (selekcija, mutacija i rekombinacija) koji se koriste pri vođenju pretrage u vidu metoda, kao i metode za dobavljanje i podešavanje informacija o populaciji.

Population		
- populationSize	: int	= 500
- individuals	: Array<Individual>	
- numberOfSignsInCombination	: int	= 4
- averageFitness	: Decimal	
- fitnessOfBestIndividual	: Decimal	
+ Population (int populationSize)	: Population	
+ InitializeIndividuals ()	: void	
+ PopulationSize ()	: int	
+ FitnessOfBestIndividual ()	: Decimal	
+ AverageFitness ()	: Decimal	
+ SumOfFitness ()	: Decimal	
+ FindBestFitedIndividual ()	: Individual	
+ RulettSelection (Decimal sumOfFitness)	: int	
+ SimpleCrossover (Individual parent1, Individual parent2, Individual child1, Individual child2)	: void	
+ UniformCrossover (Individual parent1, Individual parent2, Individual child1, Individual child2)	: void	
+ Mutation (Decimal mutationChance, Individual individual)	: void	

Slika 3.7 Population klasa

3.2.3. Klasa Generation

Ova klasa u sebi sadrži parametre potrebne za vođenje genetskog algoritma, informacije o populacijama roditelja i naslednika, kao i metode za manipulisanje tim podacima.

Metoda *FindBestIndividualForNextRow* opisuje čitav tok genetskog algoritma u unapred određenom broju ponavljanja i vraća potencijalno rešenje za sledeći red igre.

Generation		
-	numberOfIterations	: int
-	mutationThreshold	: Decimal
-	crossoverThreshold	: Decimal
-	numberOfRowsForGuessing	: int
-	currentRow	: int
-	parents	: Population
-	children	: Population
-	pastPopulations	: Array<Population>
+	CalculateFitnessOfIndividual (int currentRow, Individual individual)	: void
+	Generation ()	: Generation
+	InitializeGeneration ()	: void
+	OneIteration ()	: void
+	SaveBestIndividualFromThisIteration ()	: Individual
+	FindBestIndividualForNextRow ()	: void
+	UpdateAverageFitnessOfAllPopulations ()	: void
+	CheckIfSolutionIsFound ()	: bool
+	NumberOfIterations ()	: int
+	FitnessOfBestIndividual ()	: Decimal
+	AverageFitness ()	: Decimal
+	NumberOfRowsForGuessing ()	: int
+	CurrentRow ()	: int

Slika 3.8 Klasa Generation

3.2.4. Klasa Utility

Utility implementira interfejs `IUtility` i predstavlja poveznicu između ostalih elemenata modela i kontrolera. Pozivom njegovih metoda dobivljaju se i podešavaju podaci sadržani u prethodno opisanim klasama.

Sadrži informacije koje se tiču same igre i po potrebi ih prosleđuje ostalim elementima modela u kojima je sadržana sva funkcionalnost genetskog algoritma.

Sve operacije ove klase su navedene u interfejsu `IUtility`. Kontroler u svom konstruktoru uzima instancu klase koja implementira taj interfejs i time se povećava otpornost na eventualne izmene.

Utility	
- ordinalNumberOfSign	: int
- generation	: Generation
- solutionFound	: Boolean
- mainSolution	: Array<int>
+ Initialization ()	: void
+ Utility ()	: Utility
+ SetNumberOfSignsInCombination (int sign)	: void
+ GetNumberOfSignsInCombination ()	: int
+ AddSignToSolution (int sign)	: void
+ OrdinalNumberOfSign ()	: int
+ CurrentRow ()	: int
+ MainSolution ()	: Array<int>
+ SetNumberOfIterations (int iterations)	: void
+ GetNumberOfIterations ()	: int
+ SetNumberOfRowsForGuessing (int rows)	: void
+ GetNumberOfRowsForGuessing ()	: int
+ SetPopulationSize (int size)	: int
+ GetPopulationSize ()	: int
+ CheckCurrentCombinationWithMainSolution ()	: void
+ RemoveSignFromSolution ()	: void
+ CheckIfSolutionIsFound ()	: void
+ UpdateAverageFitnessOfAllPopulations ()	: void
+ GetAllPopulations ()	: Array<Population>

3.9 Klasa Utility

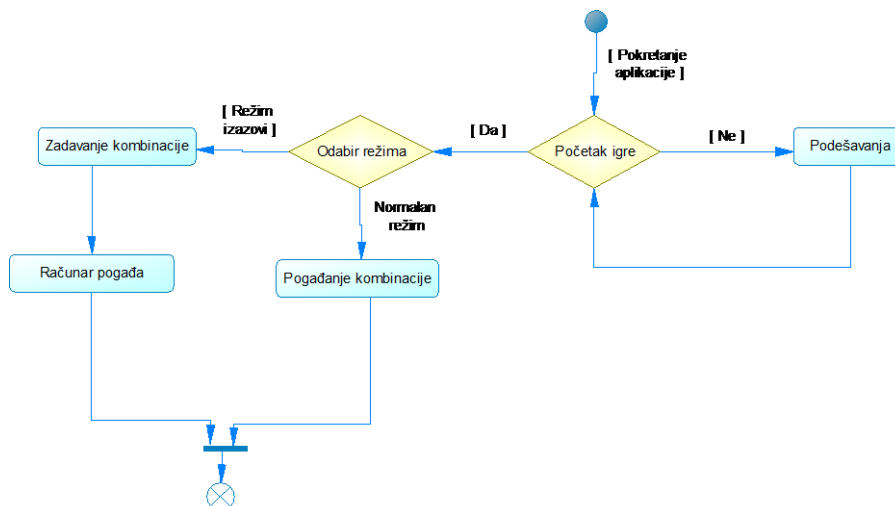
3.3. Dijagram aktivnosti aplikacije

Dijagrami aktivnosti se obično koriste za modelovanje poslovnih procesa ili modelovanje detaljne logike poslovnih pravila. Na mnoge načine ova vrsta dijagrama predstavlja objektno orijentisani ekvivalent dijagrama toka podataka u strukturnom razvoju [11].

Nakon pokretanja aplikacije korisnik bira da li želi da otvori dijalog sa podešavanjima, gde može da podesi parametre genetskog algoritma kao i dozvoljeni broj pokušaja pogađanja i znakova u kombinaciji rešenja, ili da

pokrene igru. Ukoliko odabere igru, odlučuje da li želi da je pokrene u normalnom režimu ili u režimu izazovi kompjutera.

U normalnom režimu korisnik samostalno pogađa rešenje, s tim da može da iskoristi pomoć kompjutera za sledeć red. Režim izazova podrazumeva da igrač zadaje kombinaciju znakova koju će računar pokušati da pronađe upotrebom genetskog algoritma.



Slika 3.10 Dijagram aktivnosti

3.4. Korisnički interfejs

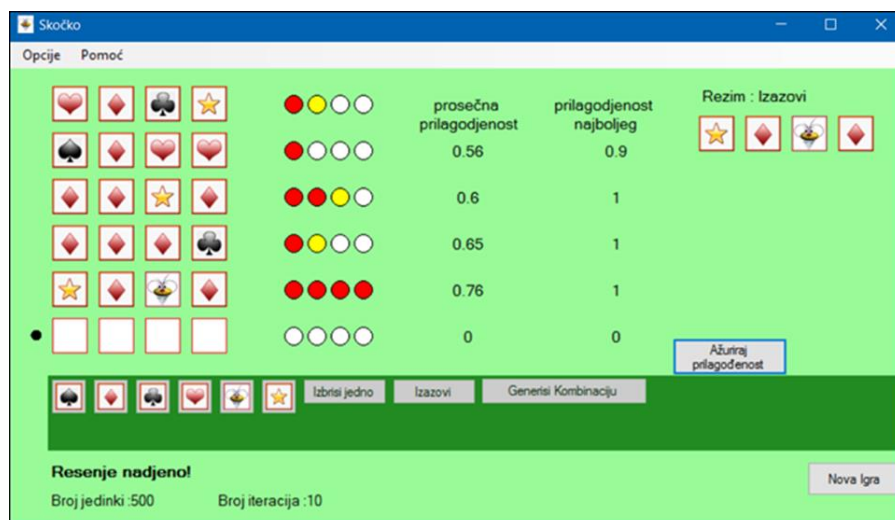
Prostor u kojem se dešava interakcija između čoveka i računara naziva se korisnički interfejs. Cilj njegovog dizajniranja je da omogući lako, efikasno i prijateljski naklonjeno okruženje iz kojeg bi se upravljalo računarom i postigli željeni rezultati uz minimum napora [12].

Osnovni elementi ovoga interfejsa su implementirani u vidu korisničkih kontrola (user controls), koje se nalaze na glavnom prozoru aplikacije.

Korisničke kontrole predstavljaju podesivu grupu kontrola i ponašanja koje se grupišu kako bi se naknadno koristile u istom sastavu.

Klase koje predstavljaju implementaciju ovih elemenata su: *TablePanel*, *CombinationPanel*, *FeedbackPanel*, *FitnessPanel* i *ChoicePanel*.

Na slici 3.11 prikazan je glavni prozor aplikacije zajedno sa navedenim komponentama.



Slika 3.11 Glavna forma aplikacije

3.4.1. Komponenta TablePanel

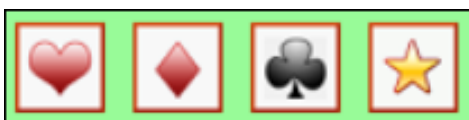
Ova komponenta označava tablu na kojoj se igra odvija. U njoj su grupisane korisničke kontrole *CombinationPanel*, *FeedbackPanel* i *FitnessPanel* u više redova, čiji broj zavisi od broja redova za pogađanje koji je odabran u opcijama igre. Metode sadržane u ovoj kontroli se odnose na manipulaciju znakova i prikaz povratnih informacija o svakom pokušaju.

♥	♦	♣	★	● ● ● ●	prosečna prilagodjenost	prilagodjenost najboljeg
♠	♦	♥	♥	● ● ● ●	0.56	0.9
♦	♦	★	♦	● ● ● ●	0.6	1
♦	♦	♦	♣	● ● ● ●	0.65	1
★	♦	♣	♦	● ● ● ●	0.76	1

Slika 3.12 Izgled TablePanel komponente

3.4.2. Komponenta CombinationPanel

Predstavlja panel u koji se unose znakovi (ili uklanjaju prilikom brisanja). Kada je ispunjen do kraja vrši se provera sličnosti sa glavnim rešenjem koje je smešteno u panel iste vrste (ali na glavnoj formi aplikacije). Potpuno ispunjena polja predstavljaju jedinku i potencijalno rešenje.



Slika 3.13 CombinationPanel

3.4.3. Komponenta FeedbackPanel

Rezultati provere prethodnog pokušaja sa rešenjem su smešteni u ovoj kontroli. Njena grafička reprezentacija je prikazana na slici 3.14. Slično kao kod kodiranja jedinke, kodiranje povratne informacije je izvršeno kao niz celih brojeva pri čemu: broj 0 označava beli krug (znak se ne nalzi u rešenju), 1 označava žuti krug (znak na pogršnom mestu), dok broj 2 označava crveni krug (znak na pravom mestu). Informacije dobijene ovim putem se koriste prilikom računanja prilagođenosti individua u genetskom algoritmu.



Slika 3.14 FeedbackPanel

3.4.4. Komponenta FitnessPanel

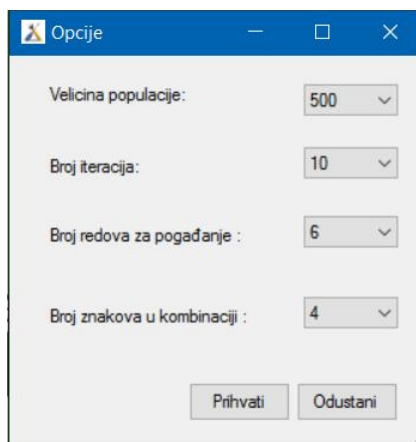
Ova korisnička kontrola prikazuje prosečnu prilagođenost najbolje generacije iz svakoga pokušaja, kao i prilagođenost najbolje jedinke iz te populacije. Ovi podaci se dobavljaju u slučajevima kada su unešene kombinacije u *TablePanel-u* rezultat rada genetskog algoritma. Za prvu kombinaciju ovih podataka nema, pošto se ona generiše slučajno s obzirom da ne postoje informacije o njenoj prilagođenosti.

prosečna prilagodjenost	prilagodjenost najboljeg
0.56	0.9
0.6	1
0.65	1
0.76	1

Slika 3.15 FitnessPanel

3.4.5. Dijalog sa opcijama

Ovaj dijalog omogućava podešavanje parametara algoritma kao što su broj jedinki u populaciji i broj ponavljanja računa pre nego što se odabere najbolja jedinka koja će biti unešena u sledeći red *TablePanel-a*. Takođe, moguće je podesiti i broj znakova u kombinaciji koja se traži (početna vrednost je 4) i broj mogućih pokušaja nalaženja rešenja.



Slika 3.16 Dijalog sa opcijama

3.5. Implementacija genetskog algoritma

Osnovni problemi prilikom primene ove vrste algoritama na određeni prostor pretrage su:

- 1) Kako kodirati potencijalno rešenje na način koji će omogućiti laku primenu operacija genetskog algoritma a isto tako sačuvati značenje tog rešenja?
- 2) Na koji način odrediti prilagođenost svake individue u populaciji?

U primeru igre Mastermind ovo ne predstavlja problem iz razloga što sama igra daje odgovor na oba navedena pitanja. Kodiranje rešenja je izvršeno putem nizova celih brojeva gde svaki broj označava tačno jedan znak. Sa druge strane, pravila igre nalažu da nakon svakog pokušaja korisnik dobije povratnu informaciju o "blizini" rešenja, te se ti podaci koriste prilikom evaluacije svake individue i to na sledeći način:

Kako bi se izračunala vrednost fitness-a za hromozom H, poredimo ga sa svakim prethodnim pokušajem i računamo broj znakova na mestu, van mesta i broj nepostojećih znakova u kombinaciji koje bi hromozom H postigao da su ti pokušaji glavno rešenje. Razlika između tako dobijenih povratnih vrednosti i povratnih vrednosti koje su ti pokušaji dobili u odnosu na glavno rešenje predstavljaju kvalitet hromozoma H. Drugim rečima, rezultati poređenja prethodnih pokušaja sa glavnim rešenjem predstavljaju „odnos“ rešenja prema tim kombinacijama, tako da ako jedinka koja se ocenjuje ima isti ili sličan odnos prema prethodnim pokušajima, njena ocena je veća.

Nakon ocenjivanja svih članova populacije, vrši se selekcija metodom ruleta kako bi se odabrali roditeljski parovi za rekombinaciju gena. Prag ukrštanja je postavljen na vrednost 0.3 , što znači da postoji 30% šanse da dođe do zamene svakog pojedinačnog gena između roditelja prilikom kreiranja naslednika. Verovatnoća mutiranja svakog gena deteta je postavljena na 0.05 ili 5%.

Na kraju ciklusa dolazi do zamene populacije roditelja i naslednika. Ovaj proces se ponavlja onoliko puta koliko je dogovoreni broj iteracija. Najbolja inividua iz poslednje iteracije se koristi kao sledeći pokušaj. Listing 3.1 opisuje jednu iteraciju genetskog algoritma.

```

public void OneIteration()
{
    SaveBestIndividualFromThisIteration();
    double sumOfFitness = parents.SumOfFitness();
    int parent1;
    int parent2;

    for (int i = 1; i < parents.individuals.Length / 2;
i++)
    {
        parent1 = parents.RulettSelection(sumOfFitness);
        parent2 = parents.RulettSelection(sumOfFitness);

        while (parent1 == parent2)
        {
            parent2 =
parents.RulettSelection(sumOfFitness);
        }
        Individual A = new
Individual(parents.individuals[parent1]);
        Individual B = new
Individual(parents.individuals[parent2]);
        Individual a = new Individual();
        Individual b = new Individual();
        a.NumberOfSignsInCombination =
numberOfSignsInCombination;
        b.NumberOfSignsInCombination =
numberOfSignsInCombination;
        a.initializeIndividual();
        b.initializeIndividual();
        parents.UniformCrossover(A, B, a, b,
crossoverThreshold);
        children.Mutation(mutationThreshold, a);
        children.Mutation(mutationThreshold, b);

        children.individuals[i * 2] = a;
        children.individuals[i * 2 + 1] = b;

    }
    parents = children;
    children = new Population(populationSize);
}

```

Listing 3.1 Jedna iteracija genetskog algoritma

Metoda *SaveBestIndividualFromThisIteration()*, pored odabira najbolje jedinke, poziva i metodu za računanje fitness-a svih članova populacije koja je prikazana u listingu 3.2.

```
public void CalculateFitnessOfIndividual(int currentRow,
Individual individual)
{
    float fitness = 0;
    int similarly;
    int[] similarityWithPastCombinations;
    for (int i = 0; i < currentRow; i++)
    {
        similarityWithPastCombinations =
CompareIndividualWithPastCombination(individual.genes,
tableForGuessing[i]);
        similarly = CompareWithResoults(i,
similarityWithPastCombinations);
        fitness += ((float)similarly) /
numberOfSignsInCombination;
    }
    if (currentRow > 0)
        fitness = ((float)fitness) / currentRow;

    if (fitness == 0)
        fitness += .01f;
    individual.Fitness = fitness;
}
```

Listing 3.2 Metoda za ocenjivanje jedinki

4. VERIFIKACIJA REŠENJA

Merenje uspešnosti rada aplikacije vršeno je eksperimentalno na osnovu ukupnog broja pokušaja koji je potreban da se pronađe rešenje. Pri kombinaciji 4 znaka od 6 mogućih ($P=4, N=6$), algoritmu demonstriranom u ovom radu bilo je potrebno u proseku 4,51 pokušaja da bi pronašao rešenje.

U literaturi je pronađeno nekoliko ranijih implementacija genetskih algoritama primenjenih na isti problem pri istim parametrima i njihovi rezultati su dati u tabeli 4.1.

Algoritam	Prosečan broj pokušaja
Knuth	4.47
Irving	4.36
Norvig	4.47
Neuwirth	4.36
Koyama and Lai	4.34
Rosu	4.34
Kooi	4.37
Bestavros and Belal	3.86

Tabela 4.1 Prosečan broj pokušaja ostalih algoritama pri $P=4, N=6$

Iz navedenih rezultata se vidi da performanse ove aplikacije ne zaostaju mnogo u odnosu na druge algoritme, pri čemu je vreme izvršavanja kraće nego kod navedenih primera (oko 3 sekunde). Ipak, treba napomenuti da u veoma malom broju slučajeva dolazi do situacije kada rešenje nije nađeno čak ni u 8 ili više koraka. To se može objasniti time što pretraga ponekad kreće u pogrešnom pravcu jer je veliki broj članova početne populacije slično a “daleko” od rešenja. Ovaj problem se može rešiti povećavanjem broja jedinki u populaciji.

Povećavanjem broja iteracija sa inicijalnih 50 na 100 ili više, ne postižu se приметно bolji rezultati a vreme izvršavanja se povećava jer se

svaka kombinacija proverava pojedinačno. Ipak ovaj korak je neophodan kada se traži rešenje od 6 ili više znakova kako bi se povećale šanse za njeno nalaženje. Prosečan broj pokušaja za pronalazak kombinacije od 8 znakova je 9,4 pri veličini populacije od 1000 jedinki i 100 iteracija algoritma.

Napredak između svakog ciklusa iteracija algoritma se može pratiti i kroz prosečnu prilagođenost najboljih jedinki iz svakog pokušaja kao što je prikazano na slici 4.1. U proseku povećanje ovih vrednosti za kombinacije od 8 znakova iznosi oko 4%, dok je za kombinacije sa manje znakova ovaj raspon veći i iznosi 8% u proseku.



Slika 4.1 Primer povećanja prosečne prilagođenosti individua

Inicijalne vrednosti parametara su 500 članova populacije i 50 iteracija u svakom ciklusu algoritma, što daje zadovoljavajuće rezultate sa kratkim vremenom izvršenja. Povećavanjem broja jedinki smanjuje se mogućnost zalaženja pretrage u lokalne optimume zbog povećane verovatnoće u raznolikosti kandidatskih rešenja.

Sa druge strane, veći broj iteracija dovodi do boljeg usmeravanja populacije individua prema rešenju ali se vreme izvršenja tada povećava više nego linearno sa svakim novim pokušajem.

5. ZAKLJUČAK

Genetski algoritmi predstavljaju metode za rešavanje optimizacionih problema, bazirane na prirodnoj selekciji, tj. procesu kojim je vođena biološka evolucija. Konstantne izmene na populaciji individua upotrebom genetskih operatora dovode do njenog „evoluiranja“ u pravcu optimalnog rešenja.

Ovi algoritmi se mogu primeniti na mnoge probleme koji nisu pogodni za standardne optimizacione algoritme, uključujući i probleme kod kojih je razmatrajuća funkcija isprekidana, stohastička ili veoma nelinearna. Takođe, njihova korisnost se ogleda u delovanju na problemske domene sa kompleksnim prostorima pretrage. Genetski operatori ukrštanja i mutacije su dizajnirani kako bi pomerali populaciju individua od lokalnih optimuma u kojima tradicionalni algoritmi „penjanja uzbrdo“ mogu „zaglaviti“.

U ovom radu je prezentovana jedna implementacija genetskog algoritma čija je namena rešavanje igre Mastermind koja zahteva kratko vreme izvršavanja i što manji broj pokušaja pronalaska tražene kombinacije.

Osnovna ideja je bila kreiranje populacije jedinki koja bi se menjala kroz zadati broj iteracija i na kraju selektovala najbolja među njima. Svaki pokušaj sa sobom donosi informacije o prilagođenosti individua što znači da je evaluaciona funkcija složenija sa svakim novim pokušajem a samim tim i kvalitet rešenja koja se odabiru.

Kvalitet opisanog rešenja je meren u odnosu na algoritme koji su nastali ranije a primenjivani su na isti problem. Iz tog posmatranja je zaključeno da su performane opisanog algoritma uporedive sa ranijim rešenjima pri standardnim podešavanjima. Takođe, primetan je uspon prosečne prilagođenosti svakog pokušaja, naročito za veće kombinacije znakova.

6. LITERATURA

- [1] Ulrich Bodenhofer , *Genetic Algorithms : Theory and Applications*
- [2] Mitchell Melanie, *An Introduction to Genetic Algorithms*
- [3] A.E. Eiben and J.E. Smith, *Introduction to Evolutionary Computing*
- [4] Phil Husbands , Peter Copley , Alice Eldridge , James Mandelis,
An Introduction to Evolutionary Computing for Musicians
- [5] Cellular Reproduction: Multiplication by Division,
<https://publications.nigms.nih.gov/insidethecell/chapter4.html>
- [6] Stuart J. Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*
- [7] Raul Rojas , *Neural Networks*
- [8] Mastermind ,
[https://en.wikipedia.org/wiki/Mastermind_\(board_game\)](https://en.wikipedia.org/wiki/Mastermind_(board_game))
- [9] Branko Perišić, *Projektovanje softvera, materijali sa predavanja*,
Univerzitet u Novim Sadu
- [10] Client Applications : Windows Forms
<https://msdn.microsoft.com/en-us/>
- [11] UML 2 Activity Diagrams: An Agile Introduction
<http://agilemodeling.com/artifacts/activityDiagram.htm>
- [12] User Interface, https://en.wikipedia.org/wiki/User_interface

7. BIOGRAFIJA

Goran Rajn je rođen 04.10.1990. godine u Somboru. Osnovnu školu “Žarko Zrenjanin” završio je 2005. godine. Srednju školu “Srednja tehnička škola” u Somboru završio je 2009. godine. Iste godine upisao se na Fakultet tehničkih nauka, odsek Elektrotehnika i računarstvo, smer Računarstvo i automatika. Školske 2011/2012. godine upisao se na smer Računarske nauke i informatika. Diplomirao je 2013. godine stekavši zvanje Diplomirani inženjer elektrotehnike i računarstva.

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, RBR :	
Identifikacioni broj, IBR :	
Tip dokumentacije, TD :	monografska publikacija
Tip zapisa, TZ :	tekstualni štampani dokument
Vrsta rada, VR :	master rad
Autor, AU :	Goran Rajn
Mentor, MN :	Dr Đorđe Obradović, docent
Naslov rada, NR :	Upotreba evolutivnih algoritama za pronalaženje rešenja u igrama dekodiranja
Jezik publikacije, JP :	srpski
Jezik izvoda, JL :	srpski / engleski
Zemlja publikovanja, ZP :	Srbija
Uže geografsko područje, UGP :	Vojvodina
Godina, GO :	2016
Izdavač, IZ :	autorski reprint
Mesto i adresa, MA :	Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6
Fizički opis rada, FO :	6/ 36 / 10 / 2 / 26 / 0 / 0
Naučna oblast, NO :	Elektrotehnika i računarstvo
Naučna disciplina, ND :	Primenjene računarske nauke i informatika
Predmetna odrednica / ključne reči, PO :	Osnove računarske inteligencije
UDK	
Čuva se, ČU :	Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad
Važna napomena, VN :	
Izvod, IZ :	U ovom radu je opisana implementacija genetskog algoritma koji je primenjen na pronalaženje rešenja u igri Mastermind. Kvalitet rada aplikacije je meren na osnovu ukupnog broja pokušaja koji je algoritmu bio potreban za nalaženje rešenja, zavisno od veličine prostora pretrage
Datum prihvatanja teme, DP :	
Datum odbrane, DO :	
Članovi komisije, KO :	
predsednik	
član	
mentor	Dr Đorđe Obradović, docent, FTN Novi Sad
Potpis mentora	

KEY WORDS DOCUMENTATION

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	monographic publication
Type of record, TR :	textual material
Contents code, CC :	Master Thesis
Author, AU :	Goran Rajn
Mentor, MN :	Đorđe Obradović, PhD, assist. prof.
Title, TI :	Use of evolutive algorithms for finding solutions in decoding games
Language of text, LT :	serbian
Language of abstract, LA :	serbian / english
Country of publication, CP :	Serbia
Locality of publication, LP :	Vojvodina
Publication year, PY :	2016
Publisher, PB :	author's reprint
Publication place, PP :	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, PD :	6 / 36 / 10 / 2 / 26 / 0 / 0
Scientific field, SF :	Electrical and computer engineering
Scientific discipline, ND :	Applied computer science and informatics
Subject / Keywords, S/KW :	Fundamentals of artificial intelligence
UDC	
Holding data, HD :	Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad
Note, N :	
Abstract, AB :	This paper describes implementation of genetic algorithm which is used to find solution in game Mastermind. Quality of application is measured as total number of attempts that algorithm requires before finding solution, depending on size of search space
Accepted by sci. board on, ASB :	
Defended on, DE :	
Defense board, DB :	
president	
member	
mentor	Đorđe Obradović, PhD, assist. prof., FTN Novi Sad
Mentor's signature	