UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Daniel Rižnar, Uroš Kosič

Umetna inteligenca 2

SEMINARSKA NALOGA
POROČILO

Mentor: Martin Možina

Ljubljana, 2011

Kazalo

Povzetek

Za oblikovanje tega dokumenta je bil uporabljen sistem LaTeX.

Ključne besede:

j

Poglavje 1

Opis naloge

RPG (role-playing game) ali igra igranja vlog je izraz za igre, v katerih igralci prevzamejo vloge namišljenih likov, postavljenih v domišljijsko okolje. Bodisi z neposrednim igranjem, bodisi z opisovanjem njihovih dejanj nato igrajo te vloge v zgodbi, ki je lahko vnaprej začrtana ali pa nastaja sproti. Uspeh ali neuspeh njihovih dejanj določa dogovorjen sistem pravil konkretne igre. Velik problem je ravnotežje različnih likov, saj se v večini primerov izkaže, da imajo nekateri liki prednost pred ostalimi in lažje zmagujejo v neposrednih dvobojih.

Posamezna RPG igra ponuja izbiro med več različnimi bojevniki, vsak ima svoje prednosti in slabosti. V tem poglavju bomo opisali konkretne značilnosti naše igre in predstavili problem.

1.1 Značilnosti igre

V igri sodelujeta dva igralca, vsak s svojim bojevnikom. Bojno polje je plošča poljubne velikosti, razdeljena na kvadratno mrežo. Bojevnika izmenično izvajate v naprej definirane akcije. Cilj igre je pokončati nasprotnika, preden on pokonča tebe. Vsak bojevnik ima na izbiro v naprej določene akcije, ki jih opisuje tabela 1.1. Posamezni bojevnik je opisan z atributi, ki jih podaja 1.2.

1.2 Bojevniki

Za potrebe seminarske naloge, smo se omejili le na dva bojevnika - Tank in Vojak. Značilnosti obeh smo določili kot v tabeli 1.3.

Akcija	Opis Akcije
Move	Akcija pomeni premik v eno izmed štirih smeri (gor, dol, levo, desno). Vsak premik zahteva določeno količino energije.
Pass	Akcija pomeni počitek, bojevniku se ob počivanju poveča energija.
Fire	Strel s primarnim orožjem. Vsak strel pomeni zmanjšanje določene količine energije in zmanjšanje življenja nasprotnika. Škoda je delno odvisna tudi od naključja.

Tabela 1.1: Akcije bojevnikov

Atribut	Opis Atributa
Life	Celo število, ki podaja preostalo življenje.
Speed	Hitrost, ki opisuje za koliko kock se bojevnik lahko premakne ob premikih.
Energy	Preostala energija bojevnika, ki jo lahko porabi za izvajanje akcij.

Tabela 1.2: Atributi bojevnikov

1.3 Opis problema

Znan problem v RPG igrah je ravnotežje različnih likov, ki tekmujejo med seboj. Kljub različnim lastnostim, morajo imeti bojevniki enake možnosti za zmago v igri. Naša naloga je bila poiskati konkretne vrednosti atributov za zgoraj opisana bojevnika tako, da bosta igrala čim bolj izenačeno igro. Problem smo ločili na dva dela - simulacijo igranja in optimizacijo izenačenosti bojevnikov. Za simulacijo igranja smo uporabili *minimax* algoritem. Za optimizacijo smo uporabili *genetske algoritme* in jih primerjali s *hill climbing*-om.

Tip	Trpežnost	Domet	Hitrost	Obnavljanje	Varčnost	Energija
Tank	boljša	boljši	višja	slabše	boljša	višja
Vojak	slabša	slabši	nižja	boljše	slabša	nižja

Tabela 1.3: Značilnosti bojevnikov

Poglavje 2

Metode dela

V tem poglavju bomo opisali uporabljene algoritme in načine izvajanja meritev. Na koncu poglavja bomo podali nekaj implementacijskih podrobnosti.

2.1 Simulacija igranja

Za simulacijo igranja igre smo uporabili algoritem *minimax*. Z njim smo poskušali doseči, da igralca igrata čim bolj optimalno igro, tj. v danem trenutku izbereta najboljšo možno potezo.

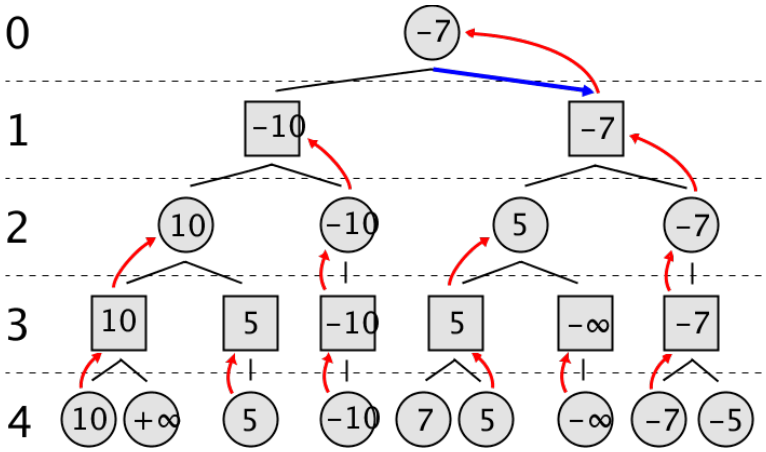
2.1.1 Algoritem minimax

Algoritem se uporablja za minimizacijo možne izgube, pri maksimizaciji možnega dobička. V naši igri igralca izmenjujeta poteze in poskušata izbrati najboljšo svojo potezo oz. maksimizirati svojo vrednost, posledično pa minimizirati nasprotnikovo. Tako glede na trenutno pozicijo identificiramo igralca MIN in MAX. Tisti ki je na potezi predstavlja igralca MAX, nasprotnik pa igralca MIN. Algoritem gradi drevo, kjer je v sodih nivojih (z začetkom v korenu) na potezi igralec MAX, na lihih pa igralec MIN. Na MAX nivojih, igralec vedno izbere stanje, ki maksimizira njegovo vrednost, na MIN nivojih pa igralec izbere stanje, ki minimizira njegovo vrednost (ker predpostavlja, da bo nasprotnik izvedel za njega najslabšo možno potezo). Naslednja poteza je tista poteza, ki jo igralec MAX izbere v korenskem vozlišču. Algoritem je predstavljen na Sliki 2.2. Propagiranje vrednosti vozlišč poteka od spodaj navzgor, zato je potrebno te vrednosti določiti v listih. Ker je prostor preiskovanja prevelik (če bi lahko preiskali celoten prostor, bi v listih vedno imeli samo dve možnosti - zmago ali

poraz), je potrebno višino drevesa omejiti, vrednost v listih pa določiti z neko hevristiko. Naša hevristika je predstavljena na 2.1.

$$val(n) = \begin{cases} -1000 & \text{če je igralec MAX mrtev} \\ +1000 & \text{če je igralec MIN mrtev} \\ +1.0 \cdot MaxLifeRatio - 1.2 \cdot MinLifeRatio + & \\ +0.1 \cdot MaxEnergyRatio - 0.1 \cdot MinEnergyRatio & \text{sicer} \end{cases}$$

Slika 2.1: Hevristika. Členi v tretjem delu funkcije predstavljajo razmerje med trenutnimi in izačetnimi vrednostmi atributov. Intuitivno, hevristika bo ocenila stanje kot dobro, če je življenje MAX igralca visoko, MIN igralca nizko, energija MAX igralca visoka in MIN igralca nizka.



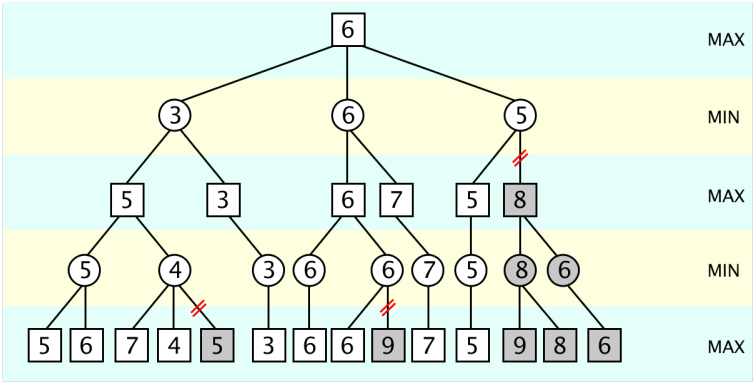
Slika 2.2: Algoritem minimax: slika predstavlja delovanje algoritma minimax. Številke v vozliščih predstavljajo vrednost posameznega stanja z vidika MAX igralca. Puščice predstavljajo propagacijo vrednosti od spodaj navzgor (igralec MIN na lihih nivojih izbira minimalno vrednost, izgralec MAX na sodih nivojih izbira maksimalno vrednost).

2.1.2 Problemi in prilagoditve

V naši igri pomembno vlogo igra tudi naključje. Akcija Fire ima definirano maksimalno in minimalno škodo, ki jo zadane nasprotniku. Vsakič ko bojevnik izbere to akcijo, se vrže kocka, ki z verjetnostjo 0.5 določi stopnjo škode (ali

maksimalna ali minimalna). Zaradi naključnosti, rezultati minimax algoritma ob istih vhodnih podatkih niso vedno enaki. Da bi omilili vpliv naključnosti, je potrebno igro simulirati večkrat, kar privede do povečanja časovne kompleksnosti (sicer za konstanto, a ker se ena simulacija izvaja precej časa, že ta konstanta pomeni veliko razliko). Zato smo implementirali *alfabeta* rezanje, ki ob zagotavljanju enake rešitve kot navaden *minimax* v večini primerov zmanjša število evaluacij listov. Na Sliki 2.3 je prikazan minimax algoritem z alfabeta rezanjem.

Z do zdaj opisanimi pravili igre se je pojavljal problem bežanja. Velikokrat se je zgodilo, da je bojevnik, ki je ugotovil da ne more zmagati, začel bežati pred drugim. Tako je igra trajala zelo dolgo ali pa se sploh ni končala. Zato smo v igro uvedli pravilo, da se bežočega bojevnika kaznuje. Pravilo smo definirali kot: če se nek bojevnik odmakne od drugega in ga v naslednji potezi nasprotnik zadane, se mu hitrost razpolovi. V primeru, da se bežoči umakne izven dosega nasprotnika, se nasprotniku moč zadetka zmanjša za 2^d, kjer je d razdalja med bežočim in zunanjo mejo dosega akcije. Tako bežočega upočasnimo in damo nasprotniku možnost, da ga dohiti oziroma pokonča. Zdaj se bojevniki manjkrat odločajo za bežanje.



Slika 2.3: Algoritem minimax z alfabeta rezanjem: slika predstavlja delovanje algoritma minimax z alfabeta rezanjem. Za najblolj desen rez velja: V vozlišču je na potezi igralec MAX in bo vedno izbral vozlišče z maksimalno vrednostjo (trenutno je to vrednost 6). Na nivoju 1 je na potezi igralec MIN in bo vedno izbral minimalno vrednost (v najbolj desnem vozlišču na nivoju 1 je trenutno to 5). Ker pa že vemo, da ima eno izmed vozlišč na nivoju 1 vrednost 6 in ker bo igralec MIN v najbolj desnem vozlišču zagotovo izbral vozlišče z vrednostjo <= 5 (ker imamo 5 že izračunano), lahko na označenem mestu opravimo rez.

2.2 Optimizacija uravnoteženosti

Za uravnoteževanje bojevnikov smo uporabili genetske algoritme in hill climbing.

2.2.1 Genetski algoritmi

Genetski algoritmi so prilagodljive metode, ki jih uporabljamo za reševanje iskalnih in optimizacijskih problemov. Temeljijo na genetskem procesu bioloških organizmov, saj se zgledujejo po evoluciji v naravi, kjer se populacija neke vrste skozi generacije razvija po načelu naravnega izbora in preživetju uspešnejšega. Genetski algoritmi posnemajo tiste procese v naravi, ki so bistveni za evolucijo. V naravi posamezniki neke populacije med seboj tekmujejo za življenjsko pomembne vire. Hitrejši in pametnejši predstavniki vrste bodo imeli več in boljče vire za preživetje, počasnejši in manj pametni jih bodo dobili težje ali pa sploh ne. Ravno tako se pojavi tekmovanje pri iskanju partnerja za razmnoževanje. Tisti, ki so uspešnejši v preživetju in pri parjenju, bodo po vsej verjetnosti imeli relativno večje število potomcev. Slabši posamezniki jih bodo imeli manj ali pa sploh ne. To pomeni, da se bodo geni dobro prilagojenih oz. ustreznih posameznikov bolj razširili na prihodnje generacije, slabši pa bodo celo izumrli. Kombinacija genov dveh ustreznih staršev lahko privede do pojava super-ustreznega potomca, ki ima boljše lastnosti od obeh staršev. Na ta način se vrste razvijajo in prilagajajo na svoje okolje. Ti algoritmi delujejo po analogiji k temu naravnemu procesu. Delujejo nad neko populacijo osebkov (kromosomov), od katerih vsak predstavlja možno rešitev danega problema. Vsakemu kromosomu se priredi ocena uspešnosti, ki je prilagojena iskanemu problemu. Ustreznejši kromosomi - tisti z boljšo oceno - imajo več možnosti za reprodukcijo od ostalih. Nad trenutno populacijo izvedemo simuliran proces evolucije. Iz naše populacije izberemo podmnožico staršev, ki se razmnožujejo. Tako za našo populacijo dobimo nove potomce, ki prevzamejo nekaj lastnosti od vsakega starša. Manj ustrezni predstavniki se bodo razmnoževali z manjšo verjetnostjo in tako izumrli. Na ta način se dobre lastnosti razširijo v naslednje generacije. Če smo genetski algoritem dobro zastavili, bo populacija konvergirala k optimalni rešitvi. V nadaljevanju bomo predstavili konkretne značilnosti genetskih algoritmov za našo domeno.

Osnovni pojmi

Osebek je možna rešitev problema. V naši domeni je osebek sestavljen iz dveh bojevnikov, oziroma množice atributov. Osebke imenujemo tudi *kromosomi*.

Geni so posamezni atributi v osebku.

Kriterijska funkcija je funkcija, s katero ocenjujemo osebke. Pri nas je ta funkcija definirana kot $\frac{1}{LifeDiff}$, kjer je LifeDiff razlika med življenjema dveh bojevnikov v osebku, kot jo vrne algoritem za simulacijo igre.

Populacija je množica osebkov. Obdelujemo jo v korakih, ki jih imenujemo *generacija*. Iz osebkov trenutne populacije (*staršev*) tvorimo naslednike (*potomce*), ki pripadajo populaciji naslednje generacije.

Elita je podmnožica najboljših osebkov v populaciji, ki privzeto napreduje v naslednjo generacijo.

Selekcija iz populacije izbere osebke, ki bodo sodelovali v razmnoževanju.

Križanje iz genskega materiala dveh staršev zgradimo dva nova otroka.

Mutacija je z majhno verjetnostjo definirana sprememba genov nekega osebka.

Genetske operacije

Na 2.4 je opisana osnovna zanka genetskega algoritma. V nadaljevanju bomo opisali vse operacije, ki jih izvršimo med izvajanjem ene iteracije genetskega algoritma.

Ustvarjanje začetne populacije
Ta operacija se izvede samo enkrat. Najprej je potrebno definirati tako imenovani model bojevnika, kjer določimo intervale na katerih se bodo nahajale vrednosti atributov. Podrobnosti o specifikaciji so podane na koncu tega dokumenta. Začetno populacijo dobimo naključno, z omejitvami ki so podane v modelu bojevnikov.

Ocenjevanje uspešnosti populacije
Uspešnost populacije ocenjujemo z *minimax* algoritmom, ki ga večkrat poženemo in vsakič beležimo razliko v življenjih bojevnikov. Boljši osebki bodo bolje uravnoteženi in tako imeli manjšo povprečno razliko v življenjih.

```
begin
  ustvari začetno populacijo;
  while ni pretekel cas do
        oceni uspešnost populacije;
        izberi elito in jo dodaj v novo generacijo;
        izberi starše za razmoževanje;
        križanje staršev;
        mutacija otrok;
  end
end
```

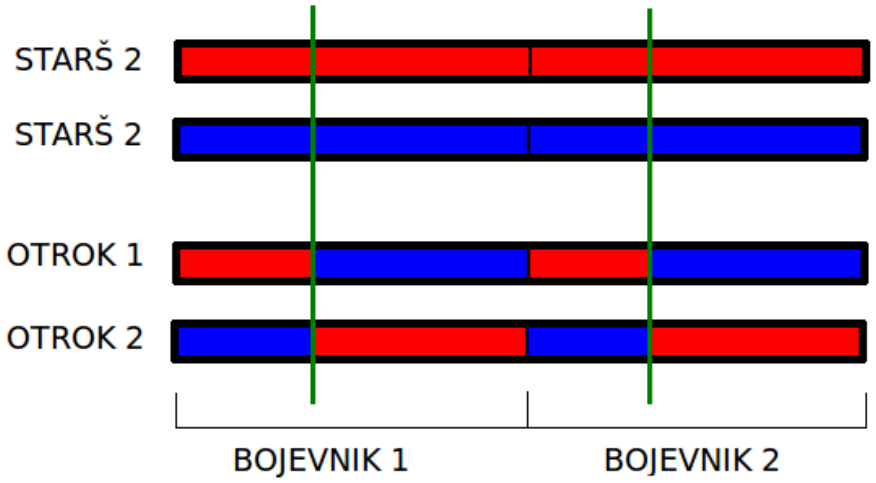
Slika 2.4: Osnovna zanka genetskega algoritma.

Izbira elite
Elito izberemo zato, da si zagotovimo, da bo nekaj najboljših osebkov zagotovo napredovalo v naslednjo generacijo. Če elite ni, se lahko zgodi, da operacija izbire staršev včasih ne izbere najboljših osebkov.

Izbira staršev
Izbira staršev poteka po metodi rulete, kjer bolje ocenjeni osebki zasedajo proporcionalno več prostora na kolesu kot slabše ocenjeni, zato je večja verjetnost da izberemo boljše kot slabše.

Križanje staršev
Križanje predstavlja Slika 2.5. Omeniti je potrebno, da poleg bojevnikovih atributov (Life, Energy, Speed) bojevnika opisujejo tudi akcije. Vsaka akcija je sestavljena iz atributov, ki so pravtako podvrženi križanju in mutaciji in so definirani enako kot "navadni"atributi bojevnika. Podrobnosti so podane na koncu tega dokumenta.

Mutacija otrok
Vsak otrok gre skozi proces mutacije. Tu se lahko vsak gen (atribut) z neko majhno verjetnostjo naključno spremeni. Mutacija je glavno orodje za preprečevanje zadrževanja v lokalnih maksimumih, saj nam omogoča preskok v nova območja raziskovanja.



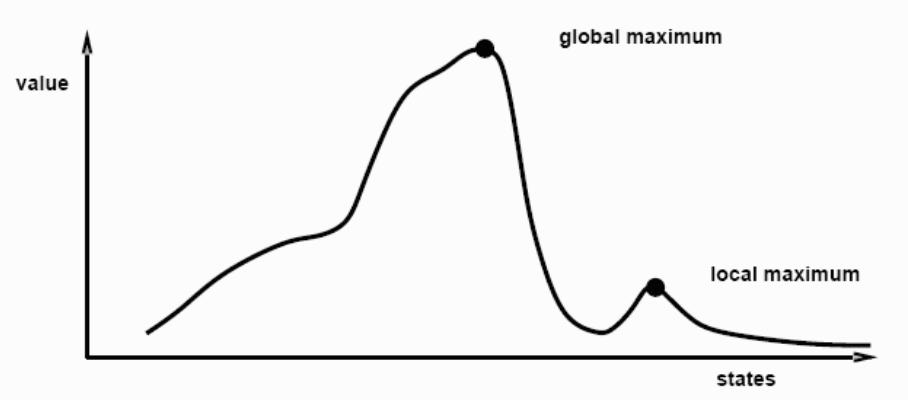
Slika 2.5: Pri križanju iz dveh staršev dobimo dva potomca. Starš je predstavljen s seznamom atributov, seznam pa je razdeljen na dva dela - vsak del predstavlja enega bojevnika. Za vsak del seznama naključno določimo odrezno točko (zeleni črti). Iz atributov sestavimo dva otroka kot prikazuje slika.

2.2.2 Hill climbing

Hill climbing je optimizacijska tehnika, ki pripada družini lokalnih iskanj. Je iterativen algoritem, ki prične z neko naključno rešitvijo problema, potem pa z ikrementalnim spreminjanjem enega elementa rešitve išče boljše stanje. Ko preizkusi vse spremembe nekega stanja, izbere tisto, ki ga privede v najboljše možno stanje od trenutnega in se premakne tja. To ponavlja tako dolgo, dokler obstaja sprememba na boljše. Problem te metode je, da se lahko zgodi, da obstane v lokalnem maksimumu (odvisno od začetnega stanja). To težavo prikazuje Slika 2.6. Problem lahko omilimo tako, da algoritem zaženemo iz večih začetnih stanj. Mi smo pri testiranju metode začeli iz stanj, ki so se generirala v prvem koraku genetskega algoritma. Ta stanja smo uredili po kvaliteti (začenši z najboljšim) in jih v tem vrstnem redu uporabljali za preiskovanje s *hill climbingom*.

2.3 Parametri algoritmov

Vsi uporabljeni algoritmi zahtevajo nastavitev določenih parametrov. V tem razdelku bomo opisali našo izbiro vrednosti parametrov in poskusili to izbiro tudi osmisliti. Kot je pogosto v računalništvu, je tudi pri našem problemu



Slika 2.6: Pri algoritmu hill climbing naletimo na problem lokalnega maksimuma. Za primer na sliki velja, da če je naše začetno stanje na desni strani lokalnega maksimuma, bomo tam tudi obtičali, saj bo iz vrha izgledalo, da ne obstaja nobeno boljše stanje.

potrebno najti kompromis med časom, ki ga imamo na voljo in natančnostjo meritev. S časovnega vidika, je ozko grlo problema simuliranje igre. Kot smo omenili v poglavju o problemih minimax algoritma, zaradi naključnosti v definiciji naše igre, rezultati večih ponovitev simuliranja igre niso enaki. Da faktor naključnosti omilimo, je treba igro simulirati večkrat, kar traja precej časa. Dolžino trajanja simulacije uravnavamo z nastavitvijo globine minimax drevesa. Globina drevesa in število ponovitev simulacije določata potreben čas za evaluacijo enega osebka. Mi smo za naše testiranje izbrali globino 4 in 40 kratno ponovitev simulacije. 95% interval zaupanja je s temi nastavitvami in bojevniki, ki zadoščajo omejitvam na koncu poglavja, v povprečju širok 2.1.

Slabost genetskih algoritmov je veliko število parametrov, ki jih je težko teoretično določiti. Potrebno je predvsem dobro poznavanje domene in preizkušanje različnih kombinacij. Eden izmed parametrov je velikost populacije. Če je leta premajhna, algoritem ne preišče dovolj prostora, da bi deloval konsistentno. V literaturi se večkrat omenja velikost populacije 50 osebkov. Mi smo ta parameter nastavili na 25, predvsem zaradi krajšega časa izvajanja in ker so eksperimenti pokazali, da daje dobre rezultate. Pomemben vpliv ima tudi verjetnost mutacije. Če je le-ta prevelika, se lahko zgodi da bo genetski algoritem potreboval ogromno časa da najde optimum. Če je premajhna, se morda nikoli ne bo premaknil iz lokalnega optimuma. Ker smo pri testirajnu uporabljali elito, smo faktor mutacije nastavili na 0.1, kar je precej velika verjetnost. Elita poskrbi, da se informacija o lokalnem optimumu prenese naprej, mutacije

pa omogočajo preskok iz lokalnega optimuma. Velikost elite smo nastavili na 20% velikosti populacije (5 osebkov).

Vsi parametri so zbrani v Tabeli 2.1.

Parameter	Vrednost
Globina Minimax Drevesa	4
Ponovitev Simulacije	40
Velikost populacije	25
Verjetnost mutacije	0.1
Faktor Elite	0.2

Tabela 2.1: Parametri algoritmov

Poglavje 3

Rezultati

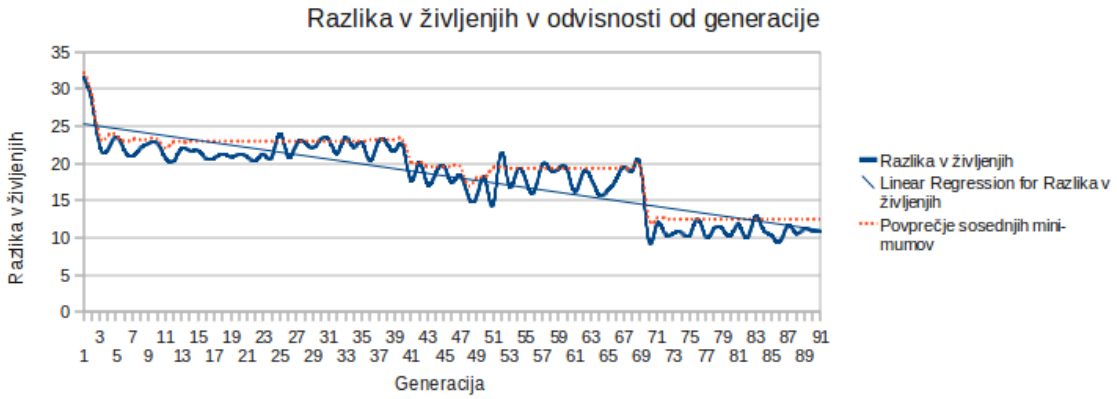
V tem poglavju predstavimo rezultate in primerjamo dva algoritma za optimizacijo - genetske algoritme in hill climbing.

3.1 Genetski algoritmi

Genetski algoritmi so se izkazali kot primerna metoda za reševanje našega problema. Iz rezultatov je razvidna konvergenca uravnoteženosti bojevnikov, prav tako pa smo z njimi potrdili nekatere lastnosti genetskih algoritmov. Na Grafu 3.1 je prikazana razlika v življenjih najboljšega osebka v odvisnosti od generacije. Opazimo lahko nihanje kvalitete najboljšega osebka (ker ga z elito postavimo v novo generacijo, kjer je na novo ocenjen), ki je posledica naključnosti v igri in se sklada z izračunano širino intervala zaupanja. Za lepši prikaz smo izračunali še povprečje med sosednjima minimumoma, ki zaduši fluktuacije (oranžna črtkana črta). Tam se lepo vidi obnašanje genetskega algoritma, ki raziskuje v lokalnih optimumih (3. do 40. generacija, 41. do 69. generacija, 70. do 91. generacija), vmes pa mu uspe preskočiti v v boljši optimum.

3.2 Hill Climbing

- potrebno še dodatno preverjanje rezultatov! - zakaj dobri rezultati (če bodo)
- kaj to pomeni v splošnem



Slika 3.1: Oranžna črtkana črta je dobljena z izračunom povprečja med lokalnim minimumom (vključno) in vrednostmi do naslednjega minimuma.

3.3 Primerjava algoritmov

- zakaj genetski boljši - zakaj je hill climbin delal dobro - graf gibanja kvalitete obeh - ali je naključno, da je hill climbin uspel?

Slike

Tabele