UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sandi Gec

Tridimenzionalna vizualizacija genetskih algoritmov

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: prof. dr. Marko Robnik Šikonja

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00017/2010
Datum: 01.10.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **SANDI GEC**

Naslov: **TRIDIMENZIONALNA VIZUALIZACIJA GENETSKIH ALGORITMOV**

VISUALIZATION OF GENETIC ALGORITHMS IN THREE-DIMENSIONAL SPACE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Pri poučevanju optimizacijskih tehnik potrebujemo orodja za boljšo predstavitev vsebin in za njihovo lažje razumevanje. V ta namen želimo vizualizirati tudi delovanje genetskih algoritmov.
Podrobno proučite delovanje genetskih algoritmov in njihovo delovanje predstavite na problemu, ki omogoča vizualno prepričljivo tridimenzionalno predstavitev. Predstavitev osebkov, prostor stanj in preiskovanje ustrezno parametrizirajte ter izdelajte programsko rešitev, ki bo omogočala spreminjanje pogojev delovanja in opazovanje vpliva parametrov na način preiskovanja in konvergenco.

Mentor:
doc. dr. Marko Robnik Šikonja

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Sandi Gec,

z vpisno številko 63050033,

sem avtor diplomskega dela z naslovom:

Tridimenzionalna vizualizacija genetskih algoritmov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Marko Robnik Šikonja
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15.4.2011 Podpis avtorja:

Zahvala

Iskreno bi se želel zahvaliti mentorju prof. dr. Marko Robnik Šikonja za vso podporo, požrtvovalnost in potrpljenje, ki ga je izkazal v procesu nastanka diplomskega dela.

Zahvalil bi se tudi vsem prijateljem ter bližnjim za vso podporo pri nastanku tega dela.

Posebna zahvala pa gre predvsem mojim staršem ter ožji družini za vso podporo v času študija.

Kazalo

Povzetek

Cilj diplomskega dela je izdelava aplikacije, ki vizualizira delovanje genetskih algoritmov na konkretnem preiskovalnem problemu v dvodimenzionalnem in tridimenzionalnem prostoru. Aplikacija služi kot pripomoček za boljše razumevanje delovanja genetskega algoritma. Najprej predstavimo zgodovino genetskih algoritmov, metodologijo s podrobnim opisom vseh faz algoritma in področja uporabe. Zastavimo si problem, ki ga z genetskim algoritmom rešujemo in potek vizualiziramo, predstavimo delovanje uporabniškega vmesnika in parametrov algoritma, opišemo delovanje genetskega algoritma v aplikaciji in predstavimo vlogo Bezierjevih krivulj. Predstavimo najpomembnejše tehnologije in orodja uporabljena pri izdelavi aplikacije, kot so programski jezik java, knjižnica Java 3D API, razvojno okolje IntelliJ Idea, orodje za izdelavo uporabniških vmesnikov JFormDesigner, orodje za gradnjo in uporavljanje programskega projekta Apache Maven in orodje za nadzor nad izvorno kodo TortoiseSVN. Opišemo celotno implementacijo problema po korakih, z izdelavo uporabniškega vmesnika, opisom delovanja in implementacijo java knjižnice TimingFramework, arhitekturo Java 3D animacije v aplikaciji in implementacijo Maven orodja pri gradnji aplikacije. V zadnjem poglavju opišemo nekaj vizualizacijkih rešitev in obnašanje algoritma pri reševanju problema v dvodimenzionalnem in tridimenzionalnem preiskovalnem prostoru. V zaključku podamo nekaj idej za nadaljne izboljšave.

Ključne besede:

genetski algoritem, vizualizacija, učni pripomoček, Bezierjeve krivulje

Abstract

The goal of this work is an application for visualization of genetic algorithms on a simple problem in two-dimensional and three-dimensional space. The purpose is to improve comprehension of genetic algorithms. We present the history of genetic algorithms, the methodology with detailed description of all phases of the algorithm and its applications. We describe a problem that we are solving with genetic algorithm. We visualize the solving process, describe the work of graphical user interface and parameters of the algorithm. We describe the role of the genetic algorithm in the application and the role of Bezier curves. We also represent the most important technologies and tools that were used in the developement of the application: Java programming language, Java 3D API library, IntelliJ Idea integrated development envioronment, JFormDesigner tools for making of the graphical user interface, the tools for making and managing the software project Apache Maven and the tools for controlling source code TortoiseSVN. We describe the whole implementation of the problem step by step by making a graphic user interface, Step by step we describe the implementation. We describe the implementation of java library Timing Framework, an architecture of java 3D animation in application and implementation of Maven tool, used for creating the application. In the last chapter we present some interesting solutions and the behavior of the algorithm when solving the problem in two dimensional and three-dimensional space. In conclusion we suggest some ideas for further improvements.

Key words:

genetic algorithm, visualization, learning tool, Bezier curves

Poglavje 1

Uvod

Uporaba evolucijskih algoritmov narašča, zaradi uspešnosti pri reševanju problemov na različnih področjih. Za razliko od popolnoma deterministčnih algoritmov evolucijski algoritmi vključujejo elemente stohastičnosti, ki omogočajo sprotno prilagajanje algoritma med reševanjem problema. Evolucijski algoritmi so bili uspešno uporabljeni pri reševanju optimizacijskih in preiskovalnih problemov, znani pa so tudi primeri interdisciplinarne rabe, npr. na področjih biologije in kemije.

Eden izmed evolucijskih algoritmov je genetski algoritem (GA). GA posnema pomembne faze evolucije - selekcijo, mutacijo in križanje. Boljši oz. uspešnejši osebki imajo pri evoluciji večjo možnost reprodukcije. Delovanje je takšno, da se v fazi selekcije izbirajo osebki na osnovi uspešnosti trenutne generacije, mutacije skrbijo za naključne spremembe, križanje pa za izmenjavo genskega zapisa. Takšna analogija evolucije se izkaže kot zelo uspešna tudi pri optimizacijskih problemih, kjer je GA sposoben poiskati rešitve. Za lažje razumevanje delovanja GA smo izdelali študijski pripomoček, ki vizualizira delovanje GA na konkretnem preiskovalnem problemu. Aplikacija vsebuje dvodimenzionalno in tridimenzionalno vizualizacijo GA, kjer je med izvajanjem simulacije mogoče prilagajati parametre algoritma, opazovati njegovo delovanje in grafično spremljati uspešnosti generacij.

V drugem poglavju diplomske naloge opišemo zgodovino GA ter metodologijo delovanja z opisom vseh faz algoritma. Navedemo tudi področja, kjer je uporaba GA najbolj razširjena.

V tretjem poglavju demonstriramo delovanja GA v aplikaciji. Predstavimo problem in opišemo elemente simulacije. Predstavimo uporabniški vmesnik, nabor funkcij in parametrov aplikacije. Opišemo delovanje osnovnih faz GA v simulaciji. Opišemo tudi Bezierjeve krivulje, ki imajo pomembno vlogo pri

vizualizaciji.

V četrtem poglavju predstavimo uporabljene tehnologije. Opišemo programski jezik java in vmesnik Java 3D za izdelavo tridimenzionalnih animacij. Predstavimo razvojno okolje IntelliJ Idea, ki posredno ali neposredno vključuje vse uporabljene tehnologije. Opišemo orodje za izdelavo uporabniških vmesnikov JFormDesigner, ki poenostavi izdelavo grafičnih vmesnikov in nudi lažje vzdrževanje. Opišemo orodje za gradnjo javanskih projektov Maven in orodje za nadzor nad izvorno kodo TortoiseSVN.

V petem poglavju predstavimo načrt implemnetacije in v fazah načrta opišemo vlogo tehnologij iz četrtega poglavja. Opišemo uporabniški vmesnik, delovanje in implementacijo vmesnika Timing Framework. Predstavimo tudi Java 3D arhitekturo tridimenzionalne različice rešitve. Opišemo tudi orodje Maven, ki aplikacijo zgradi v celoto.

V šestem poglavju predstavimo vizualizacijo delovanja dvodimenzionalne in tridimenzionalne različice iskanja z uporabo različnih parametrov.

V zadnjem poglavju podamo sklepne ugotovitve in opišemo možnosti za izboljšave in nadgradnje aplikacije.

Poglavje 2

Genetski algoritmi

V tem poglavju najprej opišemo zgodovinski razvoj genetskih algoritmov, nato metodologijo algoritmov s podrobnejšim opisom vseh faz algoritma, na koncu pa še področja, kjer je uporaba algoritma najbolj razširjena.

2.1 Zgodovina

V 50. in 60. letih 20. stoletja so mnogi znanstveniki neodvisno raziskovali evolucijo z namenom razvoja optimizacijskih algoritmov, ki bi jih uporabljali pri inženirskih problemih. Genetski algoritem je v 60. letih 20. stoletja prvi predstavil ameriški profesor John Holland z univerze v Michiganu. Razvoj algoritma se je nadaljeval naslednje desetletje. Cilj Hollanda ni bil razvoj algoritma, ki rešuje specifične probleme, ampak formalno študijo prilagajanja pojavov z narave in načina s katerimi te mehanizme implementirati v računalniških sistemih. Leta 1975 je izšla Holland-ova knjiga "Adaptation in Natural and Artificial Systems", v kateri je genetski algoritem predstavljen kot abstrakcija biološke evolucije. Knjiga podaja teoretično podlago za prilagoditev algoritma v računalniškem sistemu [2].

Zanimanje za nadaljnje preučevanje teh pojavov je med Hollandovimi študenti naraslo, sledila je nagrajena doktorska disertacija Davida Goldberga na temo apliciranja algoritma pri optimizaciji postavitev plinovodov in leta 1989 še odmevna knjiga "Genetic algorithms in Search, Optimization, and Machine Learning". Po izdaji knjige se je uporaba genetskih algoritmov razširila tudi v industriji, kot primer naj navedem leta 1989 izdan produkt Evolver podjetja Axcelis, ki je bil prvi razvit za osebne računalnike. Leto kasneje je bil produktu napisan članek o tem objavljen v New York Timesu.

Danes se genetski algoritmi aktivno uporabljajo za reševanje problemov na

različnih področjih, ki so opisana v zadnjem podpoglavju tega poglavja. Tudi v Sloveniji imamo podjetja, ki nudijo programske rešitve z genetskimi algoritmi.

2.2 Metodologija genetskih algoritmov

Genetski algoritmi delujejo po principu hevrističnega preiskovanja pri računanju in iskanju optimalnih rešitev za različne probleme. Delovanje algoritmov v grobem posnema princip biološke evolucije. Skupna ideja genetskih algoritmov je, da za vsak problem razvijemo populacijo osebkov, pri čemer je vsak osebek predstavljen s stanjem. Osebek v populaciji je običajno predstavljen binarno ali kot niz simbolov, ki jim pravimo geni, kromosomi ali genotipi genomov. V vsaki generaciji se stohastično generira naslednja generacija z uporabo transformacij, ki jih srečamo v biološki genetiki in naravni selekciji. Delovanje genetskih algoritmov lahko podrobneje opišemo s štirimi osnovnimi fazami.

2.2.1 Inicializacija

Začetno populacijo osebkov generiramo naključno. Velikost populacije je odvisna od narave problema, tipično od 10 do 1000 osebkov. Z naključnim generiranjem osebkov želimo pokriti čim večje območje možnih rešitev oz. kar največji prostor stanj.

2.2.2 Selekcija

Selekcija ali evolucijski model imenujemo izbiro osebkov za razmnoževanje. Izberemo določeno število osebkov iz populacije, s katerimi se v fazi reprodukcije ustvarijo novi osebki. Poznamo več vrst selekcije, saj ne obstaja idealna selekcija, ki bi zadovoljila vsem kriterijem, kot npr. kako ohraniti dobre osebke in pri tem zagotoviti raznolikost populacije, kako preprečiti prezgodnjo konvergenco, do katere mere naj vpliva faktor naključnosti na izbiro osebkov.

Najbolj pogoste metode selekcije so naslednje:

- stohastično univerzalno vzorčenje
- rangovna izbira
- turnirska izbira

- proporcionalna izbira

Vsaka izmed naštetih metod upošteva nekatere kriterije bolj kot druge; nekatere metode so bolj deterministične, druge bolj stohastične in dajejo prednost naključni izbiri. Kvaliteto osebkov v trenutni generaciji ocenimo na podlagi ocene kakovosti ali funkcije kvalitete (ang. fitness). S tem dosežemo, da imajo boljši osebki večjo verjetnost potomcev, slabši osebki pa imajo manj potomcev ali pa jih sploh nimajo. Zaradi stohastičnosti dosežemo večjo raznolikost populacije in preprečujemo prezgodnjo konvergenco k slabšim rešitvam.

Stohastično univerzalno vzorčenje Metoda stohastičnega univerzalnega vzorčenja ima majhno varianco pri izbiri, saj se izogne možnosti, da najboljši osebki nimajo potomcov [1]. Glede na kvaliteto osebkov f_i tvorimo verjetnostno distribucijo kot pri proporcionalni izbiri

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}.$$

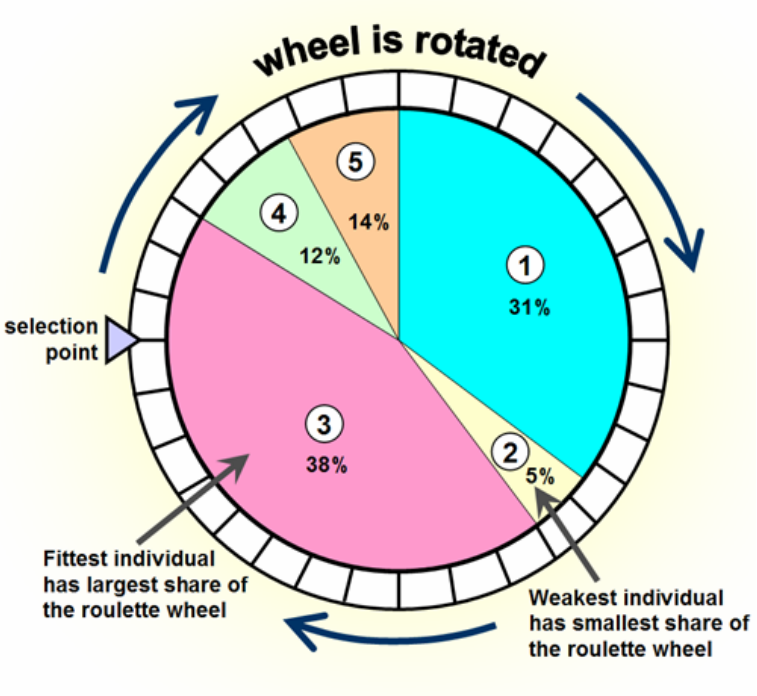
Izbirne funkcije osebkov normaliziramo tako, da je vsota funkcij kvalitete enaka 1. Na številski trak dolžine 1 naključno razvrstimo osebke in vsak dobi dolžino traku sorazmerno p_i. Določimo število N, ki predstavlja število osebkov, ki jih želimo generirati in izberemo naključno število z intervala $[0, 1/N]$. Število potomcev posameznim osebkom določimo glede na to, kolikokrat osebek prekrivajo točke $r + i\frac{1}{N}$, kjer je $i \in [0, 1, ..., N-1]$.

Izpis 2.1: Psevdokoda stohastičnega univerzalnega vzorčenja

```
//input:
// probability distribution of individuals (p_1, p_2, ..., p_m),
// number of offsprings N
// method returns: vector (c_1, c_2, ..., c_m), where c_i
// represents number of offsprings of individual i = sum_{i=1}^{m}; c_i = N
int[] SUS(double[] p, int N){
 select random double r ∈ [0, 1/N]
 sum == 0.0
 for i < m
     ci=0
     sum += pi
     while r < sum
        ci++;
        r+= 1/N;
```

```
 return c;
}
```

Proporcionalna izbira Vsakemu osebku se, glede na oceno kakovosti, izračuna verjetnost izbire. Verjetnost izbire osebka p_i je določena s formulo, kjer je f_i ocena kakovosti posameznika i v trenutni populaciji in N je velikost populacije. Izbiro osebkov si lahko predstavljamo kot ruletno kolo, kjer vsakemu osebku pripada širina kolesa sorazmerna njegovi verjetnosti. Osebki z večjo verjetnostjo izbire imajo večjo širino na kolesu, kot je prikazano na sliki 2.1.



Slika 2.1: Prikaz delovanja proporcionalne izbire.

Za lažje računanje seštejemo vse verjetnosti izbire (p_i) v populaciji in normaliziramo na 1. Analogno proporcionalni izbiri je krožnica, na kateri je vsakemu osebku, glede na verjetnost izbire, določen svoj del. V primerjavi s turnirsko izbiro je pri tej metodi večja verjetnosti, da so izbrani tudi manj obetavni osebki. V primerjavi s stohastičnim univerzalnim vzorčenjem, pri tej metodi prevladuje faktor naključja.

Izpis 2.2: Psevdokoda proporcionalne izbire

```
for all members of population
    sum += fitness of this individual
end for

for all members of population
    probability = sum of probabilities + (fitness / sum)
    sum of probabilities += probability
end for

loop until new population is full
     do this twice
         number = Random between 0 and 1
       for all members of population
           if number > probability but less than next probability
                then you have been selected
       end for
     end
     create offspring
end loop
```

Slabost proporcionalne izbire je, da ni robustna. V primeru, da ima manjša populacija osebkov bistveno večjo funkcijo kvalitete kot ostali, lahko ti osebki (in posledično njihovi potomci) povsem prevladajo in iskanje prezgodaj konvergira.

Rangovna izbira Rangovna izbira popravi robustnosti proporcionalne izbire. Osebke glede na kvaliteto sortiramo od najslabšega do najboljšega in jim določimo rang r_i na podlagi vrstnega reda v tej razvrstitvi. Verjetnostno distribucijo določimo na podlagi rangov po formuli:

$$p_i = \frac{r_i}{\sum_{j=1}^{n} r_j}$$

Na podlagi distribucije osebke vzorčimo.

Turnirska izbira Deluje tako, da med naključno izbranimi osebki iz populacije potekajo izbirni turnirji. Zmagovalec vsakega turnirja oz. tisti z najboljšo izbirno oceno je izbran za reprodukcijo. Metoda je preprosto prilagodljiva, saj izbiro prilagajamo z večanjem in manjšanjem velikosti turnirja -

večji je turnir, manjša je verjetnost da bodo izbrani osebki s slabšimi ocenami kakovosti in obratno.

Izpis 2.3: Psevdokoda turnirske izbire

```
choose: t (size of tournament) random individuals from current
    generation
choose: best individual of current tournament with probability p
choose: second best individual with probability p * (1 - p)
choose: third best individual with probability p * ((1 - p)^2)
etc.
```

Če je velikost turnirja $t = 1$ je izbira ekvivalentna naključni izbiri osebka iz populacije. Dobre lastnosti te metode so učinkovitost pri kodiranju, paralelno delovanje na več sistemih in preprosto prilagajanje metode s spremembo velikosti turnirja.

2.2.3 Reprodukcija

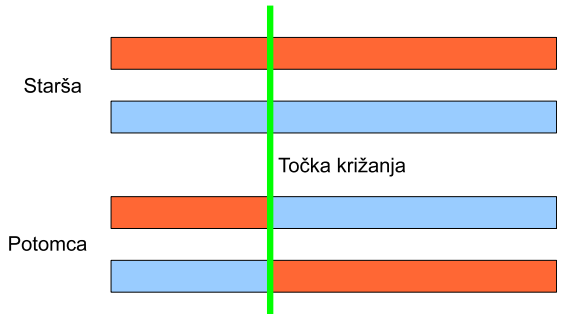
Ko imamo izbrane pare osebkov (v nadaljevanju starše) iz njih generiramo novo generacijo osebkov (v nadaljevanju potomcu). To izvedemo s transformacijama znanima iz genetike: križanjem ali rekombinacijo in mutacijo.

Križanje V genetskem zapisu se izbere točka (ali več točk). Naslednika sta sestavljena tako, da je prvi potomec sestavljen iz niza genov prvega starša od začetka do konca točke križanja in iz niza genov drugega starša od točke križanja do konca. Drugi potomec je sestavljen iz niza genov drugega starša od začetka do konca točke križanja in iz niza genov prvega starša od točke križanja do konca. Opisan postopek eno točkovnega križanja je prikazan na sliki 2.2, dvo točkovnega pa na sliki 2.3.

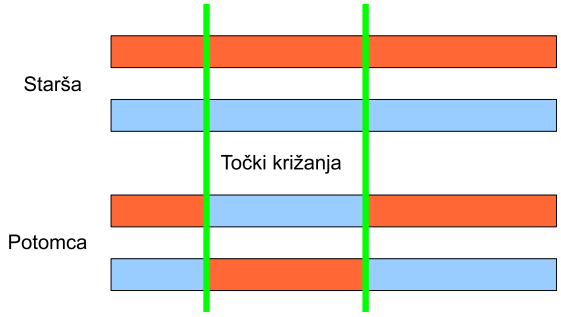
Podoben je postopek pri več točkovnem križanju, kjer se ob vsaki točki križanja zamenja del gena staršev, ki prispeva gene nasledniku. Križanje se izvede glede na verjetnost križanja P_c in sicer $n \times P_c$ krat.

Mutacija Mutacija omogoča ohranjanje genetske raznolikosti med generacijami. To je podobno kot pri biološki mutaciji. Vsak gen v populaciji se z majhno verjetnostjo spremeni. Zaradi majhne verjetnosti, mutacija običajno ne vpliva bistveno na uspešnost genetskega algoritma.

Mutacija se izvaja po reprodukciji na novo generiranih osebkih, običajno z nizko verjetnostjo. Poznamo različne tipe mutacij: inverzija, mutacija s



Slika 2.2: Prikaz eno točkovnega križanja.



Slika 2.3: Prikaz dvo točkovnega križanja.

plavajočo vejico, permutacije, zamenjave,...

Posebna vrsta mutacije je adaptivna mutacija, pri kateri verjetnost izvedbe naraste na podlagi neizboljšanja rezultatov (ocene kakovosti se ne izboljšajo) zadnjih n generacij.

Elitizem Je prenos določenega števila najboljših osebkov v naslednjo generacijo. Ti osebki se preslikajo v naslednjo generacijo, pri čemer se izognejo mutacijam in niso potomci križanih osebkov predhodne generacije. Slaba lastnost tega pristopa je možnost prezgodnje konvergence, če elita prevlada [1].

2.2.4 Ustavitev genetskega algoritma

Generacijski procesi (selekcija, križanje, reprodukcija) se ponavljajo, dokler ni dosežen ustavitveni pogoj.

Ustavitveni pogoji so [3]:

- rezultat izpolnjuje minimalne kriterije,
- doseženo je bilo maksimalno vnaprej določeno število generacij,
- dosežena je bila najboljša ocena kakovosti ali zaporedje zadnjih n generacij ne dosega boljših rezultatov,
- ročna ustavitev,
- kombinacije zgoraj naštetih pogojev.

2.3 Uporaba genetskih algoritmov

Genetski algoritmi so zaradi stohastičnosti in hevrističnega preiskovanja primerni za reševanje problemov na različih področjih. Pri problemih, kjer optimalne rešitve ni mogoče izračunati zaradi prevelikega ali celo neskončnega prostora stanj, genetski algoritmi nudijo kvalitetne približne rešitve. Spodaj so opisana področja, kjer genetske algoritme najpogosteje uporabljamo.

Avtomobilsko oblikovanje Pri avtomobilskem oblikovanju se genetski algoritmi uporabljajo za oblikovanje kompozitnih materialov aerodinamičnih oblik avtomobilov za dirke, letala in druga prevozna sredstva. Kombinacije materialov in oblike zagotavlja hitrost in optimalno težo prevoznega sredstva ter učinkovitejšo porabe goriva. Genetski algoritmi nadomestijo leta laboratorjskih raziskav polimerov in testov v vetrovnikih, ker se ta procesa hitreje in učinkoviteje modelira z uporabo genetskih algoritmov. Rezultat je nabor rešitev in možnosti, ki jih človeški oblikovalci uporabijo pri zasnovi končnih rešitev.

Inženirsko oblikovanje Genetski algoritmi so učinkoviti za probleme na področju inženirskega oblikovanja kot npr. oblikovati boljše strojne mehanske dele, kvalitetnejše materiale, za optimizacijo delovnih procesov strojev v tovarnah itd. Uporaba sega tudi v optimizacijo prenosa toplote na mehanskih delih, določanja razmikov satelitov v vesolju, izboljšanju uporabe turbin itd. Genetskih algoritmov ne uporabljamo samo za reševanje problemov, ampak tudi za analizo produktov, odkrivanje slabosti in morebitnih napak.

Robotika Robotika je področje kjer ljudje naredijo stroj, ki opravlja naloge namesto človeka. Genetski algoritmi služijo kot orodje pri učenju robotov, da svoje naloge opravljajo učinkoviteje in hitreje.

Evolucijski razvoj strojne opreme (Evolvable Hardware) Genetski algoritem se uporablja pri razvoju strojne opreme tako, da ustvarimo računalniški model strojne opreme, na katerem genetski algoritmi z uporabo stohastičnih operatorjev optimizirajo delovanje obstoječe ali razvijejo nove konfiguracije strojne opreme. Slednje je še posebej zanimivo, saj lahko morda pričakujemo stroje oz stojno opremo, ki se bo sposobna sama prilagajati razmeram (vremenskim, okoljskim, itd.) in imela možnosti samo popravil.

Optimizacija telekomunikacijskih linij Na področju telekomunikacij se genetski algoritmi uporabljajo za optimizacijo namestitev in usmerjanja telekomunikacijskih baznih postaj za kar najboljšo pokritost. Trenutno je v fazi razvoja sistem preusmerjanja telekomunikacijskih poti v primeru preobremenjenih ali nestabilnih linij. Rezultat učinkovitejši telekomunikacijski sistem.

Generator šal in besednih iger Na jezikovnem področju genetski algoritmi služijo kot učinkovito orodje pri generiranju šal in besednih iger. Manj spretni stand-up komedianti bi se lahko poslužili takšnih generatorjev. Generiranje deluje tako, da genetski algoritmi omogočajo vnos poljubne besede, ki jo oseba želi za temo besedne igre ali šale, algoritem pa vrne različne rešitve.

Preiskovanje poti Preiskovanje NP-polnih problemov rešujemo z genetskimi algoritmi. Za NP-polne probleme najverjetneje ne obstaja algoritem, ki bi našel optimalno rešitev v polinomskem času. Eden izmed takšnih je problem trgovskega potnika (Traveling Salesman Problem - TSP) za katerega velja da imamo dan neusmerjen graf, kjer ima vsaka povezava določeno dolžino/ceno. Poiskati je potrebno najkrajši cikel v grafu, ki gre skozi vsa vozlišča grafa natanko enkrat. Takšna pot se imenuje Hamiltonov cikel. TSP srečamo pri načrtovanju učinkovitih poti v prometu, transportu, raznašanju pošiljk itd. V tej vlogi so genetski algoritmi uspešni, saj lahko upoštevajo obremenitve na poteh, vremenske razmere itd. V to zvrst genetskih algoritmov spada tudi simulacija v tem diplomskem delu.

Računalniške igre Na področju računalniških iger imajo genetski algoritmi najpogosteje vlogo nasprotnika igralcu. Genetski algoritmi so sposobni učenja

uspešnih strategij iz prejšnjih iger. V strateških zvrsteh računalniških igrah je uporaba genetskega algoritma razširjena.

Enkripcija in dekripcija gesel Pri varnosti se genetski algoritmi uporabljajo tako pri ustvarjanju varnostnih ključev (enkripcija) kot pri razvozlavanju varnostnih ključev (dekripcija). Ta način uporabe tisti, ki neprestano nadgrajujejo in dopolnjujejo poskušajo dešifrirati zaščite avtorskih digitalnih del.

Uporaba genetskih algoritmov je pogosta še pri [4]:

- računalniški podpori molekularnemu oblikovanju,
- biometrični izumi,
- genskem profiliranju,
- optimizaciji kemijskih kinetičnih analiz,
- finančnih in vlagateljskih strategij,
- trženju in prodaji.

Poglavje 3

Demonstracija delovanja genetskih algoritmov

V tem poglavju najprej opišemo problem, na kateremu demonstriramo delovanje genetskega algoritma. Opišemo osebke ter parametre, ki jih uporabnik lahko spreminja, na koncu pa še Bezierjeve krivulje, ki določajo način premikanja osebkov.

3.1 Opis problema

Za vizualizacijo GA uporabljamo preiskovanje poti. Takšno zvrst problema sem si izbral, ker sem ocenil, da bo najboljša za izdelavo prepričljive simulacije. Izdelal sem dve različici problema, dvodimenzionalno in tridimenzionalno. Obe delujeta po enakem principu, razlikuje se le preiskovalni prostor, ki je v prvem primeru kvadrat, v drugem primeru pa kocka. Elementi simulacije so osebki, ovire in cilj.

Osebki So osnovne entitete GA, njihovo njihovo število v simulaciji je parameter in sicer od 2 do 100 osebkov. Vsak osebek je vizualno predstavljen s kroglo, ta se premika po svoji Bezierjevi krivulji, ki je podrobneje opisana kasneje. Osnovni parametri vsakega osebka so:

- Bezierjeva krivulja, ki predstavlja pot po kateri osebek potuje,
- oddaljenost od cilja, ki se izračuna na podlagi prepotovane poti po formulah $d_{2D} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ za dvodimenzionalno različico

in $d_{3D} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ za tridimenzionalno različico,

- hitrost premikanja osebka po poti,
- genski zapis, ki je shranjen v obliki tabele točk. Na podlagi teh točk se izračuna Bezierjeva krivulja, točke pa služijo za prenos genskega zapisa v bodoče generacije.

Ovire so v dvodimenzionalni različici predstavljene kot pravokotniki, v tridimenzionalni različici pa kot kvadri. Število ovir je poljubno, velikost vsake ovire pa ne sme presegat velikosti preiskovalnega prostora. Vloga ovire je zaustavitev osebka, čigar pot oz. Bezierjeva krivulja seka oviro.

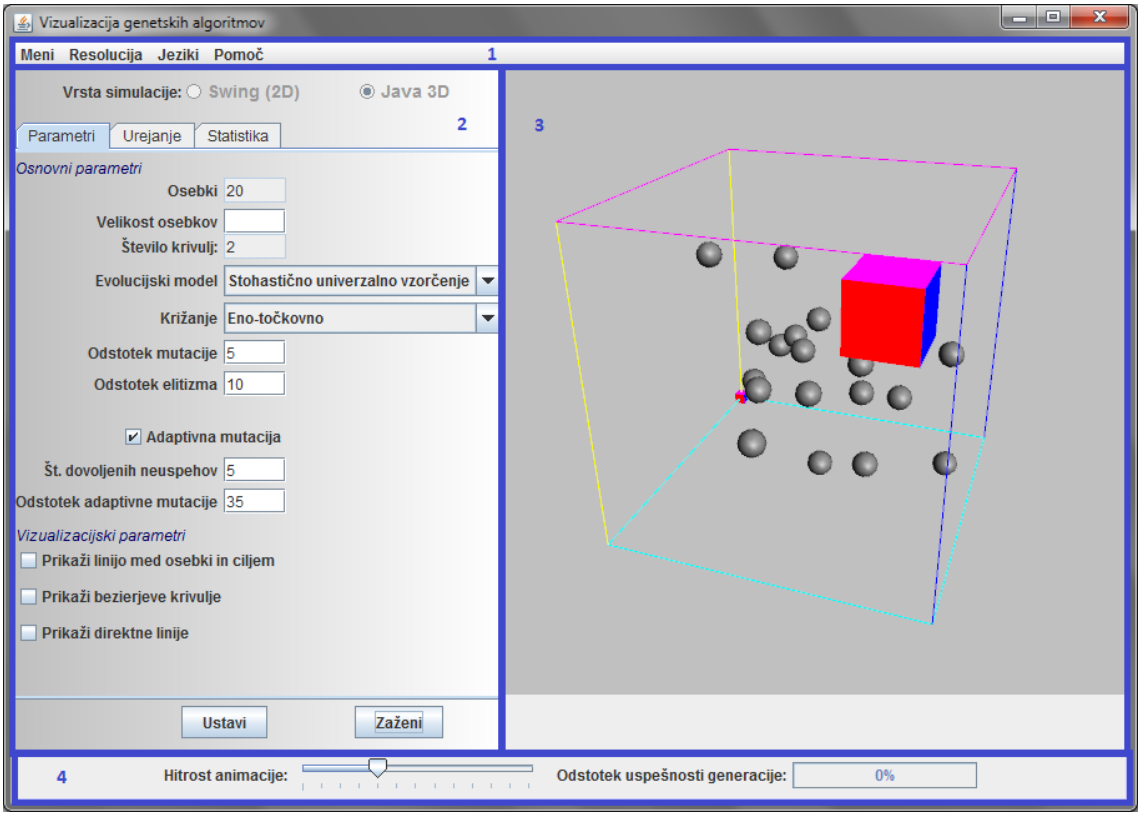
Cilj je natanko eden in ima obliko kvadrata v dvodimenzionalni ter, obliko kocke v tridimenzionalni različici. Krajša kot je oddaljenost poti osebka od središča cilja, boljša je ocena kakovosti osebka.

Iz zgoraj opisanih elementov se definicija problema glasi:
Množica osebkov želi iz skupne začetne točke A pripotovati v točko B (središče cilja). Med točkama so postavljene ovire, ki osebkom otežujejo prehod iz točke A v B, saj se osebek v primeru trka ustavi. Z GA želimo naučiti vse osebke, da dosežejo cilj, ne da bi se zaleteli v ovire.

3.2 Uporabniški vmesnik in parametri

Za zgoraj opisan problem sem pripravil Java aplikacijo z uporabniškem vmesnikom (ang. Grafic User Interface - GUI). Uporabniški vmesnik se v osnovi deli na štiri komponente, označene na sliki 3.1, in sicer:

- **osnovni meni**, ki se deli na kategorije *Meni*, *Resolucija*, *Jezik* in *Pomoč*,
- **funkcijski meni**, kjer uporabnik določa vse parametre simulacije in ima vpogled v statistiko simulacije v obliki dveh grafov,
- **simulacija**, ki prikazuje vizualizacijo GA v realnem času,
- **noga simulacije**, ki omogoča spreminjanje hitrosti simulacije in prikazuje uspešnost trenutne generacije.



Slika 3.1: Na sliki so označene štiri osnovne komponente aplikacije.

Osnovni meni Tu uporabnik prilagaja funkcije glede na lastne potrebe. Na voljo so:

- novo, ki prekine izvajajočo se simulacijo, ter omogoča uporabniku ponoven zagon simulacije,
- naloži konfiguracijo, ki omogoča uprabniku pripravo konfiguracije ovir in cilja iz XML datoteke. Primer takšne XML datoteke je v dodatku,
- shrani konfiguracijo, ki omogoča uporabniku shranit trenutno konfiguracijo postavitev ovir in cilja,
- izhod, ki zapre aplikacijo.

V izbiri *Resolucija* uporabnik spreminja ločljivost simulacije. Prevzeta ločljivost je 500x500, najvišja ločljivost pa 1500x1500. Preklop jezikov je uporabniku na voljo v kategoriji *Jeziki*. Na izbiro ima slovenski ali angleški jezik. V kategoriji

Pomoč so uporabniku na voljo informacije o uporabi aplikacije in informacije o avtorju.

Funkcijski meni Izberemo lahko tip simulacije (dvodimenzionalno ali tridimenzionalno), meni funkcij (iz treh zavihkov in gumboma za zagon) in zaustavitev simulacije.
Zavihek *parametri* omogoča spreminjanje parametrov osebkov in prikaza simulacije. Parametri osebkov so število osebkov, velikost osebkov, število Bezierjevih krivulj, izbira evolucijskega modela, vrste križanja, deleža mutacij in elitizma ter vključitev adaptivne mutacije. Na voljo sta še dva vizualizacijska parametra in sicer prikaz poti v obliki povezanih daljic in prikaz Bezierjevih poti.
Zavihek *urejanje* omogoča dodajanje, urejanje in brisanje ovir ter postavitev cilja in začetne točke izvajanja.
Zavihek *statistika* služi za prikaz grafa uspešnosti po generacijah (od 0 do 100 odstotkov) in grafa povprečne oddaljenosti od cilja osebkov vsake generacije od cilja.

Simulacija vizualizira dvodimenzionalno in tridimenzionalno simulacijo GA. Znotraj dvodimenzionalne različice je omogočeno urejanje ovir. Tridimenzionalna različica urejanje ovir ne omogoča, zaradi tretje dimenzije in posledično nepreglednosti urejanja. Omogoča pa rotacijo preiskovalnega prostora in vsebine (osebkov, ovir in cilja) okoli y osi s tipkama A (rotacija v levo) in D (rotacija v desno).

3.3 Delovanje genetskega algoritma v simulaciji

Delovanje simulacije poteka preko vseh osnovnih faz GA opisanih v prejšnjem poglavju.

Inicializacija Pri zagonu se generira število osebkov, ki jih uporabnik določi kot parameter. Vsakemu osebku se naključno pripišejo 4 točke. V dvodimenzionalni različici se prva bezierjeva krivulja generira naključno glede na velikost preiskovalnega prostora, npr. $r = [500, 500]$ po formuli:

$$T_i(x_i, y_i) = \{(x_i, y_i); x_i \in [0, r], y_i \in [0, r], i \in [1, 4]\}$$

Tridimenzionalna različica zgenerira točke z istim faktorjem preiskovalnega prostora r, z dodatno komponento z:

$$T_i(x_i, y_i, z_i) = \{(x_i, y_i, z_i); x_i \in [0, r], y_i \in [0, r], z_i \in [0, r], i \in [1, 4]\}$$

Vsaka naslednja krivulja v obeh različicah potrebuje tri točke, ker za prvo točko izbere zadnjo točko prejšnje krivulje. S tem se ohranja zveznost celotne poti bezierjevih krivulj.

Selekcija Po zaključku osebkov trenutne generacije se osebkom izračuna najkrajša oddaljenost od cilja. Na podlagi enega izmed štirih mogočih evolucijskih modelov, podrobneje opisanih v prejšnjem poglavju, se vsakemu osebku določi število potomcev.

Reprodukcija Pri osebkih, ki imajo število potomcev večje od 0, izvedemo reprodukcijo med naključno izbranimi pari osebkov iz generacije. Na voljo je možnost izbire enotočkovnega križanja ali dvotočkovnega križanja. Genski zapis osebkov so točke opisane v fazi inicializacije. Po križanju se na potomcih izvedejo mutacije genskega zapisa. Če je vključen elitizem, se delež osebkov prenese v naslednjo generacijo, ne da bi bili podvrženi križanju in mutacijam. Pri prevzeto nastavljenim petkratnem popolnem neuspehu generacije (nobeden osebek ni dosegel cilja) se sproži adaptivna mutacija, kar pomeni da se verjetnost mutacije poveča. Parameter *število neuspehov generacij* do sprožitve aktivacije adaptivne mutacije lahko spreminjamo.

Ustavitev genetskega algoritma Na voljo sta dva ustavitvena pogoja. Prvi pogoj je, da vsi osebki dosežejo cilj, drugi pogoj pa je ročna ustavitev uporabnika.

3.4 Bezierjeve krivulje

Bezierjeve (ang. Bézier) krivulje so parametrične krivulje, ki jih uporabljamo v računalniški grafiki in sorodnih področjih. Prvič so bile omenjene leta 1962, v publikacijah francoskega inženirja Pierra Bézierja kot orodje za avtomobilsko oblikovanje. Njihova uporaba je pogosta tudi v današnjih vektorskih grafičnih programih, saj jih uporablja,p kot model gladke krivulje in je mogoče skaliranje. Znane so štiri vrste Bezierjevih krivulj:

- linearna (ang. linear),

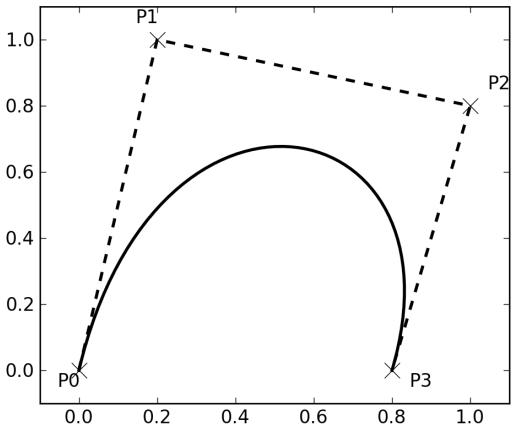
- kvadratna (ang. quadratic),
- kubična (ang. cubic),
- kvartna (ang. quartic).

Izmed zgornjih štirih vrst sem se v obeh različicah simulacije (dvodimenzionalni in tridimenzionalni) odločil za kubično Bezierjevo krivuljo, zato jo v nadaljevanju podrobneje opisujem.

3.4.1 Kubična Bezierjeva krivulja

Krivuljo opredeljujejo štiri točke P_0, P_1, P_2 in P_3 v ravnini ali v tridimenzionalnem prostoru. Krivulja se začne v točki P_0, nadaljuje pot v točko P_1 ter nato še v P_2 in konča v točki P_3. Če obstaja skupna premica, ki gre skozi vse štiri točke, potem je Bezierjeva krivulja daljica in seka točke. Ko skupna premica ne obstaja, se krivulja začne v točki P_0 in konča v točki P_3, točki P_1 in P_2 pa zagotavljata smerne informacije. Na sliki 3.2 je prikazan primer krivulje. Kubično krivuljo opisuje parametrična formula [12]:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3; t \in [0,1]$$



Slika 3.2: Slika prikazuje Bezierjevo krivuljo s točkami P_0, P_1, P_2 in P_3 [5]. Krivulja se začne v točki P_0 in konča v točki P_3. Razdalja med P_0 in P_1 določa dolžino poteka krivulje proti točki P_2, razdalja med P_1 in P_2 pa določa dolžino poteka krivulje proti končni točki P_3.

Poglavje 4

Uporabljene tehnologije

Najprej opišemo javo - osnovni programski jezik, v katerem je aplikacija napisana, nato programsko orodje Java 3D API, ki omogoča izdelavo tridimenzionalnih animacij in simulacij, razvojno okolje (ang. IDE) IntelliJ Idea, orodje za izdelavo grafičnih vmesnikov FormDev JFormDesigner, orodje za gradnjo java projektov Apache Maven ter nazadnje orodje za nadzorom nad izvorno kodo TortoiseSVN.

4.1 Java

Java je programski jezik, ki ga je leta 1995 razvil James Gosling iz podjetja Sun Microsystems. Od aprila 2009 je Sun Microsystems podružnica podjetja Oracle. Java spada v skupino visokonivojskih programskih jezikov, značilnost je objektna usmerjenost programiranja. Večji del sintakse pa izhaja iz programskih jezikov C in C++, vendar vsebuje java preprostejši objektni model in manj nizkonivojskih ukazov. Aplikacije napisane v programskem jeziku java so običajno sestavljene iz prevedenih razredov v binarni obliki (ang. bytecode), ki lahko delujejo na katerem koli javanskem navideznem stroju (ang. Java Virtual Machine - JVM).
Prednosti programskega jezika java so [7]:

- večnitnost, ki omogoča izvajanja več (ne)odvisnih operacij istočasno,
- omogoča učinkovito postavitev aplikacij in storitev (npr. spletnih storitev) na strežnike,
- omogoča varen dostop do sistemskih virov,

- omogoča razvijalcem razvoj aplikacij z visokimi zmogljivosti v realnem času in širokim dostopom do sistemskih zmogljivosti,
- razširjenost na 7 milijard elektronskih naprav vključno z mobilnimi, računalniškimi sistemi, televizijami in drugimi vgrajenimi napravami.

Slika 4.1 prikazuje logotip programskega jezika [6].

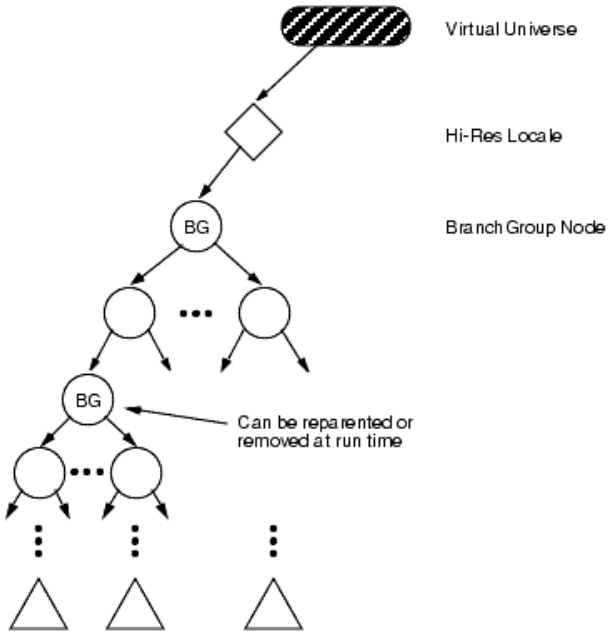
Slika 4.1: Logotip programskega jezika Java.

4.2 Java 3D

Za izdelavo tridimenzionalnih animacij ali simulacij v programskem jeziku java je na voljo programski vmesnik (ang. Application Programming Interface - API), ki teče nad grafičnim vmesnikom OpenGL ali na Direct3D. Od različice 1.2 je bil razvit v okviru skupnosti Java Community Process. Java 3D enkapsulira grafično programiranje z uporabo realnih objektno orientiranih konceptov. Animacije oz grafične scene so strukturirane v obliki drevesa, kjer posamezne veje vsebuje predmete in lastnosti predmetov, ki jih animacija prikazuje. Na sliki 4.2 prikazujemo drevesno strukturo z osnovnimi gradniki.

4.3 JetBrains IntelliJ Idea

Za razvoj aplikacij v programskem jeziku Java razvijalec potrebuje primerno razvojno okolje (ang. Integrated Development Environment - IDE). Pri razvoju aplikacije za diplomsko delo sem se odločil za IntelliJ Idea, ki je eno izmed najbolj razširjenih razvojnih okolij za programski jezik java. Razvojno okolje je plačljivo, nudi pa hitrost okolja, omogoča veliko predlog in dober vpogled v programsko kodo, povezljivost (npr. s spodaj omenjenim Apache Maven), skratka poveča storilnost [9].



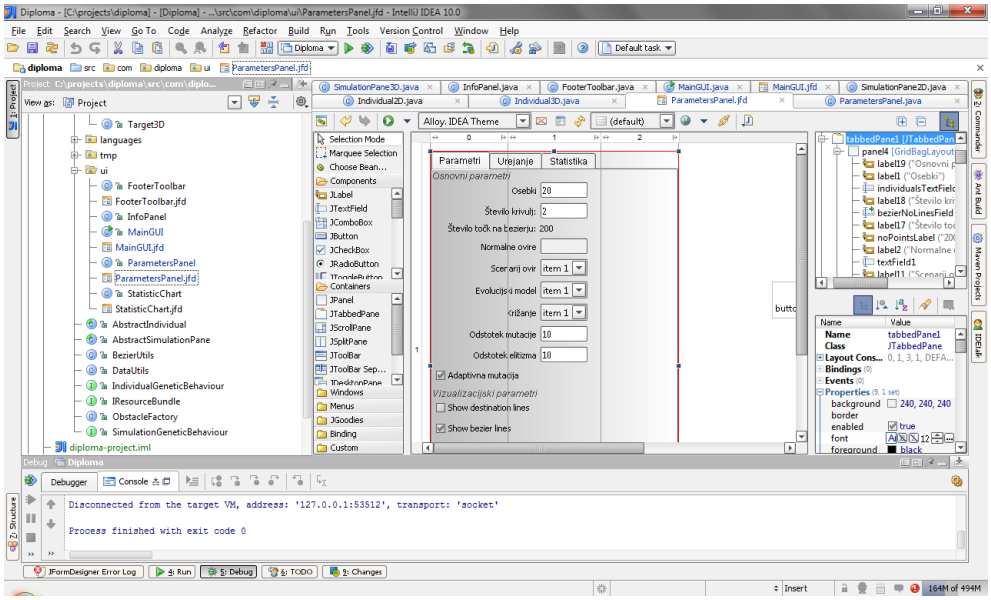
Slika 4.2: Drevesna struktura ponazarja sestavo osnovnih gradnikov vmesnika, ki je osnova za vsako animacijo Java 3D [8].

4.4 FormDev JFormDesigner

Pri zasnovi uporabniškega vmesnika aplikacije v programskem jeziku java lahko razvijalec piše programske kodo ali pa uporabi namensko oblikovalsko orodje za pisanje uporabniških vmesnikov. Odločil sem se za profesionalno orodje JFormDesinger. Uporabil sem ga kot vtič (ang. plugin) razvojnega okolja IntelliJ Idea. V osnovnem uporabniškem vmesniku ima razvijalec na voljo paleto razpoložljivih ali lastnih gradnikov, delovno površino, ki prikazuje postavitev gradnikov in približen izgled uporabniškega vmesnika, omogoča vpogled nad komponentami vsebovanimi v razvijajoč uporabniški vmesnik in določanje vrednosti in lastnosti nad komponentami. Bistveni prednosti orodja sta poenostavljeno delo s postavitvami gradnikov in fleksibilnost pri vzdrževanju uporabniškega vmesnika [10]. Slika 4.3 prikazuje orodje JFormDesigner.

4.5 Apache Maven

Maven je programsko orodje, ki se uporablja za gradnjo in upravljanje s projekti zasnovanimi v programskem jeziku Java. Glavni cilji, ki jih razvijalec



Slika 4.3: Na sliki je prikazano razvojno orodje IntelliJ Idea, ki ima vgrajen vtič JFormDesigner.

orodja Maven, želi doseči so [14]:

- poenostaviti gradnjo projekta,
- zagotoviti uniformnost gradnje projekta,
- zagotoviti kvalitetne podatke o projektu,
- zagotoviti pregleden prehod na nove funkcije in nadgradnje.

Princip gradnje projekta z orodjem Maven temelji na projektno objektnem modelu (ang. Project Object Model - POM) in naboru vtičev, ki so vgrajeni že v samem orodju. Nastavitve projekta so shranjene v enem ali več projektno objektnih modelov in sicer v datotekah pom.xml. Razvijalec ima vpogled v različico projekta, shranjene so odvisnosti med moduli oz. knjižnicami, omogoča nadzor nad JUnit testi in drugo. V tem delu je Maven uporabljen zato, da hrani odvisnosti med knjižnicami (Timing Framework in Java 3D) in omogoča gradnjo aplikacije v izvršljivo javansko arhivsko (ang. Java Archive - JAR) datoteko. Več o implementaciji orodja Maven je opisano v poglavju 5.

4.6 TortoiseSVN

Pri razvoju aplikacije sem uporabil nadzor nad izvorno kodo z uporabo orodja TortoiseSVN. Orodje deluje na platformno neodvisnem strežniku Apache Subversion. Glavni značilnosti orodja sta vpogled v zgodovino programske kode in verzije projekta [11]. Z vpogledom v zgodovino imamo na voljo pregled razvoja aplikacije, avtorjev posameznih segmentov kode, sprememb kode in možnost uporabe poljubnih starejših verzij v primeru, da smo v novi verziji kodo pokvarili. Vodenje inačic projekta poteka v povezavi z orodjem Apache Maven.

Poglavje 5

Aplikacija

V tem poglavju najprej opišemo implementacijo, nato knjižnico Timing framework kot orodje za izdelavo animacij v realnem času, implementacijo Java 3D arhitekture, na koncu pa še Maven kot orodje za gradnjo aplikacije.

5.1 Načrt implementacije

Za izdelavo simulacije GA smo potrebovali programski jezik, ki je objektno usmerjen, omogoča večnitnost in omogoča izdelavo tridimenzionalnih animacij. Programski jezik Java izpopolnjuje vse zgoraj naštete pogoje, poleg tega je na voljo kvalitetna podpora preko spleta. Celotno simulacijo smo razbil na manjše podprobleme, ki smo jih reševal po zaporedju:

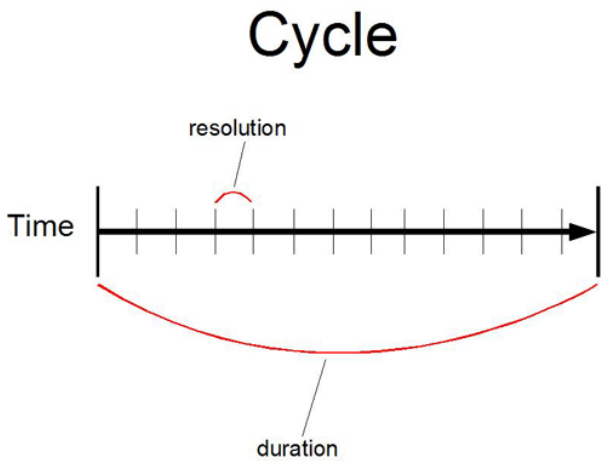
- implementacija uporabniškega grafičnega vmesnika,
- zasnova osnovne arhitekture aplikacije in objektov simulacije (osebki, ovire, cilj),
- implementacija večnitnosti z vmesnikom Timing framework,
- implementacija tridimenzionalne simulacije v Java 3D,
- implementacija Maven sistema gradnje aplikacije.

5.2 Implementacija uporabniškega grafičnega vmesnika

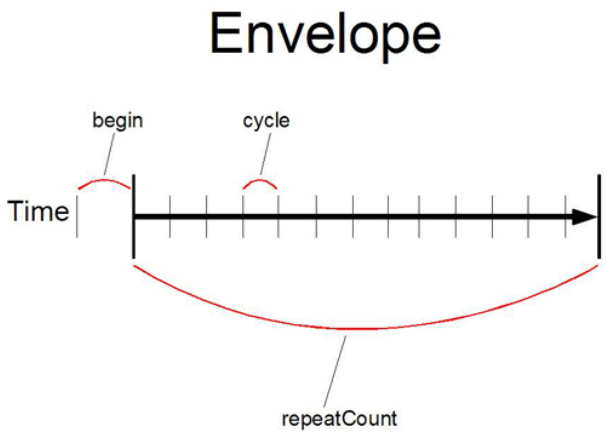
Aplikacijo bi lahko izdelali tudi brez uporabniškega vmesnika, kjer bi vse ukaze izvajal preko ukazne vrstice. Odločili smo se za uporabniški grafični vmesnik (Graphical User Interface - GUI) zato, da bi uporabniku poenostavil uporabo in omogočil boljši vpogled v izbrane parametre ter ukaze. Za implementacijo smo se poslužil orodja JFormDesigner kar olajša izdelavo grafičnega vmesnika in omogoča lažje vzdrževanje. Orodje omogoča poenostavljeno delo z javanskimi razporejevalniki (ang. layout manager) in gradniki. Aplikacijo smo razdelili na štiri osnovne grafične gradnike, njihova uporaba in delovanje sta opisani v 3. poglavju.

5.3 Implementacija vmesnika Timing framework

Aplikacije v javi prevzeto tečejo z eno nitjo. Če želimo izvajanje več stavkov istočasno, se to izkaže kot ovira. Potrebna je večnitnost, ki je iz osnovne knjižnice poznana kot razred Thread. Pri implementaciji moje rešitve bi se lahko poslužil tega razreda, vendar bi bila programska koda manj optimalna, manj pregledna in implementacija bi bila otežena. Zato smo uporabili knjižnico Timing framework, namensko orodje za izdelavo animacij in časovnih dogodkov. Razvoj knjižnice poteka v odprtokodnem projektu skupnosti java.net. Večnitni časovni model je predstavljen in opisan na sliki 5.1, razširjen večnitni časovni model predstavljen kot časovna ovojnica z vsebujočimi cikli pa je predstavljen in opisan na sliki 5.2.



Slika 5.1: Večnitni časovni model je definiran z obveznim končnim ali neskončnim trajanjem in neobveznimi nitnimi ločljivostmi (ang. resolution). V primeru, da so nitne ločljivosti definirane, se te, glede na velikost periode, izvajajo kot neodvisne niti znotraj glavne niti [13].



Slika 5.2: Večnitna časovna ovojnica pomeni periodično izvajanje večnitnega časovnega modela s končnim trajanjem. Pri zagonu večnitne časovne ovojnice se prvič izvede inicializacija, ob prekinitvi časovne ovojnice se po zaključku vseh niti izvede še metoda zaključitve [13].

Pri implementaciji dvodimenzionalne rešitve smo uporabili časovni model nad gradnikom simulacije, ki služi za periodično osveževanje slike animacije in periodično preverja stanje osebkov. Inicializacija in selekcija osebka, se izvaja tako, da je vsakemu osebku dodeljen svoj časovni model.
Implementacija tridimenzionalne rešitve deluje kot dvodimenzionalna z razliko, da se osveževanje slike, izvaja neodvisno nad vsakim osebkom. Drugačna implementacija je potrebna zaradi arhitekture vmesnika Java 3D.

5.4 Java 3D

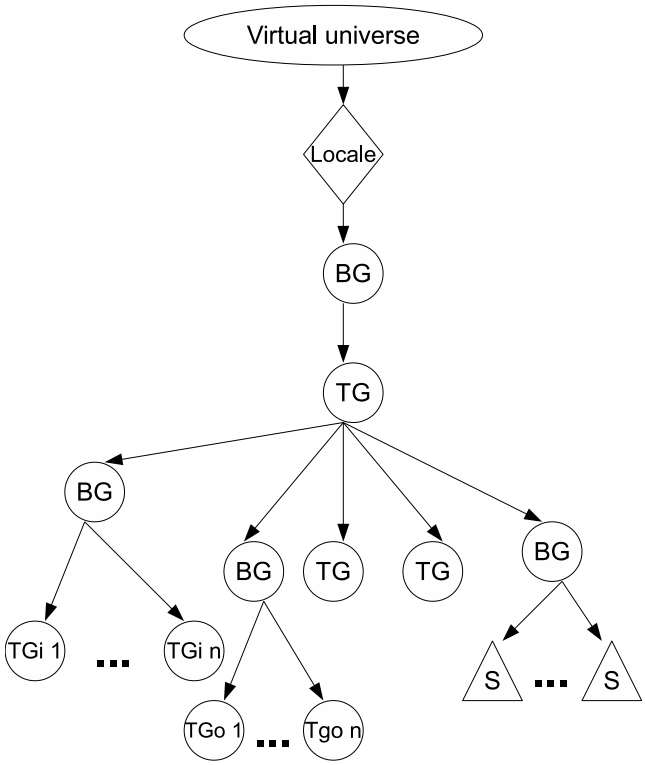
Pri implementaciji tridimenzionalne različice simulacije smo morali upoštevati arhitekturo Java 3D in lastnosti gradnikov. Na težave smo naleteli pri dodajanju in odstranjevanju Java 3D gradnikov na živi (ang. compiled) animaciji. Živa animacija namreč ne dopušča dodajanja ali odstranjevanja gradnikov tipa TransformGroup. Za te namene smo uporabili gradnik BranchGroup. Implementacija arhitekture za mojo rešitev je prikazana in opisana na sliki 5.3.

5.5 Maven

Z orodjem Maven smo pripravili POM datoteko, ki ima nalogi:

- osnovni aplikaciji dodati odvisnosti od zunanjih knjižnic Timing framework in Java 3D,
- gradnjo izvršljive JAR datoteke.

Vsebina Maven datoteke pom.xml je dostopna v dodatku A.1.



Slika 5.3: Na sliki je prikazana arhitektura tridimenzionalne animacije. Izhodišče animacije je Virtual universe, ki ima vezane lastnosti oz parametre preko komponente Locale. Vsaka Java 3D animacija ima primarno osnovno vejo BranchGroup (v nadaljevanju vejitvena grupa), ki ima pomembno vlogo, saj dopušča spreminjanje vsebine oz. spreminjati vsebino grupe. Ko je vejitvena grupa prevedena (ang. compiled), prikaže animacijo in avtomatično se prevedejo tudi vsebujoče komponente v tej veji. Pri gradnji arhitekture animacije smo na osnovno vejo vezali komponento TransformGroup (v nadaljevanju transformacijska grupa), ki omogoča transformacije. Uporabili smo rotacijo okoli preiskovalnega prostora tridimenzionalne animacije. Dodali smo grupi za cilj in za začetno točko, ki prikazuje izhodišče osebkov. Osebke smo vizualizirali tako, da smo na transformacijsko grupo vezali vejitveno grupo, ki vsebuje toliko transformacijskih grup kolikor je osebkov v animaciji. Podobno smo naredili z ovirami in prikazom bezierjevih krivulj, le da smo pri slednjem uporabili preprostejšo komponento Shape3D, ki transformacij ne dopušča.

Poglavje 6

Vizualizacija rešitve

V tem poglavju predstavimo štiri konfiguracije delovanje GA v aplikaciji na dvodimenzionalni in tridimenzionalni različici. Pri vsakemu problemu predstavimo dobljene rezultate, pri težjih problemih pa preučimo možnosti izboljšav GA s spreminjanjem parametrov med samo vizualizacijo.

6.1 Dvodimenzionalna vizualizacija

6.1.1 Primer lažjega problema

V lažjo vizualizacijo dvodimenzionalne rešitve GA v aplikaciji smo vključili dve oviri z dimenzijami:

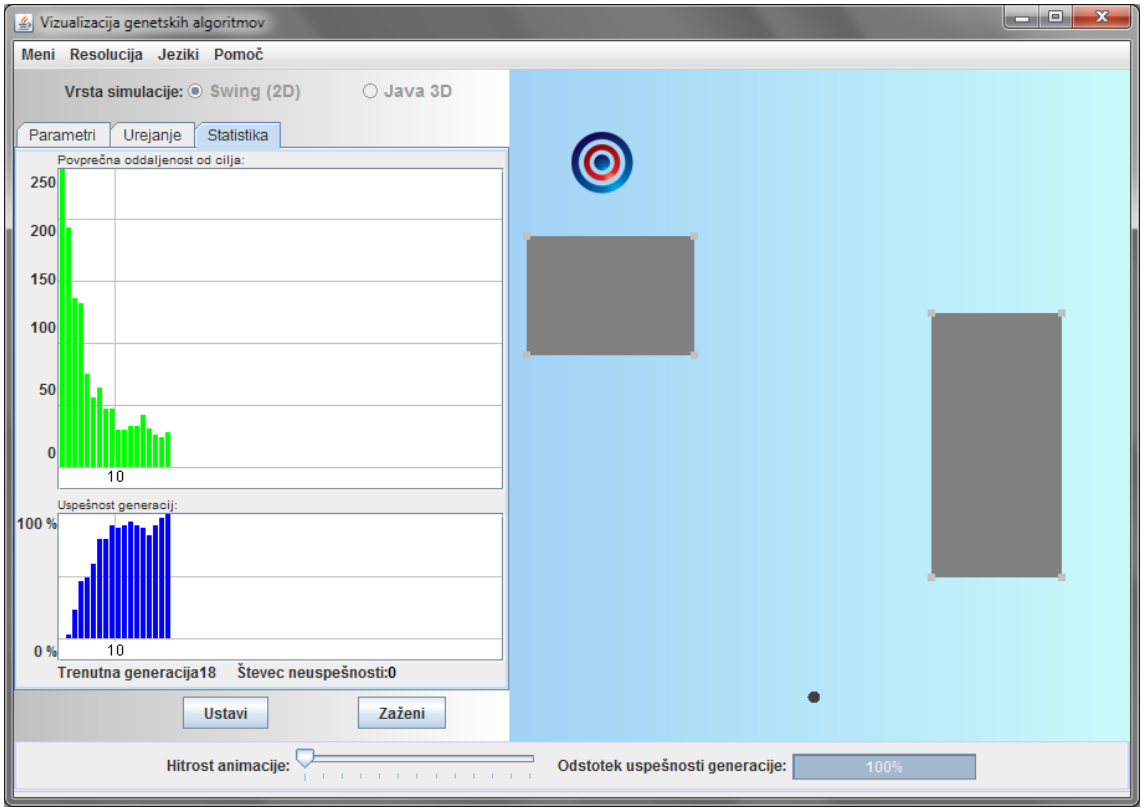
- 1. ovira, dolžina = 50 pikslov, širina = 70 pikslov,
- 2. ovira, dolžina = 100 pikslov, širina=150 pikslov.

Uporabili smo parametre, ki jih med vizualizacijo nismo spreminjali:

- 35 osebkov,
- genski zapis vsakega osebka je sestavljen iz 10 točk, ki sestavljajo 3 Bezierjeve krivulje,
- turnirska izbira z velikostjo turnirjev 4 za evolucijski model,
- dvotočkovno križanje,
- odstotek mutacije 5 %,

- odstotek elitizma 15 %,
- vključena adaptivna mutacija z 40 % verjetnostjo, ki se sproži ob 7 neuspešnih generacijah,

Cilj ima velikost stranic 50 pikslov. Postavitev konfiguracije (ovir, cilja) in rezultati posameznih generacij so vidni na sliki 6.1.



Slika 6.1: Primer vizualizacije lažjega problema. GA zastavljeni problem reši v 18 generacijah. Z grafa *uspešnosti generacij* je razvidna rast uspešnosti skozi generacije.

6.1.2 Primer težjega problema

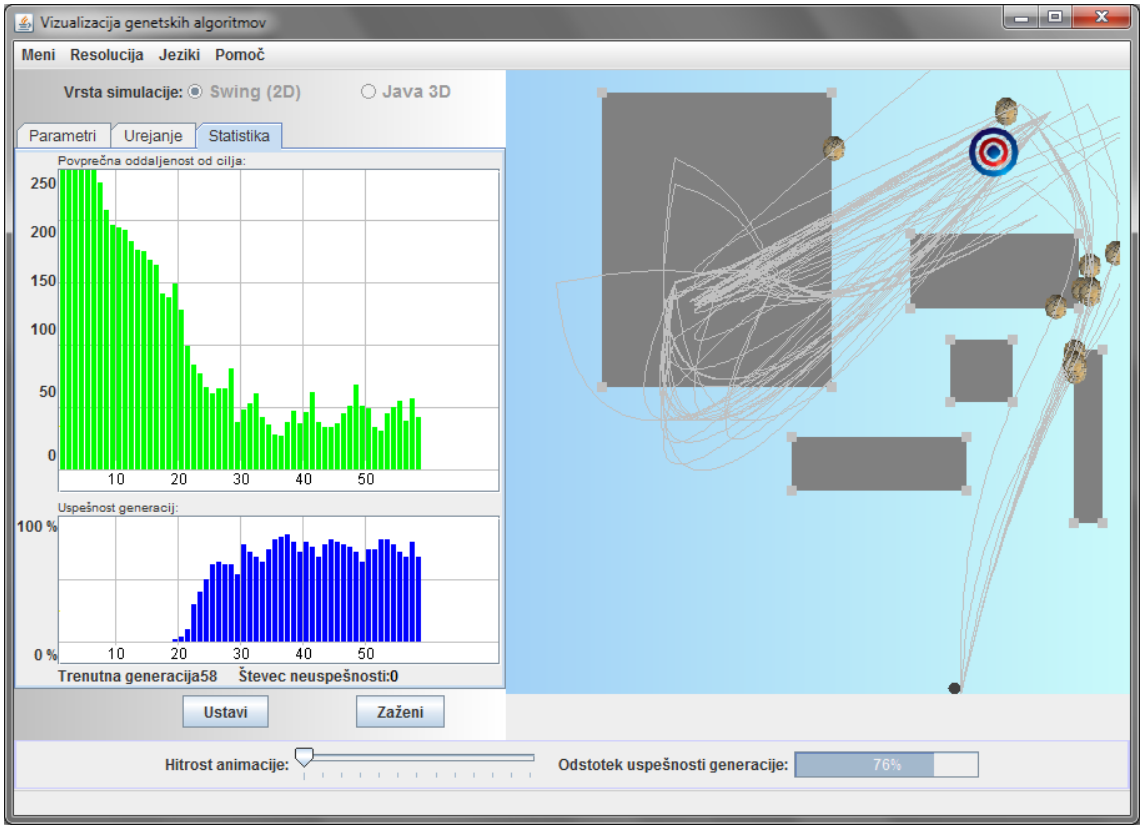
V težjo vizualizacijo dvodimenzionalne rešitve GA v aplikaciji smo vključili pet ovir z dimenzijami:

- 1. ovira, dolžina = 184 pikslov, širina = 236 pikslov,
- 2. ovira, dolžina = 135 pikslov, širina=60 pikslov.
- 3. ovira, dolžina = 140 pikslov, širina=43 pikslov.
- 4. ovira, dolžina = 23 pikslov, širina=139 pikslov.
- 5. ovira, dolžina = 50 pikslov, širina=50 pikslov.

Uporabili smo parametre:

- 50 osebkov,
- genski zapis vsakega osebka je sestavljen iz 13 točk, ki sestavljajo 4 Bezierjeve krivulje,
- proporcionalno izbiro,
- enotočkovno križanje,
- odstotek mutacije 15 %, ki smo ga med izvajanjem spreminjali do 35%,
- odstotek elitizma 10 %,

Cilj ima velikost stranic 40 pikslov. Postavitev konfiguracije (ovir, cilja) in rezultati posameznih generacij so vidni na sliki 6.2.



Slika 6.2: Primer dvodimenzionalne vizualizacije težjega problema. V tej konfiguraciji smo cilj pomanjšali, stopnjo mutacije povečali in postavili ovire okoli cilja. V 58. generaciji smo program zaustavili, rezultati pa so nihali med 70 in 80 odstotki od 26. generacije dalje. Kljub izbiri namenoma slabših parametrov, kot so visoka stopnja mutacije in izključena adaptivna mutacija, so rezultati dobri.

6.2 Tridimenzionalna vizualizacija

6.2.1 Primer lažjega problema

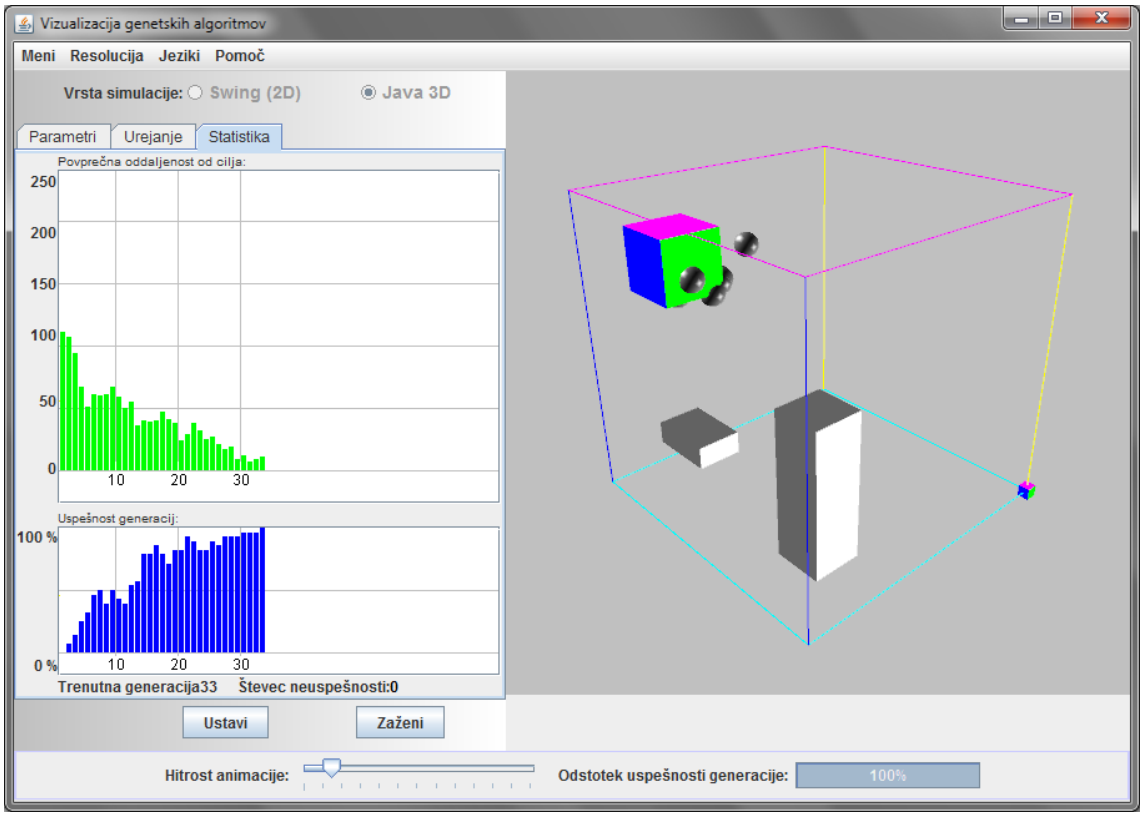
V lažjo vizualizacijo tridimenzionalne rešitve GA v aplikaciji smo vključili dve oviri z dimenzijami:

- 1. ovira, dolžina = 50 pikslov, višina = 150, širina = 50 pikslov,
- 2. ovira, dolžina = 40 pikslov, višina = 20, širina = 60 pikslov.

Uporabili smo parametre, ki jih med vizualizacijo nismo spreminjali:

- 28 osebkov,
- genski zapis vsakega osebka je sestavljen iz 10 točk, ki sestavljajo 3 Bezierjeve krivulje,
- stohastično univerzalno vzorčenje,
- dvotočkovno križanje,
- odstotek mutacije 5 %,
- odstotek elitizma 5 %,

Cilj ima velikost stranic 50 pikslov. Postavitev konfiguracije (ovir, cilja) in rezultati posameznih generacij so vidni na sliki 6.3.



Slika 6.3: Primer tridimenzionalne vizualizacije lažjega problema. GA zastavljeni problem reši v 33 generacijah. Rezultate bi lahko izboljšali tako, da bi na začetku povečali stopnjo mutacije in med samim izvajanjem večali stopnjo elitizma.

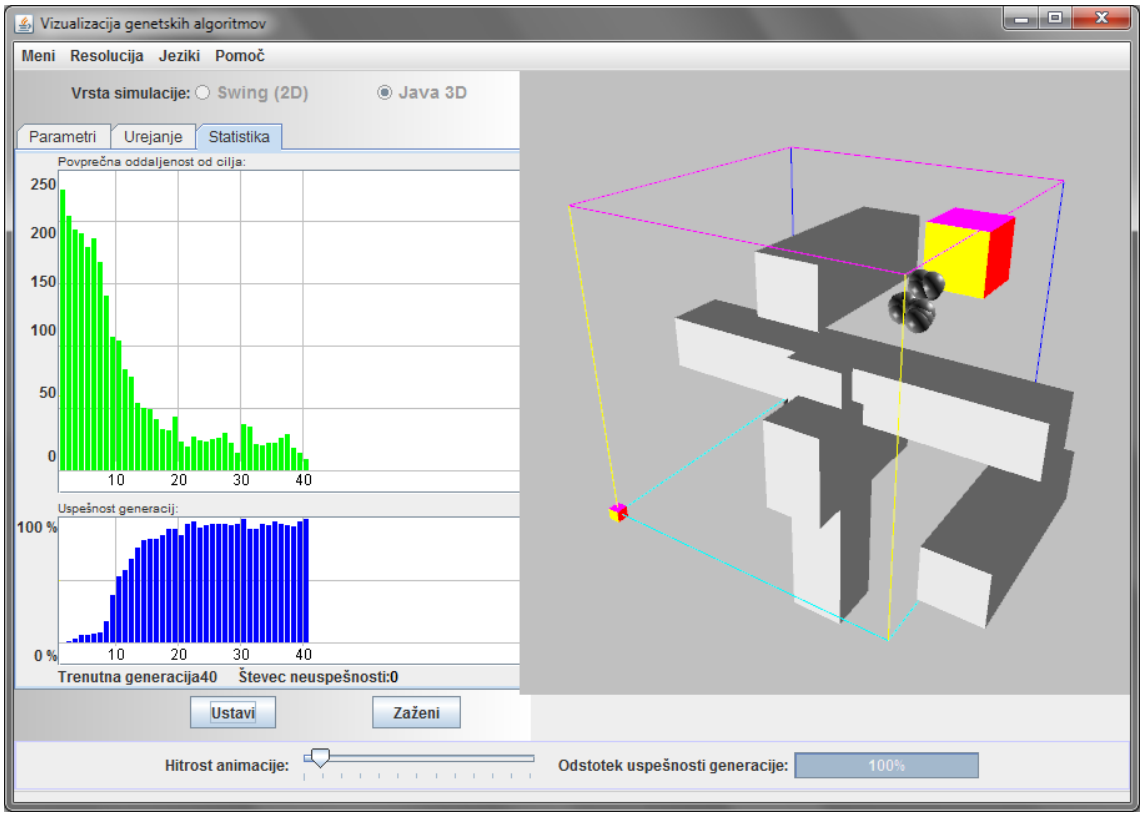
6.2.2 Primer težjega problema

V težjo vizualizacijo tridimenzionalne rešitve GA v aplikaciji smo vključili 9 ovir, ki so v obliki XML konfiguracije prikazani v dodatku A.2.

Uporabili smo parametre, ki jih med vizualizacijo nismo spreminjali:

- 88 osebkov,
- genski zapis vsakega osebka je sestavljen iz 16 točk, ki sestavljajo 5 Bezierjeve krivulje,
- proporcionalno izbiro,
- dvotočkovno križanje,
- odstotek mutacije 20 %,
- odstotek elitizma 28 %,
- vključena adaptivna mutacija z 45 % verjetnostjo, ki se sproži ob 9 neuspešnih generacijah,

Cilj ima velikost stranic 50 pikslov. Postavitev konfiguracije (ovir, cilja) in rezultati posameznih generacij so vidni na sliki 6.3.



Slika 6.4: Primer tridimenzionalne vizualizacije težjega problema. Parametre smo prilagodili konfiguraciji tako, da smo za težji problem reševanja izbrali večjo množico osebkov, višjo stopnjo elitizma in adaptivno mutacijo. Pri reševanju problema se je GA dosegel uspeh celotne populacije v 40. generaciji.

Poglavje 7

Sklep

V diplomski nalogi smo predstavili delovanje GA in razvili aplikacijo, ki vizualizira delovanje algoritma v dvodimenzionalni in tridimenzionalni različici, na preprostem preiskovalnem problemu. Preučili smo različne načine delovanja posameznih faz GA in jih implementirali kot parametre. Rezultat tega je aplikacija, ki služi kot študijski pripomoček za lažje razumevanje delovanja GA, in omogoča spreminjanje parametrov algoritma med izvajanjem. Demonstracija reševanja problemov z GA je podrobneje prikazana in opisana v šestem poglavju.

Aplikacijo smo razvili v programskem jeziku java v razvojnem okolju IntelliJ Idea, pri čemer smo uporabili knjižnico Swing za dvodimenzionalno različico in knjižnico Java 3D API za izdelavo tridimenzionalne različice. Algoritmično različici delujeta na enak način, saj uporabljata skupne metode v fazah GA, arhitekturno pa se precej razlikujeta. Swing omogoča preprosto dodajanje in odstranjevanje elementov animacije, Java 3d pa pri izdelavi animacije zgradi drevesno strukturo grup, ki ima v živi animaciji določene omejitve. Interaktivnost aplikacije smo omogočili z implementacijo knjižnice Timing Framework, ki je izboljšan sistem večnitnosti in je namenjen pripravi animacij. Uporabniški vmesnik smo izdelali z orodjem JFormDesigner, v procesu implementacije smo uporabili tudi sistem za nadzor nad izvorno kodo z orodjem TortoiseSVN. Gradnjo aplikacije v izvršljivo JAR datoteko smo pripravli z orodjem Maven, ki je obenem poskrbelo, da so bile v datoteko JAR vključene vse potrebne knjižnice (Timing Framework, Java3D API). Možnosti nadaljnjega dela, predvsem pri nadgradnji aplikacije, je veliko. Aplikacijo bi lahko vsebinsko nadgradil z novimi uporabniškimi funkcionalnostmi in izboljšal nekatere tehnične podrobnosti, ki so ostale nepopolno implementirane. Lahko bi implementiral nove elemente z različnimi vplivi na osebke (npr. implementacijo ovire,

ki ob trku osebkov odbijejo osebke) in tako pripravil kompleksnejši problem. Ena izmed zanimivih možnosti je uporaba obstoječega ogrodja aplikacije za implementacijo drugih evolucijskih algoritmov in primerjavo pri reševanju zastavljenega problema.

Literatura

[1] I.Kononenko in M. Robnik Šikonja, *Inteligentni sistemi*, FRI, Ljubljana, Slovenija, 1. izdaja, 2010.

[2] Vijay V. Vazirani, "Approximation Algorithms,",Springer-Verlang Berlin Heidelberg New York ,2003

[3] (2011) Genetic algorithms. Dostopno na: http://en.wikipedia.org/wiki/Genetic_algorithm

[4] (2011) Uporaba genetskih algoritmov. Dostopno na: http://brainz.org/15-real-world-applications-genetic-algorithms

[5] (2011) Example of cubic Bézier curve. Dostopno na: http://matplotlib.sourceforge.net/users/path_tutorial.html

[6] (2011) Java programming language. Dostopno na: http://www.oracle.com/us/technologies/java/index.html

[7] (2011) Java 3D programming language. Dostopno na: http://en.wikipedia.org/wiki/Java_3D

[8] (2011) Java 3D group nodes. Dostopno na: http://www-evasion.imag.fr/ Francois.Faure/enseignement/ ressources/java/j3dguide/GroupNodes.doc.html

[9] (2011) JetBrains IntelliJ Idea. Dostopno na: http://www.jetbrains.com/idea/

[10] (2011) FormDev JFormDesigner. Dostopno na: http://www.formdev.com/jformdesigner/

[11] (2011) TortoiseSVN. Dostopno na: http://tortoisesvn.net/

[12] (2011) Bezier curve. Dostopno na:
http://en.wikipedia.org/wiki/B

[13] (2011) Timing Framework Dostopno na:
http://java.sun.com/developer/technicalArticles/Media/timing/index.html

[14] (2011) Apache Maven Dostopno na:
http://en.wikipedia.org/wiki/Apache_Maven

Dodatek A

Implementacijske rešitve

Priloge (slike, diagrami, algoritmi, načrti), če so potrebne, kandidat izdela kot posebna poglavja (Dodatek A, Dodatek B, ...), ki jih zaradi preglednosti ni smiselno vključiti v glavni del naloge. Vsi dodatki morajo biti naslovljeni in oštevilčeni, običajno z velikimi tiskanimi črkami.

Izpis A.1: Izvorna koda Maven diplomske datoteke pom.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0h
ttp://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.diploma</groupId>
    <artifactId>diploma-project</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <dependencies>
        <!--project source code dependency-->
        <dependency>
            <groupId>com.diploma</groupId>
            <artifactId>diploma-project</artifactId>
            <version>1.0</version>
        </dependency>

        <!--Timing Framework dependency-->
        <dependency>
            <groupId>net.java.dev.timingframework</groupId>
```

```
        <artifactId>timingframework</artifactId>
        <version>1.0</version>
    </dependency>

    <!--java 3D dependency-->
    <dependency>
        <groupId>java3d</groupId>
        <artifactId>j3d-core</artifactId>
        <version>1.3.1</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>2.2-beta-2</version>
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>com.diploma.ui.MainGUI</mainClass>
                        <addClasspath>true</addClasspath>
                    </manifest>
                </archive>
                <descriptorRefs>
                    <descriptorRef>jar-with-dependencies</descriptorRef>
                    <descriptorRef>src</descriptorRef>
                </descriptorRefs>
            </configuration>
            <executions>
                <execution>
                    <id>attach-assembly-to-package</id>
```

```
                    <phase>install</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
</project>
```

Izpis A.2: Primer konfiguracije tridimenzionalne simulacije.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration_2d>
    <obstacles_3d>
        <x>0.0</x>
        <y>0.0</y>
        <z>0.0</z>
        <width>0.5</width>
        <height>2.5</height>
    </obstacles_3d>
    <obstacles_3d>
        <x>0.0</x>
        <y>-1.5</y>
        <z>0.0</z>
        <width>0.5</width>
        <height>0.4</height>
    </obstacles_3d>
    <obstacles_3d>
        <x>2.66</x>
        <y>0.54</y>
        <z>0.44</z>
        <width>0.5</width>
        <height>0.4</height>
    </obstacles_3d>
    <obstacles_3d>
        <x>0.0</x>
        <y>0.54</y>
        <z>3.0</z>
        <width>1.7</width>
```

```
        <height>0.46</height>
    </obstacles_3d>
    <obstacles_3d>
        <x>0.0</x>
        <y>0.77</y>
        <z>0.2</z>
        <width>1.4</width>
        <height>0.46</height>
    </obstacles_3d>
    <obstacles_3d>
        <x>0.0</x>
        <y>1.44</y>
        <z>0.2</z>
        <width>1.4</width>
        <height>0.46</height>
    </obstacles_3d>
    <obstacles_3d>
        <x>0.0</x>
        <y>4.0</y>
        <z>0.2</z>
        <width>1.4</width>
        <height>0.46</height>
    </obstacles_3d>
    <obstacles_3d>
        <x>0.27</x>
        <y>4.0</y>
        <z>0.2</z>
        <width>0.53</width>
        <height>0.46</height>
    </obstacles_3d>
    <obstacles_3d>
        <x>0.34</x>
        <y>2.0</y>
        <z>0.2</z>
        <width>0.53</width>
        <height>0.46</height>
    </obstacles_3d>
</configuration_3d>
```