SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Razvoj A* heuristike genetskim programiranjem

Edi Smoljan

Voditelj: *prof. dr. sc. Domagoj Jakobović*

Zagreb, svibanj, 2014.

Sadržaj

1. Uvod

1.1 Opis problema

Mnogo se praktičnih problema u računarstvu može prikazati strukturom grafa. Problemi koji uključuju pronalazak najboljeg slijeda stanja od nekog početnog do skupa završnih čvorova (engl. *path finding*) operiraju nad grafovima s nekim dodatnim atributima poput smjera bridova grafa i cijene prijelaza iz jednog čvora u drugi po nekom bridu. Rješenje ovog problema podrazumijeva pronalazak niza bridova od početnog čvora (stanja) do ciljnog čvora takvih da im je zbroj težina najmanji mogući. Rješenja ovog problema često nalaze primjenu u GPS uređajima i izradi umjetne inteligencije za računalne igre.

Razmatrani algoritmi koji rješavaju ovaj problem zahtijevaju da težine u usmjerenom grafu budu nenegativne. Jedan od najpoznatijih algoritama koji može poslužiti za rješavanje ovog problema je Dijkstrin algoritam. Taj algoritam je prvotno zamišljen za pronalaženje najkraćeg puta od početnog čvora do svih čvorova u grafu, no može se modificirati tako da se zaustavi jednom kad dođe do ciljnog čvora. Dijkstrin algoritam pri računanju prijelaza iz trenutnog stanja (čvora grafa) u sljedeće u obzir uzima samo cijenu prijelaza između ta dva stanja. Iako uvijek pronalazi najbolji put od početnog do ciljnog stanja, ovaj algoritam rezultira pretraživanjem mnogo većeg broja stanja od onog što je zapravo potrebno i zbog toga se gledaju upotrijebiti algoritmi koji će pri pronalasku najboljeg puta od točke do točke pretražiti što manji prostor stanja.

A* algoritam je sličan Dijkstrinom algoritmu, no on je, za razliku od Dijkstrinog, zamišljen za pronalaženje najkraćeg puta od početnog stanja do zadanog ciljnog stanja. Pri računanju prijelaza iz nekog trenutnog stanja u sljedeće A* algoritam kombinira cijenu između ta dva stanja i procjenu udaljenosti od sljedećeg stanja do ciljnog stanja. Ta procjena je nenegativna funkcija trenutnog stanja koja se naziva heuristikom i ona na neki način modelira dodatno znanje o konkretnom problemu. Kako bi se osigurao pronalazak najboljeg puta heuristika mora zadovoljavati svojstva optimističnosti (procjena duljine preostalog puta do ciljnog stanja mora uvijek biti manja ili jednaka stvarnoj duljini do ciljnog stanja) i konzistentnosti (vrijednost

heuristike u trenutnom stanju mora biti manja ili jednaka zbroju vrijednosti heuristike u sljedećem stanju i cijeni prelaska iz trenutnog u sljedeće stanje). Ako je heuristika pesimistična, odnosno precjenjuje duljinu preostalog puta do cilja, rezultirajući put od početnog do ciljnog stanja neće biti optimalan no u tom slučaju broj iteracija može biti dosta manji. Heuristička funkcija je specifična konkretnom problemu koji se rješava A* algoritmom.

U ovom radu se A* algoritmom rješava problem pronalaska suboptimalnog puta između nasumično odabranog početnog i ciljnog stanja na nasumično stvorenoj visinskoj mapi (engl. *height map*). Često se za ovakav problem koristi heuristika euklidske udaljenosti između do ciljnog stanja i ona uvijek pronalazi najkraći put. Neku drugu heuristiku koja bi pronašla rješenje u manje iteracija je teško direktno izgraditi, a to pogotovo vrijedi za suboptimalne heuristike koje pronalaze dovoljno dobre putove uz mali broj iteracija.

Sve navedeno ukazuje na izgradnju heuristike na neki automatizirani način, npr. uz pomoć genetskog programiranja.

1.2 Programsko ostvarenje problema

Za implementaciju problema odabran je programski jezik Java. Za postupak učenja heuristike iskorištena je implementacija *Evolutionary Computation Frameworka* [1] (u daljnjem tekstu ECF) u Javi.

Heuristika se razvija na problemu pronalaska puta na nasumično generiranim visinskim mapama. Zbog toga je implementiran generator nasumičnih mapa baziran na *diamond-square* postupku generiranja nasumičnih mapa opisanom u [2]. Mapa je dvodimenzionalno polje *float* brojeva koje u ovom radu ima dimenzije $513x513$. Parametri postupka izgradnje mape su duljina početnog simetričnog intervala brojeva oko 0 iz kojeg se stvaraju nasumične vrijednosti šuma i faktor umanjenja intervala brojeva (u radu označeno s $\mathbf{2^{-H}}$) koji u svakoj iteraciji množi duljinu intervala iz kojeg se generiraju nasumične vrijednosti šuma (i time ga smanjuje ako je faktor manji od 1). U ovom radu prvi parametar ima vrijednost 1.5, a drugi 0.65 jer su se s tim parametrima generirali prihvatljivi tereni. Dodatno je moguće postaviti početne vrijednosti nekih elemenata polja i tako usmjeriti generiranje nasumičnog terena (u

ovom radu je to napravljeno tako da dobivena mapa liči na otok). Generator na izlasku daje polje float brojeva koji se dodatno normaliziraju na interval od 0 do 1 radi lakšeg baratanja s tim vrijednostima u daljnjem postupku razvoja heuristike. Zamišljeno je da se stvorene mape sastoje i od vodenih površina pa je uveden parametar visine vode koji može poprimiti vrijednost od 0 do 1 (u ovom radu 0.5) i postavljanjem istog sve se koordinate visinske mape gdje je vrijednost visine ispod zadane vrijednosti visine vode označavaju kao vodena površina. Dodatno se zaglađuju (zamućuju) vrijednosti mape visine od 0.4 do 0.7 (što bi odgovaralo nizinama uzevši u obzir prethodne parametre) Box blur postupkom ukratko opisanom u [3] uz parametar radijusa vrijednosti 2. Ovim postupkom nastaje objekt tipa *HeightMap* koji ima funkcije za dohvaćanje visine određene koordinate visinske mape i ispitivanje nalazi li se na toj koordinati voda. Slike tako dobivenih mapa se mogu vidjeti u sljedećim poglavljima.

Uz nasumične mape potrebno je bilo napraviti i skup nasumičnih početnih i ciljnih stanja. Algoritam generiranja parova nasumičnih stanja uzima u obzir ove parametre: koordinate centra oko kojeg se stanja generiraju (u ovom radu je to centar mape), minimalnu udaljenost generiranih stanja (postavljeno na $\frac{1}{2}$ širine mape- 256) i maksimalnu udaljenost generiranih stanja (postavljeno na $\frac{7}{8}$ širine mape- 448).

A* algoritam ostvaren je pomoću zasebnog razreda koji se konfigurira primajući objekt koji računa funkciju troška prelaska iz jednog stanja u drugo i objekt koji računa vrijednost heurističke funkcije za neko dano stanje. Pri završetku pretrage A* algoritam vraća ciljni čvor u kojemu je zapisana cijena konačnog puta i sam konačni put kao slijed stanja. Nakon završetka pretrage moguće je od objekta koji ostvaruje A* algoritam saznati koliko je iteracija bilo potrebno za dolazak do ciljnog stanja.

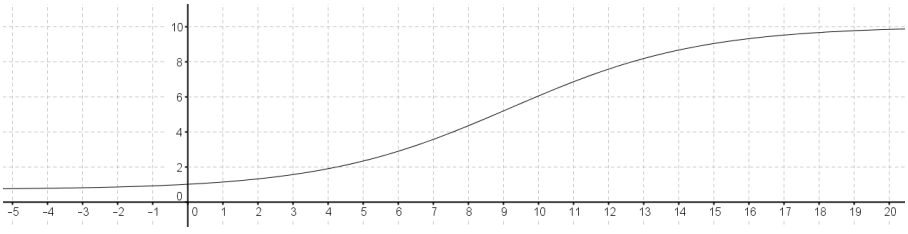
Kako bi se prethodno stvorene mape dovele u prirodni kontekst u kojem se udaljenosti mjere u metrima a ne u pikselima, odlučeno je 2D koordinate površine pomnožiti sa faktorom 10 tako da se mapa dimenzije $513\,x\,513$ može promatrati i kao površina od $5.13\,x\,5.13\,km^2$. Vertikalna dimenzija mape (prethodno normalizirana na interval od 0 do 1) je pomnožena sa faktorom 3000 tako da, uzevši u obzir kako je visina vode postavljena na 0.5, je najviši vrh na visinskoj mapi na $1500\,m$ nadmorske visine.

Problem pronalaska najkraćeg puta zahtijeva prikaz kretanja po visinskoj mapi uz pomoć stanja koja su jednostavno prikazana indeksom polja na visinskoj mapi. Po mapi je dopušteno kretanje u 8 smjerova osim u slučaju nalaženja na granici mape, odnosno skup stanja sljedbenika trenutnog stanja je veličine 8 kao što je prikazano na slici Slika 1.1: iz trenutnog stanja označenog sa brojem 5 moguće se u sljedećem koraku pomaknuti u neko od stanja označenim sa 1, 2, 3, 4, 6, 7, 8 ili 9.

1	2	3
4	5	6
7	8	9

Slika 1.1 Mogući prelasci iz trenutnog stanja

Ono što algoritam A* nastoji optimizirati je pronalazak najboljeg puta uz zadanu funkciju troška prijelaza iz jednog stanja u drugo koja u ovom radu uzima u obzir visinsku mapu i prethodno opisane faktore kojima se množe vrijednosti visinske mape. Funkcija troška u ovom radu je zamišljena na sljedeći način: ako je trenutno stanje na istoj visini kao i sljedeće vraća se euklidska udaljenost ta dva stanja tako na primjer pri prelasku iz stanja 5 u stanje 6 na slici Slika 1.1 funkcija vraća vrijednost 10, a pri prelasku iz stanja 5 u stanje 9 vrijednost 14.142. Dodatno, ako su oba stanja na vodi ta se vrijednost množi sa 2.5 čime se oponaša sporije kretanje po vodenim površinama. Ako je samo jedno od ta dva stanja na vodi onda se vrijednost množi sa 10 što oponaša sporost ulaska u vodu odnosno izlaska iz nje. U slučaju kretanja po kopnu i postojanja razlika visina trošak prelaska između dva stanja je euklidska udaljenost te dvije točke u prostoru pomnožena sa faktorom određenim funkcijom na slici Slika 1.2 koji pokušava simulirati brže kretanje po padinama (do 0.7) i sporije po uzvisinama (do 10). Brojevi na apscisi predstavljaju omjer razlike visine sljedećeg stanja i trenutnog stanja i 2D euklidske udaljenosti između ta dva stanja.



Slika 1.2 Funkcija faktora (sigmoida)

2. Opis postupka razvoja heuristike

Uz prethodno implementirane komponente visinskih mapa, početnih i završnih stanja i A* algoritma potrebno je definirati postupak razvoja (učenja) heuristike. Učenje heuristike se odvija nad jednom visinskom mapom i skupom početnih i ciljnih stanja učitanih iz datoteka. Visinska mapa i skup stanja se učitavaju iz datoteka na disku čiji je put zadan kao argument programa. Nakon učitavanja se inicijaliziraju podaci koji će se kasnije uzeti kao referenca pri procesu evaluacije jednog rješenja. Dvije trećine skupa stanja se koriste za učenje heuristike dok se jedna trećina skupa izdvaja za kasniju provjeru dobivenog skupa rješenja i izdvajanje rješenja koje na tom skupu daje najbolji rezultat. Odabrana heuristika se na kraju zapisuje u datoteku na disku serijalizacijom objekta koji predstavlja heuristiku. U ovom radu se koristio skup stanja veličine 20 početnih i ciljnih stanja.

Inicijalizacija podataka podrazumijeva da se za svaki par stanja stvore podaci koji će se koristiti prilikom evaluacije neke nove heuristike. Kako se u ovom radu rezultati uspoređuju s heuristikom euklidske udaljenosti trenutnog i ciljnog stanja definirane s (2.1) dodatni podaci za svaki par stanja su duljina puta i broj iteracija algoritma A* koje heuristika euklidske udaljenosti uzrokuje svojom primjenom u A* algoritmu.

$$\sqrt{(goalX - stateX)^2 + (goalY - stateY)^2} \qquad (2.1)$$

Samu srž procesa učenja predstavlja postupak genetskog programiranja koji je ponešto izmijenjen u odnosu na uobičajenu implementaciju koju pruža implementacija ECF-a u Javi. Kako bi se izbjegla prenaučenost heuristike ona se uči nad čitavim skupom početnih i završnih stanja na sljedeći način: svakih nekoliko generacija postavlja se novi nasumično odabrani par početnog i završnog stanja zajedno s podacima nastalima inicijalizacijom vezani uz taj par i čitava se populacija rješenja od tada na dalje evaluira nad tim parom stanja. Ispravan način učenja bi bio da se čitava populacija rješenja u jednoj iteraciji genetskog algoritma evaluira nad čitavim skupom stanja, no jedan prolazak A* algoritma je vremenski dosta skupa operacija i proces evolucije bi jako sporo napredovao. No, evaluacija nad samo jednim parom početnog i ciljnog stanja se pokazuje problematičnom: često se zna dogoditi da za samo jedan par stanja veoma jednostavne heuristike budu

dominantne nad daleko robusnijim heuristikama i proces evolucije bi u tom slučaju odabrao jednostavnu heuristiku za daljnju evoluciju što znači potencijalni gubitak dobrog rješenja. Zbog toga se radi kompromis: svako rješenje se odjednom evaluira nad slijedom od 4 parova stanja gdje je prvi par odabran nasumično. Taj izbor rješava prethodno opisan problem uz zadovoljivo vremensko trajanje jedne evaluacije.

Kako bi se dodatno ubrzao proces evaluiranja čitave populacije iskorištava se činjenica da je mnogo jedinki (rješenja) u populaciji zapravo međusobno jednako pa se uvodi mehanizam spremanja rezultata u ograničenu mapu gdje je ključ jedinka, a vrijednost ono što vrati funkcija cilja za tu jedinku. Tako se prilikom evaluacije jedne jedinke prvo provjerava postoji li zapis te jedinke u mapi i ako postoji uzima vraća se dohvaćena vrijednost iz mape. Ako ne postoji zapis u mapi jedinka se evaluira i dobivena vrijednost se sprema u mapu. Dodatno je primijećeno kako izvršavanje A* algoritma za neke heuristike traje užasno dugo pa su u A* algoritam uvedeni parametri maksimalnog broja iteracija i maksimalne duljine puta (obje postavljene na vrijednosti dva puta veće od rezultata s euklidskom heuristikom) koje označavaju kad izvršavanje treba prekinuti. Uz sve ove postupke ubrzanja evaluacije jedinke 1000 generacija algoritma u nekim složenijim slučajevima može trajati i više od jednog dana.

Učenje funkcije u ECF-u se obavlja evolucijom jedinki genotipa stabla (*Tree*). Funkcije kao operacije (čvorove stabla) mogu imati u ECF-u definirane operacije zbrajanja, oduzimanja, množenja i dijeljenja, a dodane su i operacije korjenovanja (u slučaju negativnog broja operacija vraća 0), kvadriranja i apsolutne vrijednosti broja kako bi se lakše razvila heuristika na tragu norme. Vrijednosti varijabli funkcije (listovi) ovise o konkretnom problemu koji se rješava i biti će navedene u sljedećim poglavljima. U primjeni operacija mutacije i križanja implementiranih u Java verziji ECF-a pokazalo se kako oni nisu dostatni da izvuku iz lokalnog optimuma populaciju koja zapne u njemu (to pogotovo vrijedi za operatore mutacije) pa je implementiran operator kojeg ECF koristi i koji svakih nekoliko generacija nekoliko najgorih jedinki iz populacije nadomješta potpuno novim nasumično stvorenim jedinkama i tako se simulira željeni postupak mutacije koji bi u populaciju unosio svježi genetski materijal. Kako razvijena funkcija može pri izračunavanju vrijednosti na temelju vrijednosti

zadanih varijabli dati negativnu povratnu vrijednost, a zahtjev A* algoritma je da heuristika mora biti nenegativna funkcija, ako razvijena funkcija da negativnu povratnu vrijednost heuristika vraća vrijednost 0.

Funkcija cilja koju postupak genetskog programiranja optimizira se dobiva kao kombinacija duljine puta i broja iteracija s obzirom na duljinu puta i broji stanja heuristike euklidske udaljenosti nad istim parom stanja. Dobivena duljina puta nove heuristike se dijeli s duljinom puta euklidske heuristike i od dobivenog broja se oduzima 1 pa se dobiva broj u granicama od 0 do, teoretski, $+\infty$. Dobiveni broj iteracija nove heuristike se također dijeli s brojem iteracija euklidske heuristike i od količnika se oduzima 1 i teoretski se dobiva broj u granicama od -1 do $+\infty$ gdje negativni broj znači da je uporabom te heuristike A* algoritmu bilo potrebno manje iteracija da dođe do cilja nego uporabom heuristike euklidske udaljenosti. Budući da je cilj razvijanje potencijalno suboptimalne heuristike tu suboptimalnost treba nekako izraziti. U ovom radu je to učinjeno tako da se izjednačava suboptimalan par prethodno opisanih vrijednosti omjera duljina i omjera iteracija sa parom tih vrijednosti za heuristiku euklidske udaljenosti $(0, 0)$. U ovom radu je sa heuristikom euklidske udaljenosti na taj način izjednačena heuristika koja na izlazu daje do 5% dulji put i 25% manje pretraženih stanja (par brojeva $(0.05, -0.25)$). Sve heuristike između ta dva definirana para se proglašavaju boljima i njihova vrijednost funkcije cilja je negativna (npr. za par $(0.03, -0.25)$ vrijedi da je bolji od prethodno napisanih). To je u ovom radu postignuto tako da omjer iteracija u kombiniranu funkciju cilja ulazi kao linearna funkcija, a omjer duljina kao kubna funkcija gdje obje funkcije imaju postavljene koeficijente tako da vrijedi $f(0, 0) = f(0.05, -0.25)$ i to tako da se malo veća vrijednost omjera duljina putova strogo kazni. Tako definirana funkcija cilja se postupkom genetskog programiranja minimizira.

3. Razvoj uz poznavanje koordinata trenutnog i ciljnog stanja

3.1 Opis postupka razvoja

Za razvoj heuristike koja u skupu varijabli nema varijable vrijednosti visine okoline trenutnog ili ciljnog stanja izabrana je populacija od 40 rješenja, broj iteracija genetskog algoritma je postavljen na 1000, minimalna dubina stabla funkcije na 1, maksimalna dubina na 4 do 5 u ovisnosti o broju ulaznih varijabli. Iskorišteni su svi postupci mutacije i križanja u korištenoj implementaciji ECF-a (*uniform* i *simple* križanje te *nodereplace* i *permutation* mutacija). Svakih 10 generacija se mijenja skup parova stanja za učenje i isto tako se 7 najgorih rješenja nadomješta novim nasumično stvorenim rješenjima.

Skup varijabli nad kojima se razvija funkcija se sastoji od:

- ***sx*** - x koordinata trenutnog stanja na visinskoj mapi (širina)
- ***sz*** - z koordinata trenutnog stanja na visinskoj mapi (dubina)
- ***gx*** - x koordinata ciljnog stanja na visinskoj mapi (širina)
- ***gz*** - z koordinata ciljnog stanja na visinskoj mapi (dubina)
- ***vx*** - x komponenta vektora od trenutnog stanja do ciljnog stanja (gx - sx)
- ***vz*** - z komponenta vektora od trenutnog stanja do ciljnog stanja (gz - sz)

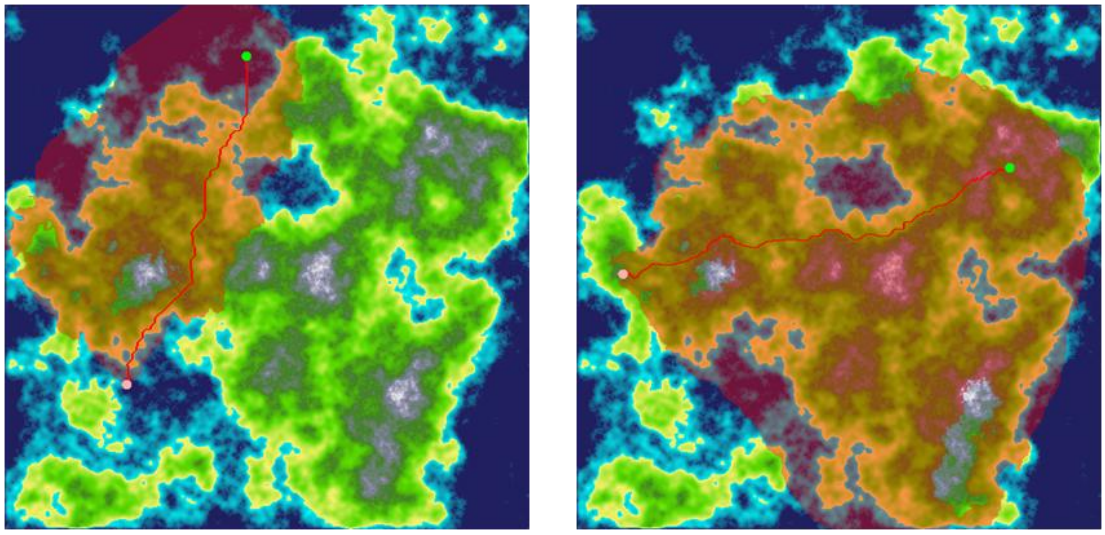
Uvođenje posljednje dvije varijable je motivirano sporošću postupka da dođe do razlike koordinata stanja koja se pokazuje izuzetno korisnom u svim naučenim heuristikama.

3.2 Prikaz rezultata

U ovom odjeljku se navode dobivene heuristike s prikazom njihovog djelovanja na prostor pretraživanja stanja A* algoritma. Uz slike su priloženi rezultati testiranja heuristika nad skupom od 30 različitih parova početnih i završnih stanja nad 3 različite mape (ukupno 90 testova po heuristici). Rezultat svakog testa jest omjer duljine puta koju daje promatrana heuristika i duljine puta koju daje referentna heuristika euklidske udaljenosti i omjer broja iteracija A* algoritma sa promatranom heuristikom i broja iteracija referentne heuristike. U ovom radu navedena je srednja vrijednost tih rezultata i njihova standardna devijacija.

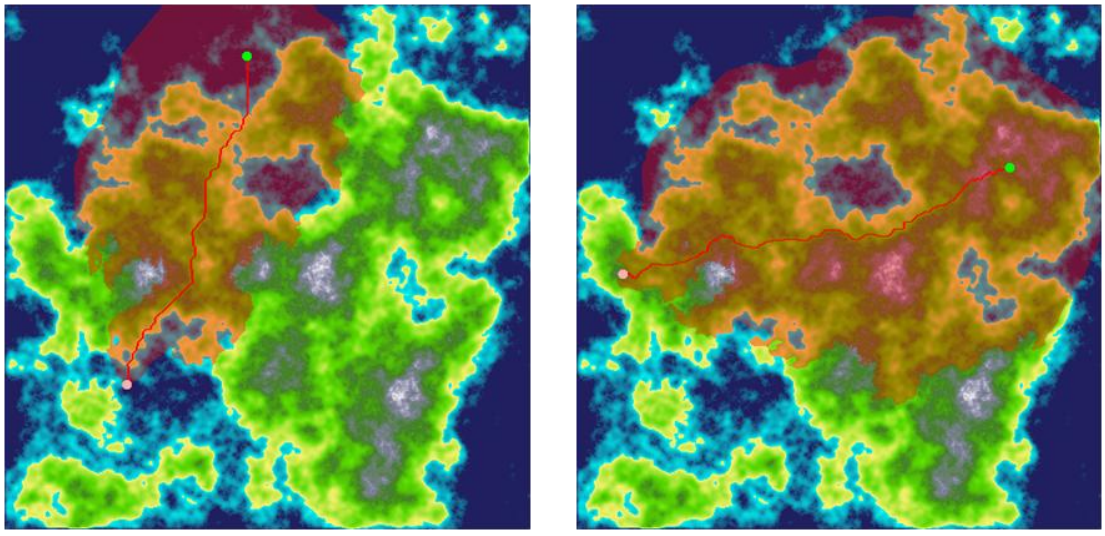
Zeleni kružić označava početno stanje, ružičasti završno stanje, neprekinuta crvena staza je pronađen put, a prozirna crvena površina označava koja su stanja posjećena u postupku pronalaska puta.

1. abs((vx-vz)) = $|\boldsymbol{vx} - \boldsymbol{vz}|$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 3.1 ima omjer duljina putova $\mu = 1.0000497$, $\sigma = 3.0580856 * 10^{-4}$ i omjer brojeva iteracija $\mu = 1.3391156$, $\sigma = 0.29359683$.



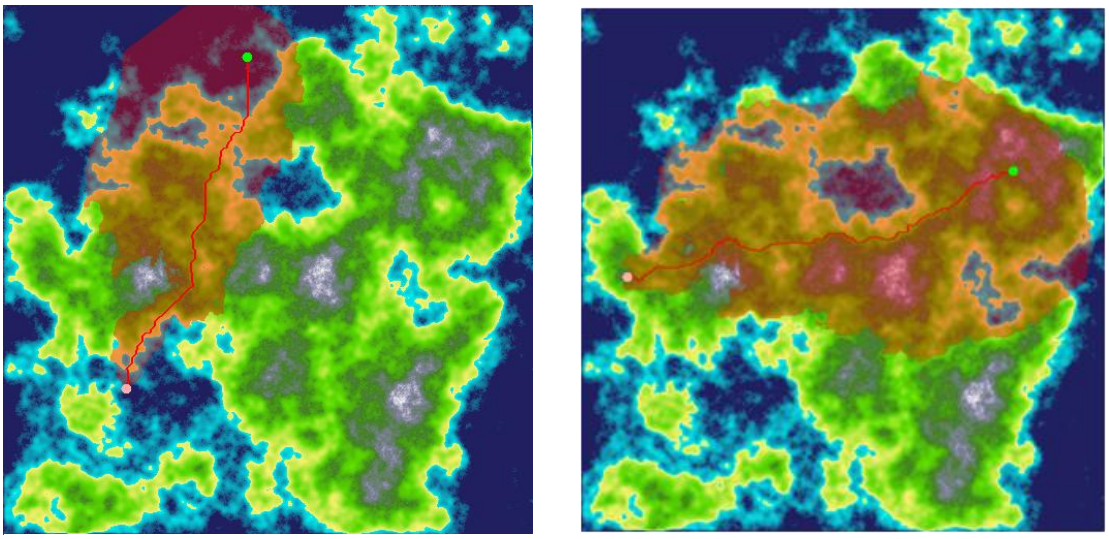
Slika 3.1 Rezultati abs((vx-vz)) heuristike

2. sqrt((sqrt(sqr(sqr(vz)))+sqrt(sqr(sqr(vx))))) = $\sqrt{\boldsymbol{vz^2} + \boldsymbol{vx^2}}$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 3.2 ima omjer duljina putova $\mu = 1.0$, $\sigma = 0.0$ i omjer brojeva iteracija $\mu = 1.0$, $\sigma = 0.0$.



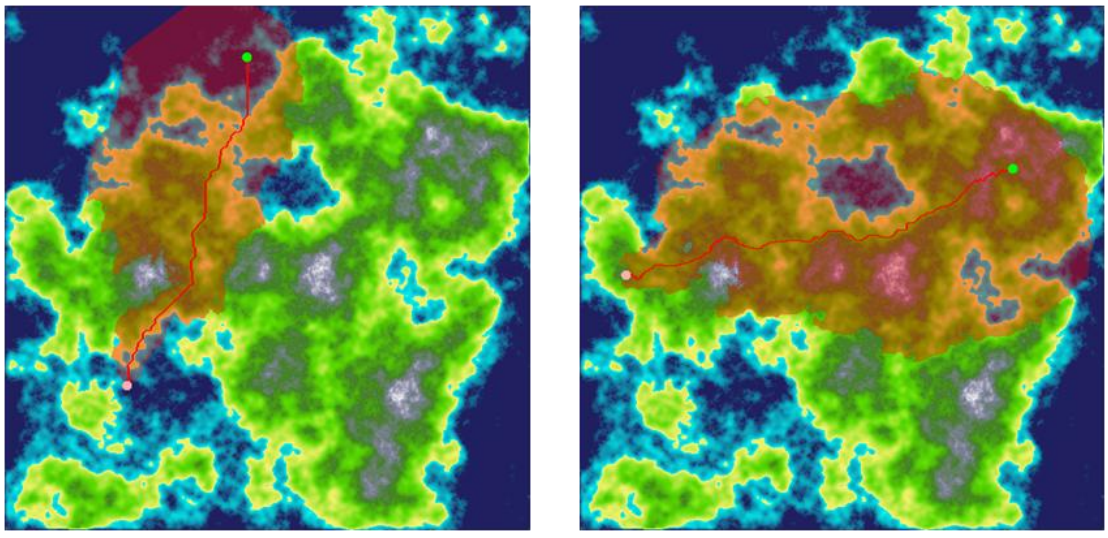
Slika 3.2 Rezultati sqrt((sqrt(sqr(sqr(vz)))+sqrt(sqr(sqr(vx))))) heuristike

3. (abs(((sz-abs(vz))-(sz-sqrt(vx))))+abs(((sz+abs(vx))+(sqrt(sx)-sz)))) = $\left|\sqrt{\boldsymbol{vx}} - |\boldsymbol{vz}|\right| + \left||\boldsymbol{vx}| + \sqrt{\boldsymbol{sx}}\right|$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 3.3 ima omjer duljina putova $\mu = 1.0000603$, $\sigma = 3.1689828 * 10^{-4}$ i omjer brojeva iteracija $\mu = 0.83603466$, $\sigma = 0.16646774$.



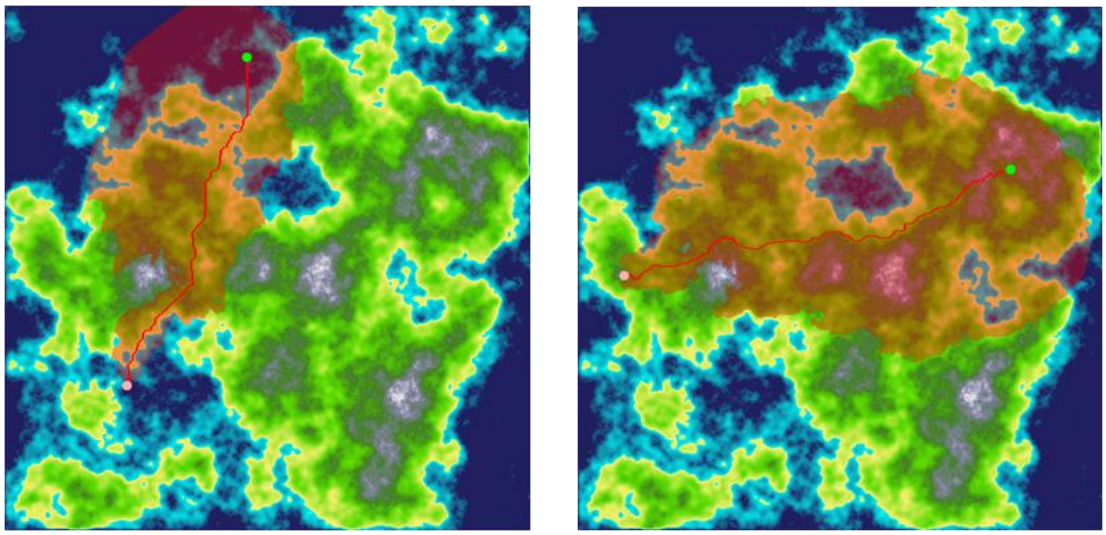
Slika 3.3 Rezultati (abs(((sz-abs(vz))-(sz-sqrt(vx))))+abs(((sz+abs(vx))+(sqrt(sx)-sz)))) heuristike

4. abs((sqrt(sqr(vz))+sqrt(sqr(vx)))) = $|\boldsymbol{vz}| + |\boldsymbol{vx}|$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 3.4 ima omjer duljina putova $\mu = 1.0000874$, $\sigma = 3.5957363 * 10^{-4}$ i omjer brojeva iteracija $\mu = 0.8353784$, $\sigma = 0.15734284$.



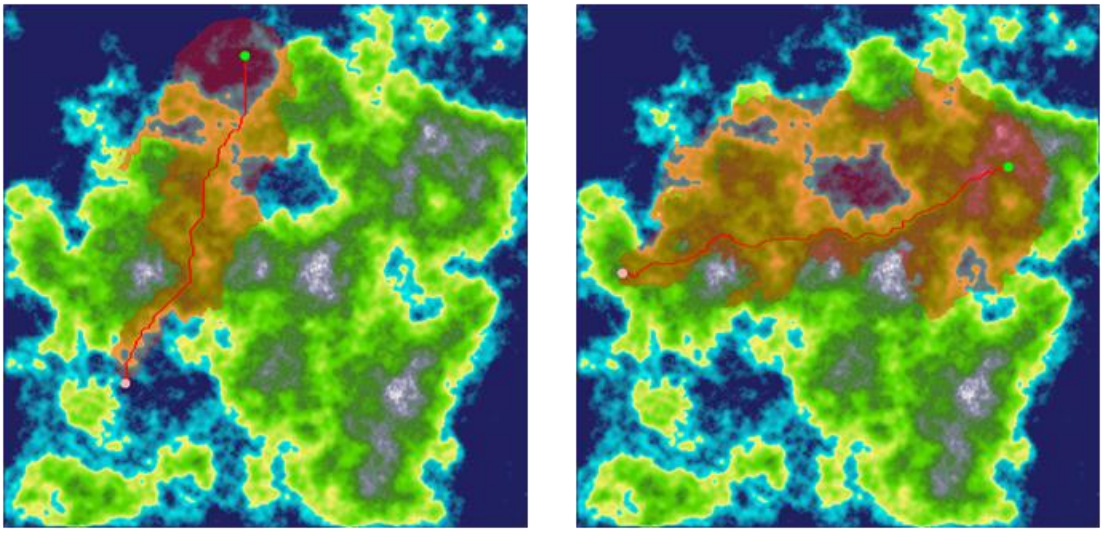
Slika 3.4 Rezultati abs((sqrt(sqr(vz))+sqrt(sqr(vx)))) heuristike

5. (abs(sqrt((abs(vx)+sqr(vz))))+abs(sqrt((abs(vz)+sqr(vx))))) = $\sqrt{|\boldsymbol{vx}| + \boldsymbol{vz^2}} + \sqrt{|\boldsymbol{vz}| + \boldsymbol{vx^2}}$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 3.5 ima omjer duljina putova $\mu = 1.0000908$, $\sigma = 3.71176 * 10^{-4}$ i omjer brojeva iteracija $\mu = 0.8259334$, $\sigma = 0.15441746$.



Slika 3.5 Rezultati (abs(sqrt((abs(vx)+sqr(vz))))+abs(sqrt((abs(vz)+sqr(vx))))) heuristike

6. sqrt(((sqr(vz)+sqr(vx))+(sqr(vx)+sqr(vz)))) = $\sqrt{\boldsymbol{2} * (\boldsymbol{vz^2} + \boldsymbol{vx^2})}$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 3.6 ima omjer duljina putova $\mu = 1.0002173$, $\sigma = 7.1628863 * 10^{-4}$ i omjer brojeva iteracija $\mu = 0.64422333$, $\sigma = 0.108440824$.



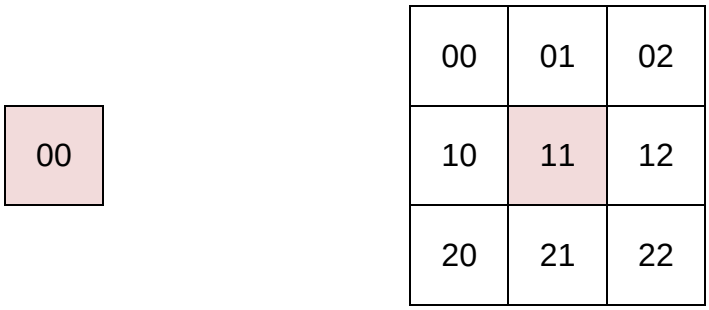
Slika 3.6 Rezultati sqrt(((sqr(vz)+sqr(vx))+(sqr(vx)+sqr(vz)))) heuristike

4. Razvoj uz dodatno poznavanje visina okoline trenutnog i ciljnog stanja

4.1 Opis postupka razvoja

U razvoju heuristike uz dodatne varijable koje predstavljaju visinu okoline stanja korištena je populacija od 50 jedinki, broj iteracija genetskog algoritma je postavljen na 1102 zato što se u 1100. generaciji stvaraju nova nasumična rješenja pa se puštaju još dvije generacije da se novi genetski materijal rasprostrani po populaciji i tako na kraju dobije skup više različitih rješenja. Minimalna dubina stabla funkcije se postavlja na 2 kako bi se spriječilo dobivanje previše jednostavnih rješenja, maksimalna dubina na 5 do 6 u ovisnosti o broju korištenih varijabli. Kao i u prethodnom načinu razvoja, iskorišteni su svi postupci mutacije i križanja u korištenoj implementaciji ECF-a (*uniform* i *simple* križanje te *nodereplace* i *permutation* mutacija). Svakih 10 generacija se mijenja skup parova stanja za učenje i isto tako se 10 najgorih rješenja nadomješta novim nasumično stvorenim rješenjima.

Indeksi u imenima varijabli koje predstavljaju visinu okoline nekog stanja postavljeni su na način prikazan na slici Slika 4.1. Na lijevoj strani slike je prikazan slučaj kad se za okolinu stanja uzme samo polje gdje se nalazi stanje, a na desnoj slučaj kad se za okolinu uzmu i polja susjedna polju na kojem se nalazi stanje (obojano).



Slika 4.1 Prikaz dodjeljivanja indeksa varijablama

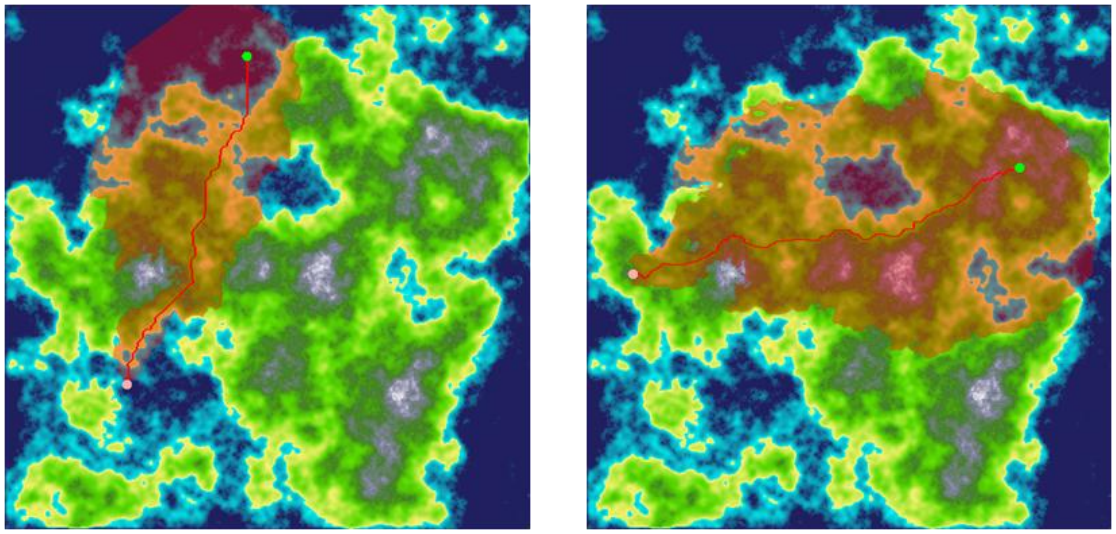
Skup varijabli nad kojima se razvija funkcija se sastoji od:

- ***vx*** - x komponenta vektora od trenutnog stanja do ciljnog stanja
- ***vz*** - z komponenta vektora od trenutnog stanja do ciljnog stanja
- ***h****ij* - visina s indeksom *ij* očitana iz visinske mape iz okoline trenutnog stanja
- ***hg****ij* - visina s indeksom *ij* očitana iz visinske mape iz okoline ciljnog stanja
- ***hd****ij* - razlika visina s indeksom *ij* (**hg**ij - **h**ij)

4.2 Prikaz rezultata

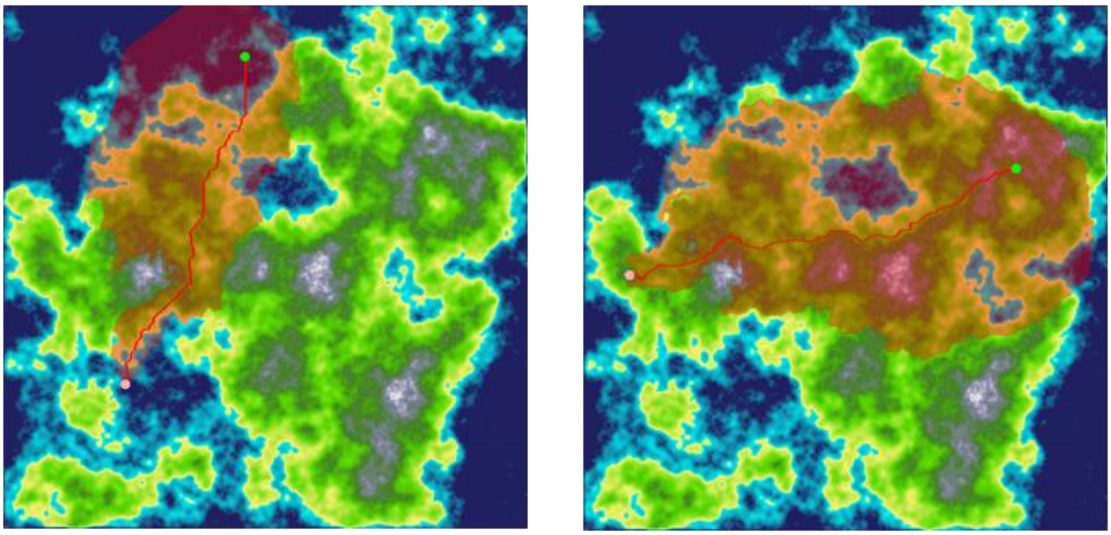
Sve navedeno na početku odjeljka 3.2 vrijedi i ovdje. Valja spomenuti da je jedino prva heuristika od dolje prikazanih nakon učenja uključivala varijable visine okoline u slučaju gdje je okolina samo polje stanja dok su sve ostale dobivene heuristike u tom slučaju bile jednake nekoj prethodnoj opisanoj heuristici i nisu uključivale varijable visine okoline.

1. (((abs(vz)+sqrt(h00))+(sqrt(hd00)+sqrt(vz)))+((abs(vx)+sqrt(hd00))+(sqrt(hd00)+sqrt(hd00)))) = $|\boldsymbol{vx}| + |\boldsymbol{vz}| + \sqrt{\boldsymbol{vz}} + \sqrt{\boldsymbol{h_{00}}} + \mathbf{3} * \sqrt{\boldsymbol{hd_{00}}}$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 4.2 ima omjer duljina putova $\mu = 1.0002401$, $\sigma = 7.4089 * 10^{-4}$ i omjer brojeva iteracija $\mu = 0.81724876$, $\sigma = 0.1737998$.



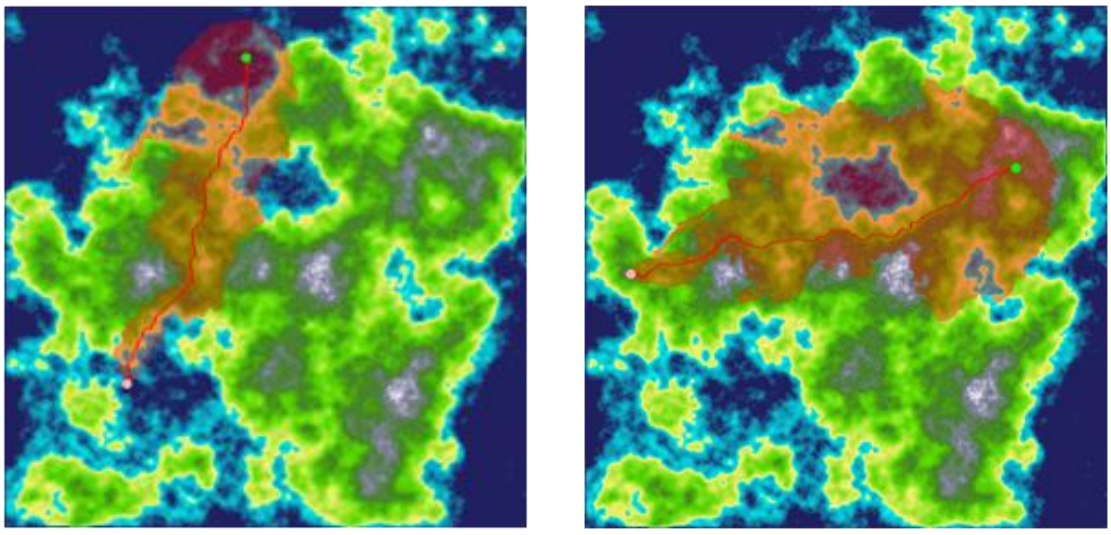
Slika 4.2 Rezultati (((abs(vz)+sqrt(h00))+(sqrt(hd00)+sqrt(vz)))+ ((abs(vx)+sqrt(hd00))+(sqrt(hd00)+sqrt(hd00)))) heuristike

2. ((abs(vz)+abs(vx))+(abs(sqrt(abs(abs(hd02))))+abs(sqrt(sqr(sqrt(hd10)))))) = $|\boldsymbol{vx}| + |\boldsymbol{vz}| + \sqrt{|\boldsymbol{hd_{02}}|} + \sqrt{\sqrt{\boldsymbol{hd_{10}}}^{\mathbf{2}}}$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 4.3 ima omjer duljina putova $\mu = 1.0001314$, $\sigma = 4.4942438 * 10^{-4}$ i omjer brojeva iteracija $\mu = 0.820655$, $\sigma = 0.15096517$.



Slika 4.3 Rezultati ((abs(vz)+abs(vx))+(abs(sqrt(abs(abs(hd02))))+ abs(sqrt(sqr(sqrt(hd10))))))) heuristike

3. sqrt((((sqr(vx)+sqr(vz))+abs((vz*hg00)))+(abs((vz*hg00))+(sqr(vx)+sqr(vz))))) = $\sqrt{\mathbf{2} * (\boldsymbol{vx^2} + \boldsymbol{vz^2} + |\boldsymbol{vz} * \boldsymbol{hg_{00}}|)}$: Heuristika čiji su rezultati izvođenja prikazani na slici Slika 4.4 ima omjer duljina putova $\mu = 1.0044892$, $\sigma = 0.007083564$ i omjer brojeva iteracija $\mu = 0.63533974$, $\sigma = 0.14073433$.



*Slika 4.4 Rezultati sqrt((((sqr(vx)+sqr(vz))+abs((vz*hg00)))+ (abs((vz*hg00))+(sqr(vx)+sqr(vz))))) heuristike*

5. Zaključak

Iz dobivenih rezultata može se zaključiti kako razvoj suboptimalnih heuristika A* algoritma za neki konkretan problem genetskim programiranjem rezultira heuristikama koje na A* algoritam djeluju tako da se cilj pronađe u prosjeku i do 35% manje iteracija uz, u prosjeku, zanemarivo dulji put što predstavlja osjetno ubrzanje i smanjenje memorijskih zahtjeva A* algoritma.

Razvojem heuristike uz varijable koje predstavljaju visinu okoline stanja jako teško se dolazi do rješenja koje te varijable uključuje, pa je za pretpostaviti kako je za razvoj takve heuristike potrebno izabrati neki drugi genotip: vektor realnih vrijednosti ili kombinaciju stabla i vektora realnih vrijednosti.

Uvidom u oblik heuristika može se uočiti da neke dobivene funkcije imaju određene dijelove koji se ponavljaju i međusobno zbrajaju pa bi jedno od mogućih poboljšanja u procesu učenja heuristike uz pomoć genetskog programiranja bilo uvođenje nasumičnih realnih konstanti kao listova stabla. No to bi uvođenje vjerojatno i zahtijevalo i poboljšanje procesa učenja funkcije dodatnim algoritmima lokalne pretrage.

6. Literatura

[1] Evolutionary Computational Framework, 10.12.2013., *Applications of Evolutionary Computation Scheduling / Timetabling / Optimization / Machine learning*, http://gp.zemris.fer.hr/ecf/, 06.05.2014.

[2] Paul Martz, Generating Random Fractal Terrain, http://www.gameprogrammer.com/fractal.html, 06.05.2014.

[3] Box Blur, http://en.wikipedia.org/wiki/Box_blur, 06.05.2014.

7. Sažetak

U ovom radu se istražuje mogućnost razvoja heuristike A* algoritma uz pomoć genetskog programiranja.

U uvodnom poglavlju ukratko je objašnjen opseg problema kojeg se rješava A* algoritmom i zašto je baš taj algoritam prvi odabir pri rješavanju tih problema. Ukratko je opisan problem pronalaska suboptimalnog puta na nasumično generiranim visinskim mapama nad kojim se uči heuristika i ostvarenje razreda koji modeliraju taj problem i sam A* algoritam.

U drugom se poglavlju opisuje postupak razvoja heuristike: opisane su neke promjene u implementaciji A* algoritma, genotip, postupak evaluacije jednog rješenja i postupci ubrzanja evaluacije. Opisan je postupak učenja nad skupom parova stanja i jednom visinskom mapom te funkcija cilja koju se genetskim programiranjem nastoji minimizirati.

U trećem i četvrtom poglavlju se opisuju rezultati nastali razvojem heuristike nad skupom od 20 stanja bez i sa varijablama koje predstavljaju visinu okoline promatranog stanja.