Vrsta rada: Originalni naučni rad
Primljen: 10.06.2023.
Prihvaćen: 04.07.2023.
UDK: 004.421:795

Algoritmi za pronalaženje putanje u igrama

Marko Novaković

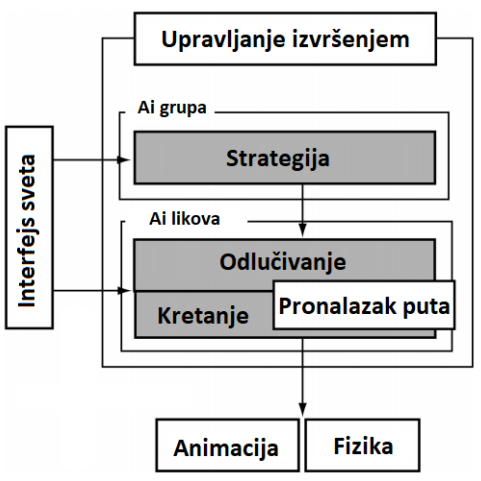
Visoka škola strukovnih studija za informacione tehnologije ITS , Beograd, Srbija, marko45521@its.edu.rs

Sažetak – U ovom članku su opisani A*, Dijkstrin i genetski algoritam za pronalaženje puteva koji se koriste u igrama radi poređenja i informisanja. Ovo nisu svi algoritmi koji se koriste u igrama pri pronalasku puteva – trenutno se najčešće koriste. Pošto postoji potreba da se u igrama prikaže što više podataka (kvalitetnija grafika, komplikovaniji sistemi za komunikaciju sa okolinom, više kvalitetnijih zvukova, komplikovaniji set pokreta koje mogu izvesti likovi, pametniji ai itd.) u što kraćem periodu, algoritmi se moraju razvijati da budu optimalniji, pa će ih u nekoj bliskoj budućnosti zameniti njihove bolje verzije ili sasvim novi algoritmi.

Ključne riječi – *ant colony optimization, ai, A*, algoritam, breadth first search, Dijkstrov algoritam, dijkstra's algorithm, game character, graf, genetski algoritam, genetic algorithm, level, patfinding, traženje putanje, optimizacija kolonije mrava*

I. UVOD

Likovi u igrama često imaju potrebu da se kreću po nivou. Ponekad ovo kretanje određuju programeri, npr. putanju po kojoj patrolira čuvar ili mali ograđeni deo po kojem pas može nasumično da se kreće. Fiksne putanje su jednostavne za implementaciju, ali takođe može lako doći do greške ako neki objekat dospe na putanju – likovi koji se nasumično kreću mogu izgledati kao da idu besciljno i mogu se lako zaglaviti.



Kompleksniji likovi ne znaju unapred gde će se kretati. Jedinica u strateškim igrama u realnom vremenu može dobiti naredbu od igrača da ode do određene tačke na mapi u bilo kom trenutku u vremenu, u igrama gde je važno biti neopažen će čuvar koji patrolira možda morati da ode do najbližeg mesta za alarm i pozove pojačanje, protivnici u platformskim igrama će možda morati da jure igrača preko provalija koristeći dostupne platforme.

Za svaki od ovih likova AI (Artificial Intelligence – veštačka inteligencija) mora biti u mogućnosti da izračuna pogodnu putanju po nivou igre da bi lik stigao od mesta gde je sad do cilja. Želeli bismo da putanja bude razumna i što kraća i brža (ne izgleda pametno ako lik hoda iz kuhinje do dnevne sobe preko tavana).

Ovo pronalaženje putanje (pathfinding) ponekad se naziva planiranje putanje (pathplanning) i svuda je u AI-ju igre. U primeru modela AI igre na slici 1 pronalaženje putanje je na granici između donošenja odluka i kretanja. Često se koristi samo da odredi kuda se kretati da bi se došlo do cilja. Cilj se određuje drugim AI-jem, a pronalazač puteva samo određuje putanju. Da bi se ovo izvelo, može se ugraditi u kontrolu sistema kretanja tako da se poziva samo kada je potrebno isplanirati putanju. Ali se isto tako AI za pronalaženje puteva može koristiti i da određuje cilj i putanju.

Velika većina igara koristi rešenja za pronalaženje puteva pomoću algoritma koji se zove A* (A star – A zvezda). Iako je efikasan i lako se implementira, A* ne može direktno da radi sa podacima nivoa igre. Potrebno je da se nivo predstavi u određenoj strukturi podatka: usmereni pozitivni ponderisani graf [1]. Slika 1. *AI model igre [1]*

II. DIJKSTROV ALGORITAM

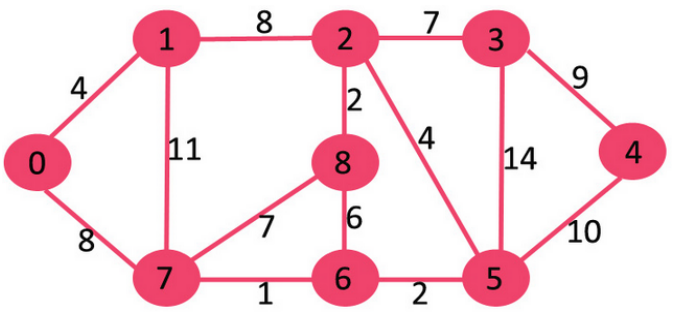
Dati su graf i početni čvor. Odrediti najkraći put od početnog čvora do svih čvorova grafa.

Generišemo stablo najkraće putanje i za koren uzimamo početni čvor. Imamo dva seta podataka. Prvi set sadrži čvorove koji su uključeni u najkraći put stabla, drugi set sadrži čvorove koji još uvek nisu uključeni u najkraći put. U svakoj iteraciji algoritma pronalazimo čvor koji se nalazi u setu koji još uvek nije uključen u najkraći put i ima najmanju udaljenost od izvora.

Detaljni koraci algoritma:

1. Kreiraj set sptSet (shortest path tree set) koji vodi računa o čvorovima koji su uključeni u najkraći put – čija je najkraća udaljenost od izvora izračunata i potvrđena; inicijalno ovaj set je prazan.
2. Dodeliti vrednosti za udaljenosti za sve čvorove grafa. Inicijalizuju se sve udaljenosti sa beskonačno. Dodeljuje se vrednost 0 za početni čvor, tako da bude izabran prvi.
3. Sve dok sptSet ne sadrži sve čvorove:
4. Izaberi čvor koji nije u sptSet i ima najmanju razdaljinu od poslednjeg čvora.
5. Uključi ga u sptSet.
6. Ažuriraj sve udaljenosti čvorova koji su susedni čvoru u (poslednji čvor). Da bi se ažurirale udaljenosti, prolazi se kroz sve susedne čvorove čvoru u. Za svaki susedni čvor v, ako je suma udaljenosti od u i pondera (težinskog faktora – udaljenost u ovom slučaju) od u do v manja od udaljenosti v, onda ažuriraj udaljenost v.

Primer grafa na slici 2 na kom će se objasniti rad algoritma:

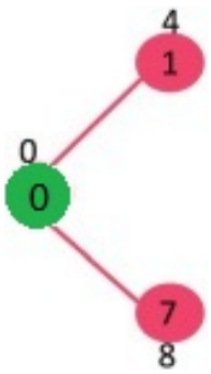


Slika 2. *Primer ponderisanog grafa [2]*

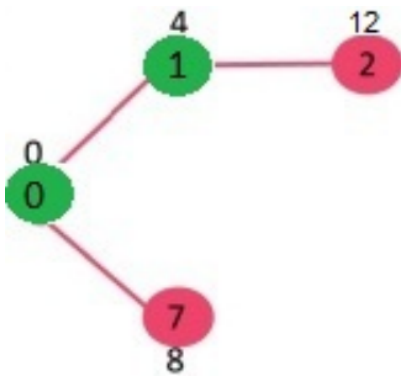
Set sptSet je na početku prazan i udaljenosti dodeljene čvorovima su {0, INF, INF, INF, INF, INF, INF, INF}, gde INF označava beskonačno (infinity). Sada se bira čvor sa minimalnom vrednosti za udaljenost. Čvor 0 je izabran, uključuje se u sptSet. Sada je sptSet {0}. Posle dodavanja 0 sptSetu, ažuriraju se vrednosti njegovih susednih čvorova. Susedni čvorovi 0 su 1 i 7. Vrednosti udaljenosti 1 i 7 su ažurirane kao 4 i 8. Sledeći podgraf prikazuje čvorove sa njihovim udaljenostima, prikazani su samo oni čvorovi koji imaju konačne vrednosti udaljenosti. Čvorovi koji su uključeni u SPT (Shortest path tree) su označeni zelenom bojom (slika 3).

Odaberi čvor sa najmanjom udaljenosti koji nije već uključen u SPT (nije u sptSetu). Čvor 1 je izabran i dodat je u sptSet. Sada sptSet izgleda: {0,1}. Ažuriraj vrednosti udaljenosti susednih čvorova čvora 1. Udaljenost čvora 2 postaje 12 (slika 4).

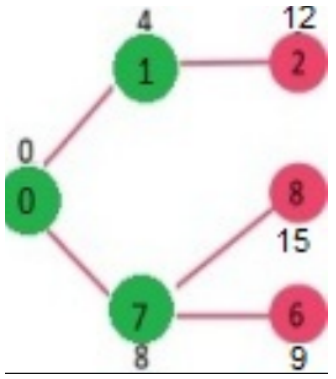
Odaberi čvor koji ima najmanju udaljenost i nije uključen u SPT (nije u sptSetu). Čvor 7 je izabran. Vrednosti udaljenosti čvorova 6 i 8 postaje konačna (15 i 9) (slika 5).



Slika 3. *Podgraf 1 [2]*

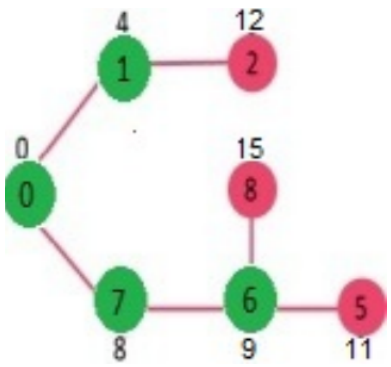


Slika 4. *Podgraf 2 [2]*



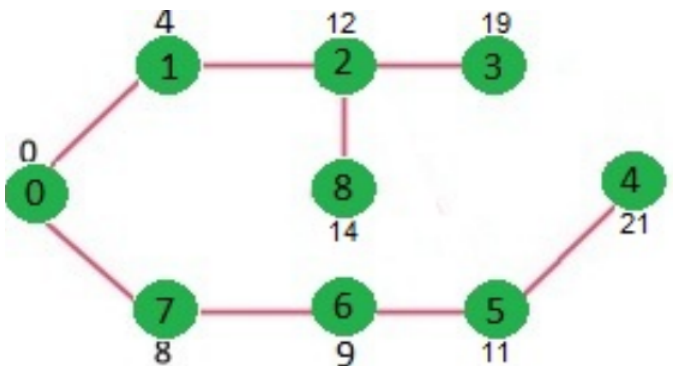
Slika 5. *Podgraf 3 [2]*

Odaberi čvorove sa najmanjom udaljenosti koji nisu već uključeni u SPT (nisu u sptSetu). Čvor 6 je odabran. Sada sptSet izgleda: {0,1,7,6}. Ažuriraj vrednosti udaljenosti susednih čvorova čvoru 6. Udaljenosti čvorova 5 i 8 su ažurirane (slika 6).



Slika 6. *Podgraf 4 [2]*

Ponavljamo ove korake dok u sptSet nisu uključeni svi čvorovi. Na kraju dobijamo sledeće stablo najkraćeg puta (SPT) (slika 7) [2].



Slika 7. *Podgraf 5 [2]*

III. A* ALGORITAM

A* (izgovara se A zvezda) jeste algoritam koji se često koristi u pronalaženju puteva i kretanju po grafu. Algoritam efikasno prikazuje putanju kretanja između čvorova grafa.

Na mapi sa mnogo prepreka pronalaženje puteva između tačke A i B može biti teško. Robot, na primer, bez dobijanja dodatnih uputstava o pravcu kretanja će nastaviti da se kreće sve dok ne dođe do prepreke (slika8).

Međutim, A* algoritam uvodi heuristiku u standardne algoritme za pretragu po grafovima, u suštini planirajući unapred svaki korak tako da se donese optimalnija odluka. Sa A* robot bi tražio put kao na slici 9.

A* je proširen Dijkstrov algoritam sa nekim karakteristikama algoritma pretrage u širinu (breadth-first search (BFS)) [3].

Kao i Dijsktrov algoritam, A* radi tako što napravi stablo najkraćeg puta od početnog čvora do ciljnog čvora. Ono što čini A* različitim i boljim za mnoge pretrage je to što za svaki čvor koristi funkciju $f(n)$, koja daje procenu totalne cene (dužine) puta kada bi se koristio taj čvor. Stoga je A* heuristička funkcija, koja se razlikuje od algoritma po tome što je heuristika više procena nego što je dokazivo tačna.

A* proširuje putanje koje su kraće (jeftinije) koristeći funkciju:

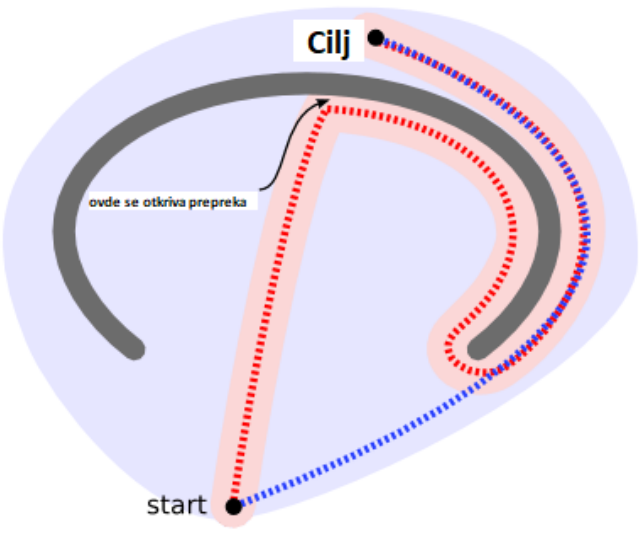
$$f(n) = g(n) + h(n),$$

gde je:

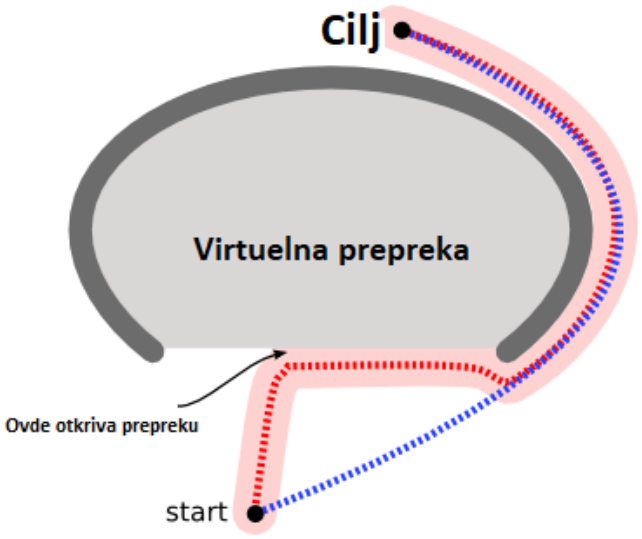
$f(n)$ = ukupna procenjena cena putanje kroz čvor n;

g(n) = akumulirana cena do čvora n;

h(n) = procenjena cena od n čvora do cilja. Ovo je heuristički deo funkcije, pa je kao pretpostavka.

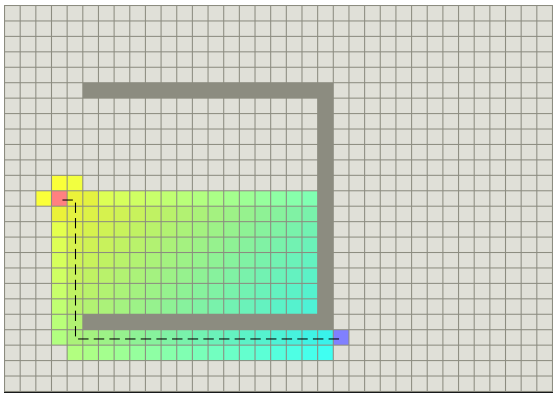


Slika 8. *Neefikasan način pronalaska puta [5]*



Slika 9. *Primer korišćenja A* u pronalasku puta [5]*

U rešetki na slici 10 A* algoritam počinje od početka (crveni čvor) i uzima u obzir sve susedne čvorove. Kada se lista susednih čvorova popuni, filtriraju se oni koji su nepristupačni (zidovi, prepreke, izvan granica). Onda se odabere čvor sa najmanjom cenom, koja se određuje sa $f(n)$. Ovaj proces se rekurzivno ponavlja sve dok se ne nađe najkraći put do cilja (plavi čvor). Proračunavanje $f(n)$ se radi heuristički, te obično daje dobre rezultate.



Slika 10. *Korišćenje A* algoritma [5]*

Izračunavanje h(n) može da se uradi na više načina:

Manhattanova udaljenost [4] od n do cilja se najčešće koristi. Ovo je standardna heuristika za rešetku.

Ako je h(n) = 0, A* postaje Dijsktrin algoritam, pomoću kog je zagarantovano da se pronađe najkraći put.

Heuristička funkcija mora biti prihvatljiva, što znači da nikad ne može da preceni cenu potrebnu da bi se došlo do cilja. I Manhattanova udaljenost i h(n) = 0 su prihvatljive.

Koristiti dobru heuristiku je važno u određivanju performansi A* algoritma. Vrednost h(n) bi idealno bila tačna cena stizanja do cilja. Ipak, ovo nije moguće zato što se i ne zna putanja. Ali se može odabrati metod koji će nekada dati tačne vrednosti, npr. kada se putuje u pravoj liniji bez prepreka. Ovo bi rezultiralo perfektnom performansom A* algoritma.

Poželjno je da se odabere h(n) funkcija, koja košta manje nego što bi koštalo stizanje do cilja. Ovo bi omogućilo da h(n) radi precizno. Ako odaberemo vrednost koja je veća, onda bi to dovelo do bržih, ali manje preciznih performansi. Tako da je često slučaj da se h(n) bira tako da bude manje nego realna cena.

Na slici 11 prikazan je pseudokod A* algoritma napisan sintaksom nalik na Python [5].

```
napravi otvorenu listu koja se sastoji samo iz početnog čvora
napravi praznu zatvorenu listu

while (nije se stiglo do ciljnog čvora):

    uvrsti čvor sa najmanjom f vrednosti u otvorenu listu

    if(ovaj čvor je naš odredišni čvor):

        završili smo

    if not:

        uvrsti trenutni čvor u zatvorenu listu i pregledaj sve njegove susedne čvorove

        for(svaki susedni čvor trenutnog čvora):

            if(ako susedni čvor ima manju g vrednost od trenutnog čvora i u zatvorenoj je listi):

                zameni suseda sa novim koji ima manju g vrednost
                trenutni čvor je sada roditelj susednom čvoru

            else if(ako je g vrednost trenutnog čvora manja od susednog i ovaj sused je u otvorenoj listi):

                zameni suseda sa novim kojim ima manju g vrednost
                ažuriraj roditelja suseda na trenutni čvor

            else if(ovaj sused nije ni u jednoj listi):

                dodaj ga u otvorenu listu i dodeli mu g vrednost
```

Slika 11. *A* pseudokod [5]*

Za više detalja o algoritmu A* pogledajte članak [6].

IV. Heurističke tehnike

Heurističke tehnike se koriste da reše problem na brži i efikasniji način, optimizujući rešenje, tačnost i preciznost [7]. Cilj heurističkih algoritama je da nađu dobro rešenje za određeni problem, kao što je pronalaženje puteva (pathfinding) u razumnom vremenu izračunavanja, ali bez zagarantovane efikasnosti. Heuristika na grčkom znači pronaći [8]. U heurističke algoritme spadaju i Dijkstrin i A* algoritam, koji su opisani u prethodnom tekstu, kao i algoritam pretrage u širinu (breadth-first search (BFS)) [3].

V. Metaheurističke tehnike

Metaheuristika je u suštini skup strategija na visokom nivou koje kombinuju tehnike nižeg nivoa za opisivanje i eksploatisanje prostora pretrage. Metaheuristika je viši nivo heuristike. Obično ima bolje performanse nego heuristika. Metaheuristika može da skrati vreme pretrage i da izgleda dovoljno dobro da reši kompleksne putanje u video-igrama. Na osnovu studija metaheuristički algoritmi, kao što su genetski algoritam i optimizacija kolonija mrava, korišćeni su u igrama da bi rešili probleme pronalaska puteva. Metaheuristika je bazirana na osnovu nekih prirodnih pojava. Najuspešniji metaheuristički algoritmi su inspirisani prirodnim sistemima. Na primer, optimizacija kolonija mrava (ant colony optimization) [9] i algoritam pčele (bee algoritam) bili su razvijeni na osnovu ponašanja životinja [10].

VI. Genetski algoritam

Genetski algoritmi su među najpopularnijim evolucionim algoritmima u smislu raznolikosti njihove primene. Za veliku većinu dobro poznatih optimizacionih problema traženo je rešenje u genetskim algoritmima. Dodatno, genetski algoritmi su zasnovani na populaciji i mnogi moderni evolucioni algoritmi su bazirani na osnovu genetskih algoritama ili imaju velikih sličnosti sa njima.

Suština genetskih algoritama je enkodiranje optimizacione funkcije kao niz bitova ili niz karaktera koji predstavljaju hromozome, kao i manipulacija stringova genetskim operatorima i selekcija pogodnih individua, sa ciljem da se pronađe dobro (pa i optimalno) rešenje problema. U daljem tekstu će se koristiti pogodnost i funkcija pogodnosti. Pogodnost se odnosi na željene karakteristike koje želimo da dobijemo iteracijama algoritma.

Ovo se najčešće radi sledećom procedurom:

1. enkodiranje ciljeva ili funkcija troškova;
2. definisanje funkcije pogodnosti (fitness function) ili kriterijuma selekcije;
3. kreiranje populacije individua;
4. sprovođenje evolucionog ciklusa ili iteracija ocenjivanjem pogodnosti svih individua populacije i kreiranje nove populacije obavljanjem prelaska i mutacije, pogodne reprodukcije itd., na kraju se menja stara populacija i vrše se iteracije korišćenjem nove populacije;
5. dekodiranje rezultata dobijenih rešenjem.

Ovi koraci mogu se predstaviti šematski kao pseudokod genetskih algoritama (slika 12).

Jedna iteracija kreiranja nove populacije naziva se generacija. Najčešće se koriste stringovi fiksne dužine u većini genetskih algoritama tokom svake generacije, iako postoji pozamašno istraživanje stringova promenljive dužine i struktura kodova. U prilagodljivim genetskim algoritmima kodiranje ciljne funkcije često je u formi binarnih nizova ili nizova sa realnim vrednostima. Zbog jednostavnosti, u diskusiji su korišćeni binarni nizovi. Genetski operatori uključuju prelazak, mutaciju i selekciju (engl. crossover, mutation and selection) iz populacije.

Prelazak dva roditeljska niza je glavni operator sa velikom verovatnoćom, označava se sa $\mathbf{P_c}$ i izvršava se tako što se zameni jedan segment na nasumično odabranoj poziciji jednog hromozoma odgovarajućim segmentom drugog hromozoma (slika 13).

Operator mutacije se dobija tako što se na nasumično odabranom bitu zameni vrednost ($0 \rightarrow 1$ ili $1 \rightarrow 0$) (slika 14), verovatnoća mutacije se označava sa $\mathbf{P_m}$ i često je mala. Dodatno se može desiti da se na više mesta dogodi mutacija, što može biti prednost u praksi i primeni.

Selekcija individua u populaciji obavlja se procenom pogodnosti i individua se može naći u sledećoj generaciji ako je određeni prag pogodnosti dostignut.

Dodatno, selekcija može biti bazirana na pogodnosti tako da razmnožavanje populacije bude proporcionalno pogodnosti, što bi značilo da je veća šansa da se individue sa većom pogodnosti razmnožavaju [11].

Ciljna funkcija $f(x)$, $x = (x_1, \ldots, x_d)^T$

Enkodiraj rešenja u hromozome (stringove)

Definiši pogodnost F (npr. $F \propto f(x)$ za maksimizaciju)

Generiši početnu proporciju

Definiši vrerovatnoću prelaska (P_c) i mutacije (P_m)

while (t < maksimalnog broja generacija)

Generiši novo rešenje prelaskom i mutacijom

Odradi prelazak sa verovatnoćom P_c

Odradi mutaciju sa verovatnoćom P_m

Prihvati nova rešenja ako im se pogodnost poveća

Izaberi trenutno najboljeg za sledeću generaciju

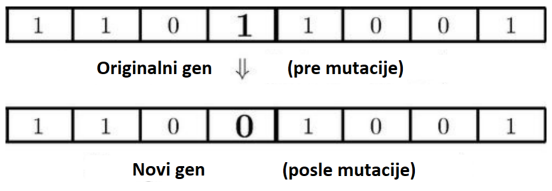
Ažuriraj t = t + 1

end while

Dekodiraj rezultate i vizualizuj

Slika 12. *Pseudokod genetskih algoritama [11*

Slika 13. *Dijagram prelaska nasumičnog segmenta u genetskim algoritmima [11]*



Slika 14. *Dijagram mutacije nasumičnog bita [11]*

Zahvalnica

Rad je rađen u okviru predmeta Osnove primenjenih istraživanja, a mentor rada je prof. dr. Slavko Pokorni.

Literatura

1. I. Millington, AI for Games third edition 2019, pp. 195–196
2. „Dijkstra's shortest path algorithm | Greedy Algo-7", GeeksForGeeks, https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/; posećeno: 4. 4. 2022.
3. „Breadth First Search (BFS) Algorithm", algotree.org, https://algotree.org/algorithms/tree_graph_traversal/breadth_first_search; posećeno: 4. 4. 2022.
4. P. E. Black, „Manhattan distance", in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed. 11 February 2019; https://www.nist.gov/dads/HTML/manhattanDistance.html; posećeno 4. 4. 2022.
5. A* Search. *Brilliant.org*. Retrieved 10:24, April 4, 2022, from https://brilliant.org/wiki/a-star-search/.

6. L. Patrick. „A* pathfinding for beginners.” *online]. GameDev WebSite.:https://www.gamedev.net/reference/articles/article2003.asp;* posećeno 4. 4. 2022. (2005).
7. L. A. Wolsey, „Heuristic Algorithms”, Integer Program., no. January, p. 17, 1998.
8. A. Rafiq *et al* 2020 *IOP Conf. Ser.: Mater. Sci. Eng.* **769** 012021 „prihvaćen za objavljivanje”
9. X. S. Yang, Nature-Inspired Optimization Algorithms, 2014, pp. 305–308.
10. X. S. Yang, Nature-Inspired Optimization Algorithms, 2014, pp. 308–312.
11. X. S. Yang, Nature-Inspired Optimization Algorithms, 2014, pp. 116–130.