Komparativna analiza algoritama za pronalaženje najkraće putanje u grafu korišćenjem Unity razvojne platforme

Nikola Vukotić, Slavimir Stošović
Katedra za informaciono-komunikacione tehnologije
Akademija tehničko-vaspitačkih strukovnih studija Niš
Niš, Srbija
nikola.vukotic@akademijanis.edu.rs
slavimir.stosovic@akademijanis.edu.rs

Sažetak — **U ovom radu su prikazani rezultati analize najpoznatijih algoritama za pretragu grafova (BFS, Dijkstrin, Greedy BFS i A* algoritam) i njihove performanse. Za analizu predstavljenih algoritama kreiran je simulator za pronalaženje najkraćeg puta između dve tačke uz pomoć Unity razvojne platforme koristeći C# programski jezik. Grafički prikaz rada algoritama je prikazan uz pomoć dvodimenzionalne mreže međusobno povezanih polja, gde se pojam težinskih čvorova i zidova može prevesti u strukturu usmerenog težinskog grafa.**

Ključne reči - Algoritmi za pretragu grafa; Path finding algoritmi; Algoritmi za nalaženje najkraćeg puta; BFS algoritam; Dijkstrin algoritam; Greedy BFS algoritam; A* algoritam; Unity; C#;

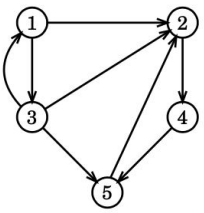
I. UVOD

Pretraživanje najkraćeg puta je osnovna komponenta mnogih važnih aplikacija u oblastima navigacije, video igara, robotike, logistike i simulacije gužve u saobraćaju. Primeri takvih problema uključuju usmeravanje telefonskog i internet saobraćaja, GPS lociranje, donošenje odluka u veštačkoj inteligenciji i robotici. Ovom problematikom se bave mnoga istraživanja što je dovelo do pronalaska nekoliko različitih rešenja, odnosno algoritama za pronalaženje najkraćeg puta u mreži [1]. Za pronalaženje najkraćeg puta u mreži, bira se algoritam u zavisnosti od zadatih parametara mreže i samog vremena izvršavanja algoritma.

U raznim primenama često se javlja potreba za predstavljanjem nekih objekata i uzajamnih odnosa koji postoje između tih objekata. Grafovi su prirodan model za takvu reprezentaciju. Zbog svoje široke praktične primene, grafovi su sveprisutni model podataka u računarstvu, a algoritmi sa grafovima su osnovni gradivni elementi složenijih algoritama. Graf predstavlja skup tačaka, odnosno čvorova, povezanih linijama, odnosno granama. Grafovi se neformalno često predstavljaju dijagramima u kojima se čvorovi prikazuju kružićima, a grane se prikazuju linijama ili strelicama koje spajaju odgovarajuće kružiće.

U zavisnosti od problema, svakoj grani se može pridružiti broj, koji se obično naziva težina grane. Pored težine, grafovi, odnosno njihove grane, mogu imati smer i takvi grafovi se nazivaju usmereni grafovi ili digrafovi. Usmereni grafovi ograničavaju kretanje, jer grana može dopustiti kretanje samo u jednom smeru. Neusmereni graf se može posmatrati kao usmereni graf kod koga su sve ivice dvosmerne. Netežinski graf se može smatrati težinskim grafom kod koga su sve težine jednake jedinici. Od sada kada se govori o grafu u ovom radu, misli se na konačan težinski usmereni graf, jer je to najopštiji tip grafa koji je korišćen u radu.

Da bi se grafovi mogli koristiti u računarstvu, moraju se predstaviti u programima na način koji obezbeđuje lako manipulisanje njihovim čvorovima i granama. Najprostija struktura koja se koristi za predstavljanje grafova je matrica susedstva. Matrica susedstva je običan dvodimenzionalni niz čiji su brojevi redova i kolona jednaki broju čvorova grafa. Na Slici 1. je prikazan primer usmerenog grafa i njegova matrica susedstva.



	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	1	1	0	0	1
4	0	0	0	0	1
5	0	1	0	0	0

Slika 1. Usmereni graf i njegova matrica susedstva

U ovom radu su predstavljeni određeni algoritmi za pronalaženje najkraćeg puta. Grafički prikaz rada algoritama je prikazan uz pomoć „grid“-a, odnosno dvodimenzionalne mreže međusobno povezanih polja. Svako polje, predstavlja čvor (*eng. Node*) i može imati različitu težinu ili biti zid, odnosno polje koje je neprohodno. Ovaj pojam težinskih čvorova i zidova se može prevesti u strukturu usmerenog težinskog grafa [2] razmišljajući o težini polja kao težini grana koje povezuju susedna polja. Zidovi se mogu smatrati čvorovima koji nisu povezani ni sa jednim drugim čvorom ili koji jednostavno ne postoje, pa prema tome nisu dostupni drugim čvorovima. Implementirani algoritmi su dovoljno opšti za rad sa bilo

kojom drugom specifikacijom grafa, a imaju značajnu ulogu u izradi igara [3].

Na geografskoj mapi, čvorovi mogu predstavljati gradove i mesta, grane bi mogle predstavljati puteve koji ih povezuju, a težine dužinu svakog puta. Tada bi se moglo postaviti pitanje koji je najbolji način da se putuje iz jednog grada u drugi, odnosno skup puteva koji bi čovek morao da pređe tako da ukupna pređena udaljenost bude minimalna uzimajući u obzir sve moguće načine kako doći iz jednog grada u drugi. Za rešenje ovog problema mogao bi se koristiti "brute force" algoritam, tako što bi pronašao svaku moguću kombinaciju za pronalaženje puta i onda izabrao onu sa minimalnim rastojanjem. Ali, kako broj čvorova i grana raste tako i broj kombinacija postaje izuzetno veliki i izvršavanje algoritma bi moglo predugo trajati, a može ponestati i memorije koja raste u skladu sa brojem čvorova, posebno na prenosnim uređajima kao što su mobilni telefoni, tableti, *embedded* uređaji i slično. Zato se specijalno u te svrhe koriste algoritmi za pronalaženje najkraćeg puta poput onih koji su predstavljeni u ovom radu. Mnogi autori se bave optimizacijom algoritama za pronalaženje najkraćeg puta [4] - [6]. Cilj ovog rada nije bila dodatna optimizacija, već kreiranje simulatora za pronalaženje najkraćeg puta korišćenjem Unity [7] game engine-a, koji se već koristi za razvoj igara i merenje brzine pronalaska najkraćeg puta korišćenjem programskog jezika C# [8].

II. ALGORITMI ZA PRONALAŽENJE NAJKRAĆE PUTANJE U GRAFU

Pre nego što budu opisani detaljno svi algoritmi za pronalaženje najkraćeg puta prvo će biti opisan deo algoritma koji je zajednički za sve algoritme opisane u ovom radu. Na Slici 2. prikazan je dijagram toka algoritma za pronalaženje najkraćeg puta.

Tok algoritma je sledeći:

1. Ulazni parametri su: *graph* - graf u kome se vrši pretraga, *startNode* - početni čvor i *goalNode* - krajnji čvor, odnosno cilj koji se traži.

2. *PriorityQueue<Node> frontierNodes* - predstavlja promenljivu tipa strukture red sa prioritetom koja sadrži element tipa *Node*, služi za smeštanje graničnih čvorova koje treba istražiti sledeće.

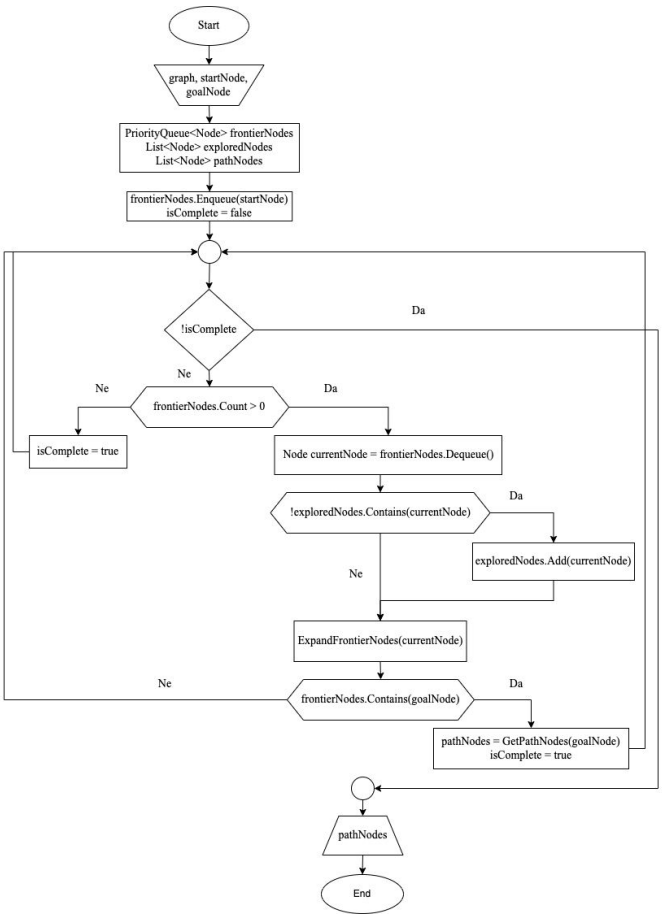
3. *List<Node> exploredNodes* - promenljiva koja je tipa liste koja služi za smeštanje čvorova koji su već istraženi.

4. *List<Node> pathNodes* - promenljiva koja je tipa liste koja sadrži čvorove u redosledu od početnog čvora do ciljnog čvora.

5. *frontierNodes.Enqueue(startNode)* - na samom početku algoritma na kraj reda *frontierNodes* se dodaje početni čvor *startNode*. Promenljiva tipa *bool isComplete* sadrži inicijalnu vrednost *false*.

6. Zatim se ulazi u *while* petlju koja se izvršava sve dok uslov ne bude ispunjen, odnosno *isComplete* ne dobije vrednost *true*.

7. Ispituje se da li je red *frontierNode* prazan a ukoliko jeste, nema čvorova koje treba ispitati dalje, *bool* promenljiva *isComplete* postaje *true* i *while* petlja se završava. U suprotnom se iz reda *frontierNode* uzima prvi element i smešta u promenljivu *currentNode* koja se dalje ispituje.



Slika 2. Dijagram toka algoritma za pronalaženje najkraće putanje u grafu

8. Ukoliko trenutni čvor koji se ispituje *currentNode* nije već prethodno istražen, odnosno ne nalazi se u listi *exploredNodes*, onda se dodaje kao element u listu *exploredNodes*, čime se obeležava da je taj čvor istražen.

9. Zatim se izvršava funkcija *ExpandFrontierNodes(currentNode)* koja je ključna i koja se razlikuje u zavisnosti od algoritma za pronalazak najkraćeg puta koji se primenjuje. Ova funkcija ispituje susedne čvorove čvora *currentNode* i proširuje, odnosno dodaje nove elemente u red *frontierNodes* koji treba biti sledeći ispitani. Prioritet po kome se određuju koji čvorovi će biti ispitani sledeći zavisi od algoritma pretrage i biće opisani u narednim sekcijama za svaki opisani algoritam kao i detaljna implementacija *ExpandFrontierNodes(currentNode)* funkcije.

10. Ukoliko red *frontierNodes* sadrži ciljni čvor *G*, znači da je cilj pronađen i promenljiva *isComplete* postaje *true*. U listu *pathNodes* se smeštaju čvorovi koji su rezultat *GetPathNodes(goalNode)* funkcije. Funkcija *GetPathNodes(goalNode)* služi da rekonstruiše putanju do ciljanog čvora, odnosno da sortira čvorove u redosledu kako su istraživani od startnog čvora do ciljnog čvora. Kod funkcije *GetPathNodes(goalNode)* prikazan je na slici 3.2.

11. Ukoliko cilj još uvek nije pronađen, *while* petlja se ponovo izvršava, u suprotnom izlazni parametar algoritma je lista *pathNodes* koja sadrži čvorove od početnog do krajnjeg čvora i najbolju moguću putanju dobijenu izabranim algoritmom za pronalazak najkraćeg puta u grafu.

Najpopularniji klasični algoritmi za pretragu su BFS i Dijsktrin koji vrše pretragu samo na osnovu informacija koje su već date problemom. Ti algoritmi će preći ceo graf sve dok ne dođu do cilja ili će u suprotnom „iscrpeti“ graf. Oni se takođe nazivaju i „neinformisani algoritmi pretrage“ ili „slepi algoritmi pretrage“. Neinformisani algoritmi pretrage mogu dobro funkcionisati u pretrazi manjih grafova, ali postaju nepouzdani ili čak nerešivi kada problem postane složeniji.

Za složenije probleme se koristi heuristička pretraga (informisana pretraga). Za razliku od klasičnih algoritama pretrage, algoritmi heurističke pretrage mogu koristiti znanje izvan same definicije problema da isprobaju putanje po redosledu „obećanja“ kako bi efikasno pronašli rešenja. Da bi dobili ove dodatne informacije, algoritmi heurističke pretrage koriste heurističku funkciju h(n). Heuristička funkcija je funkcija koja rangira alternative u algoritmima pretrage na svakom koraku grananja na osnovu dostupnih informacija da bi se donela odluka koju granu treba dalje pratiti.

Generalno, heuristička funkcija je ekvivalentna procenjenoj ceni najjeftinijeg puta od datog čvora do ciljnog čvora. U ovom radu su opisana dva heuristička algoritma Greedy BFS i A * algoritam.

A. BFS algoritam

Algoritam pretrage „najpre u širinu” (eng. *breadth first search,* skraćeno BFS) [3] je algoritam pretrage i obilaska koji se primenjuje nad strukturama podataka stabla i grafova za operaciju pretrage i obilaska. Ovaj algoritam pretražuje podjednako u svim pravcima dok ne pronađe cilj [9]. Drugim rečima, započinje pretragu od izabranog čvora i ispituje sve njegove susedne čvorove, zatim ispituje susedne čvorove već ispitanih susednih čvorova. Ovaj redosled ispitivanja čvorova se ponavlja sve dok se ne stigne do ciljanog čvora.

BFS algoritam koristi strukturu podataka red za praćenje sledećih čvorova koje treba ispitati, dodajući sve neposećene susede čvora kada se ispituje, sve dok se red ne isprazni. Istraženi čvorovi se čuvaju u skupu (eng. *set*), da se isti čvor ne bi istraživao dva puta. Skup čvorova koji nisu još istraženi često se naziva otvoreni skup, a skup posećenih čvorova zatvoreni skup. Takođe, čvorovi u otvorenom skupu se nazivaju otvoreni čvorovi, a u zatvorenom skupu zatvoreni čvorovi.

Iako BFS algoritam uspešno nalazi put do ciljnog čvora ovo nije najefikasniji algoritam za pronalazak najkraćeg puta u grafu. Jedan veliki problem je što algoritam generiše putanju samo na osnovu toga kako su čvorovi otkriveni tokom pretraživanja grafa. Ovo dovodi do toga da BFS ne nalazi uvek najkraći mogući put između dva čvora u grafu.

B. DIJKSTRIN algoritam

Dijkstrin algoritam je jedan od najpopularnijih algoritama u teoriji grafova koji se koristi za pronalaženje optimalnih putanja između čvorova u grafu sa pozitivnim težinama [10].

Dijkstrin algoritam je „pohlepan” (eng. *greedy*) algoritam. Pohlepan algoritam kreira rešenje korak po korak i u svakom koraku bira najoptimalniji put. Konkretno, Dijkstrin algoritam pronalazi najkraće puteve između čvorova u usmerenim i neusmerenim grafovima.

Ključna razlika između BFS i Dijkstrinog algoritma je u tome sto Dijsktrin algoritam vodi računa o tome kolika je dužina pređenog puta do svakog čvora. Iako takođe vrši pretragu tako što povezuje susedne čvorove, povezuje samo one čvorove čija je udaljenost najkraća od startnog čvora.

Osnovni postupak je dodeljivanje svakom čvoru vrednost udaljenosti. Početnom čvoru se dodeljuje vrednost nula a za ostale čvorove beskonačnost. Svi čvorovi su označeni kao neposećeni, a početni čvor je označen kao trenutni čvor. Sada, umesto jednostavnog reda, koristi se red prioriteta. Ispituju se svi čvorovi koji su susedi trenutnom čvoru i izračunava se njihova udaljenost od početnog čvora kroz trenutni čvor. Susedni čvorovi se ubacuju u red prioriteta koristeći udaljenost ivice, tako da će čvor sa najmanjom udaljenosti od početnog čvora biti prvi ubačen u red prioriteta. Kada se ispitaju svi susedi trenutnog čvora, trenutni čvor se označava kao posećen i neće se ponovo ispitivati. Susedni čvor sa najnižom vrednošću udaljenosti označen je kao novi trenutni čvor i postupak se ponavlja sve dok cilj ne bude označen kao posećen ili svi čvorovi budu označeni kao posećeni a da cilj nije pronađen.

C. GREEDY BFS algoritam

Pohlepna pretraga po najboljim osobinama ili čista heuristička pretraga (eng. *Greedy Best-first Search*) je algoritam koji istražuje graf tako što širi pretragu na sledeći najbolji čvor koji je izabran pomoću nekog navedenog pravila. Ovaj algoritam procenjuje čvorove korišćenjem heurističke funkcije, f (n) = h(n), gde je h(n) procenjena distanca od datog čvora do ciljnog čvora. Ova jednakost je ono što čini ovaj algoritam „pohlepnim“ (eng. *greedy*).

Greedy BFS koristi takođe red sa prioritetom, ali za razliku od Dijkstrinog algoritma koji koristi pravu udaljenost između čvorova za sortiranje čvorova u redu sa prioritetom, Greedy BFS algoritam koristi heuristiku za sortiranje. Kada se koristi heuristika za sortiranje, čvor koji je najbliži cilju će biti prvi istražen, ne uzimajući u obzir dosadašnju pređenu distancu. Čvorovi koji su najbliži cilju imaju najveći prioritet, dok čvorovi najbliži startnom čvoru imaju najmanji prioritet [11].

D. A algoritam*

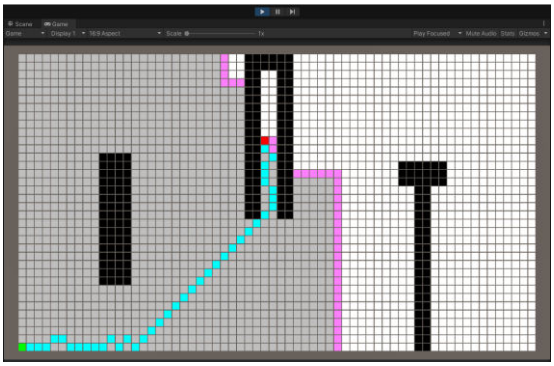
A* je informisani algoritam za pretragu, ili pretragu po najboljim osobinama, što znači da je namenjen za rad sa težinskim grafovima. Počevši od određenog početnog čvora grafa, ima za cilj da pronađe put do datog ciljnog čvora koji ima najmanji trošak. To radi tako što održava stablo svih putanja koje potiču iz početnog čvora i proširuje te putanje ivicu po ivicu dok se ne zadovolji njegov kriterijum završetka, odnosno ne stigne do ciljanog čvora [12].

A* je praktično kombinacija Dijsktrinog i Greedy BFS algoritma. Za razliku od Dijkstrinog algoritma, A* za računanje prioriteta koristi heurističku funkciju $f(n) = g(n) + h(n)$. Gde je g(n) distanca od početnog do datog čvora, a h(n) je procenjena distanca od datog do ciljnog čvora. Treba pomenuti da h(n) ne uzima u obzir zidove, odnosno prepreke do cilja, tako da prava distanca može biti dosta veća od procenjene [13], [14].

Na svakoj iteraciji svoje glavne petlje, A* treba da odredi koje od svojih putanja da se proširi. Određivanje putanje koja će se proširiti vrši se na osnovu cene putanje i procene troškova potrebnih za produženje putanje sve do cilja. U svakoj iteraciji ovog algoritma, čvor sa najnižom f(n) vrednošću se uklanja iz reda, f i g vrednosti njegovih suseda se ažuriraju u skladu sa tim, a ovi susedi se dodaju u red. Algoritam se nastavlja sve dok uklonjeni čvor ne bude ciljni čvor.

III. SIMULATOR ZA PRONALAŽENJE NAJKRAĆEG PUTA

Simulator za pronalaženje najkraćeg puta u grafu je aplikacija napisana na programskom jeziku C# uz pomoć Unity razvojnog okruženja. Korišćen je Microsoft Visual Studio kao editor za pisanje koda. Unity je najpopularnije više-platformsko razvojno okruženje za video igre. Verzija razvojnog okruženja korišćena u ovom radu je Unity 2021.3.3f1. Simulator koristi grid graf predstavljen 2D mapom, koji je prikazan na Slici 3. gde svako polje na mapi predstavlja čvor u grafu [15]. Simulator simulira rad svih algoritama predstavljenih u ovom radu na različitim mapama, meri vreme izvršenja i dužinu najkraćeg puta za svaki algoritam.



Slika 3. Simulator za pronalaženje najkraćeg puta u grafu

Svaka boja polja označava trenutno stanje čvora, a značenja su sledeća:

- zelena - startno polje
- crvena - ciljno polje
- bela - otvorena, odnosno neistražena polja
- siva - zatvorena, odnosno istražena polja
- crna - zidovi
- magenta - granična polja
- cijan - najkraći put do cilja

Pathfinder.cs skripta sadrži i korutinu u kojoj se izvršavaju algoritmi pretrage. Korutina omogućava raspoređivanje zadataka u nekoliko okvira. U Unity-u, korutina je metoda koja može pauzirati izvršenje i vratiti kontrolu Unity-u, ali zatim nastaviti tamo gde je stala na sledećem okviru. U većini slučajeva kada se pozove metoda, ona se izvršava do kraja, a zatim vraća kontrolu metodi koja je poziva i sve opcione povratne vrednosti. To znači da se svaka radnja koja se odvija unutar metode mora desiti unutar jednog ažuriranja okvira. U situacijama kada je potrebno koristiti poziv metode da bi se zadržala proceduralna animacija ili niz događaja tokom vremena, može se koristiti korutina. Važno je napomenuti da korutine nisu niti. Sinhrone operacije koje se pokreću unutar korutine se i dalje izvršavaju na glavnoj niti.

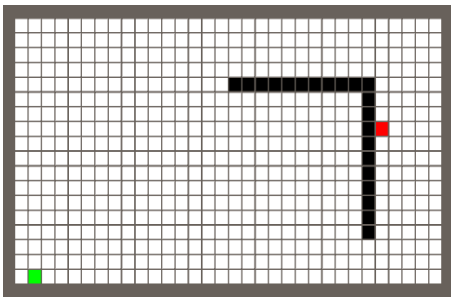
Na početku same korutine nalazi se promenljiva *timeStart* koja čuva vrednost sistemskog vremena na samom početku izvršavanja algoritma pretrage. Pristupanje sistemskom vremenu se vrši pomoću klase *Time* i *realtimeSinceStartup* metode. Na samom kraju bloka korutine, nakon *while* petlje, štampa se u konzoli razlika trenutnog sistemskog vremena i vremena sa početka korutine i time se dobija ukupno vreme potrebno za izvršenje algoritma pretrage u sekundama.

Dužina pređenog puta se čuva u parametru *distanceTraveled* svakog objekta istraženog čvora. Kada algoritam pronađe ciljni čvor u konzoli se štampa njegov parametar *distanceTraveled*.

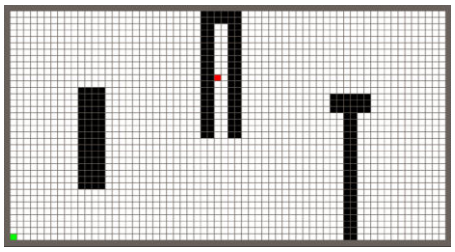
Vreme izvršavanja algoritma zavisi od toga da li se pretraga animira ili ne. Za potrebe animacije koristi se *yield* naredba koja čeka određeno vreme definisano parametrom *timeStep* pre nego što pređe na sledeću iteraciju. Vrednost parametra *timeStep* može se menjati u inspektoru na *DemoController*-u. Podrazumevana vrednost *timeStep* je 10*ms*. Da bi se za potrebe testiranja uključile ili isključile animacije da bi se dobilo što preciznije vreme izvršavanja algoritma uvedena je *bool* promenljiva *showIterations*.

IV. REZULTATI TESTIRANJA I ANALIZA REZULTATA

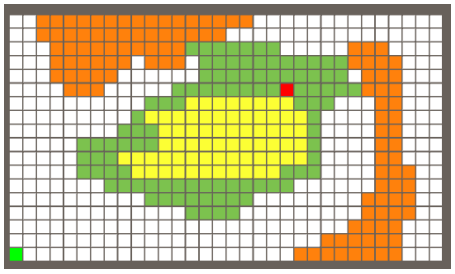
Uz pomoć prethodno opisanog simulatora u poglavlju 4, izvršeno je testiranje efikasnosti svih algoritama opisanih u ovom radu. Za potrebe testiranje koriste se tri različite mape, prikazane na Slikama 4, 5 i 6. Svaki algoritam je testiran i sa uključenim i sa isključenim animacijama.



Slika 4. Mapa za testiranje algoritama dimenzija 32x18



Slika 5. Mapa za testiranje algoritama dimenzija 64x36

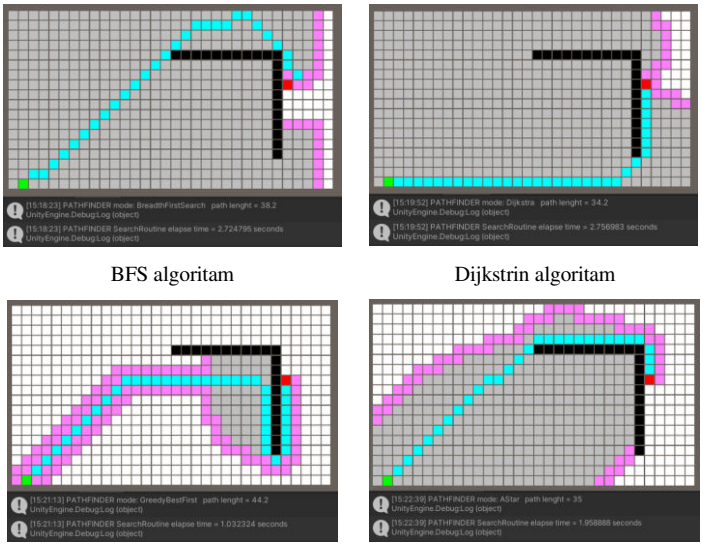


Slika 6. Mapa za testiranje algoritama sa terenom dimenzija 32x18

Kao što se može videti na slici 7. algoritmi će biti testirani na mapi sa terenom, koji može biti lakše prohodan (zelena boja), srednje prohodan (žuta boja) i teže prohodan (narandžasta boja). Svako od ovih polja dodaje određenu težinu, lakše prohodno dodaje težinu 1, srednje prohodno dodaje težinu 2 i teže prohodno težinu 3.

A. Rezultati testiranja na mapi 32x18

Rezultati testiranja algoritama na mapi dimenzija 32x18 su prikazani na Slici 7.



BFS algoritam | Dijkstrin algoritam

GREEDY BFS algoritam | A* algoritam

Slika 7. Rezultati testiranja algoritama na mapi dimenzija 32x18

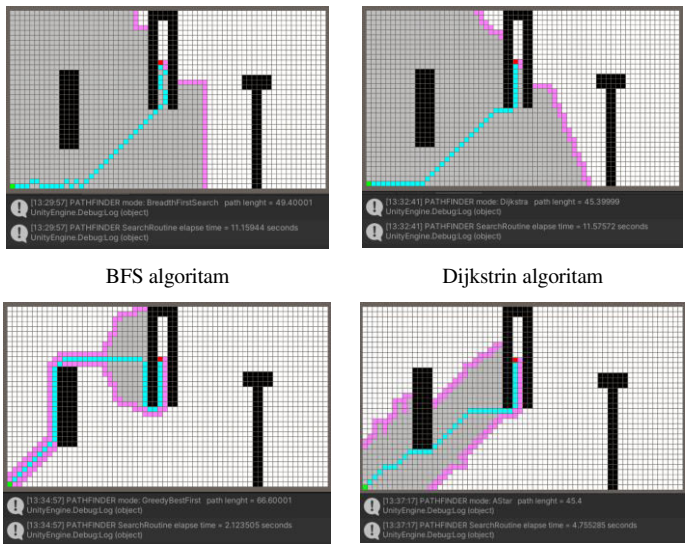
TABELA I. REZULTATI TESTIRANJA NA MAPI 32X18

Algoritam	Vreme sa animiranjem (s)	Vreme bez animiranja (s)	Pređeni put
BFS	2.724795	0.01019549	38.2
Dijkstra	2.756983	0.009798527	34.2
Greedy BFS	1.032324	0.00738585	44.2
A*	1.958888	0.009003162	35

Na osnovu ovih rezultata možemo zaključiti da na ovoj mapi najkraći put pronalazi Dijkstrin algoritam, čija je dužina pređenog puta 34.2, dok je dužina pređenog puta kod A* zanemarljivo veća, ali je zato A* skoro za 1s brži od Dijsktrinog algoritma kada su animacije uključene. Može se primetiti da je Dijkstrin algoritam u ovom slučaju slično kao i BFS pretražio skoro ceo graf dok nije stigao do cilja. Greedy BFS je ubedljivo najbrži, ali je zato pređeni put ubedljivo najveći. Tabelarni prikaz rezultata prikazan je u tabeli 5.1.

B. Rezultati testiranja na mapi 64x36

Rezultati testiranja algoritama na mapi dimenzija 64x36 su prikazani na Slici 8.



BFS algoritam | Dijkstrin algoritam

GREEDY BFS algoritam | A* algoritam

Slika 8. Rezultati testiranja algoritama na mapi dimenzija 64x36

Na osnovu ovog rezultata merenja ponovo se može zaključiti da je Greedy BFS najbrži ali što se dužine pređenog puta tiče, najmanje efikasan. Dok A* daje najefikasniji rezultat što se tiče pređenog puta i vremena sa uključenom animacijom. Tabelarni prikaz rezultata testiranja prikazan je u tabeli 5.2.

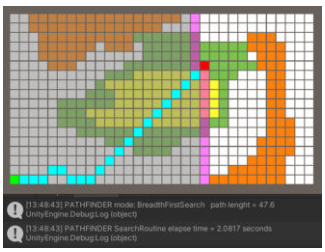
TABELA II. REZULTATI TESTIRANJA NA MAPI 64X36

Algoritam	Vreme sa animiranjem (s)	Vreme bez animiranja (s)	Pređeni put
BFS	11.15944	0.02224422	49.40001
Dijkstra	11.57572	0.002276707	45.39999
Greedy BFS	2.123505	0.00835371	66.60001
A*	4.755285	0.009902477	45.4

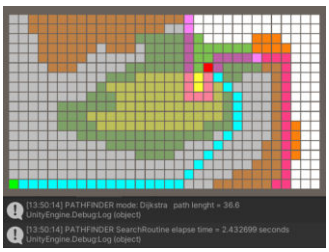
C. Rezultati testiranja na mapi sa terenom

Rezultati testiranja algoritama na mapi sa terenom su prikazani na Slici 9.

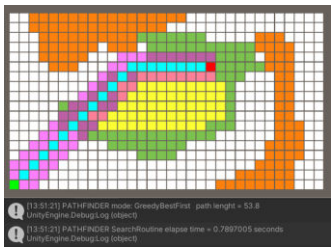
Prva stvar koja se može u očiti na osnovu ovih rezultata je da BFS potpuno ignoriše teren, što je i očekivano ponašanje BFS algoritma, zato što ovaj algoritam ne radi na težinskim grafovima, tako da korišćenje BFS algoritma za ovakav tip mape ne daje validne rezultate. A* i Dijkstra su i u ovom slučaju efikasniji, jer u svakom koraku procenjuju težinu susednih polja i zaobilaze teže prohodan teren. Tabelarni prikaz rezultata testiranja prikazan je u tabeli 5.3.



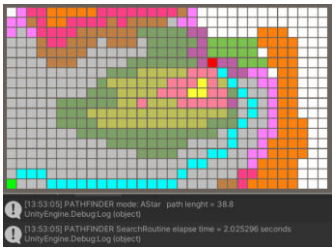
BFS algoritam



Dijkstrin algoritam



GREEDY BFS algoritam



A* algoritam

Slika 9. Rezultati testiranja algoritama na mapi sa terenom

TABELA III. REZULTATI TESTIRANJA NA MAPI SA TERENOM

Algoritam	Vreme sa animiranjem (s)	Vreme bez animiranja (s)	Pređeni put
BFS	2.0817	0.008274078	47.6
Dijkstra	2.432699	0.009364605	36.6
Greedy BFS	0.7897005	0.006685495	53.8
A*	2.025296	0.009136438	38.8

ZAKLJUČAK

U ovom radu su opisana četiri algoritma za pronalazak najkraćeg puta u grafu, od koga zapravo dva samo nalaze optimalnu putanju, a to su A* i Dijkstrin algoritam. Dijkstrin algoritam uvek pronalazi optimalne putanje, ali se može zaključiti na osnovu rezultata testova da potencijalno može pretražiti mnogo čvorova dok pronađe cilj. Sa druge strane, A* koristi heuristiku za određivanje koje čvorove je bolje sledeće pretražiti, što mu daje osećaj „pravca“ ka ciljanom čvoru, čime se izbegavaju mnoga nepotrebna istraživanja čvorova.

Greedy BFS algoritam iako je uvek najbrži ne uspeva da pronađe optimalne putanje. Međutim čak iako ne uspeva da nađe optimalne putanje, uvek će pronaći put ako postoji. S obzirom na to koliko je brži od ostalih algoritama, može biti veoma dobar izbor ukoliko je jedino potrebno dokazati da li su dva čvora povezana, odnosno proveriti da li postoji putanja između dva čvora.

Na osnovu testova može se zaključiti da je BFS algoritam beskoristan kada je u pitanju pretraga terena, jer on ne zna za pojam težine i svi čvorovi su za ovaj algoritam isti. Jednostavno pretražuje ceo graf redom dok ne dođe do cilja, i ovaj algoritam je najmanje efikasan od svih opisanih.

Postoji mnogo varijanti A* algoritma, ali osnovna ideja je da se radi o informisanoj pretrazi zasnovanoj na heuristici. Još uvek nije izmišljen novi algoritam koji pouzdano pronalazi optimalne puteve bolje od A* algoritma, čak i nakon više od 50 godina od kako je otkriven. Zato je i danas A* defakto standard kada je u pitanju pronalaženje najkraćeg puta u mnogim aplikacijama kao što je su razvoj video igara i robotika [12].

U daljem radu autori planiraju unapređenje razvijenog simulatora kroz dodavanje mogućnosti manipulacije brojem čvorova i grana. Na taj način korisnici će moći samostalno da definišu i analiziraju neograničeni broj različitih mapa.

LITERATURA

[1] D. Monzonís Laparra, „PATHFINDING ALGORITHMS IN GRAPHS AND APPLICATIONS“, Facultat de Matemàtiques i Informàtica, Universitat de Barcelona

[2] R. J. Wilson, „Introduction to Graph Theory, Fourth Edition“, Longman Group Ltd, 1996.

[3] X. Cui, H. Shi, „A*-based Pathfinding in Modern Computer Games“, IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, January 2011

[4] J. Young Hwang, J. Song Kim, S. Seok Lim and K. Ho Park, "A fast path planning by path graph optimization," in IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 33, no. 1, pp. 121-129, Jan. 2003, doi: 10.1109/TSMCA.2003.812599.

[5] J. Sidi, L. W. Fa and S. N. Junaini, "Shortest Path Simulation Using Interactive SVG Map," 2009 International Conference on Computer Technology and Development, 2009, pp. 6-9, doi: 10.1109/ICCTD.2009.204.

[6] L. Babel, Flight path planning for unmanned aerial vehicles with landmark-based visual navigation, Robotics and Autonomous Systems, Volume 62, Issue 2, 2014, Pages 142-150, ISSN 0921-8890, https://doi.org/10.1016/j.robot.2013.11.004.

[7] Unity Documentation, https://docs.unity3d.com/Manual/index.html

[8] C# Documentation, https://learn.microsoft.com/en-us/dotnet/csharp/

[9] Breadth First Search or BFS for a Graph, https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[10] Dijkstra’s Algorithm, https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[11] Greedy Best first search algorithm, https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/

[12] Introduction to A* Pathfinding, https://www.raywenderlich.com/3016-introduction-to-a-pathfinding

[13] A* Search Algorithm in Artificial Intelligence, https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/

[14] A* Search Algorithm, https://www.geeksforgeeks.org/a-search-algorithm/

[15] N. Azad & M. Farzad. (2015). Simulation and Comparison of Efficency in Pathfinding algorithms in Games. Ciência e Natura. 37. 230. 10.5902/2179460X20778.

ABSTRACT

This paper presents the results of the analysis of the most famous path finding algorithms (BFS, Dijkstra’s, Greedy BFS and A* algorithm) and their performances. To analyze the presented algorithms, a simulator was created for finding the shortest path between two points with the support of the Unity development platform and using the C# programming language. The graphical representation of the algorithm's operation is shown using a two-dimensional network of interconnected fields, where the notion of weight nodes and walls can be translated into the structure of a directed weight graph.

COMPARATIVE ANALYSIS OF PATH FINDER ALGORITHMS USING UNITY DEVELOPMENT PLATFORM

Nikola Vukotić, Slavimir Stošović