ĪSĀKĀ CEĻA MEKLĒŠANAS ALGORITMU A* UN DEIKSTRAS PIELIETOŠANAS EFEKTIVITĀTES ANALĪZE DIVDIMENSIJU REŽĢĪ

EFFICIENCY COMPARISON OF PATHFINDING ALGORITHMS A AND DIJKSTRA'S IN TWO DIMENSIONAL GRID*

Autors: **Intars ČESLIS**, e-pasts: intars96@gmail.com
Zinātniskā darba vadītājs: Dr.sc.ing., docents **Sergejs KODORS**
Rēzeknes Tehnoloģiju akadēmijas Inženieru fakultāte

Abstract. *The shortest path finding algorithms are applied more and more in different industries. There are many pathfinding algorithms, but the most of them differ from their modified versions. The goal of this work was to create A* and Dijkstra's pathfinding algorithms and compare their execution time.*

Keywords: *A*, Dijkstra's, pathfinding algorithm*

Ievads

Īsākā ceļa meklēšanas algoritmi tiek attīstīti un izmantoti arvien vairāk industrijā. Eksistē vairāki ceļa meklēšanas algoritmi, bet katrs no tiem atšķiras ar noteiktiem uzlabojumiem.

Darba mērķis ir realizēt A* un Deikstras algoritmus un salīdzināt to izpildes laiku. Darba izstrādes gaitā tika iegūtas zināšanas par šiem abiem algoritmiem, un to darbības principu. Eksperimentāli tika konstatēts, ka A* algoritms ir ātrāks par Deikstras algoritmu.

A* algoritms

A* algoritms ir populārs ceļa meklēšanas algoritms, kas paredzēts īsākā ceļa meklēšanai starp diviem punktiem, kas tiek saukti par mezgliem. A* meklē no visiem iespējamajiem ceļiem pašu izdevīgāko. Katrā iterācijā algoritmam jāizvēlas vienu no virzieniem. Tādējādi, tas izmanto funkciju $F = G + H$, kur F - parāda cik garš ir atrastais ceļš, ja iet caur noteikto mezglu, G - parāda cik izmaksās tikt no sākuma punkta līdz nākošajam izvēlētajam mezglam, H - parāda cik izmaksās no tekošā mezgla tikt līdz beigu mezglam.

A* algoritma pseidokods:

1) Tiek izveidots saraksts ar visiem mezgliem, kuri netika apstrādāti, un saraksts ar visiem mezgliem, kuri tika apstrādāti.
2) Tiek izveidots saraksts ar apskatītiem mezgliem, kur pirmais elements ir ceļa sākuma punkts.
3) Sākuma punktam G vērtību piešķir 0.
4) Sākuma punktam H vērtība tiek atrasta starp sākumpunktu un galapunktu, pēc izvēlētās heiristiskās metodes.
5) Tiek saskaitīts G un H, lai iegūtu F sākuma punkta vērtību.
6) Tiek izvēlēts mezgls ar mazāko F vērtību un tas tiek ievietots apskatīto mezglu sarakstā.
7) Ja pašreizējais mezgls ir vienāds ar galapunktu, algoritms beidz darbu, jo ceļš ir atrasts.
8) Tiek izņemts pašreizējais mezgls no saraksta ar neapstrādātajiem mezgliem.
9) Tiek ievietot pašreizējais mezgls apstrādāto mezglu sarakstā.
10) Pašreizēja mezgla kaimiņam tiek veikti šādi soļi:
11) Ja kaimiņš ir neapstrādātajā sarakstā, tad pāriet pie nākošā kaimiņa.
12) Tiek izveidots mainīgais ***izmēģinājumaG***, kas glabā tekošā mezgla G vērtību un attālumu start pašreizējo mezglu un kaimiņa summu.
13) Ja kaimiņš nav atrasts neapstrādāto mezglu sarakstā, tad to pievieno. H vērtība kaimiņam ir attālums starp kaimiņu un galapunktu, izmantojot izvēlēto heiristisko metodi.

https://doi.org/10.17770/het2020.24.6747

14) Ja mainīgajam ***izmēģinājumaG*** vērtība ir mazāka par kaimiņa *G* vērtību, tad jaunizveidotais mainīgais ***pozitīvsRez*** ir *patiess*, ja ir otrādi, tad ***pozitīvsRez*** pieņem vērtību *nepatiess*.
15) Ja ***pozitīvsRez*** ir *patiess*, tad pašreizējais mezgls tiek ievietots sarakstā ar apskatītajiem mezgliem. Kaimiņa *G* vērtību piešķir no mainīgā ***izmēģinājumaG*** un *F* vērtību piešķir, saskaitot *G* un *H* vērtības.
16) Atkārto soļus 6) - 15), kamēr neapstrādāto mezglu saraksts nav tukšs.
17) Algoritms beidz savu darbu, ceļš netika atrasts.

Pareizi izvēlēta heiristiskā metode var nodrošināt lielāku A* ātrdarbību. Ja *H* vērtība ir vienāda ar izmaksām, lai nokļūtu līdz galapunktam. A* vienmēr seko ideālākajam ceļam, netērējot laiku, apejot nevajadzīgus mezglus. *H* vērtība vienmēr noteiks ceļa meklēšanas ātrdarbību, ja vērtība ir lielāka par reālo izmaksu. A* vienmēr atradīs vislabāko iespējamo ceļu, bet, ja *H* vērtība ir mazāka par reālo izmaksu, tad A* ātrdarbība būs salīdzināma ar Deikstras algoritmu. A* veido divus sarakstus, sarakstu ar visiem apstrādātajiem mezgliem un sarakstu ar neapstrādātajiem. Katrs mezgls satur trīs vērtības *F*, *G* un *H*. Papildus šīm vērtībām, mezglam jāglabā sevī iepriekšējais mezgls, lai varētu noteikt ceļu līdz tekošajam mezglam.

Deikstras algoritms

Deikstras algoritms ir ceļa meklēšanas algoritms, kas meklē īsāko iespējamo ceļu, pielietojot grafus. Oriģināli Deikstras algoritms tika izveidots starp divu mezglu ceļa meklēšanai, bet pārsvarā to izmanto priekš ceļa meklēšanas starp sākuma mezgla un beigu mezgla. Oriģināli Deikstras algoritms neizmantoja prioritātes sarakstu, kas nozīme ka katram mezglam netika piešķirta jebkāda ceļa izmaksas vērtība.

Deikstras algoritma pseidokods:

1) Tiek izveidots saraksts ar neapskatītiem mezgliem.
2) Katram mezglam tiek noteikta izmaksas vērtība no iepriekšējā mezgla līdz šim. Mezgls tiek ievietots neapskatīto mezglu sarakstā.
3) Sākotnējam mezglam tiek piešķirta izmaksas vērtība 0.
4) Tiek izvēlēts mezgls-kaimiņš ar mazāko vērtību un tas tiek izņemts no saraksta.
5) Izvēlētais mezgls tiek pārbaudīts, vai tas nav galapunkts, ja ir, tad algoritms tiek apturēts, jo ceļš ir atrasts.
6) Tiek atrasta ceļa izmaksa katram esošajam kaimiņam, ja tekošā izmaksa ir lielāka par iespējamo, tā tiek aizvietota un iepriekšējais mezgls tiek aizvietots ar tekošo izvēlēto. Atkārto, kamēr visi kaimiņi netiks apskatīti.
7) Soļi 4) - 6) tiek atkārtoti, kamēr neapskatīto mezglu saraksts nebūs tukšs.
8) Algoritms beidz savu darbu, jo ceļš netika atrasts.

Deikstras algoritms iestātā visus mezglus kā neapskatītosar vērtību “bezgalība”, bet sākuma mezglam - 0. Tad tiek pārskatīti blakus mezgli izvēlētajam mezglam, un tiem tiek piešķirta ceļa izmaksa, ja mezgla ceļa izmaksa ir lielāka par atrasto ceļa izmaksu, tā tiek aizvietota ar atrasto ceļa izmaksu. Kad visi blakus esošie kaimiņi tika apskatīti, izvēlētais mezgls tiek atzīmēts kā pārskatīts un tiek izņemts no neapskatītā saraksta. Algoritms var beigt darbību gan apskatot visus mezglus un atrodot pašu izdevīgāko ceļu, gan atrodot pašu pirmo, bet atkarībā no izvēlēta varianta, tā ātrdarbība var atšķirties.

Materiāli un metodes

A* un Deikstras algoritmi tika izvēlēti priekš salīdzināšanas. Tika izmantots 2D masīvs, kas tiek aizpildīts ar tukšumiem un sienām. Sienas tiek izvietotas nejaušā secībā. Priekš A* tika izmantota Euklīda attāluma noteikšanas funkcija (1).

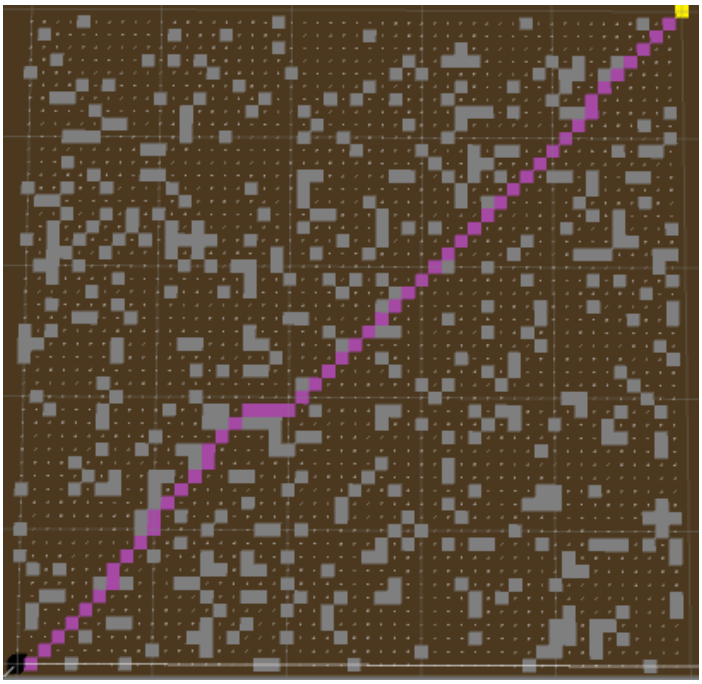
$$H = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \quad (1)$$

Tika izmantots 50x50 režģis. Algoritmiem tika atļauts izvēlēties ceļu pa diagonāli.

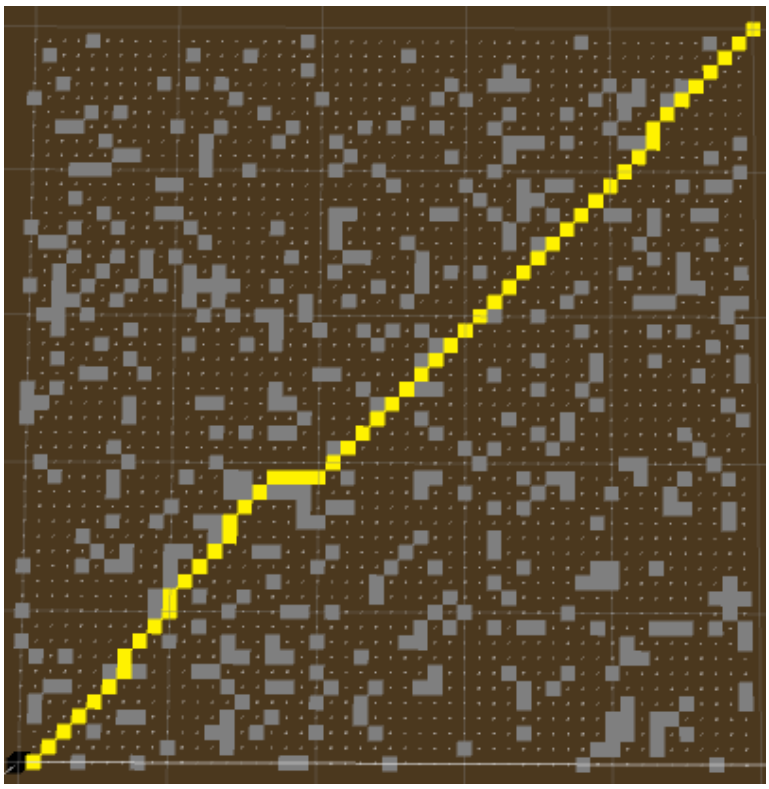
Eksperimenti tika veikti, pielietojot portatīvo datoru ar *Intel i7-5500U* procesoru un *8GB* operatīvās atmiņas.

Rezultāti un izvērtējums

Deikstras algoritma darbība ir vizuāli parādīta 1. attēlā, A* algoritms – 2. attēlā. Violetā krāsa attēlo Deikstras algoritma atrasto ceļu, bet dzeltenā krāsa – A* atrasto ceļu.

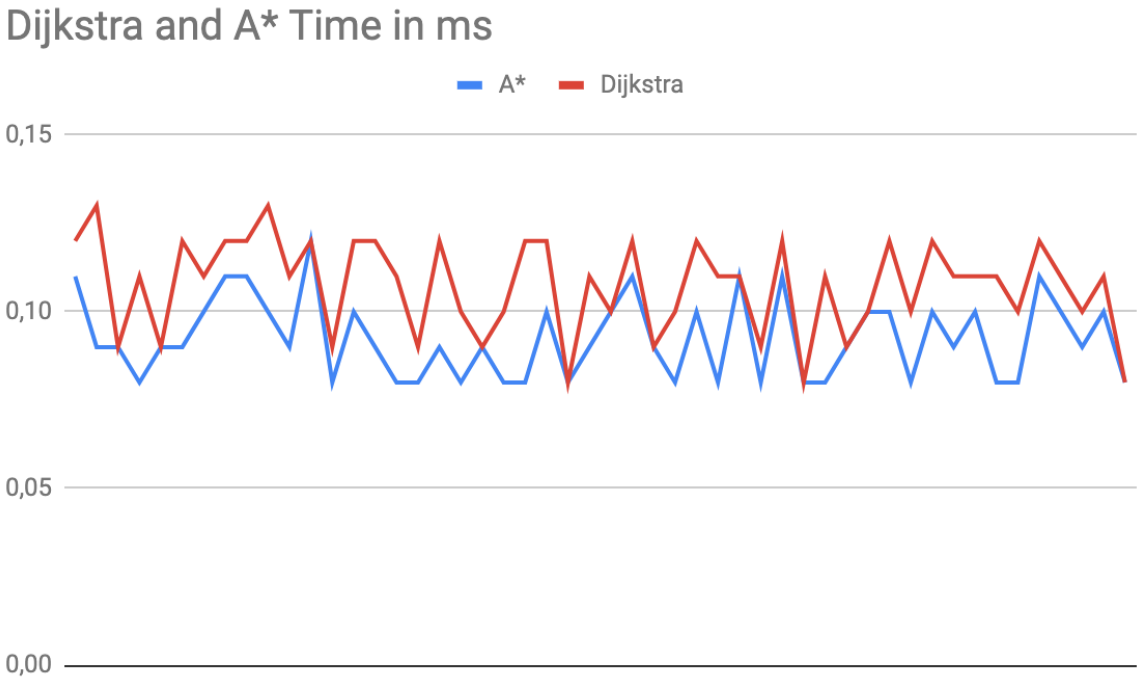


1. att. Deikstras meklēšanas algoritma rezultāts



2. att. A* meklēšanas algoritma rezultāts

3. attēlā ir attēlots A* un Dijkstras algoritmu ceļa meklēšanas laiks.



3. att. A* un Dijkstras algoritmu ceļa meklēšanas laiks

Secinājumi

Šajā darbā tika realizēti Deikstras un A* algoritmi un tika salīdzināta to efektivitāte. A* ātrdarbības pieaugums salīdzinājumā ar Deikstras algoritmu ir ievērojams, tāpēc tiek uzskatīts, ka 2D režģī ir vērts Deikstras algroritma vietā izmantot A*. Pēc darba rezultāta tika izspriests, ka A* varētu veikt daudz ātrāku darbību, ja grafs būtu daudz stiprāk aizpildīts ar nepārejamiem mezgliem. Šo pētījumu varētu turpināt, salīdzinot citus ceļa meklēšanas algoritmus.

Summary

In this work were created Dijkstra's and A algorithms and compared their efficiency. A* workspeed is better than Dijkstra's algorithm in 2D grid. This research can be continued using different pathfinding algorithms and testing their efficiency. Also nodes could be updated to different types that can increase or decrease speed movement of agents.*