LABIRINTU ĢENERĒŠANAS ALGORITMU SALĪDZINĀJUMS
COMPARISON OF MAZE GENERATION ALGORITHMS

Autori: **Igors Boldovs**, **Jurijs Vinogradovs** e-pasts: toag.iggy@gmail.com, jurijs.vinogradovs@gmail.com, +37129950173, +37127109221
Zinātniskā darba vadītājs: **Pēteris Grabusts, Dr.sc.ing.,** e-pasts: peteris.grabusts@rta.lv
Rēzeknes Tehnoloģiju akadēmija, Atbrīvošanas aleja 115, Rēzekne, Latvija

Abstract. *The author in this work provides insight into the Maze generation. Themes which are discussed are Eller's algorithm, Kruskal's algorithm and Recursive backtracking algorithm. All this algorithms were compared.*

Keywords: *Eller's algorithm, Kruskal's algorithm, Maze generation, Recursive backtracking*

Ievads

Labirintu ģenerēšana ir ļoti interesants un aizraujošs process. Mūsdienās izmanto ļoti daudzus un dažādus labirintu ģenerēšanas algoritmus, kur katrs algoritms ir savā ziņā īpašāks un efektīvāks, tāpēc autori bija ļoti ieinteresēti izvēlēties labirintu ģenerēšanas algoritmus, kurus varētu salīdzināt uz ātrdarbību un aplūkot katra algoritma īpatnības. Rezultātā tikai izvēlēti trīs labirintu ģenerēšanas algoritmi: Ellera, Kruskala un Rekursīvās atkāpšanas algoritmi. Katra algoritma darbības princips atšķiras no cita, bet mērķis visiem algoritmiem ir vienāds. Lai salīdzinātu dotos algoritmus, autori izlēma izmantot programmu *Deadalus*, kas dod iespēju analizēt ģenerētos labirintus, eksperimentā izmantoto labirintu izmēri ir 100x100.

Ellera algoritms

Ellera algoritms labirintu veidošanai ir ļoti īpašs. Tas ir ne tikai ātrāks par pārējiem labirintu veidošanas algoritmiem, bet arī ir visefektīvākais atmiņas ziņā [1]. Nav nepieciešams, lai viss labirints būtu novietots datora atmiņā, tiek tikai proporcionāli rindas izmēram pieprasīta atmiņa. Tas ļauj veidot liela izmēra labirintus ar ierobežotu atmiņas iespēju. Algoritms veido labirinta vienu rindu, kad rinda tiek ģenerēta, algoritms vairs uz to „neskatās". Katra rindas šūna ir iekļauta atsevišķā kopā - ja starp šūnām ir ceļš, tad skaitās, ka tie atrodas vienā kopā. Šī informācija ļauj ceļa fragmentiem būt sadalītiem pašreizējajā rindā, netaisot cilpas un izolācijas.

Darbības princips

Piezīme: Tiek pieņemts, ka visām kreisās puses šūnām ir kreisā robeža un visām labās puses šūnām ir robeža pa labi[2].

1. Izveido pirmo rindu. Neviena šūna nepieder nekādai kopai.
2. Šūnām, kas nepieder nekādai kopai, piešķir savu unikālo kopu.
3. Izveido labās robežas, virzoties no kreisās puses uz labo.
    A. Izlases veidā izvēlas iespēju - pievienot robežu vai nē.
        1. Ja tekošā šūna un šūna pa labi pieder vienai kopai, tad izveido robežu starp tām.
        2. Ja tiek izlemts nepievienot robežu, tad apvieno kopas, kurā atrodas pašreizējā šūna un šūna pa labi.
4. Izveido robežas lejup, kustoties no kreisās puses uz labo pusi.
    A. Izlases veidā izvēlas pievienot robežu vai nē. Jāpārliecinās, ka katrai kopai ir vismaz viena šūna bez apakšējās robežas (lai novērstu izolētas teritorijas).
        1. Ja šūna ir viena vienīgā pašreizējā kopā, tad robeža lejup netiek pievienota.
        2. Jā šūna ir vienīga šūna bez apakšējās robežas dotajā kopā, tad robeža lejup netiek pievienota.

http://dx.doi.org/10.17770/het2016.20.3508

5. Jāizlemj, vai turpināt pievienot rindas vai pabeigt labirintu:
    A. Ja izlemts pievienot vēl vienu rindu, tad:
        1.Izvada pašreizējo rindu.
        2.Izdzēš visas labās robežas.
        3.Izdzēš šūnas ar apakšējo robežu no to kopas.
        4.Izdzēš visas apakšējais robežas.
        5.Turpina no 2. soļa.
    B. Ja izlemts pabeigt labirintu, tad:
        1.Pievieno apakšējo robežu katrai šūnai.
        2.Virzoties no kreisās puses uz labo:
            1. Ja pašreizējā šūna un šūna pa labi ir no dažādām kopām, tad:
                a. Izdzēš labo robežu.
                b. Apvieno pašreizējās šūnas kopu ar labās šūnas kopu.
                c. Izvada beigu rindu.

Kruskala algoritms

Kruskala algoritms ir metode, kas veido minimālo grafa koka aptveršanu svērtajā grafā [3]. Algoritms, kas tiek aprakstīts zemāk pēc šāda principa:

1. Izvieto visas grafa malas lielā stekā.
2. Iegūst malu ar zemāko svaru. Ja mala savieno nešķērsojošos kokus, savieno kokus. Pretējā gadījumā, mala netiek turpmāk apskatīta.
3. Atkārto, tiklīdz vairs nav beigušās malas.

Oriģinālā algoritma gadījuma (*random*) versija maina tikai otro soli, tā vietā, lai izvilktu malu ar zemāko svaru, tiek izņemta mala no steka, nejaušā gadījuma veidā. Izveidojot tādu izmaiņu algoritmā, tas veido diezgan pārliecinošus labirintus.

Rekursīvās atkāpšanās algoritms

Rekursīvās atkāpšanās (*Recursive backtracking*) algoritms ir ļoti ātrs labirintu ģenerācijas algoritms. Šim algoritmam būs nepieciešams pietiekams atmiņas apjoms, lai saglabātu visu labirintu atmiņu, tomēr tas pieprasa steka apjomu proporcionāli labirinta izmēram, tāpēc lieliem labirintiem tas var būt diezgan neefektīvs. Tomēr lielākai labirintu daļai tas darbojas lieliski [4].

Darbības princips:

1. Tiek izvelēta sākuma pozīcija laukumā.
2. Gadījuma veidā izvēlas sienu dotajā pozīcijā un „izgriež” ceļu pie blakus stāvošas šūnas, bet tikai tad, ja blakus stāvošā šūna vel nebija apmeklēta. Apmeklētā šūna kļūst par jauno pozīciju.
3. Ja visas blakus stāvošās šūnas tika apmeklētas, tad jāiet atpakaļ līdz tai šūnai, kurai nav „izgrieztu” sienu un darbības atkārtojas.
4. Algoritms beidzas, kas viss process atbalsta visu ceļu līdz sākuma pozīcijai.

Algoritmu salīdzinājums

Salīdzinot apskatītos labirintu ģenerācijas algoritmus, tika izmantota programma *Daedalus 3.1* [5], kas dod iespēju analizēt noģenerētos labirintus. Eksperimentā tika izmantoti labirinti ar izmēru 100x100. Var secināt, ka no izvēlētajiem algoritmiem ar savu uzdevumu visātrāk tika galā Ellera algoritms. Rekursīvās atkāpšanās algoritmam nākas izmantot ilgāku laiku uzdevuma risināšanai.

1. tabula

Algoritmu īpašības

Algoritms	Strupceļš	Fokuss	Atmiņa	Laiks	Risinājums
Ellera algoritms	28%	Ceļš/Siena	N*	20ns	4.2 %
Kruskala algoritms	30%	Ceļš/Siena	N^2	33ns	4.1 %
Rekursīvas atkāpšanas algoritms	10%	Ceļš	N^2	27ns	19.0 %

Tabulā 1. apkopoti labirintu ģenerācijas algoritmu īpašības. Kolonu apraksti ir sekojoši:

- **Strupceļš:** Aptuvenais strupceļa šūnu skaits (procentos).
- **Fokuss:** Vairums algoritmu var īstenot divos veidos -vai nu pievienojot sienas laukumā, vai arī izgriežot ceļu sienās. Ir algoritmi, kas izmanto šīs abas iespējas.
- **Atmiņa:** Šeit parādīts, cik papildus atmiņas ir vajadzīgs, lai īstenotu algoritmu. Daži algoritmi prasa atmiņu proporcionāli rindu skaitam (N); daži pieprasa atmiņu proporcionāli šūnu skaitam (N^2); algoritmi, kuri neprasa, lai viss labirints tiktu novietots atmiņā, ir apzīmēti ar zvaigznīti (N*).
- **Laiks:** Šī kolonna dod priekšstatu par to, cik laika aizņem labirintu ģenerācija - jo mazāks skaitlis, jo ātrāks algoritms. Šie skaitli ir ņemti ģenerējot 100x100 labirintu programmā *Daedalus*.
- **Risinājums:** Šis skaitlis norāda, cik šūnu procentuāli tiek izmantoti risinājuma ceļā. Šie skaitli ir ņemti ģenerējot 100x100 labirintu programmā *Daedalus*.

Secinājumi

Apskatot un apkopojot doto informāciju, var izdarīt šādus secinājumus:

1. Ellera algoritms ir ļoti efektīvs atmiņas izmantošanas veidā.
2. Veidojot lielus (100x100) labirintus, visātrāk ar savu uzdevumu tika galā Ellera algoritms.
3. Rekursīvās atkāpšanās algoritms ir ļoti ātrs, ja ir vajadzīgs ģenerēt neliela izmēra labirintu.
4. Kruskala algoritms ir diezgan lēns salīdzinājumā ar pārējiem algoritmiem.

Summary

Maze generation is very interesting and exciting process. Nowadays there is a lot of different maze generation algorithms, where each algorithm is some way effective and specific. Authors took 3 algorithms to compare – Eller's algorithm, Kurskal's algorithm and Recursive backtracking algorithm. After creating this publication, which is meant to introduce the audience to the fact, which of three maze generation algorithms is most efficient in terms of time, conclusion was made, that generating 100x100 maze, the Eller's algorithm was the fastest one. The Eller's algorithm needed only 20 ns, to generate 100x100 maze. It is very surprisingly quick and effective maze generation algorithm. Kruskal's algorithm completed its task within 33 ns, it's also a good result, but still relatively less than Eller algorithm parameters. Recursive backtracking algorithm generated 100x100 maze within 27 ns, and it is very good result, so given algorithm completed its task faster than Kruskal's algorithm, but slower than Eller's algorithm. In future, there is desire to perform this experiment with bigger count of maze generation algorithms, experiment conditions might be the same. But the size of maze could be much bigger than 100x100.

Literatūra

1. Think Labyrinth: Maze Algorithms: http://weblog.jamisbuck.org/2010/12/29/maze-generation-eller-s-algorithm

2. Eller's Algorithm: http://www.neocomputer.org/projects/eller.html
3. Maze generation: Kruskal's algorithm: http://weblog.jamisbuck.org/2011/1/3/maze-generation-kruskal-s-algorithm
4. Maze generation: Recursive backtracking: http://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking
5. Daedalus 3.1 http://www.astrolog.org/labyrnth/daedalus.htm