LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

CEĻA ATRAŠANA LABIRINTĀ

BAKALAURA DARBS

Autors: **Ivans Gorbunovs**

Studenta apliecības Nr.: 11152

Darba vadītājs: profesors Dr. dat. Jānis Zuters

RĪGA 2019

ANOTĀCIJA

Darbs demonstrē, kā var izmantot viļņu algoritmu ceļa atrašanai labirintā. Darbā tiek izpētīts viļņu algoritms, rezultātā ir izveidota lietotne, kurā ir realizēts viļņu algoritms un trīs tā modifikācijas. Salīdzinot modifikācijas, tiek secināts, ka 3. modifikācija ir visefektīvākā.

Atslēgas vārdi: viļņu algoritms, labirints, grafs, grafu teorija.

ABSTRACT

FINDING A PATH IN A MAZE

The present research paper deals with the investigation of wave algorithm for the solving of mazes. The paper aims to create a program that finds a path in the maze. Moreover, it aims to modify wave algorithm and to compare the wave algorithm and the productivity of its modifications.

The author of the paper developed the program for maze solving. Moreover, the wave algorithm and its modifications were implemented. The study has shown that third modification was the most effective.

Key words: wave algorithm, maze, graph, graph theory.

SATURA RĀDĪTĀJS

DEFINĪCIJAS, AKRONĪMI UN SAĪSINĀJUMI

Definīcija, akronīms vai saīsinājums	Paskaidrojums
Visual Basic	Programmēšanas valoda no Microsoft.
BFS	Meklēšana plašumā (angļu: breadth-first search).
DFS	Meklēšana dziļumā (angļu: depth-first search).

IEVADS

Autors ir sācis šo darbu ar vēlmi izpētīt viļņu algoritmu un izveidot lietotni, kas demonstrēs viļņu algoritma darbību. Par lietotnes pamatu bija paņemta labirinta ideja: ir dots labirints, sākuma un beigu punkti, un ir nepieciešams atrast ceļu (vai arī konstatēt, ka ceļš neeksistē).

Realizējot parasto viļņu algoritmu, radās ideja modificēt šo algoritmu, lai palielinātu to efektivitāti un samazinātu apmeklēto šūnu skaitu.

Rezultātā bija izveidotās 3 modifikācijas. Lai noskaidrotu, kura no modifikācijām ir vislabākā, bija izveidota testu sistēma un visas 4 viļņu algoritma realizācijas bija notestētas un analizētas.

Darba mērķi:

- Izveidot lietotni ceļa atrašanai labirintā izmantojot viļņu algoritmu;
- Modificēt viļņu algoritmu;
- Salīdzināt viļņu algoritma un tā modifikāciju produktivitāti.

Mērķu īstenošanai tika izvirzīti **uzdevumi:**

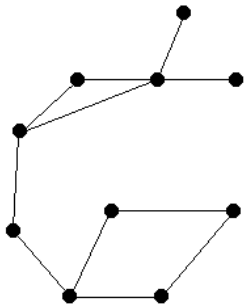
- Izstrādāt programmu ar viļņu algoritma realizāciju;
- Veikt viļņu algoritma modifikācijas, realizēt tās;
- Izstrādāt testu sistēmu, testēt oriģinālu viļņu algoritmu un tā modifikācijas, izveidot tabulu ar rezultātiem;
- Analizēt iegūtos rezultātus un izdarīt secinājumus.

Darbs sastāv no ievada, 2 nodaļām, rezultātiem, secinājumiem, izmantotās informācijas avotu saraksta, 14 pielikumu un dokumentārās lapas.

1. TEORĒTISKĀ DAĻA

1.1. GRAFI UN GRAFU TEORIJA

Grafs ir sakārtots kopu pāris $G = (V, E)$, kur V ir virsotņu kopa un E ir šķautņu kopa (skat. 1.1. att.). Katrai šķautnei (no kopas E) ir piekārtotas viena vai divas virsotnes (no kopas V), kuras sauc par šīs šķautnes galapunktiem un saka, ka šķautne savieno tās galapunktus.



1.1. att. **Grafs**

Grafu teorija ir diskrētās matemātikas nozare, kurā pēta grafu kombinatoriskās un topoloģiskās īpašības. Ar grafu teorijas elementiem var tikt risinātas zinātniskas un tehniskas problēmas.

Grafu teorija ļoti plaši tiek izmantota loģistika, elektrotehnikā un ģeogrāfisko karšu izstrādāšanā.

1.2. GRAFU PIEMĒRI

Grafs ir ideāls modelis daudzu uzdevumu risināšanai. Piemēram, kā grafu var attēlot AirBaltic lidojuma shēmu vai Rīgas ūdens kanalizācijas sistēmu.

Grafus var izmanot arī citās jomās un grafa veidā var attēlot dažādus uzdevumus un fiziskus objektus. Daudziem no tiem ir īpašs nosaukums. Piemēram:

1. Matemātisku apgalvojumu grafs (virsotnes - apgalvojumi, šķautnes - loģiskie secinājumi);
2. Funkcionālais grafs (virsotnes - kopas elementi, šķautnes – funkcijas darbība);
3. Lielumu-sakarību grafs (virsotnes - skaitliski lielumu un sakarības starp tiem, šķautnes - lieluma piedalīšanās sakarībā);
4. Metrikas grafs (virsotnes - jebkura veida fiziski vai nefiziski objekti vai to kopas, šķautnes - objektu ģeometriskā, strukturālā, funkcionālā vai evolucionārā tuvība; pielieto lielu kopu analīzē);
5. Sistēmas grafs (virsotnes - sistēmas komponentes, šķautnes - komponenšu mijiedarbība, pielieto sistēmu projektēšanā un analīzē);
6. Spēles grafs (virsotnes - spēles stāvokļi, šķautnes - spēles noteikumu atļautas pārejas (gājieni) starp stāvokļiem; pielieto spēļu uzvarošo stratēģiju izstrādāšanā);
7. Datortīkls (vispārīgā gadījumā - komunikāciju tīkls) (virsotnes - datori vai komunikāciju mezgli, šķautnes - sakaru līnijas; pielieto datortīklu projektēšanā un analīzē);

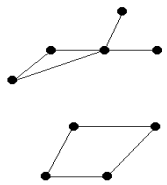
8. Sociālais grafs (virsotnes - cilvēki vai to kopas, šķautnes - pazīšanās, ekonomiskas vai cita veida attiecības; pielieto sabiedrības analīzē un attīstības plānošanā);
9. Darbinieku-pienākumu grafs (virsotnes - darbinieki un pienākumi vai darbi, šķautnes - attiecības, kas darbiniekiem pakārto to iespējamos pienākumus);
10. Ģenealoģiskais koks (virsotnes - cilvēki, šķautnes - ”vecāku-bērnu” attiecības);
11. Ceļu grafs (virsotnes - pilsētas, šķautnes - ceļi);
12. Ielu grafs (virsotnes - krustojumi, šķautnes - ielas);
13. Asinsvadu grafs (virsotnes - asinsvadu sazarojumi, šķautnes - asinsvadi).

Ja kādu uzdevumu var attēlot grafā veidā, tad grafu algoritmi ir pielietojami arī šim uzdevumam. Tāpēc algoritmi ar grafiem dod iespēju risināt dažāda sarežģītības līmeņa uzdevumus.

1.3. GRAFU VEIDI

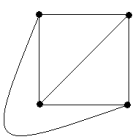
Ir daudz grafu veidu. Tomēr var definēt vissvarīgākos un visplašāk izmantotos grafu veidus [1, 2]:

1. Sakarīgs grafs (skat. 1.2. att.). Tas ir tāds grafs, kuram no jebkuras virsotnes var aiziet uz jebkuru citu virsotni, pārvietojoties tikai pa grafa šķautnēm. Nesakarīga grafa piemērs – skat. 1.2. att.



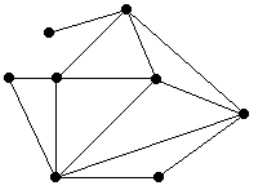
1.2. att. **Nesakarīgs grafs**

2. Planārs grafs (skat. 1.3. att.). Tas ir grafs, ko iespējams attēlot plaknē tā, ka tā šķautnes nekrustojas.



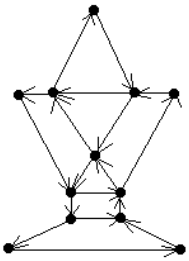
1.3. att. **Planārs grafs**

3. Neorientēts grafs (skat. 1.4. att.). Tas ir grafs, kura šķautnēm nav pakārtots virziens. Parasti ar terminu grafs pēc noklusēšanas tiek domāts neorientēts grafs.
   Par neorientēta grafa piemēru var minēt taku, pa kuru var iet abos virzienos.



1.4. att. **Neorientēts grafs**

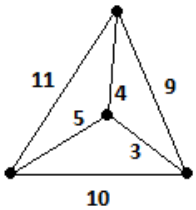
4. Orientēts grafs (skat. 1.5. att.). Tas ir grafs, kura katrai šķautnei ir piekārtots virziens. Labs orientēta grafa piemērs ir vienvirziena ceļš: transports var braukt tikai vienā virzienā.



1.5. att. **Orientēts grafs**

5. Grafi ar svariem (skat. 1.6. att.). Tas ir grafs, kura katrai virsotnei un/vai šķautnei var būt pakārtots skaitlis vai svars. Tas varētu būt gan pozitīvs, gan negatīvs, gan arī nulle. Ja kaut kādā uzdevumā tiek lietoti negatīvi svari, tad ne visi algoritmi ir pielietojami šim uzdevumam. Piemēram, Deikstras algoritms nestrādā ar negatīviem svariem, bet līdzīgs Belmana-Forda algoritms pieļauj negatīvus šķautņu svarus, tomēr ir lēnāks.

   Viens no grafa ar svariem piemēriem ir dzelzceļš. Grafa virsotnes var apskatīt, kā stacijas, bet šķautnes – kā ceļu. Par svaru šajā uzdevumā varētu uzskatīt attālumu starp stacijām vai biļetes cenu.



1.6. att. **Grafs ar svariem**

6. Multigrafs (skat. 1.7. att.). Tas ir grafs, kuram ir divas virsotnes, kas savienotas ar vairāk nekā vienu šķautni.

1.7. att. **Multigrafs**

Kā jau iepriekš tika minēts, ir ļoti daudz grafu veidu, taču uzdevumos ar grafiem īpaši bieži tiek izmantoti neorientēti grafi, orientēti grafi un grafi ar svariem.

1.4. GRAFU UZDOŠANA (GRAFA PIERAKSTA FORMAS)

Grafu var uzdot dažādos veidos. Zemāk ir aprakstīti trīs galvenie veidi.

1.4.1. BLAKUS VIRSOTŅU MATRICA

Blakus virsotņu matrica (skat. 1.9. att.) – tas ir divdimensiju masīvs ar izmēru NxN, to var arī saukt par tabulu. Tās kolonnas un rindas atbilst attiecīgām grafa virsotnēm. Katrā tabulas šūnā ir rakstīts viens skaitlis.

Piemēram, ja grafs nav svarots, tad tas ir "1", ja starp divām virsotnēm ir saite un "0", ja starp tiem saites nav (tabulas rindas numurs – pirmā virsotne, tabulas kolonnas numurs – otrā virsotne).

Grafiem ar svariem varētu būt citādi skaitļi, ne tikai viens un nulle, tomēr ideja ir tāda pati: ja uzdevumā ir minēts, ka grafa svari ir negatīvi, tad tabulas šūnā raksta to svaru, pretējā gadījumā, kad saites starp virsotnēm nav, parasti raksta "-1". Gadījumā, kad grafa svari ir jebkuri veseli skaitļi, tad izmantot "-1" nav apdomīgi. Šajā gadījumā šūnā parasti raksta kaut kādu ļoti lielu skaitli, kas ir lielāks par uzdevuma maksimāli pieļaujamo svara vērtību.

Viens no plusiem šim pierakstam ir vienkāršība. Grafa uzdošana ir ļoti viegla un saprotama, to var izmantot arī algoritmos.

Galvenais mīnuss šim pierakstam ir lielas atmiņas prasības.

1.4.2. ŠĶAUTŅU SARAKSTS

Šķautņu saraksts (skat. 1.10. att.) – izplatīts grafa uzdošanas veids daudzos uzdevumos. Uzdevumu risināšanai (no algoritmu viedokļa) glabāt grafu šajā veidā nav ļoti ērti. Taču cilvēkam uzdot grafu (piemēram, veidojot uzdevumu formulējumus un ievadot šķautņu sarakstu datnē) varētu būt pietiekami ērti. Šis grafu pieraksts algoritmos ar grafiem izmantojams reti. Tomēr izmantot tieši šo grafu pierakstu Belmana-Forda algoritmā ir ļoti izdevīgi un pamatoti.

Kā jau bija minēts, viens no plusiem šim pierakstam ir ērtība un saprotamība cilvēkam, bet no mīnusiem ir neērtības, izmantojot šo pierakstu algoritmos.

1.4.3. VIRSOTNES KAIMIŅU SARAKSTS

Virsotnes kaimiņu saraksti – ir viens no labākajiem grafa uzdošanas veidiem. Šo pierakstu ir ļoti ērti izmantot daudzos algoritmos, piemēram, Deikstras algoritmā. Šis grafu pieraksts dod iespēju strādāt algoritmam ātrāk un tērēt mazāk atmiņas grafa glabāšanai.

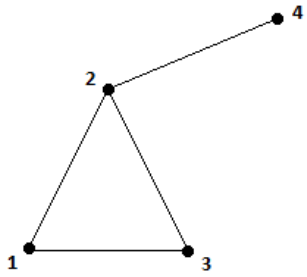
Tātad šim pierakstam plusi ir: ātrums un apsvērta atmiņas izmantošana grafa glabāšanai.

Protams, visus šo pierakstus var pārveidot no viena veida uz citu, bet tas aizņem kādu laiku.

Kopumā, vislabākais grafu pieraksts ir virsotnes kaimiņu saraksts, tāpēc uzdevumos, kur laika un atmiņas ierobežojumiem ir liela nozīme, ieteicams izmantot tieši šo grafa pieraksta veidu.

1.4.4. GRAFA PIERAKSTA FORMAS PIEMĒRS

Apskatīsim grafa piemēru (skat. 1.8. att.) un dažādas grafa pieraksta formas šim grafam.



1.8. att. **Grafa piemērs**

Blakus virsotņu matrica (skat. 1.9. att.) šajā piemērā tiek aizpildīta sekojošā veidā.

	1	***2***	***3***	***4***
1	0	1	1	0
2	1	0	1	1
3	1	1	0	0
4	0	1	0	0

1.9. att. **Blakus virsotņu matrica**

Vispirms nullējam visu tabulu, t.i., visas tabulas šūnās ierakstam nulli. Pēc tam apskatam visas virsotnes, sākot no pirmās, un strādājam pēc šāda algoritma.

Redzam, ka virsotnei ar numuru „1" ir divas šķautnes, kuras savieno pirmo virsotni ar divām citām virsotnēm (ar otro un ar trešo). Vispirms atzīmējam ceļu starp pirmo un otro virstoni. Tabulā (matricā) šūnā ar rindas numuru „1" (sākuma virsotne) un kolonnas numuru „2" (beigu virsotne) rakstām vieninieku (vieninieks nozīmē, ka ceļš starp virsotnēm pastāv, bet nulle – ka ceļa nav). Tālāk atzīmējam ceļu starp pirmo un trešo virsotni – ierakstām tabulas šūnā ar rindas numuru „1" un kolonnas numuru „3" vieninieku.

Pēc tām apskatām visas pārējās virsotnes un pēc tāda paša algoritma aizpildām tabulu.

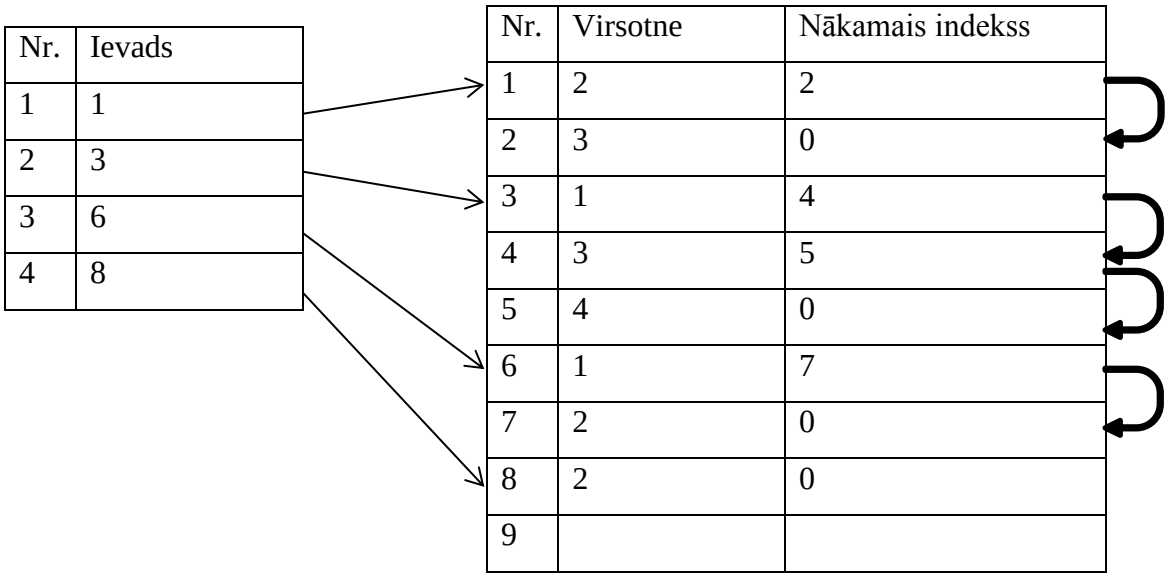
Redzam, ka blakus virsotņu matrica (skat. 1.9. att.) ir simetriska pret diagonāli. Tas ir tāpēc, ka šajā piemērā grafs ir neorientēts.

Šķautņu sarakstu (skat. 1.10. att.) veidot ir vēl vieglāk. Vajag tikai stabiņā rakstīt divas grafu virsotnes, kuras ir savienotas ar vienu šķautni. Var arī atzīmēt to, ka neorientētā grafā virsotņu pāri nav vajadzīgs dublēt (piemēram, ja kāda šķautne savieno pirmo un otro virsotni, tad sarakstā jāraksta tikai „1;2", ieraksts „2;1" nozīmē to pašu, tāpēc tas nav jāraksta).

1	2
1	3
2	3
2	4

1.10. att. **Šķautņu saraksts**

Lai glabātu grafu kā virsotnes kaimiņu sarakstu (skat. 1.11. att.), nepieciešami daži masīvi (parasti trīs, bet atkarībā no uzdevuma, masīvu skaits varētu būt arī lielāks). Šajā piemērā ir pietiek ar trīs masīviem. Pirmā masīva nosaukums ir „Pirmais indekss", otrā masīva – „Virsotne" un trešā – „Nākamais indekss". Masīvā „Pirmais indekss", kura izmērs ir N (virsotņu skaits), atbilstošajā elementā ir pierakstīts indekss, kur sākas kaimiņu aprakstīšana masīvā „Virsotne". Piemēram, ja masīvā „Virsotne" otras virsotnes kaimiņu aprakstīšana sākas no trešā elementa, tad masīva „Ievads" otrajā elementā jāraksta indekss trīs. Tātad katrā masīva „Ievads" elementā ir rakstīts masīvam „Virsotne" atbilstošs indekss. Masīvs „Nākamais indekss" satur indeksu uz nākamo kaimiņu vai nulli, ja pašreizējais kaimiņš bija pēdējais. Grafu pieraksts virsotnes kaimiņu saraksta veidā dod iespēju ātri pievienot un dzēst virsotnes un šķautnes no grafa.



1.11. att. **Virsotnes kaimiņu saraksts**

Kopumā grafa pieraksts virsotnes kaimiņu saraksta veidā ir optimāls uzdevumu risināšanai, jo salīdzinājumā ar divām pārējiem pieraksta veidiem, šis pieraksts dod iespēju strādāt ātrāk un izmantot mazāk atmiņas.

1.5. GRAFU CAURSKATS

Par grafa caurskati sauc sistemātisku tā virsotņu un šķautņu aplūkošanu. Parasti uzmanību vērš uz šķautnēm, jo tad virsotnes aplūkojas automātiski.
Grafu teorijā ir 2 vissvarīgākās grafu caurskates metodes. Tās ir: [3, 4]

1. Metode plašumā (angļu valodā – BFS) – metode darbojas, secīgi apskatot atsevišķus grafa līmeņus, sākot no pirmās virsotnes. Šī metode izmanto rindu.
2. Metode dziļumā (angļu valodā – DFS) – algoritma darbības laikā grafa virsotne mēģina iet uz kaimiņu virsotnēm – ja tā vēl nav apstaigāta, tad uz to pāriet un atkārto procesu. Šī metode izmanto steku.

Daudzi grafu algoritmi balstās uz šīm metodēm. Lielākajā daļā uzdevumu tiek izmantota metode plašumā.

1.5.1. CAURSKATE PLAŠUMĀ (BFS)

Caurskate plašumā – ir viens no pamatalgoritmiem, kurš ir pamatā daudzos citos grafu algoritmos. Piemēram, Deikstras algoritms un Primas algoritms ir caurskates plašumā vispārinājums.

Caurskate plašumā atrod ceļu ar visīsāko garumu nesvarotā grafā (t.i., ceļš, kurš satur minimālo šķeltņu skaitu).

Caurskati BFS var lietot grafa sakarības noteikšanai, jo beigās ir viegli pārliecināties, vai katra virsotne tikusi kādā slānī. Var arī pa ceļam saskaitīt aplūkotās virsotnes un tad šo skaitu salīdzināt ar N (kopējais virsotņu skaits). Nekādas grūtības nesagādā arī sakarīgo komponenšu noteikšana.

Algoritma sarežģītība ir O(n+m), kur n – ir virsotņu skaits, m – šķeltņu skaits.

Algoritms saņem no ievada grafu bez svariem un sākuma virsotni. Grafs varētu būt gan orientēts, gan neorientēts, algoritms strādā pareizi abos gadījumos.

Algoritmu var saprast kā grafa „aizdedzināšanas" procesu: vispirms „aizdedzinās" sākuma virsotni, pēc tam katrā nākamajā solī „uguns" no katras „degošas" virsotnes pārlēks uz visiem kaimiņiem. Tātad katrā solī uguns riņķis palielinās plašumā par vienu vienību.

Detalizēti algoritms strādā šādā veidā (pseidokodu sk. 12. pielikumā):

1. Vispirms ir jāizveido rinda, tās nosaukums būs „Degošās virsotnes”. Šajā rindā glabāsies degošās virsotnes numuri. Būs nepieciešams vēl viens masīvs. Tā nosaukums būs „Izmantotās virsotnes”. Šajā masīvā glabāsies informācija par to, vai kāda virsotne ir ugunī, vai ne (bija apmeklēta virsotne vai nebija).
2. Sākumā rindā „Degošās virsotnes” tika ievietota sākuma virsotne. Masīva „Izmantotās virsotnes” elementam ar sākuma virsotnes numuru ir piešķir zīme (vērtība ‘true’), pārējiem masīva elementiem tādas zīmes nav (tāpēc tiem tiek piešķirta vērtība ‘false’).
3. No rindas tekošā elementa tiek paņemts virsotnes numurs. No tās tiek apskatīti visi virsotnes kaimiņi un, ja kādi no tiem vēl nav degoši, tad tie tiek aizdedzināti un tiek ievietoti rindas „Degošās virsotnes” beigās. Masīva „Izmantotās virsotnes” elementiem, kuri tiek aizdedzināti, ir jāpiešķir zīme (vērtība ‘true’, lai pēc tam būtu zināms, ka šo virsotni nevajag pievienot rindai).
4. Tālāk algoritms strādā cikliski: kamēr rinda nav tukša, tiek atkārtots 3. punkts.
5. Beigās, kad rinda ir tukša, caurskate plašumā apstaigās visas sasniedzamās virsotnes no sākuma virsotnes, turklāt sasniedzot tās pa īsāko (pēc šķautnes skaita) ceļu.

1.5.2. CAURSKATE DZIĻUMĀ (DFS)

Caurskate dziļumā – ir otrā grafu apskatīšanās metode. Algoritmam ir nepieciešami steks un viens masīvs, kuru nosaukums būs „Izmantotās virsotnes”.

Algoritms darbojas pēc šādas idejas (pseidokodu sk. 13. pielikumā):

1. Paņemam sākuma virsotni un pievienojam to stekam. Par pašreizējo virsotni sauksim pēdējo virsotni stekā. Masīvā „Izmantotās virsotnes” elementā ar sākuma virsotnes indeksu tiek pievienota vērtība „1” (true).
2. Notiek tekošās virsotnes kaimiņu apskatīšana. Ja kāda no tām vēl nav apstaigāta (masīva „Izmantotās virsotnes” elementā ar kaimiņu virsotnes indeksu vērtība ir vienāda ar nulli), tad šīs kaimiņu virsotnes ir jāpievieno stekam un šī izmantošana tiek atzīmēta arī „Izmantotās virsotnes” masīvā: elementā ar izvēlēto kaimiņu virsotnes numuru tiek ielikta vērtība „1” (true).
3. Ja visi kaimiņi jau bija izmantoti, tad pašreizējais elements no steka tiek noņemts, un steka izmērs samazinās par vienu vienību.
4. Algoritms darbojas, kamēr steks nav tukšs.
5. Beigās, kad steks ir tukšs, caurskate dziļumā atradīs visas sasniedzamās virsotnes no dotās sākuma virsotnes. Tiem elementiem, kuri būs sasniedzami masīvā „Izmantotās virsotnes” atbilstošajā elementā, būs vērtība „1” (true).

1.6. VIĻŅU ALGORITMS (LI ALGORITMS)

Viļņu algoritms [5, 6] – ir viens no ceļu meklēšanas algoritmiem. Ar šo algoritmu var atrast īsāko ceļu planārā grafā. Viļņu algoritms ir viens no tiem algoritmiem, kura pamatā ir caurskate plašumā.

Parasti šis algoritms tiek izmantots shēmas plates trasēšana. Vēl viens algoritma pielietošanas piemērs ir īsākā ceļa meklēšana datorspēlēs – īpaši stratēģiskās spēles. Protams, viļņu algoritmu var arī izmanot uzdevumos, kur vajag atrast īsāko ceļu labirintā starp divām virsotnēm (vai pateikt, ka ceļa nav).

Algoritms strādā šādā veidā:

1. Uzdevumu risināšanai būs nepieciešama rinda un divi masīvi. Pirmais – ar nosaukumu „Matrica”. Masīvā ir informācija par katru šūnu: vai tā ir sākuma virsotne, beigu virsotne, vai tas ir šķērslis, vai tajā nekas nav (tā ir brīva). Otrā masīva nosaukums ir „Ceļa garums”. Masīvā šūnā atrodas viens skaitlis – ceļa garums no sākuma virsotnes līdz pašreizējai. Protams, masīva elementā ar sākuma virsotnes numuru pierakstītais cipars ir nulle. Visiem pārējiem elementiem tiek piešķirta vērtība „-1” (šūna vēl nav apmeklēta).
2. Rindā tiek pievienota sākuma virsotne.
3. Tālāk notiek rindas pašreizējās virsotnes apskats. Visi kaimiņi, kuri vēl nebija sasniedzami (līdz šim brīdim), tiek pievienoti rindā, un masīvā „Ceļa garums” atbilstošajā šūnā tiek ierakstīta vērtība, kura ir vienāda ar ceļa garumu līdz pašreizējajai virsotnei plus viens.
4. Algoritms darbojas, kamēr kāda no virsotnēm, kas tiek pievienota rindā, nebūs vajadzīga (t.i., beigu virsotne), vai, kad rinda būs tukša (visas sasniedzamās šūnas jau būs apskatītas).
5. Gadījumā, ja īsākais ceļš ir atrasts, un uzdevumā ir prasīts atjaunot ceļu – atzīmēt šūnas, pa kurām iet ceļš, tad to var izdarīt šādā veidā: jāsāk iet no beigu virsotnes. Tagad tekošā virsotne ir beigu virsotne. Tad notiek kaimiņu apskatīšana no tekošās virsotnes. Ir jāizvēlas kaimiņš ar ceļa garumu, kurš ir vienāds ar tekošās virsotnes ceļa garumu mīnus viens. Virsotne, kas tiek izvēlēta, kļūst par pašreizējo virsotni. Protams, visas virsotnes, kuras bija tekošās ir jāsaglabā palīgmasīvā „Ceļš”.
6. Īsākā ceļa garums no divām dotajām virsotnēm ir atrasts – rezultāts ir masīva „Ceļa garums” šūnā ar beigu virsotnes indeksu. Īsākais ceļš ir saglabāts palīgmasīvā „Ceļš”.

Ir divi veidi, kā apskatīt kaimiņus: par kaimiņšūnām var izvēlēties tikai četras šūnas (tikai tās, kurām ar pašreizējo šūnu ir kopīga skaldne) vai astoņas šūnas (ari tās šūnas, kas atrodas pa diagonāli no tekošās šūnas).

Viens no algoritma plusiem ir garantēta īsākā ceļa atrašana jebkurā labirintā (protams, ja ceļš eksistē). Galvenais mīnus šim algoritmam ir tas, ka ir nepieciešams pietiekami liels atmiņas daudzums un tas, ka lielos piemēros algoritma izpildīšanas laiks var būt ļoti ilgs.

2. PRAKTISKĀ DAĻA

Darba gaitā tiek izstrādāta lietotne „LabHinter”, kas ietver sevī oriģinālo viļņu algoritmu un trīs tā modifikācijas.

2.1. LIETOTNES APRAKSTS

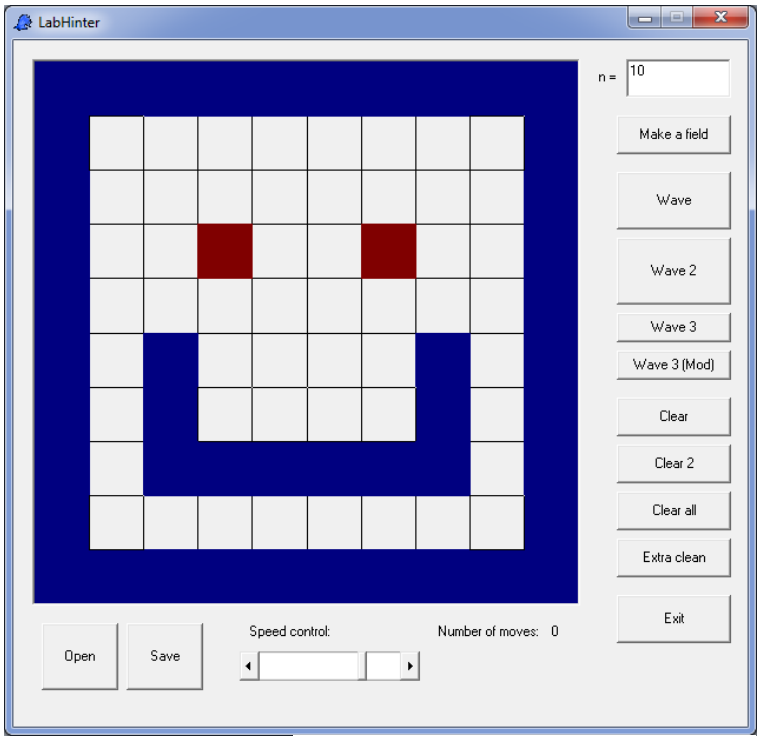
Izmantota valoda: Visual Basic 6.0.

Grafu uzdošanas veids: blakus virsotņu matrica.

Attēls: skat. 2.1. att.

Lietotnes apraksts:

Lietotnes labajā stūrī laukā pie kuram ir ierakstīts „n = ” jāievada naturālais skaitlis (maksimālais skaitlis ir 100). Pēc tam, nospiežot pogu „Make a field”, tiek izveidots NxN rūtains laukums, apkārt laukumam tiek izbūvēts žogs.



2.1. att. **Lietotne „LabHinter”**

Ar peles palīdzību var viegli konstruēt dažādus labirintus. Lai uzbūvētu sienu (šķērsli), jānoklikšķina ar peles kreiso pogu noteiktā šūnā. Programmā sienas (un šķērsli) ir parādītas zilā krāsā. Lai atzīmētu sākumpunktu un galapunktu, jānoklikšķina ar peles labo pogu. Sākuma un beigu punkti ir attēloti sarkanā krāsā.

Kad abi punkti tiks izvēlēti, var palaist kādu no algoritmiem. Sākumā lietotnē bija realizēts tikai viens algoritms – viļņu algoritms, bet pēc tam tika realizētas vēl trīs tā modifikācijas. Katra no pogām, kurā nosaukumā ir vārds „Wave”, startē oriģinālo viļņu algoritmu vai tā modifikāciju. Oriģinālajam viļņu algoritmam atbilst poga „Wave”, 1.modifikācijai atbilst poga ar nosaukumu „Wave 2”, 2. modifikācijai – poga „Wave 3”, un pēdējai 3. modifikācijai atbilst poga „Wave 3 (Mod)”. Pēdējām divām modifikācijām ir tādi nosaukumi, tāpēc ka pēc uzbūves algoritmi ir ļoti līdzīgi.

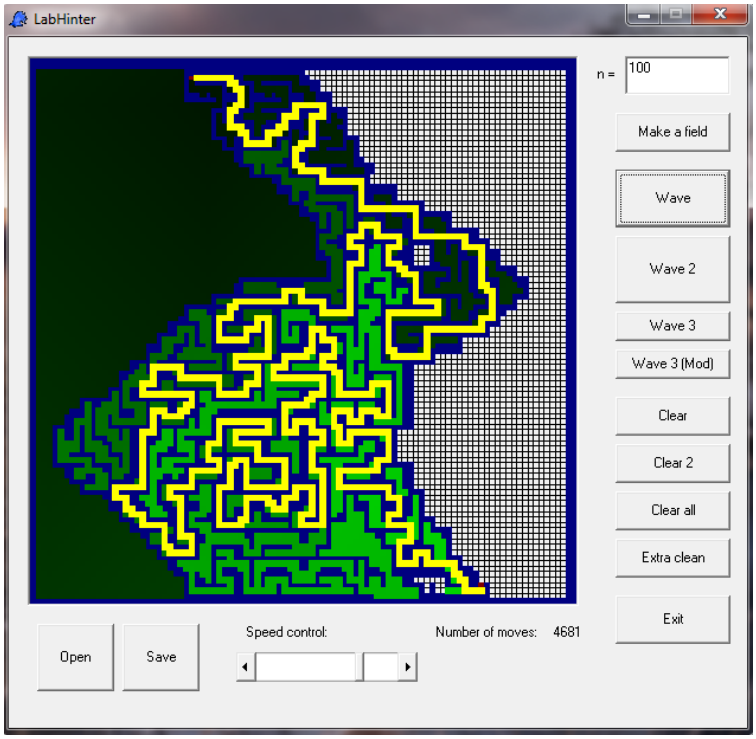
Pēc algoritma darbības laukā „Number of moves” tiek parādīts skaitlis – cik daudz šūnu tikušas apmeklētas.

Lietotnei ir četras speciālas pogas, lai attīrītu laukumu. Tas ir izdarīts, lai būtu ērtāk strādāt ar programmu, un vieglāk veidot dažādus testa piemērus un testēt tos. Katrai pogai ir atšķirīga nozīme. Poga „Clear” attīra laukumu tikai no algoritma darbības. Poga „Clear 2” attīra laukumu no algoritma darbības un sākuma un beigu punktus. Poga „Clear all” attīra visu laukumu, atstājot tikai žogu lauka perimetrā. Poga „Extra clean” attīra visu laukumu, neko neatstājot.

Lietotnē ir realizēta iespēja saglabāt uzbūvētu labirintu datnē, ka arī iespēja to atvērt un ielādēt. Šīs funkcijas izpilda pogas „Save” un „Open”. Programmas apakšdaļā atrodas ātruma regulētājs („Speed control”). Ar to palīdzību var uzstādīt piemērotu ātrumu.

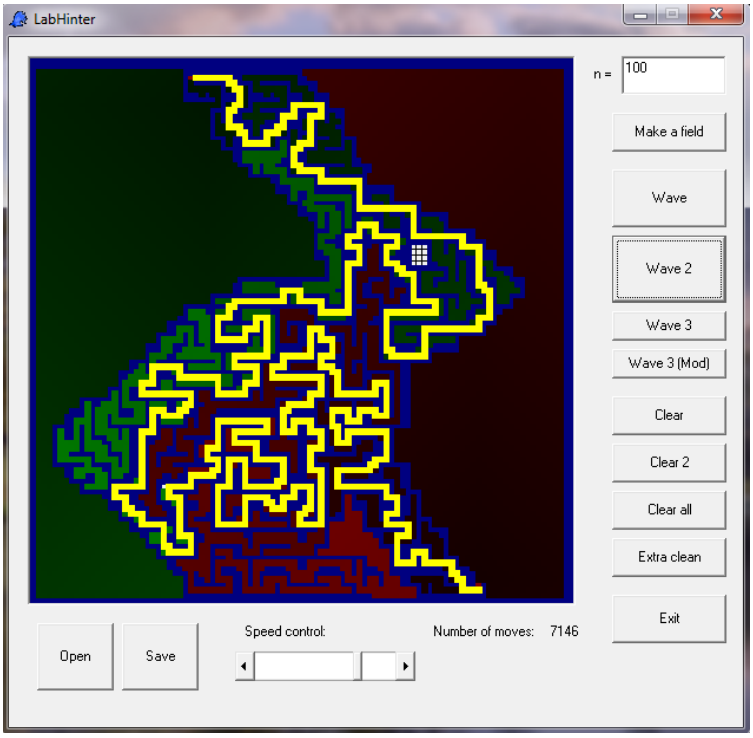
2.2. VIĻŅU ALGORITMA MODIFIKĀCIJAS

Tika realizēts pats viļņu algoritms un trīs tā modifikācijas. Oriģinālā algoritma viļņa izplatīšanu skat. 2.2. att.



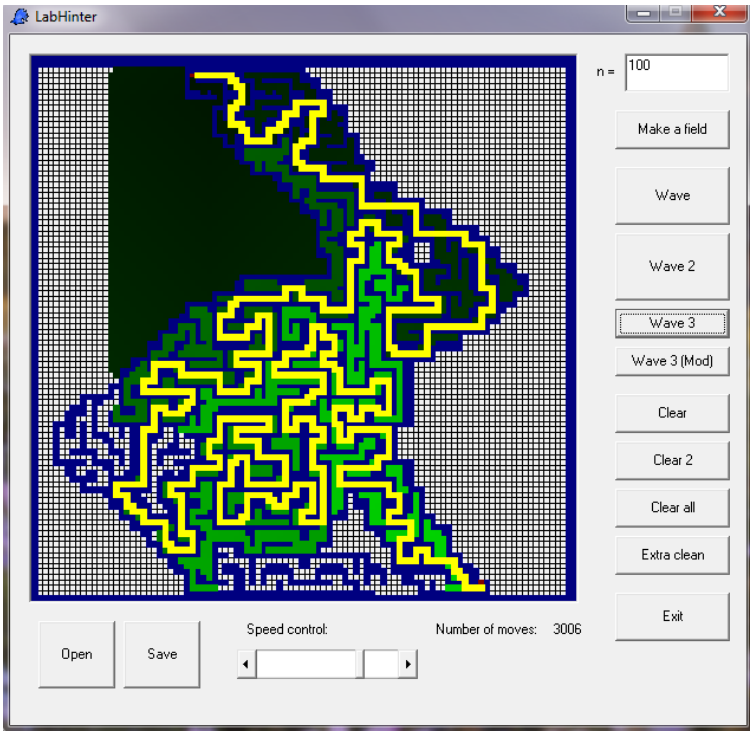
2.2. att. **Oriģinālais viļņu algoritms**

1. modifikācija (sk. 2.3. att.): viļņi no abām virsotnēm. Algoritms strādā tādā pašā veidā, kā oriģinālais viļņu algoritms, tikai ar vienu atšķirību: viļņu izplatīšana notiek vienlaicīgi no sākuma punkta un no beigu punkta. Kad abi viļņi satiekas, algoritms apstājas.



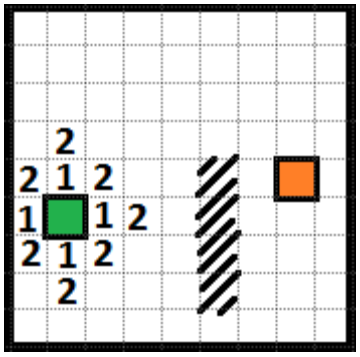
2.3. att. **1. modifikācija: divi viļņi**

2. modifikācija (sk. 2.4. att.): ierobežojuma funkcija. Tāda paša realizācija, kā parastā viļņu algoritmā, tikai atšķirība ir tāda, ka vilnis pēc iespējas neapmeklē tādas šūnas, kas atrodas ārpus divu (sākuma un beigu) virsotņu laukuma.

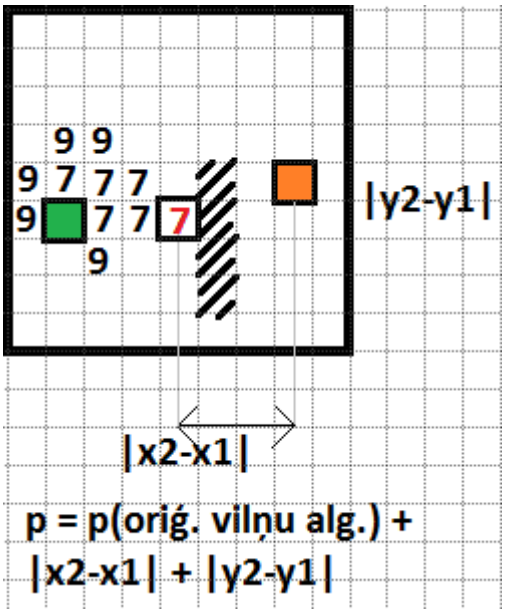


2.4. att. **2. modifikācija: ierobežojuma funkcija**

3. modifikācija (sk. 2.7. att.): priekšnoteikšanas funkcija (sk. 11. pielikumu, 14. pielikumu). Katrai šūnai tiek pievienots prognozējamais attālums no tās līdz beigu punktam. P = P(oriģ.viļņu alg.) + |x1-x2| + |y1-y2|, (x1,y1) – pašreizējās šūnas koordinātes, (x2,y2) – otras (beigu) virsotnes koordinātes.



2.5. att. **Oriģinālā algoritma viļņa izplatīšana**

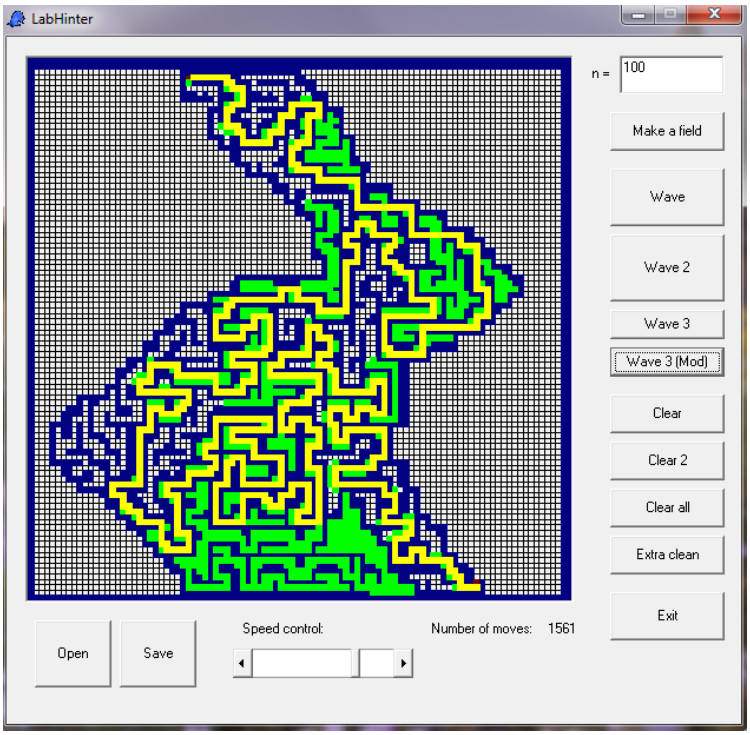


2.6. att. **3. modifikācijas viļņa izplatīšana**

Atšķirības starp oriģinālo un 3. modifikācijas viļņu izplatīšanos ir parādītas 2.5. attēlā un 2.6. attēlā.

Varam redzēt, ka oriģinālajā viļņu algoritmā tiek apskatītas vispirms tādas šūnas, kurās ir cipars 1, pēc tam tādas šūnas, kurās ir cipars 2, utt. 3. modifikācijā vilnis izplatās citādāk. Vispirms algoritms aprēķina prognozējamo attālumu un pēc tam apskata šūnas ar vismazāko prognozējamo vērtību. 2.6. attēlā algoritms vispirms apskatīs visas šūnas, kurās ir cipars 7, pēc tam šūnas ar ciparu 9, un tā tālāk, kamēr ceļš nav atrasts vai visas šūnas ir apmeklētas.

Rezultāts pēc 3. modifikācijas darbības ir parādīts 2.7. attēlā.



2.7. att. **3. modifikācija: priekšnoteikšanas funkcija**

Ir redzams, ka šūnas, kas atrodas pa kreisi no sākumpunkta, netika apmeklētas. Kamēr ceļš ir, algoritms mēģina tuvoties beigu punktam. Salīdzinot ar oriģinālo viļņu algoritmu (sk. 2.2. att.), 3. modifikācija apmeklēja 3 reizes mazāku šūnu skaitu. Lielos labirintos tas ir nozīmīgs ieguvums.

2.3. ALGORITMU SALĪDZINĀJUMS

Tika izstrādāta testu sistēma visām četrām algoritma realizācijām (sk. pielikumus 1-10). Testu sistēma satur gan vienkāršus labirintus, gan vidējas grūtības labirintus, gan arī sarežģītus labirintus.

Testa sarežģītību nosaka šķēršļu skaits: jo tas ir lielāks, jo sarežģīts ir labirints.

Visu četru algoritmu izpilde dod šādus rezultātus (skat. 2.1. tabulu).

2.1. tabula

Testēšanas rezultāti – gājienu skaits

Testa Nr.	Oriģinālais viļņu algoritms	Modifikācija Nr.1	Modifikācija Nr.2	Modifikācija Nr.3
1.	7056	4608	96	96
2.	9601	9312	9601	193
3.	5213	5192	1398	1180
4.	8025	8806	3362	1581
5.	7553	6740	2998	2003
6.	8877	8902	2720	2720
7.	9412	9440	4883	4880
8.	3086	2560	3086	914
9.	4024	3390	3957	2812
10.	2168	2148	1997	1560

Tabula atspoguļo apmeklēto šūnu skaitu, kas ir atkarīgs no algoritma realizācijas tipa un testa numura. Jo mazāks ir šis skaitlis, jo ātrāks un efektīvāks ir algoritms.

Oriģinālā viļņu algoritma un tā modifikāciju salīdzinājums ir apkopots 2.2. tabulā. Gājienu skaits oriģinālajam viļņu algoritmam tiek izdalīts ar katras modifikācijas vērtību.

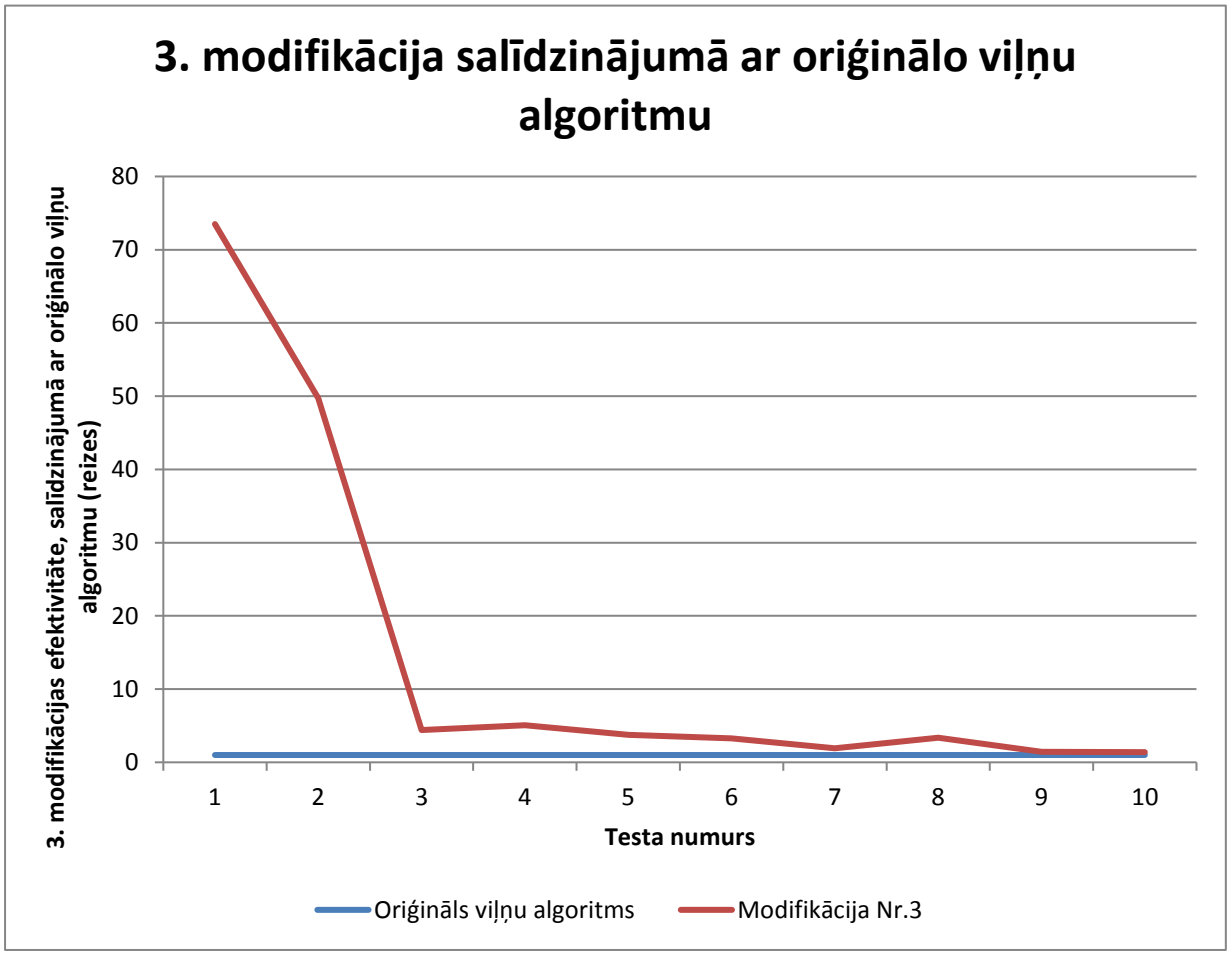
2.2. tabula

Oriģinālais viļņu algoritms pret modifikācijām (reizēs)

Testa Nr.	Modifikācija Nr.1	Modifikācija Nr.2	Modifikācija Nr.3
1.	1,53	73,50	73,50
2.	1,03	1,00	49,75
3.	1,00	3,73	4,42
4.	0,91	2,39	5,08
5.	1,12	2,52	3,77
6.	1,00	3,26	3,26
7.	1,00	1,93	1,93
8.	1,21	1,00	3,38
9.	1,19	1,02	1,43
10.	1,01	1,09	1,39

Tabulas dati atspoguļo, cik reizēs mazāk katra no modifikācijām apmeklēja šūnas nekā oriģinālais viļņu algoritms.

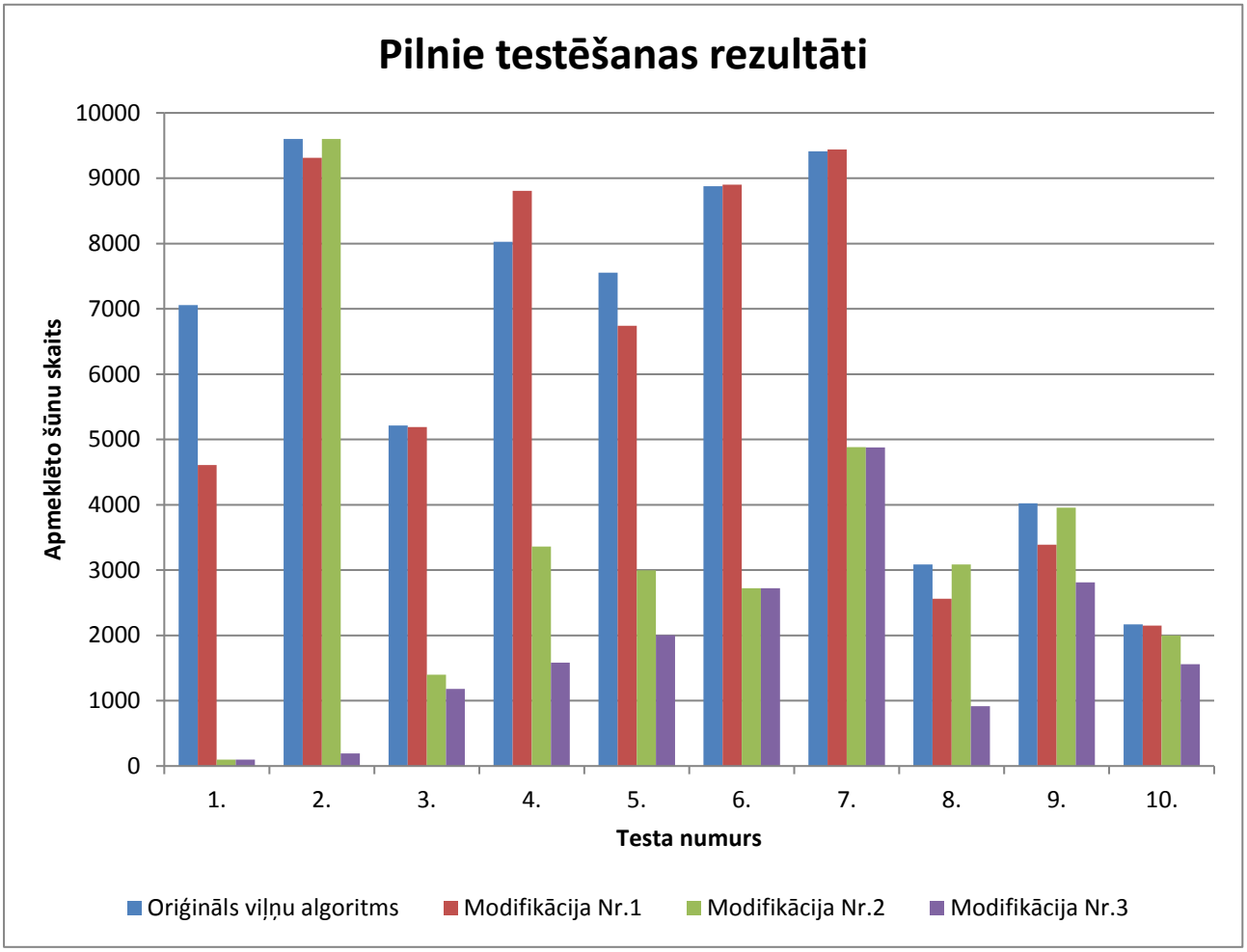
No dotās tabulas izriet tas, ka 3. modifikācija ir visefektīvākā. Salīdzinot ar oriģinālo viļņu algoritmu, 3. modifikācija apmeklēja 1,4 – 73,5 reizēs mazāku šūnu skaitu. Šo faktu var uzskatāmi redzēt diagrammā (skat. 2.8. att.).



2.8. att. **3. modifikācija salīdzinājumā ar oriģinālo viļņu algoritmu**

Diagramma parāda, cik reizēs, atkarīgi no testa numura, 3. modifikācija ir efektīvāka, salīdzinot ar oriģinālo viļņu algoritmu. Pirmajos testos, kur šķēršļu skaits nav liels, 3. modifikācijai ir redzams pārākums.

Detalizētus rezultātus par gājienu skaitu visos testos un visiem četriem algoritmiem var redzēt diagrammā (skat. 2.9. att.):



2.9. att. **Pilnie testēšanas rezultāti**

Diagrammā ir redzami četru algoritmu testēšanas rezultāti. Skaitli no 1 līdz 10 – ir testa numuri (pašus testus var redzēt pielikumos 1-10). Vislēnākais algoritms no četriem ir oriģinālais viļņu algoritms (diagrammā tas ir attēlots zilā krasā), bet visātrākais no četriem ir viļņu algoritma 3. modifikācija (diagrammā – violetā krāsā). Labus rezultātus sasniedza arī 2. modifikācija, tomēr dažos testos (piemēram, otrajā, astotajā un devītajā) tā zaudē ne tikai 3. modifikācijai, bet arī 1. modifikācijai. Kopumā par 1. modifikāciju var pateikt to, ka tās rezultāti daudzos testos ir ļoti līdzīgi oriģinālajam viļņu algoritmam.

Tātad no diagrammas var izdarīt secinājumu, ka 3. modifikācija ir visefektīvākā, jo visos testos tā, salīdzinot ar pārējām, apskatīja vismazāko šūnu skaitu, lai atrastu ceļu starp diviem punktiem labirintā.

Tāpēc lielos uzdevumos un labirintos ir vērts izmantot tieši 3. modifikāciju.

REZULTĀTI

Darba gaitā bija sasniegti sekojoši rezultāti:

- Tika izveidota lietotne „LabHinter” ar kuras palīdzību ir iespējams atrast ceļu labirintā;
- Tika realizēts viļņu algoritms un trīs tā modifikācijas;
- Tika izveidota testu sistēma, kas sastāv no 10 labirintiem;
- Tika veikta viļņu algoritma un tā modifikāciju testēšana;
- Tika izanalizēti un salīdzināti viļņu algoritma un tā modifikāciju testēšanas rezultāti.

Tātad visi darba izvirzītie mērķi tiek sasniegti, un visi uzdevumi šajā darbā ir izpildīti.

SECINĀJUMI

Analizējot iegūtos rezultātus, secina:

- Lietotne „LabHinter” dod iespēju veidot dažāda izmēra un sarežģītības labirintus, ka arī atrast ceļu tajos;
- Lietotne dod priekšstatu par to, kā darbojas viļņu algoritms;
- Lietotnē ir vizualizēta viļņu izplatīšana, ar iespēju regulēt animācijas ātrumu;
- Salīdzinot modifikācijas, atklājas, ka 3. modifikācija ir visefektīvākā.

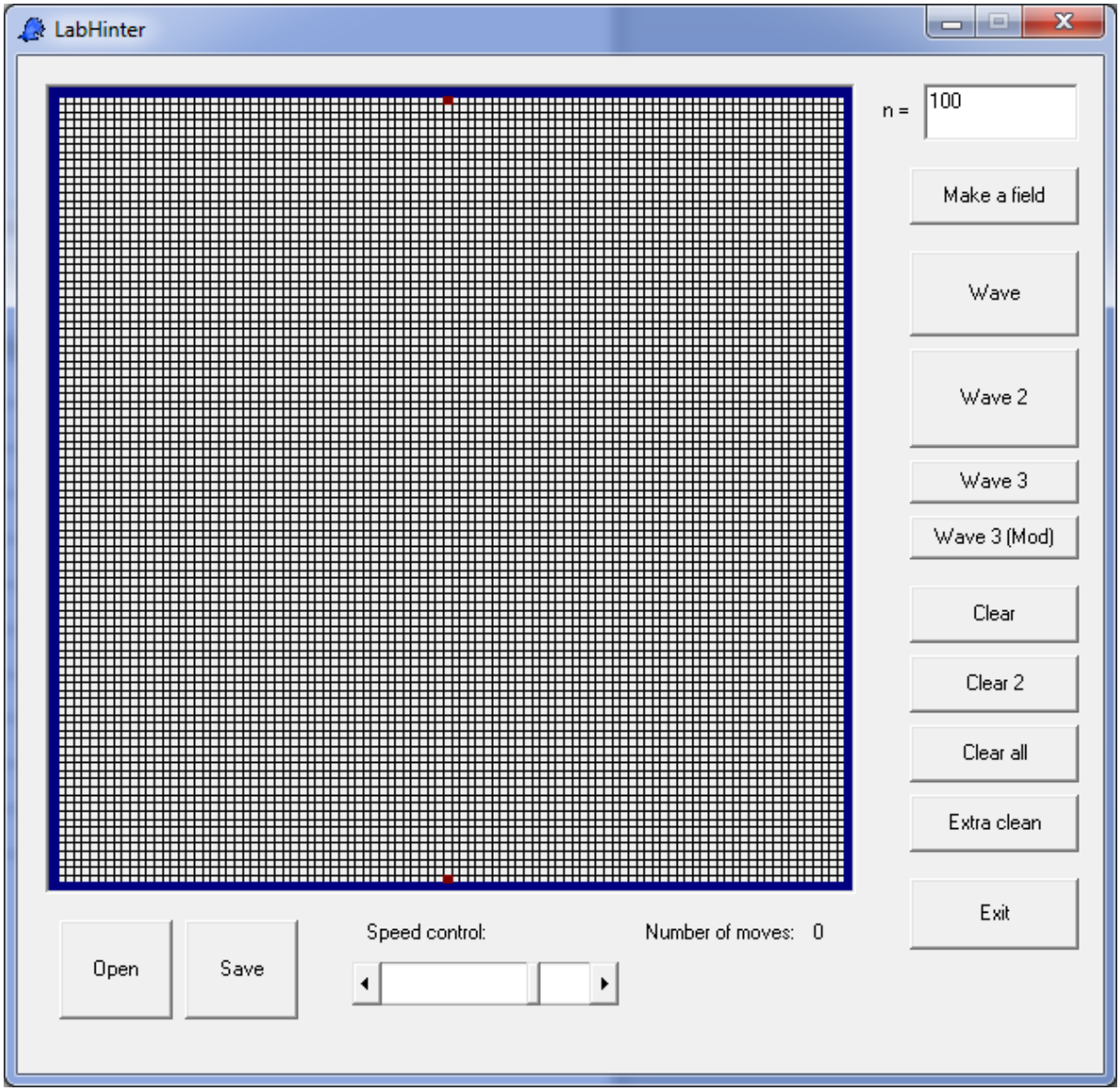
Šo darbu var izmantot kā vizuālo materiālu studenti un vidusskolēni, kas vēlas iepazīties ar viļņu algoritmu un grafu teorijas pamatiem.

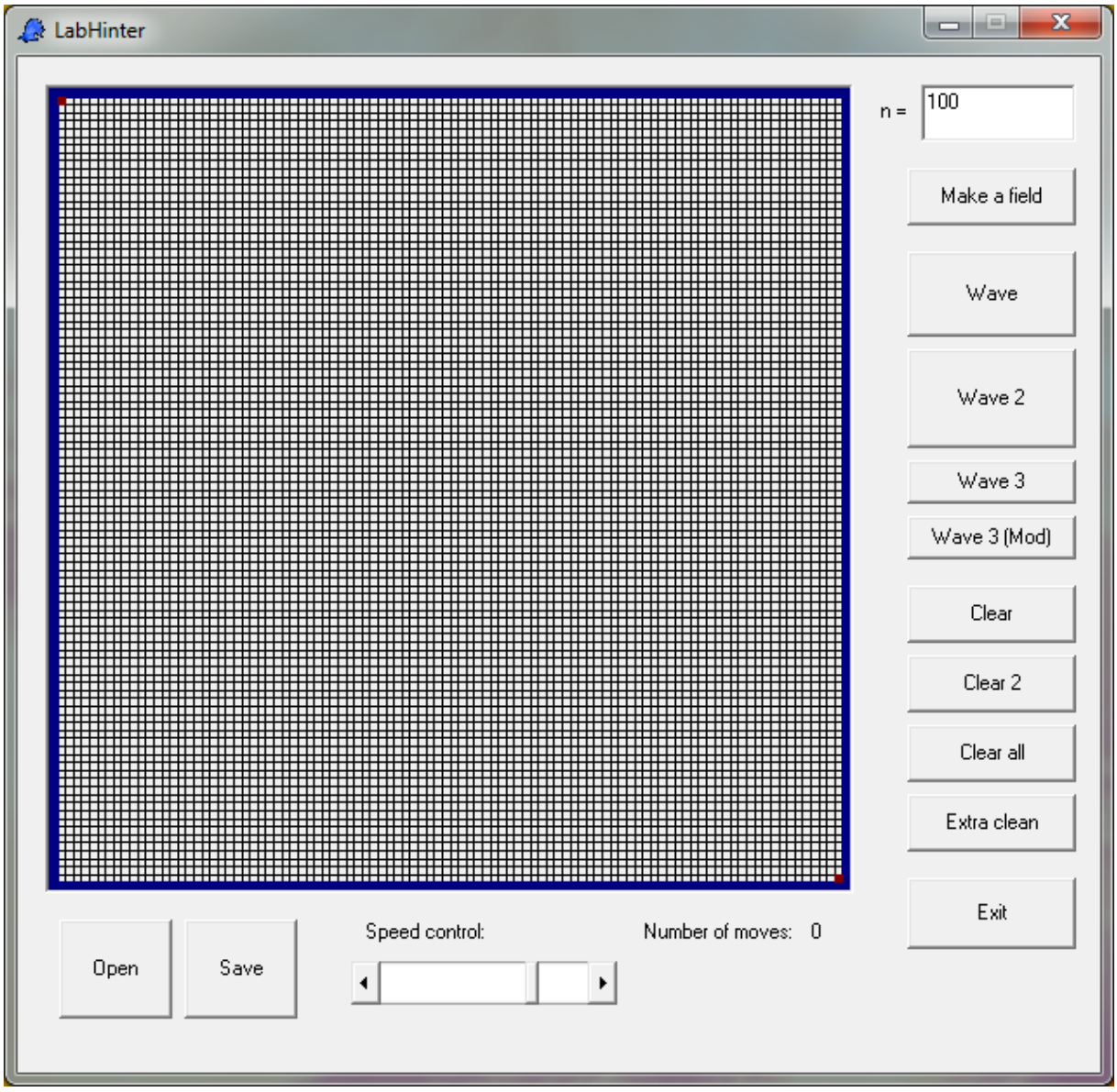
IZMANTOTIE INFORMĀCIJAS AVOTI

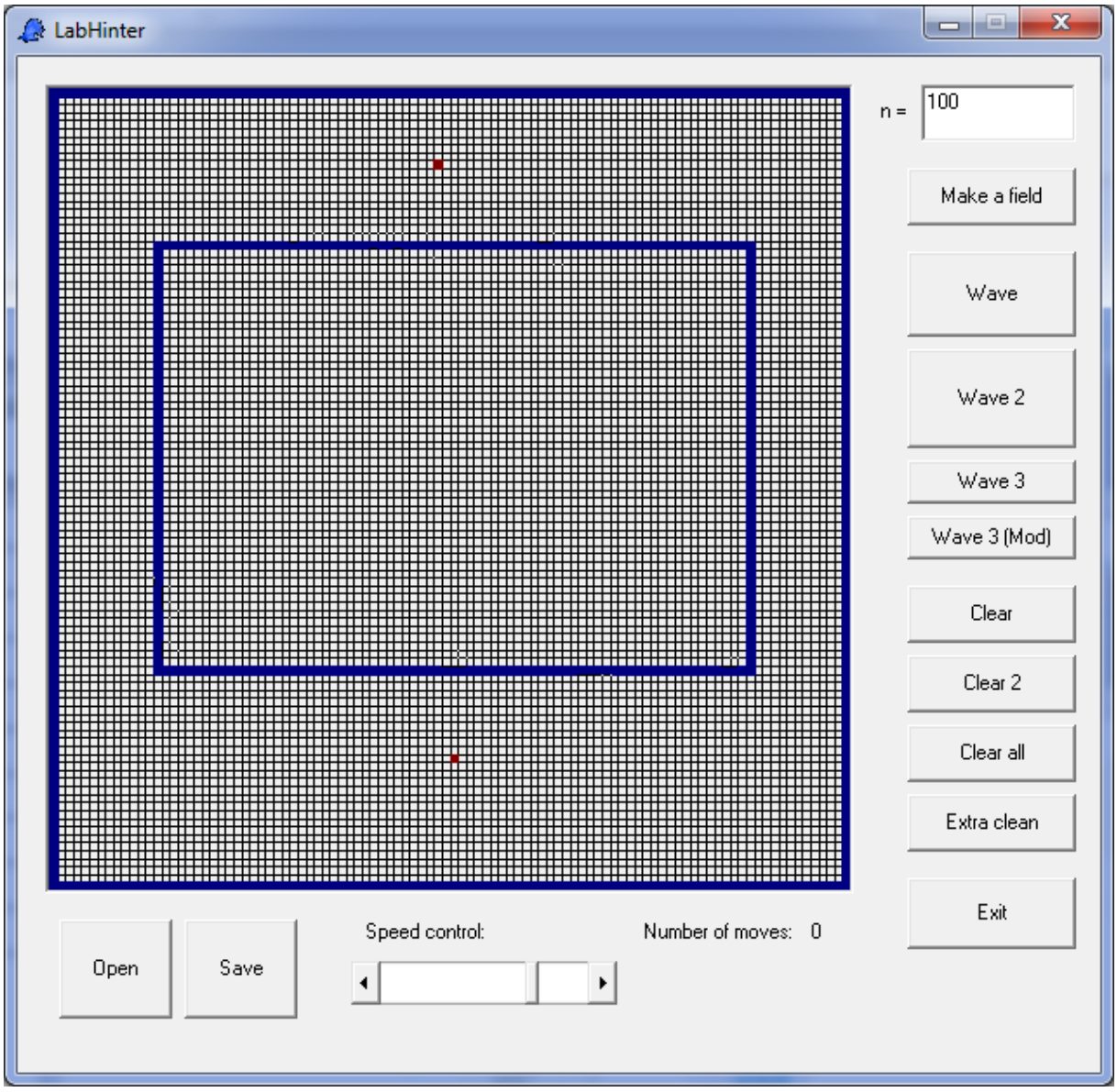
1. Bang-Jensen, J., Gutin, G. Digraphs: Theory, Algorithms and Applications (2nd ed.). USA: Springer, 2009. 798 pages. ISBN 978-1-84800-998-1.
2. Bondy, A., Murty, U.S.R. Graph Theory. USA: Springer, 2008. 654 pages. ISBN 978-1-84628-969-9
3. Introduction to Algorithms (2nd ed.). Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. USA: MIT Press and McGraw-Hill, 2001. 1184 pages. ISBN 0-262-03293-7.
4. Ruohonen, K. Graph Theory. Tampere University of Technology, 2008. 114 pages.
5. C. Y. Lee, "An algorithm for path connection and its application," IRE Trans. on Electronic Computer, vol. EC-10, 1961.
6. Maze Routing. [tiešsaiste]. [Skatīts 26.05.2019]. Pieejams: http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf

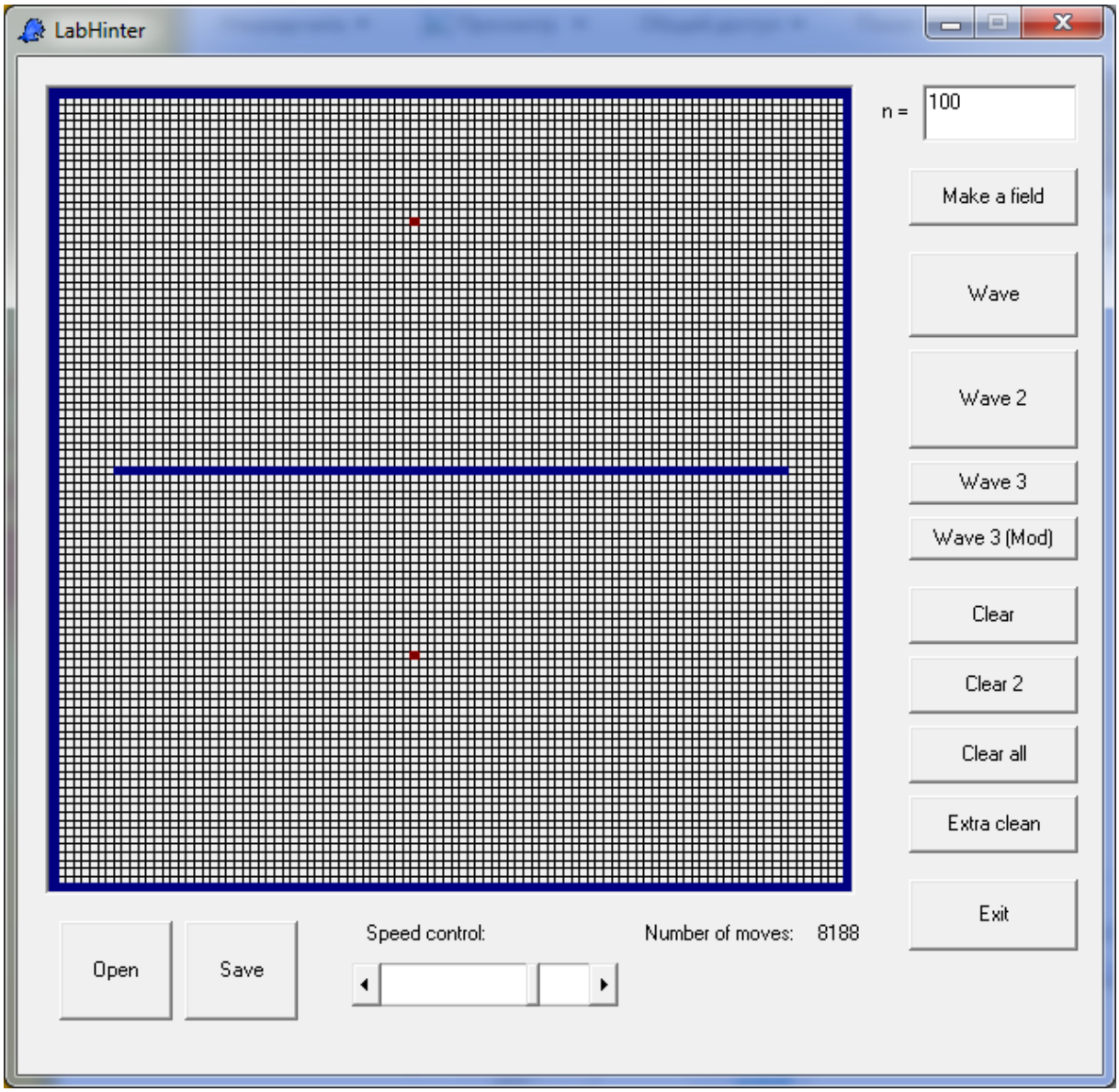
PIELIKUMI

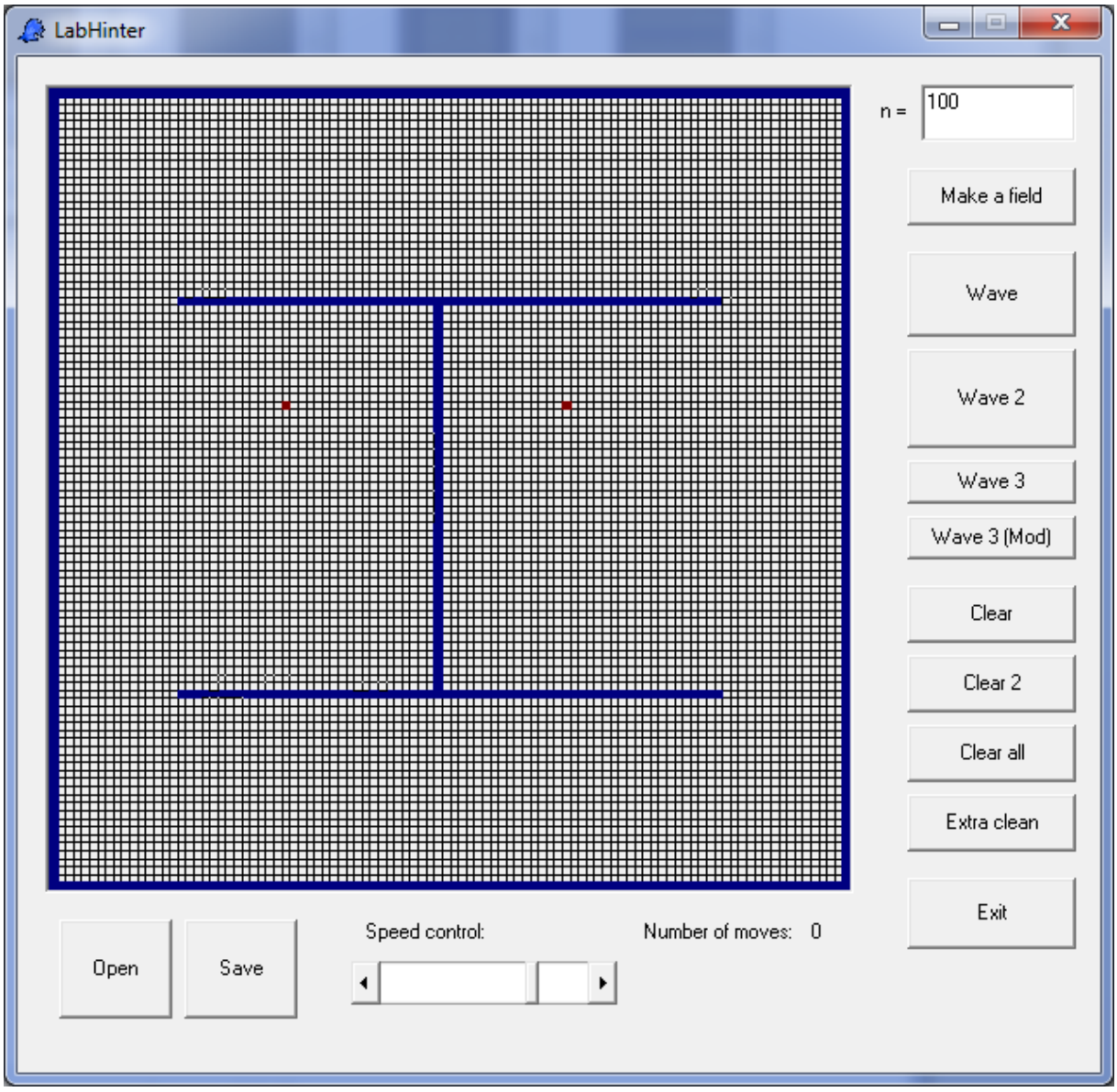
1. pielikums. Testpiemērs Nr. 1

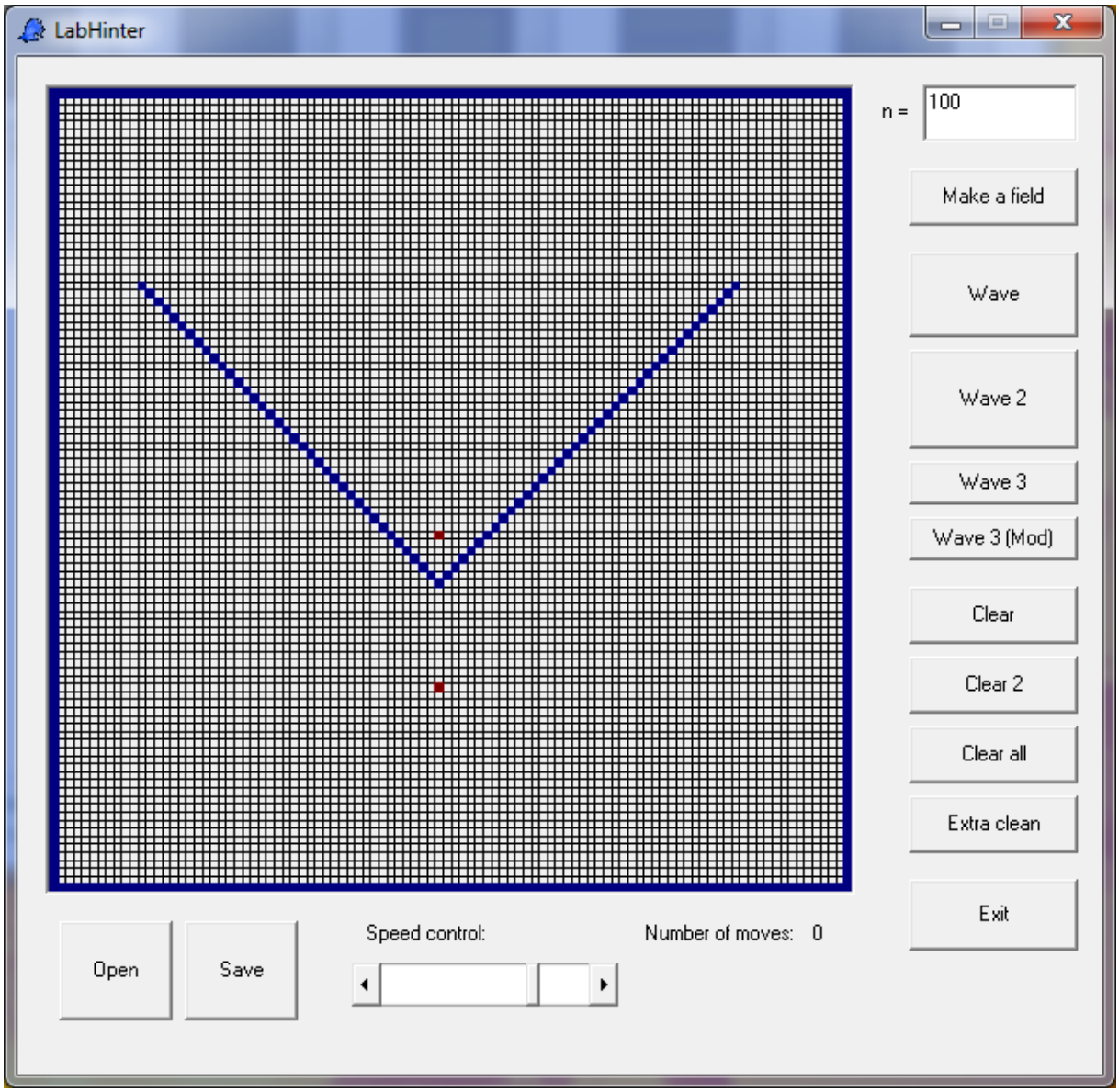


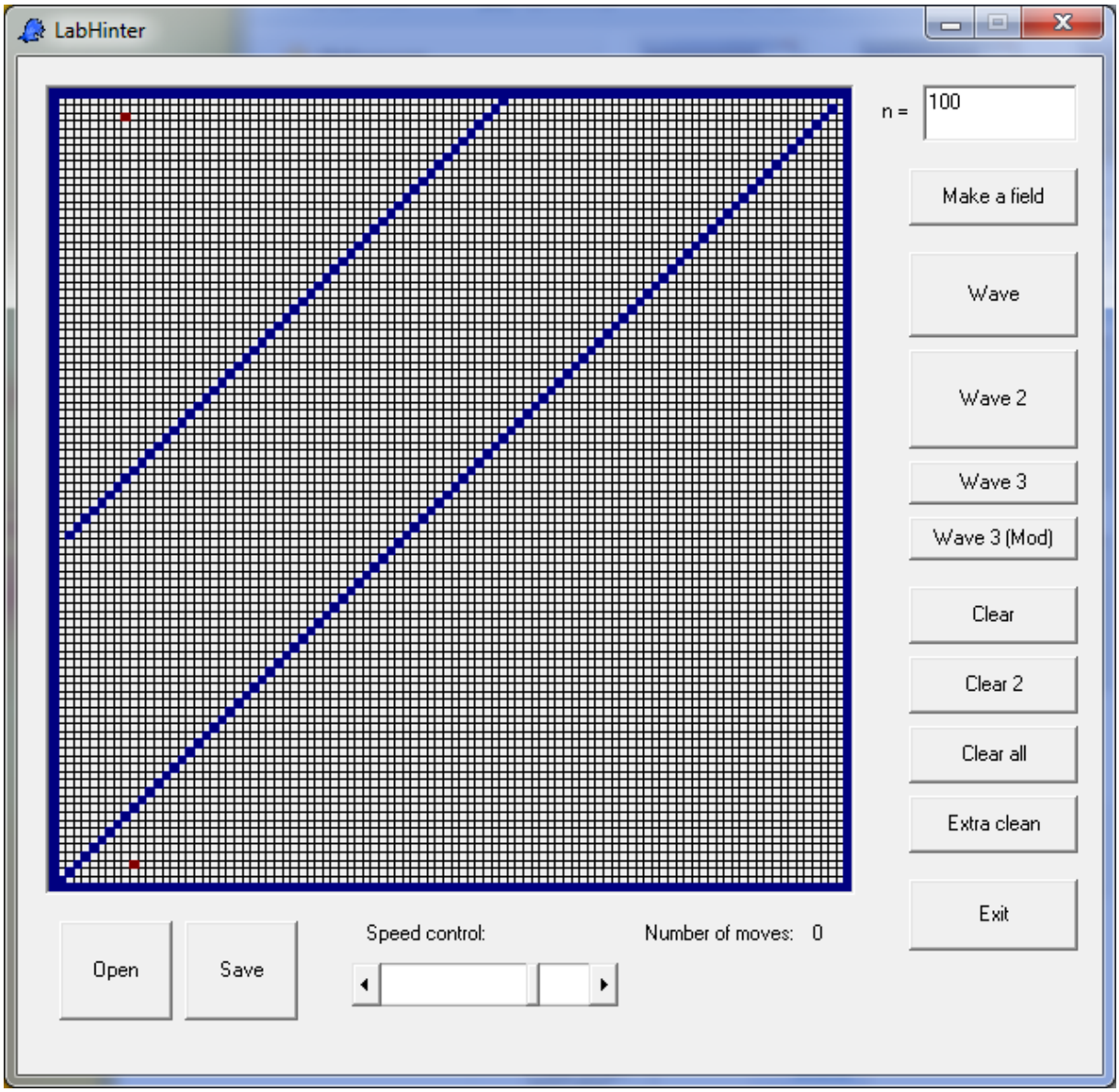

LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves: 0

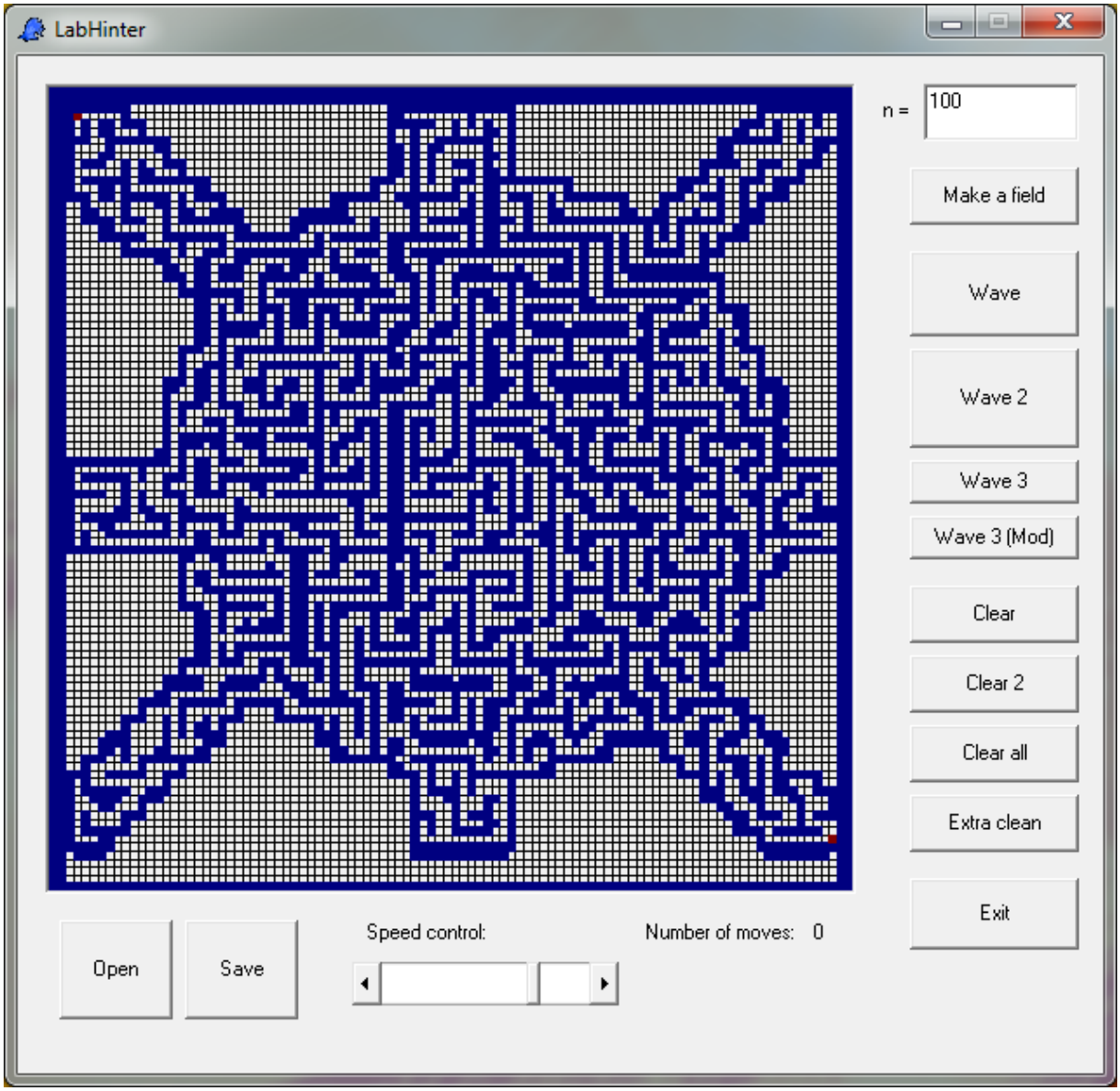

LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves: 0

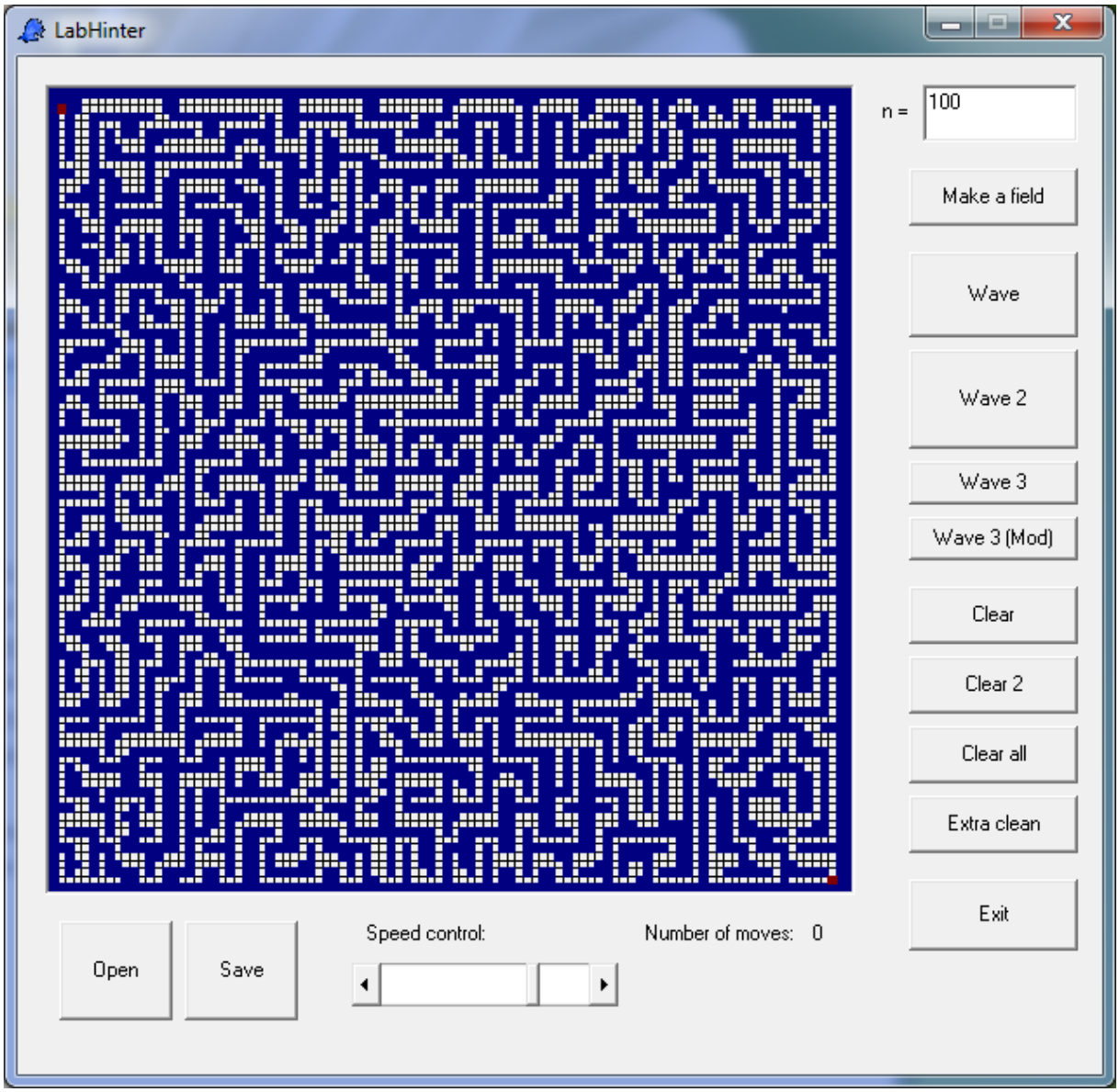

LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves:
8188

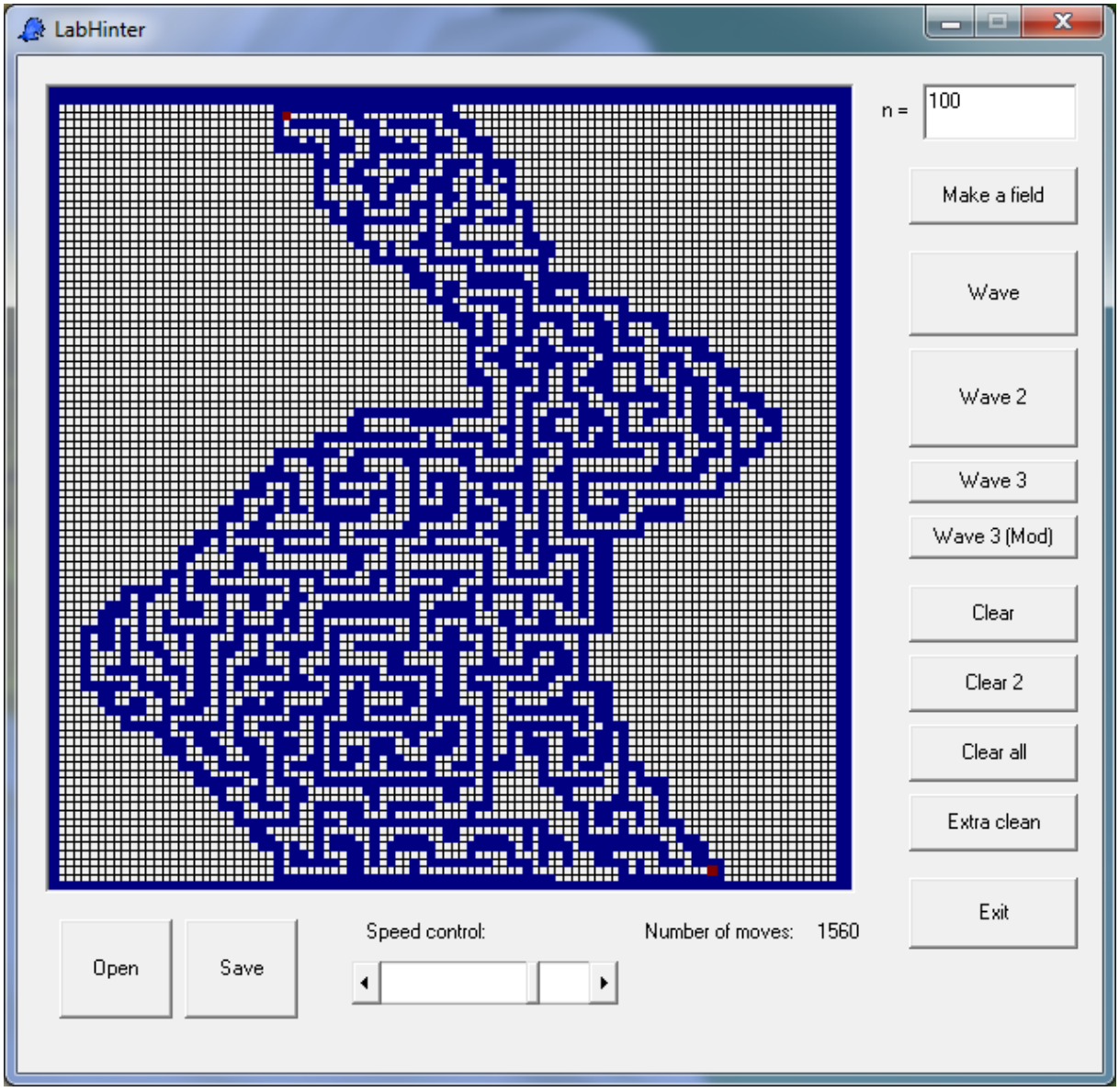

LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves: 0


LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves: 0


LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves: 0


LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves: 0


LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves: 0


LabHinter
n =
100
Make a field
Wave
Wave 2
Wave 3
Wave 3 (Mod)
Clear
Clear 2
Clear all
Extra clean
Exit
Open
Save
Speed control:
Number of moves: 1560

```
Picture1.Cls
operacija (n)
temper = zachistka(n, 0)
noError = True

errorChecker (2)
If noError = True Then
exist = False

For i = 1 To 10100
 vhod(i) = 0
 For k = 1 To 6
  rinda(i, k) = 0
 Next
Next

hodov = 0
Label4.Caption = "0"

rinda(1, 1) = x1
rinda(1, 2) = y1
rinda(1, 3) = 0
rinda(1, 4) = 0

rstart = 1
rfin = 2
kolvo = 1

dlina = Abs(x1 - x2) + Abs(y1 - y2)
vhod(dlina) = 1
rinda(1, 5) = dlina

Do While rstart <> rfin

 For i = 1 To 4
  X = rinda(rstart, 1) + z(i, 1)
  Y = rinda(rstart, 2) + z(i, 2)
  dlina = Abs(x2 - X) + Abs(y2 - Y)
  If (X = x2 And Y = y2) Then
   exist = True
   previndex = rstart
   Exit Do
  End If

  If a(X, Y) = 0 Then
   rinda(rfin, 1) = X
   rinda(rfin, 2) = Y
   rinda(rfin, 3) = rinda(rstart, 3) + 1
   rinda(rfin, 5) = dlina
   rinda(rfin, 6) = rstart

   temper = add(dlina, rfin)

   a(X, Y) = rinda(rstart, 3) + 1
   Picture1.Line (X, Y)-(X + 1, Y + 1), QBColor(10), BF
   rfin = rfin + 1

  End If

 Next
```

```
 prev = rstart

 vhod(rinda(rstart, 5)) = rinda(rstart, 4)

 temprez = -1
 temper = findnext()
 If temprez <> -1 Then rstart = temprez
 hodov = hodov + 1
 Label4.Caption = Str(hodov)
 If (rstart = prev) Or (temprez = -1) Then Exit Do

Loop

If exist = False Then
 temper = MsgBox("Path does not exist", vbCritical, "Path does not exist")
Else
 Do While (rinda(previndex, 1) <> x1 Or rinda(previndex, 2) <> y1) And previndex > 0
  Picture1.Line (rinda(previndex, 1), rinda(previndex, 2))-(rinda(previndex, 1) + 1,
rinda(previndex, 2) + 1), QBColor(14), BF
  previndex = rinda(previndex, 6)
 Loop
End If

End If
```

12. pielikums. Caurskate plašumā (pseidokods)

```
visited[start] = True
Q.push(start)
While Q Is Not empty
       current <-- Q.pop()
       foreach neighbor Of rinda[current]
            If visited[neighbor] = False
                          visited[neighbor] = True
                          Q.push(neighbor)
```

13. pielikums. Caurskate dziļumā (pseidokods)

```
visited[1..n] = False
dfs(current)
       visited[current] = True
       foreach neighbor, where neighbor Is Not visited
        And edge(current, neighbor) exists
                  dfs(neighbor)
```

14. pielikums. Viļņu algoritma 3. modifikācija (pseidokods)

```
rinda[i][1..6] = 0

length = Abs(x1 - x2) + Abs(y1 - y2)
rinda = (x1, y1, length)

While rinda Is Not empty
       current = min_length(rinda)

       foreach neighbor Of rinda[current]
              If (neighbor was visited Or neighbor Is a wall) continue
              (X, Y) = coordinates of neighbor
              If (X = x2 And Y = y2) Exit

              Add to rinda neighbor with length =  Abs(x2 - X) + Abs(y2 - Y)


       pop rinda[current]
```

Bakalaura darbs „***Ceļa atrašana labirintā***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: ***Ivans Gorbunovs*** **_____ ___.05.2019**.

Rekomendēju/nerekomendēju darbu aizstāvēšanai (nevajadzīgo izsvītrot)

Vadītājs: **profesors Dr. dat. Jānis Zuters _____ ___.05.2019**.

Recenzents: **M. dat. Ilvars Mizniks**

Darbs iesniegts Datorikas fakultātē 27.05.2019.

Dekāna pilnvarotā persona: **vecākā metodiķe Ārija Sproģe _____**

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

___.06.2019. prot. Nr. _____

Komisijas sekretārs(-e): _____