Шеста београдска гимназија

Милана Ракића 33

Београд

**Матурски рад из програмирања**

**ПРОГРАМСКА РЕШЕЊА ЗА ИЗЛАЗ ИЗ ЛАВИРИНТА**

Професор ментор: Ученик:

Мирјана Ћетојевић Милош Младеновић IV-10

Београд, мај 2023. године

САДРЖАЈ

**Страна**

[1. УВОД 3](#_Toc134021570)

[1.1. Развој решења за претрагу пута кроз лавиринт кроз историју 3](#_Toc134021571)

[2. РАЗЛИЧИТИ НАЧИНИ ПРОГРАМСКЕ РЕПРЕЗЕНТАЦИЈА ЛАВИРИТА 4](#_Toc134021572)

[2.1. Програмска репрезентација лавиринта примењена у симулатору 4](#_Toc134021573)

[2.1.1. Програмска репрезентација поља лавиринта 4](#_Toc134021574)

[2.1.2. Репрезентација лавиринта помоћу одговарајуће класе 7](#_Toc134021575)

[3. РАЗЛИЧИТИ ПРИСТУПИ У ПРОНАЛАЖЕЊУ ПУТАЊЕ КРОЗ ЛАВИРИНТ 11](#_Toc134021576)

[3.1. Алгоритам претраге у ширину (BFS) 11](#_Toc134021577)

[3.2. Дајкстрин алгоритам 12](#_Toc134021578)

[3.3. Похлепни алгоритам претраге по најбољем својству (Best First Search) 12](#_Toc134021579)

[3.4. А\* алгоритам 12](#_Toc134021580)

[4. НАСЛОВ ДРУГОГ ПОГЛАВЉА 13](#_Toc134021581)

[4.1. Први поднаслов другог поглавља 13](#_Toc134021582)

[4.2. Други поднаслов другог поглавља 13](#_Toc134021583)

[4.2.1. Наслов нижег реда 13](#_Toc134021584)

[4.2.2. Наслов нижег реда 13](#_Toc134021585)

[5. НАСЛОВ ТРЕЋЕГ ПОГЛАВЉА 14](#_Toc134021586)

[5.1. Поднаслов трећег поглавља 14](#_Toc134021587)

[Прилог / Прилози 16](#_Toc134021588)

[Прилог 1 16](#_Toc134021589)

[Садржај CD-а 17](#_Toc134021590)

[Литература 18](#_Toc134021591)

[БИОГРАФИЈА МАТУРАНТА 19](#_Toc134021592)

# УВОД

Лавиринти су конструкције које су кроз историју биле инспирација за многа уметничка дела и предмет проучавања у различитим научним дисциплинама. Лавиринт се појављује у разним културама и цивилизацијама кроз историју, као симбол загонетке, опасности и изазова. Један од најпознатијих примера је грчка митологија, према којој је Минотаур, полу бик полу човек, био затворен у лавиринту на Криту, а Тезеј је морао да пронађе пут кроз лавиринт и убије Минотаура како би ослободио Атињане данка. У античкој Грчкој лавиринти су били популарни као облик забаве и рекреације, а у средњем веку су често били укључени у утврђења и замкове као облик одбране. Данас се лавиринти често користе у архитектури и пејзажном дизајну као декоративни елементи или као средство за медитацију и релаксацију.

## Развој решења за претрагу пута кроз лавиринт кроз историју

Од старог века, па до данас, проналажење пута кроз лавиринт представљало је изазов вредан проучавања. Различите културе и цивилизације кроз историју су стварале различита решења за пролазак кроз лавиринте, од класичног физичког сналажења до разних симболичких и духовних приступа. Решење које је по митологији користио Тезеј је подразумевало провлачење конопца кроз лавиринт помоћу којег је знао којим је путем пролазио. Тај приступ се може упоредити са алгоритмом претраге у дубину, који је данас у широкој употреби. Још један приступ, који је посебно проучаван током 19. века је алгоритам праћења зида, према коме ако особа од улаза прати један (леви или десни) зид мора да дође до излаза. Џон Плеџ је проучавао овај алгоритам и унапређивао га за употребу у нетипичним лавиринтима.

Посебно интересовање за проблем проласка кроз лавиринт настало је у савремено доба развојем технологије. Развојем технологије, појавила су се различита програмска решења за овај проблем, далеко ефикаснија и поузданија од претходно поменутих. Решења проласка кроз лавиринт се данас примењују у различитим областима, од видео игара до роботских, навигационих и телекомуникационих система. Примена ових решења је поготово широка када узмемо у обзир да је лавиринт заправо врста графа, а алгоритам за пролазак кроз лавиринт алгоритам за претрагу графа.

Нека од данас најчешће коришћених алгоритама за решевање овог проблема ће бити објешњена у наставку овог рада. У циљу описа тих алгоритама, за овај рад је креирана Windows десктоп апликација помоћу које се сваки од алгоритама може симулирати на произвољном лавиринту.

# РАЗЛИЧИТИ НАЧИНИ ПРОГРАМСКЕ РЕПРЕЗЕНТАЦИЈА ЛАВИРИТА

Прва, и најинтуитивнија могућност програмске репрезентације лавиринта је репрезентација матрицом. Овај приступ подразумева матрицу у којој свако поље може бити у једном од два стања – проходно или непроходно. Овај приступ је одговарајући у већини ситуација, јер се алгоритми претраге на њему лако могу имплементирати. Међутим, мана овог приступа је алоцирање понекад сувишне меморије када говоримо о лавиринтима великих димензија. На споменутом симулатору креираном за потребе овог рада је коришћен овај приступ јер је симулатор намењен илустрацији алгоритма на лавиринту који уцртава корисник максималних димензија 50x50.

Као што је већ споменуто лавиринт се може посматрати као репрезентација неусмеронг графа код кога сваки је сваки чвор репрезентација једне позиције у лавиринту. Тај чвор може да има до 4 или 8 повезаних чворова, тј. суседних позиција, у зависности од тога да ли је дозвољено дијагонално кретање (са изузетком чворова ивичних позиција – они имају до 3 или 2 повезана чвора у ћошковима, а на страницама до 5 или 3). Овакав граф се може реализовати помоћу објеката класе чворова повезаних показивачима, као и матрицом суседности у којој члан (*i,j*) означава постојање или непостојање везе између чворова *i* и *j.* Овако репрезентовани графови су такође погодни за имплементацију алгоритама претраге.

## Програмска репрезентација лавиринта примењена у симулатору

У симулатору креираном за потребе овог рада, као што је наведено, за репрезентацију лавиринта се користи матрица у којој свако поље може бити проходно и непроходно. Поље такве матрице би у основном случају могло да се представи као једна логичка променљива са вредностима тачно или нетачно. Међутим како је у конкретном случају потребно пољима придружити више својстава, креирана је сложенија класа чија су поља објекти.

Такође, иако у основном случају лавиринт може бити стандардана дводимензионална матрица, ради лакшег управљања лавиринтом, извршавања претраге и омогућавања интеракције корисника са лавиринтом, креирана је класа Lavirint са свим додатно потребним атрибутима и методама.

### Програмска репрезентација поља лавиринта

public class Polje

{

public StanjePolja stanje;

public Point Pozicija;

public SolidBrush sb;

public int stranica;

public static int xstart, ystart;

public double G, H;

public Polje parent;

public Polje(int str, Point pozicija, int xstart, int ystart, StanjePolja stanje = StanjePolja.zid)

{

this.Pozicija = pozicija;

this.stanje = stanje;

this.stranica = str;

Polje.xstart = xstart;

Polje.ystart = ystart;

sb = new SolidBrush((stanje == StanjePolja.zid) ? Boje.boja\_Zid : Boje.boja\_Slobodno);

}

public void Crtaj(Graphics g)

{

g.FillRectangle(sb, xstart+Pozicija.X \* (stranica + 2), ystart+Pozicija.Y \* (stranica + 2), stranica, stranica);

if (stanje == StanjePolja.start)

using (Image startImage = Properties.Resources.start)

{

g.DrawImage(startImage, xstart + Pozicija.X \* (stranica + 2), ystart + Pozicija.Y \* (stranica + 2), stranica, stranica);

G = 0;

}

else if (stanje == StanjePolja.cilj)

using (Image finishImage = Properties.Resources.finish)

{

g.DrawImage(finishImage, xstart + Pozicija.X \* (stranica + 2), ystart + Pozicija.Y \* (stranica + 2), stranica, stranica);

}

}

public void Click(Graphics g, int StartOrCilj, bool sveZauzeto)

{

if (StartOrCilj == 0)

{

stanje = stanje == StanjePolja.slobodno ? StanjePolja.zid : StanjePolja.slobodno;

sb = new SolidBrush((stanje == StanjePolja.slobodno) ? Boje.boja\_Slobodno : Boje.boja\_Zid);

}

else

{

sb = new SolidBrush(sveZauzeto ? Boje.boja\_Zid : Boje.boja\_Slobodno);

if (StartOrCilj == 1)

{

((MainForm)Application.OpenForms[0]).lavirint.Start = Pozicija;

stanje = StanjePolja.start;

}

else if (StartOrCilj == 2)

{

((MainForm)Application.OpenForms[0]).lavirint.Finish = Pozicija;

stanje = StanjePolja.cilj;

}

}

Crtaj(g);

}

public void Clear(Graphics g, bool zid)

{

stanje = zid ? StanjePolja.zid : StanjePolja.slobodno;

sb = new SolidBrush(zid ? Boje.boja\_Zid : Boje.boja\_Slobodno);

Crtaj(g);

}

public void MarkOtvoren(Graphics g)

{

sb.Color = Boje.boja\_Otvoren;

Crtaj(g);

}

public void CrtajPutanju(Graphics g, int delta\_t)

{

sb.Color = Boje.boja\_Putanja;

this.Crtaj(g);

Thread.Sleep(delta\_t);

Console.Beep();

if (stanje != StanjePolja.start)

this.parent.CrtajPutanju(g, delta\_t);

}

public void Resize(int stranica, int xstart, int ystart)

{

this.stranica = stranica;

Polje.xstart = xstart;

Polje.ystart = ystart;

}

public void Restart(bool zauzeto, Graphics g)

{

sb.Color = zauzeto ? Boje.boja\_Zid : Boje.boja\_Slobodno;

stanje = zauzeto ? StanjePolja.zid : StanjePolja.slobodno;

Crtaj(g);

}

}

* 1. Атрибути који су намењени за тачно и јасно исцртавање поља у корисничком интерфејсу су:
     1. pozicija – позиција конкретног поља у лавиринту;
     2. xstart, ystart – координате горњег левог ћошка лавиринта, у односу на који се исцртавају сва поља. Ови атрибути су статички јер су у овој имплементацији заједнички свим објектима класе поља;
     3. stranica - дужина, тј. ширина једног поља (ова вредност се израчунава на основу броја поља и димензија корисничког прозора).
  2. Атрибути који ће се користити у алгоритмима претраге су:
     1. stanje – атрибут набројивог типа, који одређује у ком стању је поље тренутно. Набројиви тип овог атрибута је дефинисан на следећи начин:

public enum StanjePolja { zid, slobodno, start, cilj }

* + 1. G, H – атрибути реалног типа који се користе у Дајкстрином алгоритму, A\* алгоритму и претрази по најбољем својству и чине срж тих алгоритама. О њима ће бити више речи у наставку.
    2. parent – показивач на родитељско, тј. претходно поље у путањи према траженом пољу у лавиринту. Та путања је преко пказивача parent састављена као листа повезаних чворова, у којој је последњи чвор тражено поље. Реконструкција пронађене путање ће се вршити помоћу ових повезаности.
  1. Методе класе поља:
     1. Crtaj – исцртава поље у корисничком интерфејсу на одговарајући начин, у зависности од стања, позиције и димензије поља
     2. Click – врши промену стања поља, а позива се када корисник мишем кликне на поље у циљу уцртавања произвољног лавиринта. На крају ова метода позива претходно наведену методу за цртање, како би измене на пољу биле приказане.
     3. Clear, Restart и Resize су још неке од метода које омогућавају интуитивну интеракцију косрисника са пољима (брисање садржаја, поновно постављање стања поља и промена величине поља)
     4. MarkOtvoren и CrtajPutanju су методе које се позивају кроз рад алгоритама претраге и служе за графички приказ поља проверених у току претраге, и за реконструкцију пронађене путање помоћу показивача са поља на претходно поље (parent) у путањи. Метода CrtajPutanju ту реконструкцију врши на рекурзиван начин.

### Репрезентација лавиринта помоћу одговарајуће класе

Следи код класе лавирината, у којој су изузети садржаји метода коришћених у алгоритмима претраге, јер ће те методе и алгоритми бити објешњене у наставку:

public class Lavirint

{

public Polje[,] polja;

public int stranicaPolja;

public Tuple<int,int> Dimenzije; //U poljima

public int xstart = 0, ystart = 0;

public Point Start;

public Point Finish;

public bool sveZazeto\_onStart = true;

int simulator\_delta\_t = 50;

public Lavirint(int FormWidth, int FormHeight, Tuple<int,int> dimenzije)

{

Dimenzije = dimenzije;

stranicaPolja = FormWidth / dimenzije.Item1 - 2;

if(stranicaPolja\*dimenzije.Item2 > (FormHeight - dimenzije.Item2 \* 2))

stranicaPolja = FormHeight / dimenzije.Item2 - 2;

if(dimenzije.Item1 \* (stranicaPolja+2) < FormWidth)

xstart = (FormWidth - dimenzije.Item1 \* (stranicaPolja+2)) / 2;

if (dimenzije.Item2 \* (stranicaPolja + 2) < FormHeight)

ystart = (FormHeight - dimenzije.Item2 \* (stranicaPolja+2)) / 2;

polja = new Polje[Dimenzije.Item2, Dimenzije.Item1];

Start = new Point(-1,-1);

Finish= new Point(-1,-1);

for (int i = 0; i < Dimenzije.Item2; i++)

{

for (int j = 0; j < Dimenzije.Item1; j++)

{

polja[i, j] = new Polje(stranicaPolja,new Point(j,i), xstart, ystart);

}

}

}

public void Resize(int FormWidth, int FormHeight)

{

stranicaPolja = FormWidth / Dimenzije.Item1 - 2;

if (stranicaPolja \* Dimenzije.Item2 > (FormHeight - Dimenzije.Item2 \* 2))

stranicaPolja = FormHeight / Dimenzije.Item2 - 2;

if (Dimenzije.Item1 \* (stranicaPolja + 2) < FormWidth)

xstart = (FormWidth - Dimenzije.Item1 \* (stranicaPolja + 2)) / 2;

if (Dimenzije.Item2 \* (stranicaPolja + 2) < FormHeight)

ystart = (FormHeight - Dimenzije.Item2 \* (stranicaPolja + 2)) / 2;

for (int i = 0; i < Dimenzije.Item2; i++)

for (int j = 0; j < Dimenzije.Item1; j++)

polja[i, j].Resize(stranicaPolja,xstart,ystart);

}

public void Crtaj(Graphics g)

{

for (int i = 0; i < Dimenzije.Item2; i++)

{

for (int j = 0; j < Dimenzije.Item1; j++)

{

polja[i, j].Crtaj(g);

}

}

}

public Point Polje\_by\_XY(int x, int y)

{

int i = (int)((x - xstart) / (stranicaPolja+2));

int j = (int)((y - ystart) / (stranicaPolja+2));

return new Point(i > (Dimenzije.Item1 - 1) ? (Dimenzije.Item1 - 1) : i, j > (Dimenzije.Item2 - 1) ? (Dimenzije.Item2 - 1) : j);

}

public void SetStartFinish(Graphics g, int x, int y, bool start)

{

if (start)

{

if(Start.X != -1)

polja[Start.Y, Start.X].Clear(g,sveZazeto\_onStart);

Start = new Point(x, y);

polja[y, x].Click(g, 1, false);

}

else

{

if(Finish.X != -1)

polja[Finish.Y, Finish.X].Clear(g, sveZazeto\_onStart);

Finish = new Point(x, y);

polja[y, x].Click(g, 2, false);

}

}

IEnumerable<Polje> SusednaPolja(Polje p, bool strane4)

{...}

public void AStar(bool strane4, Graphics g)

{...}

public Polje MinimumF(HashSet<Polje> nodes)

{...}

public Polje MinimumH(HashSet<Polje> nodes)

{...}

public Polje MinimumG(HashSet<Polje> nodes)

{...}

public void BestFS(bool strane4, Graphics g)

{...}

public void Dijkstra(bool strane4, Graphics g)

{...}

public void BFS(bool strane4, Graphics g)

{...}

public void DFS(bool strane4, Graphics g)

{...}

public void Clear(bool svezauzeto, Graphics g)

{

Start = new Point(-1,-1);

Finish = new Point(-1,-1);

sveZazeto\_onStart = svezauzeto;

for (int i = 0; i < Dimenzije.Item2; i++)

for (int j = 0; j < Dimenzije.Item1; j++)

polja[i, j].Restart(svezauzeto,g);

}

public void RemovePaths(Graphics g) {

for (int i = 0; i < Dimenzije.Item2; i++)

for (int j = 0; j < Dimenzije.Item1; j++)

{

polja[i,j].sb.Color = (polja[i,j].stanje == StanjePolja.zid) ? Boje.boja\_Zid : Boje.boja\_Slobodno;

polja[i, j].Crtaj(g);

}

}

}

}

* 1. Атрибути који чине класу Lavirint су:
     1. polja – Матрица објеката класе Polje, која је основа у овој репрезентацији лавиринта;
     2. stranicaPolja, xstart, ystart – Дужина странице поља и координате горње леве тачке лавиринта, то су атрибути намењени за прецизно цртање лавиринта на корисничком интерфејсу;
     3. Start, Finish – показивачи на стартно поље у лавиринту и поље до кога треба пронаћи путању;
     4. sveZazeto\_onStart – логичка променљива која показује да ли је лавирнт био потпуно проходан, или потпуно непроходан пре него је корисник почео да уцртава лавиринт. Користи се у методама за приказ лавиринта,
     5. simulator\_delta\_t – целобројна вредност, која представља време у милисекундама које треба да протекне између две графичке измене у лавиринту приликом претраге, тј. успорава претрагу тако да корисник може да испрати догађеје у њој. Израчунава се на основу удаљености старта од циља;
  2. Методе и конструктори класе лавирината:
     1. Lavirint – конструктор класе којим се креира нови објекат на основу димензија лавирнта у пољима и димензија корисничког прозора у пикселима;
     2. Resize – метода која се позива приликом промене величине корисничког прозора и омогућава респонсивност апликације;
     3. Crtaj – метода која исцртава лавиринт на графички интерфејс на основу његових особина;
     4. Polje\_by\_XY – метода која враћа позцију поља у лавиринту на основу координата на екрану. Користи се при препознавању поља на које је кориник кликнуо мишем;
     5. Clear – метода која брише измене на лавиринту и омогућава кориснику да крене са уцртавањем лавиринта из почетка;
     6. RemovePaths – метода која уклања графички приказ пронађене путање и проверених поља у лавиринту, те приказује само проходност поља.

# РАЗЛИЧИТИ ПРИСТУПИ У ПРОНАЛАЖЕЊУ ПУТАЊЕ КРОЗ ЛАВИРИНТ

Од појаве рачунарства развила су се многа различита решења за претрагу пута кроз лавиринт. Модерна решења овог проблема су заправо имплементације алгоритама за претрагу графова јер су лавиринти, као што је наведено, заправо неусмерени графови. Алгоритами који се користе у ову сврху, а данас су у широкој употреби се могу поделити на информисане и неинформисане алгоритме. Неинформисани, или слепи, алгоритми претраге су они који у датом моменту немају слику о њиховом домену, па могу само да препознају да ли је одређено поље тражено или не, али не могу да доносе одлуке о томе које поље је боље обићи следеће у току претраге. За разлику од њих информисани алгоритми претраге имају слику о домену на коме раде и о траженом пољу. Зато је у информисаним алгоритмима могуће применити хеуристичке функције помоћу којих се доноси одлука о томе које од суседних поља треба обићи следеће у датом моменту. У наставку ће бити описани алгоритми оба типа, који су примењени у креираном симулатору.

## Алгоритам претраге у ширину (BFS)

Алгоритам претраге у ширину (енг. Breadth-first Search) је један од неинформисаних алгоритама претраге. Овај алгоритам започиње претрагу од стартног поља и у следећем кораку обилази сва суседна поља редом (тј. све чворове који су директно повезани са стартним чвором графа). Затим обилази сва поља следећег слоја (сва поља која од стартног поља дели слој претходно проверених поља). Тај поступак се понавља све док се не дође до траженог поља или до момента када више нема могућности за даљи обилазак. У овом алгоритму се уобичајено користи ред (енг. queue) за чување поља, тј. чворова, који су „наследници“ чворова претходног слоја и који се редом обилазе по принципу „први ушао први изашао“ (енг. First In First Out - FIFO). Приликом обиласка поља једног слоја у исти ред се додају (Enqueue) поља следећег слојa, тј. наследничка поља, а проверена поља претходног слоја се избацују (Dequeue).

Следи метода класе Lavirint која је раније изостављена у класи Lavirint, а која врши претрагу у ширину, кораци алгоритма су објашњени кроз код у коментарима:

/// <summary>

/// Algoritam pretrage u širinu

/// </summary>

/// <param name="strane4"> Logička promenljiva, pokazuje da li je dozvoljeno dijagonalno kretanje ili ne</param>

/// <param name="g"> Grafika korisničkog interfejsak po kojoj se iscrtava prolaz kroz lavirint </param>

public void BFS(bool strane4, Graphics g)

{

//Izračunava se vreme potrebno da bi se algoritam usporio radi bolje preglednosti, prema proporciji

//25 : P = X : 40, P-povrsina pravougaonika odredjena start i finish tackama - obrnuta proporcija dobijena testriranjem

simulator\_delta\_t = (int)(25 \* 40 / ((Math.Abs(Start.X - Finish.X)+1) \* (Math.Abs(Start.Y - Finish.Y)+1)));

Queue<Polje> Polja\_za\_proveru = new Queue<Polje>();

HashSet<Polje> poseceno = new HashSet<Polje>();

Polje startno\_polje = polja[Start.Y, Start.X];

//Startno polje se dodaje polje u red za proveru

Polja\_za\_proveru.Enqueue(startno\_polje);

//Iteracija se ponavlja sve dok ima mogućnosti za proveru

while (Polja\_za\_proveru.Count > 0)

{

//Trenutno polje se skida sa početka reda, to je polje koje je najduže bilo u redu (FIFO)

Polje trenutno = Polja\_za\_proveru.Dequeue();

//Ako je trenutno polje traženo, pretraga je uspešno završena, a rekosnstruisana putanja se iscrtava

if (trenutno.Pozicija.X == Finish.X && trenutno.Pozicija.Y == Finish.Y)

{

trenutno.CrtajPutanju(g, 2\*simulator\_delta\_t); //Grafički prikaz rekonstruisane putanje na lavirintu

return;

}

//Trenutno polje se dodaje u skup posećenih polja, da se ne bi ponavljalo

poseceno.Add(trenutno);

//Grafiički prikaz otvorenog i proverenog polja

trenutno.MarkOtvoren(g);

//Usporavanje algortma radi bolje preglednosti

Thread.Sleep(simulator\_delta\_t);

// Prolazak kroz sva susedna polja trenutnog polja, susedna polja se određuju i na osnovu toga da li je dozvoljeno dijagnalno kretanje

foreach (Polje sused in SusednaPolja(trenutno, strane4))

{

//Ako je već posećeno ili dodato u red, preskače se

if (poseceno.Contains(sused) || Polja\_za\_proveru.Contains(sused))

continue;

//Trenutno polje se postavlja kao prethodničko susednom polju

sused.parent = trenutno;

//Susedno polje se dodaje na kraj reda kako bi bilo provereno tokom dalje iteracije

Polja\_za\_proveru.Enqueue(sused);

}

}

}

## Алгоритам претраге у дубину (DFS)

Алгоритам претраге у дубину је још један од неинформисаних алгоритама претраге. Алгоритам претраге у дубину, као и већина других, претрагу започиње од стартног поља. Након провере стартног поља узима се једно, прво, суседно поље као почетно поље прве гране графа (обично се различите гране пролазе редом с лева на десно). Овим алгоритмом се пролази кроз сва поља једне гране од почетног поља (или поља где постоји гранање) до највеће могуће дубине, на којој се налази или мртви крај или тражено поље, а затим се враћа у назад до првог претходног гранања уколико тражено поље није пронађено (енг. backtracking) и тај поступак се понавља све док се не дође до траженог поља или до момента када су све могућности тј. гране проверене. Овај алгоритам за меморисање поља која треба обићи користи структуру стек, која функционише по принципу „последњи ушао први изашао“ (eng. LIFO). Редослед кoјим се поља налазе и додају у стек омогућава обилажење целе гране пре преласка на следећу. Алгоритам претраге у дубину се углавном користи ради провере да ли је пролаз кроз лавиринт (тј. кроз граф) уопште могућ, а не када је потребно пронаћи најкраћу путању, јер овај алгоритам не гарантује проналазак најкраћег пута, већ било ког.

У наставку следи метода за претрагу у дубину класе Lavirint, која је ранаије изостављена из кода те класе:

/// <summary>

/// Algoritam pretrage u dubinu - DFS

/// </summary>

/// <param name="strane4"> Logička promenljiva, pokazuje da li je dozvoljeno dijagonalno kretanje ili ne</param>

/// <param name="g"> Grafika korisničkog interfejsak po kojoj se iscrtava prolaz kroz lavirint </param>

public void DFS(bool strane4, Graphics g)

{

//Izračunava se vreme potrebno da bi se algoritam uspori radi bolje preglednosti, prema proporciji

simulator\_delta\_t = (int)(25 \* 40 / ((Math.Abs(Start.X - Finish.X)+1) \* (Math.Abs(Start.Y - Finish.Y)+1)));

Stack<Polje> Polja\_za\_proveru = new Stack<Polje>();

HashSet<Polje> poseceno = new HashSet<Polje>();

Polje startNode = polja[Start.Y, Start.X];

//Početno polje se dodaje u stek polja za proveru

Polja\_za\_proveru.Push(startNode);

//Iteracija se ponavlja sve dok ima mogućnosti za proveru

while (Polja\_za\_proveru.Count > 0)

{

//Za trenutno polje u ovom momentu se skida polje sa kraja steka

Polje trenutno = Polja\_za\_proveru.Pop();

//Ako je trenutno polje traženo, pretraga je uspešno završena, a rekosnstruisana putanja se iscrtava

if (trenutno.Pozicija.X == Finish.X && trenutno.Pozicija.Y == Finish.Y)

{

trenutno.CrtajPutanju(g, 2 \* simulator\_delta\_t);

return;

}

//trenutno polje se dodaje u skup posećenih polja, da se ne bi ponavljalo

poseceno.Add(trenutno);

//Grafički prikaz prolaza kroz polje

trenutno.MarkOtvoren(g);

//Usporavanje algoitma radi bolje preglednosti

Thread.Sleep(simulator\_delta\_t);

foreach (Polje sused in SusednaPolja(trenutno, strane4))

{

if (!poseceno.Contains(sused))

{

//Neposećeni sused se dodaje na kraj hijerarhije u dosadašnjoj putanji (pomoću pokazivača na prethodnika)

sused.parent = trenutno;

//Taj sused se dodaje na kraj steka, kako bi bio proveren

Polja\_za\_proveru.Push(sused);

}

}

}

}

## Дајкстрин алгоритам

Дајкстрин алгоритам је такође неинформисани алгоритам претраге. Овај алгоритам је осмислио холандски научник Едсгер Дајкстра 1956. године, а објавио га 1959. године у часопису Numerische Mathematik.

## Похлепни алгоритам претраге по најбољем својству (Best First Search)

## А\* алгоритам

# НАСЛОВ ДРУГОГ ПОГЛАВЉА

Текст

## Први поднаслов другог поглавља

Текст

## Други поднаслов другог поглавља

Текст

### Наслов нижег реда

Текст

### Наслов нижег реда

Текст

# НАСЛОВ ТРЕЋЕГ ПОГЛАВЉА

Текст

## Поднаслов трећег поглавља

Текст

Закључак

Текст...

# Прилог / Прилози

## Прилог 1

Овај део рада је предвиђен уколико има потребе да се уз рад приложе неке слике, скице, резултати истраживања и слично што није већ сасатвни део ранијих поглавља, него се из њих упућује на прилог. Уколико нема прилога, треба обрисати ову страницу.

# Садржај CD-а

Овај део рада се користи само ако се уз рад заиста прилаже CD, тада треба навести шта се налази на њему. У супротном, треба обрисати ову страницу.

# Литература

Пример навођења литературе (по азбучном реду):

1. Душанић, С. (2011). *Родне норме и ризична понашања младића у региону*. У  Д. Бранковић (ур.), часопис: *Култура и образовање – детерминанте друштвеног прогреса*, стр 263-285. Бањалука: Филозофски факултет.
2. Јанковић, Б. Милојевић, С. (2011).*Међународна полицијска сарадња у борби против насиља на фудбалским утакмицама*, Зборник радова, „Сузбијање криминала у оквиру међународне полицијске сарадње, Тара, стр. 149—161. - зборник радова
3. Станојчић, Ж., Поповић, Љ., (2010), *Граматика српског језика*, Завод за уџбенике, Београд. - књига

следи навођење литературе са интернта, на пример,

1. Word 2010 videos and tutorials, <https://support.office.com/en-us/article/Word-2010-videos-and-tutorials-cfa75118-e522-4ea5-963e-2b56d25fb9a5?ui=en-US&rs=en-US&ad=US> (јануар 2018)
2. Гимназија Јован Јовановић Змај Нови Сад, [http://jjzmaj.edu.rs](http://jjzmaj.edu.rs/) (јануар 2018)

Слике

[Слика 1. Стилови 5](#_Toc531169037)

[Слика 2. Нумерисање и натпис слике 5](#_Toc531169038)

(слично, табеле и графикони)

# БИОГРАФИЈА МАТУРАНТА

Фотографија матуранта

Датум предаје матурског рада: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Комисија:

Председник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Испитивач \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члан \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Коментар:

Датум одбране: \_\_\_\_\_\_\_\_\_\_\_\_\_ Оцена\_\_\_\_\_\_\_\_\_\_ (\_\_\_)