**Colegiul Naţional “A.T.Laurian” Botoşani**

**Profilul matematică-informatică**

LUCRARE DE ATESTAT PROFESIONAL

LA INFORMATICĂ

Profesor coordonator

Cerasela Daniela CARDAS

Nume candidat: Ioniță Filip-Arian

**Tema Lucrării:**

**Vizualizarea Algoritmilor de tip „Pathfinding”**

[Argument 4](#_Toc103529859)

[Limbajul C# 5](#_Toc103529860)

[Descrierea aplicaţiei și Instrucțiuni de rulare 6](#_Toc103529861)

[Implementarea 7](#_Toc103529862)

[Concluzii 19](#_Toc103529863)

[Webografie 20](#_Toc103529864)

# Argument

Adesea elevii de orice vârstă primesc algoritmii într-un format plictisitor, stufos și demodat, mai exact, ei primesc codul, apoi este strict problema lor dacă îl înțeleg și îl pot vizualiza sau nu. Acest lucru este o problemă pentru elevii ce simt nevoia să vadă în timp real sau încetinit pentru a părea real și inteligibil. Astfel, cu această implementare în C#, Windows Forms, consider că am reușit să abordez și chiar să rezolv această problemă pentru unii din cei mai importanți algoritmi de această natură.

# Limbajul C#

**C#** este un [limbaj de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) orientat-obiect conceput de [Microsoft](https://ro.wikipedia.org/wiki/Microsoft) la sfârșitul anilor 90. A fost conceput ca un concurent pentru [limbajul Java](https://ro.wikipedia.org/wiki/Limbajul_Java). Ca și acesta, C# este un derivat al limbajului de programare [C++](https://ro.wikipedia.org/wiki/C%2B%2B).

Ca parte a ansamblului strategiei .NET, dezvoltată de Microsoft, la finele anilor ’90 a fost creat limbajul C#. C# este direct înrudit cu C, C++ și Java. “Bunicul” limbajului C# este C-ul. De la C, C# mosteneste sintaxa, multe din cuvintele cheie și operatorii. De asemenea, C# construieste peste modelul de obiecte definit în in C++. Relatia dintre C# și Java este mai complicata. Java deriva la randul sau din C și C++. Ca și Java, C# a fost proiectat pentru a produce cod portabil. Limbajul C# nu derivă din Java. Intre C# și Java exista o relatie similară celei dintre “veri”, ele deriva din acelasi strămoș, dar deosebindu-se prin multe caracteristici importante.

Limbajul C# contine mai multe facilități inovatoare, dintre care cele mai importante se referă la suportul incorporat pentru componente software. C# dispune de facilitati care implementeaza direct elementele care alcatuiesc componentele software, cum ar fi proprietățile, metodele și evenimentele. Poate cea mai importanta facilitate de care dispune C# este posibilitatea de a lucra intr-un mediu cu limbaj mixt.

C# are o legatură deosebită cu mediul său de rulare, arhitectura .NET. Pe de o parte, C# a fost dezvoltat pentru crearea codului pentru arhitectura .NET, iar pe de alta parte bibliotecile utilizate de C# sunt cele ale arhitecturii .NET. Ce este arhitectura .NET ? Arhitectura .NET defineste un mediu de programare care permite dezvoltarea și executia aplicatiilor indiferent de platforma. Aceasta permite programarea în limbaj mixt și oferă facilități de securitate și portabilitate a programelor. Este disponibila deocamdată pentru platformele Windows.

Legat de C#, arhitectura .NET definește doua entitati importante și anume biblioteca de clase .NET și motorul comun de programare sau Common Language Runtime (CLR). C# nu are o biblioteca de clase proprie ci utilizeaza direct biblioteca de clase .NET. De exemplu, cand se ruleaza un program care efectueaza operatii de intrare-iesire, cum ar fi afisarea unui text pe ecran, se utilizeaza biblioteca de clase .NET.

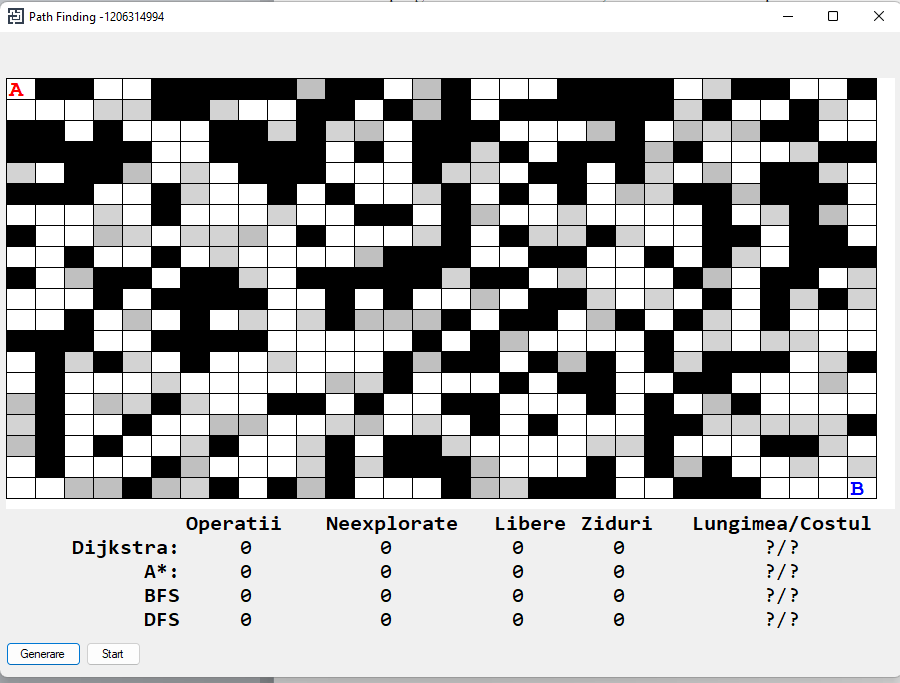
Motorul comun de programare (CLR) se ocupa de executia programelor C#. El asigura de asemenea programarea în limbaj mixt, securitatea și portabilitatea programelor.

Atunci cand este compilat un program C#, sau un program în limbaj mixt, rezultatul compilarii nu este un cod executabil. în locul acestuia, se produce un fisier care contine un tip de pseudocod numit limbaj intermediar sau pe scurt IL (Intermediate Language). Acest fisier IL poate fi copiat în orice calculator care dispune de .NET CLR.

Prin intermediul unui compilator denumit JIT (Just în Time), motorul comun de pogramare transforma codul intermediar în cod executabil. Procesul de conversie decurge astfel: atunci cand un program .NET este executat, CLR activeaza compilatorul JIT.

Compilatorul JIT converteste IL în cod executabil pe masura ce fiecare parte a programului este neceasra. în concluzie, orice program compilat pana în format IL poate rula în orice mediu pentru care CLR este implementat. în acest fel arhitectura .NET asigura portabilitatea.

# Descrierea aplicaţiei și Instrucțiuni de rulare

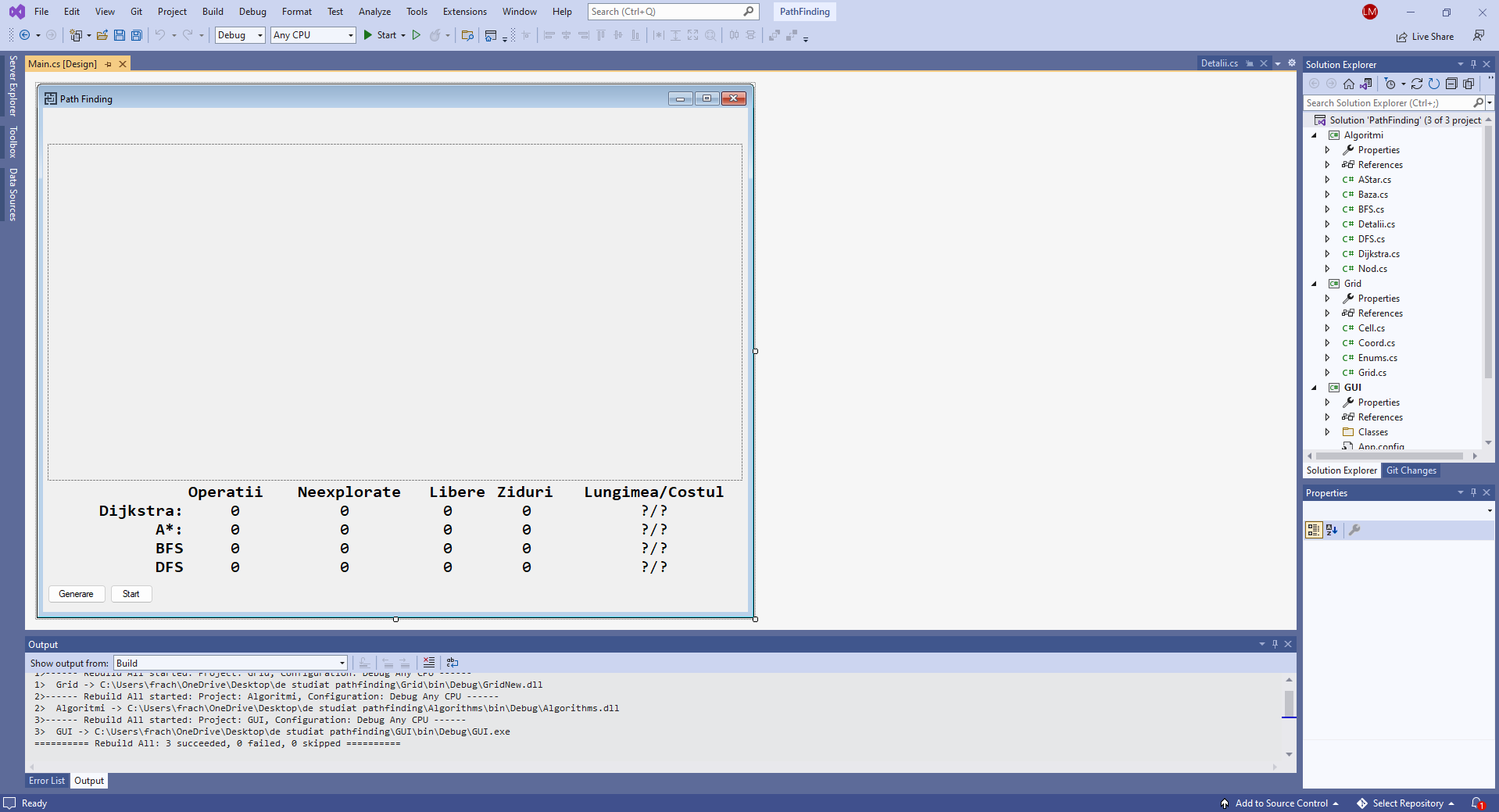
Aplicația oferă soluția problemei menționată anterior, mai exact, o vizualizare a patru din cei mai importanți algoritmi (Dijkstra, A\*, BFS – Breadth First Search și DFS – Depth First Search). La acest lucru contribuie și matricea generată aleatoriu fie la începutul programului, fie la utilizarea butonului „Generare”. În poza de mai jos este meniul de care este întâmpinat utilizatorul la execuție.

Dată fiind împărțirea în clase și namespace-uri diferite, execuția nu este una uzuală unde se poate apăsa pe butonul „Start” iar programul va rula. Acesta are anumite dependențe, respectiv o ordine de execuție. Astfel, metoda recomandată este utilizarea combinației de taste CTRL+SHIFT+B, ce va duce la compilarea programului, ce poate fi lansat apoi din directorul /GUI/bin/Debug/GUI.exe sau, după caz, din /GUI/bin/Release/GUI.exe dacă setările compilatorului sunt pe modul „Release”.

# Implementarea

În aplicația C# am urmărit 2 etape de dezvoltare:

1. **Etapa de proiectare:** În această etapă am adăugat label-urile ce oferă numele algoritmilor alături de numărul de operații efectuate, numărul de noduri neexplorate, numărul de noduri libere, numărul de ziduri și lungimea/costul/ponderea drumului. De asemenea am pus un pictureBox în care va fi reprezentată matricea generată și unde vom putea urmări progresul algoritmilor spre ieșirea din labirint. Ultimele două lucruri adăugate în această etapă sunt butoanele de Generare, care generează un labirint valid (cu ieșire) și Start, ce pornește vizualizarea algoritmului următor.



1. **Etapa de scriere a codului**: Deoarece întregul cod este imens, împărțit în trei namespace-uri (GUI, Grid și Algoritmi), cu câte o clasă pentru fiecare algoritm, am ales să prezint codul din spatele main-ului ce conține metode precum cea de Inițializare, dar și una din cele mai importante metode, cea de verificare a existenței unui drum. Pe lângă main, voi prezenta și algoritmul A\* împreună cu „Baza” fiecărui algoritm de verificare a vecinilor, mai rar întâlnit în mediul școlar din cauza problemelor acestuia când vine vorba de inteligibilitate, iar drept ultimă parte de cod, clasa Grid ce ne oferă matricile și randomizarea lor. De asemenea, codurile nu sunt descrise separat, ci sub formă de comentarii.

## Main.cs-ul GUI-ului()

namespace GUI

{

using Classes;

using System;

using System.Windows.Forms;

using Algoritmi;

using Grid;

using System.Linq;

public partial class Main : Form

{

private MazeDrawer \_matrice\_de\_desenat;

private Baza[] \_algoritmi;

private int \_algoritm\_Actual;

private LabelCollection[] \_nume = new LabelCollection[4];

private System.Timers.Timer \_pathTimer;

private const int Delay = 5;

public Main()

{

InitializeComponent();

// Un Timer care ofera un delay pentru a putea vizualiza algoritmul

\_pathTimer = new System.Timers.Timer(Delay);

\_pathTimer.Elapsed += PathTimer\_Elapsed;

// Vectorul pentru colectarea rezultatelor algoritmilor

\_nume[0] = new LabelCollection

{

AlgoritmId = 0,

Labels = new[] { lblAlgoritm, lblDijkstraOp, lblDijkstraNE, lblDijkstraOpen, lblDijkstraInchis, lblDijkstraLung }

};

\_nume[1] = new LabelCollection

{

AlgoritmId = 1,

Labels = new[] { lblAlgoritm, lblAStarOp, lblAStarNE, lblAStarOpen, lblAStarInchis, lblAStarLung }

};

\_nume[2] = new LabelCollection

{

AlgoritmId = 2,

Labels = new[] { lblAlgoritm, lblBfsOp, lblBfsNE, lblBreadthFirstOpen, lblBreadthFirstInchis, lblBfsLung }

};

\_nume[3] = new LabelCollection

{

AlgoritmId = 3,

Labels = new[] { lblAlgoritm, lblDfsOp, lblDfsNE, lblDepthFirstOpen, lblDepthFirstInchis, lblDfsLung }

};

Initializare();

}

// Funcția de inițializare (Matrice+Variabile)

private void Initializare()

{

\_pathTimer.Stop();

// Generam matrici pana gasim una cu drum valid din A in B

var radacina\_buna = 0;

while (radacina\_buna == 0)

radacina\_buna = Gaseste\_Radacina\_Buna();

// Initializam algoritmul si desenam matricea

\_algoritm\_Actual = -1;

\_matrice\_de\_desenat = new MazeDrawer(pbMaze, radacina\_buna);

\_algoritmi = new Baza[] { new Dijkstra(\_matrice\_de\_desenat.Grid), new AStar(\_matrice\_de\_desenat.Grid), new BFS(\_matrice\_de\_desenat.Grid), new DFS(\_matrice\_de\_desenat.Grid) };

Text = @"Path Finding " + \_matrice\_de\_desenat.Seed;

\_matrice\_de\_desenat.Draw();

}

//Această funcție se ocupă de găsirea propriu-zisă a unei matrici. Matricile sunt generate aleatroiu și probate pentru existența unui drum cu metoda DFS.

private int Gaseste\_Radacina\_Buna()

{

var testMazeDrawer = new MazeDrawer(pbMaze);

var testPathFinder = new DFS(testMazeDrawer.Grid);

var progress = testPathFinder.GetPathTick();

while (progress.ExistaDrum && !progress.GasitDrum)

{

progress = testPathFinder.GetPathTick();

}

return progress.GasitDrum ? testMazeDrawer.Seed : 0;

}

// Obținem urmatorul pas din cautarea drumului

private void PathTimer\_Elapsed(object sender, System.Timers.ElapsedEventArgs e)

{

\_pathTimer.Stop();

var resetTimer = false;

// Treci la urmatorul pas in algoritm

var searchStatus = \_algoritmi[\_algoritm\_Actual].GetPathTick();

StatsUpdate(searchStatus);

// Daca am gasit un drum, il desenam, altfel, actualizam pozitia actuala in matrice

if (searchStatus.GasitDrum)

{

Deseneaza\_Drum(searchStatus);

}

else

{

\_matrice\_de\_desenat.Draw();

resetTimer = true;

}

if (resetTimer) \_pathTimer.Start();

}

//Desenarea caii gasite in matrice

private void Deseneaza\_Drum(Detalii details)

{

for (var i = 1; i < details.Path.Length - 1; i++)

{

var pas = details.Path[i];

\_matrice\_de\_desenat.Grid.SetCell(pas.X, pas.Y, Enums.CellType.Path);

\_matrice\_de\_desenat.Draw();

System.Threading.Thread.Sleep(25);

}

}

// Actualizam starea (afisajul) in functie de algoritmul actual

private void StatsUpdate(Detalii details)

{

var labelCollection = \_nume.First(x => x.AlgoritmId == \_algoritm\_Actual);

labelCollection.Labels[0].BeginInvoke((MethodInvoker)delegate { labelCollection.Labels[0].Text = "Algoritm utilizat: " + \_algoritmi[\_algoritm\_Actual].Nume\_Algoritm; });

labelCollection.Labels[1].BeginInvoke((MethodInvoker)delegate { labelCollection.Labels[1].Text = details.NrOperatii.ToString(); });

labelCollection.Labels[2].BeginInvoke((MethodInvoker)delegate { labelCollection.Labels[2].Text = details.NrNoduriNeexplorate.ToString(); });

labelCollection.Labels[3].BeginInvoke((MethodInvoker)delegate { labelCollection.Labels[3].Text = details.NrNoduriDeschise.ToString(); });

labelCollection.Labels[4].BeginInvoke((MethodInvoker)delegate { labelCollection.Labels[4].Text = details.NrNoduriInchise.ToString(); });

labelCollection.Labels[5].BeginInvoke((MethodInvoker)delegate { labelCollection.Labels[5].Text = details.GasitDrum ? $"{details.Path.Length}/{details.PathCost}" : "?/?"; });

}

// Porneste urmatorul algoritm

private void BtnGo\_Click(object sender, EventArgs e)

{

\_algoritm\_Actual++;

if (\_algoritm\_Actual == \_algoritmi.Length) return;

\_matrice\_de\_desenat.Reset();

\_pathTimer.Start();

}

//Genereaza o noua matrice

private void BtnMaze\_Click(object sender, EventArgs e)

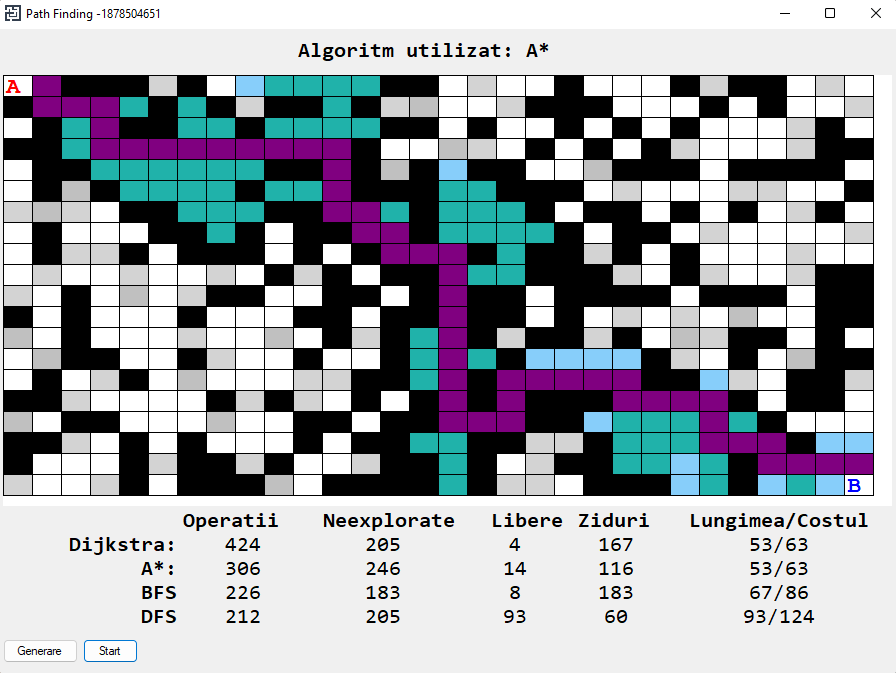
{

Initializare();

}

}

}

În urma utilizării butonului Start, utilizatorul va fi întâmpinat de parcurgerea în direct a labirintului, astfel, după finalizarea parcurgerii, drumul va fi evidențiat cu mov, celulele explorate cu verde albăstrui, iar celulele ce urmau să fie explorate în caz ca nu se ajungea la destinație vor fi marcate cu albastru deschis.

## Baza Algoritmilor:

namespace Algoritmi

{

using Grid;

using System;

using System.Collections.Generic;

using System.Linq;

public abstract class Baza

{

protected readonly Grid Grid;

protected readonly List<Nod> Inchis;

protected List<Coord> Path;

protected readonly Coord origine;

protected readonly Coord Destinatie;

protected int Id;

protected Nod Nod\_Curent;

protected int NrOperatii;

public string Nume\_Algoritm;

protected Baza(Grid grid)

{

Grid = grid;

Inchis = new List<Nod>();

origine = Grid.GetStart().Coord;

Destinatie = Grid.GetEnd().Coord;

NrOperatii = 0;

Id = 1;

}

public abstract Detalii GetPathTick();

//Cautam coordonatele Nord, Sud, Est, Vest presupunand ca sunt valide

protected virtual IEnumerable<Coord> ObtVecini(Nod Actual)

{

var vecini = new List<Cell>

{

Grid.GetCell(Actual.Coord.X - 1, Actual.Coord.Y),

Grid.GetCell(Actual.Coord.X + 1, Actual.Coord.Y),

Grid.GetCell(Actual.Coord.X, Actual.Coord.Y - 1),

Grid.GetCell(Actual.Coord.X, Actual.Coord.Y + 1)

};

return vecini.Where(x => x.Type != Enums.CellType.Invalid && x.Type != Enums.CellType.Solid).Select(x => x.Coord).ToArray();

}

protected abstract Detalii ObtDetalii();

protected static bool CoordsMatch(Coord a, Coord b) => a.X == b.X && a.Y == b.Y;

//Returneaza distanta in "celule" ale matricii de la un punct X pana intr-un punct Y

protected static int Manhatten(Coord origine, Coord destinatie)

{

return Math.Abs(origine.X - destinatie.X) + Math.Abs(origine.Y - destinatie.Y);

}

//returneaza costul drumului intre A si B si returneaza 0 daca drumul e invalid

protected int GetPathCost()

{

if (Path == null)

return 0;

var cost = 0;

foreach (var step in Path)

cost += Grid.GetCell(step.X, step.Y).Weight;

return cost;

}

}

}

**Algoritmul A\*:**

namespace Algoritmi

{

using Grid;

using System.Collections.Generic;

using System.Linq;

public class AStar : Baza

{

private readonly List<Nod> \_deschise = new List<Nod>();

private readonly List<Coord> \_vecini;

public AStar(Grid grid) : base(grid)

{

Nume\_Algoritm = "A\*";

\_vecini = new List<Coord>();

// Adauga nodul de inceput in lista

\_deschise.Add(new Nod(Id++, null, origine, 0, GetH(origine, Destinatie)));

}

public override Detalii GetPathTick()

{

if (Nod\_Curent == null)

{

if (!\_deschise.Any()) return ObtDetalii();

// Scoatem nodul din lista pentru a-l verifica

Nod\_Curent = \_deschise.OrderBy(x => x.F).ThenBy(x => x.H).First();

// il mutam in cealalta lista pentru a nu-l verifica din nou

\_deschise.Remove(Nod\_Curent);

Inchis.Add(Nod\_Curent);

Grid.SetCell(Nod\_Curent.Coord, Enums.CellType.Inchis);

\_vecini.AddRange(ObtVecini(Nod\_Curent));

}

if (\_vecini.Any())

{

Grid.SetCell(Nod\_Curent.Coord, Enums.CellType.Actual);

var thisvecin = \_vecini.First();

\_vecini.Remove(thisvecin);

// Verifica daca vecinul este destinatia

if (CoordsMatch(thisvecin, Destinatie))

{

// Construieste un drum parcurgand invers lista utilizata pana cand nu mai avem noduri de tip "parinte"

Path = new List<Coord> { thisvecin };

int? parentId = Nod\_Curent.Id;

while (parentId.HasValue)

{

var nextNod = Inchis.First(x => x.Id == parentId);

Path.Add(nextNod.Coord);

parentId = nextNod.ParentId;

}

// Inversam drumul pentru a porni de la inceput si a se termina in final

Path.Reverse();

return ObtDetalii();

}

// Obtine costul nodului actual plus pas (Resursele pentru A\* provin de pe GitHub)

var hFromHere = GetH(thisvecin, Destinatie);

var cellWeight = Grid.GetCell(thisvecin.X, thisvecin.Y).Weight;

var vecinCost = Nod\_Curent.G + cellWeight + hFromHere;

// Verifica daca nodul este pe lista deja si daca are un cost mai mare

var openListItem = \_deschise.FirstOrDefault(x => x.Id == ObtNodExistent(true, thisvecin));

if (openListItem != null && openListItem.F > vecinCost)

{

//Redirectionam nodul din lista pentru a folosi drumul de cost minim

openListItem.F = vecinCost;

openListItem.ParentId = Nod\_Curent.Id;

}

// Verifica daca nodul e pe lista deja si daca are un cost mai mare

var InchisListItem = Inchis.FirstOrDefault(x => x.Id == ObtNodExistent(false, thisvecin));

if (InchisListItem != null && InchisListItem.F > vecinCost)

{

//Redirectionam nodul din lista pentru a folosi drumul de cost minim

InchisListItem.F = vecinCost;

InchisListItem.ParentId = Nod\_Curent.Id;

}

// Daca nodul vecin nu e pe nicio lista, il adaugam

if (openListItem != null || InchisListItem != null) return ObtDetalii();

\_deschise.Add(new Nod(Id++, Nod\_Curent.Id, thisvecin, Nod\_Curent.G + cellWeight, hFromHere));

Grid.SetCell(thisvecin.X, thisvecin.Y, Enums.CellType.Open);

}

else

{

Grid.SetCell(Nod\_Curent.Coord, Enums.CellType.Inchis);

Nod\_Curent = null;

return GetPathTick();

}

return ObtDetalii();

}

private static int GetH(Coord origine, Coord destinatie)

{

return Manhattan(origine, destinatie); //După cum știm și din matematică, metoda Manhattan este una din cele mai eficiente metode pentru a obține distanța dintre două puncte într-o matrice. A primit acest nume datorită orașului American, Manhattan, de unde se și inspiră, existând o intersecție la aproape fiecare colț de bloc.

}

private int? ObtNodExistent(bool ListaDeschise, Coord coordDeVerificat)

{

return ListaDeschise ? \_deschise.FirstOrDefault(x => CoordsMatch(x.Coord, coordDeVerificat))?.Id : Inchis.FirstOrDefault(x => CoordsMatch(x.Coord, coordDeVerificat))?.Id;

}

protected override Detalii ObtDetalii()

{

return new Detalii

{

Path = Path?.ToArray(),

PathCost = GetPathCost(),

UltimulNod = Nod\_Curent,

DistantaNodCurent = Nod\_Curent == null ? 0 : GetH(Nod\_Curent.Coord, Destinatie),

NrNoduriDeschise = \_deschise.Count,

NrNoduriInchise = Inchis.Count,

NrNoduriNeexplorate = Grid.GetCountOfType(Enums.CellType.Liber),

NrOperatii = NrOperatii++

};

}

}

}

## Clasa Grid

namespace Grid

{

using System;

using System.Linq;

using static Enums;

public class Grid

{

private readonly Cell[,] \_grid;

public Grid(int CelOrizontale, int CelVerticale)

{

\_grid = new Cell[CelOrizontale, CelVerticale];

for (var x = 0; x < \_grid.GetLength(0); x++)

{

for (var y = 0; y < \_grid.GetLength(1); y++)

{

SetCell(x, y, CellType.Liber);

}

}

SetStartAndEnd();

}

public void Randomize()

{

Randomize((int)DateTime.Now.Ticks);

}

public void Randomize(int seed)

{

var rand = new Random(seed);

// Trece prin toata matricea

for (var x = 0; x < \_grid.GetLength(0); x++)

{

for (var y = 0; y < \_grid.GetLength(1); y++)

{

// Umple casetele cu valori de 1 si 0 aleatoriu

\_grid[x, y].Type = rand.Next(0, 10) > 5 ? CellType.Solid : CellType.Liber;

if (\_grid[x, y].Type != CellType.Liber) continue;

//Daca este liber nodul, adauga un cost aleatoriu acestuia

var weightSpread = rand.Next(0, 10);

if (weightSpread > 8)

\_grid[x, y].Weight = 3;

else if (weightSpread > 6)

\_grid[x, y].Weight = 2;

else

\_grid[x, y].Weight = 1;

}

}

SetStartAndEnd();

}

public Cell GetCell(int x, int y)

{

if (x > \_grid.GetLength(0) - 1 || x < 0 || y > \_grid.GetLength(1) - 1 || y < 0) return new Cell { Coord = new Coord(-1, -1), Type = CellType.Invalid };

return \_grid[x, y];

}

public Cell GetStart()

{

return \_grid.Cast<Cell>().FirstOrDefault(cell => cell.Type == CellType.A);

}

public Cell GetEnd()

{

return \_grid.Cast<Cell>().FirstOrDefault(cell => cell.Type == CellType.B);

}

public void SetCell(int x, int y, CellType type)

{

\_grid[x, y] = new Cell

{

Coord = new Coord(x, y),

Type = type,

Weight = GetCell(x, y)?.Weight ?? 0

};

SetStartAndEnd();

}

public void SetCell(Coord coord, CellType type)

{

SetCell(coord.X, coord.Y, type);

}

public int GetCountOfType(CellType type)

{

var total = 0;

foreach (var cell in \_grid)

{

total += cell.Type == type ? 1 : 0;

}

return total;

}

public int GetTraversableCells()

{

return GetCountOfType(CellType.Open) + GetCountOfType(CellType.A) + GetCountOfType(CellType.B);

}

private void SetStartAndEnd()

{

\_grid[0, 0] = new Cell

{

Coord = new Coord(0, 0),

Type = CellType.A

};

\_grid[\_grid.GetLength(0) - 1, \_grid.GetLength(1) - 1] = new Cell

{

Coord = new Coord(\_grid.GetLength(0) - 1, \_grid.GetLength(1) - 1),

Type = CellType.B

};

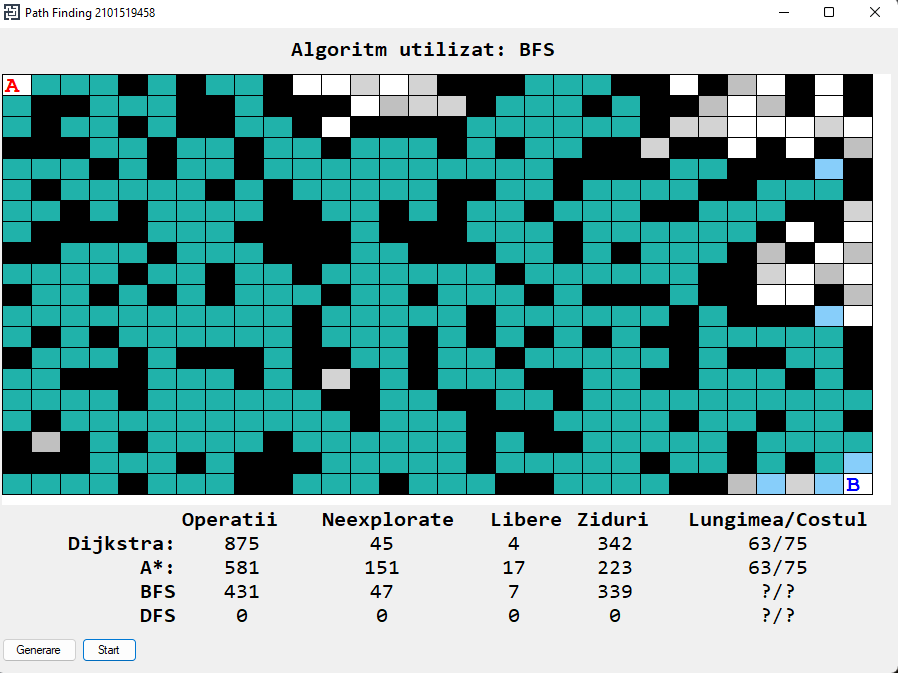
}

}

}

**Am preferat o abordare cu clase atât pentru a-mi lansa o provocare, cât și pentru a oferi un cod clar și citeț. Principiul fiind împărțirea codului în clase ce conțin doar metode și variabile legate, clase ce mai târziu, pot fi instanțate la nevoie.**

## Capturi de ecran din timpul funcționării



# Concluzii

C# este un limbaj de programare simplu de folosit, dar modern și puternic, care din fericire prezintă o documentație vastă cu resurse ușor accesibile. Programarea în C#, în proiecte Windows Forms permite crearea unor aplicații vizuale care facilitează înțelegerea tehnicilor de programare. C# este un limbaj de programare popular, fiindcă are o mentenanță extrem de bună. Pentru acesta există numeroase cadre de lucru (frameworks), biblioteci și instrumente, precum și cursuri C#.

# Webografie

<https://dotnetcoretutorials.com/2020/07/25/a-search-pathfinding-algorithm-in-c/>

<https://www.geeksforgeeks.org/csharp-program-for-dijkstras-shortest-path-algorithm-greedy-algo-7/>

<https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/working-with-linq>

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/>

<https://dotnet.microsoft.com/learn/csharp>

<https://stackoverflow.com/>