**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

**КУРСОВА РОБОТА**

на тему:

«Генерація та Вирішення лабіринту»

з предмету об’єктно орієнтовне програмування С#

студента 2 курсу групи ПД-21

кафедри інженерії програмного забезпечення

Решетніка Нікіти Олександрович

Викладач кафедри

інженерії програмного забезпечення,

Гребенюк Віктор Вікторович

оцінка \_\_\_\_\_\_\_\_\_\_

Київ 2021

**ЗМІСТ**

[ВСТУП 3](#_Toc74258305)

[РОЗДІЛ 1. Теоретичний огляд 4](#_Toc74258306)

[РОЗДІЛ 2. Опис роботи 5](#_Toc74258307)

[ВИСНОВОК 9](#_Toc74258308)

[Список використаної літератури 10](#_Toc74258309)

[ДОДАТОК 1 11](#_Toc74258310)

# ВСТУП

Темою даної курсової роботи являється реалізація генератора «ідеального лабіринту».

Для генерації лабіринту був використаний «Алгоритм Еллера» який полягає в такій послідовності:

1. Створення матриці клітинок з всіма клітинками які відділені стінками
2. Створення початкової клітинки та відмічення її як «відвідана»
3. Поки є не відвідані клітинки
   1. Якщо у клітинки є невідвідані «сусіди»
      1. Вибрати випадкову клітинку з сусідніх
      2. Видалити стіну між поточною клітинкою та обраною
      3. Вибрати обрану клітинку як поточну та відмітити її як «відвідана»

Побудований лабіринт буде «ідеальним» тому що в ньому не буде замкнених областей, і з будь-якої клітинки можна буди дійти до іншої.

# РОЗДІЛ 1. Теоретичний огляд

**Алгоритм Еллера** – алгоритм, що дозволяю створювати лабіринти, які мають тільки один шлях між двома точками. Сам по собі алгоритм дуже швидкий, та використовую пам’ять ефективно ніж інші популярні алгоритми, що дозволяє створити лабіринти більшого розміру при обмежених розмірах оперативної пам’яті

**C#** - об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена **Андерсом Гейлсбергом**, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft).

**Windows Forms** — інтерфейс програмування додатків (API), відповідальний за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за допомогою створення обгортки для Win32 API в керованому коді.

Всередині .NET Framework, Windows Forms реалізується в межах простору імен System.Windows.Forms.

**JetBrains Rider** - кроссплатформенне інтегроване середовище розробки програмного забезпечення для платформи .NET, що розробляється компанією JetBrains. Підтримуються мови програмування C #, VB.NET і F #.

# РОЗДІЛ 2. Опис роботи

Для початку ми повинні створити тип для зберігання інформації про клітинку:

public struct Cell  
{  
 public int X { get; set; }  
 public int Y { get; set; }  
 public bool IsCell { get; set; }  
 public bool IsVisited { get; set; }  
  
 public Cell(int x, int y, bool isVisited = false, bool isCell = true)  
 {  
 X = x;  
 Y = y;  
 IsCell = isCell;  
 IsVisited = isVisited;  
 }  
}

Далі створимо клас для створення лабіринту:

class Maze  
{  
 public readonly Cell[,] \_cells;  
 private int \_width;  
 private int \_height;  
 public Stack<Cell> \_path = new Stack<Cell>();  
 public List<Cell> \_neighbours = new List<Cell>();  
 public Random rnd = new Random();  
 public Cell start;  
 public Cell finish;  
 public Maze(int width, int height)  
 {  
 start = new Cell(1, 1, true, true);  
 finish = new Cell(width - 3, height - 3, true, true);  
 \_width = width;  
 \_height = height;  
 \_cells = new Cell[width, height];  
 for (var i = 0; i < width; i++)  
 for (var j = 0; j < height; j++)  
 if ((i % 2 != 0 && j % 2 != 0) && (i < \_width - 1 && j < \_height - 1))   
 {  
 \_cells[i, j] = new Cell(i, j);   
 }  
 else  
 {  
 \_cells[i, j] = new Cell(i, j, false, false);  
 }  
 \_path.Push(start);  
 \_cells[start.X, start.Y] = start;  
 }  
}

**\_cells –** Масив матриці клітинок.

**\_path –** Стек яких зберігає у собі клітинки проводжуваного шляху

**\_neighbours –** Список сусідніх клітинок

**\_start –** Стартова клітинка

**\_finish –** Фінішна клітинка

Далі треба створити візуальне відтворення лабіринту. Для цього було обрано WinForms.

Наступним кроком є відмальовування лабіринту. Для цього я буду використувавати BitMap де кожна клітинка буде замальовуватися відповідно до того чи является вона стіною чи ні.

Метод для відмальовування лабіринту:

void DrawMaze()  
{  
 \_inBm.Dispose();  
 var bm = new Bitmap(  
 \_cellWid \* (maze.Finish.X + 2),  
 \_cellHgt \* (maze.Finish.Y + 2), PixelFormat.Format16bppRgb555);  
  
 Brush whiteBrush = new SolidBrush(Color.White);  
 Brush blackBrush = new SolidBrush(Color.Black);  
  
 using (var gr = Graphics.FromImage(bm))  
 {  
 gr.SmoothingMode = SmoothingMode.AntiAlias;  
 for (var i = 0; i < maze.Cells.GetUpperBound(0) + oddW; i++)  
 for (var j = 0; j < maze.Cells.GetUpperBound(1) + oddH; j++)  
 {  
 var point = new Point(i \* \_cellWid, j \* \_cellWid);  
 var size = new Size(\_cellWid, \_cellWid);  
 var rec = new Rectangle(point, size);  
 if (maze.Cells[i, j].IsCell)  
 gr.FillRectangle(whiteBrush, rec);  
 else  
 gr.FillRectangle(blackBrush, rec);  
 }  
  
 gr.FillRectangle(new SolidBrush(Color.Green),

new Rectangle(new Point(maze.Start.X \* \_cellWid, maze.Start.Y \* \_cellWid),  
 new Size(\_cellWid, \_cellWid)));  
 gr.FillRectangle(new SolidBrush(Color.Red),   
 new Rectangle(new Point(maze.Finish.X \* \_cellWid, maze.Finish.Y \* \_cellWid),  
 new Size(\_cellWid, \_cellWid)));  
 }  
  
 picMaze.Image = bm

\_inBm = bm;  
}

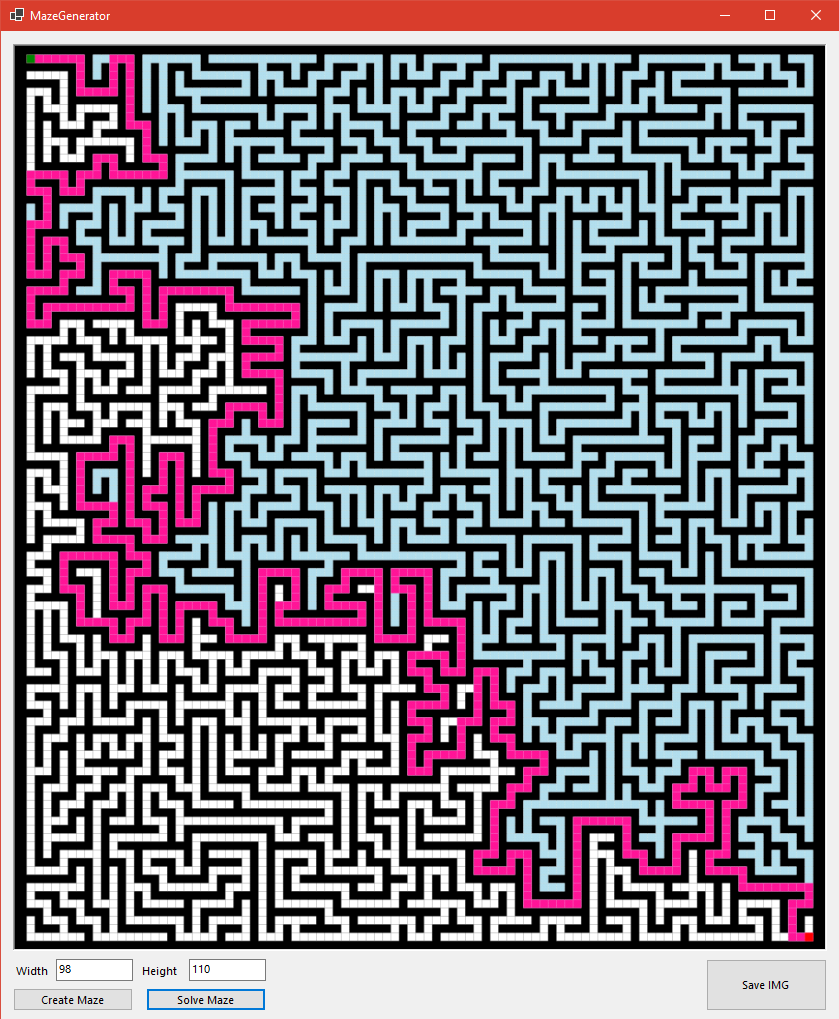
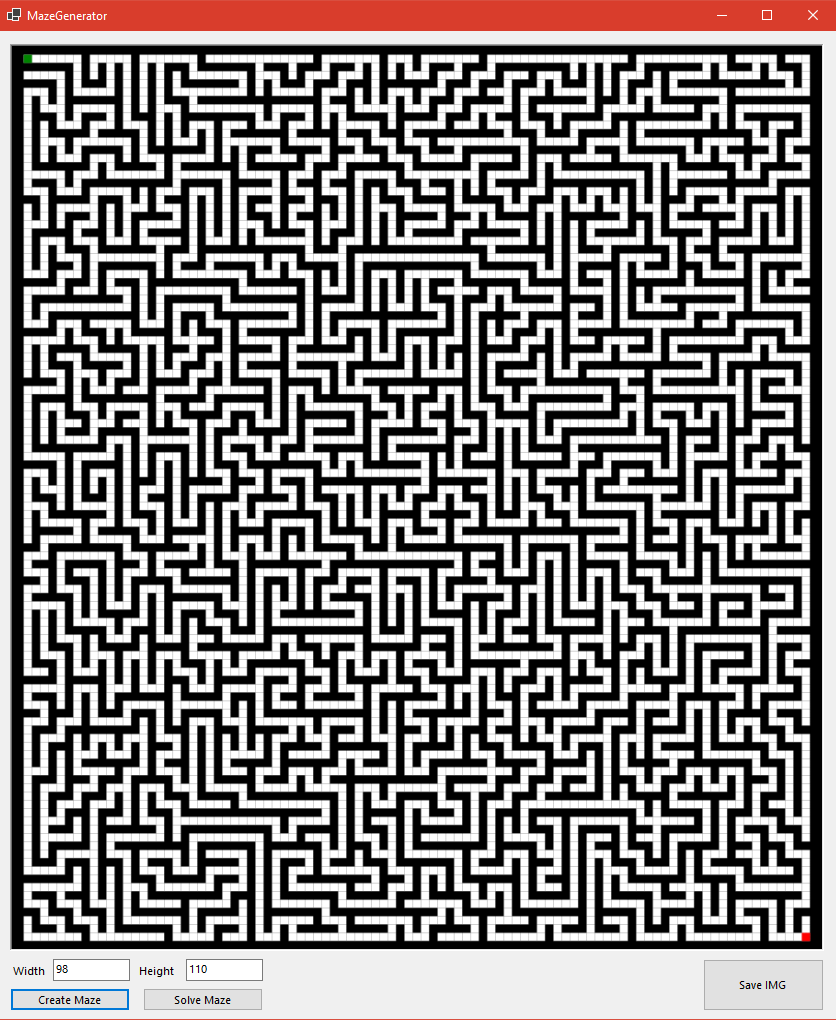
Наступним кроком є вирішення згенерованого лабіринту.

Алгоритм для вирішення лабіринту:

1. Створення початкової клітинки та відмічення її як «відвідана»
2. Поки є не відвідані клітинки
   1. Якщо у клітинки є невідвідані «сусіди»
      1. Вибрати випадкову клітинку з сусідніх
      2. Вибрати обрану клітинку як поточну та відмітити її як «відвідана»

При пошуку рішення лабіринту нам потрібно зберігати шлях який ми подолали від старту до фінішу, а також ми будем зберігати та потім відображати відвідані клітинки. Для відмальовування вірного шляху для проходження будемо використовувати той самий спосіб, що і для відмальовування лабіринту, тобто за допомогою Bitmap.

Результати виконання програми:



# ВИСНОВОК

Отож, у даній курсовій роботі було продемонстровано процес розробки додатку за допомогою мови C# та таких інструментів як WinForms, Rider.

У процесі створення додатку було покращенно мої навички володіння мовою C# та фреймворку WinForms, також корисним бонусом є нові знання про алгоритми генеруваня лабіринтів та іх рішення. Самостійно ознайомитись з додатком ви можете на [GitHub](https://github.com/aldecode/MazeGenerator)

# Список використаної літератури

1. [Генерація і рішення лабіринту за допомогою пошуку по графу](https://habr.com/ru/post/262345/) – Михайло Лубінець Режим доступу: <https://habr.com/ru/post/262345/>
2. [Алгоритм Еллера для генерції лабіринту](https://habr.com/ru/post/176671/) – Режим доступу: <https://habr.com/ru/post/176671/>
3. [Wikipedia: Eller`s algorithm](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%95%D0%BB%D0%BB%D0%B5%D1%80%D0%B0_%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%97_%D0%BB%D0%B0%D0%B1%D1%96%D1%80%D0%B8%D0%BD%D1%82%D1%96%D0%B2) – Режим доступу: <https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%95%D0%BB%D0%BB%D0%B5%D1%80%D0%B0_%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%97_%D0%BB%D0%B0%D0%B1%D1%96%D1%80%D0%B8%D0%BD%D1%82%D1%96%D0%B2>
4. Windows Forms [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://ru.wikipedia.org/wiki/Windows_Forms>.

# ДОДАТОК 1

Код на [**GitHub**](https://github.com/aldecode/MazeGenerator)

MainForm.cs

using System;  
using System.Drawing;  
using System.Drawing.Drawing2D;  
using System.Drawing.Imaging;  
using System.Windows.Forms;  
  
namespace MazeGenerator  
{  
 public partial class Form1 : Form  
 {  
 private int \_cellWid, \_cellHgt;  
 private Bitmap \_inBm = new(1, 1);  
 private Maze \_inMaze = new(10, 10);  
  
 public Form1()  
 {  
 InitializeComponent();  
 }  
  
 private void saveBtn\_Click(object sender, EventArgs e)  
 {  
 var savedialog = new SaveFileDialog();  
 savedialog.Title = "Сохранить картинку как...";  
   
 //отображать ли предупреждение, если пользователь указывает имя уже существующего файла  
 savedialog.OverwritePrompt = true;  
   
 //отображать ли предупреждение, если пользователь указывает несуществующий путь  
 savedialog.CheckPathExists = true;  
   
 //список форматов файла, отображаемый в поле "Тип файла"  
 savedialog.Filter = "Image Files(\*.JPG)|\*.JPG|Image Files(\*.PNG)|\*.PNG|All files (\*.\*)|\*.\*";  
   
 //отображается ли кнопка "Справка" в диалоговом окне  
 savedialog.ShowHelp = true;  
   
 if (savedialog.ShowDialog() == DialogResult.OK)  
 try  
 {  
 picMaze.Image.Save(savedialog.FileName, ImageFormat.Jpeg);  
 }  
 catch  
 {  
 MessageBox.Show("Невозможно сохранить изображение", "Ошибка",  
 MessageBoxButtons.OK, MessageBoxIcon.Error);  
 }  
 }  
  
 private void solveBtn\_Click(object sender, EventArgs e)  
 {  
 \_inMaze.SolveMaze();  
 DrawSolve();  
  
 void DrawSolve()  
 {  
 Brush blueBrush = new SolidBrush(Color.DeepPink);  
 Brush pinkBrush = new SolidBrush(Color.LightBlue);  
 using (var gr = Graphics.FromImage(\_inBm))  
 {  
 gr.SmoothingMode = SmoothingMode.AntiAlias;  
  
 foreach (var c in \_inMaze.Visited)  
 {  
 var point = new Point(c.X \* \_cellWid, c.Y \* \_cellWid);  
 var size = new Size(\_cellWid, \_cellWid);  
 var rec = new Rectangle(point, size);  
 gr.FillRectangle(pinkBrush, rec);  
 }  
  
 foreach (var c in \_inMaze.Solve)  
 {  
 var point = new Point(c.X \* \_cellWid, c.Y \* \_cellWid);  
 var size = new Size(\_cellWid, \_cellWid);  
 var rec = new Rectangle(point, size);  
 gr.FillRectangle(blueBrush, rec);  
 }  
  
 gr.FillRectangle(new SolidBrush(Color.Green),  
 new Rectangle(new Point(\_inMaze.Start.X \* \_cellWid, \_inMaze.Start.Y \* \_cellWid),  
 new Size(\_cellWid, \_cellWid)));  
 gr.FillRectangle(new SolidBrush(Color.Red),  
 new Rectangle(new Point(\_inMaze.Finish.X \* \_cellWid, \_inMaze.Finish.Y \* \_cellWid),  
 new Size(\_cellWid, \_cellWid)));  
 }  
  
 picMaze.Image = \_inBm;  
 }  
 }  
  
  
 private void createBtn\_Click(object sender, EventArgs e)  
 {  
 var width = 0;  
 var height = 0;  
  
 try  
 {  
 width = int.Parse(txtWidth.Text);  
 height = int.Parse(txtHeight.Text);  
  
 if (width == 0 || height == 0) throw new FormatException();  
 }  
 catch (FormatException)  
 {  
 const string message = "Размерность должна быть числом, больше 0.";  
 var caption = "Ошибка ввода размерности";  
 var buttons = MessageBoxButtons.OK;  
 MessageBox.Show(message, caption, buttons);  
 txtWidth.Text = @"10";  
 txtHeight.Text = @"10";  
  
 return;  
 }  
  
  
 var oddW = 0;  
 var oddH = 0;  
   
 if (width % 2 != 0) oddW = 1;  
 if (height % 2 != 0) oddH = 1;  
   
 \_cellWid = picMaze.ClientSize.Width / (width + 2);  
 \_cellHgt = picMaze.ClientSize.Height / (height + 2);  
   
 var cellMin = 10;  
 if (\_cellWid < cellMin)  
 {  
 \_cellWid = cellMin;  
 \_cellHgt = \_cellWid;  
 }  
 else if (\_cellHgt < cellMin)  
 {  
 \_cellHgt = cellMin;  
 \_cellWid = \_cellHgt;  
 }  
 else if (\_cellWid > \_cellHgt)  
 {  
 \_cellWid = \_cellHgt;  
 }  
 else  
 {  
 \_cellHgt = \_cellWid;  
 }  
  
  
 var maze = new Maze(width, height);  
   
 maze.Finish.X = maze.Finish.X + oddW;  
 maze.Finish.Y = maze.Finish.Y + oddH;  
 maze.CreateMaze();  
 DrawMaze();  
  
 \_inMaze = maze;  
  
 void DrawMaze()  
 {  
 \_inBm.Dispose();  
   
 var bm = new Bitmap(  
 \_cellWid \* (maze.Finish.X + 2),  
 \_cellHgt \* (maze.Finish.Y + 2), PixelFormat.Format16bppRgb555);  
  
 Brush whiteBrush = new SolidBrush(Color.White);  
 Brush blackBrush = new SolidBrush(Color.Black);  
  
 using (var gr = Graphics.FromImage(bm))  
 {  
 gr.SmoothingMode = SmoothingMode.AntiAlias;  
 for (var i = 0; i < maze.Cells.GetUpperBound(0) + oddW; i++)  
 for (var j = 0; j < maze.Cells.GetUpperBound(1) + oddH; j++)  
 {  
 var point = new Point(i \* \_cellWid, j \* \_cellWid);  
 var size = new Size(\_cellWid, \_cellWid);  
 var rec = new Rectangle(point, size);  
 if (maze.Cells[i, j].IsCell)  
 gr.FillRectangle(whiteBrush, rec);  
 else  
 gr.FillRectangle(blackBrush, rec);  
 }  
  
 gr.FillRectangle(new SolidBrush(Color.Green),  
 new Rectangle(new Point(maze.Start.X \* \_cellWid, maze.Start.Y \* \_cellWid),  
 new Size(\_cellWid, \_cellWid)));  
 gr.FillRectangle(new SolidBrush(Color.Red),  
 new Rectangle(new Point(maze.Finish.X \* \_cellWid, maze.Finish.Y \* \_cellWid),  
 new Size(\_cellWid, \_cellWid)));  
 }  
  
 picMaze.Image = bm;  
 \_inBm = bm;  
 }  
 }  
 }  
}

Cell.cs

namespace MazeGenerator  
{  
 public struct Cell  
 {  
 public int X { get; set; }  
 public int Y { get; set; }  
 public bool IsCell { get; set; }  
 public bool IsVisited { get; set; }  
  
 public Cell(int x, int y, bool isVisited = false, bool isCell = true)  
 {  
 X = x;  
 Y = y;  
 IsCell = isCell;  
 IsVisited = isVisited;  
 }  
 }  
}

Maze.cs

using System;  
using System.Collections.Generic;  
  
namespace MazeGenerator  
{  
 internal class Maze  
 {  
 public readonly Cell[,] Cells;  
 public readonly int Height;  
 public readonly List<Cell> Neighbours = new();  
 public Cell Finish;  
 public Stack<Cell> Path = new();  
 public Random Rnd = new();  
 public List<Cell> Solve = new();  
 public Cell Start;  
 public List<Cell> Visited = new();  
 public int Width;  
  
 public Maze(int width, int height)  
 {  
 Start = new Cell(1, 1, true);  
 Finish = new Cell(width - 3, height - 3, true);  
  
  
 Width = width;  
 Height = height;  
 Cells = new Cell[width, height];  
 for (var i = 0; i < width; i++)  
 for (var j = 0; j < height; j++)  
 if (i % 2 != 0 && j % 2 != 0 && i < Width - 1 &&  
 j < Height - 1)  
 Cells[i, j] = new Cell(i, j);  
 else  
 Cells[i, j] = new Cell(i, j, false, false);  
 Path.Push(Start);  
 Cells[Start.X, Start.Y] = Start;  
 }  
  
 public void SolveMaze()  
 {  
 var flag = false;  
 foreach (var c in Cells)  
 if (Cells[c.X, c.Y].IsCell)  
 Cells[c.X, c.Y].IsVisited = false;  
  
 Path.Clear();  
 Path.Push(Start);  
  
 while (Path.Count != 0)  
 {  
 if (Path.Peek().X == Finish.X && Path.Peek().Y == Finish.Y) flag = true;  
  
 if (!flag)  
 {  
 Neighbours.Clear();  
 GetNeighboursSolve(Path.Peek());  
 if (Neighbours.Count != 0)  
 {  
 var nextCell = ChooseNeighbour(Neighbours);  
 nextCell.IsVisited = true;  
 Cells[nextCell.X, nextCell.Y].IsVisited = true;  
 Path.Push(nextCell);  
 Visited.Add(Path.Peek());  
 }  
 else  
 {  
 Path.Pop();  
 }  
 }  
 else  
 {  
 Solve.Add(Path.Peek());  
 Path.Pop();  
 }  
 }  
 }  
  
  
 public void CreateMaze()  
 {  
 Cells[Start.X, Start.Y] = Start;  
 while (Path.Count != 0)  
 {  
 Neighbours.Clear();  
 GetNeighbours(Path.Peek());  
 if (Neighbours.Count != 0)  
 {  
 var nextCell = ChooseNeighbour(Neighbours);  
 RemoveWall(Path.Peek(), nextCell);  
 nextCell.IsVisited = true;  
 Cells[nextCell.X, nextCell.Y].IsVisited = true;  
 Path.Push(nextCell);  
 }  
 else  
 {  
 Path.Pop();  
 }  
 }  
 }  
  
 private void GetNeighbours(Cell localcell)  
 {  
 var x = localcell.X;  
 var y = localcell.Y;  
 const int distance = 2;  
 Cell[] possibleNeighbours =  
 {  
 new(x, y - distance),  
 new(x + distance, y),  
 new(x, y + distance),  
 new(x - distance, y)  
 };  
 for (var i = 0; i < 4; i++)  
 {  
 var curNeighbour = possibleNeighbours[i];  
 if (curNeighbour.X > 0 && curNeighbour.X < Width && curNeighbour.Y > 0 && curNeighbour.Y < Height)  
  
 if (Cells[curNeighbour.X, curNeighbour.Y].IsCell &&  
 !Cells[curNeighbour.X, curNeighbour.Y].IsVisited)  
  
 Neighbours.Add(curNeighbour);  
 }  
 }  
  
 private void GetNeighboursSolve(Cell localcell)  
 {  
 var x = localcell.X;  
 var y = localcell.Y;  
 const int distance = 1;  
 Cell[] possibleNeighbours =  
 {  
 new(x, y - distance),  
 new(x + distance, y),  
 new(x, y + distance),  
 new(x - distance, y)  
 };  
 for (var i = 0; i < 4; i++)  
 {  
 var curNeighbour = possibleNeighbours[i];  
 if (curNeighbour.X > 0 && curNeighbour.X < Width && curNeighbour.Y > 0 && curNeighbour.Y < Height)  
 if (Cells[curNeighbour.X, curNeighbour.Y].IsCell &&  
 !Cells[curNeighbour.X, curNeighbour.Y].IsVisited)  
  
 Neighbours.Add(curNeighbour);  
 }  
 }  
  
 private Cell ChooseNeighbour(List<Cell> neighbours)  
 {  
 var r = Rnd.Next(neighbours.Count);  
 return neighbours[r];  
 }  
  
 private void RemoveWall(Cell first, Cell second)  
 {  
 var xDiff = second.X - first.X;  
 var yDiff = second.Y - first.Y;  
 var addX = xDiff != 0 ? xDiff / Math.Abs(xDiff) : 0;  
 var addY = yDiff != 0 ? yDiff / Math.Abs(yDiff) : 0;  
  
  
 Cells[first.X + addX, first.Y + addY].IsCell = true;  
 Cells[first.X + addX, first.Y + addY].IsVisited = true;  
 second.IsVisited = true;  
 Cells[second.X, second.Y] = second;  
 }  
 }  
}

Program.cs

using System;  
using System.Windows.Forms;  
  
namespace MazeGenerator  
{  
 internal static class Program  
 {  
 /// <summary>  
 /// The main entry point for the application.  
 /// </summary>  
 [STAThread]  
 private static void Main()  
 {  
 Application.EnableVisualStyles();  
 Application.SetCompatibleTextRenderingDefault(false);  
 Application.Run(new Form1());  
 }  
 }  
}