

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Objektinis programavimas II (P175B123)
Darbų aplankas

Atliko:

IFF-8/2 gr. studentas

Nedas Šimoliūnas

2019 m. kovo 3 d.

Priėmė:

Lekt. Mindaugas Jančiukas

TURINYS

1. Rekursija (L1).....	3
1.1. Darbo užduotis	3
1.2. Grafinės vartotojo sąsajos schema	3
1.3. Sąsajoje panaudotų komponentų keičiamos savybės	4
1.4. Klasių diagrama.....	5
1.5. Programos vartotojo vadovas	5
1.6. Programos tekstas.....	5
1.7. Pradiniai duomenys ir rezultatai.....	17
1.8. Dėstytojo pastabos.....	19
2. Dinaminis atminties valdymas (L2).....	20
2.1. Darbo užduotis	20
2.2. Grafinės vartotojo sąsajos schema	20
2.3. Sąsajoje panaudotų komponentų keičiamos savybės	20
2.4. Klasių diagrama.....	20
2.5. Programos vartotojo vadovas	20
2.6. Programos tekstas.....	20
2.7. Pradiniai duomenys ir rezultatai.....	20
2.8. Dėstytojo pastabos.....	20
3. Bendrinės klasės ir sąsajos (L3).....	21
3.1. Darbo užduotis	21
3.2. Grafinės vartotojo sąsajos schema	21
3.3. Sąsajoje panaudotų komponentų keičiamos savybės	21
3.4. Klasių diagrama.....	21
3.5. Programos vartotojo vadovas	21
3.6. Programos tekstas.....	21
3.7. Pradiniai duomenys ir rezultatai.....	21
3.8. Dėstytojo pastabos.....	21
4. Kolekcijos ir išimčių valdymas (L4).....	22
4.1. Darbo užduotis	22
4.2. Grafinės vartotojo sąsajos schema	22
4.3. Sąsajoje panaudotų komponentų keičiamos savybės	22
4.4. Klasių diagrama.....	22
4.5. Programos vartotojo vadovas	22
4.6. Programos tekstas.....	22
4.7. Pradiniai duomenys ir rezultatai.....	22
4.8. Dėstytojo pastabos.....	22
5. Deklaratyvusis programavimas (L5).....	23
5.1. Darbo užduotis	23
5.2. Grafinės vartotojo sąsajos schema	23
5.3. Sąsajoje panaudotų komponentų keičiamos savybės	23
5.4. Klasių diagrama.....	23
5.5. Programos vartotojo vadovas	23
5.6. Programos tekstas.....	23
5.7. Pradiniai duomenys ir rezultatai.....	23
5.8. Dėstytojo pastabos.....	23

1. Rekursija (L1)

1.1. Darbo užduotis

LD_24.Susitikimas.

Grupė draugų nusprendė susitikti mieste, o po to kartu eiti valgyti picos. Bet tuomet jie susiginčijo, kur geriausia susitikti, ir kurioje picerijoje valgyti. Galiausiai draugai nusprendė, jog patogiausia išsirinkti tokią susitikimo vietą ir piceriją, kad jų nueitų kelių iki susitikimo vietos, po to iki picerijos ir atgal į pradžios taškus suma būtų mažiausia. Parašykite programą, kuri rastų patogiausią susitikimo vietą ir piceriją.

Duomenys. Pirmoje „U3.txt“ eilutėje duoti du skaičiai n ir m — miesto žemėlapis aukštis ir plotis ($2 \leq n, m \leq 10$). Toliau pateiktas miesto žemėlapis — n eilučių, kiekvienoje jų — m simbolių. Galimi šie simboliai:

. — langelis **pereinamas**.

X — langelis **nepereinamas** (pastatas, tvora...)

P — picerija. Langelis **nepereinamas**, į piceriją galima įeiti iš gretimo langelio ir išeiti į bet kurį gretimą langelį. Įeiti galima tik į tą piceriją, kurioje bus valgoma pica.

S — susitikimo vieta. Langelis **pereinamas**.

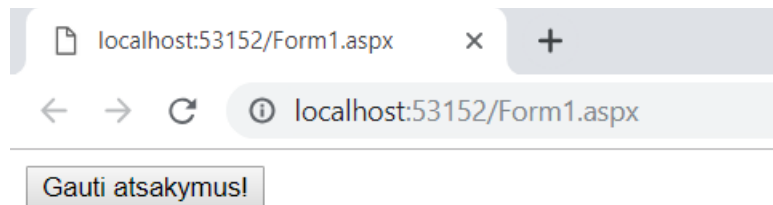
D — vieno iš draugų pradinė buvimo vieta. Langelis **pereinamas**.

Judėti galima tik aukštyn, žemyn, į kairę arba į dešinę (ne įstrižai).

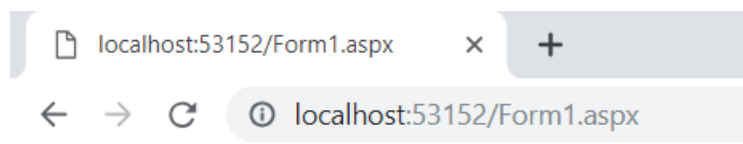
Rezultatai. Ekrane atskirose eilutėse atspausdinkite pradines draugų koordinates (koordinatų pradžia – apatinis kairysis langelis, numeruojama nuo 1, pirmiausiai nurodoma x koordinatė, po to y). Atskirose eilutėse atspausdinkite: „Susitikimo vieta“ ir susitikimo vietos koordinates; „Picerija“ ir picerijos koordinates; „Nueita“ ir draugų nueitų kelių sumą. Jei susitikimo vietos ar picerijos, kurias visi draugai galėtų pasiekti, nėra, atspausdinkite žodį „Neįmanoma“.

1.2. Grafinės vartotojo sąsajos schema

Įsijungus programą:



Paspaudus mygtuką:



Gauti atsakymus!

1 10

2 10

Susitikimo vieta 1 9

Picerija 4 10

Nueita 54

10	D	D	X	P
9	S	.	X
8	.	.	X
7	.	.	X
6	.	.	X
5
4	.	.	X	X	X	X	X	X	X	.
3
2
1	P	.	.	.
	1	2	3	4	5	6	7	8	9	10

P - picerija

D - vieno iš draugų pradinė vieta

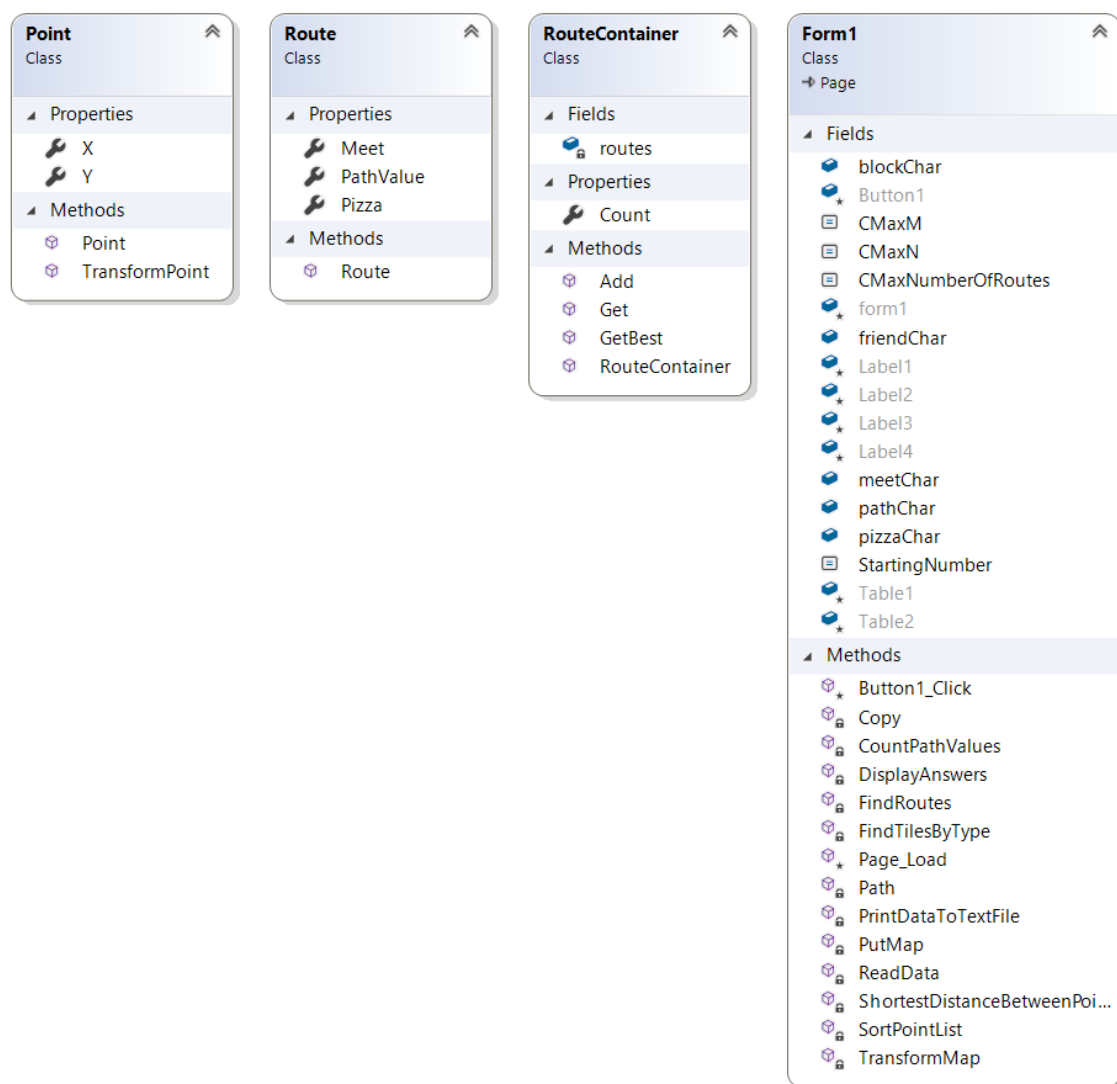
S - galima susitikimo vieta

. - praeinamas langelis

1.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Button1	Click	Kreipiasi į skaičiavimo ir spaudinimo metodus
Table1	Text	Rezultatai
Table2	Text	Pradiniai duomenys
Label1	Text	Pradiniai duomenys
Label2	Text	Pradiniai duomenys
Label3	Text	Pradiniai duomenys
Label4	Text	Pradiniai duomenys

1.4. Klasių diagrama



1.5. Programos vartotojo vadovas

Programos aplanke „App_Data“ esančio failo „U3.txt“ pirmoje eilutėje yra 2 skaičiai, reiškiantys žemėlapių aukštį ir plotį. Kitose eilutėse iš įvairių ženklų sudarytas žemėlapis. Pačioje programoje matome vieną mygtuką, kurį paspaudę, matome užduoties rezultatus, pavaizduotą žemėlapi bei paaiškinimus.

1.6. Programos tekstas

Point.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace L1
{
    /// <summary>
    /// A class designed to store data for Point.
    /// </summary>
    public class Point
    {
```

```

public int X { get; set; }
public int Y { get; set; }

/// <summary>
/// Constructor with parameters.
/// </summary>
/// <param name="x_"> Point's X coordinate </param>
/// <param name="y_"> Point's Y coordinate </param>
public Point(int x_, int y_)
{
    X = x_;
    Y = y_;
}

/// <summary>
/// Changes X and Y values to for correct answer.
/// </summary>
/// <param name="n"> Map's height </param>
public void TransformPoint(int n)
{
    int xTemp = X;
    X = Y + 1;
    Y = n - xTemp;
}
}

```

Route.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace L1
{
    /// <summary>
    /// Class designed to store data for individual Route.
    /// </summary>
    public class Route
    {
        public Point Meet { get; set; }
        public Point Pizza { get; set; }
        public int PathValue { get; set; }

        /// <summary>
        /// Constructor with parameters.
        /// </summary>
        /// <param name="meetX"> Meeting place's X coordinate </param>
        /// <param name="meetY"> Meeting place's Y coordinate </param>
        /// <param name="pizzaX"> Pizzeria's X coordinate </param>
        /// <param name="pizzaY"> Pizzeria's Y coordinate </param>
        public Route(int meetX, int meetY, int pizzaX, int pizzaY)
        {
            Meet = new Point(meetX, meetY);
            Pizza = new Point(pizzaX, pizzaY);
            PathValue = -1;
            //sets distance as -1, will change if the path is valid
        }
    }
}

```

RouteContainer.cs:

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Web;

namespace L1
{
    /// <summary>
    /// Class designed to store data of various Routes.
    /// </summary>
    public class RouteContainer
    {
        Route[] routes;
        public int Count { get; private set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="size"> Max size for array </param>
        public RouteContainer(int size)
        {
            routes = new Route[size];
            Count = 0;
        }

        /// <summary>
        /// Adds a Route to array
        /// </summary>
        /// <param name="route"> Route to add </param>
        public void Add(Route route)
        {
            routes[Count++] = route;
        }

        /// <summary>
        /// Returns a Route
        /// </summary>
        /// <param name="index"> Index of the Route </param>
        /// <returns> Route by index </returns>
        public Route Get(int index)
        {
            return routes[index];
        }

        /// <summary>
        /// Returns a Route which has the lowest positive value of PathValue.
        /// </summary>
        /// <returns> Most optimal Route </returns>
        public Route GetBest()
        {
            Route route = routes[0];

            for (int i = 1; i < Count; i++)
            {
                if(routes[i].PathValue > 0 &&
                    route.PathValue > routes[i].PathValue)
                {
                    route = routes[i];
                }
            }

            return route;
        }
    }
}

```

Form1.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Form1.aspx.cs"
Inherits="L1.Form1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Text="Gauti atsakymus!" />
            <asp:Table ID="Table1" runat="server">
            </asp:Table>
            <br />
            <asp:Table ID="Table2" runat="server" GridLines="Both">
            </asp:Table>
        </div>
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
        <br />
        <asp:Label ID="Label2" runat="server"></asp:Label>
        <br />
        <asp:Label ID="Label3" runat="server"></asp:Label>
        <br />
        <asp:Label ID="Label4" runat="server"></asp:Label>
    </form>
</body>
</html>
```

Form1.aspx.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;

namespace L1
{
    public partial class Form1 : System.Web.UI.Page
    {
        public const int CMaxNumberOfRoutes = 50;
        // Maximum number of routes
        public char friendChar = 'D';
        // Friend tile in map
        public char meetChar = 'S';
        // Meeting place tile in map
        public char pizzaChar = 'P';
        // Pizzeria tile in map
        public char blockChar = 'X';
        // Blocking tile in map
        public char pathChar = '.';
        // Free movement tile in map
        public const int CMaxN = 20;
        // Maximum number of map height
        public const int CMaxM = 20;
        // Maximum number of map width
        public const int StartingNumber = 65;
        // Constant number to determine various path values
```



```

protected void Page_Load(object sender, EventArgs e)
{

}

protected void Button1_Click(object sender, EventArgs e)
{
    string fileName = "App_Data/U3.txt";
    string resultFile = "App_Data/ResultFile.txt";
    int n, m;
    char[,] map;

    ReadData(fileName, out n, out m, out map);

    RouteContainer routes;
    List<Point> friends;

    FindTilesByType(out friends, map, n, m, friendChar);

    FindRoutes(map, n, m, out routes);

    CountPathValues(map, routes, friends, n, m, StartingNumber);

    Route bestRoute = routes.GetBest();

    DisplayAnswers(friends, bestRoute, n);

    PutMap(map, n, m);

    PrintDataToTextFile(map, n, m, friends, bestRoute, resultFile);
}

/// <summary>
/// Prints starting and ending data to file
/// </summary>
/// <param name="map"> Char array of map elements </param>
/// <param name="n"> Map's height </param>
/// <param name="m"> Map's width </param>
/// <param name="friends"> List of friends coordinates </param>
/// <param name="route"> Most optimal route </param>
/// <param name="fileName"> Name of file </param>
void PrintDataToTextFile(char[,] map, int n, int m,
                        List<Point> friends, Route route,
                        string fileName)
{
    using(StreamWriter writer =
        new StreamWriter(Server.MapPath(fileName)))
    {
        int lineLength = 2 * m + 1;
        string line = new string('-', lineLength);
        writer.WriteLine(line);
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                writer.Write("|" + map[i, j]);
            }
            writer.Write("|");
            writer.WriteLine();
            writer.WriteLine(line);
        }
        writer.WriteLine();
        foreach(Point point in friends)
        {
            writer.WriteLine(point.X + " " + point.Y);
        }
    }
}

```

```

    }
    if(route.PathValue < 0)
    {
        writer.WriteLine("Neįmanoma");
    }
    else
    {
        writer.WriteLine("Susitikimo vieta " + route.Meet.X +
            " " + route.Meet.Y);
        writer.WriteLine("Picerija " + route.Pizza.X +
            " " + route.Pizza.Y);
        writer.WriteLine("Nueita " + route.PathValue);
    }
}

}

/// <summary>
/// Reads file data
/// </summary>
/// <param name="fileName"> Name of file </param>
/// <param name="n"> Map's height, returned by argument </param>
/// <param name="m"> Map's width, returned by argument </param>
/// <param name="map"> Char array of map elements,
/// returned by argument </param>
void ReadData(string fileName, out int n, out int m, out char[,] map)
{
    string[] lines = File.ReadAllLines(Server.MapPath(fileName));
    string[] values = lines[0].Split(' ');
    n = int.Parse(values[0]);
    m = int.Parse(values[1]);
    map = new char[CMaxN, CMaxM];
    for (int i = 0; i < n; i++)
    {
        string line = lines[i + 1];
        for (int j = 0; j < m; j++)
        {
            map[i, j] = line[j];
        }
    }
}

/// <summary>
/// Finds all points in map by given type
/// </summary>
/// <param name="points"> List of found points
/// to be returned by argument </param>
/// <param name="map"> Char array of map elements </param>
/// <param name="n"> Map's height </param>
/// <param name="m"> Map's width </param>
/// <param name="type"> Char of tile to look for </param>
void FindTilesByType(out List<Point> points, char[,] map,
    int n, int m, char type)
{
    points = new List<Point>();
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if(map[i, j] == type)
            {
                points.Add(new Point(i, j));
            }
        }
    }
}
}

```

```

/// <summary>
/// Finds all the routes in the map
/// </summary>
/// <param name="map"> Char array of map elements </param>
/// <param name="n"> Map's height </param>
/// <param name="m"> Map's width </param>
/// <param name="routes"> Newly created route container
/// returned through argument </param>
void FindRoutes(char[,] map, int n, int m, out RouteContainer routes)
{
    routes = new RouteContainer(CMaxNumberOfRoutes);

    List<Point> meets;
    List<Point> pizzas;

    FindTilesByType(out meets, map, n, m, meetChar);
    FindTilesByType(out pizzas, map, n, m, pizzaChar);

    // Creates all routes, which contains
    // one meeting place and one pizzeria
    foreach(Point meet in meets)
    {
        foreach(Point pizza in pizzas)
        {
            Route route = new Route(meet.X, meet.Y, pizza.X, pizza.Y);
            routes.Add(route);
        }
    }
}

/// <summary>
/// Counts path values for each route
/// </summary>
/// <param name="map"> Char array of map elements </param>
/// <param name="routes"> Route container </param>
/// <param name="friends"> List of all of friends coordinates </param>
/// <param name="n"> Map's height </param>
/// <param name="m"> Map's width </param>
/// <param name="number"> Constant which determines length </param>
void CountPathValues(char[,] map, RouteContainer routes,
    List<Point> friends, int n,
    int m, int number)
{
    bool t = true;

    for (int i = 0; i < routes.Count; i++)
    {
        Route route = routes.Get(i);
        int value = 0;

        foreach(Point point in friends)
        {
            // Finds shortest distance between
            // friend and a meeting place
            int value1 =
                ShortestDistanceBetweenPoints(map, n, m, point,
                    route.Meet, number);

            // If a friend can go there, his value will be positive.
            if(value1 <= 0)
            {
                // Sets bool value to false, later the whole route will
                // be ignored.
                t = false;
                continue;
            }
        }
    }
}

```

```

    }

    // Friend goes to the meeting place and back, hence
    // multiplying value by 2
    value += 2 * value1;
}

// Finds shortest distance between
// routes meeting place and pizzeria
int value2 =
    ShortestDistanceBetweenPoints(map, n, m, route.Meet,
                                   route.Pizza, number);

// If path is not valid or at least one friend can't get there,
// we move on to another route.
if(value2 <= 0 || !t)
{
    continue;
}

// Each friend goes to the pizzeria and back to meeting place,
// hence multiplying by 2, and all of them takes the same route,
// hence multiplying by friends count
value += 2 * value2 * friends.Count;

// Sets routes path value. If the route is not valid,
// path value will stay as -1 by default action in the
// constructor
routes.Get(i).PathValue = value;
}
}

/// <summary>
/// Finds value of shortest distance between two given points
/// </summary>
/// <param name="map"> Char array of map elements </param>
/// <param name="n"> Map's height </param>
/// <param name="m"> Map's width </param>
/// <param name="a"> First given point </param>
/// <param name="b"> Second given point </param>
/// <param name="number"> Constant which determines length </param>
/// <returns> Value of shortest distance </returns>
int ShortestDistanceBetweenPoints(char[,] map, int n, int m,
                                   Point a, Point b, int number)
{
    char[,] mapCopy = new char[CMaxN, CMaxM];

    // Creates map copy, because we don't want to change the
    // original map
    mapCopy = Copy(map, n, m);
    TransformMap(mapCopy, n, m, a, b);

    // Adds starting point to list, which will be used
    // as a center of the wave created in the Path method.
    List<Point> points = new List<Point>();
    points.Add(a);

    Path(mapCopy, points, b, number, n, m);

    // Starting constant, subtracted from ending point's value
    // will evaluate to distance between 2 given points.
    return mapCopy[b.X, b.Y] - number;
}

/// <summary>
/// Copies all of char array's elements to other

```

```

/// </summary>
/// <param name="map"> Char array of map elements </param>
/// <param name="n"> Map's height </param>
/// <param name="m"> Map's width </param>
/// <returns> A copy of given char array </returns>
char[,] Copy(char[,] map, int n, int m)
{
    char[,] copy = new char[CMaxN, CMaxM];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            copy[i, j] = map[i, j];
        }
    }
    return copy;
}

/// <summary>
/// Changes blocking tiles to 1's and other to 0's
/// </summary>
/// <param name="map"> Char array of map elements </param>
/// <param name="n"> Map's height </param>
/// <param name="m"> Map's width </param>
/// <param name="a"> First point of interest </param>
/// <param name="b"> Second point of interest </param>
void TransformMap(char[,] map, int n, int m, Point a, Point b)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if(map[i, j] == blockChar ||
                map[i, j] == pizzaChar)
            {
                map[i, j] = '1';
            }
            if(map[i, j] == friendChar ||
                map[i, j] == meetChar ||
                map[i, j] == pathChar)
            {
                map[i, j] = '0';
            }
        }
    }
    map[a.X, a.Y] = '0';
    map[b.X, b.Y] = '0';
}

/// <summary>
/// Recursive method, which changes map values
/// </summary>
/// <param name="map"> Char array of map elements </param>
/// <param name="points"> List of points which are being used </param>
/// <param name="end"> End coordinate </param>
/// <param name="number"> Number which is used
/// to change map values </param>
/// <param name="n"> Map's height </param>
/// <param name="m"> Map's width </param>
void Path(char[,] map, List<Point> points, Point end,
    int number, int n, int m)
{
    List<Point> nextPoints = new List<Point>();

    // Checks if ending point in a map is changed, or
    // there aren't any points in a list to

```

```

// spread the wave in a map.
if(map[end.X, end.Y] > StartingNumber ||
    points.Count == 0)
{
    return;
}

// For each point in list, we're checking 4 neighbours
// and if their values haven't changed or they can still
// be improved, it changes to set number, which will
// change by one after each wave movement.
foreach(Point point in points)
{
    if(point.X + 1 < n &&
        (map[point.X + 1, point.Y] == '0' ||
         map[point.X + 1, point.Y] > number))
    {
        map[point.X + 1, point.Y] = (char)(number + 1);
        nextPoints.Add(new Point(point.X + 1, point.Y));
    }

    if (point.Y - 1 >= 0 &&
        (map[point.X, point.Y - 1] == '0' ||
         map[point.X, point.Y - 1] > number))
    {
        map[point.X, point.Y - 1] = (char)(number + 1);
        nextPoints.Add(new Point(point.X, point.Y - 1));
    }

    if (point.X - 1 >= 0 &&
        (map[point.X - 1, point.Y] == '0' ||
         map[point.X - 1, point.Y] > number))
    {
        map[point.X - 1, point.Y] = (char)(number + 1);
        nextPoints.Add(new Point(point.X - 1, point.Y));
    }

    if (point.Y + 1 < m &&
        (map[point.X, point.Y + 1] == '0' ||
         map[point.X, point.Y + 1] > number))
    {
        map[point.X, point.Y + 1] = (char)(number + 1);
        nextPoints.Add(new Point(point.X, point.Y + 1));
    }
}

// Calls same method with next set of points in a wave and a
// one added to number to indicate that it is the next wave
Path(map, nextPoints, end, number + 1, n, m);
}

/// <summary>
/// Displays answers
/// </summary>
/// <param name="friends"> List of all of friends coordinates </param>
/// <param name="route"> Most optimal route </param>
/// <param name="n"> Map's height </param>
void DisplayAnswers(List<Point> friends, Route route, int n)
{
    foreach(Point point in friends)
    {
        point.TransformPoint(n);
    }
    friends = SortPointList(friends);

    // Puts each friends coordinates

```

```

foreach (Point point in friends)
{
    TableCell cell = new TableCell();
    TableRow row = new TableRow();
    cell.Text = point.X + " " + point.Y;
    row.Cells.Add(cell);
    Table1.Rows.Add(row);
}

// Puts answer if none of the routes work
if (route.PathValue <= 0)
{
    TableCell cell = new TableCell();
    TableRow row = new TableRow();
    cell.Text = "Neįmanoma";
    row.Cells.Add(cell);
    Table1.Rows.Add(row);
}
// Puts answer for the best route
else
{
    TableCell cell = new TableCell();
    TableRow row = new TableRow();

    route.Meet.TransformPoint(n);
    route.Pizza.TransformPoint(n);

    cell.Text = "Susitikimo vieta " +
        route.Meet.X + " " +
        route.Meet.Y;
    row.Cells.Add(cell);
    Table1.Rows.Add(row);

    TableCell cell1 = new TableCell();
    TableRow row1 = new TableRow();
    cell1.Text = "Picerija " +
        route.Pizza.X + " " +
        route.Pizza.Y;
    row1.Cells.Add(cell1);
    Table1.Rows.Add(row1);

    TableCell cell2 = new TableCell();
    TableRow row2 = new TableRow();
    cell2.Text = "Nueita " + route.PathValue;
    row2.Cells.Add(cell2);
    Table1.Rows.Add(row2);
}
}

/// <summary>
/// Sorts Point list.
/// </summary>
/// <param name="points"> List of points </param>
/// <returns> Sorted list of points </returns>
List<Point> SortPointList(List<Point> points)
{
    for (int i = 0; i < points.Count - 1; i++)
    {
        for (int j = i + 1; j < points.Count; j++)
        {
            if (points[i].X > points[j].X ||
                (points[i].X == points[j].X &&
                 points[i].Y > points[j].Y))
            {
                Point temp = points[i];
                points[i] = points[j];
            }
        }
    }
}

```

```

        points[j] = temp;
    }
}
return points;
}

/// <summary>
/// Prints map
/// </summary>
/// <param name="map"> Char array of map elements </param>
/// <param name="n"> Maps height </param>
/// <param name="m"> Map width </param>
void PutMap(char[,] map, int n, int m)
{
    for (int i = 0; i < n; i++)
    {
        TableRow row = new TableRow();
        {
            TableCell cell = new TableCell();
            cell.Width = 20;
            cell.Height = 20;
            cell.Text = (n - i).ToString();
            cell.VerticalAlign = VerticalAlign.Middle;
            cell.HorizontalAlign = HorizontalAlign.Center;
            row.Cells.Add(cell);
        }
        for (int j = 0; j < m; j++)
        {
            TableCell cell = new TableCell();
            cell.Width = 20;
            cell.Height = 20;
            cell.Text = map[i, j].ToString();
            cell.VerticalAlign = VerticalAlign.Middle;
            cell.HorizontalAlign = HorizontalAlign.Center;
            row.Cells.Add(cell);
        }
        Table2.Rows.Add(row);
    }

    TableCell cell0 = new TableCell();
    TableRow row0 = new TableRow();
    cell0.Text = "";
    cell0.Width = 20;
    cell0.Height = 20;
    cell0.VerticalAlign = VerticalAlign.Middle;
    cell0.HorizontalAlign = HorizontalAlign.Center;
    row0.Cells.Add(cell0);
    for (int i = 0; i < m; i++)
    {
        TableCell cell = new TableCell();
        cell.Text = (i + 1).ToString();
        cell.Width = 20;
        cell.Height = 20;
        cell.VerticalAlign = VerticalAlign.Middle;
        cell.HorizontalAlign = HorizontalAlign.Center;
        row0.Cells.Add(cell);
    }
    Table2.Rows.Add(row0);

    Label1.Text = "P - picerija";
    Label2.Text = "D - vieno iš draugų pradinė vieta";
    Label3.Text = "S - galima susitikimo vieta";
    Label4.Text = ". - praeinamas langelis";
}
}

```


}

1.7. Pradiniai duomenys ir rezultatai

Pirmas testas:

U3.txt:

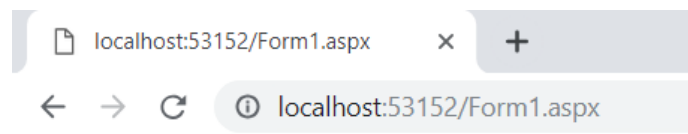
```
5 5
P.X.P
..XS.
S.X..
..X.D
D.X..
```

ResultFile.txt:

```
-----
|P|. |X|. |P|
-----
|. |. |X|S|. |
-----
|S|. |X|. |. |
-----
|. |. |X|. |D|
-----
|D|. |X|. |. |
-----
```

```
1 1
5 2
Neįmanoma
```

Rezultatai, spausdinami į ekraną:



Gauti atsakymus!

```
1 1
5 2
Neįmanoma
```

5	P	.	X	.	P
4	.	.	X	S	.
3	S	.	X	.	.
2	.	.	X	.	D
1	D	.	X	.	.
	1	2	3	4	5

P - picerija
D - vieno iš draugų pradinė vieta
S - galima susitikimo vieta
.- praeinamas langelis

Antras testas:

U3.txt:

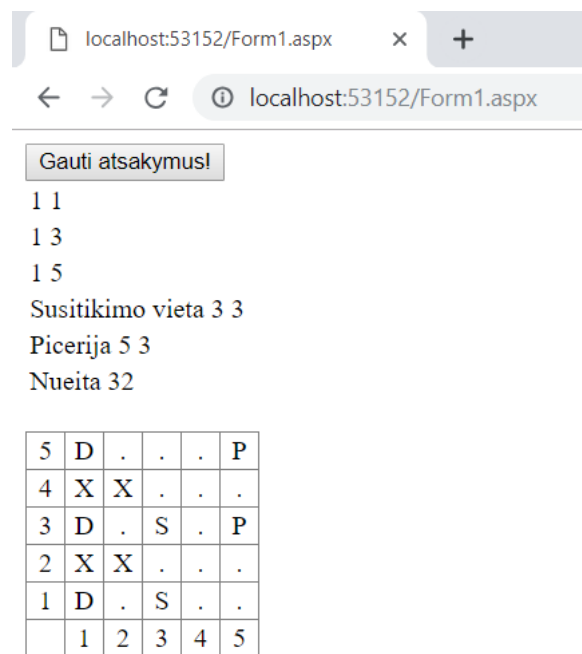
```
5 5
D...P
XX...
D.S.P
XX...
D.S..
```

ResultFile.txt:

```
-----
|D|.|.|.P|
-----
|X|X|.|.|.|
-----
|D|.S|.P|
-----
|X|X|.|.|.|
-----
|D|.S|.|.|
-----
```

```
1 1
1 3
1 5
Susitikimo vieta 3 3
Picerija 5 3
Nueita 32
```

Rezultatai, spausdinami į ekraną:



Gauti atsakymus!

```
1 1
1 3
1 5
Susitikimo vieta 3 3
Picerija 5 3
Nueita 32
```

5	D	.	.	.	P
4	X	X	.	.	.
3	D	.	S	.	P
2	X	X	.	.	.
1	D	.	S	.	.
	1	2	3	4	5

P - picerija
D - vieno iš draugų pradinė vieta
S - galima susitikimo vieta
.- praeinamas langelis

1.8. Dėstytojo pastabos

2. Dinaminis atminties valdymas (L2)

2.1. Darbo užduotis

2.2. Grafinės vartotojo sąsajos schema

2.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

2.4. Klasių diagrama

2.5. Programos vartotojo vadovas

2.6. Programos tekstas

2.7. Pradiniai duomenys ir rezultatai

2.8. Dėstytojo pastabos

3. Bendrinės klasės ir sąsajos (L3)

3.1. Darbo užduotis

3.2. Grafinės vartotojo sąsajos schema

3.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

3.4. Klasių diagrama

3.5. Programos vartotojo vadovas

3.6. Programos tekstas

3.7. Pradiniai duomenys ir rezultatai

3.8. Dėstytojo pastabos

4. Kolekcijos ir išimčių valdymas (L4)

4.1. Darbo užduotis

4.2. Grafinės vartotojo sąsajos schema

4.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

4.4. Klasų diagrama

4.5. Programos vartotojo vadovas

4.6. Programos tekstas

4.7. Pradiniai duomenys ir rezultatai

4.8. Dėstytojo pastabos

5. Deklaratyvusis programavimas (L5)

5.1. Darbo užduotis

5.2. Grafinės vartotojo sąsajos schema

5.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

5.4. Klasių diagrama

5.5. Programos vartotojo vadovas

5.6. Programos tekstas

5.7. Pradiniai duomenys ir rezultatai

5.8. Dėstytojo pastabos