Universitatea Politehnica Timișoara  
Facultatea de Automatică și Calculatoare  
Ingineria Sistemelor

Programare Concurentă

Proiectul nr. 9

***Semaforizarea unui traseu   
în undă verde***

An universitar 2017-2018

Semestrul II

*Cuprins*

1. *Introducere ............................................................................................ pag 3*
2. *Tehnologii utilizate ............................................................................... pag 5*
3. *Specificatiile aplicatiei ......................................................................... pag 6*
4. *Implementare ....................................................................................... pag 9*
5. *Utilizarea sistemului ........................................................................... pag 42*
6. *Testarea aplicatiei ............................................................................... pag 44*
7. *Concluzii ............................................................................................. pag 46*
8. *Bibliografie ......................................................................................... pag 50*
9. *INTRODUCERE*

* ***Tema proiectului***

Realizarea unei aplicatii care sa asigure deplasarea in unda verde a tuturor masinilor care pornesc dintr-un capat al traseului stabilit.

Se urmărește bulevardul *Liviu Rebreanu*, porțiunea dintre Calea Martirilor și Calea Șagului. Acesta are câte două benzi pe sens, separate de linia de tramvai, cu două sensuri, care are semafoare proprii.

În intersecțiile cu bulevardele mari, sensul de mers se trifurcă, asigurând două benzi de mers înainte (cea de-a doua fiind și pentru virat la stânga) și o bandă separată pentru a vira la dreapta(cu semafor intermitent).

În ceea ce privește celelalte intersecții, acestea au două benzi pe sens, ambele fiind și pentru viraje, acolo unde este permis.

Totodată, intersecțiile cu bulevardele mari dispun de senzori de trafic, câte unul pe fiecare sens; prin activarea acestui senzor (însemnând număr mare de mașini), timpul de trecere pe acel sens se prelungește cu încă 10 secunde față de cel stabilit inițial. De asemenea, aceste intersecții sunt dotate cu camere video pentru înregistrarea traficului. Mașinile ce forțează intersecția, trecând pe roșu vor fi salvate într-o evidență pentru a li se întocmai proces verbal de constatare a contravenției.

* ***Semaforizarea inteligenta***

Sistemele de reglare a circulatiei reprezinta ansambluri functionale caracterizate prin existenta unor puncte mobile sau fixe, dispersate in spatiu in care se efectueaza schimburi de informatie in vederea realizarii unui transport in timp util, comod, ritmic si in conditii impuse de siguranta pentru procesul de circulatie. Traficul este caracterizat in primul rand prin viteza de circulatie si prin densitatea vehicolelor. Totodata marimea traficului este in stransa legatura cu dezvoltarea economica a teritoriului respectiv.

In conditiile dezvoltarii in ritm sustinut a economiei pe intreg teritoriul tarii se impune necesitatea unui progres tehnic mai rapid, a modernizarii radicale a productiei si tehnologiilor.

Proiectare si realizarea arterelor de circulatie presupune si anticiparea faptului ca odata cu dezvoltarea economica a tarii respective vor aparea aglomerari urbane, care la randul lor se vor reflecta in modul de organizare si desfasurare a deplasarii persoanelor, a autovehiculelor rutiere. Astfel un loc important in coordonarea dirijarii traficului il au instalatiile de semaforizare.

Semaforizarea unei intersectii stradale nu reprezinta decat un aspect izolat al problemei dirijarii automate a traficului urban. Indiferent de corectitudinea proiectarii si instalrii unei instalatii de semaforizare intr-o intersectie si de promptitudinea si rigurozitatea cu care programele de functionare a semafoarelor raspund la schimbarile permanente ale traficului, daca diferitele instalatii de semaforizare, aferente intersectiilor din acelasi teritoriu urban, prezinta un mod de functionare propriu, circulatia din zona respectiva va avea un caracter intimplator.

Se impune o coordonare a modului de functionare a semafoarelor, succesiunea proprie a fazelor in cadrul aceluiasi ciclu, momentele precise pentru o anumita intersectie, dar intamplatoare la nivel de zona, la care este servita o anumita faza de circulatie, conduc la un caracter cu totul intamplator al desfasurarii circulatiei din zona urbana respectiva.

Aceasta impune o coordonare a modului de functionare a diferitelor instalatii de semaforizare, la nivelul unei zone urbane sau al unui intreg oras. Avantajele unor asemenea tipuri de coordonari sunt legate de :

- fluidizarea circulatiei de-a lungul traseelor considerate ;

- realizarea unor importante economii de timp si combustibil ;

- diminuarea poluarii.

In mod practic aceasta coordonare se realizeaza, functie de cerintele majore ale traficului, pentru urmatoarele genuri de situatii :

- de-a lungul unui traseu liniar (coordonare de tip unda verde) ;

- in cadrul unei zone urbane(coordonare de suprafata).

Cand doua sau mai multe intersectii se gasesc in adiacenta pe aceeasi artera principala de circulatie ,sunt necesare anumite forme de coordonare a dirjarii circulatiei, pentru a reduce intarzierile si a preveni caracterul de continuitate al opririlor.

In cele ce urmeaza vom descrie cateva din sistemele utilizate pentru coordonarea instalatiilor de semaforizare asociate unui sistem liniar de dirijare a traficului, in special sistemul „unda verde”.

Sistemul „***unda verde***” este de fapt un dispozitiv pentru dirijarea traficului vehiculelor pe o artera de circulatie in unda verde cu ajutorul semafoarelor, astfel incat un vehicul sa poata parcurge aceasta artera fara sa opreasca. Acest sistem este obtinut prin integrarea functionala a tuturor echipamentelor amplasate in intersectiile succesive ale unei magistrale urbane cu un numar mare de intersectii si ccu trafic intens.

In acest mod un anumit vehicul care circula cu o anumita viteza indicata,va gasi succesiv culoarea verde la semafoarele S1,S2,S3,....Pentru aceasta se folosesc semafoarele de sincronizare prin intermediul carora se impune o functionare secventiala a posturilor din intersectii,spre a permite accesul vehiculelor si din arterele secundare adiacente(Sa,Sb,Sc,...).

Scopul acestui sistem consta in asigurarea tercerii unui volum maxim de trafic, de-a lungul traseului coordonat, fara oprire, in inlaturarea cozilor de asteptare ale vehiculelor la intersectie.

1. *Tehnologii utilizate*

Aplicatia a fost realizata in C# (C Sharp), folosind mediul de dezvoltare Visul Studio.

Am decis sa folosim C# deoarece acest limbaj prezinta diverse avantaje:

* **C#** este un limbaj de programare orientat pe obiecte, simplu, modern, ce deriva din C++ si Java.
* **Simplitatea** limbajului este data de absenta pointerilor, mostenirea facilitatilor de „garbage collection” si managementul automat al memoriei datorita integrarii pe platforma .NET, iar operatiile nesigure precum manipularea directa a memoriei nu sunt premise.
* **Modernitatea** limbajului: C# este conceput pentru construirea de operatii scalabile, robuste si interoperabile. Include un suport “built in” pentru transformarea componentelor in serviciu web pentru un acces mai usor.
* **Orientat pe obiecte**: suporta incapsularea datelor. Interfetele, mostenirea si polimorfismul. A introdus structurile (structs) pentru a permite tipurilor primitive sa devina obiecte.
* **Portabilitate**: Codul C# poate rula pe mai mult de 2,2 miliarde de dispozitive cu Windows, iOS, Android, Linux. C# este portabil intr-o gama larga de medii cum ar fi telefonia mobila, Embedded, desktop si servere de calcul.

**Microsoft Visual Studio** este un mediu de dezvoltare integrat (integrated development environment - IDE) de la Microsoft. Acesta poate fi folosit pentru a dezvolta aplicaţii consolă şi aplicații cu interfaţă grafică pentru toate platformele suportate de Microsoft Windows (ex. .NET Framework, Windows Mobile etc).

Acesta oferă editor, compilator/debugger și mediu de proiectare (designer) pentru mai multe limbaje de programare. Limbaje de programare incluse: Microsoft Visual C++, Microsoft Visual C#, Microsoft Visual Basic, Microsoft Visual Web Developer, Team Foundation Server.

.NET Framework este o platformă/cadru (Framework) de dezvoltare software unitară, înzestrată cu un set mare de clase, structuri, enumerări etc., organizate într-un set de spaţii de nume (namespaces) bazate pe un limbaj comun. Tehnologia .NET pune laolaltă mai multe tehnologii şi limbaje de programare (ex.: C++, C#) asigurând, totodată, atât portabilitatea codului compilat între diferite calculatoare cu sistem Windows, cât şi reutilizarea codului în programe, indiferent de limbajul de programare utilizat.

Pentru a ne usura munca si pentru a avea fiecare acces la datele proiectului si munca fiecarui coleg am folosit **Git Hub**. Acesta permite web developerilor să colaboreze mult mai eficient, întrucât pot descărca oricând cea mai recentă versiune a software-ului în cauză, îi pot face modificări și pot încărca ulterior versiunea actualizată. Fiecare dezvoltator web poate vedea aceste modificări și își poate aduce la rândul său contribuția.

1. *Specificatiile aplicatiei*

* ***Prezentarea functiilor***

Pentru a marii timpul culorii verzi la semafor folosim urmatoarea functie :

public void increaseGreenTime()

{

\_greenWaitTime += 2;

}

Pentru a micsora timpul culorii verzi a unui semafor folosim :

public void decreaseGreenTime()

{

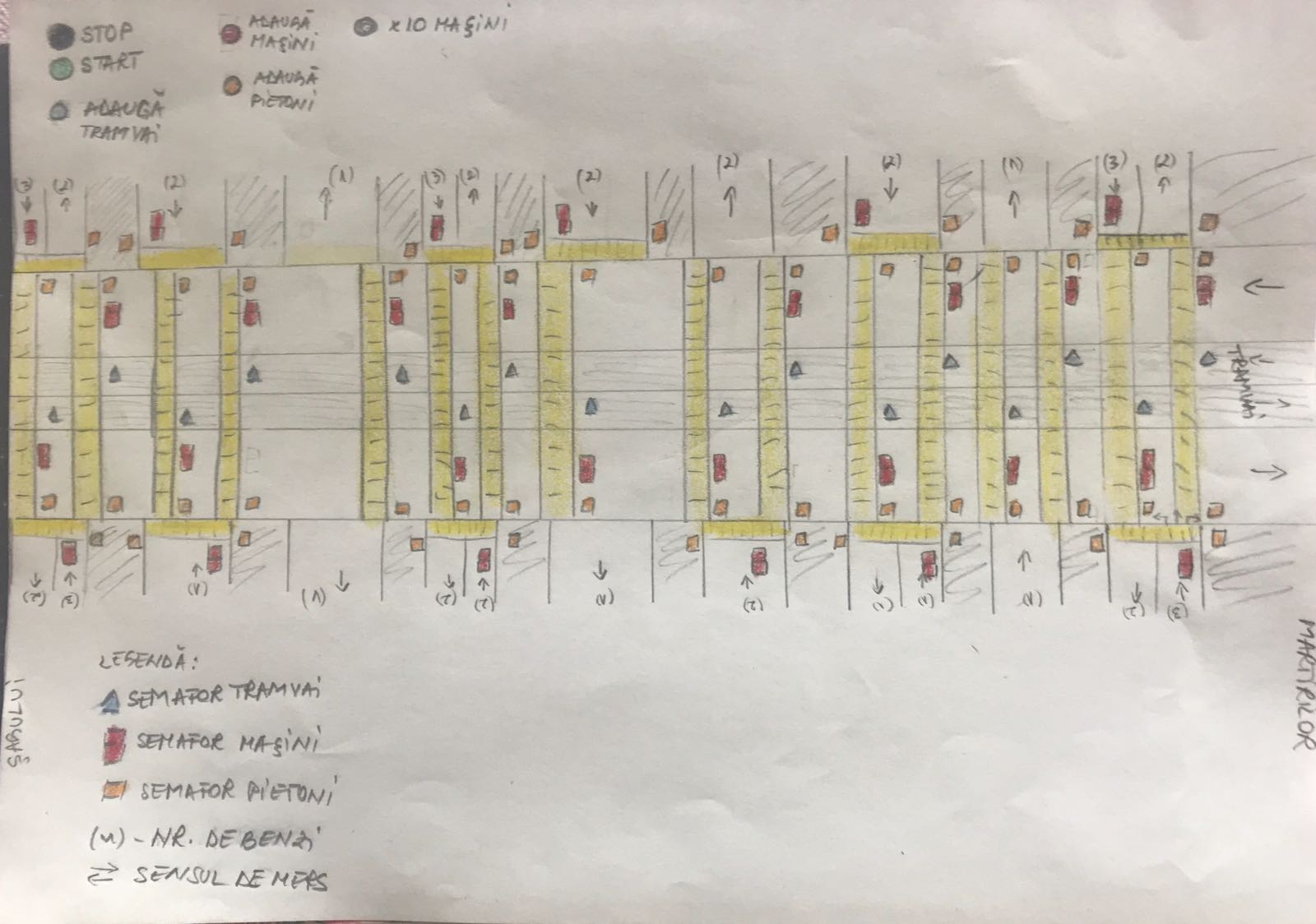
\_greenWaitTime -= 2;

}

Pentru a aprinde culoarea rosie sau verde a semaforului folosim o functie, in cadrul careia folosim un task ce asigura functionarea independenta a fiecarui semafor.

Functia „public void StartIntersectionSync” porneste sincronizarea semafoarelor din cadrul intersectiei, in cadrul ei este implementata o lista de task-uri ce retine task-ul ce porneste un semafor si tot in cadrul functiei se realizeaza reguli de sincronizare intre semafoare.

Folosim o functie de adaugare a masinutei in fereastra principala , dar si un task ce asigura crearea in paralel a tuturo masinutelor .



Imaginea a fost realizata la inceputl aplicatiei, mai apoi fiind folosita ca o schita pentru realizarea designului pe PC.

Alaturi de poza au fost atasate cateva informatii despre cum ar trebui sa functioneze sitemul de semaforizare unde sunt testate urmatoarele cazuri :

„La apasarea butonului start semafoarele vor functiona in unda verde indiferent daca exista vehicule sau nu.

Ulterior prin apasarea butoanelor Adauga tramvai, masini, pietoni se vor introduce si aceste elemente pentru observarea mai buna a functionalitatii.

Butonul x10 masini va înmulti numarul de masini cu 10 pentru a observa cum schimbarea acestui numar influenteaza timpul in care semafoarele au culoarea verde.”

In continuare vom prezenta varianta finala a aplicatiei, urmata de schema logica:



NU

NU

NU

NU

DA

DA

DA

DA

DA

Au fost adaugate elemente pentru simulare?

Sunt eliminate elementele de trafic

Intersectie aglomerata?

Durata de trecere a masinilor creste cu 10 secunde

A fost apasat butonul ‘Clear’?

Maisinile beneficiaza de Unda Verde

Simulare in desfasurare

Sistem pornit

A fost activata sincronizarea semafoarelor?

A fost activata Unda Verde?

Au fost activati senzorii?

Se monitorizeaza Numarul de masini din fiecare intersectie

NU

DA

1. *Implementare*

SpeedPanel

class SpeedPanel

{

MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referinta la fereastra principala

public int \_positionFromTop { get; set; } // pozitii fata de fereastra

public int \_positionFromRight { get; set; }

public int speed { get; set; } // viteza curenta citita de senzor

public int \_indexIntersectie { get; set; } // indexul intersectiei unde se afla senzorul

public int \_indexSem { get; set; }

public Canvas canv;

public Rectangle speedBoard; // panoul unde se afiseaza viteza

public Label speedLabel;

SolidColorBrush whiteBrush = new SolidColorBrush(); // culoare panoului

// Constructor clasa speedpanel

public SpeedPanel(int positionFromTop, int positionFromRight, int indexIntersectie, int indexSem)

{

whiteBrush.Color = Color.FromArgb(150,255,255,255);

speed = 0;

\_positionFromTop = positionFromTop;

\_positionFromRight = positionFromRight;

\_indexIntersectie = indexIntersectie;

\_indexSem = indexSem;

canv = new Canvas();

speedBoard = new Rectangle()

{

Width = 70,

Height = 30,

Fill = whiteBrush

};

speedLabel = new Label()

{

Content = (speed + "pixel/s").ToString(), // afisare viteza in interfata

};

Canvas.SetTop(speedBoard, \_positionFromTop);

Canvas.SetRight(speedBoard, positionFromRight);

Canvas.SetTop(speedLabel, \_positionFromTop);

Canvas.SetRight(speedLabel, positionFromRight);

canv.Children.Add(speedBoard); // se adauga panelul la canvas

canv.Children.Add(speedLabel); // se adauga labelul la canvas

mainWin.mapGrid.Children.Add(canv); // se adauga canvasul la fereastra principala

}

}

Animation

public class Animation

{

MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referinta catre fereastra principala

public Storyboard story = new Storyboard(); // Obiect ce retine o anumita animatie

private Point \_startPoint; // punct de pornire al animatiei

private Point \_endPoint; // punct de oprire al animatiei

private int \_additionalAnims; // optiuni ptr animatie (urmarirea tangentei)

// Constructor clasa Animatie

public Animation(Point startPoint,Point endPoint, int additionalAnims)

{

\_startPoint = startPoint;

\_endPoint = endPoint;

\_additionalAnims = additionalAnims;

}

/\* Functie care porneste o animatie a unei anumite masinute care dureaza un anumit timp\*/

public void startAnimation(Car \_animateObject, int animationTime, int delay)

{

// Animatia este inserata in coada dispecerului threadului de management al interfetei si este pornita

Dispatcher.CurrentDispatcher.BeginInvoke(new Action(async () =>

{

NameScope.SetNameScope(mainWin, new NameScope()); // este setat contextul animatiei catre fereastra principala

MatrixTransform carTransform = new MatrixTransform(); // creare obiect ce va retine transformarile vizuale din timpul animatiei asupra masinutei

\_animateObject.\_carImg.RenderTransform = carTransform; // atasare masinuta la matricea de transformare

mainWin.RegisterName("carTransform", carTransform);

PathGeometry animPath = new PathGeometry(); // creare geometrie animatie

PathFigure pathFigure = new PathFigure();

pathFigure.StartPoint = \_startPoint;

pathFigure.Segments.Add(new LineSegment(\_endPoint, true));

animPath.Figures.Add(pathFigure);

animPath.Freeze(); // optimizare animatie

MatrixAnimationUsingPath mAnim = new MatrixAnimationUsingPath(); // creare obiect animatie

if (\_additionalAnims == 1)

mAnim.DoesRotateWithTangent = true;

mAnim.PathGeometry = animPath; // atasare geometrie

mAnim.Duration = TimeSpan.FromSeconds(animationTime); // durata animatie

Storyboard.SetTargetName(mAnim, "carTransform");

Storyboard.SetTargetProperty(mAnim, new PropertyPath(MatrixTransform.MatrixProperty));

story.Children.Add(mAnim);

await Task.Delay(delay);

story.Begin(mainWin, true); // incepere animatie

}));

}

public double speedCalculation(Car car)

{

return (Math.Sqrt(Math.Pow(0.01 \* \_endPoint.X - 0.01 \* \_startPoint.X, 2) + Math.Pow(0.01 \* \_endPoint.Y - 0.01 \* \_startPoint.Y, 2)) \* 5000 / Math.Pow(car.\_speed,1.5f));

}

}

MainWindows

public class credits

{

public int nr { get; set; }

public string Name { get; set; }

}

// clasa folosita pentru stocarea elementelor selectate de utilizator din meniu

public class selectedItem

{

public string carType { get; set; }

public string path { get; set; }

public bool isBad { get; set; }

public int speed { get; set; }

}

// clasa folosita pentru stocarea Pathului si semafoarelor intalnite de o masinuta intr-o anumita animatie

public class SelectedPath

{

public List<Animation> Anims;

public List<ints> intersectionAndSemType;

public int posFromTop;

public int posFromRight;

public SelectedPath(List<Animation> anm, List<ints> ints, int posT, int posR)

{

Anims = anm;

intersectionAndSemType = ints;

posFromTop = posT;

posFromRight = posR;

}

}

public partial class MainWindow : Window

{

// Coordonate Semafoare

private readonly List<Point> \_intersection1SemPoints = new List<Point>() {

new Point (40, 85),

new Point (5, 235),

new Point (230, 240 ),

new Point (235, 90 ),

};

private readonly List<Point> \_intersection2SemPoints = new List<Point>() {

new Point (10, 455 ),

new Point (35, 635 ),

new Point (230, 645 ),

new Point (235, 465 ),

new Point (120,600),

new Point (140,600)

};

private readonly List<Point> \_intersection3SemPoints = new List<Point>() {

new Point (40, 980 ),

new Point (35, 1130 ),

new Point (230, 1135 ),

new Point (240, 987 ),

};

// Liste cu optiunile ce pot fi selectate de utilizator

private List<string> AvailableVehicles = new List<string>() {"Grey car", "Red car", "Train"};

private List<string> AvailablePaths = new List<string>()

{

"L.Rebreanu Mart B1 --> L.Rebreanu Sag B1", //0

"L.Rebreanu Mart B2 --> L.Rebreanu Sag B2", //1

"L.Rebreanu Mart B3 --> Drubeta 1", //2

"L.Rebreanu Mart B2 --> C-tin Brancoveanu 1", //3

"L.Rebreanu Sag B1 --> L.Rebreanu Mart B1", //4

"L.Rebreanu Sag B2 --> L.Rebreanu Mart B2", //5

"L.Rebreanu Sag B1 --> Drubeta 1", //6

"Drubeta 1 --> Drubeta 2", //7

"Linie Tramvai Mart --> Linie Tramvai Sag", //8

"Linie Tramvai Sag --> Linie Tramvai Mart", //9

};

private List<Intersection> Intersections = new List<Intersection>();

private ObservableCollection<Sensor> Sensors = new ObservableCollection<Sensor>();

private List<SpeedPanel> speedPanels = new List<SpeedPanel>();

private char name = 'A';

// Current Scenario data structures

private List<Car> carsList = new List<Car>();

private List<Task> listOfTasks = new List<Task>();

ObservableCollection<Car> listOfBadCars = new ObservableCollection<Car>();

ObservableCollection<selectedItem> selectedThings = new ObservableCollection<selectedItem>();

CancellationTokenSource tokenSource = new CancellationTokenSource(); // token folosit pentru oprirea unui anumit Task

public MainWindow()

{

InitializeComponent();

List<credits> credits = new List<credits>();

string names = "Andreea Carp @AndreeaCamelia, Andrei Chirap @AndreiChirap, Adrian-Gabriel Balanescu @adrianB3, Gabriel Bizdoc @GabiBVG ,Raul-Adrian Chincea @RaulChincea, Diana Dalea @dianadalea,Andreea Balasoiu @AndreeaBalasoiu, Alina Bacalete @AlinaBacalete, Voicu Carole @carolevoicu, Anamaria Larisa Bala @AnamariaLarisa, Simona-Rebeca Buse @SimonaRebeca, Adrian Coneac @adrianconeac, Mario-Razvan Cioara @MarioCioara, Raluca-Andreea Cozma @ralucacozma, Raul Cojocaru @raulcojocaru, Robert Burdusel @robertb21";

var n = names.Split(',');

int index = 1;

foreach (string item in n)

{

credits.Add(new credits() { Name = item , nr = index++});

}

theCreators.ItemsSource = credits;

Intersections.Add(new Intersection(\_intersection1SemPoints));

Intersections.Add(new Intersection(\_intersection2SemPoints));

Intersections.Add(new Intersection(\_intersection3SemPoints));

Sensors.Add(new Sensor("Sensor1",0,0,90,155));

Sensors.Add(new Sensor("Sensor2", 1,0,450,155));

Sensors.Add(new Sensor("Sensor3", 2,0,980,155));

Sensors.Add(new Sensor("Sensor4", 2,2,1150,250));

Sensors.Add(new Sensor("Sensor5", 1,2,650,250));

Sensors.Add(new Sensor("Sensor6", 0,2,300,250));

speedPanels.Add(new SpeedPanel(60,60,0,0));

speedPanels.Add(new SpeedPanel(35,420,1,0));

speedPanels.Add(new SpeedPanel(340,1150,2,2));

speedPanels.Add(new SpeedPanel(300,690,1,2));

AvailableCarTypes.ItemsSource = AvailableVehicles;

AvailablePathsListBox.ItemsSource = AvailablePaths;

selectedItemsListView.ItemsSource = selectedThings;

listBoxOfBadCars.ItemsSource = listOfBadCars;

}

// Event apelat la inchiderea ferestri

private void ApplicationExit(object sender, EventArgs e)

{

Application.Current.Shutdown();

}

// Event apelat la apasarea butonului "Start Simulation"

private void startAnimation(object sender, RoutedEventArgs e)

{

// Creare task pentru fiecare masinuta

foreach (var car in carsList)

{

if (tokenSource.IsCancellationRequested)

{

break;

}

car.createImage();

// Lista de taskuri care asigura deplasarea unei masinute in mod paralel fata de celelalte masinute

listOfTasks.Add(new Task(async () =>

{

int i = 0;

foreach (var animation in car.\_animationsList)

{

// Declansare secventa de animatii pentru fiecare masinuta

animation.startAnimation(car, (int)animation.speedCalculation(car), 0);

Parallel.ForEach(speedPanels, (speedPanel) =>

{

if (speedPanel.\_indexIntersectie == car.intSem[i].intersection &&

speedPanel.\_indexSem == car.intSem[i].semType

)

speedPanel.speed = (int) car.\_speed;

});

Parallel.ForEach(Sensors, (sensor) =>

{

if (sensor.\_isActivated)

{

if (sensor.\_indexIntersectie == car.intSem[i].intersection &&

sensor.\_indexSemafor == car.intSem[i].semType

)

{

sensor.\_Signal();

}

}

});

if (Sensors[0].\_isCrowded() && Sensors[3].\_isCrowded())

{

ActivateGreenWave(new object(), new RoutedEventArgs());

}

await Task.Delay((int)animation.speedCalculation(car)\*1000);

// daca semaforul la care a ajuns masinuta la un anumit moment este rosu, taskul asteapta ca acel semafor sa devina verde

while (Intersections[car.intSem[i].intersection].\_TrafficLights[car.intSem[i].semType].isRed() && car.\_isABadCar == false)

{

await Task.Delay(100);

}

foreach (var sensor in Sensors)

{

if (sensor.\_indexIntersectie == car.intSem[i].intersection &&

sensor.\_indexSemafor == car.intSem[i].semType

)

{

sensor.\_Reset();

}

}

foreach (var speedPanel in speedPanels)

{

if (speedPanel.\_indexIntersectie == car.intSem[i].intersection &&

speedPanel.\_indexSem == car.intSem[i].semType

)

speedPanel.speed = 0;

}

if (i < car.intSem.Count - 1)

{

// daca o anumita masinuta trece pe rosu este adaugata in lista masinutelor ce au trecut pe rosu

if (car.\_isABadCar && Intersections[car.intSem[i].intersection].\_TrafficLights[car.intSem[i].semType].isRed())

{

listOfBadCars.Add(car);

}

i++;

}

}

},tokenSource.Token));

}

// pornire taskuri ptr fiecare masinuta

foreach (var task in listOfTasks)

{

task.Start(TaskScheduler.FromCurrentSynchronizationContext());

}

Task.Factory.StartNew(async () =>

{

while (true)

{

ProgressBar1.Value = Sensors[0].\_numberOfCars;

label1.Content = Sensors[0].\_numberOfCars;

ProgressBar2.Value = Sensors[1].\_numberOfCars;

label2.Content = Sensors[1].\_numberOfCars;

ProgressBar3.Value = Sensors[2].\_numberOfCars;

label3.Content = Sensors[2].\_numberOfCars;

ProgressBar4.Value = Sensors[3].\_numberOfCars;

label4.Content = Sensors[3].\_numberOfCars;

ProgressBar5.Value = Sensors[4].\_numberOfCars;

label5.Content = Sensors[4].\_numberOfCars;

ProgressBar6.Value = Sensors[5].\_numberOfCars;

label6.Content = Sensors[5].\_numberOfCars;

await Task.Delay(100);

}

},tokenSource.Token, TaskCreationOptions.None, TaskScheduler.FromCurrentSynchronizationContext());

Task.Factory.StartNew(async () =>

{

while (true)

{

foreach (var speedPanel in speedPanels)

{

speedPanel.speedLabel.Content = speedPanel.speed.ToString();

}

await Task.Delay(100);

}

}, tokenSource.Token, TaskCreationOptions.None, TaskScheduler.FromCurrentSynchronizationContext());

}

private void windowLoaded(object sender, RoutedEventArgs e)

{

}

// Event declansat la apasare butonului de aprindere a semafoarelor

private void StartTrafficLightsSync(object sender, RoutedEventArgs e)

{

// Task ce asigura pornirea sincronizarii fiecarei intersectii in paralel

Task intersectionSyncTask = new Task(async () =>

{

foreach (var intersection in Intersections)

{

intersection.StartIntersectionSync();

await Task.Delay(2000);

}

});

intersectionSyncTask.Start(TaskScheduler.FromCurrentSynchronizationContext());

}

// Event declansat la apasare butonului Add din Meniu prin care este adaugata o noua masinuta in cadrul scenariului de simulare

private void AddCar(object sender, RoutedEventArgs e)

{

// Creare lista de animatii ptr fiecare traseu disponibil

string imgSource = "car.png";

int posT = 160;

int posR = 0;

List<Animation> Animations = new List<Animation>();

Animations.Add(new Animation(new Point(0, 0), new Point(-90, 0),0));

Animations.Add(new Animation(new Point(-90, 0), new Point(-480, 0),0));

Animations.Add(new Animation(new Point(-480, 0), new Point(-985, 0),0));

Animations.Add(new Animation(new Point(-985, 0), new Point(-1200, 0),0));

List<ints> dict = new List<ints>()

{

new ints() {intersection = 0, semType = 0},

new ints() {intersection = 1, semType = 0},

new ints() {intersection = 2, semType = 0},

};

SelectedPath pth = new SelectedPath(Animations, dict, posT, posR);

if (AvailablePathsListBox.SelectedIndex == 0)

{

// L.Rebreanu Mart B1 --> L.Rebreanu Sag B1

posT = 160;

posR = 0;

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

pth = new SelectedPath(Animations, dict, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 1)

{

// L.Rebreanu Mart B2 --> L.Rebreanu Sag B2

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

posT = 140;

posR = 0;

pth = new SelectedPath(Animations, dict, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 2)

{

// L.Rebreanu Mart B3 --> Drubeta 1

posT = 140;

posR = 0;

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

List<Animation> Animations1 = new List<Animation>();

Animations1.Add(new Animation(new Point(0, 0), new Point(-90, 0),0));

Animations1.Add(new Animation(new Point(-90, 0), new Point(-280, 0),0));

Animations1.Add(new Animation(new Point(-280, 0), new Point(-290, -25),0));

Animations1.Add(new Animation(new Point(-290, -5), new Point(-400, -5),1));

Animations1.Add(new Animation(new Point(-400, -5), new Point(-400, -20),1));

Animations1.Add(new Animation(new Point(-500, -20), new Point(-500, -100),1));

List<ints> dict1 = new List<ints>()

{

new ints() {intersection = 0, semType = 0},

new ints() {intersection = 1, semType = 0},

};

pth = new SelectedPath(Animations1, dict1, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 3)

{

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

// L.Rebreanu Mart B2 --> C-tin Brancoveanu 1

posT = 140;

posR = 0;

List<Animation> Animations2 = new List<Animation>();

Animations2.Add(new Animation(new Point(0, 0), new Point(-90, 0), 0));

Animations2.Add(new Animation(new Point(-90, 0), new Point(-470, 0), 0));

Animations2.Add(new Animation(new Point(-470, 0), new Point(-985, 0), 0));

Animations2.Add(new Animation(new Point(-1015, 0), new Point(-1015, -100), 1));

List<ints> dict2 = new List<ints>()

{

new ints() {intersection = 0, semType = 0},

new ints() {intersection = 1, semType = 0},

new ints() {intersection = 2, semType = 0},

};

pth = new SelectedPath(Animations2, dict2, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 4)

{

// L.Rebreanu Sag B1 --> L.Rebreanu Mart B1

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car180.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar180.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

posT = 260;

posR = 1200;

List<Animation> Animations3 = new List<Animation>();

Animations3.Add(new Animation(new Point(0, 0), new Point(60, 0), 0));

Animations3.Add(new Animation(new Point(60, 0), new Point(580, 0), 0));

Animations3.Add(new Animation(new Point(580, 0), new Point(950, 0), 0));

Animations3.Add(new Animation(new Point(950, 0), new Point(1200, 0), 0));

List<ints> dict3 = new List<ints>()

{

new ints() {intersection = 2, semType = 2},

new ints() {intersection = 1, semType = 2},

new ints() {intersection = 0, semType = 2},

};

pth = new SelectedPath(Animations3, dict3, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 5)

{

// L.Rebreanu Sag B2 --> L.Rebreanu Mart B2

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car180.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar180.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

posT = 260;

posR = 1200;

List<Animation> Animations4 = new List<Animation>();

Animations4.Add(new Animation(new Point(0, 0), new Point(60, 0), 1));

Animations4.Add(new Animation(new Point(60, 0), new Point(580, 0), 1));

Animations4.Add(new Animation(new Point(580, 0), new Point(950, 0), 1));

Animations4.Add(new Animation(new Point(950, 0), new Point(1200, 0), 1));

List<ints> dict4 = new List<ints>()

{

new ints() {intersection = 2, semType = 2},

new ints() {intersection = 1, semType = 2},

new ints() {intersection = 0, semType = 2},

};

pth = new SelectedPath(Animations4, dict4, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 6)

{

// L.Rebreanu Sag B1 --> Drubeta 1

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car180.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar180.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

posT = 230;

posR = 1200;

List<Animation> Animations5 = new List<Animation>();

Animations5.Add(new Animation(new Point(0, 0), new Point(60, 0), 0));

Animations5.Add(new Animation(new Point(60, 0), new Point(580, 0), 0));

Animations5.Add(new Animation(new Point(700, 0), new Point(700, -200), 1));

List<ints> dict5 = new List<ints>()

{

new ints() {intersection = 2, semType = 2},

new ints() {intersection = 1, semType = 2},

};

pth = new SelectedPath(Animations5, dict5, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 7)

{

// Drubeta 1 --> Drubeta 2

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car180.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar180.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

posT = 0;

posR = 520;

List<Animation> Animations6 = new List<Animation>();

Animations6.Add(new Animation(new Point(0, 0), new Point(0, 100), 1));

Animations6.Add(new Animation(new Point(0, 100), new Point(0, 500), 1));

List<ints> dict6 = new List<ints>()

{

new ints() {intersection = 1, semType = 1},

};

pth = new SelectedPath(Animations6, dict6, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 8)

{

// Linie Tramvai Mart --> Linie Tramvai Sag

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "train180.png";

}

posT = 180;

posR = 0;

List<Animation> Animations7 = new List<Animation>();

Animations7.Add(new Animation(new Point(0, 0), new Point(-470, 0), 0));

Animations7.Add(new Animation(new Point(-470,0), new Point(-1200, 0), 0));

List<ints> dict7 = new List<ints>()

{

new ints() {intersection = 1, semType = 4},

};

pth = new SelectedPath(Animations7, dict7, posT, posR);

}

if (AvailablePathsListBox.SelectedIndex == 9)

{

// Linie Tramvai Sag --> Linie Tramvai Mart

if (AvailableCarTypes.SelectedItem.ToString() == "Grey car")

{

imgSource = "car.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Red car")

{

imgSource = "redcar.png";

}

if (AvailableCarTypes.SelectedItem.ToString() == "Train")

{

imgSource = "Train.png";

}

posT = 210;

posR = 1150;

List<Animation> Animations8 = new List<Animation>();

Animations8.Add(new Animation(new Point(0, 0), new Point(480, 0), 0));

Animations8.Add(new Animation(new Point(480, 0), new Point(1200, 0), 0));

List<ints> dict8 = new List<ints>()

{

new ints() {intersection = 1, semType = 5},

};

pth = new SelectedPath(Animations8, dict8, posT, posR);

}

selectedThings.Add(new selectedItem()

{

carType = AvailableCarTypes.SelectedItem.ToString(),

path = AvailablePathsListBox.SelectedItem.ToString(),

isBad = isBadCheckBox.IsChecked.Value,

speed = Int32.Parse(speedTextBox.Text)

});

carsList.Add(new Car(pth.intersectionAndSemType,pth.Anims,imgSource,"Vehicle" + name++,45,25,pth.posFromTop, pth.posFromRight, Int32.Parse(speedTextBox.Text), isBadCheckBox.IsChecked.Value));

}

Intersection

class Intersection

{

private MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referinta catre fereastra principala

public List<TrafficLight> \_TrafficLights = new List<TrafficLight>(); // lista de semafoare din cadrul intersectiei

// Constructor clasa Intersectie ce preia coordonatele la care se afla semafoarele

public Intersection(List<Point> coordinates)

{

var coordinates1 = coordinates;

char i = 'a';

string[] directions = new[] {"90left", "inverse", "90right", "normal", "90left", "90right"};

int j = 0;

foreach (var point in coordinates1)

{

\_TrafficLights.Add(new TrafficLight("sem" + i++, (int)point.X, (int)point.Y,3000,directions[j++])); // creare semafor in intersectie

}

}

// Functie ce porneste sincronizarea semafoarelor din cadrul intersectiei

public void StartIntersectionSync()

{

var listOfTasks = new List<Task>(); // lista de Taskuri ce retine taskul ce poreste un semafor

foreach (var trafficLight in \_TrafficLights)

{

listOfTasks.Add(trafficLight.LightUp());

}

// Pornire semafoare, creare reguli de sincronizare intre semafoare

foreach (var tsk in listOfTasks)

{

\_TrafficLights[0].\_color = false;

\_TrafficLights[1].\_color = !\_TrafficLights[0].\_color;

\_TrafficLights[2].\_color = \_TrafficLights[0].\_color;

\_TrafficLights[3].\_color = !\_TrafficLights[0].\_color;

tsk.Start(TaskScheduler.FromCurrentSynchronizationContext()); // pornire task in cadrul threadului de management al interfetei

}

}

}

}

TrafficLights

class TrafficLight

{

MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referinta catre fereastra principala

public bool \_color { get; set; } // culoarea curenta a semaforului

public int \_greenWaitTime { get; set; } // timpul de asteptare pe verde

public int \_redWaitTime { get; set; } // timpul de asteptare pe rosu

public int \_delay { get; set; }

private int \_positionFromTop { get; set; } // pozitie fata de partea de sus a ferestrei

private int \_positionFromRight { get; set; } // pozitie fata de partea de jos a ferestrei

private string \_name { get; set; } // nume semafor

private string \_orientation { get; set; } // orientare semafor (normal, 90 grade la dreapta/stanga, invers)

Canvas canv = new Canvas(); // obiect care va retine imaginea cu semaforul

Ellipse redLight = new Ellipse(); // forma ce va retine culoarea rosie

SolidColorBrush colorBrush = new SolidColorBrush(); // culoarea rosie

Ellipse greenLight = new Ellipse(); // forma ce va retine culoare verde

SolidColorBrush colorBrush1 = new SolidColorBrush(); // culoarea verde

// Constructor semafor

public TrafficLight(string name = "", int positionFromTop = 0, int positionFromRight = 0, int delay = 0,string orientation = "normal", bool color = false, int greenWaitTime = 5, int redWaitTime = 5)

{

\_color = color;

\_greenWaitTime = greenWaitTime;

\_redWaitTime = redWaitTime;

\_delay = delay;

\_positionFromTop = positionFromTop;

\_positionFromRight = positionFromRight;

\_name = name;

\_orientation = orientation;

colorBrush.Color = Color.FromArgb(255, 255, 0, 0);

redLight.Fill = colorBrush;

redLight.Width = 13;

redLight.Height = 13;

colorBrush1.Color = Color.FromArgb(255, 0, 255, 0);

greenLight.Fill = colorBrush1;

greenLight.Width = 13;

greenLight.Height = 13;

BitmapImage semBitmap = new BitmapImage();

semBitmap.BeginInit();//

// pozitionare imagine semafor

if (orientation == "normal")

{

semBitmap.UriSource = new Uri(@"pack://application:,,,/Images/semaphore.png", UriKind.RelativeOrAbsolute);

Canvas.SetRight(redLight, \_positionFromRight + 13);

Canvas.SetTop(redLight, \_positionFromTop + 55);

Canvas.SetRight(greenLight, \_positionFromRight + 13);

Canvas.SetTop(greenLight, \_positionFromTop + 80);

}

if (orientation == "90left")

{

semBitmap.UriSource = new Uri(@"pack://application:,,,/Images/semaphore90l.png", UriKind.RelativeOrAbsolute);

Canvas.SetRight(redLight, \_positionFromRight + 25);

Canvas.SetTop(redLight, \_positionFromTop + 68);

Canvas.SetRight(greenLight, \_positionFromRight);

Canvas.SetTop(greenLight, \_positionFromTop + 68);

}

if (orientation == "90right")

{

semBitmap.UriSource=new Uri(@"pack://application:,,,/Images/semaphore\_90r.png", UriKind.RelativeOrAbsolute);

Canvas.SetRight(redLight, \_positionFromRight);

Canvas.SetTop(redLight, \_positionFromTop + 68);

Canvas.SetRight(greenLight, \_positionFromRight + 25);

Canvas.SetTop(greenLight, \_positionFromTop + 68);

}

if (orientation == "inverse")

{

semBitmap.UriSource = new Uri(@"pack://application:,,,/Images/semaphore\_inverse.png", UriKind.RelativeOrAbsolute);

Canvas.SetRight(redLight, \_positionFromRight + 13);

Canvas.SetTop(redLight, \_positionFromTop + 80);

Canvas.SetRight(greenLight, \_positionFromRight + 13);

Canvas.SetTop(greenLight, \_positionFromTop + 55);

}

semBitmap.EndInit();

Image semImage = new Image

{

Source = semBitmap,

Height = 150,

Width = 40,

Name = \_name

};

Canvas.SetRight(semImage, \_positionFromRight);

Canvas.SetTop(semImage, \_positionFromTop);

canv.Children.Add(semImage);

mainWin.mapGrid.Children.Add(canv); // adaugare semafor la fereastra principala

}

// functie ce mareste timpul de verde

public void increaseGreenTime()

{

\_greenWaitTime += 2;

}

// functie ce micsoreaza timpul de verde

public void decreaseGreenTime()

{

\_greenWaitTime -= 2;

}

public void increaseRedTime()

{

\_redWaitTime += 2;

}

// functie ce micsoreaza timpul de verde

public void decreaseRedTime()

{

\_redWaitTime -= 2;

}

public bool isGreen()

{

return \_color ? true : false;

}

public bool isRed()

{

return \_color ? false : true;

}

// functie ce aprinde culoarea rosie sau verde a semaforului

public Task LightUp()//o actiune ce sse executa asincron

{

// Task ce asigura functionarea independenta a fiecarui semafor

var tsk = new Task(async () =>

{

while (true)

{

canv.Children.Remove(redLight);

canv.Children.Remove(greenLight);

if (isGreen())

{

canv.Children.Add(greenLight);

await Task.Delay(\_greenWaitTime\*1000);

\_color = false;

}

else

{

canv.Children.Add(redLight);

await Task.Delay(\_redWaitTime\*1000);

\_color = true;

}

}

});

return tsk;//returneaza pt a fi activat din clasa intersectie

}

}

}

Sensor

public class Sensor

{

MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referinta catre fereastra principala

public string \_name { get; set; } // id senzor

public int \_numberOfCars { get; set; } // numar curent de masini citite

public int \_indexIntersectie { get; set; } // retine indexul intersectiei unde se afla senzorul

public int \_indexSemafor { get; set; }

public bool \_isActivated { get; set; } // variabila ce retine daca un senzor e activat sau nu

Canvas canv = new Canvas(); // container pentru alte forme

Ellipse blueLight = new Ellipse(); // imaginea senzorului

ImageBrush colorBrush = new ImageBrush(); // culoarea senzorului

// Constructor clasa Senzor

public Sensor(string name,int indexIntersectie, int indexSemafor, int positionFromRight, int positionFromTop)

{

\_isActivated = false;

\_indexIntersectie = indexIntersectie;

\_indexSemafor = indexSemafor;

\_numberOfCars = 0;

\_name = name;

colorBrush.ImageSource = new BitmapImage(new Uri(@"pack://application:,,,/Images/blueLed.png", UriKind.RelativeOrAbsolute));

blueLight.Fill = new SolidColorBrush(Color.FromArgb(100,255,255,255));

blueLight.Width = 15;

blueLight.Height = 15;

Canvas.SetRight(blueLight, positionFromRight);

Canvas.SetTop(blueLight, positionFromTop);

canv.Children.Add(blueLight);

mainWin.mapGrid.Children.Add(canv); // adaugare canvas la fereastra principala

}

// Functie care returneaza daca o intersectie e prea aglomerata

public bool \_isCrowded()

{

return \_numberOfCars > 5 ? true : false;

}

// Functie care incrementeaza nr de masini care trec de senzor

public void \_Signal()

{

\_numberOfCars++;

}

// Functie care reseteaza senzorul

public void \_Reset()

{

\_numberOfCars = 0;

}

// Functie care porneste un senzor

public void startSensor()

{

blueLight.Fill = colorBrush;

\_isActivated = true;

}

// Functie care opreste un senzor

public void stopSensor()

{

blueLight.Fill = new SolidColorBrush(Color.FromArgb(100, 255, 255, 255));

\_isActivated = false;

}

}

}

Car

public struct ints

{

public int intersection;

public int semType;

}

public class Car

{

MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referinta catre fereastra principala

private string \_imgSource { get; set; } // sursa imaginii cu masinuta

public string \_name { get; set; } // nume masinuta

private int \_width { get; set; } // dimensiuni

private int \_height { get; set; }

private int \_positionFromTop { get; set; } // positii in cadrul ferestri

private int \_positionFromRight { get; set; }

public Image \_carImg;

public float \_speed { get; set; } // \_speed is a car statistic that represents the rate at which a car travels across a map. One \_speed point translates to one hundred distance units traveled per second

public List<Animation> \_animationsList; // lista cu animatiile ce vor fi executate de masinuta

public List<ints> intSem; // semafoarele prin care va trece masinuta in timpul animatiilor

private Canvas canv;

public bool \_isABadCar; // var ce va retine daca masinuta incalca legea

// Constuctor clasa masina

public Car(

List<ints> intS, // lista intersectii si semafoare

List<Animation> animationsList, // secventa animatii

string imgSource = "car.png",

string name = "car",

int width = 45,

int height = 25,

int positionFromTop = 0,

int positionFromRight = 0,

float speed = 50,

bool isBad = false

)

{

\_imgSource = imgSource;

\_name = name;

\_width = width;

\_height = height;

\_positionFromTop = positionFromTop;

\_positionFromRight = positionFromRight;

\_speed = speed;

\_carImg = new Image();

intSem = intS;

\_isABadCar = isBad;

BitmapImage carBitmap = new BitmapImage();

carBitmap.BeginInit();

carBitmap.UriSource = new Uri(@"pack://application:,,,/Images/" + \_imgSource, UriKind.RelativeOrAbsolute);

carBitmap.EndInit();

\_carImg.Source = carBitmap;

\_carImg.Width = \_width;

\_carImg.Height = \_height;

\_carImg.Name = \_name;

ToolTip tl = new ToolTip

{

Content = \_name

};

\_carImg.ToolTip = tl;

\_animationsList = animationsList;

canv = new Canvas();

}

// Functie de adaugare a masinutei in fereastra principala

public void createImage()

{

// Task ce asigura crearea in paralel a tuturor masinutelor

Task tsk = new Task(() =>

{

Canvas.SetRight(\_carImg, \_positionFromRight);

Canvas.SetTop(\_carImg, \_positionFromTop);

canv.Children.Add(\_carImg);

mainWin.mapGrid.Children.Add(canv);

});

tsk.Start(TaskScheduler.FromCurrentSynchronizationContext());

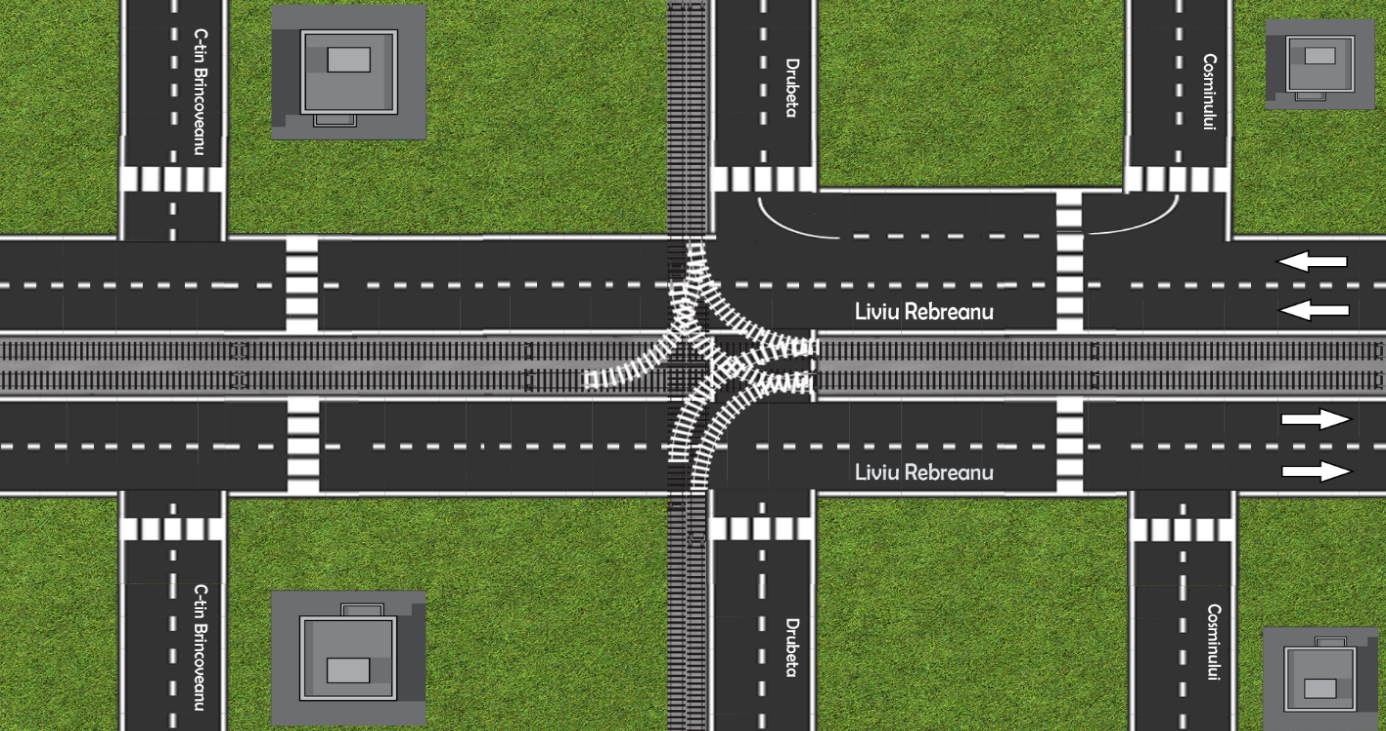
}

1. *Utilizarea sistemului*

Acest sistem simuleaza functionalitatea unui semafor in timp real. Sistemul poate simula mai multe situatii ipotetice ce se pot intampla in viata de zi cu zi in trafic: masurarea traficului, crearea de unda verde, determinarea vitezei autovehiculelor.



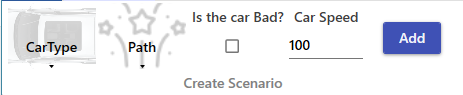
Pentru inceput s-a lucrat in photoshop la design-ul proiectului:



Mai apoi s-a inceput adaugarea de elemente in cadrul proiectului.

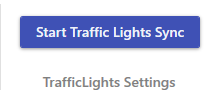
In componenta programului putem gasi urmatoarele elemente:

1. Meniul Create Scenario:



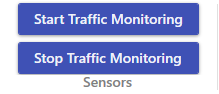
* CarType: Se pot alege diferite autovehicule cat si tramvaie care sa fie introduce in simulare;
* Path: Selecteaza traseul pe care il va urma masina;
* Starea masinii;
* Viteza masinii.

1. Meniul TrafficLights Settings :



* Start Traffic Lights Sync: Acest buton porneste sistemul de semaforizare.

1. Meniul Sensors:



* Start Traffic Monitoring: Porneste monitorizarea traficului(in cazul unui numar mare de masini se mareste timpul pe culoarea verde);
* Stop Traffic Monitoring: Opreste monitorizarea traficului.

1. Meniul Green Wave:



* Activate: Activeaza unda verde;
* Deactivate: Dezactiveaza unda verde.

In coltul din stanga-jos se mai gasesc si butoanele:



* Start simulation: Porneste simularea sistemului;
* Clear: Sterge toate masinile adaugate.

1. *Testarea aplicatiei*

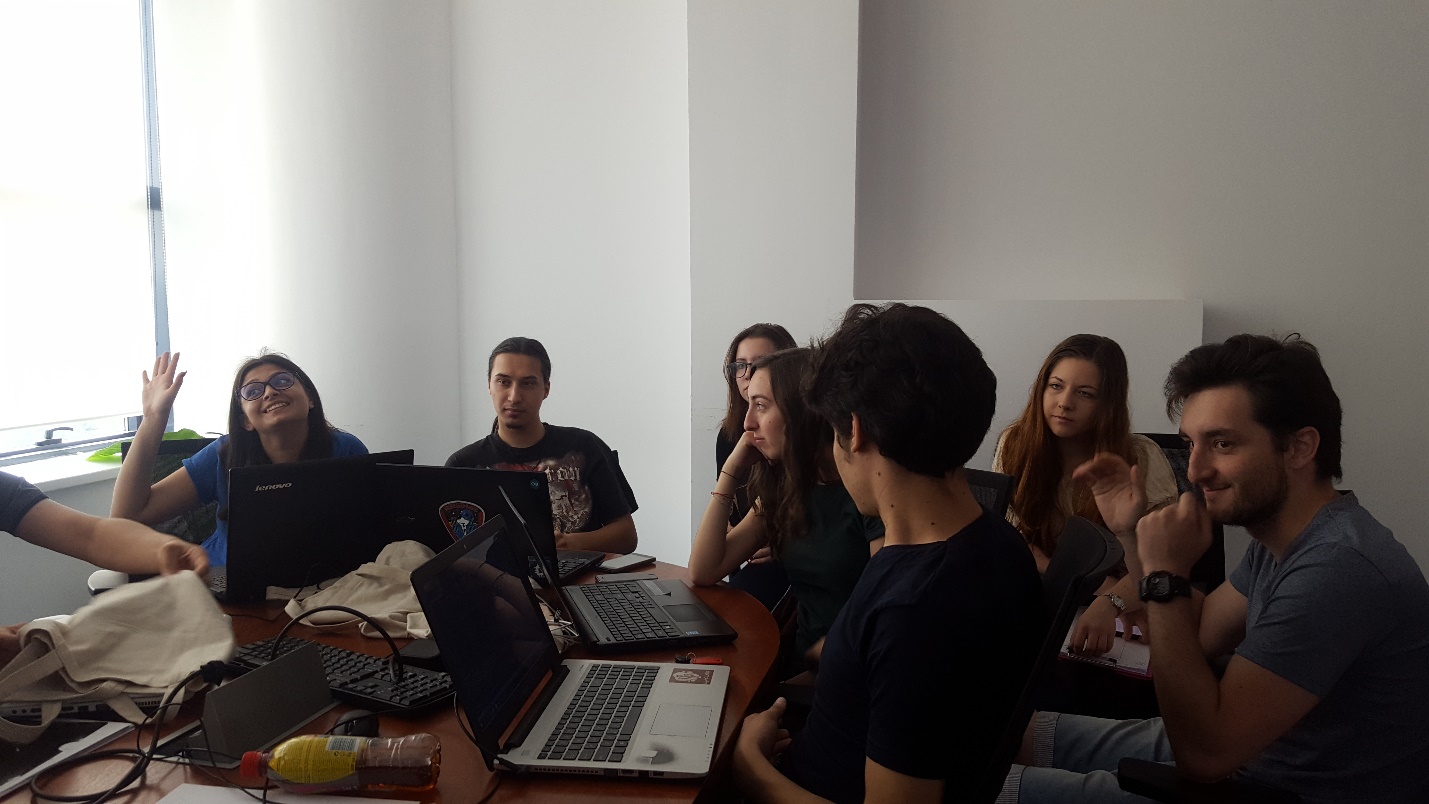
|  |  |  |
| --- | --- | --- |
| **Actiune testor** | **Raspuns asteptat** | **Raspuns real** |
| selectare **CarType** | * permite selectarea obiectelor pentru desfasurarea simularii | * permite selectarea unor masini diferite (culoare rosie/gri) * permite alegerea tramvaiului |
| selectare **Path** | * utilizatorul poate selecta banda si traseul pe care se va circula | * utilizatorul poate selecta banda si traseul pe care se va circula |
| bifare **Is the car bad?** | * masina va trece pe culoarea rosie a semaforului * masina va fi inregistrata intr-o evidenta, fiind vazuta de camerele de supraveghere | * masina trece pe culoarea rosie a semaforului * este urmarita de camerele de supraveghere si inregistrata in evidenta |
| introducere date **Car Speed** | * introducerea vitezei de deplasare a masinii | * introducerea vitezei de deplasare a masinii |
| apasare buton **Add** | * adauga masina cu specificatiile stabilite | * adauga masina cu specificatiile stabilite |
| apasare buton  **Start Traffic Lights Sync** | * sincronizare semafoare | * sincronizare semafoare |
| Senzor:  apasare buton  **Start Traffic Monitoring** | * activare senzori (in caz de trafic, se adauga secunde in plus la culoarea verde) | * activare senzori (adauga secunde in plus la culoarea verde) |
| Senzor:  apasare buton  **Stop Traffic Monitoring** | * oprirea senzorilor | * oprirea senzorilor (semaforul sta pe verde cat este setat initial in program) |
| Unda verde:  apasare buton **Activate** | * se activeaza unda verde | * se activeaza unda verde (masinile ce merg cu viteza recomandata vor prinde verde pe tot traseul) |
| Unda verde:  apasare buton **Deactivate** | * se opreste unda verde | * se opreste unda verde (masinile vor prinde semafoare pe rosu) |
| apasare buton  **Start Simulation** | * porneste simularea | * porneste simularea |
| apasare buton  **Stop Simulation** | * opreste simularea | * opreste simularea |
| apasare buton  **Clear** | * reinitializeaza scenariul | * reinitializeaza scenariul |
| selectare **Scenario** | * se afiseaza informatii referitoare la datele stabilite initial | * se afiseaza informatii referitoare la datele stabilite initial * afiseaza timpul masinii, traseul, daca trece sau nu pe rosu, respectiv viteza |
| selectare **Statistics** | * afiseaza evidenta masinilor care au trecut pe culoarea rosie | * afiseaza evidenta masinilor care au trecut pe culoarea rosie |

1. *Concluzii*

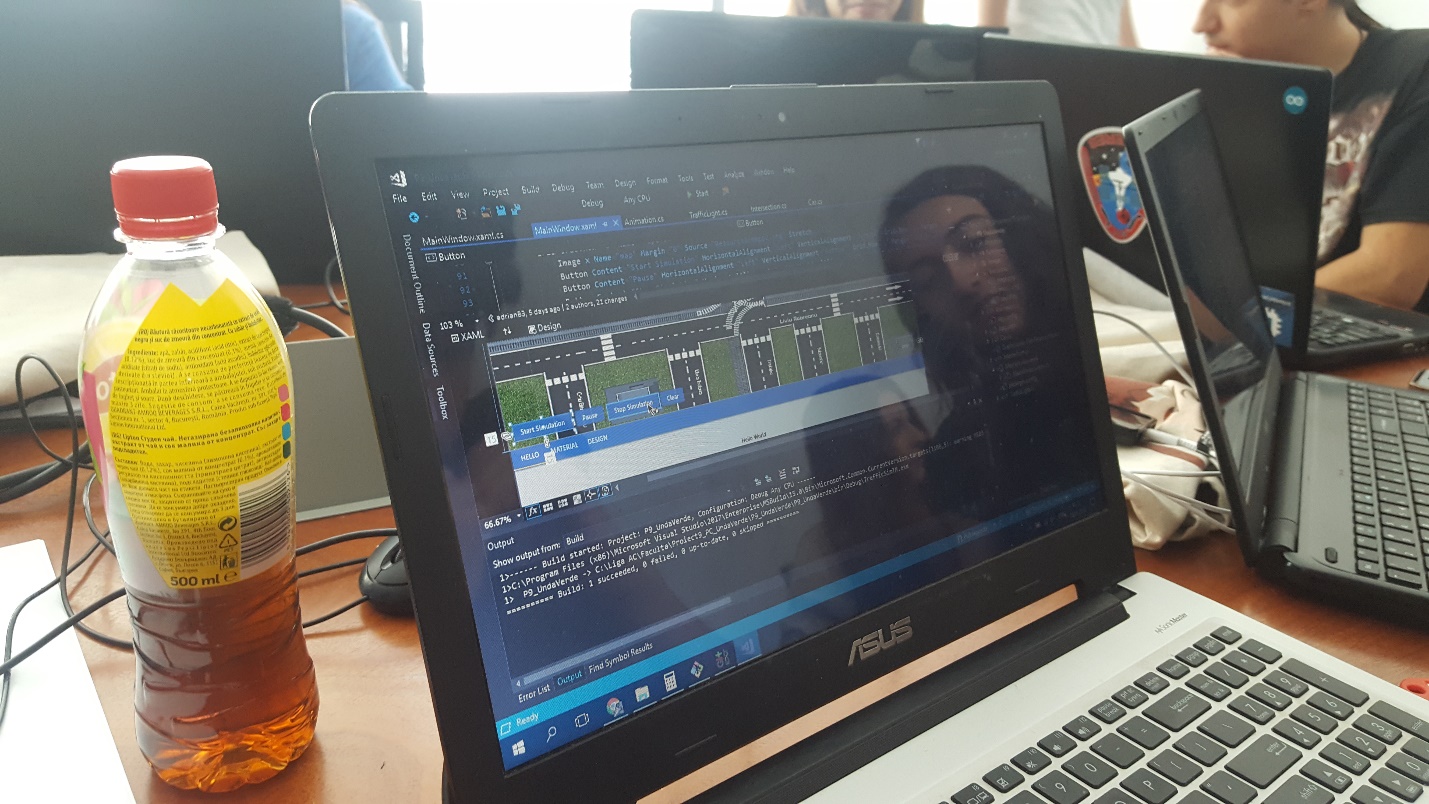
In final, s-a reusit realizarea unei aplicatii care simuleaza traficul pe bulevardul Liviu Rebreanu. Mai exact, aplicatia ofera utilizatorului posibilitatea de a crea un anume scenariu, in functie de preferinte, care permite acestuia sa observe cum se desfasoara traficul conform setarilor alese.

Ne-a interesat in mod special unda verde pe bulevardul respectiv, adica masinile ce pornesc dintr-un capat al traseului si merg cu viteza recomandata, sa reuseasca sa ajunga in celalalt capat fara a fi nevoite sa opreasca la vreun semafor.

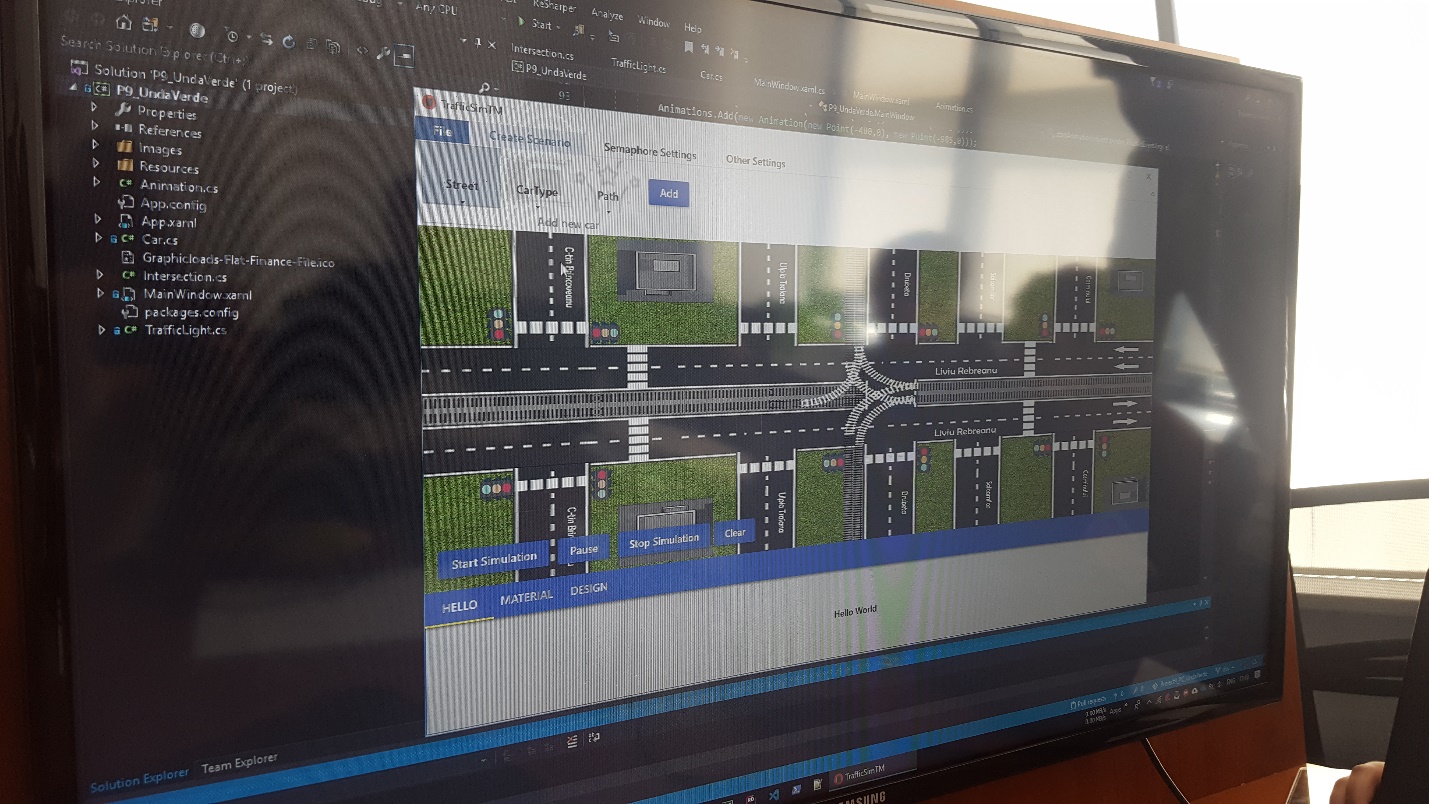
Inca de la inceput, grupa s-a impartit in mai multe arii de lucru: DevTeam, DesignTeam si DocTeam.



DevTeam s-a ocupat efectiv de scrierea si implementarea codului pentru semafoare.



DesignTeam s-a ocupat cu desenarea unor schite dupa care sa se aleaga un design final al proiectului.Tot ei s-au ocupat de desenarea tuturor elementelor din cadrul proiectului (strazi, semafoare, masini, camere de filmat, senzori).



DocTeam s-a ocupat de scrierea documentatiei, poze si explicarea in detaliu a proiectului, cat si de realizarea prezentarii PowerPoint.



Acest proiect ne-a ajutat sa intelegem mai bine programarea concurenta, programarea in C Sharp si am avut ocazia sa folosim medii de dezvoltare noi. Am invatat sa fim organizati si sa impartim sarcinile astfel incat sa eficientizam munca. Am reusit sa lucram in echipa, fiind coordonati de seful de proiect, sa indeplinim task-urile in timp util si sa le punem impreuna pentru a obtine rezultate. Consideram ca lucrul in echipa a avut un impact pozitiv asupra grupei, fapt ce ne va ajuta si pe viitor cand ne vom angaja si vom fi nevoiti sa lucram in echipe.



**Proiect9\_PC\_UndaVerde**

**Project Manager**

* Andreea Carp @AndreeaCamelia

**DevTeam**

* Andrei Chirap @AndreiChirap
* Adrian-Gabriel Balanescu @adrianB3
* Gabriel Bizdoc @GabiBVG
* Raul-Adrian Chincea @RaulChincea
* Diana Dalea @dianadalea
* Andreea Balasoiu @AndreeaBalasoiu
* Alina Bacalete @AlinaBacalete
* Voicu Carole @carolevoicu

**DesignTeam**

* Anamaria Larisa Bala @AnamariaLarisa
* Simona-Rebeca Buse @SimonaRebeca
* Adrian Coneac @adrianconeac
* Mario-Razvan Cioara @MarioCioara
* Raluca-Andreea Cozma @ralucacozma
* Diana Dalea @dianadalea

**DocsTeam**

* Anamaria Larisa Bala @AnamariaLarisa
* Simona-Rebeca Buse @SimonaRebeca
* Adrian Coneac @adrianconeac
* Raul Cojocaru @raulcojocaru
* Robert Burdusel @robertb21

1. *Bibliografie*

**Programare concurenta: mecanisme suport orientate timp real**   
– prof.dr.ing. Nicolae Robu

<https://github.com/adrianB3/Proiect9_PC_UndaVerde>

<http://mapmf.pmfst.unist.hr/~tdadic/Apress.Pro.dotNET.4.Parallel.Programming.in.CSharp.May.2010.pdf>

<http://www.tutorialsteacher.com/csharp/csharp-tutorials>

<http://jonskeet.uk/csharp/threads/>

<https://docs.microsoft.com/en-us/dotnet/framework/wpf/graphics-multimedia/animation-overview>

<http://www.math.uaic.ro/~mapetrii/POO/depozit/Curs7.pdf>