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. *Specificațiile aplicației ......................................................................... pag 6*
4. *Implementare ....................................................................................... pag 9*
5. *Utilizarea sistemului ........................................................................... pag 42*
6. *Testarea aplicației ............................................................................... pag 44*
7. *Concluzii ............................................................................................. pag 46*
8. *Bibliografie ......................................................................................... pag 50*
9. *INTRODUCERE*

* ***Tema proiectului***

Realizarea unei aplicații care să asigure deplasarea în undă verde a tuturor mașinilor care pornesc dintr-un capăt 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 inteligentă***

Sistemele de reglare a circulației reprezintă ansambluri funcționale caracterizate prin existența unor puncte mobile sau fixe, dispersate în spațiu în care se efectuează schimburi de informație în vederea realizării unui transport în timp util, comod, ritmic și în condiții impuse de siguranță pentru procesul de circulație. Traficul este caracterizat, în primul rând, prin viteza de circulație și prin densitatea vehiculelor. Totodată, mărimea traficului este în strânsă legatură cu dezvoltarea economică a teritoriului respectiv.

În condițiile dezvoltării în ritm susținut a economiei pe întreg teritoriul țării se impune necesitatea unui progres tehnic mai rapid, a modernizării radicale a producției și tehnologiilor.

Proiectare și realizarea arterelor de circulație presupune și anticiparea faptului că, odată cu dezvoltarea economică a țării respective vor apărea aglomerări urbane, care la rândul lor se vor reflecta în modul de organizare și desfășurare a deplasării persoanelor, a autovehiculelor rutiere. Astfel, un loc important în coordonarea dirijării traficului îl au instalațiile de semaforizare.

Semaforizarea unei intersecții stradale nu reprezintă decât un aspect izolat al problemei dirijării automate a traficului urban. Indiferent de corectitudinea proiectării și instalării unei instalații de semaforizare într-o intersecție și de promptitudinea și rigurozitatea cu care programele de funcționare a semafoarelor răspund la schimbările permanente ale traficului, dacă diferitele instalații de semaforizare, aferente intersecțiilor din același teritoriu urban, prezintă un mod de funcționare propriu, circulația din zona respectivă va avea un caracter întâmplător.

Se impune o coordonare a modului de funcționare a semafoarelor, succesiunea proprie a fazelor în cadrul aceluiași ciclu, momentele precise pentru o anumită intersecție, dar întâmplătoare la nivel de zonă, la care este servită o anumită fază de circulație, conduc la un caracter cu totul întâmplător al desfășurării circulației din zona urbană respectivă.

Aceasta impune o coordonare a modului de funcționare a diferitelor instalații de semaforizare, la nivelul unei zone urbane sau al unui întreg oraș. Avantajele unor asemenea tipuri de coordonări sunt legate de :

- fluidizarea circulației de-a lungul traseelor considerate ;

- realizarea unor importante economii de timp și combustibil ;

- diminuarea poluării.

În mod practic această coordonare se realizează, funcție de cerințele majore ale traficului, pentru următoarele genuri de situații :

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

- în cadrul unei zone urbane(coordonare de suprafață).

Când două sau mai multe intersecții se găsesc în adiacență pe aceeași arteră principală de circulație, sunt necesare anumite forme de coordonare a dirjării circulației, pentru a reduce întârzierile și a preveni caracterul de continuitate al opririlor.

În cele ce urmează, vom descrie câteva din sistemele utilizate pentru coordonarea instalațiilor de semaforizare asociate unui sistem liniar de dirijare a traficului, în special sistemul „undă verde”.

Sistemul „***undă verde***” este, de fapt, un dispozitiv pentru dirijarea traficului vehiculelor pe o arteră de circulație în undă verde cu ajutorul semafoarelor, astfel încât un vehicul să poată parcurge această arteră fără să oprească. Acest sistem este obținut prin integrarea funcțională a tuturor echipamentelor amplasate în intersecțiile succesive ale unei magistrale urbane cu un număr mare de intersecții și cu trafic intens.

În acest mod un anumit vehicul care circulă cu o anumită viteză indicată,va găsi succesiv culoarea verde la semafoarele S1,S2,S3,....Pentru aceasta, se folosesc semafoarele de sincronizare prin intermediul cărora se impune o funcționare secvențială a posturilor din intersecții,spre a permite accesul vehiculelor și din arterele secundare adiacente(Sa,Sb,Sc,...).

Scopul acestui sistem constă în asigurarea trecerii unui volum maxim de trafic, de-a lungul traseului coordonat, fără oprire, în înlăturarea cozilor de așteptare ale vehiculelor la intersecție.

1. *Tehnologii utilizate*

Aplicația a fost realizată în C# (C Sharp), folosind mediul de dezvoltare Visul Studio.

Am decis să folosim C# deoarece acest limbaj prezintă diverse avantaje:

* **C#** este un limbaj de programare orientat pe obiecte, simplu, modern, ce derivă din C++ și Java.
* **Simplitatea** limbajului este dată de absența pointerilor, moștenirea facilităților de „garbage collection” și managementul automat al memoriei datorită integrării pe platforma .NET, iar operațiile nesigure precum manipularea directă a memoriei nu sunt permise.
* **Modernitatea** limbajului: C# este conceput pentru construirea de operații scalabile, robuste și interoperabile. Include un suport “built in” pentru transformarea componentelor în serviciu web pentru un acces mai ușor.
* **Orientat pe obiecte**: suportă încapsularea datelor. Interfețele, moștenirea și polimorfismul. A introdus structurile (structs) pentru a permite tipurilor primitive să devină obiecte.
* **Portabilitate**: Codul C# poate rula pe mai mult de 2,2 miliarde de dispozitive cu Windows, iOS, Android, Linux. C# este portabil într-o gamă largă de medii cum ar fi telefonia mobila, Embedded, desktop și 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 ușura munca și pentru a avea fiecare acces la datele proiectului și munca fiecărui 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. *Specificațiile aplicației*

* ***Prezentarea funcțiilor***

Pentru a mări timpul culorii verzi la semafor, folosim următoarea funcție :

public void increaseGreenTime()

{

\_greenWaitTime += 2;

}

Pentru a micșora timpul culorii verzi a unui semafor, folosim :

public void decreaseGreenTime()

{

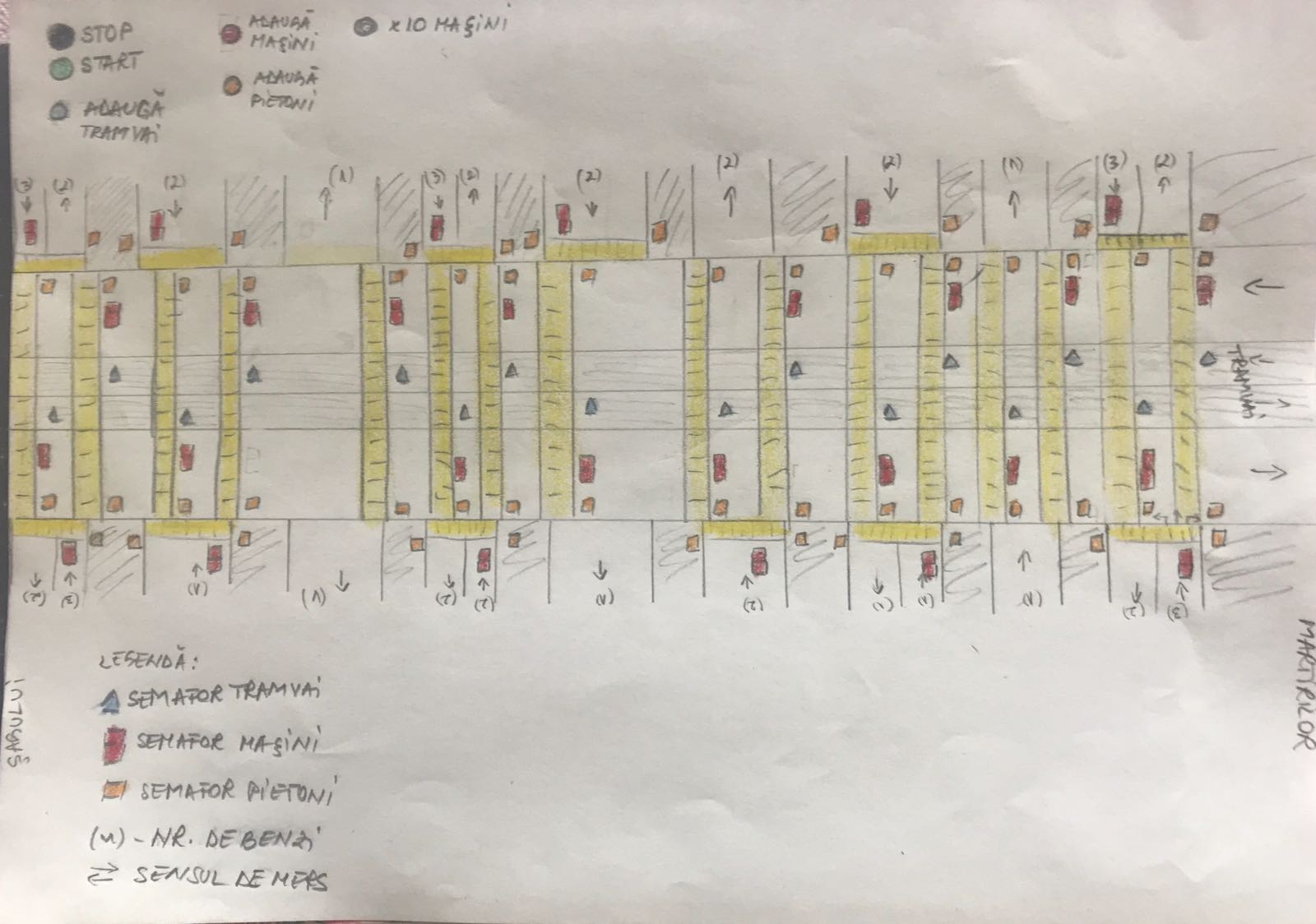
\_greenWaitTime -= 2;

}

Pentru a aprinde culoarea rosie sau verde a semaforului folosim o funcție, în cadrul căreia folosim un task ce asigură funcționarea independentă a fiecărui semafor.

Funcția „public void StartIntersectionSync” pornește sincronizarea semafoarelor din cadrul intersecției, în cadrul ei este implementată o listă de task-uri ce reține task-ul care pornește un semafor și tot în cadrul funcției se realizează reguli de sincronizare între semafoare.

Folosim o funcție de adăugare a mașinuței în fereastra principală , dar și un task ce asigură crearea în paralel a tuturor mașinuțelor .



Imaginea a fost realizată la începutul aplicației, mai apoi fiind folosită ca o schiță pentru realizarea designului pe PC.

Alături de poză, au fost atașate câteva informații despre cum ar trebui să funcționeze sistemul de semaforizare unde sunt testate următoarele cazuri :

„La apăsarea butonului start, semafoarele vor funcționa în undă verde, indiferent dacă există vehicule sau nu.

Ulterior, prin apăsarea butoanelor Adaugă tramvai, mașini, pietoni se vor introduce și aceste elemente pentru observarea mai bună a funcționalității.

Butonul x10 mașini va înmulți numărul de mașini cu 10 pentru a observa cum schimbarea acestui număr influențează timpul în care semafoarele au culoarea verde.”

În continuare, vom prezenta varianta finală a aplicației, urmată de schema logică:



NU

NU

NU

NU

DA

DA

DA

DA

DA

Au fost adăugate elemente pentru simulare?

Sunt eliminate elementele de trafic

Intersecție aglomerată?

Durata de trecere a mașinilor crește cu 10 secunde

A fost apăsat butonul ‘Clear’?

Mașinile beneficiază de Unda Verde

Simulare în desfășurare

Sistem pornit

A fost activată sincronizarea semafoarelor?

A fost activată Unda Verde?

Au fost activați senzorii?

Se monitorizează Numărul de mașini din fiecare intersecție

NU

DA

1. *Implementare*

SpeedPanel

class SpeedPanel

{

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

public int \_positionFromTop { get; set; } // poziții față de fereastră

public int \_positionFromRight { get; set; }

public int speed { get; set; } // viteza curentă citită de senzor

public int \_indexIntersectie { get; set; } // indexul intersecției unde se află senzorul

public int \_indexSem { get; set; }

public Canvas canv;

public Rectangle speedBoard; // panoul unde se afișează viteza

public Label speedLabel;

SolidColorBrush whiteBrush = new SolidColorBrush(); // culoarea 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(), // afișare viteză în interfață

};

Canvas.SetTop(speedBoard, \_positionFromTop);

Canvas.SetRight(speedBoard, positionFromRight);

Canvas.SetTop(speedLabel, \_positionFromTop);

Canvas.SetRight(speedLabel, positionFromRight);

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

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

mainWin.mapGrid.Children.Add(canv); // se adaugă canvasul la fereastra principală

}

}

Animation

public class Animation

{

MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referință către fereastra principală

public Storyboard story = new Storyboard(); // Obiect ce reține o anumită animație

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

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

private int \_additionalAnims; // opțiuni ptr animație (urmărirea tangentei)

// Constructor clasa Animatie

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

{

\_startPoint = startPoint;

\_endPoint = endPoint;

\_additionalAnims = additionalAnims;

}

/\* Funcție care pornește o animație a unei anumite mașinute care durează un anumit timp\*/

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

{

// Animația este inserată în coada dispecerului threadului de management al interfeței și este pornită

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

{

NameScope.SetNameScope(mainWin, new NameScope()); // este setat contextul animației către fereastra principală

MatrixTransform carTransform = new MatrixTransform(); // creare obiect ce va reține transformările vizuale din timpul animației asupra mașinuței

\_animateObject.\_carImg.RenderTransform = carTransform; // atașare mașinuță la matricea de transformare

mainWin.RegisterName("carTransform", carTransform);

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

PathFigure pathFigure = new PathFigure();

pathFigure.StartPoint = \_startPoint;

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

animPath.Figures.Add(pathFigure);

animPath.Freeze(); // optimizare animație

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

if (\_additionalAnims == 1)

mAnim.DoesRotateWithTangent = true;

mAnim.PathGeometry = animPath; // atașare geometrie

mAnim.Duration = TimeSpan.FromSeconds(animationTime); // durată animație

Storyboard.SetTargetName(mAnim, "carTransform");

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

story.Children.Add(mAnim);

await Task.Delay(delay);

story.Begin(mainWin, true); // începere animație

}));

}

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 folosită 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 folosită pentru stocarea Pathului și semafoarelor întâlnite de o mașinuță într-o anumită animație

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 opțiunile 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 închiderea ferestrei

private void ApplicationExit(object sender, EventArgs e)

{

Application.Current.Shutdown();

}

// Event apelat la apăsarea butonului "Start Simulation"

private void startAnimation(object sender, RoutedEventArgs e)

{

// Creare task pentru fiecare mașinuță

foreach (var car in carsList)

{

if (tokenSource.IsCancellationRequested)

{

break;

}

car.createImage();

// Lista de taskuri care asigură deplasarea unei mașinuțe în mod paralel față de celelalte mașinute

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

{

int i = 0;

foreach (var animation in car.\_animationsList)

{

// Declanșare secventă de animații pentru fiecare mașinuță

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);

// dacă semaforul la care a ajuns mașinuța la un anumit moment este roșu, taskul așteaptă ca acel semafor să devină 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)

{

// dacă o anumită mașinuță trece pe roșu este adăugată în lista mașinuțelor ce au trecut pe roșu

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

{

listOfBadCars.Add(car);

}

i++;

}

}

},tokenSource.Token));

}

// pornire taskuri ptr fiecare mașinuță

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 declanșat la apăsarea butonului de aprindere a semafoarelor

private void StartTrafficLightsSync(object sender, RoutedEventArgs e)

{

// Task ce asigură pornirea sincronizării fiecărei intersecții în paralel

Task intersectionSyncTask = new Task(async () =>

{

foreach (var intersection in Intersections)

{

intersection.StartIntersectionSync();

await Task.Delay(2000);

}

});

intersectionSyncTask.Start(TaskScheduler.FromCurrentSynchronizationContext());

}

// Event declanșat la apăsarea butonului Add din Meniu prin care este adăugată o nouă mașinuță în cadrul scenariului de simulare

private void AddCar(object sender, RoutedEventArgs e)

{

// Creare listă de animații 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; // referință către fereastra principală

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

// Constructor clasa Intersecție ce preia coordonatele la care se află 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 în intersecție

}

}

// Funcție ce pornește sincronizarea semafoarelor din cadrul intersecției

public void StartIntersectionSync()

{

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

foreach (var trafficLight in \_TrafficLights)

{

listOfTasks.Add(trafficLight.LightUp());

}

// Pornire semafoare, creare reguli de sincronizare între 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 în cadrul threadului de management al interfeței

}

}

}

}

TrafficLights

class TrafficLight

{

MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referința către fereastra principală

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

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

public int \_redWaitTime { get; set; } // timpul de așteptare pe roșu

public int \_delay { get; set; }

private int \_positionFromTop { get; set; } // poziție față de partea de sus a ferestrei

private int \_positionFromRight { get; set; } // poziție față 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 reține imaginea cu semaforul

Ellipse redLight = new Ellipse(); // forma ce va reține culoarea roșie

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

Ellipse greenLight = new Ellipse(); // forma ce va reține 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); // adăugare semafor la fereastra principală

}

// funcție ce mărește timpul de verde

public void increaseGreenTime()

{

\_greenWaitTime += 2;

}

// funcție ce micșorează timpul de verde

public void decreaseGreenTime()

{

\_greenWaitTime -= 2;

}

public void increaseRedTime()

{

\_redWaitTime += 2;

}

// funcție ce micșoreaza timpul de verde

public void decreaseRedTime()

{

\_redWaitTime -= 2;

}

public bool isGreen()

{

return \_color ? true : false;

}

public bool isRed()

{

return \_color ? false : true;

}

// funcție ce aprinde culoarea roșie sau verde a semaforului

public Task LightUp()//o acțiune ce se execută asincron

{

// Task ce asigură funcționarea independentă a fiecărui 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;//returnează pt a fi activat din clasa intersecție

}

}

}

Sensor

public class Sensor

{

MainWindow mainWin = Application.Current.Windows[0] as MainWindow; // referință către fereastra principală

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

public int \_numberOfCars { get; set; } // număr curent de mașini citite

public int \_indexIntersectie { get; set; } // reține indexul intersecției unde se află senzorul

public int \_indexSemafor { get; set; }

public bool \_isActivated { get; set; } // variabilă ce reține dacă 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); // adăugare canvas la fereastra principală

}

// Funcție care returnează dacă o intersecție e prea aglomerată

public bool \_isCrowded()

{

return \_numberOfCars > 5 ? true : false;

}

// Funcție care incrementează nr de mașini care trec de senzor

public void \_Signal()

{

\_numberOfCars++;

}

// Funcție care resetează senzorul

public void \_Reset()

{

\_numberOfCars = 0;

}

// Funcție care pornește un senzor

public void startSensor()

{

blueLight.Fill = colorBrush;

\_isActivated = true;

}

// Funcție care oprește 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; // referință către fereastra principală

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

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

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

private int \_height { get; set; }

private int \_positionFromTop { get; set; } // poziții în cadrul ferestrei

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 animațiile ce vor fi executate de mașinuță

public List<ints> intSem; // semafoarele prin care va trece mașinuța în timpul animațiilor

private Canvas canv;

public bool \_isABadCar; // var ce va reține dacă mașinuța încalcă legea

// Constuctor clasă mașină

public Car(

List<ints> intS, // lista intersecții și semafoare

List<Animation> animationsList, // secvență animații

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();

}

// Funcție de adăugare a mașinuței în fereastra principală

public void createImage()

{

// Task ce asigură crearea în paralel a tuturor mașinuțelor

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());

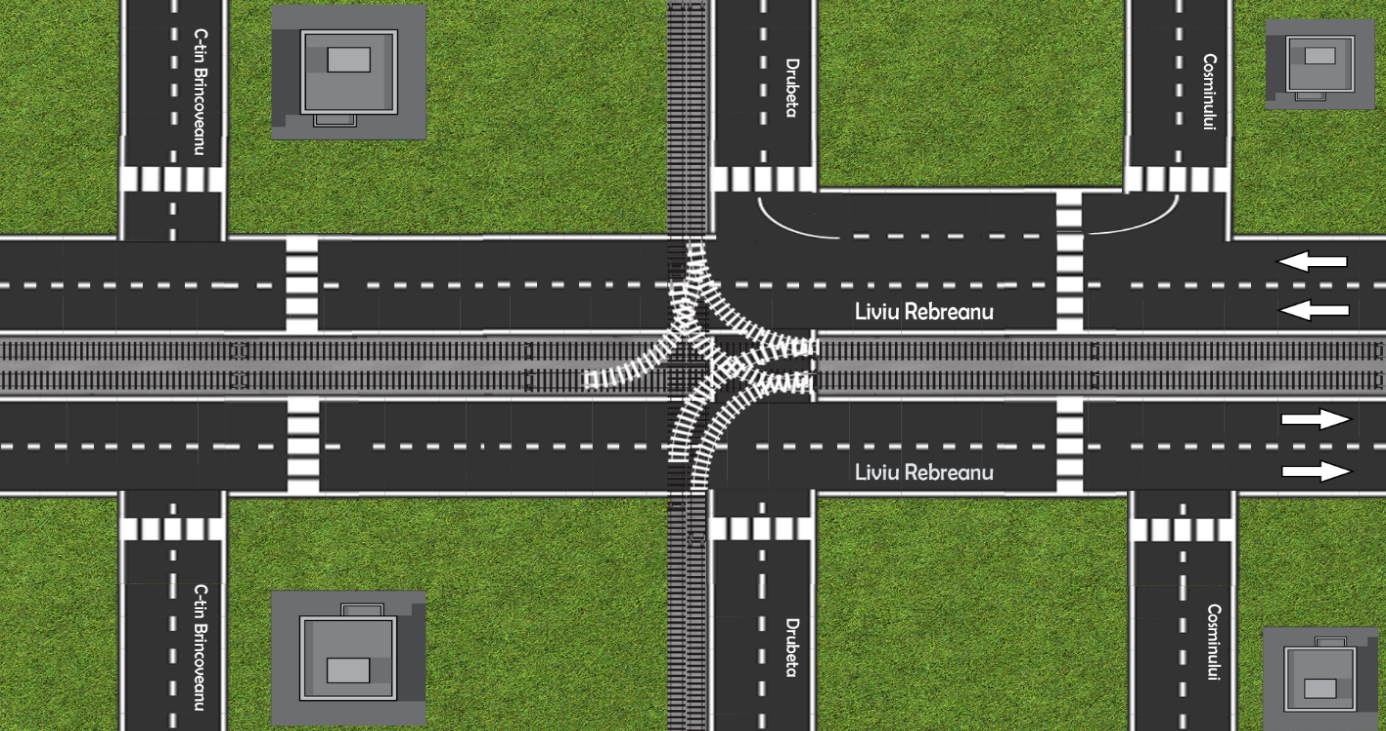
}

1. *Utilizarea sistemului*

Acest sistem simulează funcționalitatea unui semafor în timp real. Sistemul poate simula mai multe situații ipotetice ce se pot întâmpla în viața de zi cu zi în trafic: măsurarea traficului, crearea de undă verde, determinarea vitezei autovehiculelor.



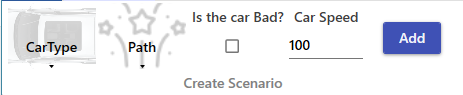
Pentru început, s-a lucrat în photoshop la design-ul proiectului:



Mai apoi, s-a început adăugarea de elemente în cadrul proiectului.

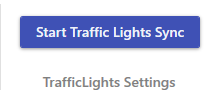
În componenta programului putem găsi următoarele elemente:

1. Meniul Create Scenario:



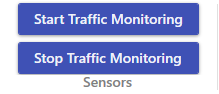
* CarType: Se pot alege diferite autovehicule cât și tramvaie care să fie introduse în simulare;
* Path: Selectează traseul pe care îl va urma mașina;
* Starea mașinii;
* Viteza mașinii.

1. Meniul TrafficLights Settings :



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

1. Meniul Sensors:



* Start Traffic Monitoring: Pornește monitorizarea traficului(în cazul unui număr mare de mașini se mărește timpul pe culoarea verde);
* Stop Traffic Monitoring: Oprește monitorizarea traficului.

1. Meniul Green Wave:



* Activate: Activează unda verde;
* Deactivate: Dezactivează unda verde.

În colțul din stânga-jos se mai găsesc și butoanele:



* Start simulation: Pornește simularea sistemului;
* Clear: Șterge toate mașinile adăugate.

1. *Testarea aplicației*

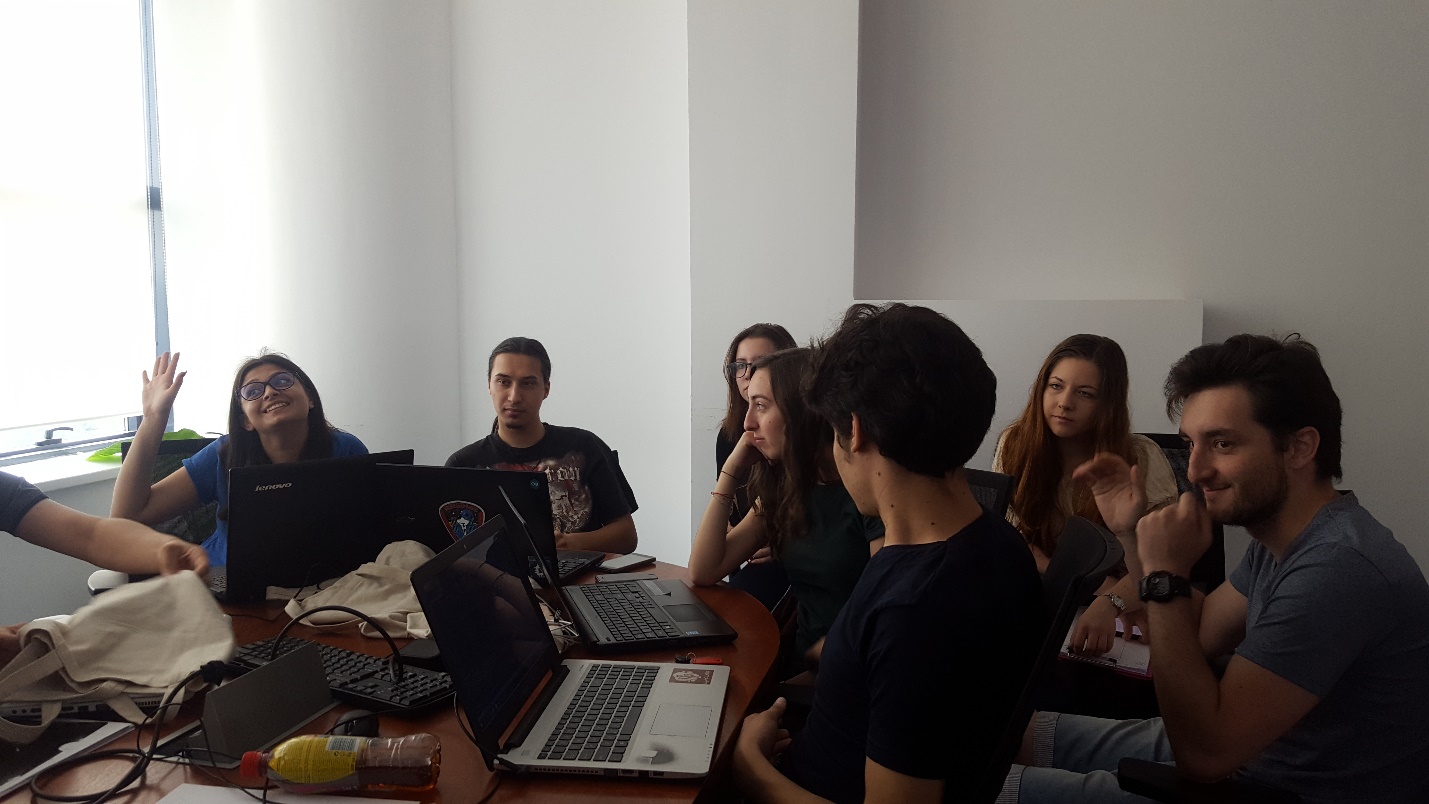
|  |  |  |
| --- | --- | --- |
| **Actiune testor** | **Răspuns așteptat** | **Răspuns real** |
| selectare **CarType** | * permite selectarea obiectelor pentru desfășurarea simulării | * permite selectarea unor mașini diferite (culoare roșie/gri) * permite alegerea tramvaiului |
| selectare **Path** | * utilizatorul poate selecta banda și traseul pe care se va circula | * utilizatorul poate selecta banda și traseul pe care se va circula |
| bifare **Is the car bad?** | * mașina va trece pe culoarea roșie a semaforului * mașina va fi înregistrată într-o evidență, fiind vazută de camerele de supraveghere | * mașina trece pe culoarea roșie a semaforului * este urmărită de camerele de supraveghere și înregistrată în evidență |
| introducere date **Car Speed** | * introducerea vitezei de deplasare a mașinii | * introducerea vitezei de deplasare a mașinii |
| apăsare buton **Add** | * adaugă mașina cu specificațiile stabilite | * adaugă mașina cu specificațiile stabilite |
| apăsare buton  **Start Traffic Lights Sync** | * sincronizare semafoare | * sincronizare semafoare |
| Senzor:  apăsare buton  **Start Traffic Monitoring** | * activare senzori (in caz de trafic, se adaugă secunde în plus la culoarea verde) | * activare senzori (adaugă secunde în plus la culoarea verde) |
| Senzor:  apăsare buton  **Stop Traffic Monitoring** | * oprirea senzorilor | * oprirea senzorilor (semaforul stă pe verde cât este setat inițial în program) |
| Unda verde:  apăsare buton **Activate** | * se activează unda verde | * se activează unda verde (mașinile ce merg cu viteza recomandată vor prinde verde pe tot traseul) |
| Unda verde:  apăsare buton **Deactivate** | * se oprește unda verde | * se oprește unda verde (mașinile vor prinde semafoare pe roșu) |
| apăsare buton  **Start Simulation** | * pornește simularea | * pornește simularea |
| apăsare buton  **Stop Simulation** | * oprește simularea | * oprește simularea |
| apăsare buton  **Clear** | * reinițializează scenariul | * reinițializează scenariul |
| selectare **Scenario** | * se afișează informații referitoare la datele stabilite inițial | * se afișează informații referitoare la datele stabilite inițial * afișeaza timpul mașinii, traseul, dacă trece sau nu pe roșu, respectiv viteza |
| selectare **Statistics** | * afișează evidența mașinilor care au trecut pe culoarea roșie | * afișează evidența mașinilor care au trecut pe culoarea roșie |

1. *Concluzii*

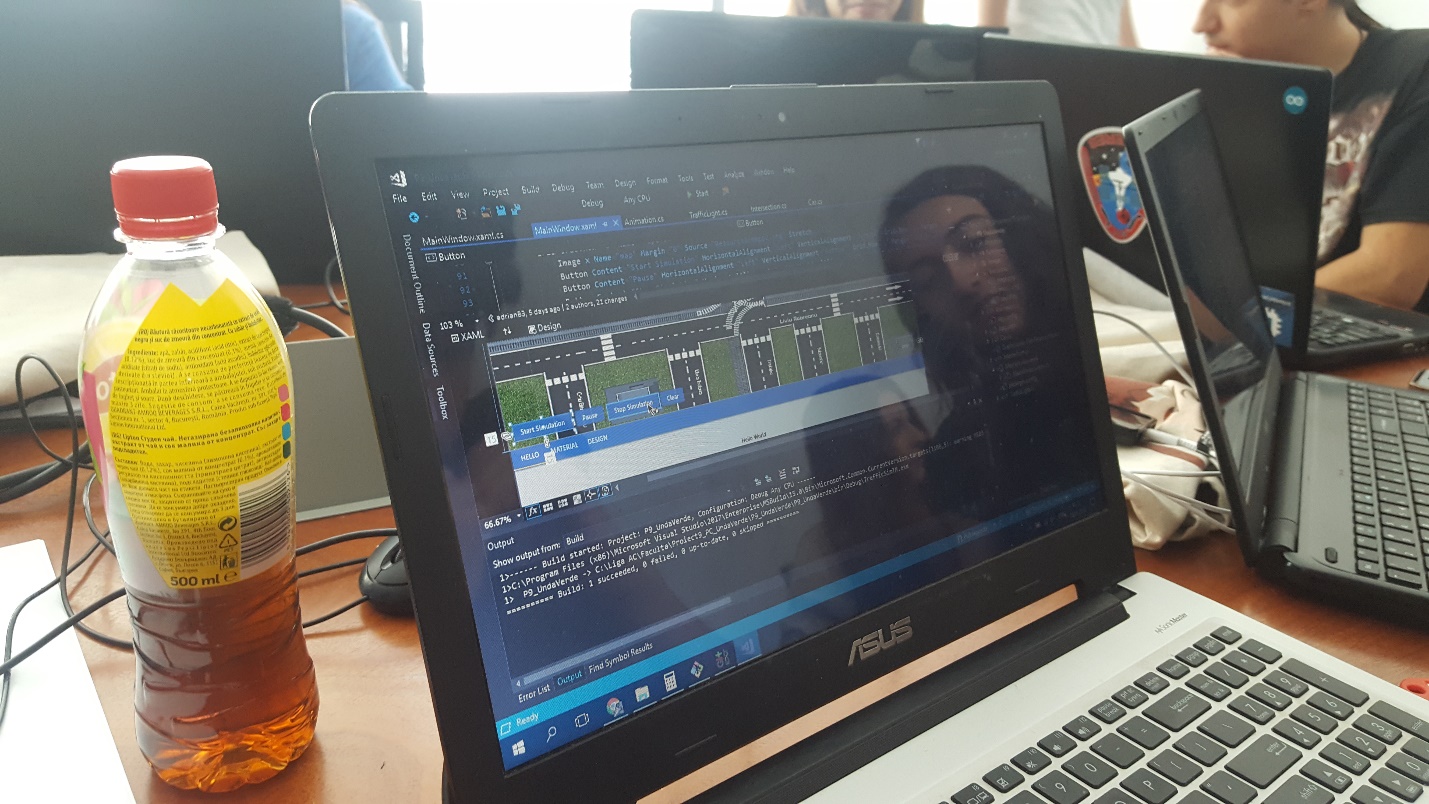
În final, s-a reușit realizarea unei aplicații care simulează traficul pe bulevardul Liviu Rebreanu. Mai exact, aplicația oferă utilizatorului posibilitatea de a crea un anume scenariu, în funcție de preferințe, care permite acestuia să observe cum se desfășoară traficul conform setărilor alese.

Ne-a interesat în mod special unda verde pe bulevardul respectiv, adică mașinile ce pornesc dintr-un capăt al traseului și merg cu viteza recomandată, să reușească să ajungă în celălalt capăt fără a fi nevoite să oprească la vreun semafor.

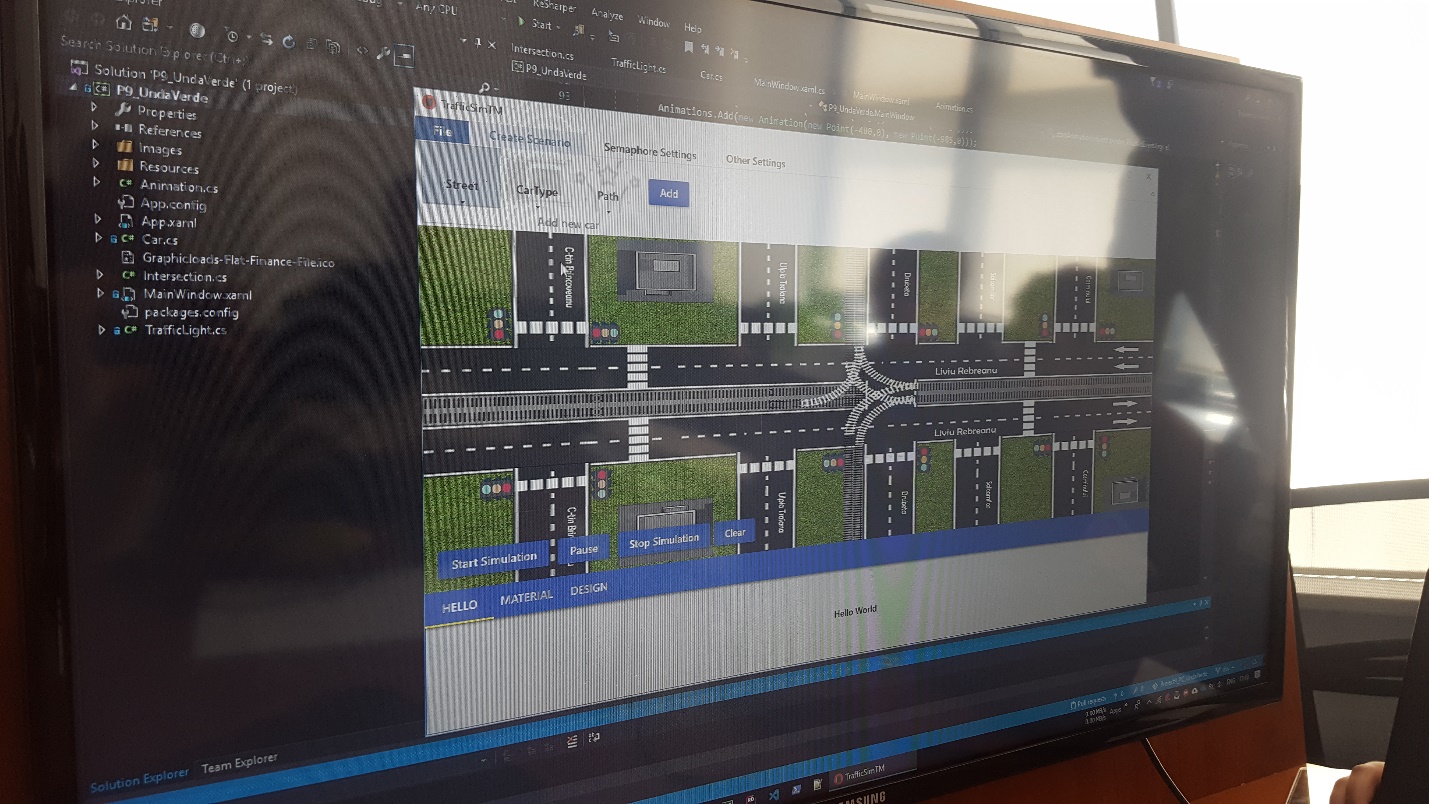
Încă de la început, grupa s-a împărțit în mai multe arii de lucru: DevTeam, DesignTeam și DocTeam.



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



DesignTeam s-a ocupat cu desenarea unor schițe după care să se aleagă un design final al proiectului.Tot ei s-au ocupat de desenarea tuturor elementelor din cadrul proiectului (străzi, semafoare, mașini, camere de filmat, senzori).



DocTeam s-a ocupat de scrierea documentației, poze și explicarea în detaliu a proiectului, cât și de realizarea prezentării PowerPoint.



Acest proiect ne-a ajutat să înțelegem mai bine programarea concurentă, programarea în C Sharp și am avut ocazia să folosim medii de dezvoltare noi. Am învățat să fim organizați și să împărțim sarcinile astfel încât să eficientizăm munca. Am reușit să lucrăm în echipă, fiind coordonați de șeful de proiect, să îndeplinim task-urile în timp util și să le punem împreună pentru a obține rezultate. Considerăm că lucrul în echipă a avut un impact pozitiv asupra grupei, fapt ce ne va ajuta și pe viitor când ne vom angaja și vom fi nevoiți să lucrăm în echipe.



**Proiect9\_PC\_UndaVerde**

**Project Manager**

* Andreea Carp @AndreeaCamelia

**DevTeam**

* Andrei Chirap @AndreiChirap
* Adrian-Gabriel Bălănescu @adrianB3
* Gabriel Bîzdoc @GabiBVG
* Raul-Adrian Chincea @RaulChincea
* Diana Dalea @dianadalea
* Andreea Bălășoiu @AndreeaBalasoiu
* Alina Băcălete @AlinaBacalete
* Voicu Carole @carolevoicu

**DesignTeam**

* Anamaria Larisa Bălă @AnamariaLarisa
* Simona-Rebeca Bușe @SimonaRebeca
* Adrian Coneac @adrianconeac
* Mario-Răzvan Cioară @MarioCioara
* Raluca-Andreea Cozma @ralucacozma
* Diana Dalea @dianadalea

**DocsTeam**

* Anamaria Larisa Bălă @AnamariaLarisa
* Simona-Rebeca Bușe @SimonaRebeca
* Adrian Coneac @adrianconeac
* Raul Cojocaru @raulcojocaru
* Robert Burdușel @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>