III. 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 urmatoarea functie ,in cadrul careia folosim un task ce asigura functionarea independenta a fiecarui semafor

public Task LightUp()

{

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;

}

}

}

Urmatoarea functie porneste sincronizarea semafoarelor din cadrul intersectiei ,in cadrul ei este implementata o lista de taskuri ce retine taskul ce porneste un semafor si tot in cadrul functiei se realizeaza reguli de sincronizare intre semafoare.

public void StartIntersectionSync()

{

var listOfTasks = new List<Task>(); foreach (var trafficLight in \_TrafficLights)

{

listOfTasks.Add(trafficLight.LightUp());

}

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 }

}

}

}

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

public void createImage()

{

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());

}

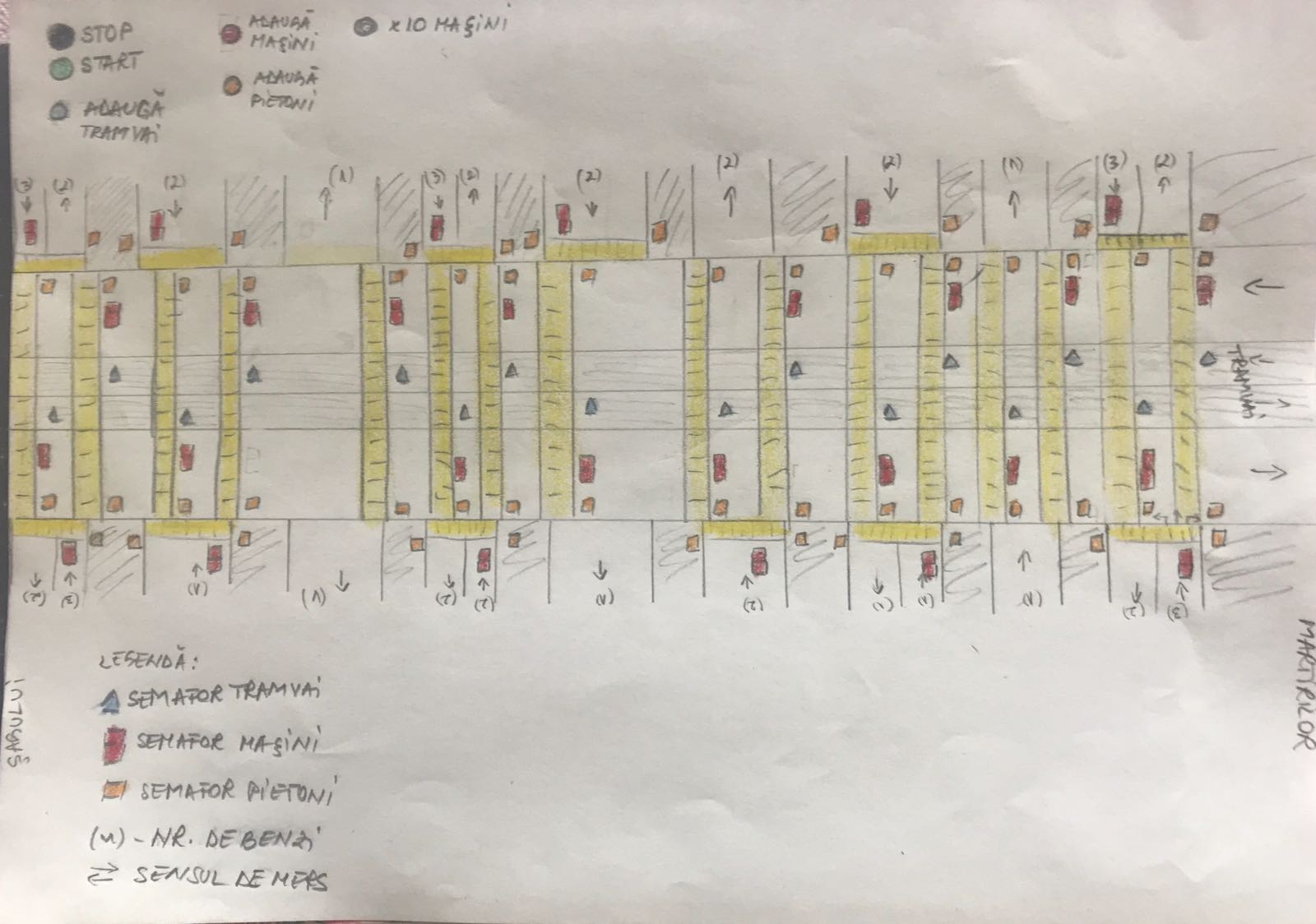
public void removeImage()

{

canv.Children.Remove(\_carImg);

}

} }



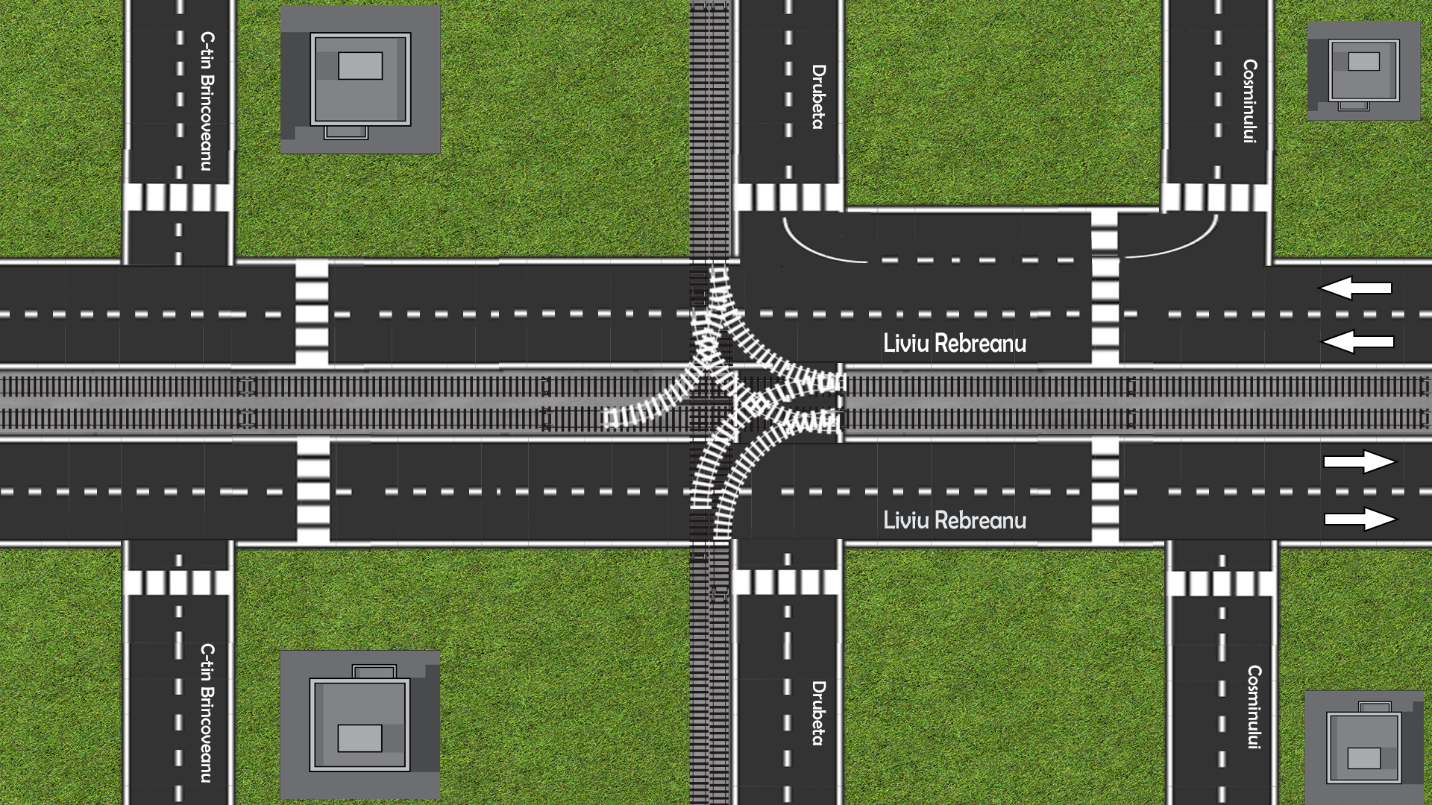
Poza este realizata la inceputl aplicatiei ,mai apoi fiind folosita ca o schita pentru realizarea designului pe PC.

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 produsul final al mapei



IV. Implementare:

MainWindow\_xaml\_cs

// 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)

};

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

"Cosminului 1 --> Drubeta 1", //8

"Cosminului 1 --> Drubeta 2", //9

};

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

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

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()

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));

AvailableCarTypes.ItemsSource = AvailableVehicles;

AvailablePathsListBox.ItemsSource = AvailablePaths;

selectedItemsListView.ItemsSource = selectedThings;

listBoxOfBadCars.ItemsSource = listOfBadCars;

sensorsListView.ItemsSource = Sensors;

}

// 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 unuei 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, animation.speedCalculation(car), 0);

foreach (var sensor in Sensors)

{

sensor.startSensor();

sensor.\_Signal();

}

await Task.Delay(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);

}

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

{

i++;

// 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);

}

}

}

},tokenSource.Token));

}

// pornire taskuri ptr fiecare masinuta

foreach (var task in listOfTasks)

{

task.Start(TaskScheduler.FromCurrentSynchronizationContext());

}

}

private void windowLoaded(object sender, RoutedEventArgs e)

{

}

private void stopAnimation(object sender, RoutedEventArgs e)

{

// tokenSource.Cancel();

}

// 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;

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

}

if (AvailablePathsListBox.SelectedIndex == 1)

{

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

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;

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(-500, -5),1));

Animations1.Add(new Animation(new Point(-500, -5), 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)

{

// 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

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 = 0},

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

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

};

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

}

if (AvailablePathsListBox.SelectedIndex == 5)

{

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

posT = 260;

posR = 1200;

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

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

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

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

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

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

{

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

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

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

};

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

}

if (AvailablePathsListBox.SelectedIndex == 6)

{

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

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(655, 0), new Point(655, -250), 1));

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

{

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

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

};

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

}

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";

}

selectedThings.Add(new selectedItem()

{

carType = AvailableCarTypes.SelectedItem.ToString(),

path = AvailablePathsListBox.SelectedItem.ToString(),

isBad = isBadCheckBox.IsChecked.Value,

speed = Int32.Parse(speedTextBox.Text)

});

// Event ce asigura reinitializarea unui scenariu

private void clearScenario(object sender, RoutedEventArgs e)

{

foreach (var car in carsList)

{

car.removeImage();

}

carsList.Clear();

listOfTasks.Clear();

selectedThings.Clear();

listOfBadCars.Clear(); // TODO: After the first simulation the bad cars are no longer added to the list of bad cars

}

// Event ce asigura pornirea senzorilor de monitorizare a traficului

private void StartSensorMonitor(object sender, RoutedEventArgs e)

{

foreach (var sensor in Sensors)

{

sensor.startSensor();

}

}

private void StopSensorMonitor(object sender, RoutedEventArgs e)

{

foreach (var sensor in Sensors)

{

sensor.stopSensor();

}

}

// Event ce asigura pornirea sistemului de Unda Verde

private void ActivateGreenWave(object sender, RoutedEventArgs e)

{

foreach (var intersection in Intersections)

{

intersection.\_TrafficLights[0].increaseGreenTime();

intersection.\_TrafficLights[2].increaseGreenTime();

// TODO: there are situations in witch all the traffic lights in an intersection are green

}

}

private void DeactivateGreenWave(object sender, RoutedEventArgs e)

{

foreach (var intersection in Intersections)

{

intersection.\_TrafficLights[0].decreaseGreenTime();

intersection.\_TrafficLights[2].decreaseGreenTime();

}

}

}

}

TrafficLights

namespace TrafficSimTM

{

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 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()

{

// 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;

}

}

}

Intersection

namespace TrafficSimTM

{

class Intersection

{

// 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"};

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

}

}

}

}

Car\_cs

namespace TrafficSimTM

{

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,

List<Animation> animationsList,

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

{

IsEnabled = true,

IsOpen = true,

StaysOpen = true,

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());

}

// Functie folosita pentru inlaturarea masinutei din fereastra principala

public void removeImage()

{

canv.Children.Remove(\_carImg);

}

}

}

Animation\_cs

namespace TrafficSimTM

{

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 void stopAnimation()

{

story.Stop(mainWin);

}

public void pauseAnimation()

{

story.Pause(mainWin);

}

public void resumeAnimation()

{

story.Resume(mainWin);

}

public int speedCalculation(Car car)

{

return Convert.ToInt32(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,1.8f));

}

}

}