*V1.3-Design Document*

****

***Team members:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Ahmed AL-Ajmi* | *Obaid Habib* | *Vytautas*  *Skinkys* | *Rimgaudas Tamulevicius* | *Nurjahan Akter* | *Gustavo Braynner* |

***Date: 21/11/2014***

**Table of Contents**

[Introduction 3](#_Toc404339151)

[Class Diagram 3](#_Toc404339152)

[Class diagram description 3](#_Toc404339153)

[Simulation Class 3](#_Toc404339154)

[Crossing Class 5](#_Toc404339155)

[Road Class 6](#_Toc404339156)

[Lane Class 6](#_Toc404339157)

[LaneIn class 8](#_Toc404339158)

[LaneOut 8](#_Toc404339159)

[Car Class 8](#_Toc404339160)

[TrafficLight Class 10](#_Toc404339161)

[CrossWalk Class 10](#_Toc404339162)

[Pedestrains Class 11](#_Toc404339163)

[Sequence Diagram 12](#_Toc404339164)

[Add Crossing 12](#_Toc404339165)

[Run Simulator 12](#_Toc404339166)

[GUI in C# 12](#_Toc404339167)

# Introduction

In the design document we are going to make the class diagram that implement the use cases and the requirements we have in the URS document. This document will have the sequence diagrams that will help making the class diagram better.

# Class Diagram

[***The full picture is include in the same folder.***](Class%20Diagram.png)

## Class diagram description

### Simulation Class

Simulation is responsible for drawing crossings and Cars in the grid.

#### Properties:

**Crossing Map [12]:**

Array of crossings where crossing are stored in this order.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |

**String name:**

By default will be the same as a file

**int UpdateFrequency:**

**I**nfluence how often the screen is updated

**List Destinations <int[2]>:**

Destinations list with a parameter int array which stores all destinations for cars that haven't been assigned yet. (Still in the process of the running simulation)

#### Methods:

**Void AddCrossing(Int location, Int type):**

Add Crossing method with two parameter: location is the position on Map [12], type – type of crossing (type1, type2).

**Void RemoveCrossing(Int location):**

Remove crossing method with a parameter location - position on Map [12] which crossing will be deleted.

**Void RunSimulation():**

Deletes all statistics information from previous run and starts running a new simulation according by set up parameters.

**Void UpdateTrafficLights():**

Updates all traffic lights.

**Void UpdateCars():**

Updates all cars’ positions.

**Void UpdateCarsOnLanesOut():**

Updates all cars positions in all lanes out if necessary move cars object to another lanes in

**bool MoveCarsFromLaneOutToLaneIn(int[], int[]):**

Checks if there is a possibility for the car to move into a new lane (No traffic jam in that lane).

**Void StopSimulation():**

Stops simulation.

**Void SetInOutTraficFlow(int [], int value):**

**Location** in Map [12], int **road**-road in the crossing. (0-up, 1-right, 2-down, 3-left), **lane**- lane on the road, **value**- cars/min incoming cars or % of leaving cars.

**bool CheckSumOfOutFlow():**

If values set of leaving cars are greater or less than 100% will return false.

**Void UpdateOutFlowRelativly():**

Sets leaving cars values to be total of 100% relatively.

**Int MouseClickToLocation(point p):**

Returns location corresponding map [12], point p-mouse click.

**Void GenerateCars():**

Creates cars objects with destinations from List Destinations<int> and removes just used destination.

**Void GenerateDestinations():**

Generates destinations according to parameters set:

Sum of cars/min parameter (trafficInputCarsPerMinute) set in each LaneIn will determine total

number of destinations

Parameter (trafficOutProc) will determine percentage of destinations set to that Road and

LaneOut.

**Void Draw():**

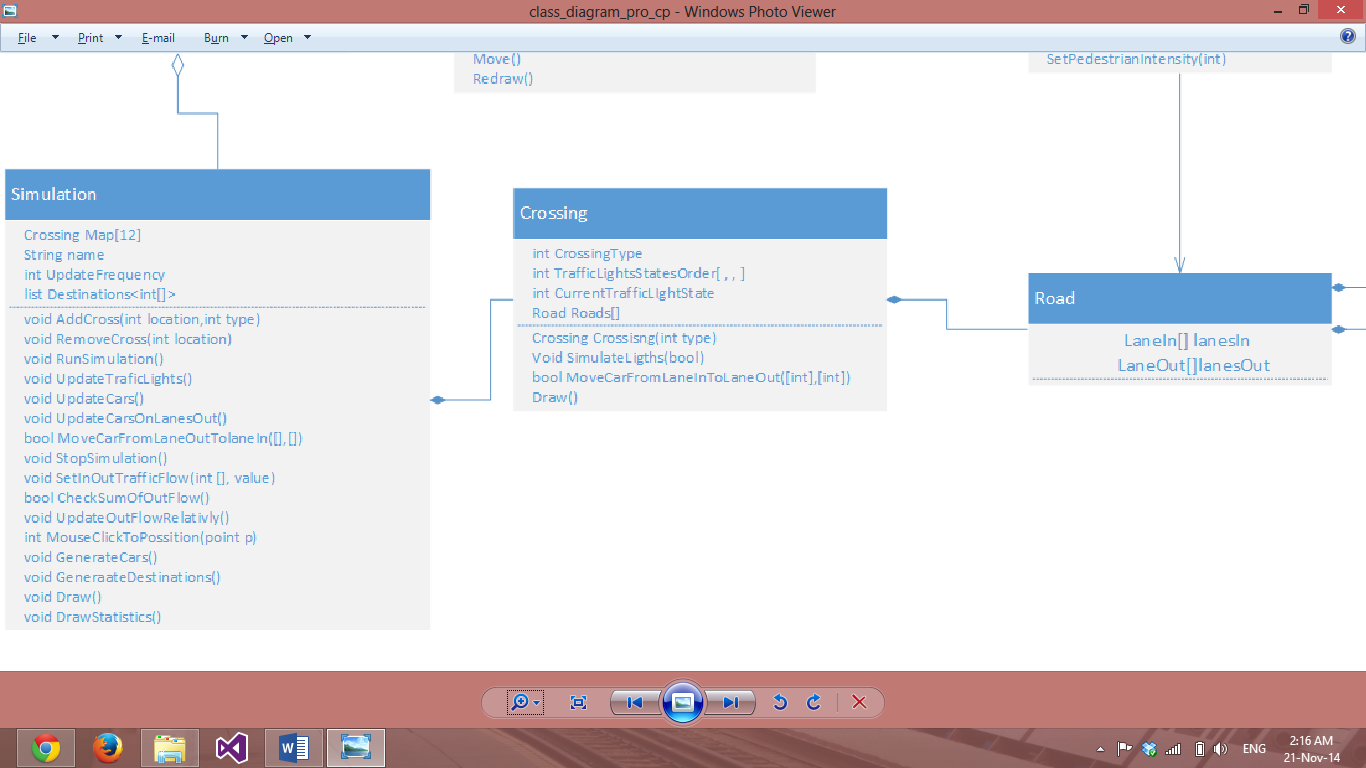
Draws everything on the screen;

**void DrawStatistics();**

Draws statistics on top.

### Crossing Class

#### Properties:

**int CrossingType:**

Crossing type (1-type1, 2-type2...)

**int TrafficLightStatesOrder[,,]:**

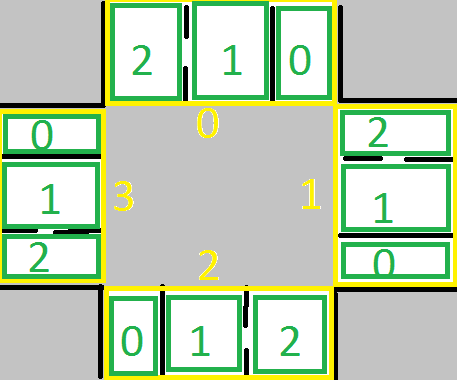
Returns the traffic light (green’s order).

Explanation of array dimensions:

1. Position = [road, laneIn];
2. TrafficLightSameTime = [[position],[position], etc.]
3. Order [[TrafficLightSameTime], [TrafficLightSameTime]]

**int CurrentTrafficLightState:**

States number in Traffic Light State Order.

**Road Roads[]:**

Contains array of objects roads (Yellow in the picture). Each road contains lanes (green in picture)

#### Methods:

**Crossing Crossing(int type):**

Constructor with parameter type Of Crossing (predefined crossing with number of roads and lanes crosswalks, sensors and default values).

**Void SimulateLights(bool):**

Updates traffic lights.

**bool MoveCarFromLaneInToLaneOut(int[address1], int[address2]):**

Checks if there is a possibility for the car to move from LaneIn Address1 laneOut Address2 (if No traffic jam in that lane).

**Draw():**

Draws crossing with traffic lights and all cars in it in the selected grid.

### Road Class

#### Properties:

**LaneIn[] lanesIn:**

An array that stores all lanes in.

**LaneOut[] lanesOut:**

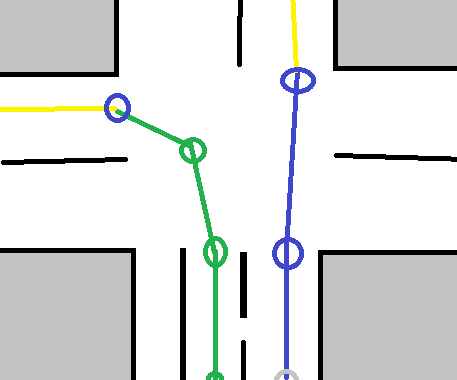
An array that stores all lanes out.

### Lane Class

#### Properties:

**int statistics[]:**

Total number of cars, cars last minute, average cars/min, maximum cars/min extra info for calculating statistics.

**LaneCurve list<line>**:

List of lines for graphical representation of car movement.  
Green lines belongs road-2, lane-1 and yellow to   
road-3, lane 0.

**List CarsOnLane<Car>;**

Contains all cars on the lane same order as they are on the lane.

#### Methods:

**Void AddCar(Car c):**

Adds a car into the lane list as lastPossition.

**Car GetCar():**

Returns a car.

**DeleteCar():**

Removes a car from the list.

**UpdateCars():**

Updates car in the lane.

**Void UpdateStatistics():**

Updates values on Statistics[].

**int GetCarsPerMinAverage():**

Returns Cars/Min from statistics[].

**int GetCarsPerMinMax();**

Returns Cars/Min maximum from statistics[]

**int GetDirectionAndTurn();**

Return direction and Turn code

0-Out lane  
1-Left

2-Left+Straight

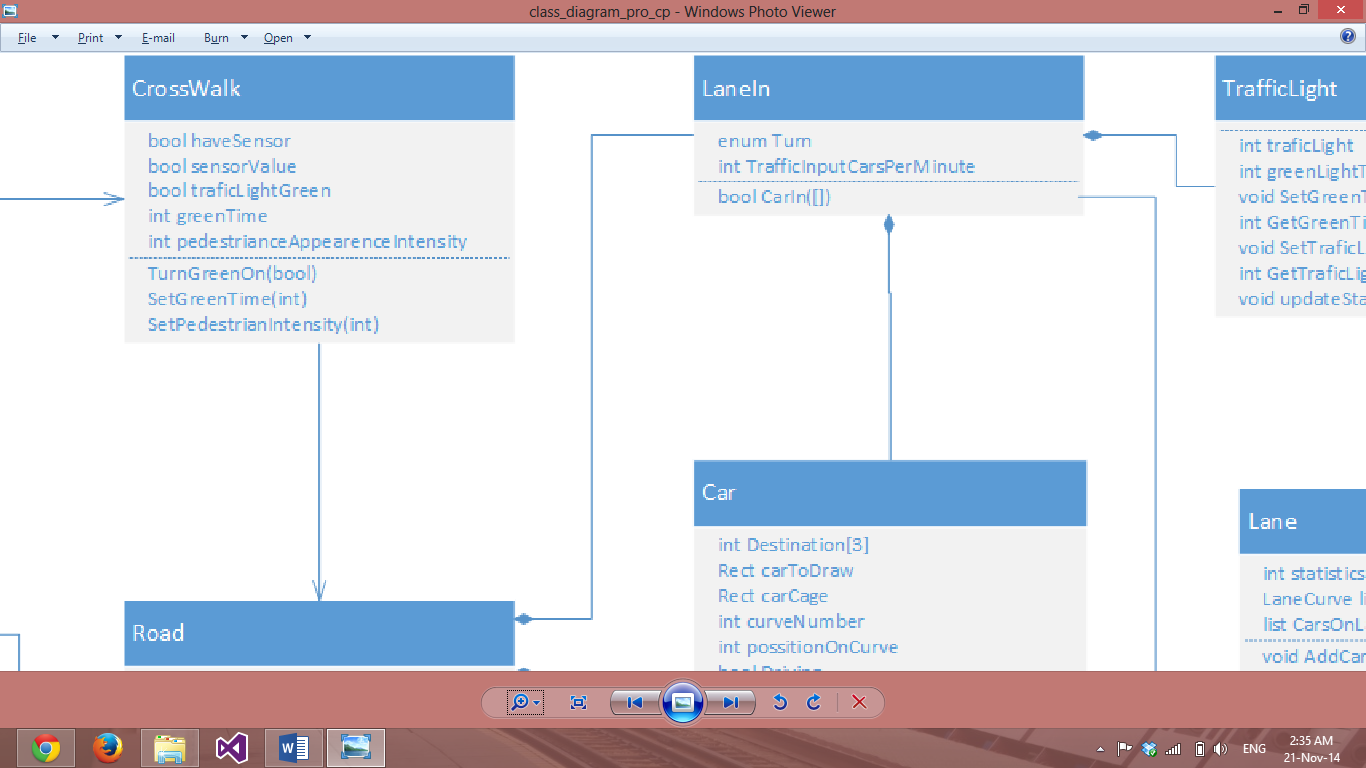
3-Straight

4-Straight+Right

5-Right

### LaneIn class

#### Properties:

**enum Turn:**

Stores Turns names:

0-Out lane  
1-Left

2-Left+Straight

3-Straight

4-Straight+Right

5-Right

**int TrafficInputCarsPerMinute:**

Stores the value of cars per minutes

#### Methods:

**bool CarIn(int[]):**

Checks if there is a car on the beginning of the lane, parameter is destination.

[crossing, road]

### LaneOut

#### Property:

**int TrafficOutProc:**

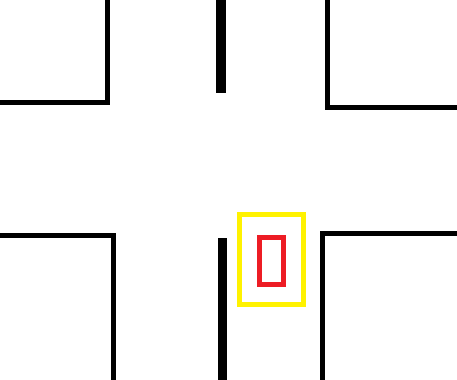
Stores the percentage of the traffic out in each lane set by the user. (If there road is connected with another road then the value is 0 and cannot be changed)

### Car Class

#### Properties:

**int Destination [3]:**

[Crossing, road, lane]

**Rect carToDrawn:**

Car represented as rectangular (red rectangular in the picture)

**Rect carCage:**

Rectangular around the car to calculate when cars get to close to each other (represented as yellow rectangular).

**Int curveNumber:**

Represents number on which curve car is from curves list (laneCurve list<line>)

**Int possitionOnCurve:**

Position on that curve.

**Bool Driving:**

True if car is moving.

#### Methods

**bool interactsWithObject(rect object);**

Return true if two cars cages intersects, or cars and pedestrians cage intersects.

**Rect GetCage();**

Return cage(yellow in the picture).

**void move(int distance)**

Move car by distance on the curve(from curveList), if after the move car is not in the same lane anymore, return error code(TBD) and car is copied to another lane and removed from this.

**Redraw();**

Car is drawn depending on its position in the lane.

### TrafficLight Class

#### Properties:

**int trafficLight:**

0-red 1-orange 2-green

**int greenLightTime:**

How long green will stay

**Methods:**

**Void SetGreenTime(int time):**

Sets greenLightTime if entered value is positive

**int GetGreenTime():**

Returns greenLightTime.

**Int GetTraficLight():**

Returns the color code()

0: red, 1: orange, 2: Green

**Void UpdateStatistics():**

Updates statistics.

### CrossWalk Class

#### Properties:

**bool haveSensor;**

True if have sensors.

**Bool sensorValue;**

Will be set and used if have Sensor is true, true if sensor is activated.

**bool trafficLightGreen**

True if traffic light is green for pedestrians.

**Int greenTime**

Green time.

**Int pedestriansAppearenceIntensity**

How many pedestrians will be generated per minute.

#### Methods:

**TurnGreenOn (bool value)**

Turns green on if true else turns red on.

**SetGreenTime(int)**

Sets green Time.

**SetPedestriansIntensity(int)**

Sets pedestrians Appearance Intensity.

### Pedestrains Class

#### Properties:

**int NumberOfPedestrians**

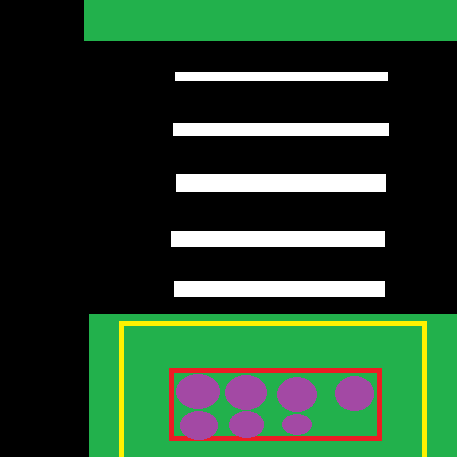
Number of pedestrains.

**Rect Pedestrians**

Pedestrians represented as circles and they move as one block (red rectangular).

**Rect pedestriansCage**

To calculate then car come to close then pedestrians is crossing. (Yellow in the picture)

**List Pedestrians<points>**

Represented as violet circles in red rectangular.

#### Methods:

**Rect GetCage();**

Returns pedestriansCage

**void RandomizeNextPedestrian();**

Depending set intensity sets timer to generate next pedestrians(add extra dot and activate sensor if one exist)

**move();**

Then tragic light is green, pedestrians stars to move and disappears then they are finished crossing the road.

**Redraw()**

Draw pedestrians according on their position.

# Sequence Diagram

## Add Crossing

[***In a picture included with this folder***](Add%20Crossing%20Sequence%20Diagram.png)

## Run Simulator

[***In a picture included with this folder***](Run%20Simulation%20Sequence%20Diagram.png)

# GUI in C#

