Project Report

Distributed Control Systems

Authors: **Trif Diana-Nicoleta (30343)**

**Oprea Sergiu-Daniel (30343)**

**Pleșca Evelyn-Iulia (30342)**

Instructor: **As. Dr. Ing. Dahlia AL-JANABI**

Cuprins

[1 Introduction 2](#_Toc156083827)

[1.1 General Context 2](#_Toc156083828)

[1.2 Objectives 4](#_Toc156083829)

[2 Intersections & Trafic behaviour 5](#_Toc156083830)

[2.1 Petri Nets 5](#_Toc156083831)

[2.2 OETPN model 7](#_Toc156083832)

[3 Intersection Trafic Controllers 11](#_Toc156083833)

[3.1 Relationship with the intersections 11](#_Toc156083834)

[3.2 OETPN model 11](#_Toc156083835)

[4 Implementation 12](#_Toc156083836)

[4.1 Controller\_Intersection2 class 12](#_Toc156083837)

[4.2 Intersection1 class 12](#_Toc156083838)

[4.3 Intersection2 class 13](#_Toc156083839)

[4.4 MiddleStreet class 13](#_Toc156083840)

[5 Testing 15](#_Toc156083841)

[5.1 Test 1 15](#_Toc156083842)

[5.2 Test 2 – Traffic Jam 17](#_Toc156083843)

# Introduction

## General Context

The purpose of this project is modelling and controlling two real-life intersections (our Plant) randomly assigned per team with the use of OETPNs which communicate via network.

The intersections which were modelled in our project are: <https://maps.app.goo.gl/BaDmMauP535nbKxy9> and <https://maps.app.goo.gl/5ntoHkVLeHPCPEFp6>.

O imagine care conține text, hartă

Descriere generată automat

Figure 1 - Both intersections and the middle road connecting them, as seen on Google Maps

O imagine care conține captură de ecran, text

Descriere generată automat

Figure 2 - First intersection, as seen on Google Maps

O imagine care conține text, captură de ecran, hartă

Descriere generată automat

Figure 3 - Second intersection, as seen on Google Maps

O imagine care conține captură de ecran, șubler, cântar

Descriere generată automat

Figure 4 – Simplified map, graphical model of the two intersections and the middle lane that connects them

The Intersections & Traffic modelling chapter includes the OETPN model, with all the places, transitions, guards and maps included in the project and diagrams and Petri Nets that we modelled after our system.

Intersection Traffic Controllers includes the controller used for modelling the behaviour of the traffic lights.

Implementation includes code snippets from the project.

Logs and Screenshots include the logs and the screenshots which prove the functionality of our project, as seen after the testing of the application.

## Objectives

Our objectives for this projects were simplifying the real-life intersections, providing a graphical representation of them and modelling the intersections using Petri nets and Java and the OETPN Framework provided during the laboratories.

Our Plant will have input and output channels used for modelling it, as well as a controller which interacts with the Plant and determines the evolution of the system as required by the specification requirements. The model is created afterwards, formally describing the required behaviour of the Plant.

# Intersections & Trafic behaviour

## Petri Nets

Below are attached the Petri Nets representing the intersections.

O imagine care conține schiță, diagramă, hartă

Descriere generată automat

Figure 5 - Intersection 1 modelled using Petri Nets

O imagine care conține schiță, desen, diagramă, hartă

Descriere generată automat

Figure 6 - Intersection 2 modelled using Petri Nets

O imagine care conține diagramă, linie

Descriere generată automat

Figure 7 - Middle road, modelled using Petri Nets

## OETPN model

In order to model the behaviour of the traffic, we chose the following places and transitions for Intersection 1.

Places:

* Data Car types: P\_a1, P\_a2, P\_a3, P\_a4, P\_a5, P\_o1e, P\_o2e, P\_o3e, P\_o4e, P\_o5e, P\_4I
* Data Transfer type: P\_4E
* Data Car Queue type: P\_o1, P\_o2, P\_o3, P\_o4, P\_o5, P\_x1, P\_x2, P\_x3, P\_x4, P\_x5, P1, P2, P3, P4, P5

Transitions:

* t1:
  + Input places: P1
  + grd\_1: (m(P1), HaveCarForMe) AND (m(P2), CanAddCars)
  + map\_1: m(P1), PopElementWithTargetToQueue (P2)
* t2:
  + Input places: P2
  + grd\_2: (m(P2), HaveCarForMe) AND (m(P3), CanAddCars)
  + map\_2: m(P2), PopElementWithTargetToQueue (P3)
* t3:
  + Input places: P3
  + grd\_3: (m(P3), HaveCarForMe) AND (m(P4), CanAddCars)
  + map\_3: m(P3), PopElementWithTargetToQueue (P4)
* t4:
  + Input places: P4
  + grd\_4: (m(P4), HaveCarForMe) AND (m(P5), CanAddCars)
  + map\_4: m(P4), PopElementWithTargetToQueue (P5)
* t5:
  + Input places: P5
  + grd\_5: (m(P5), HaveCarForMe) AND (m(P1), CanAddCars)
  + map\_5: m(P5), PopElementWithTargetToQueue (P1)
* t\_i1:
  + Input places: P1, P\_x1
  + grd\_6: (m(P\_x1), HaveCarForMe) AND (m(P1), CanAddCars)
  + map\_6: (m(P\_x1), PopElementWithTargetToQueue (P1))
* t\_i2:
  + Input places: P2, P\_x2
  + grd\_7: (m(P\_x2), HaveCarForMe) AND (m(P2), CanAddCars)
  + map\_7: (m(P\_x2), PopElementWithTargetToQueue (P2))
* t\_i3:
  + Input places: P3, P\_x3
  + grd\_8: (m(P\_x3), HaveCarForMe) AND (m(P3), CanAddCars)
  + map\_8: (m(P\_x3), PopElementWithTargetToQueue (P3))
* t\_i4:
  + Input places: P4, P\_x4
  + grd\_9: (m(P\_x4), HaveCarForMe) AND (m(P4), CanAddCars)
  + map\_9: (m(P\_x4), PopElementWithTargetToQueue (P4))
* t\_i5:
  + Input places: P5, P\_x5
  + grd\_10: (m(P\_x5), HaveCarForMe) AND (m(P5), CanAddCars)
  + map\_10: (m(P\_x5), PopElementWithTargetToQueue(P5))
* t\_o1:
  + Input places: P\_o1, P1
  + grd\_11: (m(P1), HaveCarForMe) AND (m(P\_o1), CanADDCars)
  + map\_11: (m(P1).PopElementWithTargetToQueue(m(P\_o1))
* t\_o2:
  + Input places: P\_o2, P2
  + grd\_12: (m(P2), HaveCarForMe) AND (m(P\_o2), CanADDCars)
  + map\_12: (m(P2).PopElementWithTargetToQueue(m(P\_o2))
* t\_o3:
  + Input places: P\_o3, P3
  + grd\_13: (m(P3), HaveCarForMe) AND (m(P\_o3), CanADDCars)
  + map\_13: (m(P3).PopElementWithTargetToQueue(m(P\_o3))
* t\_o4:
  + Input places: P\_o4, P4
  + grd\_14: (m(P4), HaveCarForMe) AND (m(P\_o4), CanADDCars)
  + map\_14: (m(P4).PopElementWithTargetToQueue(m(P\_o4))
* t\_o5:
  + Input places: P\_o5, P5
  + grd\_15: (m(P5), HaveCarForMe) AND (m(P\_o5), CanADDCars)
  + map\_15: (m(P5).PopElementWithTargetToQueue(m(P\_o5))
* t\_a1:
  + Input places: P\_a1, P\_x1
  + grd\_16: (m(P\_a1) ≠ Ø) AND (m(P\_x1), CanAddCars)
  + map\_16: (m(P\_a1).addElement(m(p\_x1))
* t\_a2:
  + Input places: P\_a2, P\_x2
  + grd\_17: (m(P\_a2) ≠ Ø) AND (m(P\_x2), CanAddCars)
  + map\_17: (m(P\_a2).addElement(m(p\_x2))
* t\_a3:
  + Input places: P\_a3, P\_x3
  + grd\_18: (m(P\_a3) ≠ Ø) AND (m(P\_x3), CanAddCars)
  + map\_18: (m(P\_a3).addElement(m(p\_x3))
* t\_a4:
  + Input places: P\_a4, P\_x4
  + grd\_19: (m(P\_a4) ≠ Ø) AND (m(P\_x4), CanAddCars)
  + map\_19: (m(P\_a4).addElement(m(p\_x4))
* t\_a5:
  + Input places: P\_a5, P\_x5
  + grd\_20: (m(P\_a5) ≠ Ø) AND (m(P\_x5), CanAddCars)
  + map\_20: (m(P\_a5).addElement(m(p\_x5))
* t\_o1e:
  + Input places: P\_o1
  + grd\_21: (m(P\_o1).HaveCar)
  + map\_21: m(P\_o1).PopElementWithoutTarget(m(P\_o1e))
* t\_o2e:
  + Input places: P\_o2
  + grd\_22: (m(P\_o2).HaveCar)
  + map\_22: m(P\_o2).PopElementWithoutTarget(m(P\_o2e))
* t\_o3e:
  + Input places: P\_o3
  + grd\_23: (m(P\_o3).HaveCar)
  + map\_23: m(P\_o3).PopElementWithoutTarget(m(P\_o3e))
* t\_o4e:
  + Input places: P\_o4
  + grd\_24: (m(P\_o4).HaveCar)
  + map\_24: m(P\_o4).PopElementWithoutTarget(m(P\_o4e))
* t\_o5e:
  + Input places: P\_o5
  + grd\_25: (m(P\_o5).HaveCar)
  + map\_25: m(P\_o5).PopElementWithoutTarget(m(P\_o5e))
* t\_4I:
  + Input place: P\_41
  + grd\_26: (m(P\_4I) ≠ Ø) AND (m(P\_a4).CanAddCars)
  + map\_26: (m(P\_4I).addElement(m(p1))
* t\_4E:
  + Input place: P\_o4e
  + grd\_27: (m(P\_o4E) ≠ Ø)
  + map\_27: m(P\_o4E).SendOverNetwork(), m(P\_4E) = m(P\_4E)

Middle street:

Places:

* Data Car types: P5, P6, P10, P\_2E, P9, P\_4E
* Data Transfer type: P\_4I, P\_2I
* Data Car Queue type: P2, P3, P8

Transitions:

* t1:
  + Input places: P2
  + grd\_1: (m(P2), HaveCar))
  + map\_1: m(P2), PopElementWithoutTarget(P10)
* tt:
* Input places: P10
  + grd\_2: (m(P10), NotNull))
  + map\_2: m(P10).SendOverNetwork, m(P\_4I) = m(P\_4I)
* t2:
  + Input places: P3
  + grd\_3: (m(P3), HaveCarForMe AND m(P2), CanAddCars))
  + map\_3: m(P3), PopElementWithTargetToQueue(P2)
* t3:
* Input places: P\_2E
  + grd\_4: (m(P\_2E), NotNull AND m(P3), CanAddCars))
* map\_4: m(P\_2E).AddElement(m(P3))
* t4:
* Input places: P5
  + grd\_5: (m(P5), NotNull AND m(P2), CanAddCars))
* map\_5: m(P5).AddElement(m(P2))
* t5:
* Input places: P3
* grd\_6: (m(P3), HaveCar))
  + map\_6: m(P3), PopElementWithoutTarget(P6)
* t6:
  + Input places: P\_4E
  + grd\_7: (m(P\_4E), NotNull AND m(P8), CanAddCars))
  + map\_7: m(P\_4E).AddElement(m(P8))
* t7:
* Input places: P8
* grd\_8: (m(P8), HaveCar))
* map\_8: m(P8), PopElementWithoutTarget(P9)
* t8:
* Input places: P9
* grd\_9: (m(P9), NotNull)
* map\_9: m(P9).SendOverNetwork, m(P\_2I) = m(P\_2I)

# Intersection Trafic Controllers

## Petri Nets

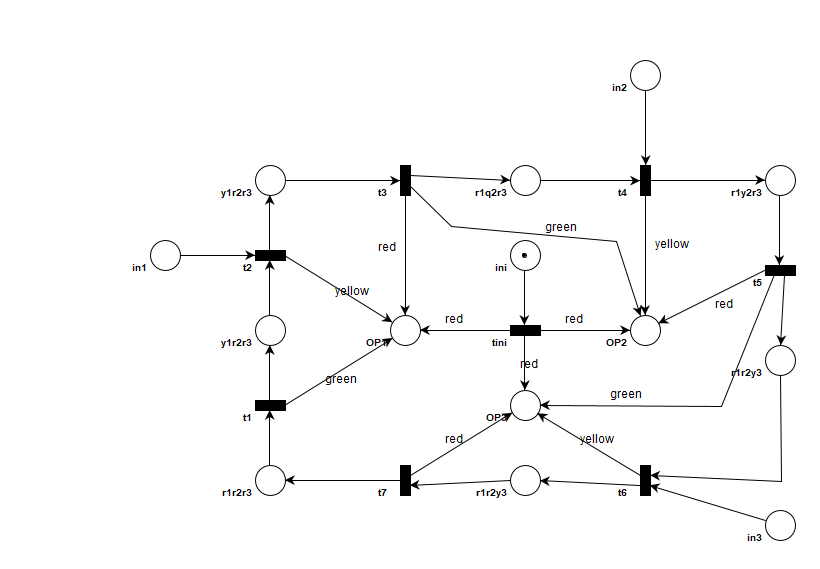


Figure 8 - Controller, modelled using Petri Nets

In intersection 3, there are 3 traffic lights at the exit of the roundabout, which can be seen as ports: both the Controller and Intersection 2 have 6 ports. Controller has 3 input and 3 output ports and Intersection 2 has 3 output ports and 3 traffic light ports.

O imagine care conține text, diagramă, linie, Plan

Descriere generată automat

Figure 9 - Component diagram of the entire system

The second intersection has a closed-loop traffic controller that communicates with the model of the second intersection, as seen in *Figure 9* above.

## OETPN model

Places:

* Data String types: ini, red, green, yellow, in1, in2, in3, r1r2r3, g1r2r3, r1g2r3, r1r2g3, r1r2y3, y1r2r3, r1y2r3
* Data Integer: Five, Ten
* Data Transfer type: OP1, OP2, OP3

Transitions:

* tini:
  + Input places: ini
  + grd\_1: (m(ini) ≠ Ø)
  + map\_1: m(ini).SendOverNetwork(), m(OP1) = m(OP1),

m(ini).SendOverNetwork(), m(OP2) = m(OP2),

m(ini).SendOverNetwork(), m(OP3) = m(OP3),

m(ini).MakeNull(ini)

* t1:
* Input places: r1r2r3
* grd\_2: (m(r1r2r3) ≠ Ø)
* map\_2: m(r1r2r3).Move(m(g1r2r3)),

m(r1r2r3).SendOverNetwork(), m(OP1) = m(OP1)

* t2:
* Input places: g1r2r3, in1
* grd\_3: (m(g1r2r3) ≠ Ø AND m(in1) = Ø)
* map\_3: m(g1r2r3).Move(m(y1r2r3)),

yellow.SendOverNetwork(), m(OP1) = m(OP1),

Five.DynamicDelay()

* grd\_4: (m(g1r2r3) ≠ Ø AND m(in1) ≠ Ø)
* map\_4: m(g1r2r3).Move(m(y1r2r3))

yellow.SendOverNetwork(), m(OP1) = m(OP1),

Ten.DynamicDelay()

* t3:
* Input places: y1r2r3
* grd\_5: (m(y1r2r3) ≠ Ø)
* map\_5: m(y1r2r3).Move(m(r1g2r3))

red.SendOverNetwork(), m(OP1) = m(OP1),

green.sendOverNetwork(), m(OP2) = m(OP2)

* t4:
* Input places: r1g2r3, in2
* grd\_6: (m(r1g2r3) ≠ Ø AND m(in2) = Ø)
* map\_6: m(r1g2r3).Move(m(r1y2r3)),

yellow.SendOverNetwork(), m(OP2) = m(OP2),

Five.DynamicDelay()

* grd\_7: (m(r1g2r3) ≠ Ø AND m(in2) ≠ Ø)
* map\_7: m(r1g2r3).Move(m(r1y2r3))

yellow.SendOverNetwork(), m(OP2) = m(OP2),

Ten.DynamicDelay()

* t5:
* Input places: r1y2r3
* grd\_5: (m(r1y2r3) ≠ Ø)
* map\_5: m(r1y2r3).Move(m(r1r2g3))

red.SendOverNetwork(), m(OP1) = m(OP2),

green.sendOverNetwork(), m(OP2) = m(OP3)

* t6:
* Input places: r1r2g3, in3
* grd\_6: (m(r1r2g3) ≠ Ø AND m(in3) = Ø)
* map\_6: m(r1r2g3).Move(m(r1r2y3)),

yellow.SendOverNetwork(), m(OP3) = m(OP3),

Five.DynamicDelay()

* grd\_7: (m(r1r2g3) ≠ Ø AND m(in3) ≠ Ø)
* map\_7: m(r1r2g3).Move(m(r1r2y3))

yellow.SendOverNetwork(), m(OP3) = m(OP3),

Ten.DynamicDelay()

* t7:
* Input places: r1r2y3
* grd\_5: (m(r1r2y3) ≠ Ø)
* map\_5: m(r1r2y3).Move(m(r1r2r3))

red.SendOverNetwork(), m(OP3) = m(OP3)

# Implementation

For our implementation, we extended the functionality of the existing OETPN framework and added our own classes which were needed for solving the given problem.

We added the Controller\_Intersection2, Intersection1, Intersection2 and MiddleStreet classes. All of these are added into the FINAL\_PROJECT folder.

We used Github for source control and the implementation of our project can be found at the following repository link: <https://github.com/DianaT08/DCS_Project>.

## Controller\_Intersection2 class

We implemented the controller that communicates with the second intersection in this class.

We implemented the transitions presented in Chapter 2 of this document here.

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Figure 0 - Screenshot from the implementation of Controller\_Intersection2 class in IntelliJ

## Intersection1 class

We implemented all the transitions related to the first intersection in the Intersection1 class. We added comments in the code in order to separate the code implementation of the places and transitions related to lanes, exit lanes and the intersection.

O imagine care conține captură de ecran, text, software, Software multimedia

Descriere generată automat

Figure 11 - Screenshot from the implementation of Intersection1 class in IntelliJ

## Intersection2 class

We implemented the second intersection in this class, in similar manner to the implementation of the first intersection.

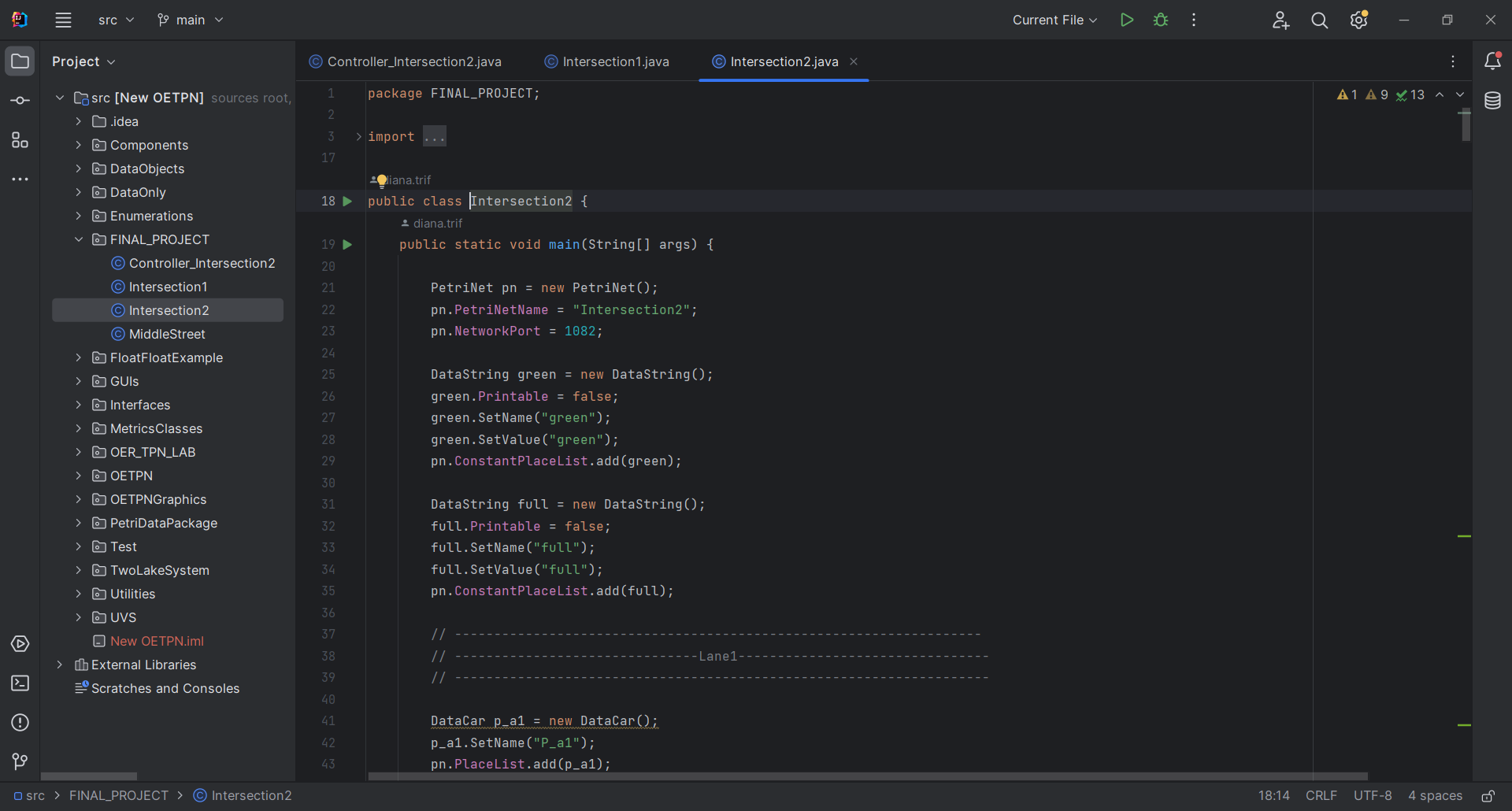


Figure 1 - Screenshot from the implementation of Intersection2 class in IntelliJ

## MiddleStreet class

The middle street connecting the two intersections was implemented in this class.

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Figure 1 - Screenshot from the implementation of Intersection2 class in IntelliJ

# Testing

## Test 1

For this case, the car entered lane 1 from intersection 1, exited through lane exit 4, entered the middle street, after which it entered intersection 2 through lane 2 and exited through lane 4.

Screenshots of the execution can be found below.

O imagine care conține text, electronice, captură de ecran, software

Descriere generată automat

Figure 1 - Execution screenshot of test 1

O imagine care conține text, captură de ecran, număr, software

Descriere generată automat

Figure 1 - Execution screenshot of test 1

O imagine care conține text, electronice, captură de ecran, afișaj

Descriere generată automat

Figure 1 - Execution screenshot of test 1

O imagine care conține text, captură de ecran, software, afișaj

Descriere generată automat

Figure 1 - Execution screenshot of test 1

## Test 2 – Traffic Jam

For each intersection, we created a traffic jam case by sending the maximum number of cars to the input lane of the intersection, started the controller, then sent the last car.

The controller received a signal from the plant (intersection) and the transition that responsible for sending a yellow light to the lane where the cars have been input changed the delay to 10 sec.

The controller OETPN ran until it reached the same transition (2 loops) to show that the delay is changed back to 5 sec.

Screenshots from the logs saved in this moment can be seen below.

O imagine care conține text, alb și negru, Font, hârtie

Descriere generată automat

Figure 1 - Screenshot from the log saved after executing test 2

O imagine care conține text, alb și negru, document, monocrom

Descriere generată automat

Figure 1 - Screenshot from the log saved after executing test 2