LA Team: DCS Project Report

Year IV, AIA English, Group 30342&30341

Students: Malita Alin, Tudor Ada, Mosnegutu Vlad

Table of Contents

[1. Introduction 1](#_Toc155774899)

[1.1 Premise and requirements 1](#_Toc155774900)

[1.2 Specifications 2](#_Toc155774901)

[2. Design 3](#_Toc155774902)

[2.1 Intersections - Transitions 3](#_Toc155774903)

[2.3 Controllers - Transitions 7](#_Toc155774904)

[2.4 Component diagram 9](#_Toc155774905)

[3. Implementation 9](#_Toc155774906)

[4. Testing 9](#_Toc155774907)

[4.1 Test 1 9](#_Toc155774908)

[4.2 Test 2 11](#_Toc155774909)

# Introduction

## Premise and requirements

The following report represents the detailed description of the project of team LA, involving two given intersection which require control through a traffic light system. This project is composed of 4 main sections:

1. Specifications: the illustration of the subject intersections through map and drawings.

2. Design: presentation of the OETPN models for the plants, controllers and connection street, including place types and guard maps. This section also includes the component diagram of the system.

3. Implementation: Git repository link to all the components of the project, especially the code and the activity history of the team members.

4. Testing: The demonstration of the functionality of the OETPN models and control through testing 2 requested scenarios

## 1.2 Specifications

Figure 1.2 Intersection 1 (Pin1)

Figure 1.4 Intersection 2 (Pin2)

Figure 1.3 Connection Street

Figure 1.1 Google Earth view

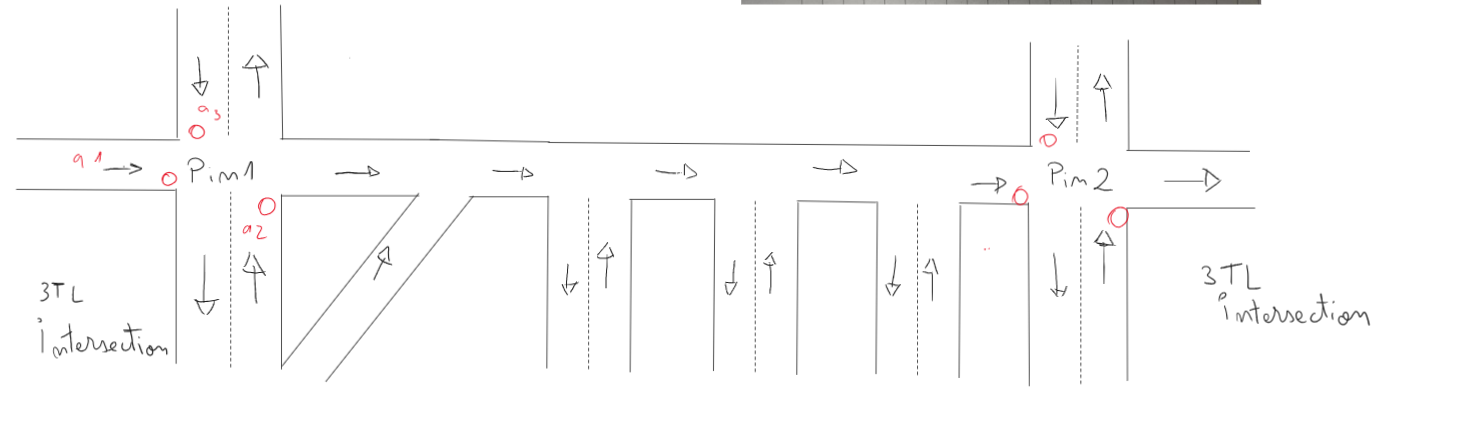


Figure 1.5 Illustrated Intersection System

Based on the given Intersections, the prospect of the project is the representation, modelling and control of the intersection using the OETPN\_OERTPN java framework. Below the system is simplified through illustration fig1.5, the red dots representing the position of control places TL (traffic lights) which will dictate the flow of traffic inside the intersections. Both intersection have similar forms and require 3 control points. The traffic lights of the connection street are ignored.

# 2. Design

## 2.1 Intersections - Transitions

Since both intersections are extremely similar, their transitions are the same, with the exception of transition t\_4n and place P\_4N present only for Intersection1.

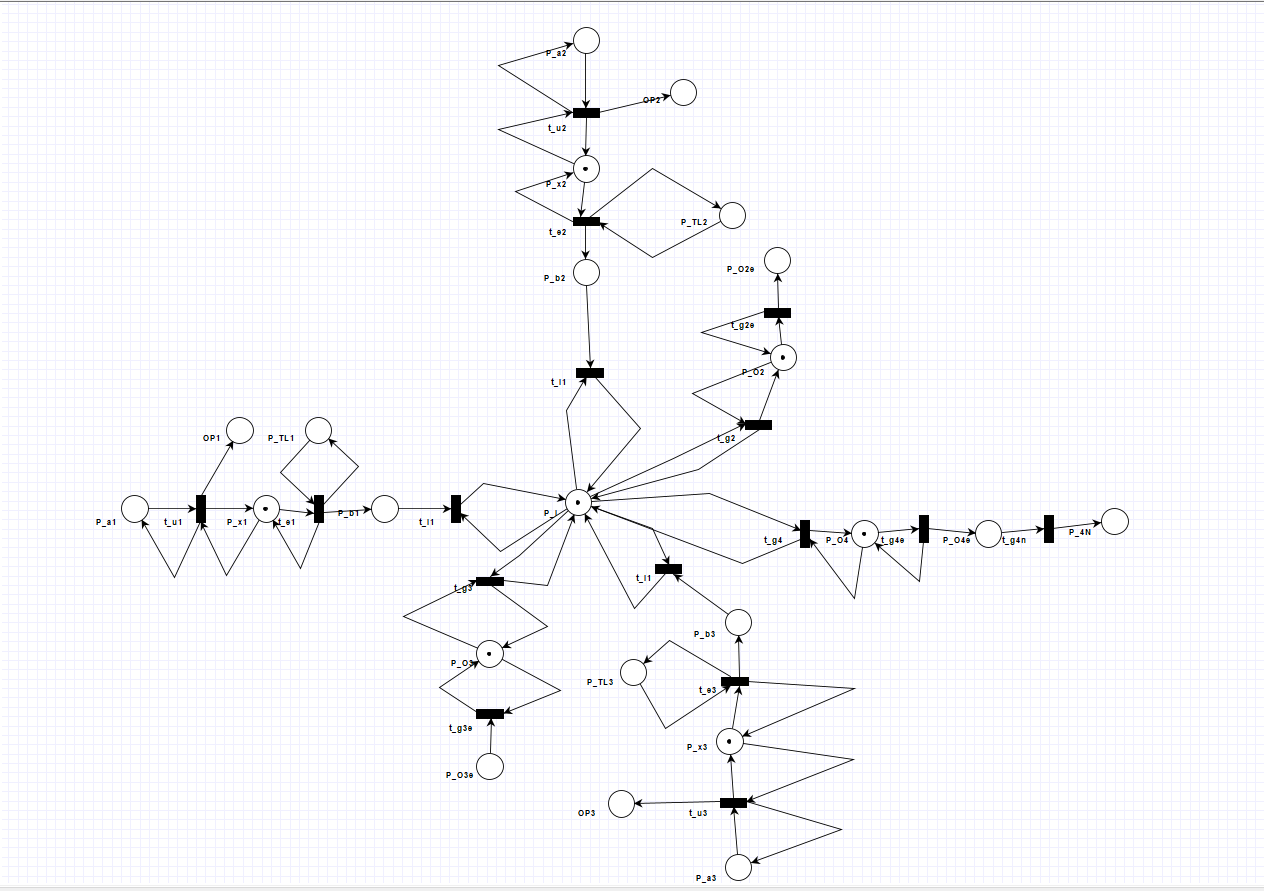


Figure 2.1.1 Intersection 1 Petri Net

**PLACES:**

* P\_\_a1, P\_a2, P\_a3, P\_b1,P\_b2, P\_b3, P\_o2Exit, P\_o3Exit, P\_o4Exit = DataCar type
* OP1, OP2, OP3, P\_4N = DataTransfer type
* P\_x1, P\_x2, P\_x3, P\_I, P\_o2, P\_o3, P\_o4 = DataCarQueue type
* P\_TL1, P\_TL2, P\_TL3 = DataString type

**TRANSITIONS**:

* t\_u1: input place: P\_a1, P\_x1

grd1: (m(P\_a1) ≠ ϕ And m(P\_x1).CanAddCars) );

map1: m(P\_a1).addElement() (m(P\_x1))

grd2: (m(P\_a1) ≠ ϕ And m(P\_x1).CanNotAddCars) );

map2: m(P\_a1).Move() (m(P\_a1)) ; m(OP1) .SendOverNetwork(full)

same logic applies to t\_u2 and t\_u3

* t\_e1: input place: P\_x1, P\_TL1

grd: (m(P\_x1).HaveCar And m(P\_TL1)=green );

map: m(P\_x1).PopElementWithoutTarget() (m(P\_b1)); m(P\_TL1).Move() m(P\_TL1)

same logic applies to t\_e2 and t\_e3

* t\_i1: input place: P\_\_b1, P\_I

grd: (m(P\_b1) ≠ ϕ And m(P\_I).CanAddCars) );

map: m(P\_b1).AddElement() (m(P\_I))

same logic applies to t\_i2 and t\_i3

* t\_g2: input place: P\_I, P\_o2

grd: (m(P\_I).HaveCar And m(P\_o2).CanAddCars) );

map: m(P\_I).PopELementWithTargetToQueue() (m(P\_o2))

same logic applies to t\_g3 and t\_g4

* t\_g2e: input place: P\_o2

grd: (m(P\_o2).HaveCar) );

map: m(P\_o2).PopElementWithoutTarget() (m(P\_o2Exit))

same logic applies to t\_g3e and t\_g4e

* t\_g4N: input place: P\_04e

grd: (m(P\_04e) ≠ ϕ);

map: m(P\_04e).SendOverNetwork() (m(P\_4N) = m(P\_4N)

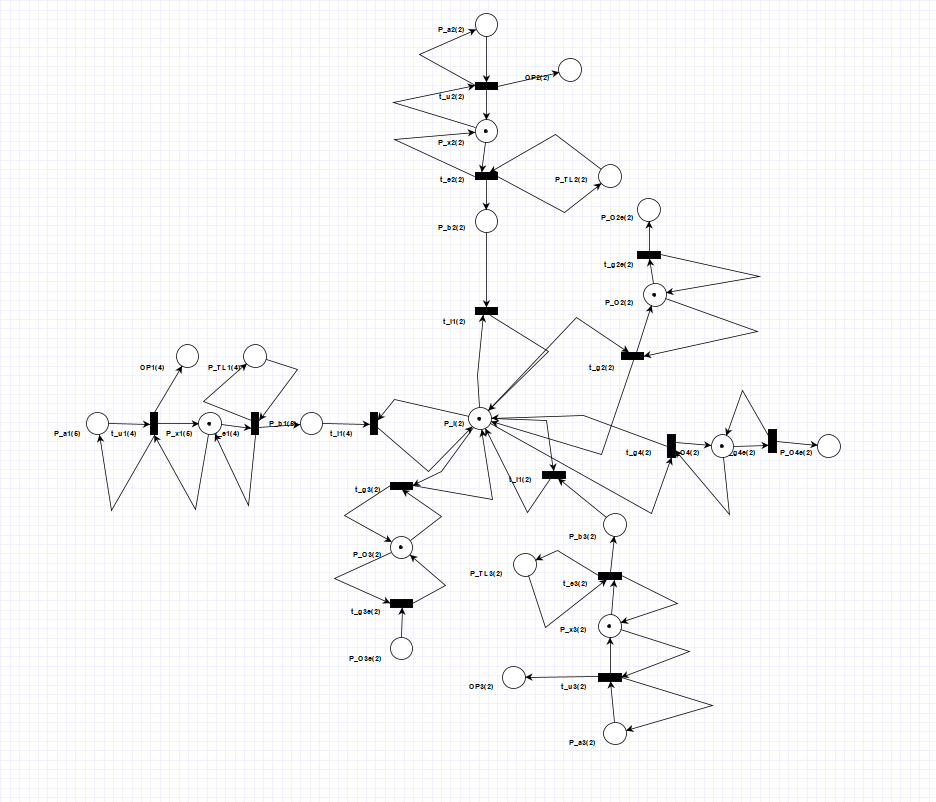


Figure 2.1.2 Intersection 2 Petri Net

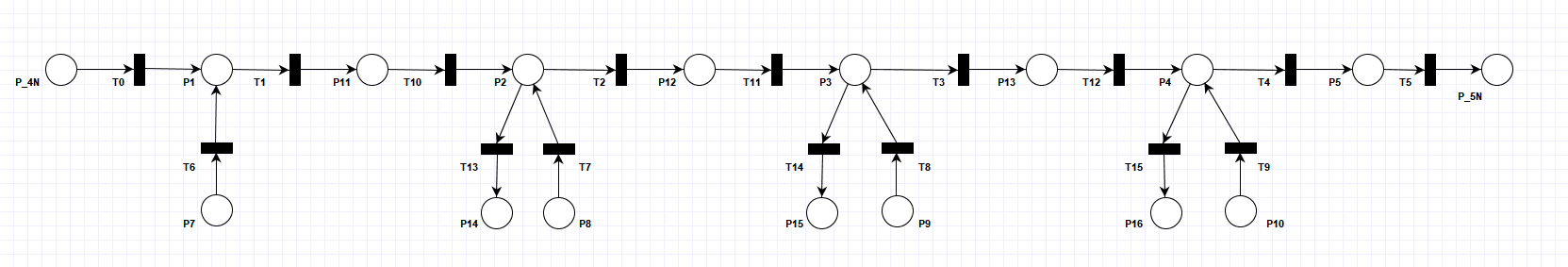
Connecting Street – Transitions

Figure 2.2.1 Connection Street PetriNet

**PLACES:**

* P\_4N, P5, P7, P8, ….P15,P16 = DataCar type
* P\_5N = DataTransfer type
* P1, P2, P3, P4 = DataCarQueue type

**TRANSITIONS:**

* First Group of transitions
* t0 : input place: P\_4N

grd: (m(P\_4N) ≠ ϕ And m(p1).CanAddCars) );

map: m(P\_4N).addElement() (m(p1))

* t6 : input place: p7

grd: (m(p7) ≠ ϕ And m(p1).CanAddCars) );

map: m(p7).addElement() (m(p1))

* t7 : input place: p8

grd: (m(p8) ≠ ϕ And m(p2).CanAddCars) );

map: m(p8).addElement() (m(p2))

* t8 : input place: p9

grd: (m(p9) ≠ ϕ And m(p3).CanAddCars) );

map: m(p9).addElement() (m(p3))

* t9 : input place :p10

grd: (m(p10) ≠ ϕ And m(p4).CanAddCars) );

map: m(p10).addElement() (m(p4))

t10, t11 and t12 will follow the same logic as the t0 ->t9 transitions

* Second Group
* t1 : input place: p1

grd: (m(p1).HaveCarForMe);

map: m(p1).PopElementWithTarget() (m(p11))

* t2 : input place: p2

grd: (m(p2).HaveCarForMe);

map: m(p2).PopElementWithTarget() (m(p12))

* t3 : input place: p3

grd: (m(p3).HaveCarForMe);

map: m(p3).PopElementWithTarget() (m(p13))

* t13 : input place: p2

grd: (m(p2).HaveCarForMe);

map: m(p2).PopElementWithTarget() (m(p14))

* t14 : input place: p3

grd: (m(p3).HaveCarForMe);

map: m(p3).PopElementWithTarget() (m(p15))

transitions t1,t2,t3,t4 and t13,t14,t15 all have similar logic

* Transition t5: input place: p5

grd: (m(p5) ≠ ϕ);

map: m(p5).SendOverNetwork() (m(p5n) = m(p\_a1)

## 2.3 Controllers - Transitions

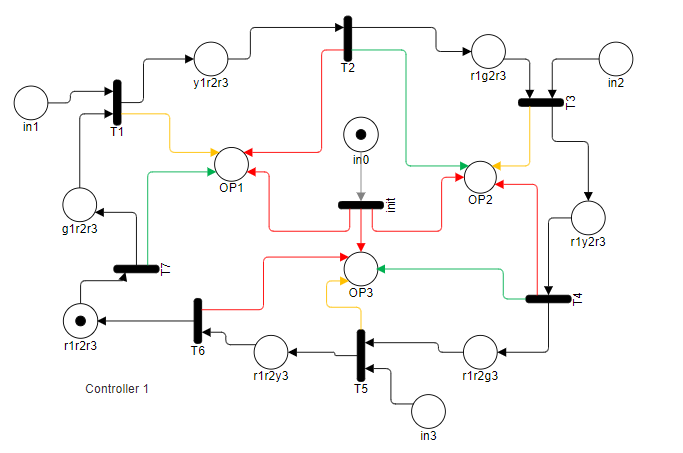
Since the intersections are so similar, their controllers will have the same logic for their transitions and places

Figure 2.3.1 Controller 1 (same as Controller 2) Petri Net

**PLACES:**

* in0, in1, in2, in3 = DataString type
* OP1, OP2, OP3 = DataTransfer type
* r1r2r3,g1r2r3,y1r2r3,r1g2r3,r1y2r3,r1r2g3,r1r2y3 = DataString type

**TRANSITIONS:**

* init: input Place: in0

grd: (m(in0) ≠ ϕ);

map: m(in0).MakeNull; m(OP1).SendOverNetwork(in0)

m(OP2).SendOverNetwork(in0)

m(OP3).SendOverNetwork(in0)

* First Group of transitions
* T1 : input place: in1, g1r2r3

grd1: (m(in1)= ϕ And (m(g1r2r3) ≠ ϕ) );

map1: m(g1r2r3).Move() (m(y1r2r3))

m(OP1).SendOverNetwork(yellow)

DynamicDelay(Five)

grd2: (m(in1) ≠ ϕ And (m(g1r2r3) ≠ ϕ) );

map2: m(g1r2r3).Move() (m(y1r2r3))

m(OP1).SendOverNetwork(yellow)

DynamicDelay(Ten)

t1, t3 and t5 follow the same logic

* Second Group of transitions
* t2 : input place: y1r2r3

grd: (m(y1r2r3) ≠ ϕ);

map: m(y1r2r3).Move() (m(r1g2r3));

m(OP1).SendOverNetwork(red)

m(OP2).SendOverNetwork(green)

t2, t4 and t6 follow the same logic

## 2.4 Component diagram

Figure 2.4.1 Component Diagram

As explained in fig.2.4.1, controllers and intersections communicate through a series of input and outputs, sending signals on which the evolution of the system depends. The connection street simply receives cars from Intersection\_1 and allows them to travel into Intersection\_2.

# Implementation

* Inside the [Git repository](https://github.com/MalitaAlin/LA_DCS_Project), the location of the project file can be found at the following address: OETPN\_OERTPN\_Framework/New OETPN/New OETPN/src/ProjectLA
* Testing Folder offers all images and log files related to the 2 tests required for the project’s system
* The Component Diagram is in the main folder of the project
* The Drawn\_Petri\_Nets folder holds all petri nets included in this project, created with diagram tools

# Testing

The testing sequence includes 2 tests: 1. Car travelling through intersections; 2. Car jam management

## Test 1

For the first test, a DataCar object was sent using the input GUI, entering the system at Intersection\_1: place P\_a1. The car then travels through the intersection, needing to exit through P\_o4exit into the Connection\_Street. The car will ignore the exits to its right and will reach P\_5N of the Connection\_Street, from which it will enter Intersection\_2 through place P\_a1. Then, on the similar trajectory, it will enter Intersection\_2 and exit through P\_o4exit.

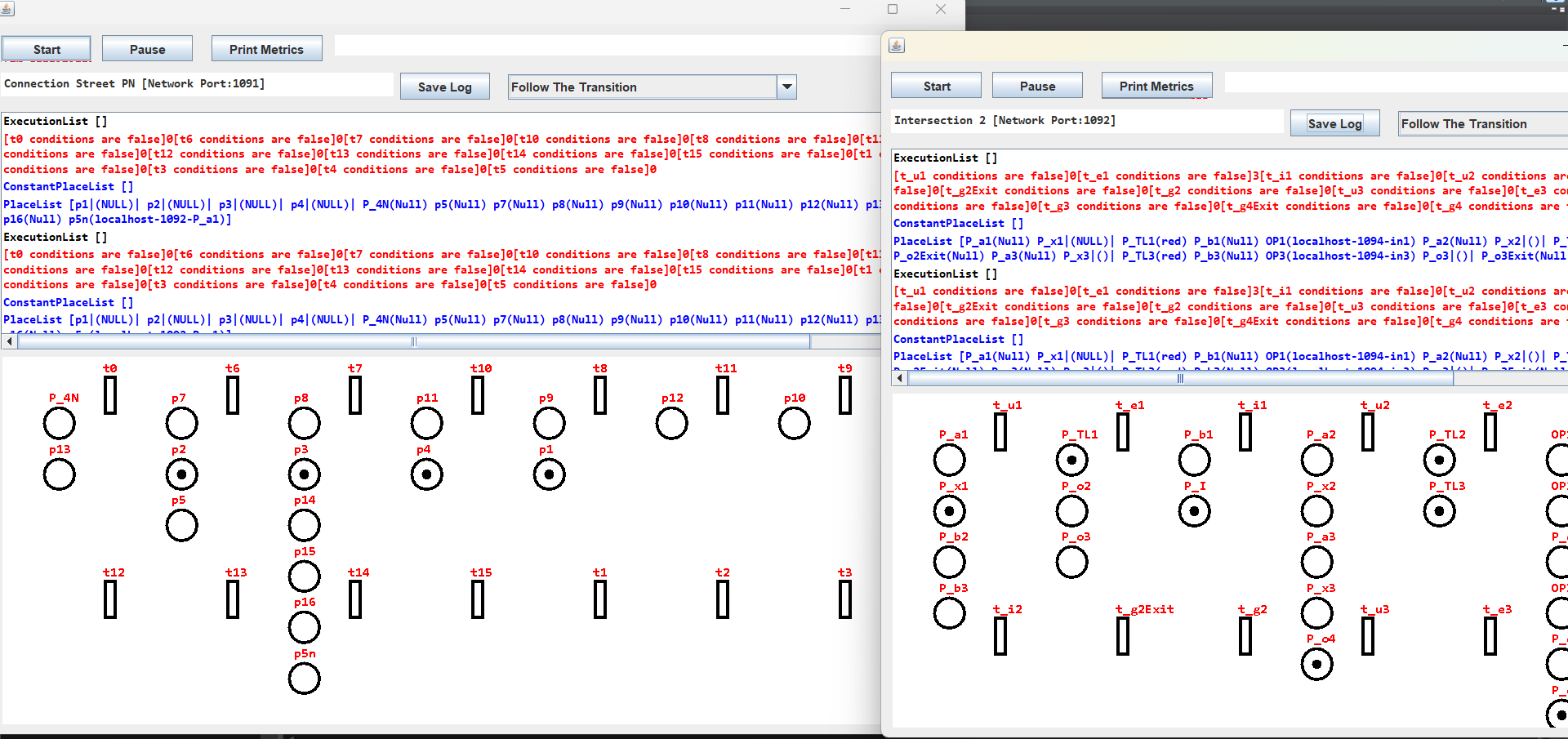
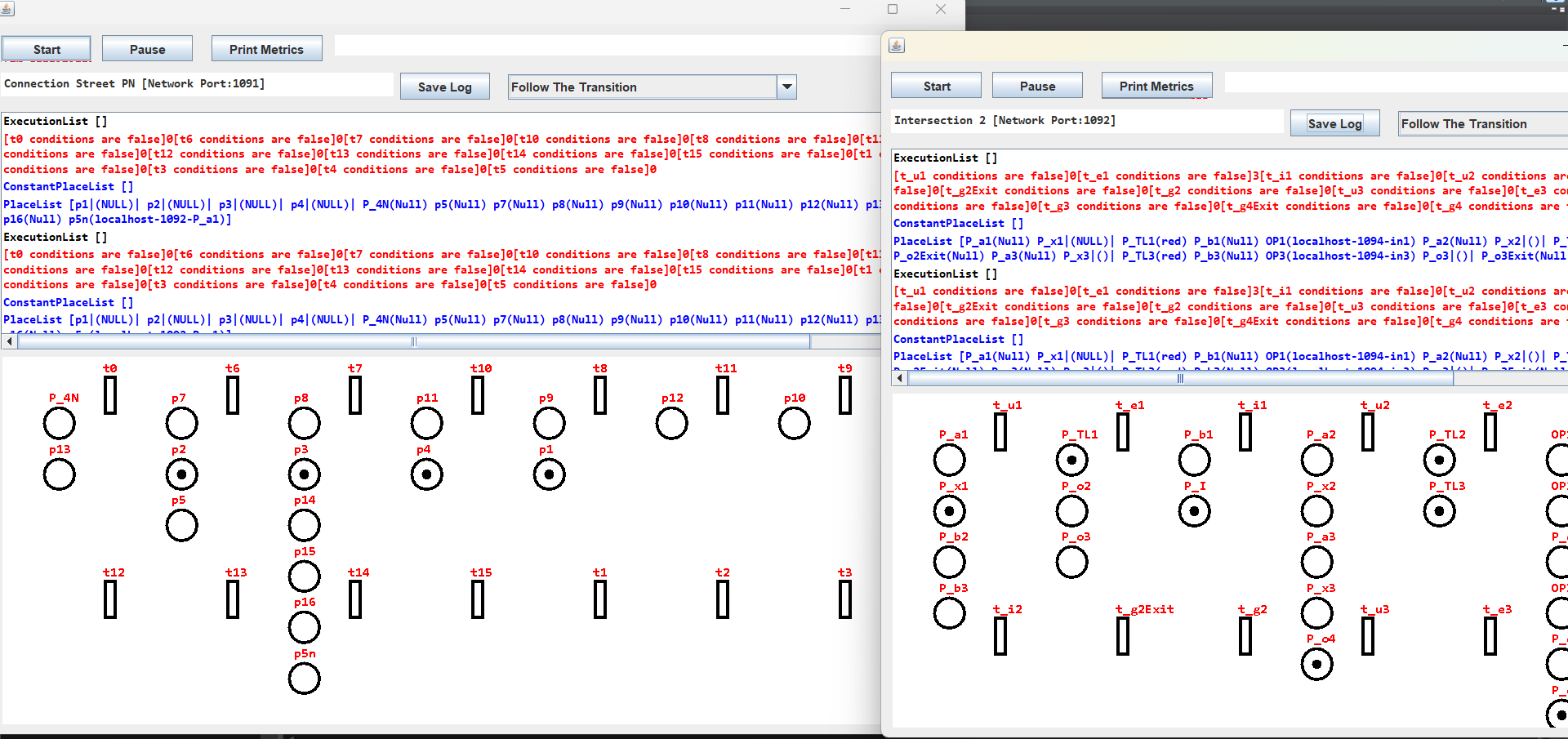
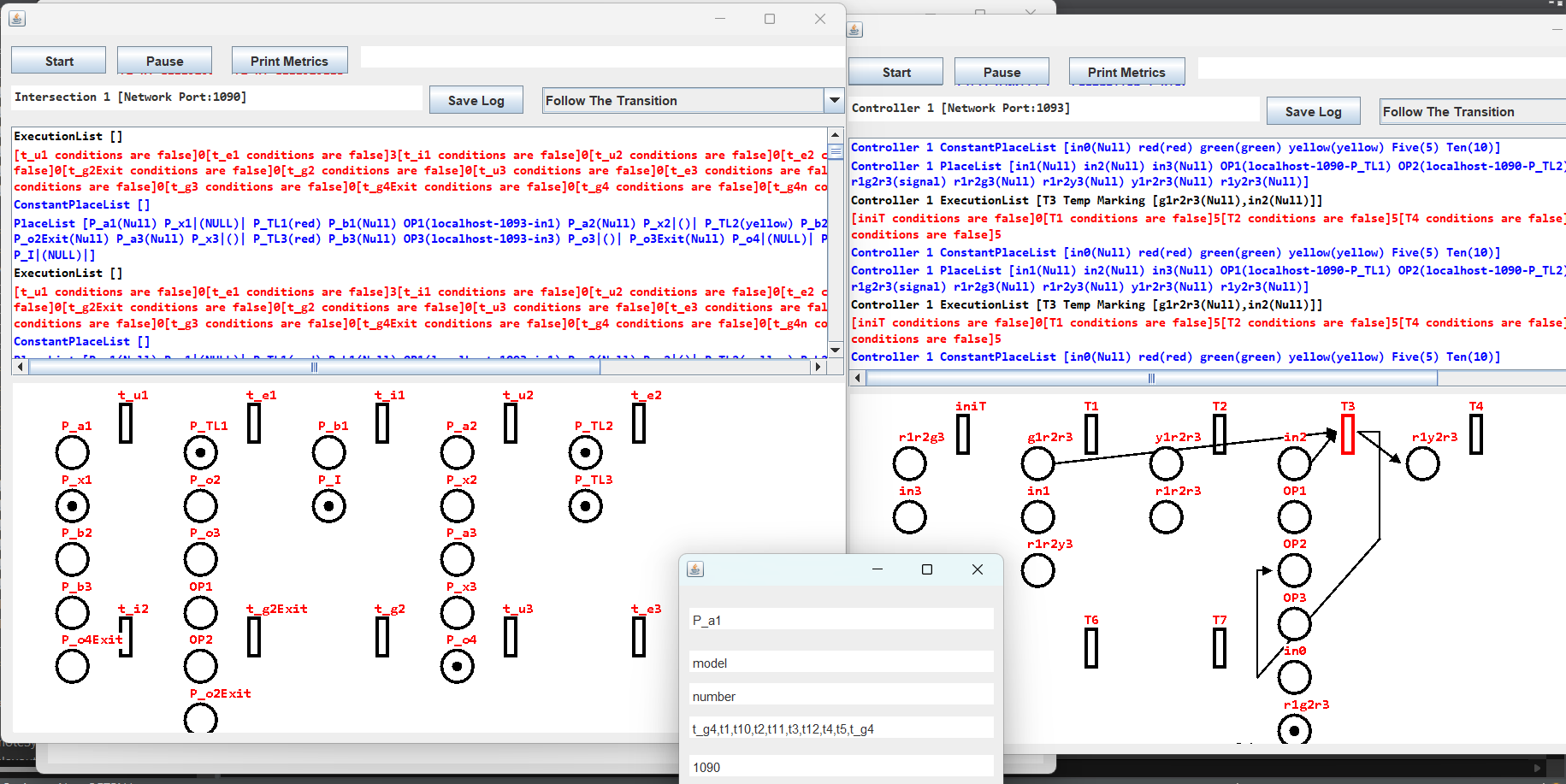


Figure 4.1.3 Test 1 – Intersection 2

Figure 4.1.2 Test 1 –Connection Street

Figure 4.1.1 Test 1 – Intersection 1

## 4.2 Test 2

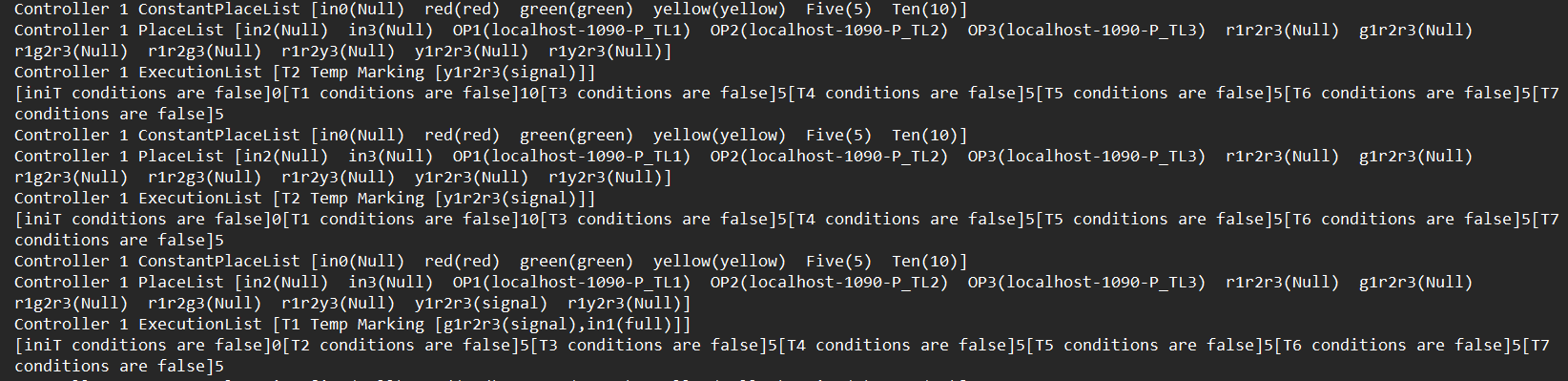
A traffic jam is created in both intersections by sending a number of cars higher than the capacity of the DataCarQueue, sending the ‘full’ signal to the corresponding input of the controller. In turn, the controller will continue to the transition that is responsible for sending a yellow light to that lane where you input the cars to, should have changed the 10 (Ten) seconds delay. After a few loops, it will return the delay to 5 (Five) seconds.

Figure 4.2.1 Terminal Log for Controller 1

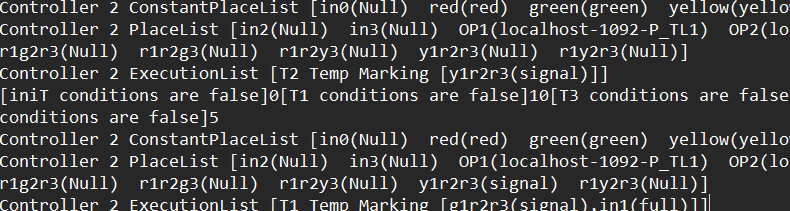
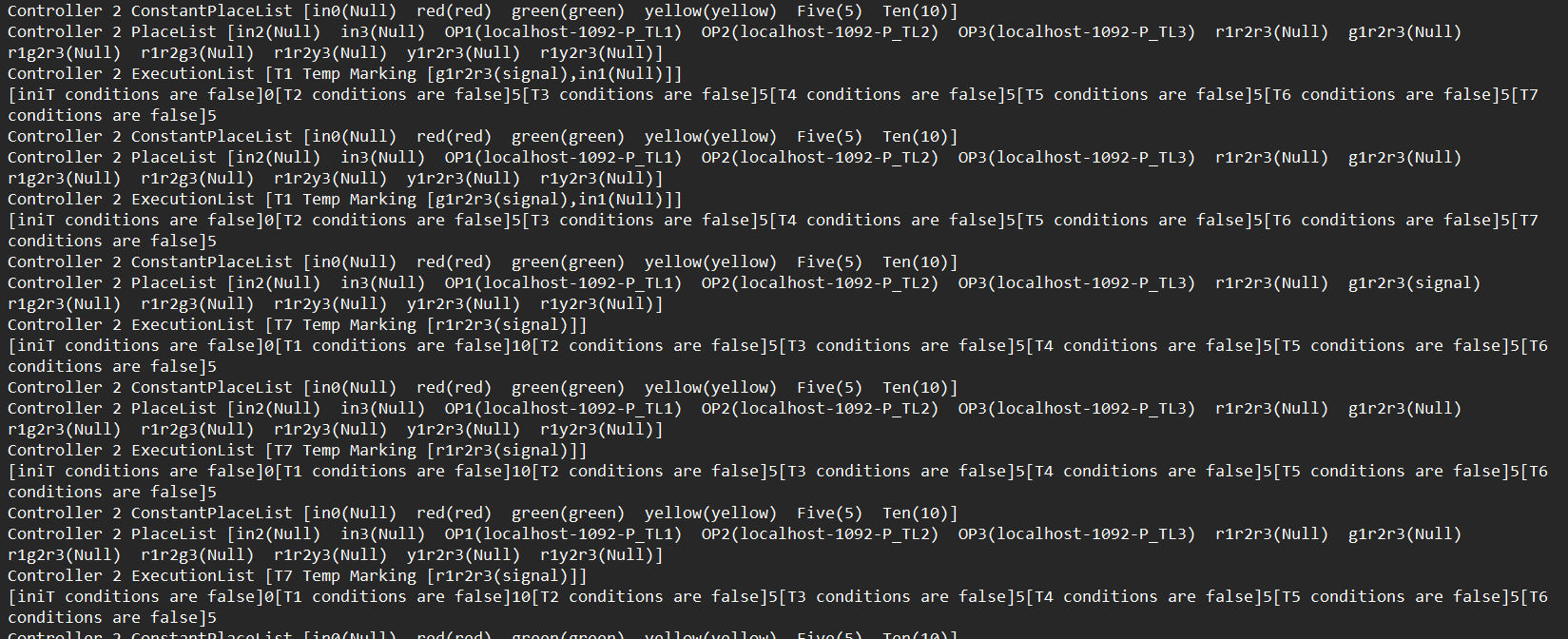


Figure 4.2.2. Terminal Log for Controller 2