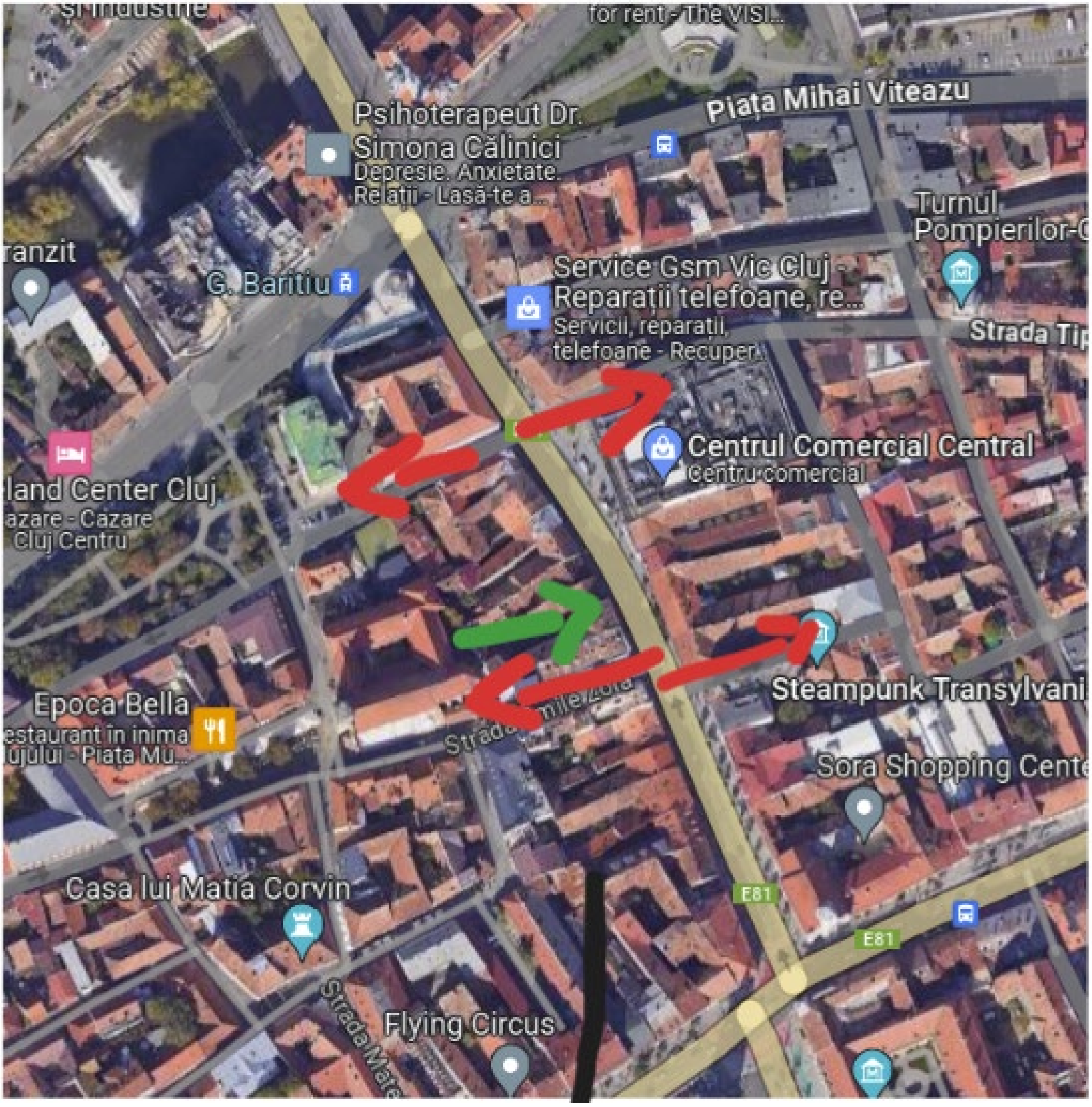
Report

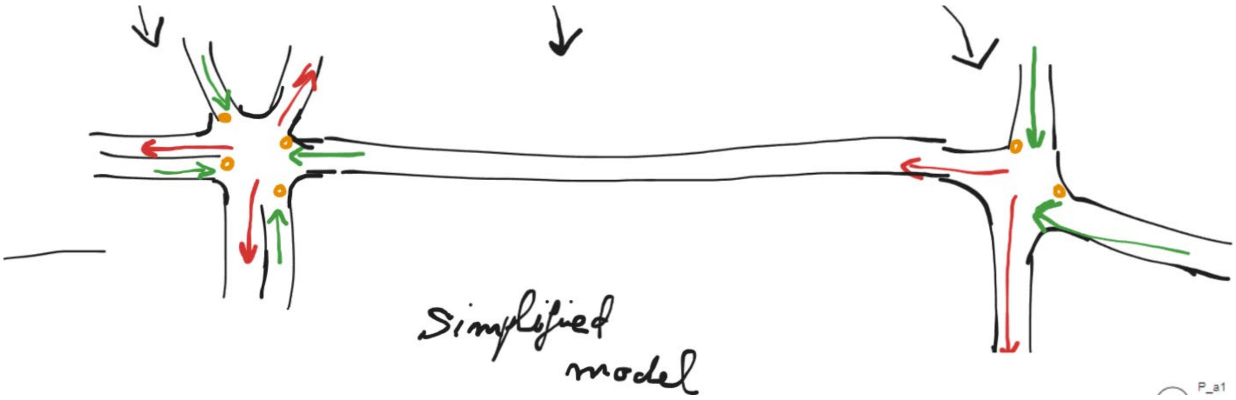
According to the map given to each team, develop a controller for each intersection (plant), that controller is a closed-loop one (with the *in*$_{(1..n)}$ input channels that is connected to its intersection's output channels *op*$_{(1..n)}$ and an Intersections (with the *OPs* output channels). The controller must have dynamic delays feature to extend the time of the green light in case of a traffic jam. (Project session 3, and 5)

Specifications:

1. Screen shot of the entire given map,

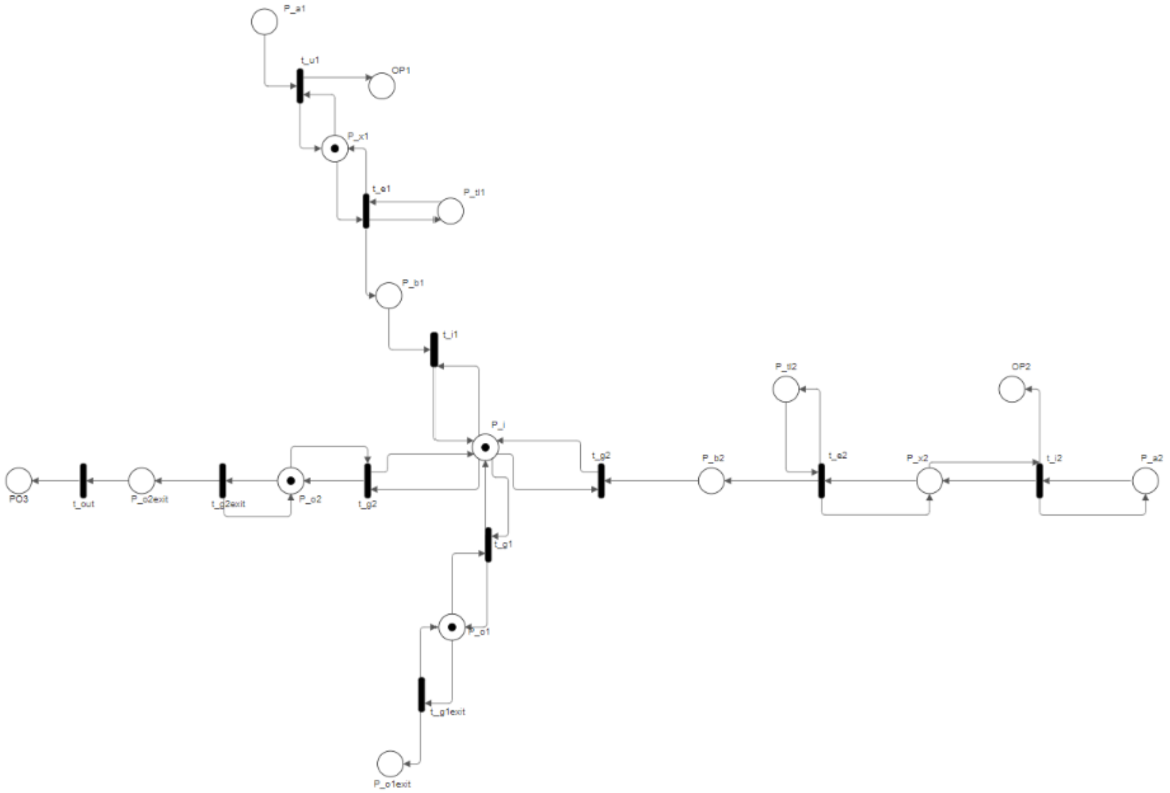


2. Simplified one showing the intersections and the middle street that connects them, if the street has output and input lanes, they should be drawn and implemented at the end.

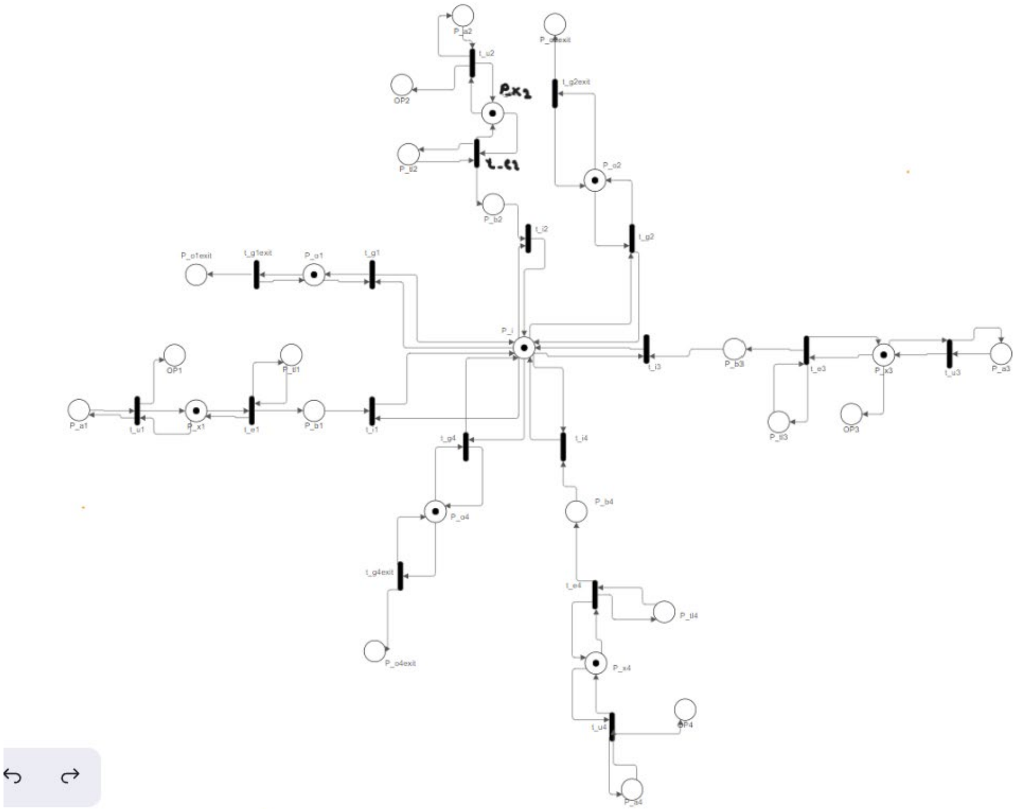


Design:

1. The OETPN model for the Plant (the intersections and the middle street).
    a. Intersection 1



    b. Intersection 2



2. Place types, grd&map for the entire OETPN.

Place Types – Intersection 1

Input lanes:

P_a1, P_b1, P_a2, P_b2	DataCar
P_x1, P_x2	DataCarQueue
P_tl1, P_tl2	DataString
OP1, OP2	DataTransfer

Output lanes:

P_o1, P_o2	DataCarQueue
P_o1exit, P_o2exit	DataCar
PO2	DataTransfer

P_i DataCarQueue

Grd&map – Intersection 1

t_u1: (P_a1 != NULL && P_x1 can add cars)

```
        P_x1.add(P_a1)
      (P_a1 != NULL && P_x1 can not add cars)
        OP1.send("FULL")
        P_a1 = P_a1
*same for t_u2
```

```
t_e1: (P_x1 haveCar && P_tl1 == "green")
        P_x1.popElementWithoutTarget(P_b1)
        P_tl1 = P_tl1
*same for t_e2
```

```
t_i1: (P_b1 != NULL && P_i canaddcars)
        P_i.addElement(P_b1)
*same for t_i2
```

```
t_g1: (P_i.HaveCarForMe && P_o1 CanAddCars)
        P_i.PopElementWithTargetToQueue(P_o1)
*same for t_g2
```

```
t_g1exit: (P_o1.HaveCar)
        P_o1.PopElementWithoutTarget(P_o1Exit)
*same for t_g2exit
```

Place Types – Intersection 2

Input lanes:

P_a1, P_b1, P_a2, P_b2, P_a3, P_b3, P_a4, P_b4	DataCar
P_x1, P_x2, P_x3, P_x4	DataCarQueue
P_tl1, P_tl2, P_tl3, P_tl4	DataString
OP1, OP2, OP3, OP4	DataTransfer

Output lanes:

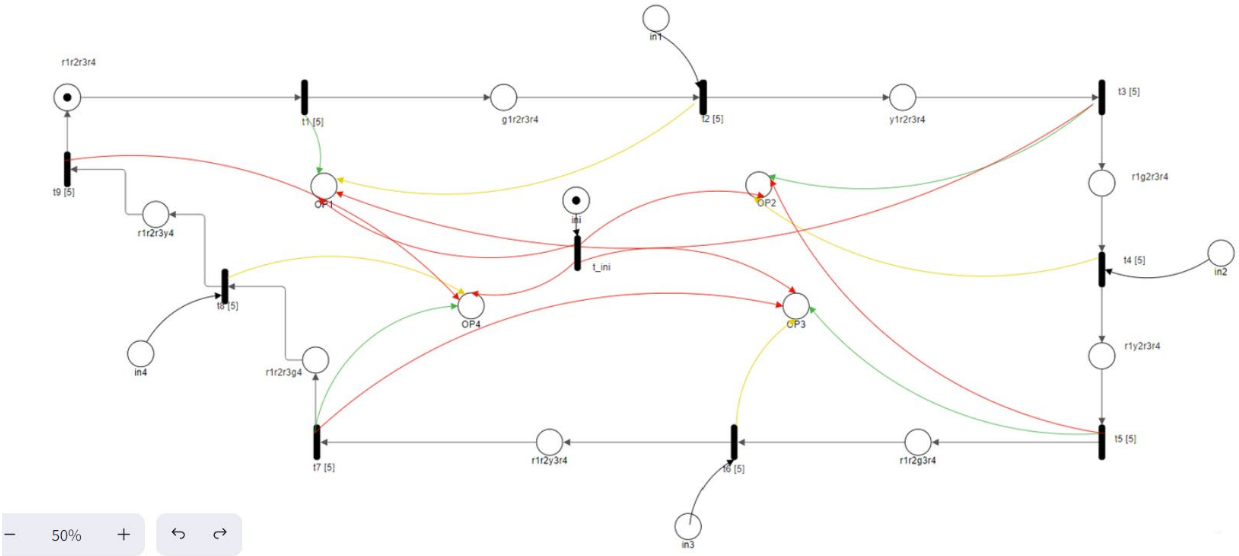
P_o1, P_o2, P_o4	DataCarQueue
P_o1exit, P_o2exit, P_o4exit	DataCar

Grd&map – Intersection 2

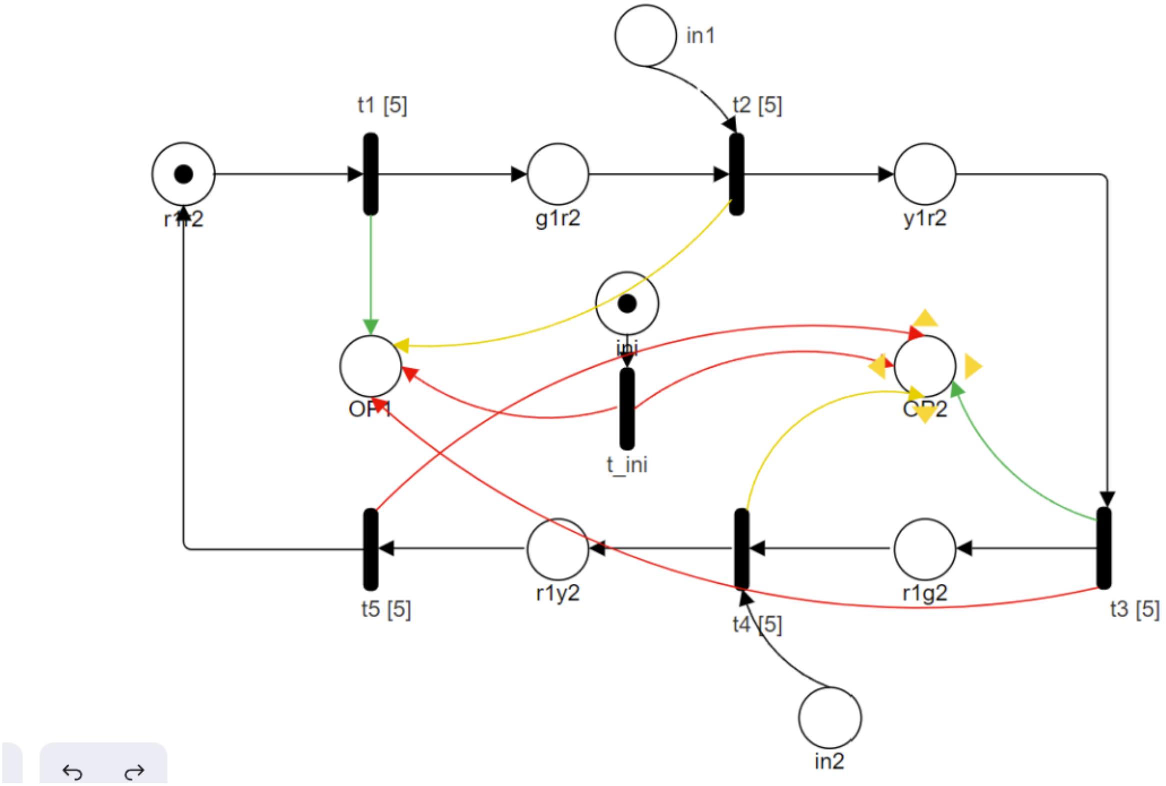
```
t_u1: (P_a1 != NULL && P_x1 can add cars)
        P_x1.add(P_a1)
      (P_a1 != NULL && P_x1 can not add cars)
```

```
        OP1.send("FULL")
        P_a1 = P_a1
*same for t_u2, t_u3, t_u4


t_e1: (P_x1 haveCar && P_tl1 == "green")
        P_x1.popElementWithoutTarget(P_b1)
        P_tl1 = P_tl1
*same for t_e2, t_e3, t_e4

t_i1: (P_b1 != NULL && P_i canaddcars)
        P_i.addElement(P_b1)
*same for t_i2, t_i3, t_i4


t_g1: (P_i.HaveCarForMe && P_o1 CanAddCars)
        P_i.PopElementWithTargetToQueue(P_o1)
*same for t_g2, t_g4

t_g1exit: (P_o1.HaveCar)
        P_o1.PopElementWithoutTarget(P_o1Exit)
*same for t_g2exit, t_g4exit
```

3. The OETPN model for the controllers.



Controller 1



Controller 2

4. The Place types, grd&map for the entire OETPN.

Controller 1

Place Types

FIVE, TEN– Data Integer

r1r2,g1r2,y1r2,r1g2,r1y2, in1, in2 - DataString

OP1, OP2 - DataTransfer

ini – DataString

Grd&map

t_ini: (ini != NULL)

OP1.sendOverNetwork(ini)

OP2.sendOverNetwork(ini)

ini.makeNull

t1: (r1r2 != NULL)

```
   OP1.SendOverNetwork("green")
   g1r2 = r1r2
*same for t3, t5
```

```
t2: (g1r2 != NULL && in1 == NULL)
    OP1.SendOverNetwork("yellow")
    y1r2 = g1r2
    DynamicDelay("FIVE")
   (g1r2 != NULL && in1 != NULL)
     OP1.SendOverNetwork("yellow)
     y1r2 = g1r2
     DynamicDelay("TEN")
*same for t4
```

Controller 2

Place Types

Five, Ten – Data Integer

r1r2r3r4, g1r2r3r4, y1r2r3r4, r1g2r3r4, r1y2r3r4, r1r2g3r4, r1r2y3r4, r1r2r3g4, r1r2r3y4 - DataString

OP1, OP2, OP3, OP4, in1, in2, in3, in4 - DataTransfer

ini - DataString

Grd&map

```
t_ini: (ini != NULL)
   OP1.sendOverNetwork(ini);
   OP2.sendOverNetwork(ini);
   OP3.sendOverNetwork(ini);
    OP4.sendOverNetwork(ini);
```

```
  ini.makeNull();
```

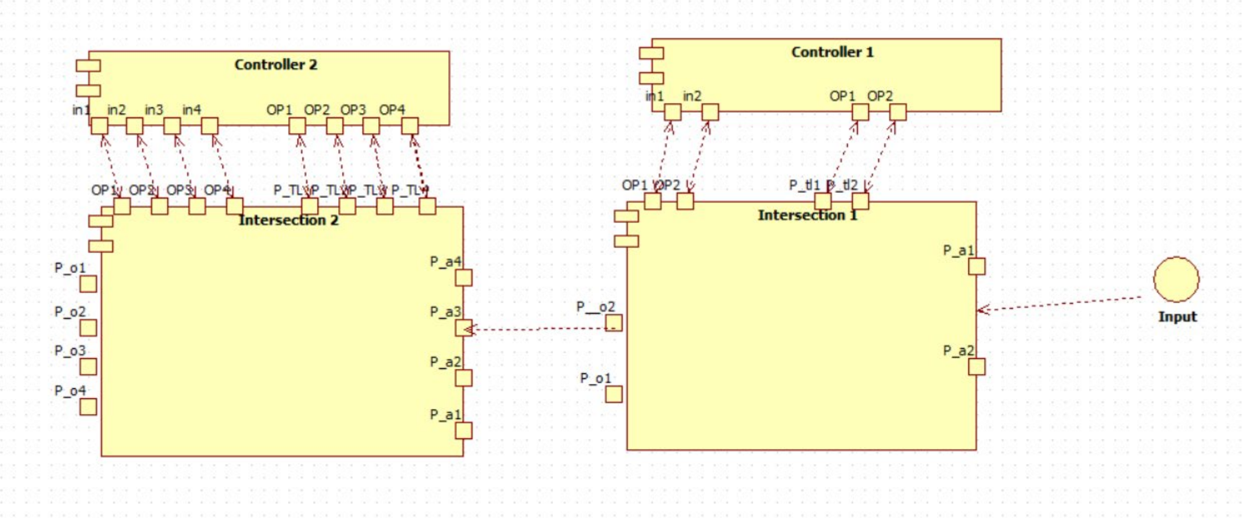
```
t1: (r1r2r3r4 != NULL)
  OP1.SendOverNetwork("green")
  // OP1. SendOverNetwork(“red”) ---> for t3, t5, t7, t9
  g1r2r3r4 = r1r2r3r4
```

*same for **t3, t5, t7, t9**

```
t2: (g1r2r3r4 != NULL && in1 == NULL)
   OP1.SendOverNetwork("yellow")
   y1r2r3r4 = g1r2r3r4
   DynamicDelay("FIVE")
  (g1r2r3 != NULL && ini != NULL)
   OP1.SendOverNetwork("yellow")
   y1r2r3r4 = g1r2r3r4
   DynamicDelay("TEN")
```

*same for **t4, t6, t8.**

5. The component diagram for the entire system (depending on your implementation, each OETPN is considered a component) and show the names of the input and output channels.



Implementation:

1. Link for the repository:

https://github.com/ciomin/dcs-project

Testing:

1. Send a car from the 1st intersection, that should go through the middle street and exit from one of the exit lanes from the 2nd intersection. Attach screen shots showing how the car moves and at the end of the test, pause the intersection OETPN and click on the save log button, save it as test1_intersection 1.txt and test1_intersection 2.txt if you have implemented them in two separate OETPNs. Then add the text file/s to the repository.
2. Traffic jam: for each intersection, create a traffic jam case by sending the maximum number of cars to the input lane of the intersection, start the controller, then send the last car. The controller should receive a signal from the plant (intersection) and the transition that is responsible for sending a yellow light to that lane where you input the cars to, should have changed the delay to 10 sec. Let the controller OETPN run until it reaches the same transition (2 loops) to show that the delay is changed back to 5 sec. pause the controller OETPN and click on the save log button, save it as test2.txt and add the text file to the repository.