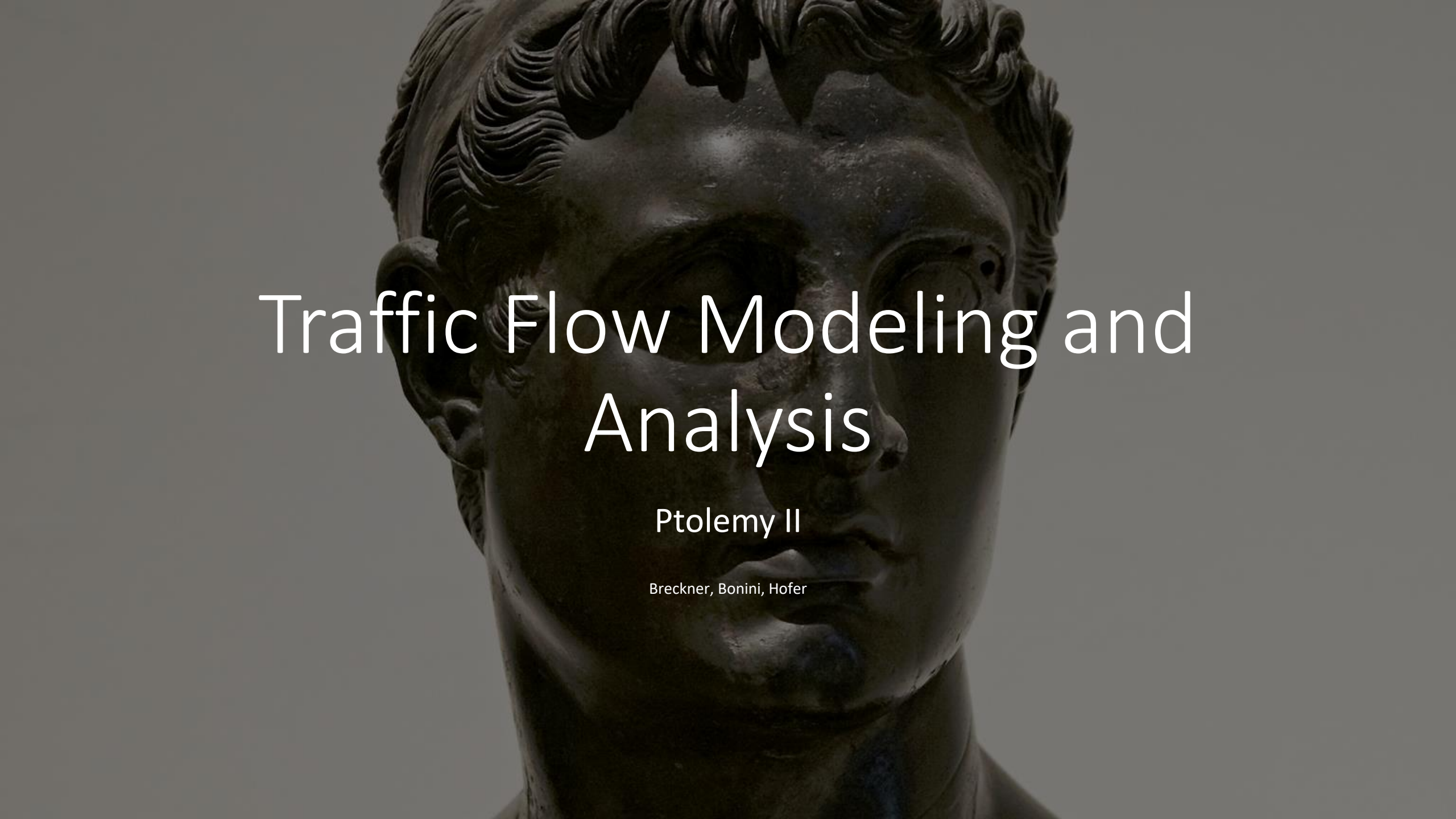


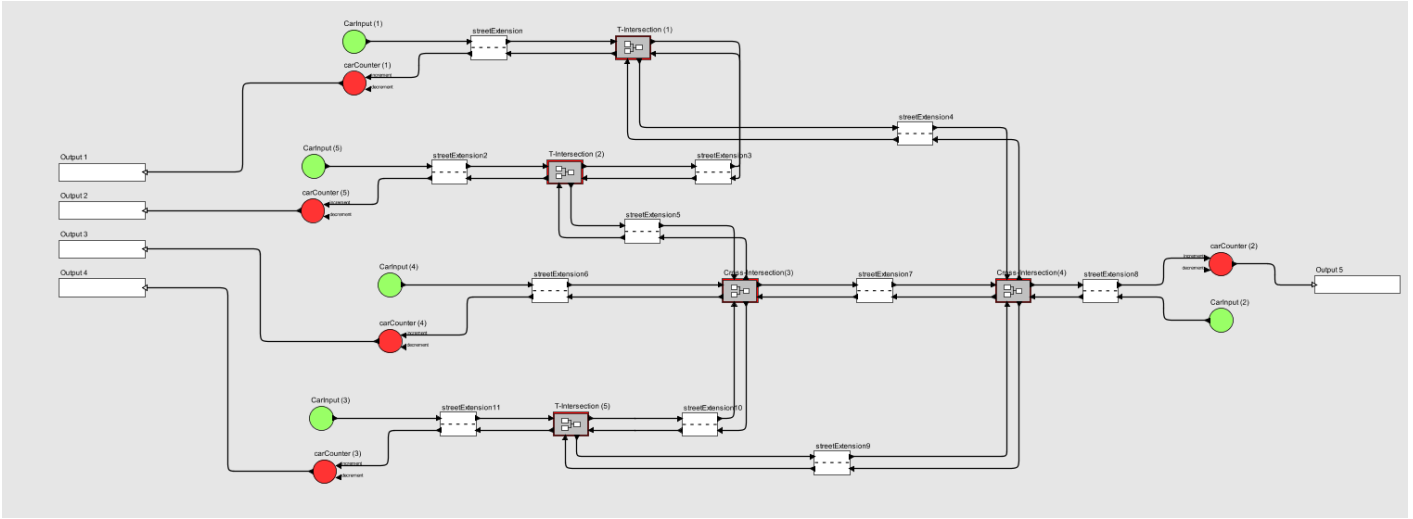
Traffic Flow Modeling and Analysis

Ptolemy II

Breckner, Bonini, Hofer

Ziel

- Ptolemy Traffic Model mit verschiedenen Kreuzungen (T- und Cross-Intersection)
- Rechtsregel implementieren
- Straßen mit Handhabung der Autos
- Autos aus mehreren Komponenten
- Analyse um starken Verkehr zu erkennen
- Design, das es ermöglicht, einfach Verkehrsmodelle auszutauschen



Ptolemy II

- Ptolemy II ist ein open-source Software Framework, dass das experimentieren mit actor-oriented design supportet

- Director -> definiert die Semantik von einem Model
- Actor -> gleichzeitig ausgeführt und teilen Daten miteinander
- Composite Actor -> Kombination aus Actors

Start-Up Phase

Ptolemy Demos dursuchen → „Air Traffic Model“ scheint Ähnlichkeiten aufzuweisen

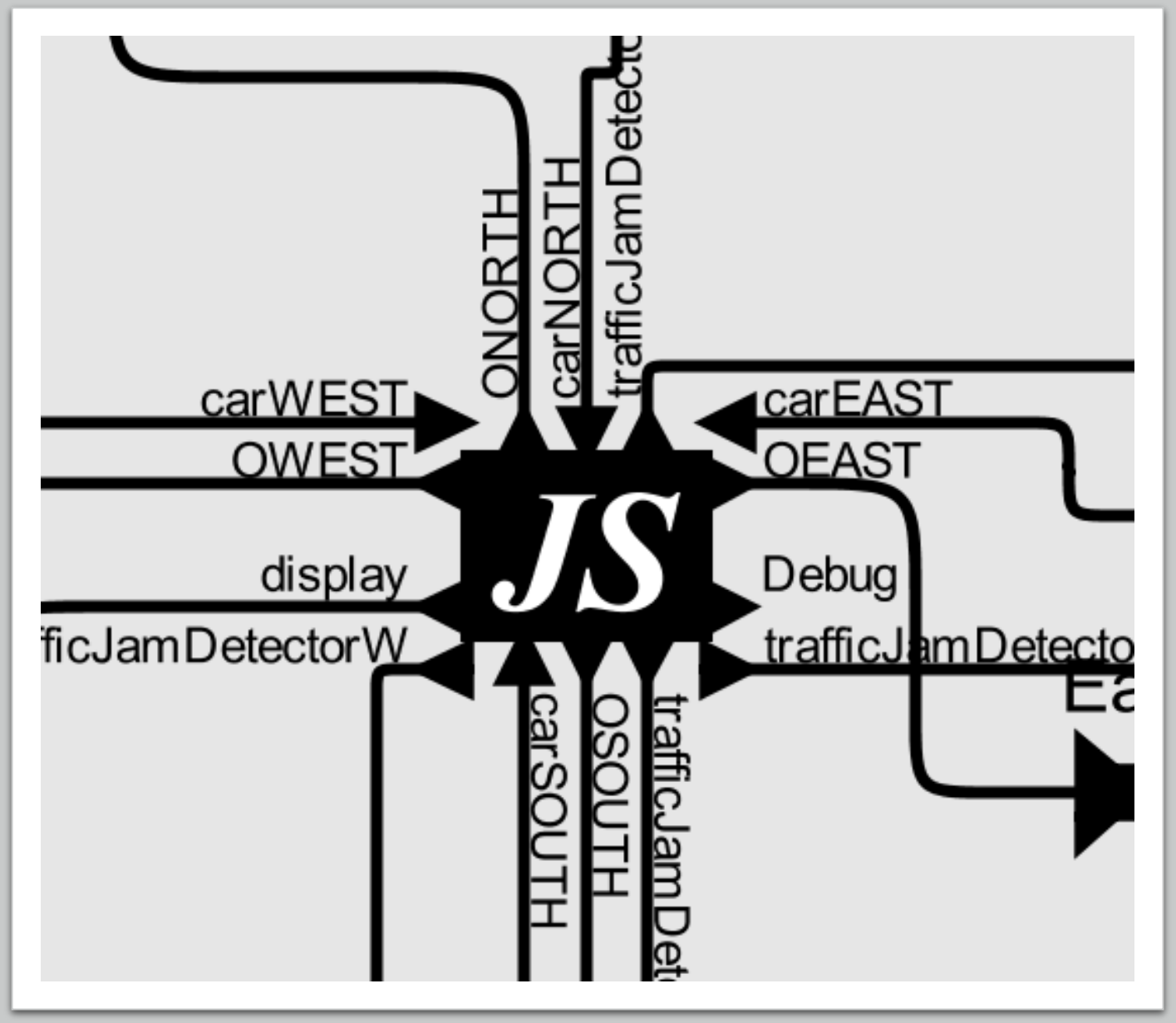
Eigenen Director geschrieben

Eigene Models und Components geschrieben

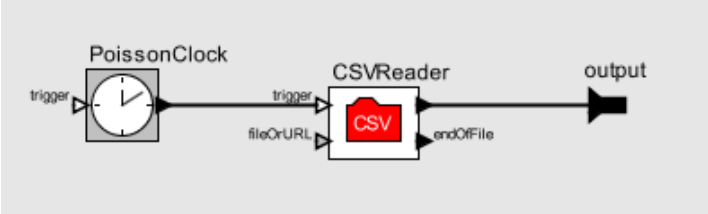
Zu viele Fehler bei Einbindung….

Entdeckung des JavaScript-Actors

- Wir begannen mit JavaScript eigenen Code in unser Modell einzuschleusen
- Damit konnten wir simple if-Bedingungen erzeugen und die Input-Ports besser kontrollieren



Wie bekommen wir Daten zu unseren Autos?



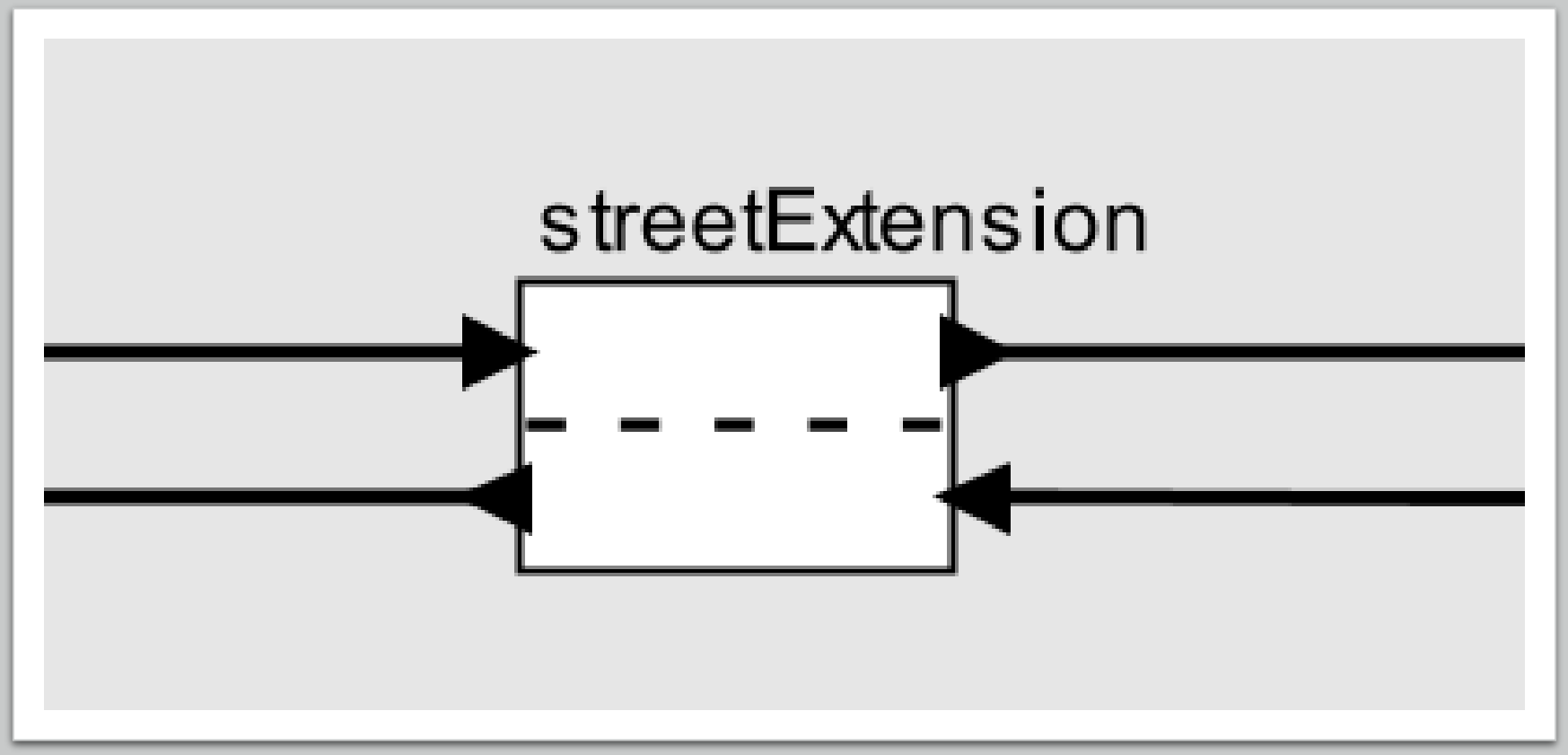
- Actor „CSVReader“
  - Kann CSV Dateien handhaben
- Actor wurde verwendet um die von uns erstellen „car.txt“ Dateien einzulesen und richtig zu konvertieren

- Actor „PoissonClock“
  - Sendet zu einer zufälligen Zeiten ein Signal
  - Sendet ein Signal zu „CSVReader“ um ein Auto zu unserer „Street“ zu senden

cars.txt

- Ein Auto besteht, wie erwähnt aus verschiedenen Komponenten
- Getrennt durch ein Tab

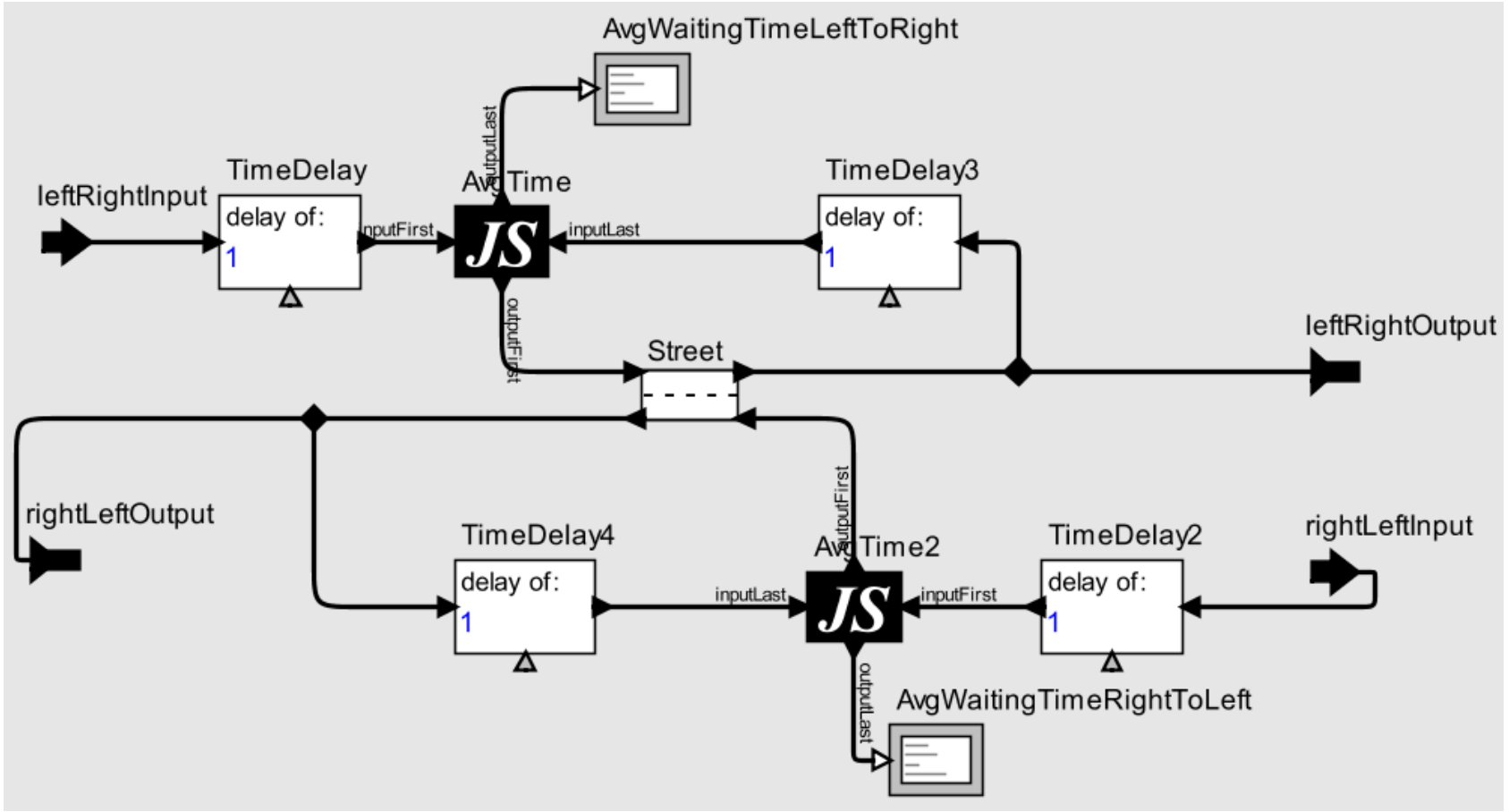
carId	roadMap	indicator	intersectionCount
0	{2, 4}	0	0
1	{4, 5, 10}	0	0
2	{2, 6, 12}	0	0
3	{2, 4, 5, 8}	0	0



Street Extension

Bestehend aus zwei Schichten

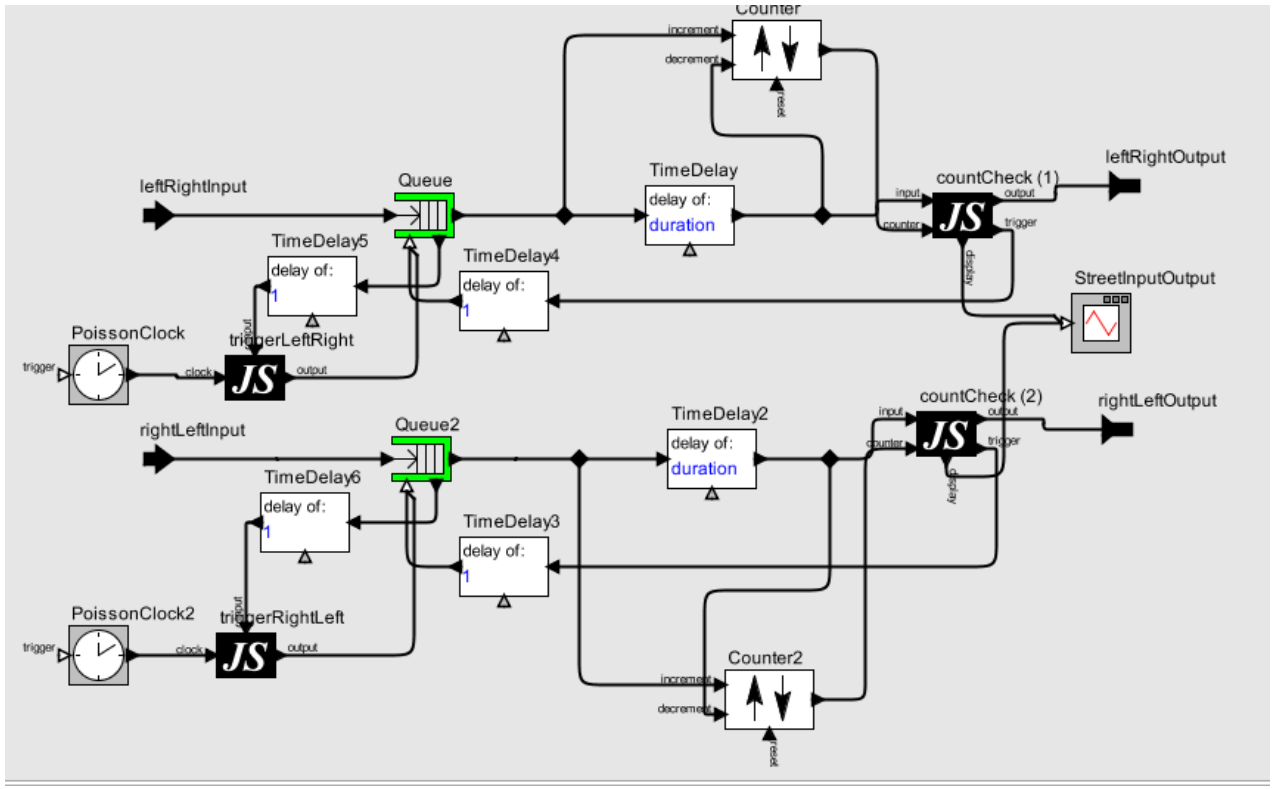
Street-Analysis (later more…)

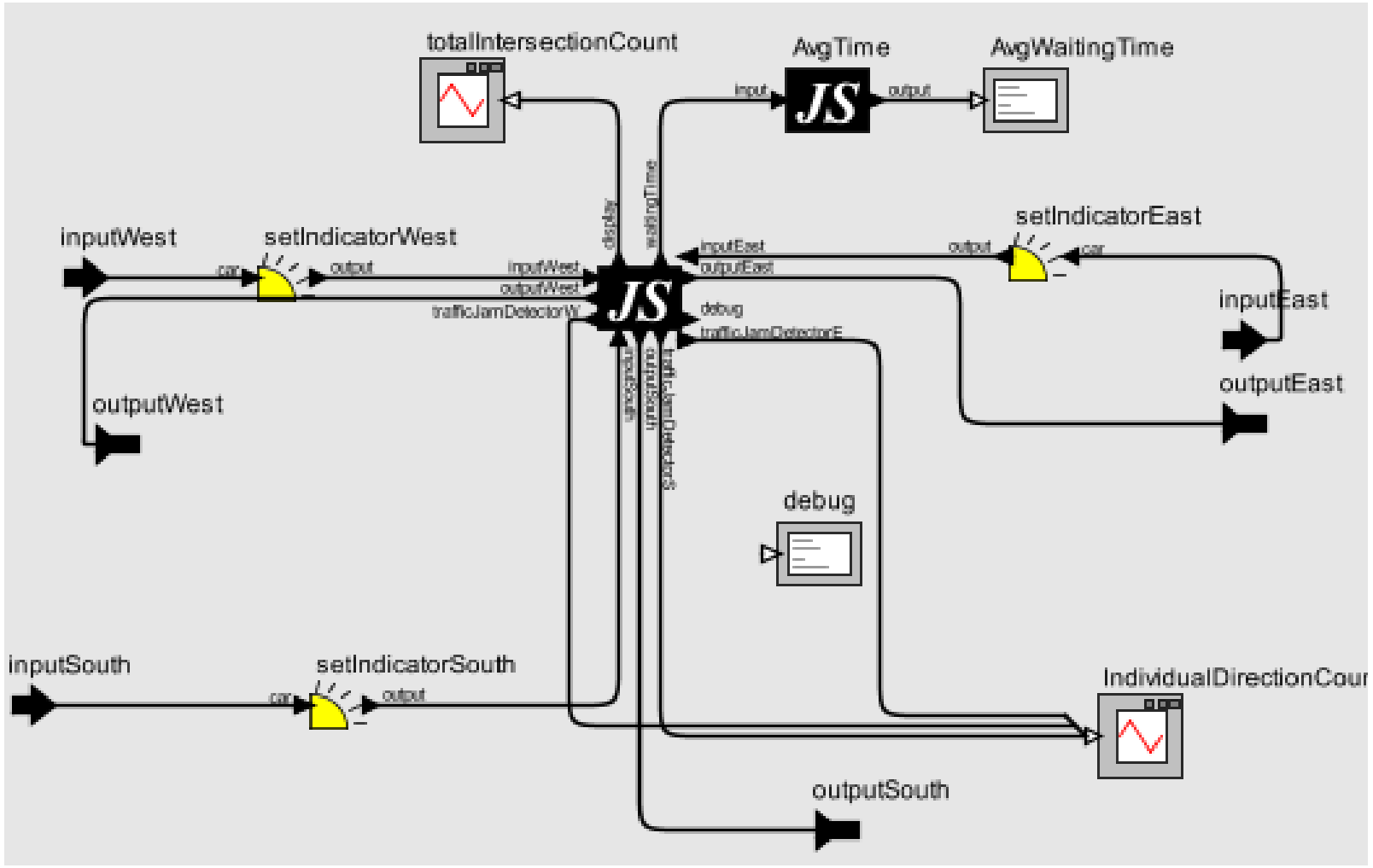


- Außerhalb des wirklichen Street-Actors, befindet sich zunächst die Analyse der Dauer und Zeit

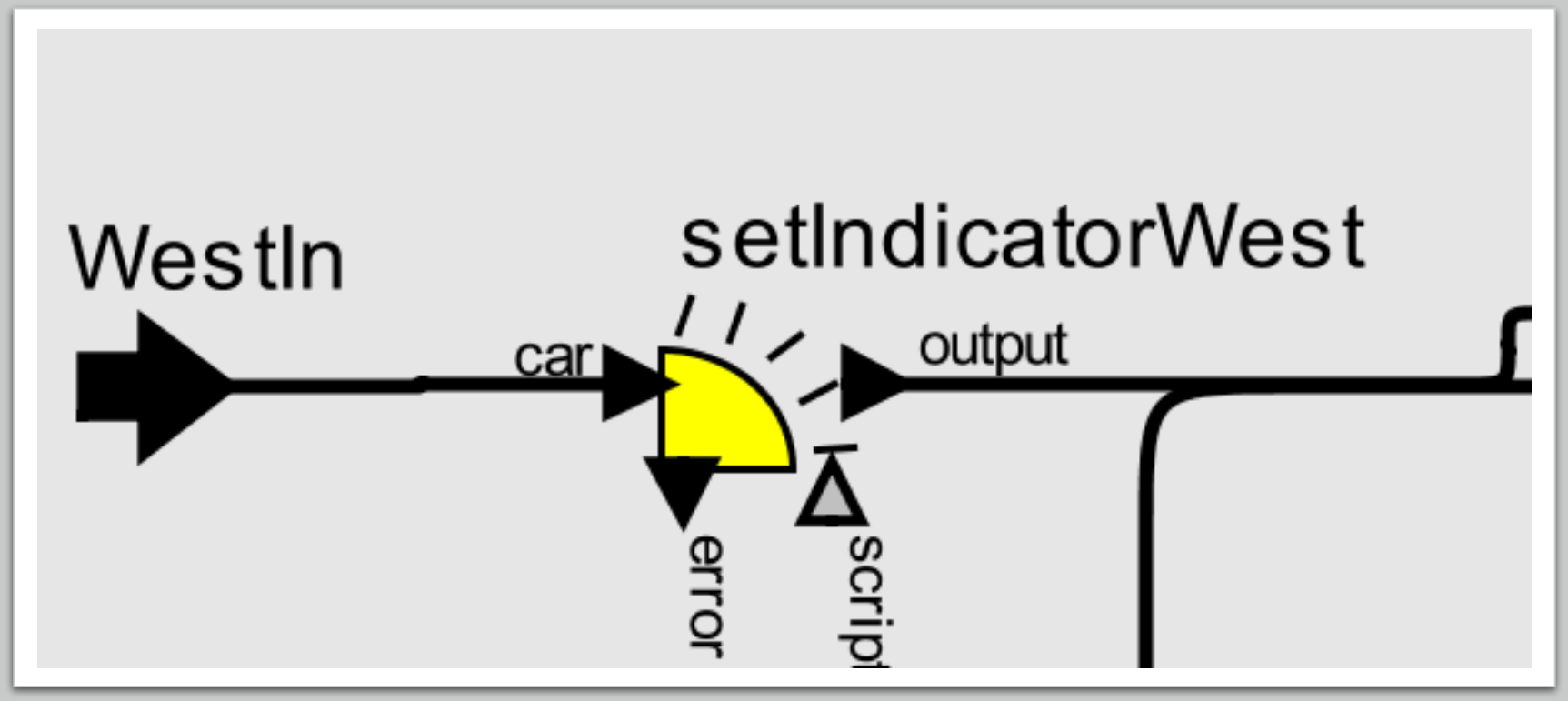
Street-Actor

- Straße beinhaltet:
  - Eine Queue
  - Feste Straßenlänge
  - Dauer, die Straße zu überqueren
  - Counter zur Analyse
- Alle Parameter von Außen setzbar (= configurable)
- Delays müssen zwischengeschaltet werden





T-Intersection besteht aus…



Indicator

- Der ‚Indicator' ist nichts weiter, als der Blinker eines jeden Autos
- Er wird für jedes Auto vor der Intersection gesetzt, um zu zeigen, in welche Richtung es möchte

Indicator

- Bei jedem Auto wird in seiner „RoadMap“ gelesen, wohin es fahren möchte, und dementsprechend wird der Blinker auf eine Richtung gesetzt
- Richtungen:
  - Links (-1)
  - Rechts (1)
  - Gerade aus (0)
- So weiß die Kreuzung wohin Autos wollen
- Durch den „intersectionCounter“ weiß man, wo man in der „RoadMap“ lesen muss (bzw. wie viele Intersections schon passiert wurden

```
exports.setup = function () {
    this.input('car');

    this.output('output');
};

exports.fire = function () {
    carWest = this.get('car');

    if(carWest.roadMap[carWest.intersectionCount] == this.getParameter('SOUTH')) {
        carWest.intersectionCount = carWest.intersectionCount + 1;
        carWest.indicator = 1;
        this.send('output', carWest);
    }
    else if(carWest.roadMap[carWest.intersectionCount] == this.getParameter('EAST')) {
        carWest.intersectionCount = carWest.intersectionCount + 1;
        carWest.indicator = 0;
        this.send('output', carWest);
    }
    else if(carWest.roadMap[carWest.intersectionCount] == this.getParameter('NORTH')) {
        carWest.intersectionCount = carWest.intersectionCount + 1;
        carWest.indicator = -1;
        this.send('output', carWest);
    }
};
```

T-intersection-itself

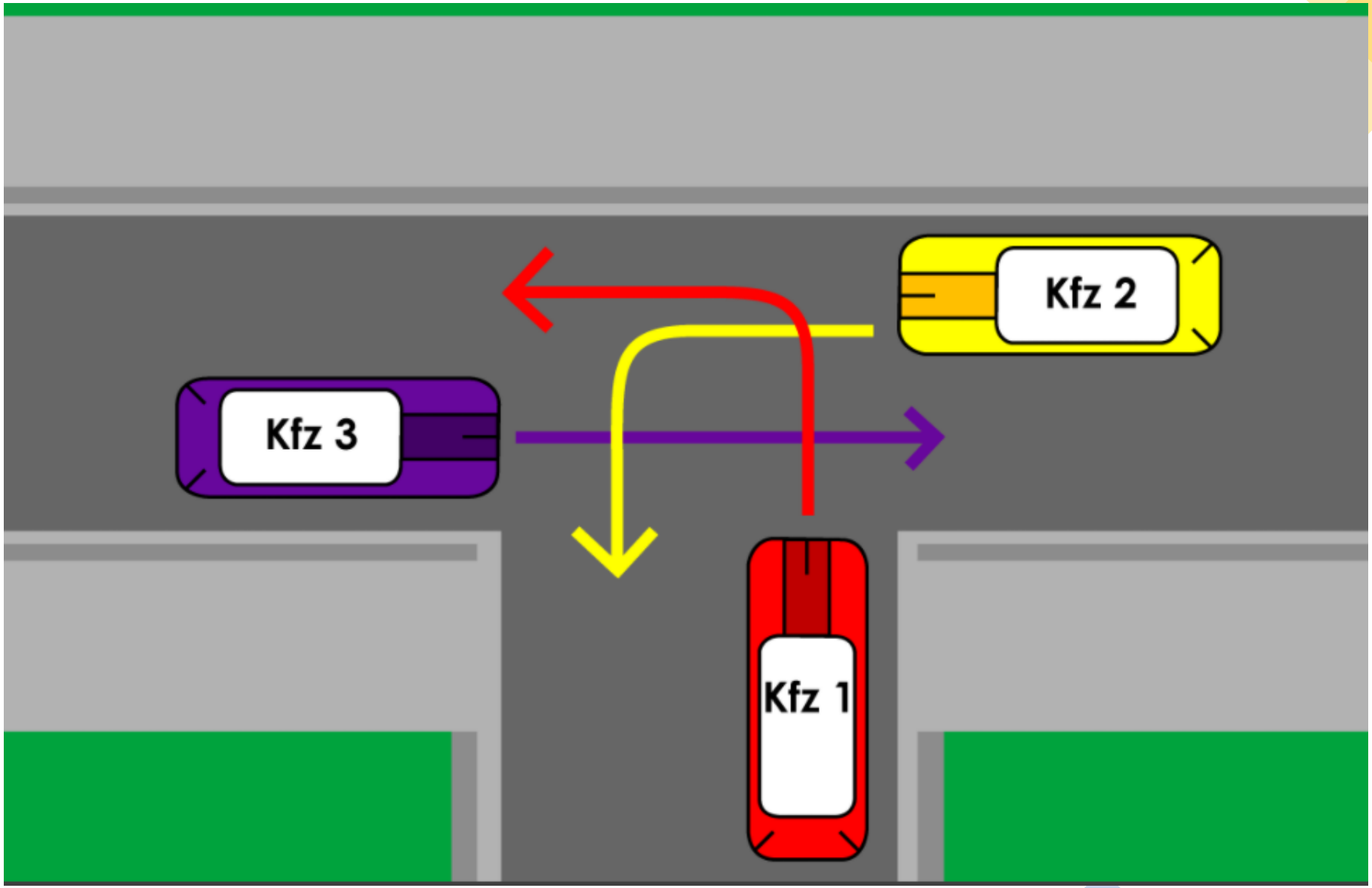
- In der Kreuzung selbst befinden sich Queues an allen Eingängen, in die die Autos rein kommen, sobald sie ankommen
- Viele verschiedene Inputs/Outputs
- Für die Analyse wird ein Counter mitgetragen
- „right_hand_rule()“ → Methode, die die Rechts-Regel implementiert

```
exports.setup = function () {
    //the input-ports for the incoming cars
    this.input('inputWest');
    this.input('inputEast');
    this.input('inputSouth');
    //the output-ports for the leaving cars
    this.output('outputWest');
    this.output('outputEast');
    this.output('outputSouth');
    //the output-ports for analysis
    this.output('display');
    this.output('trafficJamDetectorW');
    this.output('trafficJamDetectorE');
    this.output('trafficJamDetectorS');
    this.output('waitingTime');

    this.output('debug');
}

exports.initialize = function () {
    //InputHandlers add cars to the queues of the corresponding direction

    this.addInputHandler('inputWest', function() {
        var car = this.get('inputWest');
        car.time = Math.round(new Date().getTime() / 1000); //gets a unix timestamp
        queueW.unshift(car);
        carCount++;
        countW++;
        // send counters for analysis
        this.send('display', carCount);
        this.send('trafficJamDetectorW', countW);
        // call right hand rule only if the car is the first in the intersection
        if(carCount == 1) {
            setTimeout(right_hand_rule, 2000);
        }
    });
```


Kfz 2
Kfz 3
Kfz 1

```
if(QueueW.length != 0) { // car from WEST
    this.send('Debug',"CarW " + QueueW[QueueW.length - 1]);
    if(QueueW[QueueW.length - 1].indicator == 0) { //WEST -> STR
        if(QueueS.length != 0) { // car from SOUTH
            //stop
        } else { // no car from SOUTH
            this.send('OEAST',QueueW[QueueW.length - 1]);
            SentW = true;
        }
    }else if(QueueW[QueueW.length - 1].indicator == 1) { // WEST
        this.send('OSOUTH',QueueW[QueueW.length - 1]);
        SentW = true;
    }else if(QueueW[QueueW.length - 1].indicator == -1) { // WES
        if(QueueE.length != 0 && QueueE[QueueE.length - 1].indic
            this.send('ONORTH',QueueW[QueueW.length - 1]);
            SentW = true;
        } else if(QueueS.length != 0) { // car from SOUTH
            // stop
        } else {
            this.send('ONORTH',QueueW[QueueW.length - 1]);
            SentW = true;
        }
    }
}
```

Right Hand Rule

- Für jede Richtung gibt es einen solchen Block
- Hier wird der Indicator betrachtet und ob von anderen Richtungen Autos kommen, die eventuell die Vorfahrt beeinflussen
- Wenn man fahren darf, wird das entsprechende Auto an den Output gesendet und von der Queue ge-popt
- Gleiches Prinzip für eine Cross-intersection nur mit komplexerer Rechtsregel, da eine Richtung dazu kommt

Cross-Intersection

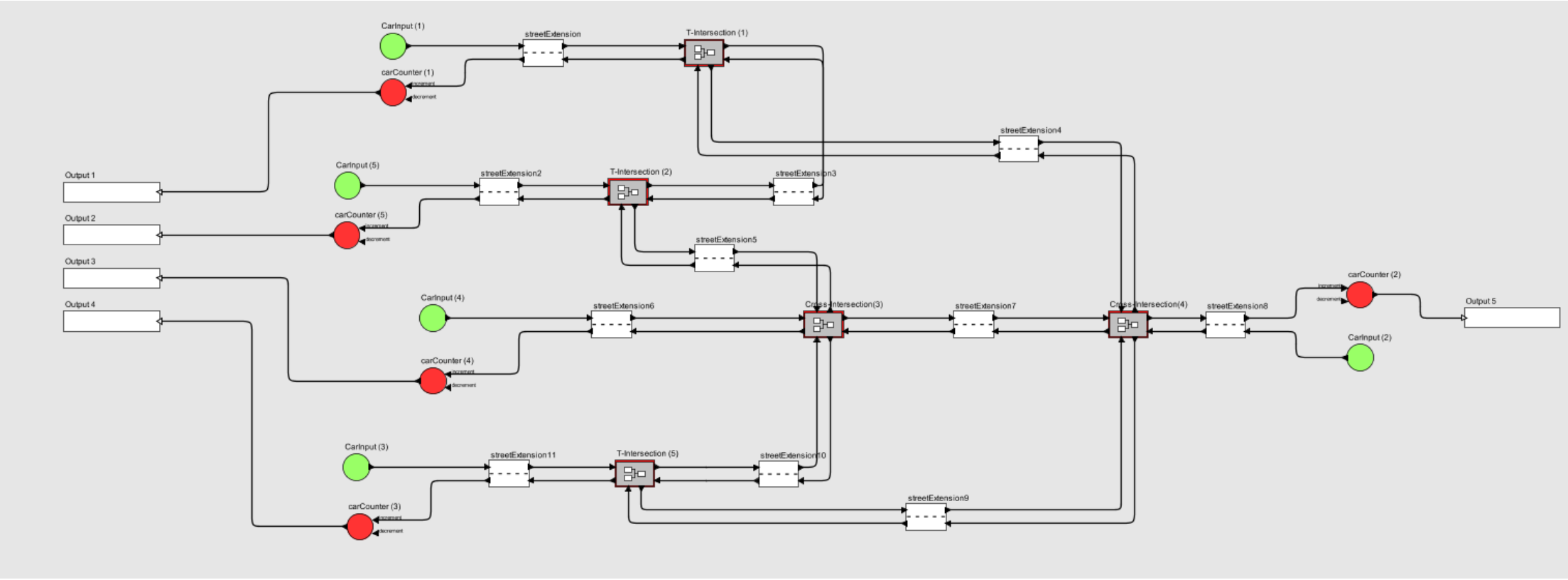
- Komplexere Rechtsregel Implementation
- Spezialfall: Kein Auto hat Vorrang, Lösung durch randomisiertes Vorfahrt bestimmen

```
//Special case if there is a car on each side of the intersection and no car could drive
//decides randomly which car can drive
if(!isSentW && !isSentE && !isSentS && !isSentN) {
    //0 = West, 1 = East, 2 = South, 3 = North
    var random = Math.floor(Math.random() * 3);

    //sends information to debug-ports
    this.send('debug',"queueW RAND " + queueW.length);
    this.send('debug',"queueE RAND " + queueE.length);
    this.send('debug',"queueS RAND " + queueS.length);
    this.send('debug',"queueN RAND " + queueN.length);
    this.send('debug', "carCount RAND " + carCount);
    this.send('debug', "random numb: " + random);

    if(random == 0) {
        if(queueW[queueW.length - 1].indicator == 0) {
            this.send('outputEast',queueW[queueW.length - 1]);
        }
        else if(queueW[queueW.length - 1].indicator == 1) {
            this.send('outputSouth',queueW[queueW.length - 1]);
        }
        else if(queueW[queueW.length - 1].indicator == -1) {
            this.send('outputNorth',queueW[queueW.length - 1]);
        }
        isSentW = true;
    }
```

Streetnetwork



Analyse

- Wie viele Autos sich in einer Intersection oder Street befinden um Staubildung zu erkennen
- Die durchschnittliche Wartezeit in Streets
- Die durchschnittliche Wartezeit in Intersections
- Tests werden vorgenommen mit insgesamt 125 Autos und insgesamt 200 Autos

Intersection Wartezeit in Sekunden (125 Autos)

Intersection	Durchschnittliche Wartezeit
T-Intersection(1)	2,63
T-Intersection(2)	2,34
Cross-Intersection(3)	18,39
Cross-Intersection(4)	3,75
T-Intersection(5)	1,93

Street Wartezeit in Sekunden (125 Autos)

STREET (VERBUNDENE INTERSECTIONS)	DURCHSCHNITTLICHE WARTEZEIT (L -> R)	DURCHSCHNITTLICHE WARTEZEIT (R -> L)
1 (IO-1)		
2 (IO-2)	1,04	1,37
3 (2-1)	2,76	1
4 (1-4)	1	1
5 (2-3)	1	1
6 (IO-3)	2,12	1
7 (3-4)	1	0,96
8 (4-IO)	1	0,96
9 (5-4)	1	1
10 (5-3)	0,95	1
11 (IO-5)	1,08	1

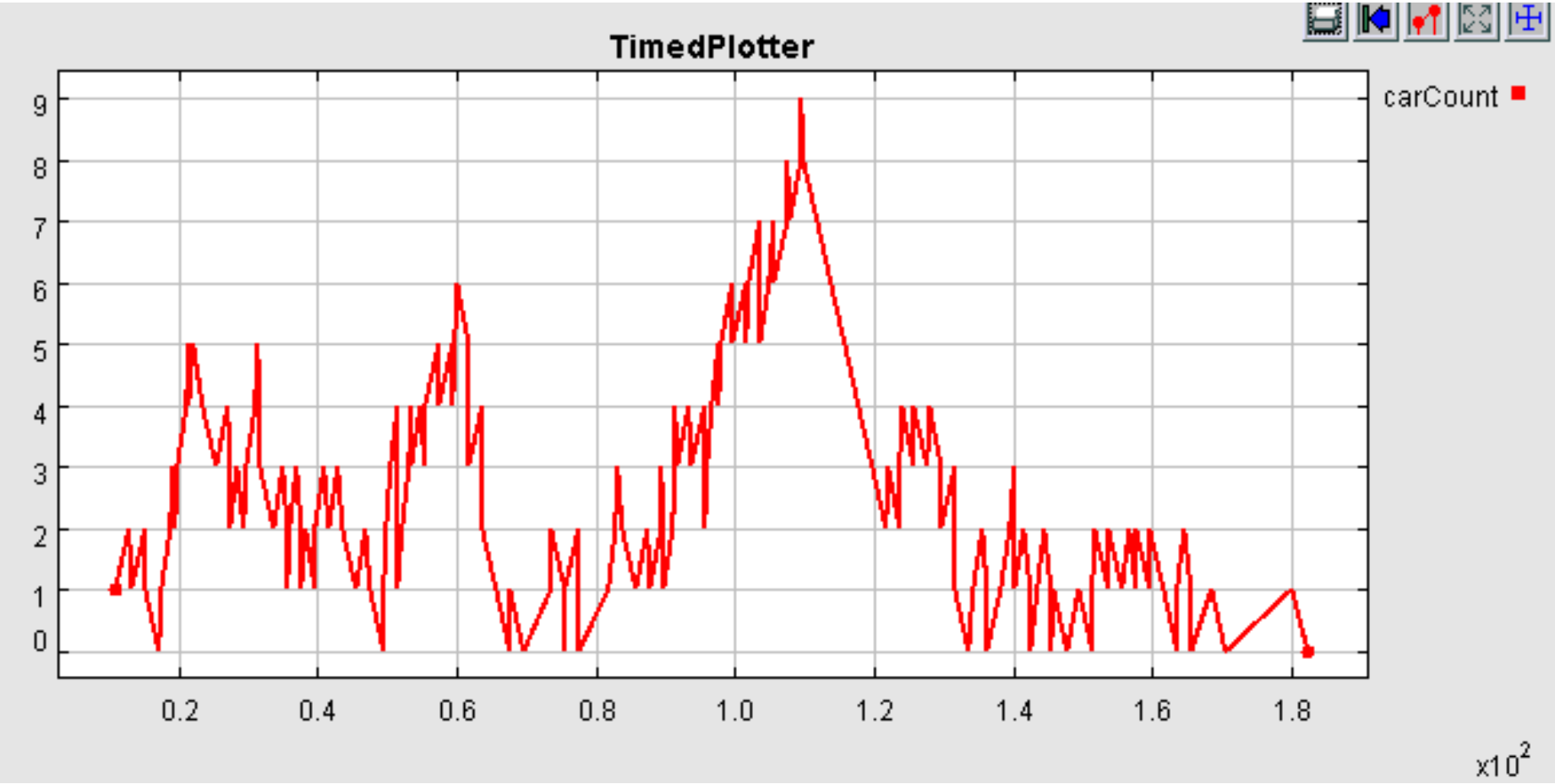
Intersection Wartezeit in Sekunden (200 Autos)

Intersection	Durchschnittliche Wartezeit
T-Intersection(1)	3,18
T-Intersection(2)	2,57
Cross-Intersection(3)	22,22
Cross-Intersection(4)	9,9
T-Intersection(5)	1,82

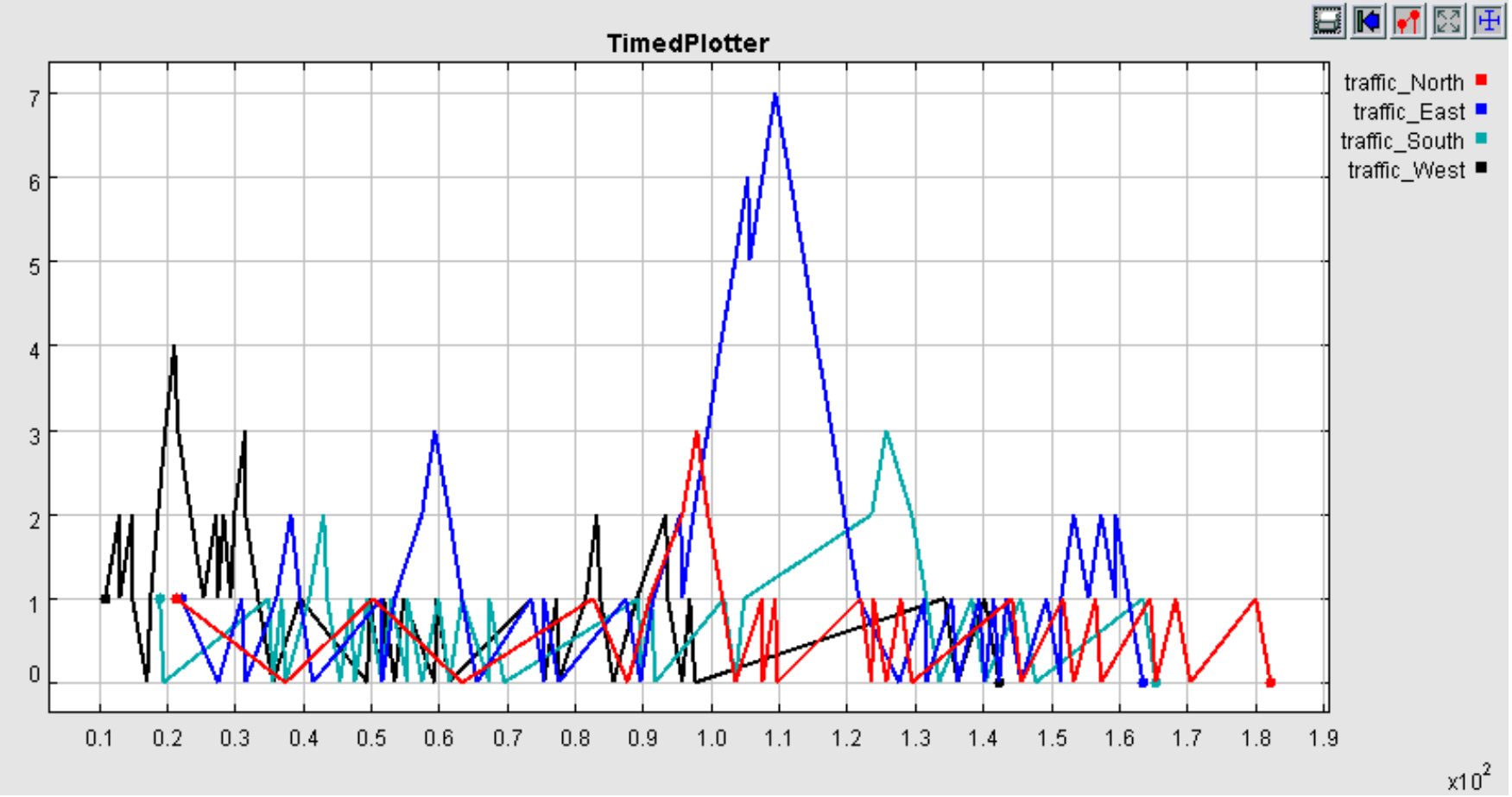
Street Wartezeit in Sekunden (200 Autos)

Street (verbundene Intersections)	Durchschnittliche Wartezeit (L -> R)	Durchschnittliche Wartezeit (R -> L)
1 (IO-1)	1,45	1,07
2 (IO-2)	2,3	1
3 (2-1)	1	1
4 (1-4)	1	1
5 (2-3)	1	1
6 (IO-3)	1,32	1
7 (3-4)	1,04	1
8 (4-IO)	1	1,05
9 (5-4)	1	1
10 (5-3)	1	1
11 (IO-5)	1,3	0,98

Total Intersection Count (3)



Individidual Direction Count (3)



Street Input Output (6)