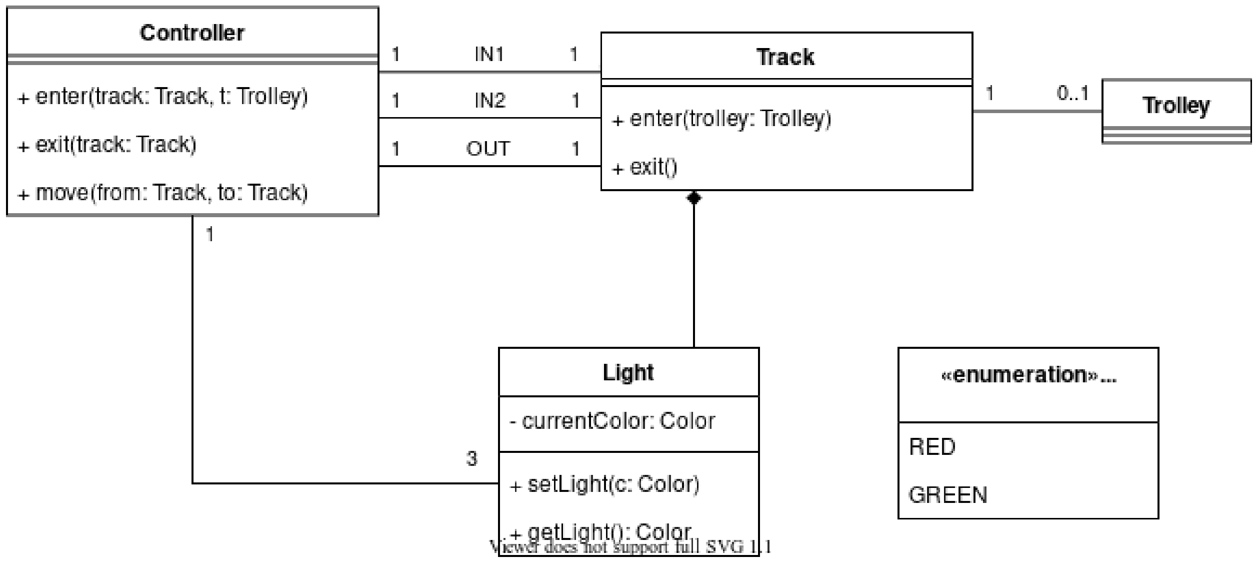
# Requirements Checking Assignment

By Arno Deceuninck (s0181217) and Dogukan Altay (s0211552)



The requirements we have to verify are:

3. If a trolley is on the junction, all traffic lights will remain red until that trolley has left the junction. (called remain red)
4. If two trollies are waiting to enter the junction at the same time, permission will be granted in order of arrival (i.e., the first trolley to arrive will get a green light, and the second one has to wait). (called First Come First Served (FCFS))

## Use Cases

### Remain Red

| | |
|---|---|
| Use Case Name | RemainRed |
| Scope | Junction Light Controller |
| Level | Subfunction Level |
| Intention in Context | The intention of the Trollies is to pass the junction without accidents. For this reason, all lights must remain red when a trolley is on the junction until |

| | |
|---|---|
| | it has left the junction. |
| Multiplicity | Only one instance of RemainRed running at any moment |
| Primary Actor | Light |
| Secondary Actors | Trolley, Track, Controller |
| Main Success Scenario (sequence of interaction steps) | 1. Trolley 1 enters the junction. 2. Now trolley 1 has entered, all lights become red and remain red until Trolley 1 leaves the junction. 3. Trolley 1 leaves the junction and the lights no longer have to remain red. |
| Extensions & Exceptions (additional / alternative interaction steps) | / |

## First Come First Served

| | |
|---|---|
| Use Case Name | FirstComeFirstServed |
| Scope | Junction Light Controller |
| Level | Subfunction Level |
| Intention in Context | The intention of the Trollies is to pass the junction as fast as possible. The intention of the controller is to avoid any starvation for any trolley that is waiting to enter the junction. For this reason, the lights will become green in order of arrival. |
| Multiplicity | Only one instance of FirstComeFirstServed running at any moment |
| Primary Actor | Controller |
| Secondary Actors | Trolley, Track and Light |
| Main Success Scenario (sequence of interaction steps) | 1. Trolley 1 arrives at the controller, but has a red light. 2. Trolley 2 arrives later at the controller, but has a red light. 3. The light of the track of Trolley 1 becomes green and Trolley 1 enters and leaves the junction. 4. The light of the track of Trolley 2 becomes green and enters the junction. |

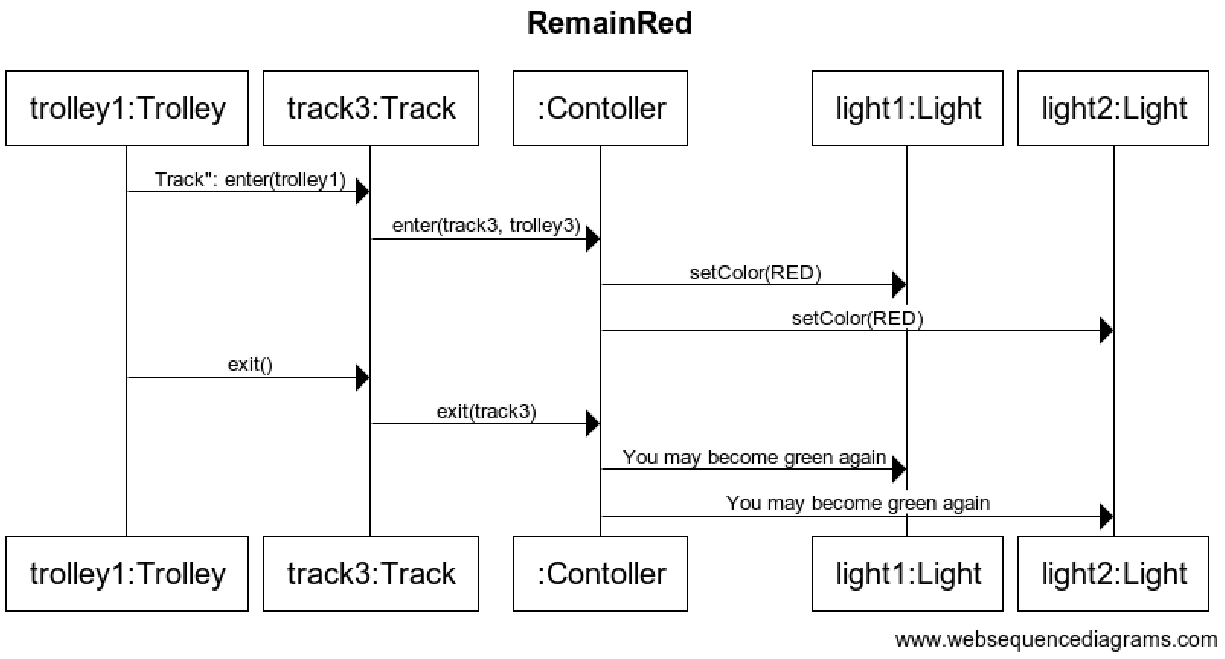
|  |  |
|---|---|
| Extensions & Exceptions (additional / alternative interaction steps) | 2a. Trolley 1 and 2 arrive at the exact same time. In this case, the trollies will be prioritized based on the track they're coming from. The Trolley entered via Track IN1 will be called Trolley 1, the Trolley entered via Track IN2 will be called Trolley 2. |

## UML Sequence diagram

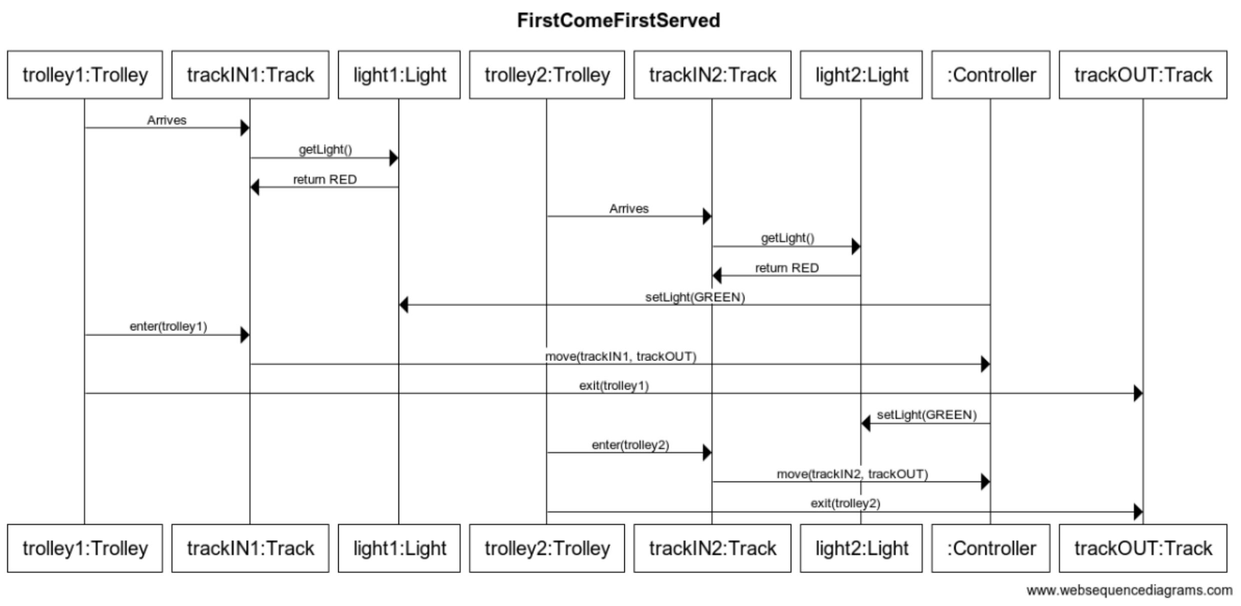
The diagrams were created using this site: https://www.websequencediagrams.com/

### Remain Red

We had to ignore light 3 here, otherwise there wouldn't be a way for the trolley in the junction to leave. We interpreted track 1 and 2 as the tracks leading to the junction and track 3 for when you are on the junction, so the lights only remain red when the trolley is on track 3 (which is the junction for us).



### First Come First Served



# Regular Expressions

## Mappings used

| | |
|---|---|
| E | A trolley enters the specified segment |
| R | A red signal is sent to the specified segment |
| G | A green signal is sent to the specified segment |
| X | A trolley leaves the specified segment |
| 1 | Left incoming railway segment |
| 2 | Right incoming railway segment |
| 3 | Outgoing railway segment |

## Assumptions

- We assumed that X1 and X2 means E3 as entering to the junction. We have examined all the output traces and clearly see that there is no other trace found between X1 or X2 and E 3. Thus we say that while X2 != X3, X1=E3 and X2=E3.

- Although otherwise is not specified but after trace examination, we deducted that there is no G3 in the traces. Therefore we excluded its existence to simplify our regexes and implementation.

## Remain Red

We're using a negative match for this regular expression. When a trolley enters the junction, we don't want a green light before the trolley leaves the junction, so we don't want some instructions that isn't the trolley that leaves and then a green light.

If any green light occurs after E 3(aka. Being on the junction) this regex will find a negative match. Our assumption is, after any X1 or X2 there is a E3 which means we can use E3 as our starting point. As a result, our regex is significantly shorter.

**((E 3)\n)([RE] [12]\n)*(G [12]\n)**

## First Come First Served

We're also using a negative match for this regular expressions. When two trolleys have entered, we don't want the last one to enter to get a green light first.
The function of light three is still unclear, but since it isn't used in any of the traces, we left it out of the regular expression.

Anything except a green light for track 1 or a trolley that enters track 2:
([RX] [123]|G 2|E [13])\n

Anything except a green light for track 1 or 2: ([REX] [123])\n

For when a Trolley on track 1 enters first:
E 1\n(([RX] [123]|G 2|E[13])\n)*E 2\n(([REX] [123])\n)*G 2
For when a Trolley on track 2 enters first:
E 2\n(([RX] [123]|G 1|E[23])\n)*E 1\n(([REX] [123])\n)*G 1

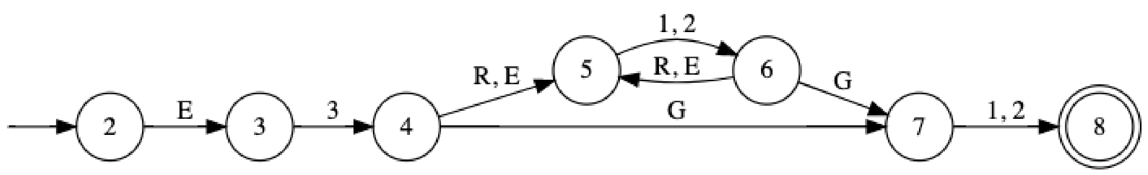
So combined, this gives us:
**(E 1\n(([RX] [123]|G 2|E[13])\n)*E 2\n(([REX] [123])\n)*G 2) | E 2\n(([RX] [123]|G 1|E[23])\n)*E 2\n(([REX] [123])\n)*G 2**

If any substring of the trace matches this regular expression, the requirement isn't satisfied.

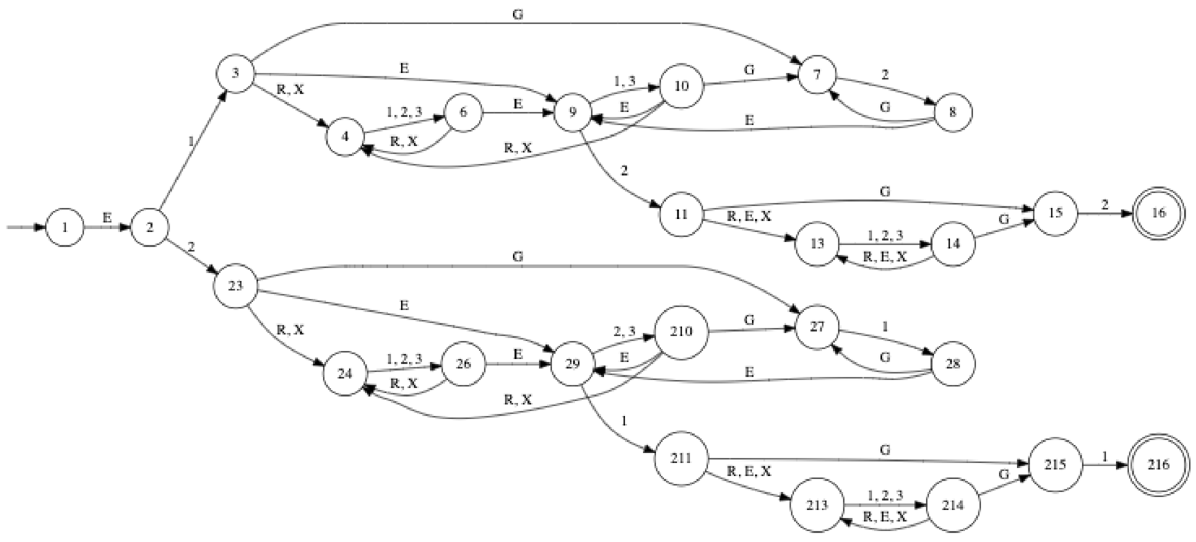
# Finite State Automaton

We used GraphViz https://edotor.net/ for this. Some transitions contain multiple characters, which means all the characters must follow sequentially to take that transition. A comma represents “or”, meaning just any of the strings separated with a comma must match. If there are no matches, you must go to a trash state. Once arrived in the accepting state, the requirement isn’t satisfied. Whitespaces and newlines are excluded to shorten the FSA diagram and code implementation.

## Remain Red



## First Come First Served



# Implemented FSA

## Remain Red

The FSA below implemented in the scanner.py as RemainRed object.

# First Come First Served

The FSA below implemented in the scanner.py as FirstComeFirstServed object.

# Trace results

For evaluating test results we have used scanner.py framework which gave us whether a trace.txt input violated or not. However, in order to debug and evaluate the trace results we have used the regex101.com website. Also including whole traces in the report would be too long. Therefore we supplemented the scanner.py traces as a separate txt file in the submission. Also We only provide some examples from the website and give how many violations occurred in which traces.

## Remain Red

***No violations found in all of the trace files.***

## First Come First Served

Trace1.txt: No violation found.
Trace2.txt: 77 Violations found in the trace. Therefore Use Case 4 violated.
Trace3.txt: 57 Violations found in the trace. Therefore Use Case 4 violated.
Trace4.txt: 2 Violations found in the trace. Therefore Use Case 4 violated.
Trace5.txt: No violation found.
Trace6.txt: 12 Violations found in the trace. Therefore Use Case 4 violated.

Example Violation For Trace2.txt:
E 1R 1R 2R 1R 2R 1R 2R 1R 2R 1R 2R 1R 2R 1R 2X 3E 2R 1R 2R 1R 2R 1G 2
Clearly shows that system given permission(aka. Green light to the specified segment) while a trolley in segment 1 waits for permission.

# Violated requirement

As a result, we found out that Use Case 4 violated in 4 different system traces. All violations caused by giving permission to latest trolley that arrived at the junction.