Traffic Control International- Report -

Group 8

December 15, 2019

Contents

Chapter 1

Introduction

The report is a documentation of the requirements, workflow and implementation of group 8, whereas the intended result of the project is to work together as close as possible and apply design patterns. The case study entails the task to develop a complex traffic light management system for a new start-up company with the objective to take over the Dutch and German markets. By moving away from orchestrated traffic light controls to intelligent traffic light control, the company intends to extend it's reach to the remaining members of the European Union. This task is very complex and demanding, as it's core features extend into two directions. Due to the nature of the different directions of implementation, the core values lie in maintainability and extendability.

The report will explain the various phases and tools group 8 had to employ to work in an efficient manner. By making the most use of a collaboration tool, weekly deadlines were set and divided into individual sprints. Using the tool in conjunction with an agile process framework created a clear overview of the remaining to-do's and completed sprints.

The report will be including individual reports of each group member, reflecting on the participation in a personal report and what each group member has individually learned from this group project.

Chapter 2

Design

2.1 Overall Architecture

In this chapter we will describe the overall architecture of our project.

2.1.1 General

We chose to aim for a very extendable and modular design of our application. In order to do so, we applied the design patters described in the *Design Patterns* section of this report. To make our application as extendable as possible we wanted a clear separation between Business Logic and GUI . We managed to create a traffic controll system - including the simulation of cars - that is running independently from to GUI and therefore usable in almost every context.

2.1.2 Business Logic

As a result of our modular design we have some entites that can compose a whole structured map of Intersection that later could even communicate with each other (think green wave). Our current application consists of *Car Lanes* which is a lane that can be connected to any other lane. We used them to connect the main elements of our application, the *Intersections*. There can be different implementations of an Intersection with varying numbers of traffic lights, lanes containing pedestrian crossings or not. At any time a new Implementation of an Intersection can be added. For the use in different countries we used the Strategy Pattern to give the traffic lights different behaviours for switching their states defined by our use of the State Pattern.
Our implementation of an Intersection is a *Simple Pedestrian Crossing* which consist of two *Inbound Roads*. An Inbound Road has (in this implementation) only one *Inbound Lane*, which is similar to a Car Lane, but contains a *Traffic Light*. An Inbound Road could have multiple Inbound Lanes for going left, right or straight.

2.1.3 GUI

We chose to use JavaFX for our GUI. In contrast to other groups other groups we are using a JavaFX-Canvas which is a JavaFX element that you can draw shapes or pictures on. Due to this fact we do not have to create JavaFX elements that are specified in an FXML file, but simply have to specify how our entities have to be drawn onto the canvas. *Drawables* can be drawn on a JavaFX's GraphicsContext and then be displayed in the GUI. This means every entity described in the Business Logic section has a JavaFX pondon that is specifying how something is displayed in the GUI.
To increase performance and make the application more structured we have different layers that can be either static or dynamic. Roads and Lanes do not change or move during runtime, so they only have to be drawn once. Traffic Lights however change their colour to they are on a separate dynamic layer that is continously drawn by a GraphicsEngine. The same is applicable to cars.

2.2 Design Patterns

Design patterns in software engineering describe reusable solutions for recurring problems within software design. As design patterns is an umbrella term for various patterns, there are three subpatterns with each their own approach to different problems. The following subsections subpatterns will be described in a short and concise manner and the usage of these patterns will be elaborated upon and how they were applied to the project.

2.2.1 Behavioral Patterns

2.2.1.1 Observer Pattern

subject traffic lights; observer: movable entitiy. The car gets notified of the traffic lights status. The moment the car gets added to the inbound lane class, the specific traffic light subject learns about the observer.

2.2.1.2 State Pattern

State enum: WAIT, READY, WARNING, GO. The states are implemented into the strategies and the traffic lights have a state. This makes the traffic light behaviour dependent on the corresponding state.

2.2.1.3 Strategy Pattern

country specific behaviour; extendability: The patterns responsible for the extendable part, essentially allowing to create new strategies if needed.

2.2.1.4 Template Pattern

GermanPedCrossingFxComposer: Template structure is defined in AbstractPedCrossingFxComposer; specific implementation in GermanPedCrossingFxComposer with country-specific return value.

2.2.2 Creational Patterns

Abstract Factory Pattern: abstract class AbstractTrafficLightFactory; country specific traffic lights extend this abstract class builder pattern: GermanPedCrossingFxComposer is director; builder: GermanTrafficLightFactory and GermanFxTrafficLightFactory composed by director singleton pattern: factory method:

2.3 Design Principles

- Program to an interface, not an implementation
- Identify the aspects that vary and seperate them from what stys the same.
- Classes should be open for extension, but closed for modification.
- Depend upon abstraction, not upon concrete classes.

Chapter 3

Development Process

This chapter contains scrum-related content. Scrum is a framework to provide (development) teams with an efficient working environment. In the following sections, we will describe in which way our group implemented this framework.

3.1 Scrum Roles

Usually, a scrum team consists of a scrum master, the development team and the product owner. The scrum master ensures that the scrum framework is being applied in the most effective way. But not as a leader in the classic sense, but more in a supportive way. The product owner represents the business and tells the development team what the company needs. The development team will then do the required work.

In our team we applied the roles in the following way: Obviously as, everybody was part of the development team everyone worked on the implementation, design part or report. As mentioned before, the product owner usually represents the business but in our case we were part of the business in question, meaing every team member took this role as well. Regarding the scrum master, we decided that it would be useful to rotate this role weekly. Every week a different team member slipped into the role of the scrum master. We chose this approach because this allows everybody the chance and possibility to gain experience as a scrum master.

3.2 Scrum Meetings

Meetings are an important element of agile development. There are several different types to it. Sprint planning, daily stand-up, review meeting and retrospective. Sprint planning is all about planning the upcoming sprint. During the planning, the team decides what to achieve in this iteration. In the daily stand-up, every team member briefly informs every other member about their

current progress of the sprint. At the end of each sprint, each member shows which backlog items they've completed (by their definition of done) - optional the team presents a demo including the new features. This is the so-called review meeting. The last meeting element is the retrospective. This session usually takes place at the end of each sprint. The retrospective is about what went well in the sprint and what could be changed to make the next sprint more productive. In our case, we did the sprint planning meeting at the beginning of each week (mondays) to distribute the tasks for the upcoming sprint. The daily stand-up was held briefly before we started to work. Every friday we discussed what we achieved in the sprint including a short demo of the new working version (Review Meeting). Afterwards we did the retrospective - we talked about the things that went well and in which areas we can improve for the next sprint. From these discussions, we derived some action items. These action items are concrete tasks how our teamwork can be improved. To keep track of the discussed topics of the retrospective, we decided to create a separate Trelloboard (more details see chaper 3.3).

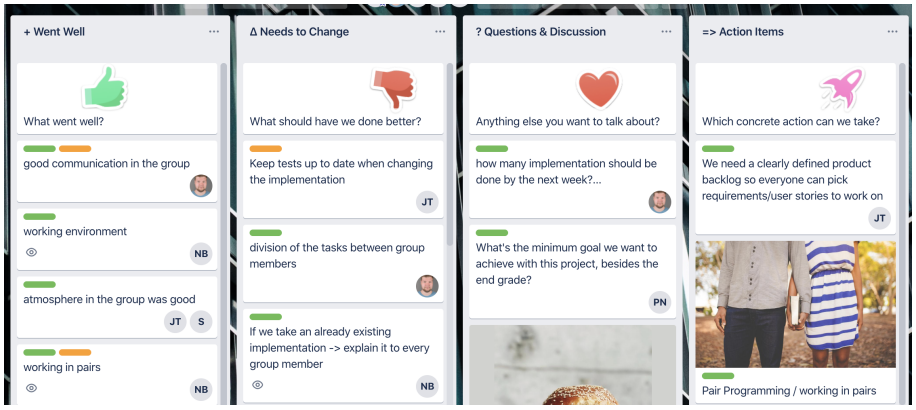

Figure 3.1: Retrospective Trelloboard

As you can see in figure 3.1 we created four different categories: Went Well, Needs to Change, Questions & Discussion and Action Items. In every retrospective session we discussed together what went well, where to improve and other project-related matters. For 5 minutes every group member created cards in the specific category. Afterwards we discussed the different aspects and asked ourself which concrete action can we take to improve our collaboration. As a result we deduced tasks, so called action items. Futhermore we labeled the cards to keep track of the results of each week. In our opinion such a retrospective Trelloboard is a functional and easy way to organize the scrum elements.

3.3 Kanban Board

A kanban board is a useful tool to visualize work, to-do tasks and other project-related information. In our case we used the tool Trello, a web-based project management software. We organized our trelloboard in the following way:

Figure 3.2: Trelloboard

As you can see in 3.2 the board is divided into six different columns with several cards. Cards are concrete tasks, for instance in the form of requirements. The first column, "Project Information", contains general project-related information and tasks. This category does not include programming-related work. In the "Backlog", every single requirement needed for the project is listed. Whereas the "Sprint Backlog" only contains the requirement for the current sprint. If you, as a developer, decide to tackle a specific task, you choose a card and move it into the "In Progress" column. When the task is done (by our definition of done), the card has to be moved to the "In Review" category. This status means that another developer has to review the completed task. After going through these various steps to ensure task completion, the card can be moved to the category "Done".

3.4 ScrumXP

Besides the aforementioned scrum methods, we also applied a part of the so-called "Extreme Programming" (XP) principle. Pair programming is an important component of it. In our opinion this approach has several advantages: if developers code in pairs, more often than not more efficient ideas can be implemented, due to the ability to immediately share information and various coding attempts. Furthermore, code review becomes unnecessary because the partner that works together in pair programming can immediately give feedback and review the code.

Chapter 4

Individual Reflection

This chapter will reflect upon our roles within the group, the project, the workflow with other group members and what we have learned during this project as individuals and as a team. For readablity purposes this introduction will lead over for the individual reports.

4.1 Nils Bauroth

First of all, I would like to point out that we as a group applied Pair Programming during the project. Many tasks were completed together. I was responsible/involved for instance in the following ones: SRS-related tasks: introduction, use case descriptions, requirements Report-related tasks: design, development process implementation-related tasks: graphic elements, (first) simple GUI, extendability, and exchangeability of a traffic light (green to red and red to green behavior), "communication" between pedestrian and car traffic lights, scrum-related tasks: creating retrospective trelloboard, beeing a product owner & scrum master, participant/stakeholder of other groups sprint review event
During the project weeks, we gained a lot of different experiences in several scopes of action: programming itself, discussing and applying design patterns and principles, agile software developing (Scrum framework) and so on. I was able to gather valuable knowledge and broaden my horizon in the following areas: Since one of the main aspects of this project was the application of design patterns, I learned a lot in the area of creational and behavioral design patterns. Especially the reason when to use such a pattern, why a developer should use these and what the implementation looks like. Furthermore, I improved my java skills as well, particularly concerning the concrete implementation of design patterns and programming of a graphical user interface. I also learned a lot about scrum in general. As a result of the rotation of the Scrum Master role within our group I was able to gain experience in the following elements: support our group during a sprint, ensuring that everyone understands the Scrum theory and the importance and realization of the different scrum meetings. At the beginning

of the project, it was quite difficult to imagine what the implementation should look like. We did have a road map, but we were not sure about what we wanted to achieve. That was as well the reason why some of our requirements and expectations were slightly too high. Luckily, during a retrospective meeting, we decided to aim lower. The work atmospehere was relaxed and productive at the same time. Each member had a field of expertise and supported the other members in this area.

4.2 Glenn Jonkers

In this chapter I will explain my findings and tell about what I have done and learned during this project. When working together in a group of 5 we used the defined roles(Scrum Master, Product owner) to give everyone a taste of how a Scrum Master and product owner work. We also noticed that working together in pairs on code that needed to be written and on the SRS as well as this report was a preferred choice by me and the other group members.

My involvement was mostly in working on testing and helping with the writing of the SRS and working on timings used to sequence the trafficlight cycle according to a strategy. I started with working on the beginnings for Visuals for the application to work on, these ended up not being used due to time issues. Rather, we started using the individual trafficlights that were used in different states. These visuals displayed our planned ideas using three lanes in we wanted a car that could go and stop or whether a car could turn individually. Later on when progressed a lot more, I used my time to work on the testing part and the SRS, and worked in pairs on the testing for the created implementation.

The team worked well together using paor programming to work on parts of this project. We divided the tasks up to lighten our burdens in coding. We had some civilized arguments like in any group to proceed with our vision of the project, we used everyones ideas to come up with the solutions that were implemented in this project.

I learned a lot about patterns this project. We used several in this project and it has really helped me to understand why we should use patterns as a way to make our program more extendable and more flexible to work with. I also learned a lot about the scrum way of working and working in pairs to communicate with several people at the same time. Being the the scrum master for a couple of weeks makes you notice how important it is to have one to work in a scrum based environment.

I enjoyed working in this group and would like to think it went well. I am proud to have been part of this team.

4.3 Philipp Noz

For introductory purposes, I will lead the individual report off with a short recap on the purpose of the project and the case study in question. Afterwards I will explain the working process during project hours and what I have learned from this project.

The project requires a group of five members to work closely together, in such a way that communication doesn't turn detrimental for the project. The case study is about implementing a complex traffic light management system for an outside company. The case study entails that extendability and maintainability are core features to the system, which is where the importance of communication stems from. As oversight of the project can be lost easily, the need for a documentation system and an agile process framework arises. Each project hour, we started with a quick stand-up meeting in order to update every group member on everyone's status. Afterwards we would check in with the other group members to ensure they can keep working on their sprint and everything is in order. Had any team member a question, we would offer full assistance and occasionally switch into pair programming for short durations, to increase efficiency.

Due to the team's environment playing an important role for group projects, we decided to take a different approach as mentioned before in a previous chapter. The group did not assign designated roles, as this creates a very unflexible working environment, which was unfavorable in our opinion. For this reason one of the few roles, for example the scrum master, have been rotated weekly among the members. Doing so allowed each group member to experience the workflow and circumstances created by each new environment. Personally I can speak from experience now, that the scrum master role is not an easy role to fulfill, despite appearances. Treading a fine line between encouraging members to work with motivation and making members work against their will on something else an be very difficult at times.

As we decided to split the work accordingly, we also made the decision to work in pair programming. The advantages of doing so have been stated in a previous chapter (view Chapter 3.4 ScrumXP). Code I have worked on includes implementing exchange- and extendability of traffic lights, essentially coding the red to green behavior and vice versa. I also worked on code that synchronizes the traffic light of cars with the pedestrian lights, so that their individual states wouldn't overlap. On top of that, I worked on additional files, some of which have been chosen to be overwritten or changed, as we progressed through the project, due to difficulties of further implementation or extendability. In pair programming we also changed code in a way which would impact the quality of life for the developers, in this case my team members, to create overall greater readability. On top of that, me having proficient english skills, I took upon myself to proofread both the entire SRS and the report, corrected mistakes and

checked both documents for consistency. I also wrote the introduction for both the SRS and the report, but not to grow complacent and get overconfident, I requested all texts written and proofread by me to be checked for mistakes I may have made or missed.

I have learned a lot from this project, not only due to the tasks nature, but also from my team members. To begin with, I have realized the incredible importance of strategies for extend- and maintainability in projects. Due to constant possibility of having to add features, renew software or just increase performance, strategy pattern become indispensable tools. Even though strategy patterns are that much of a life-changer for creating a flexible working environment, it also required that much more communication due to it's complex nature. I am glad to have had communicative team member, willing to take their time to explain their code to other team members.
Another important change compared to previous project was the switch to a new versioning tool called "GitHub". In comparison to the old versioning tool we used, there were a lot fewer errors and complications in the working process, which greatly improved the overall quality of life of the project.

I am very thankful towards my team members and look forward to working with them again in the future.

4.4 Sven Rediske

In this chapter I will show what I worked on and what I have learned during this project.
In the first weeks my main task was to work out the class diagram and thinking about our general design. I continued by working on the traffic lights with their startegies making them exchanagble during runtime. For the next task I partly worked together with Jonas. We synchronized the traffic lights in the intersection and managed the composition of all the sub elements of an intersection. In the last week I also tried making the bulb covers of the traffic lights exchangeable which works now but is not finished properly for all traffic lights.
While composing all the objects into the intersection we found out that we should not abstract too much because it simply is out of scope for this 7 week project. More important than that I learned that we should create even smaller trello tasks and not choose too many tasks for a sprint in a planning meeting. This is what we learned through a retrospective. I also learned that pair programming makes a lot of sense. This is due to the fact that the code you write is less error prone and together one always can think about the bigger picture while the other one is coding.
At the beginning of the project it was hard to organize because the task was very vague. I think it was good to work on the class diagram for a couple weeks week but on the other hand that was not very agile. We could have just started implementing something and let the diagram follow.

To conclude I have to say that it would have been nice to have some more time to practice scrum even more than we already did. I also learned a lot about when and how to apply design patterns in an almost real project.

4.5 Jonas Terschlüsen

To start off my individual reflection, I will first describe my participation in the different parts of project three in detail. Following up is a personal assessment of what I learned during the group work in project three.
In the first weeks of this project we started to familiarise ourselves with the business domain. Whilst doing so, I provided my project from the DARC (Design and Architecture) practical work, as a starting point. This project already included a cycling trafficlight using FXML-elements that changed their colour as a GUI.
Since we already started with a working project, we chose to aim for a more modular design using JavaFX-Canvas for displaying our application. I developed most of the GUI, consisting of the GraphicsEngine and the SimulationLayers that are responsible for the continous drawing of e.g. Cars or TrafficLights on the canvases. Whilst working on the implementation, I was maintaining the consistency of the class diagram, which all of us neglected in the last two weeks of the project, since there were many changes made to the actual implementation. Sven and I took care of assembling all of the sub entities inside of a Composer-Class that creates everything needed for an Intersection to exist and function, using pair programming. The complexity in this task was to position everything relative to each other.
In addition to that, I made a class that is connecting intersections using car lanes (*SPCConnector*) and can connect lanes to a certain position.
I created the cars and adapted the lanes so cars could actually drive on them. To make the application more interesting whilst presenting I also implemented a spawner that is spawning cars at a certain rate on a certain lane. Lastly I assembled two intersections and two spawners to the application that was presented on Friday.

What did I learn during project three? To apply Design Patterns in a large project and where to use them. Whenever a problem occured we thought of a design pattern that could solve it and in almost every case it solved the problem. (e.g. Traffic Lights and Cars: How do they communicate? $\rightarrow$ Observer Pattern) This project also showed me that applying design patterns and trying to make an application as extendable as possible, comes at a high price: very high complexity and a vast amount of classes and packages.
There is also a problem when working agile in this project, because without a good initial design, you will spend a lot of time refactoring and changing the design, making your work very inefficient. Additionally there is the problem of iterdependency between requirements making the partition of work not always easy. So I think a full on agile approach is not always the way to go.

My personal conclusion for our project is positive. We first had the aim to be able to present multiple four-way-crossings with a green wave, cars and pedestrians. But then realised that we would not manage to reach this goal. Therefore we adapted the scope to a Simple Pedestrian Crossing and driving cars that react to the traffic lights on intersections.

Traffic Control International - SRS -

Group 8

December 15, 2019

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to give a detailed description of the requirements for the traffic control software. It shall make clear to all stakeholders which operating system, which libraries shall be used, which interfaces the system will have, in a nutshell all requirements to satisfy the customer.
This document is written for all stakeholders of the project, which includes the start-up Traffic Control International, the scrum team which has to implement the software and the Dutch and Germany's Ministry of Transport.

1.2 Scope

Traffic control international is a new start-up which aims to provide a full service traffic management system, which means that the product covers traffic control for bicycles, cars and pedestrians across cities. Because the system will supervise every traffic control related element, it does not have sub or supersystems and is therefore independent. The end goal is a smart traffic handling system which can be reused in most countries by just adding the country specific traffic behaviour to the system. This shall be achieved by special focus on maintain- and extendability. The new system will replace existing traffic systems in the Netherlands and Germany first and if the service is successful, it will be expanded to other countries.

1.3 Definitions, acronyms and abbreviations

Table 1.1: Data Dictionary

Number	Term	Definition
1	The system	The system described in this document
2	Shall	Requirements marked with “shall” must be implemented and verified
3	Will	Requirements marked with “will” must be implemented and cannot/will not be verified
4	Should	Describes a design goal the system tries to meet
5	UC	Use case
6	Emergency mode	The mode is activated for traffic control, when a police, ambulance or government vehicle sends a radio signal
7	Button	A pedestrian button to request green
8	TL	Traffic light
8	TCI	Traffic control international (the company this software will be developed for)
9	Stakeholders	Every person interested or concerned about the project
10	bulb cover	The cover of a traffic lights bulbs. E.g. walking person, arrow, donkey of Wesel etc...

1.4 References

connect.fontys.nl
Wikipedia.org Road traffic control
Dutch traffic regulations
.

1.5 Overview

- full service: isolated traffic control, crossing control, multiple crossing control, smart crossing control (from easy to difficult traffic scenarios)
- ”main case”: 3 lanes (left, right, straight)

Chapter two contains general business related information such as the product functions, user characteristics, overall description of the product and other inherent restrictions. Chapter three provides detailed requirements about external interfaces, non functional reqirements and design constraints.

Chapter 2

Overall description

2.1 Product perspective

Figure 2.1: Diagram

2.1.1 User interfaces

Not applicable to our program

2.1.2 Hardware interfaces

- TrafficLight
- Pedestrian button
- Car detector
- InboundRoad
- Intersection
- Lane
- Bulbcover

The system needs an interface to the trafficlights, pedestrian buttons, normal car detector and an InboundRoad, Intersection, Lane, and Bulbcover to visualize. (could also be a software interface to the existing "intersection software")

2.1.3 Software interfaces

- monitoring and controlling software of traffic management organisations

The system needs a software interface to the existing traffic monitoring and controlling software.

2.1.4 Communication interfaces

The only communication interface this system needs is https to the responsible traffic management organisation.

2.1.5 System interfaces

No System interfaces

2.1.6 Memory Constraints

No memory Constraints

2.1.7 Site adaption requirements

- update software in the existing traffic management software

2.2 Product functions

Figure 2.2: Use Case Diagram

2.2.1 Use Case Descriptions

Single pedestrian light	
ID	**UC01**
Actor	Pedestrian
Precondition	None
Scenario	1. Actor approaches at the light 2. Traffic light is red 3. Actor presses button 4. Traffic light switches from red to green
Exceptions	None
Extensions	None
Result	Pedestrian light switches from red to green

Table 2.1: Use Case Description: "Single pedestrian light"

Cycle single pedestrian light	
ID	**UC02**
Actor	None
Precondition	None
Scenario	1. Pedestrian light is red for n seconds 2. Pedestrian light switches from red to green 3. Pedestrian light is green for n seconds 4. Pedestrian light switches from green to red 5. Return to step 1
Exceptions	None
Extensions	None
Result	Pedestrian light switches from red to green

Table 2.2: Use Case Description: "Cycle single pedestrian light"

Cycle single car traffic light	
ID	**UC03**
Actor	None
Precondition	None
Scenario	1. Traffic light is red for n seconds 2. Traffic light switches from red to green 3. Traffic light is grteen for n seconds 4. Traffic light switches from green to red 5. Return to step 1
Exceptions	None
Extensions	None
Result	Traffic light switches from red to green

Table 2.3: Use Case Description: "Cycle single traffic light"

Pedestrian Crossing	
ID	**UC04**
Actor	Pedestrian
Precondition	none
Scenario	1. The Actor arrives at the traffic light 2. The System show a red light on both traffic lights 3. The Actor waits until the traffic lights turn green 4. The System shows a green light on both traffic lights 5. THe Actor crosses the street
Result	The Actor has crossed the predestrian crossing

Table 2.4: Use Case Description: "Pedestrian Crossing"

Four Way Crossing	
ID	**UC05**
Actor	Car
Precondition	none
Scenario	1. The Actor arrives at a crossing, driving on the lane heading west 2. The System shows a red light for the lanes going east and west, a green light for the lanes going north and south 3. The Actor waits 4. The System switches the traffic lights from east and west to green and the traffic lights going north and south to red 5. The Actor crosses the crossing
Result	The Actor crosses the crossing

Table 2.5: Use Case Description: "Four Way Crossing"

Additional Signals (e.g. Green Arrow)	
ID	**UC06**
Actor	none
Precondition	Turn lane
Scenario	1. Traffic light is red for n seconds 2. Traffic light switches from red to green 3. Traffic light switches from red to green special signal 4. Traffic lights are green for n seconds 5. Traffic lights switch from green to red 6. Return to step 1
Result	The additional signal turned green

Table 2.6: Use Case Description: "Additional Signals (e.g. Green Arrows)"

Additional modes (e.g. emergency)	
ID	**UC07**
Actor	none
Precondition	emergency
Scenario	1. Traffic light is red for n seconds 2. Traffic light system enters emergency mode for n seconds 3. Traffic light system returns to normal mode
Result	The Traffic light system enters and leaves emergency mode

Table 2.7: Use Case Description: "Additional modes (e.g. emergency)"

Four Way Crossing including Pedestrian lights	
ID	**UC08**
Actor	Pedestrian, Car
Precondition	none
Scenario	1. The Car drives from east to west 2. The System shows a red light for the lanes from east to west and a green light from south to north 3. The Pedestrian walkes from south to north 4. The System shows a green light for the lanes from south to north 5. The Pedestrian crosses the street 6. The System shows a red light for the lanes from south to north and a green light for east to west 7. The Car crosses the intersection
Result	Actors have crossed the street

Table 2.8: Use Case Description: "Four way Crossing including Pedestrian lights"

2.3 User characteristics

2.4 Constraints

- no real constraints
- road map as orientation
- only "constraints" = special focus on maintain- and extendability

2.5 Assumptions & Dependencies

2.5.1 Assumptions

- There are no Assumptions taken.

2.5.2 Dependencies

- Java / JavaFx

Chapter 3

Specific Requirements

3.1 External interface requirements

This Section of the IEEE 830-1998 standard is not applicable for this document.

3.2 Functional Requirements

ID	**FREQ001**[1]
UC[2]	**UC01 - UC08**
Category	functional
Description	A traffic light shall have an exchangeable red to green behaviour.
Rationale	To be usable in different countries

Table 3.1: Functional Requirement: "exchangeable red to green behaviour"

[1] Functional Requirement
[2] Associated Usecase

ID	**FREQ021**[3]
UC[4]	**UC01 - UC08**
Category	functional
Description	A traffic light shall have an exchangeable green to red behaviour.
Rationale	To be usable in different countries

Table 3.2: Functional Requirement: "exchangeable green to red behaviour"

ID	**FREQ002**
UC	**UC01, UC02**
Category	Functional
Description	The Pedestrian Traffic Light shall have a button available for the actor to interact with.
Rationale	To be usable in different countries

Table 3.3: Functional Requirement: "Pedestrian Button"

ID	**FREQ003**
UC	**UC01 - UC08**
Category	Interface
Description	The Traffic Light shall have an exchangeable bulb cover.
Rationale	To be usable in different countries

Table 3.4: Functional Requirement: "Bulb Cover"

ID	**FREQ004**
UC	**UC01 - UC08**
Category	functional
Description	Road Users[5] should not be crossing a red light.
Rationale	A red light signals road users to stop, so others can go.

Table 3.5: Functional Requirement: "Stop at red light"

[3]Functional Requirement
[4]Associated Usecase
[5]Everyone in traffic

ID	**FREQ005**
UC	**UC03 - UC08**
Category	functional
Description	Cars that approach a green traffic light should stop if there are pedestrians on the road.
Rationale	Car kills pedestrian.

Table 3.6: Functional Requirement: "You must not murder.(Exodus 20:13)"

ID	**FREQ006**
UC	**UC05, UC08**
Category	functional
Description	Only Traffic Lights, on one crossing, (when not separate turn lane) that are facing in directly oposite directions, shall display the same signal. E.g. North and South = GO; West and East = WAIT;
Rationale	Avoid crashes.

Table 3.7: Functional Requirement: "You shall not crash!"

ID	**FREQ007**
UC	**UC05, UC08**
Category	functional
Description	A car may only change lanes if there is enough space on new lane, next to the car.
Rationale	Avoid crashes.

Table 3.8: Functional Requirement: "Changing lanes"

ID	**FREQ008**
UC	**UC04, UC05**
Category	functional
Description	A pedestrian shall only travel on walkways.
Rationale	Pedestrians are not supposed to walk on a road.

Table 3.9: Functional Requirement: "Walkways"

ID	**FREQ009**
UC	**UC05-UC08**
Category	functional
Description	A car shall be able to decide which lane to use.
Rationale	To be able to switch to a turn lane.

Table 3.10: Functional Requirement: "Lane choice"

ID	**FREQ010**
UC	**UC05, UC08**
Category	functional
Description	A car shall be able to decide in which direction to move on when approaching an intersection.
Rationale	Cars should be able to turn.

Table 3.11: Functional Requirement: "Cars change direction"

ID	**FREQ011**
UC	**UC04, UC08**
Category	functional
Description	Car Traffic Lights and Pedestrian Traffic Lights shall be in sync. (If car == GO) {pedestrians = WAIT}
Rationale	To not run over pedestrians

Table 3.12: Functional Requirement: "Synched Traffic Lights"

ID	**FREQ012**
UC	**UC03, UC05 - UC08**
Category	functional
Description	A car shall be able to move forwards, backwards, to the left and to the right.
Rationale	To move in traffic.

Table 3.13: Functional Requirement: "Cars can move"

ID	**FREQ013**
UC	**UC03, UC05 - UC08**
Category	functional
Description	A car shall be able to change its rotation.
Rationale	To be able to turn.

Table 3.14: Functional Requirement: "Cars can turn"

ID	**FREQ014**
UC	**UC01, UC02, UC04, UC08**
Category	functional
Description	Pedestrians shall be able to move forwards, backwards, to the left and to the right.
Rationale	To move in traffic.

Table 3.15: Functional Requirement: "Pedestrians can move"

ID	**FREQ015**
UC	**UC01, UC02, UC04, UC08**
Category	functional
Description	If a car traffic light is green in a four way crossing, the perpendicular pedestrian lights can also be green.
Rationale	To get more throughput in the intersection.

Table 3.16: Functional Requirement: "Pedestrians can go when perpendicular cars have green"

ID	**FREQ016**
UC	**UC05, UC06, UC08**
Category	functional
Description	A lane shall be able to contain more than one car.
Rationale	Common sense

Table 3.17: Functional Requirement: "More than one car on a lane"

ID	**FREQ017**
UC	**UC05, UC08**
Category	functional
Description	All car traffic lights of an intersection shall be manually contollable. (Not Separately)
Rationale	Maintainance and energy saving at night.

Table 3.18: Functional Requirement: "Manually controllable intersection"

ID	**FREQ018**
UC	**UC01-UC08**
Category	functional
Description	Cars shall be able to accelerate when setting of a traffic light.
Rationale	Cars accelerate when they were at hold before.

Table 3.19: Functional Requirement: "acceleratation"

ID	**FREQ019**
UC	**UC01-UC08**
Category	functional
Description	A car shall be able to decelerate when approaching slower car or a set of traffic lights.
Rationale	To avoid crashing.

Table 3.20: Functional Requirement: "deceleratation"

ID	**FREQ020**
UC	**UC01-UC08**
Category	functional
Description	The red-green-red time shall be changable.
Rationale	To ensure extendability.

Table 3.21: Functional Requirement: "Changable red-green-red time"

ID	**FREQ021**
UC	**UC01-UC08**
Category	functional
Description	An intersection shall be able to connect to another intersection via a CarLane.
Rationale	To make cars drive from one intersection to another one.

Table 3.22: Functional Requirement: "Connect Intersections"

ID	**FREQ022**
UC	**UC01-UC08**
Category	functional
Description	AN intersection shall be able to have a different rotation.
Rationale	To be able to build a propper roadmap.

Table 3.23: Functional Requirement: "Rotate Intersections"

	01	**02**	**03**	**04**	**05**	**06**	**07**	**08**
FREQ001	x	x	x	x	x	x	x	x
FREQ002	x	x						
FREQ003	x	x	x	x	x	x	x	x
FREQ004	x	x	x	x	x	x	x	x
FREQ005			x	x	x	x	x	x
FREQ006					x			x
FREQ007					x			x
FREQ008				x	x			
FREQ009					x	x	x	x
FREQ010					x			x
FREQ011				x				x
FREQ012			x		x			x
FREQ013			x		x			x
FREQ014	x	x		x				x
FREQ015	x	x		x				x
FREQ016					x	x		x
FREQ017					x			x
FREQ018								
FREQ019								
FREQ020								
FREQ021					x			x

Table 3.24: Tracablility Matrix for Functional Requirements

3.3 Performance requirements

ID	**UPREQ000**
Category	Nonfunctional
Description	The Time to cross the road can be calculated using the velocity of the person and the distance between the roads.
Rationale	To be usable in different countries

Table 3.25: Use Case Requirement: "Single pedestrian light in the morning"

	UC01	**UC02**	**UC03**	**UC04**	**UC05**
PREQ000	x	x	x	x	x
PREQ001	x	x	x	x	x
PREQ002	x	x			x
PREQ003	x	x			x
PREQ004	x	x	x	x	x
PREQ005	x	x	x	x	x

Table 3.26: Tracablility Matrix for Non-Functional Requirements

3.4 Logical database requirements

test

3.5 Design Constraints

- c1

List of Tables