UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Suido Valli

Visualization of simulations of a robot operated car park system

Master's Thesis (30 ECTS)

Supervisor: Dirk Oliver Theis

Tartu 2016

Visualization of simulations of a robot operated car park system

Abstract:

The aim of the MA thesis "Visualization of simulations of a robot operated car park system" is to identify the best practices and technologies to present the effectiveness of robot operated car park system's underlying algorithm to potential customers. An application demonstrating the work of the robots is built using the identified technologies and best practices. The user experience of the application and the application's compliance to the requirements of the tool are validated in the thesis.

Keywords:

Web application, simulation, visualization, Python, JavaScript, HTML canvas element

CERCS:

P175, Informatics, systems theory

Lühikokkuvõte:

Magistritöö „Robotiseeritud parkimissüsteemi simulatsioonide visualiseerimine“ eesmärgiks on identifitseerida parimad praktikad ja tehnoloogiad robotiseeritud parkimissüsteemi algoritmi efektiivsuse esitlemiseks potentsiaalsetele klientidele. Magistritöös leitud praktikaid ja tehnoloogiaid kasutades luuakse robotite tööd demonstreeriv veebirakendus. Töö lõpus hinnatakse rakenduse pakutavat kasutajakogemust ning vastavust seatud nõuetele.

Võtmesõnad:

Veebirakendus, simulatsioon, visualiseerimine, Python, JavaScript, HTML canvas element

CERCS:

P175, Informaatika, süsteemiteooria

Table of Contents

1 Introduction 5

2 Overview 6

2.1 Robot car park system 6

2.2 Similar solutions 7

PathFinding.js 8

VisuAlgo 8

Robotized car parking visualizations 9

3 Application requirements 10

3.1 Non-functional requirements 10

Scalability 10

Usability 10

Reliability 10

Correctness 10

Efficiency 10

Reusability 10

Portability 11

Extendibility 11

3.2 Functional requirements 11

Application is able to read the robroute file format 11

Application is able to construct parking lot and machines' instructions from a robroute instruction file 12

Application is showing static images of start and end state of the simulation 12

Application is visualizing the simulation of the algorithm 13

Application is able to scale accordingly to parking lot and viewport size 13

User is able to choose the scenario to visualize from application 13

4 System architecture 14

4.1 Server side platform selection 14

Python web framework selection 15

4.2 Data object transmitting standards 15

4.3 Client side platform selection 16

JavaScript Libraries 17

4.4 Machine movement approach 18

5 Application overview 20

5.1 Application flow 21

Rendering the index page 21
Choosing the parking lot and instructions 21
Rendering the main workspace 24
Visualizing the scenario 24
5.2 Application functions 25
Index page functionality 26
Visualization page functionality 27
6 Validation 29
6.1 Sanity testing 29
6.2 Efficiency of application 31
Client side efficiency 31
6.3 Integrity 32
6.4 Validation of presentation choices 33
7 Conclusions and future opportunities 35
8 References 36
Appendix 38
I. Glossary 38
II. Source code 39
III. Installation guide 40
IV. User guide 41
V. License 42

1 Introduction

In this thesis a new web application is introduced to visualize the simulation of robot operated car park system. The car movement algorithm is developed by Algorithms & Theory group, a sub-group of Theoretical Computer Science at the University of Tartu led by Dirk Oliver Theis. The objective of the thesis is to identify the best practices to present the workings of the mentioned algorithm to potential clients of the robot car park system. The thesis tries to find answers for following questions:

1. What technologies to use for making the application that complies with the set requirements derived from the business needs of the project?
2. What approach to use for visualizing machine movement in a parking lot?

As the research questions suggest, the main obstacles to overcome are to find the best technology stack and approach that satisfy the business needs. Objective of this project is to have an appropriate tool to present the effectiveness of the algorithm to potential enterprise clients. The research description and results for finding the best approach, tools and technologies can be found in chapter 4.

In addition, the thesis tries to identify the best apprach to visualize car parking, taking into account the comprehensibility of the car and robot movement by users. In the thesis, an approach to visualize the robotized car parking system is introduced and validated.

As a proof of concept, web application is built using the identified tools, technologies and practices. The thesis covers the extensive practical work during the development of the web application, the requirements of the application and validation of the web application's compliance with the requirements and user experience.

The findings of the thesis are demonstrated in developing the aforementioned visualization application.

2 Overview

In the following chapter the background of the application is described and compared to the similar solutions. In addition, it is explained why the proposed solution fits the best for the needs of the project.

2.1 Robot car park system

Nowadays parking a car is more and more challenging task – especially in metropolitan areas. The problem is that the cities are getting denser, there are more people and cars in the centre of the cities than there has ever been – but the area for parking spaces is not changing correlatively with the increase of cars.

To overcome that problem, cars have to be fitted to the parking lot more densely than it is possible in regular parking lots. There are options in the market to do that, for example Ferris wheel-like parking systems, also known as paternoster systems [1]. These systems have been around from 1920s and are starting to make a comeback in Europe and USA. The problem of paternoster system is the high cost of construction and the need to change existing infrastructure extensively.

Additional problem that can be observed in bigger or multi-story car parking lots is the time it takes to park and retrieve the car. It can take exceptionally long time, especially when the parking lot is operating almost in full capacity. Finding a place can be really arduous.

Newer parking lots and parking houses have got smarter. They have sensors that can identify if the parking space is occupied or not. Information received from the sensors can be aggregated to show how many parking spaces are free in the parking lot. It is additionally possible to have lights above every parking space to identify whether the space is free or occupied.

Although these systems have made parking more convenient, the capacity of a parking lot or parking house can be improved even more. The parking/retrieval time for the car owner can also be reduced dramatically.

This can be done by using an automated parking system that can use existing infrastructure with minimal changes needed. Suggested robot car park system (RCPS for short) is using a specified amount of small robots to park the cars.

The simplified flow of parking in RCPS parking lot from driver's perspective is as follows:

- Driver drives the car to the designated drop off area of the parking lot.
- Driver and passengers get out of the car.
- Driver locks the car and takes a ticket from the parking lot printer or uses a mobile application to park the car.
- Robot retrieves the car from the area driver left it.
- Robot takes the car into the parking lot and parks it.

As one can see, the process of a driver finding a parking space in a parking lot is eliminated fully from the parking process with RCPS.

The simplified flow of retrieving the car in RCPS parking lot from driver's perspective is described:

- Driver inserts the ticket received from the parking lot printer to the parking lot information system or driver uses mobile application to order the robot to retrieve the car.

- Robot retrieves the car from parking lot and parks it outside the parking lot in designated retrieval area.
- Driver and passenger(s) enter the car and drive away.

In conclusion, advantages of RCPS compared to usual parking lots are:

- Bigger car density,
- no damage on cars (paint scratches by other people parking etc.),
- no exhaust fumes (robots are working with electrical motors),
- dramatically reduced noise pollution (no combustion engines and people talking),
- quicker car parking and retrieval for clients,
- less stressful parking experience.

These advantages can be found in other automated parking systems compared to the usual parking lot, but not usually at once. One of the main advantages compared to other automated parking systems of RCPS is the relative cheapness of it.

As said before, the basic infrastructure of the already existing parking lot or parking lot does not need to be demolished and can be used as a basis for RCPS. All the needed components of RCPS can be installed easily to regular parking lot. The usage of existing parking lots is the main factor that differs RCPS's solution from other robotized parking systems.

The robots used by RCPS are rectangle shaped and low enough to drive under cars. Robots move cars by driving underneath the platform the car is on and lifting the platform up. The robot will carry car to the parking space using the instructions got from the main controller of the parking lot and lower the platform to the ground.

The movement of cars and robots is controlled by the main controller, that gives instructions to the robots what to do at what timeframe. The algorithm for the said controller is being developed in University of Tartu and is part of a project developed together with an external company.

To be able to present the movement of the machines in the parking lot and to prove that the RCPS algorithm works as intended, an application is to be made that can be used to satisfy these requirements.

2.2 Similar solutions

The main objective the RCPS application tries to accomplish is to show potential customers that RCPS is more efficient with the parking space than the usual parking lot and the customers of parking lot will have better user experience.

The user experience consists of, amongst other indicators, the time it is needed to park the car. This indicator is directly correlated to how effectively the parking spaces in the parking lot are used. In busy days, there can be times that all of the spaces of a parking lot are full. Now, the question is how does the RCPS handle these busy days and does it handle them better than people in usual parking lots would do that.

To present the RCPS to a customer, two different paths can be taken – the RCPS can be either presented on a big screen by the RCPS provider's company to the customer, or customer can interact himself or herself with the virtual RCPS parking lot. The objective is to build an application that can be used for both options.

In essence RCPS is an application that takes a parking lot as an input and will start executing movements on the screen relative of time. Also, it is, in essence, visualizing the work of an

algorithm. Taken those two approaches into account, similar solution can be found in algorithm visualization and time-based graphical movement. In some cases, they overlap.

PathFinding.js

PathFinding.js itself is a pathfinding library written in JavaScript for tile-based games[1]. The online demo of the library[2] is for visualizing the different pathfinding algorithms implemented in the library. Upon opening the demonstration, client is introduced to a screen with a grid where all but two squares are white. The green square is the start position and red square the end position for path finding. User can add obstacles to the grid by using the mouse and move the start and end positions from their original positions. There is possibility to choose from 8 different algorithms and compare how they find the solution for the problem. When the user starts the visualization, every step of algorithm is shown on the screen with grey, green or blue squares, depending on the algorithm. When the algorithm has finished, user can see the length of the path, time it took and how many operations did the algorithm have to do.

The PathFinding.js demonstration is similar to RCPS in many ways – it visualizes the work of an algorithm, is somewhat interactive (user can give the input for the program) and is an application in Web browser. On the other hand, there are many differences – PathFinding.js demonstration is meant for demonstrating only one type of algorithms and is not capable of working with parking lots. Furthermore, the movement of the grids is not smooth – something that is a requirement to make the visualization of machine's movement resemble real life situations as much as possible.

VisuAlgo

VisuAlgo is an application that visualizes data structures and algorithms through animation[3]. User has a plethora of different data structures to choose from where each of the data structures has its own algorithms that can be visualized. The visualization can be shown automatically or step-by-step. In addition, pseudocode of the algorithm with highlighted step is shown on the screen for better understanding of inner workings of the algorithm. VisuAlgo has two modes: exploration mode where a user can simply discover the data structures and algorithms that go with them him/herself and e-Lecture mode, where the user will first learn about the data structure and has a tutorial of how to animate the algorithms before actually using them.

The overlapping of RCPS's and VisuAlgo's functionality is quite small – the only similarity is the abstract algorithm visualization. VisuAlgo is showing the algorithms quite differently from what RCPS has to. The purpose of VisuAlgo is to educate people on how different data structures and algorithm works. The aim of RCPS is to show that the algorithm used for car parking robots is more efficient than the contemporary manual parking system. The algorithm beneath it is a trade secret and the inner workings of it are not meant to be public.

[1] https://github.com/qiao/PathFinding.js
[2] https://qiao.github.io/PathFinding.js/visual/
[3] http://visualgo.net/

Robotized car parking visualizations

Several robotized car parking visualizations can be found on the Internet, for example Park plus[4] and FATA Automated Parking System[5] videos. The visualization resembles the most with what are the requirements of the project – the movement of the robots and cars in the parking lot is shown. Visualization for RCPS differs from the videos by showing how the algorithm reacts to a particular configuration. RCPS visualization will allow the client to provide initial and terminal situations of the whole parking lot. The visualization will answer the client's question on what will the RCPS's algorithm do in this situation and if it will be able to retrieve the requested cars in time.

[4] https://www.youtube.com/watch?v=Dm-twxXt5D8
[5] https://www.youtube.com/watch?v=VwS1QwXqgpk

3 Application requirements

In the following chapter functional and non-functional requirements are described that the application to be created has to fulfil.

3.1 Non-functional requirements

Following non-functional requirements define the design of the application. The requirements are distributed to 7 subcategories.

Scalability

The application can display parking lot with up to 50 parking spaces in a row and 25 parking spaces in a column. The smallest parking space size is 25 pixels wide and 50 pixels high - this is to ensure that the movement of the machines can still be perceived by the users of the application. To ensure that the usage of the application needs minimal scrolling by the users, aforementioned parking lot limits are put to place taking into account the w3schools.com screen resolution statistics where in January 2016, the most popular screen resolution amongst the visitors of the site was 1920x1080 pixels [2].

Up to 200 users can use the application concurrently, which is the reasonable maximum amount of people in the audience of the RCPS presentation.

Usability

The application can be used without external instructions by an English speaking person with medium computer usage skills. As the application is meant to be used by potential customers of the RCPS, it has to be ensured that an average person with no special technical skills could use the application with minimal set of instructions shown in the application.

Reliability

The application works without any fatal errors 99% of the uptime to ensure the potential customers can use the application at any time.

Correctness

The application shows the correct movement of the machines on the screen 99% of execution times to ensure that potential customers get the same user experience.

Efficiency

The application must use less than 1 MB of network resources loading the application. This is to ensure that the application is loaded in reasonable time taking into account the average Internet speed of 5.6 Mbps of the world in Q4 2015 according to the Akamai's review [3].

Reusability

The application has to have the ability to be rewritten in minimal effort to visualize any upcoming robot car parking solutions with either implicit or explicit instruction set.

In implicit instruction set, the whole parking layout and every parking space's state is written out for every instruction step. It is necessary for the RCPS application to convert these instructions to explicit instructions.

Explicit instruction set has explicit orders for every machine on the lot for every movement step. For example, `"R0", "E"` would mean that robot with ID R0 should move one step east.

Portability

The client application has to be usable from any contemporary Linux, OS X and Windows operating system using Mozilla Firefox, Safari, Chrome or Microsoft Edge web browsers. In addition, the client application has to be usable from Android and iOS mobile devices using respective native web browsers. This requirement ensures that vast majority of the customers could use the application from either their PC or mobile device.

According to w3schools.com statistics, in April 2016 76,5% used Windows 7, 8 or 10, 5,5% Linux and 10.6% Mac OS X to access the w3schools.com website. Also, 92,8 % of users used web browsers mentioned in the requirement [4]. Requirements also cover 96,7% of the mobile devices on the market in Q2 2015 according to IDC Research [5].

The server application has to run on Linux and Windows operating systems with Python 3.5+, bottle and Jinja2 installed. This requirement ensures that the server application can be run on more than 99,9% of the web servers [6].

Extendibility

The application has to be written in such way that following simulation features could be added to the application with minimal changes on the source code:

- Clicking on a car to simulate the owner of the car requesting its retrieval,
- Clicking on a button that represents an outside car to simulate an owner arriving who wants to have his/her car parked,
- Transmitting the changes to the server and from the server to the compute server,
- Real-time retrieval of an updated robroute file which accommodates the user's demands.

3.2 Functional requirements

Following requirements describe the functionality of the application to be created

- Application is able to read the robroute file format.
- Application is able to construct parking lot layout and machines' instructions from a robroute instruction file.
- Application is showing static images of the start and end state of the simulation.
- Application is visualizing the simulation of the algorithm.
- Application is able to scale accordingly to parking lot size and viewport size.
- User is able to choose the scenario from the application to visualize.

In the following subchapters the requirements are explained in detail.

Application is able to read the robroute file format

Robroute file format is the outcome of the robot car park system algorithm. It consists largely of four that can be seen in code example 1:

1. The width and height of the parking lot
2. The layout of parking lot
3. Number of movement steps
4. Movement steps

```
# Robroute file -- part of the Robots project http://ac.cs.ut.ee
```

```
4 4
BHHGBPPOBPPOBNNM
100
cA|[cA|[cA|[cA|[eA|[cA|[cA|[cA|[cA|[dA|[cA|[cA|[cA|[cA|[cB|[fA|[
…
cA|[cA|[cA|[cA|[eA|[cA|[cA|[cA|[cA|[dA|[cA|[cA|[cA|[fB|[cA|[cA|[
```

Code example 1. Excerpt of a robroute file.

Width and height of the parking lot are two integers that describe the size of the parking lot.

The layout of the parking lot describes in which directions it is possible to move from the parking spaces.

Number of movement steps shows how many movement instructions will follow in the file.

Movement steps show the state of every parking space in given step. Every parking space has four state variables.

1. onNode state shows what type of car is in the space at this step or if the space is empty.
2. ndStat shows the robot's state. There can either be no robot or the robot can be ready, moving or lifting/dropping a car. Also, on the state it is shown either the robot is there with a type of car or alone.
3. rVertical state describes the vertical movement of the robot. It can either lift the car (there are 5 different levels of lifting), drop the car or there might be no vertical movement.
4. rMove state describes the horizontal movement of the robot. It describes the movement in all four possible directions (North, South, East or West), if the robot is moving with or without the car and is it accelerating or already moving.

To make robroute files smaller in size, the states of parking spaces are encoded into characters as can be seen from code example 1. The states have to be decoded by the application.

Application is able to construct parking lot and machines' instructions from a robroute instruction file

As mentioned before, the robroute file includes the width and height of the parking lot and the layout of parking lot. The parking spaces have to be given coordinates in a grid (x and y coordinates). In addition, the different types of parking spaces have to be considered as there might be cases where the parking lot is not perfect rectangle without any obstructions in the middle.

The instructions in robroute file are generated by C++ algorithm which uses bitwise addition to characters for differentiating the states. The states need to be converted back from characters to ensure meaningful state enumeration.

Furthermore, the instructions are not explicit as in they do not give a unique ID to the machine. For visualization purposes the instructions have to be converted to explicit instructions.

Application is showing static images of start and end state of the simulation

In order to grasp the simulation fully by the user, start and end state of the simulation has to be seen on the screen.

Application is visualizing the simulation of the algorithm

User has to be able to see all of the steps of algorithm simulation on screen. The simulation has to be visualized continuously – it cannot have step-by-step static images.

Application is able to scale accordingly to parking lot and viewport size

As the application can be used from variety of devices with different screen resolutions and the parking lot sizes can vary vastly, the application has to be able to scale the parking lot accordingly.

User is able to choose the scenario to visualize from application

There are different scenarios that can be visualized. User has to be able to choose between them before the simulation. User has to be able to go back to choosing the scenario at any time during the visualization of a scenario.

4 System architecture

In this chapter, the architecture of the system and the selection of the development technologies are described. In addition, the selection of machine movement approach is described.

Technologies used are state-of-art and work for the best result. Web application approach was chosen as in recent years, the browser support for HTML5 technology stack standards has improved drastically, which makes web application developed for HTML5 truly crossbrowser and cross-device experience. The other advantage is the ability to access the application from any location.

Originally, Phaser was chosen as a JavaScript library supporting the visualization part of the development as it has the right support for the requirements of the application. This includes scalability in screen sizes and JSON support. JSON is used as data object transmitting standard as it is quicker to parse and transmit, but also because of its integration in JavaScript language. In the early stages of development process, where the architecture of the application came more apparent, Phaser was ditched from the technology stack as it did not support the core methodologies used in the application.

Python is used as a server side language for its easy to read syntax and possibility to implement new features quickly. Bottle is used as a web framework for its lightweightness and simplicity. It has the ability for function-call mapping for clean and dynamic URLs.

4.1 Server side platform selection

Nowadays server side programming can be made by many different languages that have different approaches on the web server design patterns. These language include Java, ASP.NET, Python, PHP, Ruby on Rails et cetera. For this project four languages were taken under review that could fill the requirements set by the client side application. These languages are Python, PHP, Java and ASP.NET, which builds on C#.

Java is a general-purpose programming languages. The approach in Java is to let developers “write once, run anywhere” [7]. Java is used in a huge variety of use cases, including desktop applications, mobile applications and web applications, client and server side.

PHP is a server-side scripting language which is designed for web development. It can also be used as a general-purpose programming language. PHP is still one of the popular choices of web developers, known for huge community, big variety of frameworks and good documentation. ASP.NET is Microsoft’s open-source web application framework designed to produce dynamic Web pages.

ASP.NET uses C# programming language. A comparative study [8] compares the three aforementioned web technologies using MVC (Model-View-Controller) design pattern on all of the cases to build a web application. In Java that means using JSP web pages (*.jsp) as views, the controllers are developed using Java servlets and the models are developed using Enterprise Java Beans and Java Persistence API. In ASP.NET, views were developed using Active Server Pages. The controllers are in C# code and views can be developed by either razor or aspx. As PHP does not have its own MVC design pattern, PHP system was implemented using CakePHP framework.

The study concluded that Java processes Login HTTP requests faster than two other systems. The study also shows that Java is the quickest to process GET method requests, but ASP.NET is the quickest in processing POST method requests. All in all, Java and PHP were found to be cheaper to implement compared to ASP.NET. The downside of Java is the difficulty of using it compared to CakePHP and ASP.NET.

Python is a general-purpose dynamic programming language first appeared on 20 February 1991. Amongst others, these are the guiding principles of Python's design [9]:

- "Beautiful is better than ugly",
- "Explicit is better than implicit",
- "Simple is better than complex".

For web serving purposes, Python uses Python Web Server Gateway Interface (WSGI) as a standard interface between web servers and Python web applications or frameworks [10]. In addition of Python having clean, simple syntax, it also has a wide variety of WSGI frameworks to choose from.

Taking into account the findings in the study, Python is used in the development of server side application for robot car parking system visualization. The reasoning is the advantages over other mentioned programming languages – cleaner syntax without much broilerplate (compared to ASP.NET and Java) and clearer function naming (compared to PHP).

Python web framework selection

Choosing the right web framework for the application can be troublesome as there is vast number of different frameworks available. As the application does not need all the functionality that full-stack frameworks provide, the lightweight frameworks will suit the best. Three of most popular ones are compared: Bottle, CherryPy and Flask.

Bottle

Bottle is a lightweight Python micro web framework. It was first released on July 1 2009 by Marcel Hellkamp. It is easy to use, has built-in template engine, support for JSON client data and can be extended with different plugins [11]. Furthermore, Bottle has a built-in web development server and supports any WSGI capable HTTP server. Built-in web server makes it easy to develop and test the code.

CherryPy

CherryPy is an object-oriented web framework for Python. It has a flexible plugin system, powerful configuration system, reliable WSGI thread-pooled webserver amongst other features. CherryPy has been available over ten years and is open-source project. [12]

Flask

Flask is a micro framework for Python that is based on Wekzeug and Jinja 2. Flask's intention is to keep the framework core simple and at the same time extensible, where an extension can be added to the project when it really is needed. This makes Flask minimal and keeps points of failures low. [13]

All of the three aforementioned web frameworks provide similar functionality keeping the application simple without adding any unnecessary overhead. Bottle framework was chosen for building the application. The reasoning was the subjective ease of starting and clearness of documentation by author.

4.2 Data object transmitting standards

The traffic between client and server in the discussed web application in this thesis will, in majority of times, be by asynchronous calls. AJAX [14] standard will be used to accomplish that. Under review are two most popular data object transmitting standards, XML and JSON.

XML (The Extensible Markup Language) is considered the 'holy grail' of computing for its universal data representation format. The priorities when designing the languages were simplicity and human readability. It is primarily used for Remote Procedure Calls. XML does not have any pre-defined tag sets – everything can be configured by user. An example of an object in XML is provided in code example 2.

```
<person>
   <firstname>Suido</firstname>
   <lastname>Valli</lastname>
</person>
```

Code example 2. Example of XML object.

JSON is designed to be human readable and easily parsed and used by computers. JSON is directly supported inside JavaScript. It is estimated to parse up to hundred times faster than XML. The disadvantages over XML include lack of namespace support, lack of input validation and extensibility drawbacks. An example of an object in JSON is provided in code example 3.

```
{
"firstname": "Suido",
"lastname": "Valli"
}
```

Code example 3. Example of JSON object.

From the comparative case study [15] it can be concluded that sending data in JSON encoding is in general faster than sending it in XML encoding. On the other hand, the transmission times of XML are lower when fever objects are transmitted. As for the fact that sending and parsing JSON data is generally faster than doing the same with XML data and that JSON is natively supported in JavaScript, JSON will be used in the thesis project for transmitting data objects between client application and server.

4.3 Client side platform selection

There are quite a lot web technologies in use to make a graphic animation. Three of the biggest are Flash, Java and combination of HTML5, CSS3 and JavaScript.

Adobe Flash, formerly known as Macromedia Flash and Shockwave Flash, is a platform for creating rich Internet applications, usually used for development of Web-based games and interactive tools. Flash Player plugin is required for Flash content to work as it runs as a client-side sandboxed virtual machine [16]. Downside of Flash is the fact that it is proprietary technology.

Java is a powerful development platform. In Web browsers, Java runs in sandboxed virtual machine. For that fact, Java applets are theoretically platform-independent. The problem with Java is its need for the plugin and the fact that different applications may need different versions of the plugin. Furthermore, the initialization of the applet can be long. Also, there are serious security issues, due to which Java applets are not executed automatically in Web browsers anymore [17].

HTML5, CSS3 and JavaScript provide open source alternative to Java and Flash. HTML5 is used to provide static content, CSS3 to style the said content and JavaScript is used to make the content dynamic. As HTML5 is supported by all major browsers without needing a plugin, the users do not have to be worried about security risks. One potential disadvantage of HTML5 would be the different interpretation of the standards by different browsers, but these can be overcome with some JavaScript libraries, for example jQuery. The main

disadvantage of HTML5 is the relative ease to see the source code. As this thesis is in public domain, this fact is not accounted as a disadvantage when choosing the web technologies [18].

As the main reason the web application approach for visualization was chosen was the application to run seamlessly on plethora of devices, Java and Flash are ruled out mostly because of their plugin requirement, but also for the fact that they are obsolete technologies and, as use of them decreases steadily, they might not be supported soon. The technology used for making the visualization application is HTML5 technology stack: HTML5, JavaScript and CSS3.

JavaScript Libraries

JavaScript as a language is very extensive and for now, the language that web programmers use. For visualization, one could start from scratch and invent the logic behind the animation him/herself. As there are plenty of different graphics frameworks written on top of JavaScript, it is neccessary to consider them as an option instead of building the visualization directly on HTML5 native canvas element.

Ten different options of WebGL Graphics Engine from survey [19] will be studied and best suited for the application's purpose is chosen.

CAAT (Canvas Advanced Animation Toolkit) is a 2D director based scene graph renderer. CAAT has a chance to have multiple instances as you can have unlimited number of directors for each web page. It is very diverse toolkit with unlimited number of timelines and actors. CAAT has an abstracted input system and it does resource management and preloading. Disadvantages of CAAT is that it was last updated over two years ago and it does not have an active community behind it.

CakeJS is a scene graph library written in JavaScript meant for HTML5 canvas element. It has animation timelines and it supports mouse events. It does not support easy resizing of application in different screen sizes and aspect ratios. Furthermore, this project has been archived.

Canvas Engine is a library for easily creating games in HTML5 Canvas. It works on modern browsers and also smartphones. The scenes can be structured and preloaded. In addition, Canvas Engine has a model for handling server-side events to develop a multiplayer game. Although Canvas Engine is meant for building games, it is also suitable for the purpose of robot car parking system visualization and simulation. The main disadvantage is the smallness of community and the fact that Canvas Engine has not been updated for over a year.

ChesterGL (Chester Game Library) focuses on ease of use and performance. It supports a simple scene graph. It has time based actions, different shaders (for WebGL) and batched sprites. The disadvantage of ChesterGL is smallness of community and infrequent updates, lastly updated 2 years ago.

Cocos2d-JS is a game engine for web and native games. It has high performance, has modern JavaScript API, full platform without needing plugins. Furthermore, there is a capability to test and debug the developed application on the browsers before pushing them to target device. It has vast API including transitions, events, persistence et cetera. Cocos2d-JS is a good candidate for the JavaScript library to be used in the thesis practical work.

Construct 2 is designed for creating 2D applications rapidly. It is very high level framework. Construct 2 is not open source nor free as majority of other libraries discussed in this thesis.

The major difference of Construct 2 is that it does not require any coding – the primary method of programming is through the event sheets. As Construct 2 is not free and there can be issues interfacing with server that cannot be issued inside the event sheets environment, it is not suitable for the purpose of the application developed in thesis.

Crafty is a flexible 2D framework for JavaScript games. It can either use Canvas or DOM, it has sprite map support and collision detection. It is also lightweight framework and open source. Furthermore, it is actively developed and has active community. All in all, Crafty is a considerable candidate, taking into account the requirements of the application developed.

EaselJS is a 2D graphics engine which supports objects nesting and has mouse interaction model. It uses familiar approach for developers which should make it easy to work with. EaselJS is frequently used in making HTML advertisements. ImpactJS is a 2D and isometric HTML5 graphics engine. Tit supports real time multi user application and implements a scene graph based architecture. It has a built in physics module.

ImpactJS is a considerable framework to use for the application, as it allows creating in addition to two-dimensional scenes isometric scenes, which can be beneficial for understandability of algorithm workings.

Phaser is a free 2D game framework that supports Canvas and WebGL rendering. It uses Pixi.js internally for rendering. Phaser can automatically switch between Canvas and WebGL rendering, according to the available technologies in the device. Phaser can use JSON and XML for asynchronous calls, it supports inputs from mouse, touch screen, keyboard. Furthermore, with Phaser has built-in Scale Manager which allows developer to scale the application to fit any size screen. Phaser is actively developed and has active community.

Although there are plenty of good candidates in the 10 reviewed libraries, the author chose Phaser as it has the most useful features needed for the development of parking system visualization built in. For example, it can use JSON or XML data without any further libraries or development on programmer part.

In the early stages of development it came apparent that the chosen framework is not the best choice for developing an application specified by requirements mentioned in chapter 3.2. Phaser is meant to develop games and as such there is no easy way to give instructions where an object has to be on specified frame. After considering alternatives, author ditched the additional layer of 2D framework and used pure JS and jQuery to construct a working visualization on HTML5 canvas element.

4.4 Machine movement approach

The presentation of machine movement is a crucial part of this application. Taking into account the nature of the task and the requirements of the application, 2D approach was chosen.

The parking lot is presented to the user from top-down perspective. This lets the users grasp the parking lot as a whole and arguably the movement of the cars can be presented concisely in this perspective.

Presenting horizontal movement of machines is straight-forward using the top-down perspective – the machine is moving up, down, left or right on the screen depending on the instructions (N, S, W, E).

Presenting vertical movement of machines can be more challenging. The solution is to combine following techniques:

- Scale the machine size up by a degree when lifting
- Move the machine diagonally by a degree when lifting
- Cast a shadow of the machines – distance of the shadow from the machine depends on the height of the machine from the ground

In addition to the chosen perspective and techniques, 3D and isometric presentation were also considered. The 3D and isometric approaches were dismissed because of the little added value for the effort. In addition, 3D visualization requires more system and networking resources than 2D visualization, which would narrow the range of devices that could run the visualization.

The approach is validated in chapter 6 of the thesis.

5 Application overview

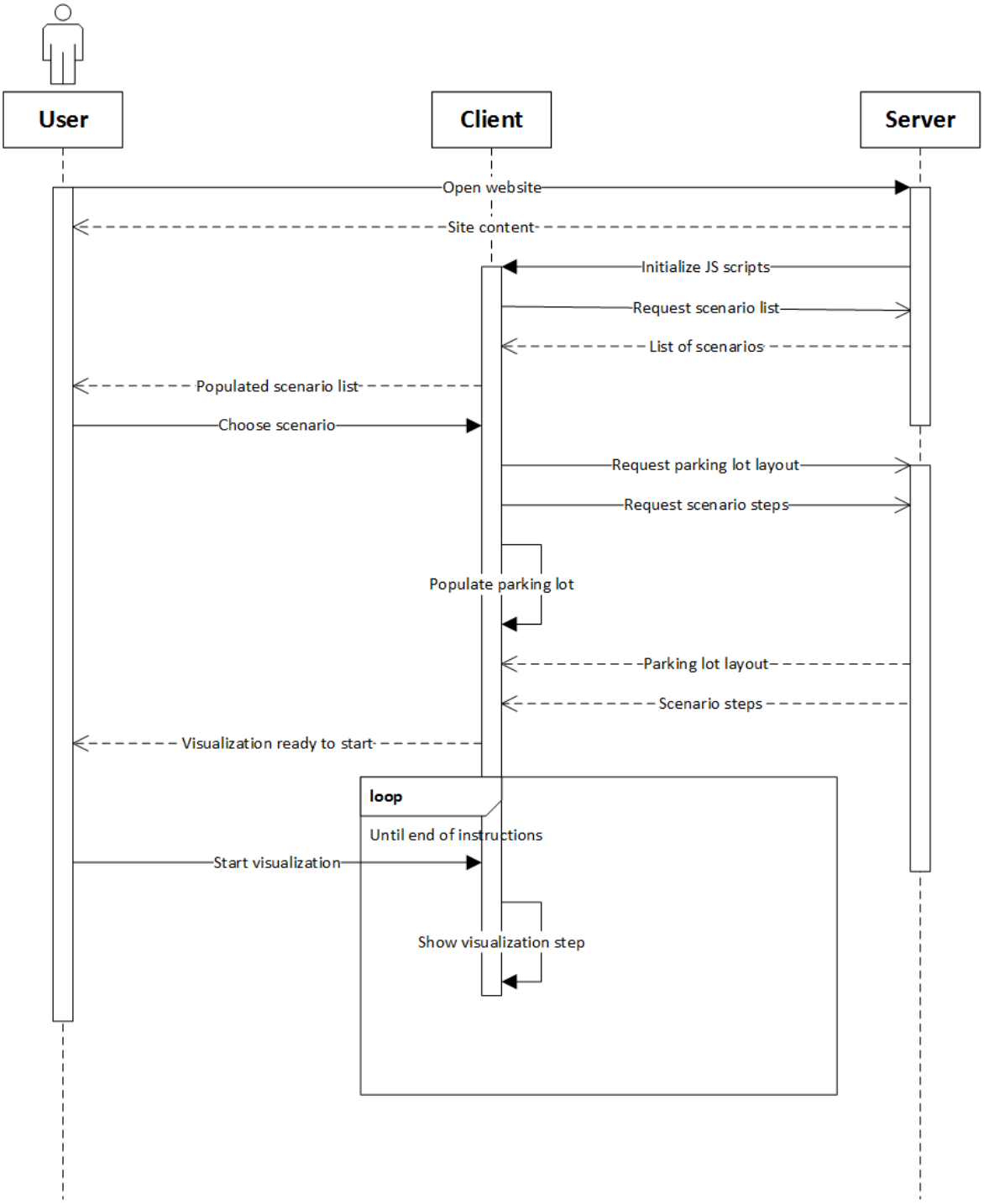


Figure 1. Top level sequence diagram of the application

From high level standpoint, the application consists of two separate parts - the client application and server application. Client application is built using HTML5, vanilla JavaScript and jQuery. Server is built using Python 3.x and uses Bottle web framework with added Jinja2 templating library for building HTML page and serving the web content.

As seen from Figure 1 which describes the top level program sequence, after user opens the application website, the static page content is returned and shortly after, the list of scenarios

is requested from server by client. When client has chosen the scenario, the layout and instructions are requested from server using AJAX calls to RESTful web service. Once the server has sent the responses for aforementioned calls, visualization page is rendered and ready to visualize the scenario.

In the following subchapters, the detailed flow of the application and application functions are described.

5.1 Application flow

Rendering the index page

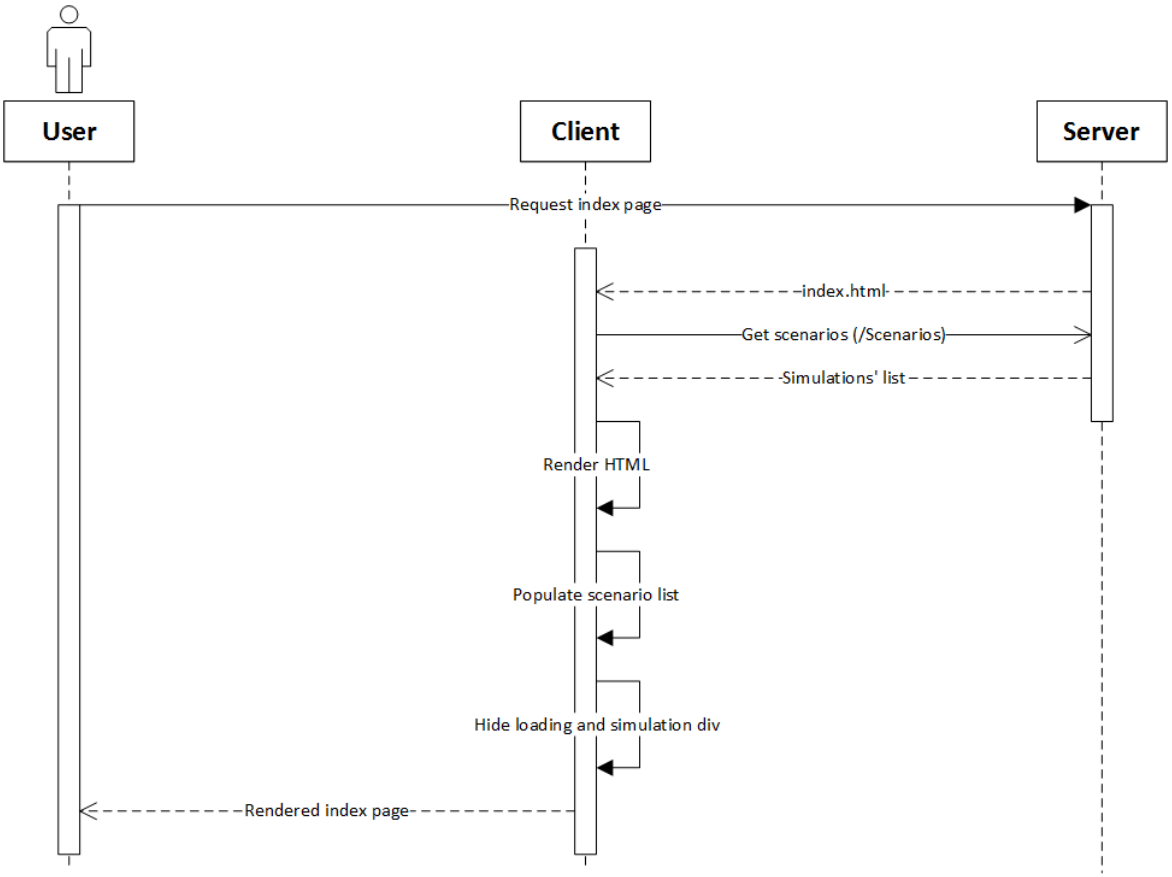


Figure 2. Index page rendering sequence diagram.

Application starts by user going to the index page of application web page. For development process, it is http://localhost:8080. Upon arrival to the page, the web server will return the index page template. Next, AJAX call is made to `/Scenarios` page of server to retrieve the available parking scenarios. The scenarios are held in `./Examples` folder on server. When server gets a request on `/Scenarios` address, `get_json_route_list()` function is called which finds all the files with `.robroute` extension in Example folder and returns a JSON array with the names and values of the available scenarios.

Choosing the parking lot and instructions

When the available scenarios have been retrieved, user can choose scenario of what he or she wants to see the visualization of. Also, user has to specify if realistic visualization or visualization with the ability to choose speed is needed. When a scenario gets chosen, the client makes three AJAX calls to server. First AJAX call is made to URL `/Scenarios/<scenarioNr>/Layout/0` to get the end state of the visualization and shortly after

that `/Scenarios<scenarioNr>/Layout/1` to get the starting state. The need for two calls is to show the user static images of the parking lot at the beginning and end of visualization at all times. Third AJAX call is made to `/Scenarios/scenarioNr/Instructions/Realistic` to get the instruction steps for the parking lot. Instead of keyword `Realistic` 0 or 1 is used – when user requested realistic visualization, the keyword is 1, otherwise it is 0.

Retrieving parking lot layout.

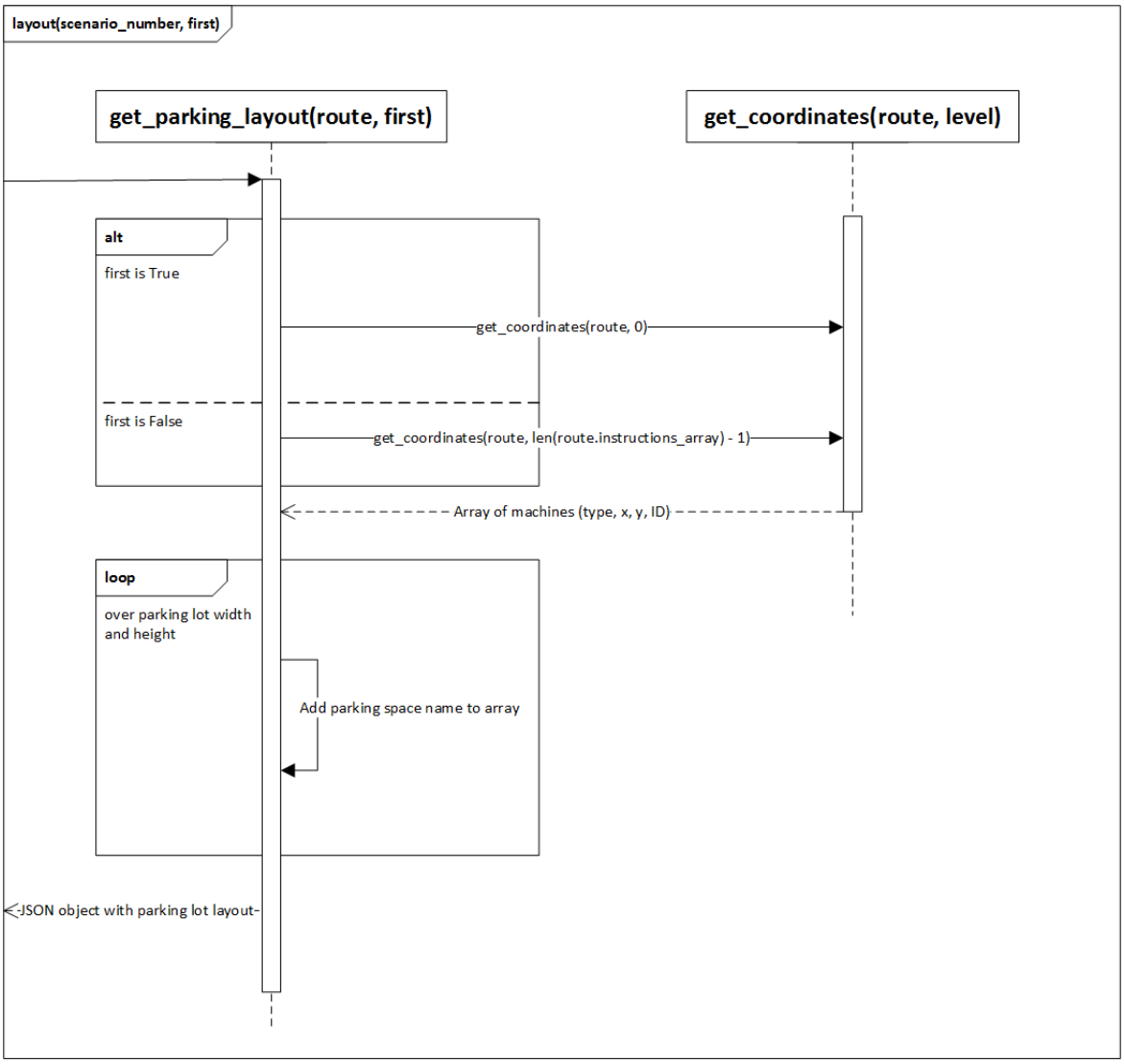


Figure 3. Sequence diagram of server side coordinates retrieval

Server, upon getting a request to return parking lot layout, calls `get_parking_layout(route)` function. Route is an object that is returned by function `get_route(filename)`.

`get_route(filename)` reads the file with the name given in function argument and returns Route class object. Route class object consists of three parts:

- `lot_size`, which is an array with two values – number of parking spaces in parking lot vertically and horizontally,
- `lot_layout`, which is a two-dimensional array that holds the parking space's state (to which directions is it able to move from this parking space),
- `instructions_array`, which is an array of instruction steps that hold the four different states for each parking space in each array.

`get_parking_layout(route)` returns a JSON object describing the width and height of the parking lot concerning the parking spaces, layout with parking spaces states and machine array that describes the position of the machines in the grid, their ID and their sprite type. Detailed diagram of the function can be seen on Figure 3.

Machine array is retrieved from `get_coordinates(route, level, making_instructions = False)` function which iterates over the scenario's instructions at the given level. When it finds a machine on any parking space, it will add its type (either car or robot), coordinates, unique ID and sprite image type to the array that is going to be returned in the end of the iteration. Example of server's response to layout request can be seen on Code example 4.

```
{
   "layout": [
      ["wallNW", "wallN", "wallN", "wallNE"],
      ["wallW", "nowall", "nowall", "wallE"],
      ["wallW", "nowall", "nowall", "wallE"],
      ["wallSW", "wallS", "wallS", "wallES"]
   ],
   "machines": [
      ["C0", 0, 1, "C0"],
      ["R", 1, 2, "R0"]
   ],
   "width": 10,
   "height": 4
}
```

Code example 4. Example of layout JSON response

Retrieving instructions

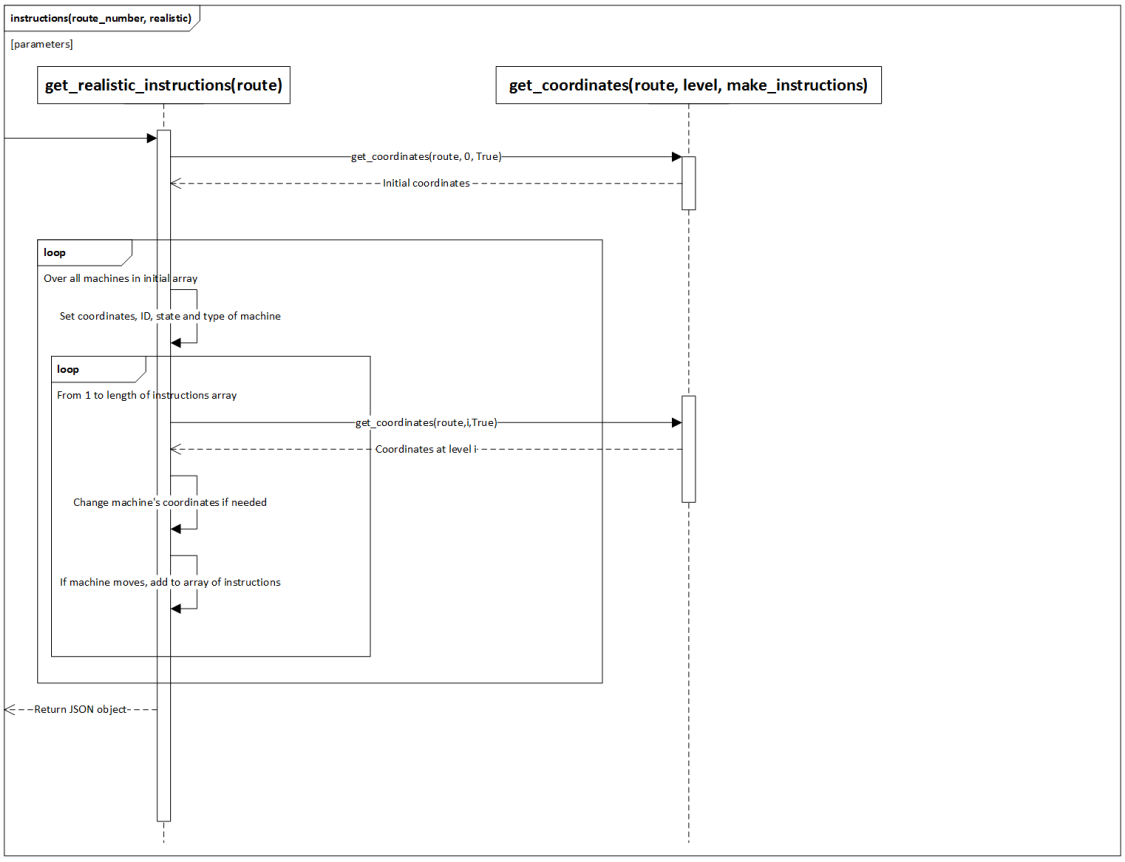


Figure 4. Sequence diagram of server side instructions retrieval

Function `get_instructions(route)` is called upon server getting a request to `/Route/routeNr/Instructions`. The function will take initial machine array as a starting point by calling function `get_coordinates(route ,0, True)`. Adding `True` as third argument will add the four states to every machine's array. Next, the function loops over every machine in the machine array and goes over all the instruction steps. When looping over the instructions, it keeps record of machine's ID and coordinates and will change the machine's coordinates accordingly to the states. If machine is moving at some level, the machine ID, instruction direction and speed is added to the instruction array.

The function returns the array of explicit instructions needed by the client to move the machines accordingly as a JSON object. The example of said JSON object can be seen at Code example 5.

```
[
   [
      ["R0", "W", 2],
      ["R1", "S", 4]
   ],
   [
      ["R0", "W", 1],
      ["R1", "S", 2]
   ],
   [
      ["R0", "W", 1],
      ["R1", "S", 4]
   ],
   [
      ["R0", "W", 1]
   ],
   [
      ["R0", "W", 1]
   ]
]
```

Code example 5. Example of instructions JSON response.

Rendering the main workspace

After client application receives the instructions and parking layout JSON objects, it does the following

1. Renders the final state of scenario on canvas, makes it an image and adds it to the menu,
2. Renders the start state of scenario on canvas, makes it an image and adds it to the menu,
3. Hides loading div, shows visualization workspace,
4. Starts visualization loop.

The application only requests the types of parking lot images from the server that are needed for the scenario. Retrieved images are cached in an array to keep network traffic low. Furthermore, the parking space is rendered in two or three layers, depending on if the user wants to see the grid on the parking lot or not. This solution allows to change the underlying asphalt or concrete pattern with minimal effort – only one image needs to be changed.

Visualizing the scenario

After the rendering of the scenario, user has two options: either to start automatic visualization or step-by-step visualization. Automatic visualization can be changed to step-by-step visualization and vice versa during the scenario. The visualization works on frame basis. That means that each frame `simulationLoop()` function is called. Said function controls all

of the visualization. Firstly, it increments variable STEPS by one if there is any movement on the screen. After that, it will clear the canvas element from everything and calculates the instruction array index from the STEPS variable. If the function discovers that it is time for new instructions, it will firstly stop all the movement on the screen by calling `stopAllMovement()` function. Next it will call `moves(routeInstructions, step)` function that gives new instruction to all the machines which will move at next instruction step. After that, parking lot is rendered by calling `createParkingLayout()` function, then robots are rendered calling `renderMachines(robots)` and lastly, cars with `renderMachines(cars)`. Last but not least, frames per second are calculated in the `simulationLoop()` function. As `simulationLoop()` calls `window.requestAnimationFrame(simulationLoop)` function as the first thing, the next time that browser is ready to get the next frame, `simulationLoop()` is called again.

Function moves()

Function `moves(instructions, stepNr)` takes the array of instructions and the step of instructions as arguments. Function checks if the new step is in the bounds of instructions array. If it is, it will iterate over all of the instructions in that step and calls `moveMachine(machine, instruction, speed)` function which gives the machine object direction on what to do next. This function is described in details in the next chapter. `moveMachine(…)` function also changes the text shown to the user describing what machines are moving where at what speed in that step.

Function moveMachine()

Function `moveMachine(machine, instruction, speed)` gives instructions to the machine object. It goes through the different instruction cases and updates the x and y coordinate direction of the machine, the speed and calls the machine's update function. The sprite class, that all machines have, will be covered in detail in next subchapter.

Class sprite()

Class sprite(options) is the class for all machines. It holds all the settings of the machine, most important of them are the width, height, image, x coordinate, y coordinate and moving, lifting flags of the machine. The class also has methods `render()`, `update(dirX, dirY)` and `addPixel(dirX, dirY, updateshadow = true)`. Method `render()` renders the machine itself and its shadow. Method update will increment the machine's `speedCount` variable and will call `addPixel()` method when the machine's direction, speed and `speedCount` variables match the set conditions are matching. For north, east, west and south movement the machine will be moved certain amount of pixels away from its current location. When machine is lifting or dropping, the machine is, in addition to coordinates movement, scaled up or down, depending on the vertical movement.

5.2 Application functions

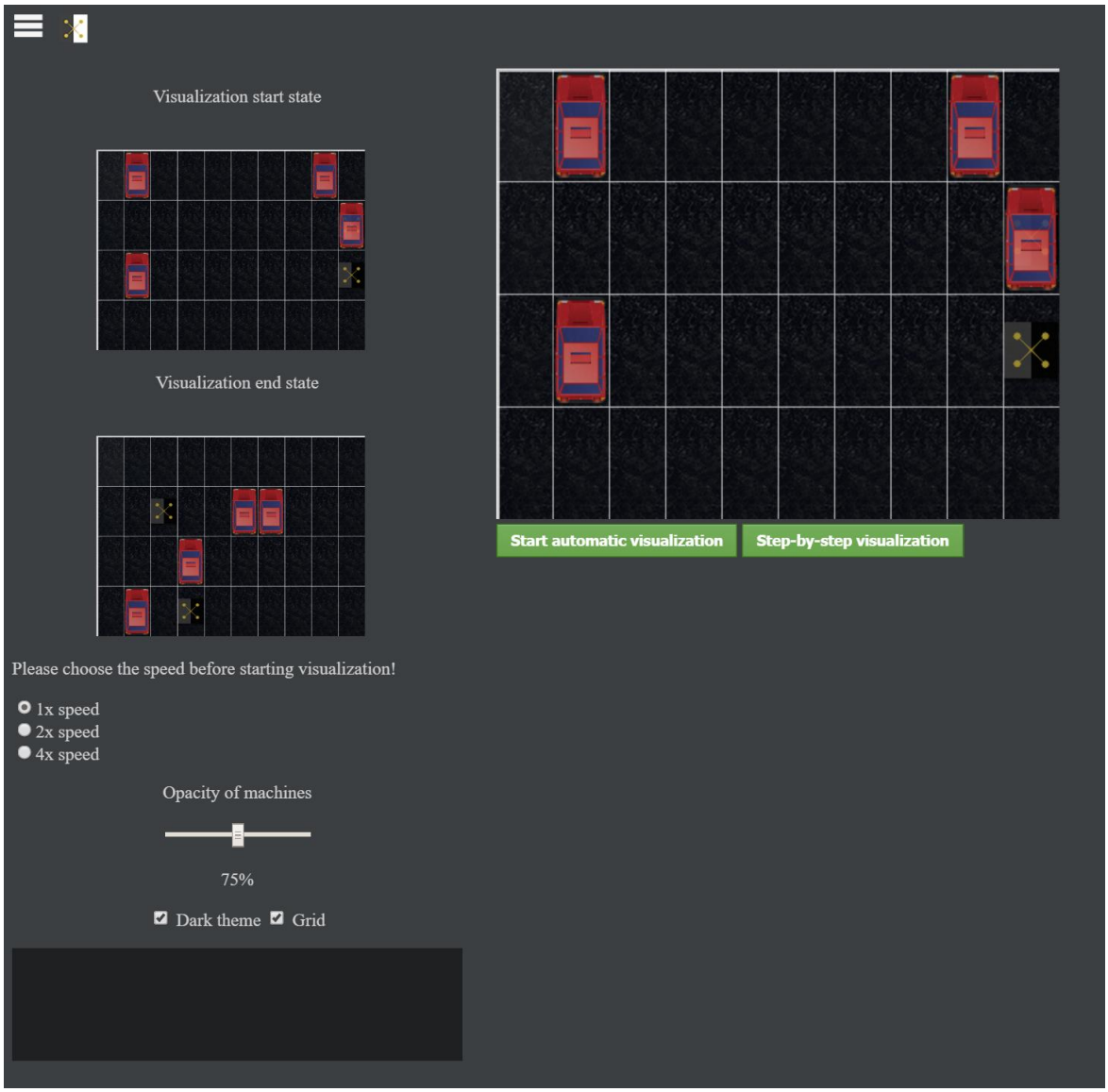
In this subchapter, the functions of the application are described. The functionality is divided into two parts – the index page functionality and visualization page functionality.

Index page functionality

Screenshot 1. Index page of the application

On the index page, user can choose between different visualization scenarios that are present in the server. User also has the opportunity to choose if the visualization is going to be realistic or if there is option to change the speed of the visualization. Realistic mode means that machines in the visualization are accelerating and decelerating as cars and robots do in real life.

Visualization page functionality



Screenshot 2. Visualization page with settings menu expanded.

On the visualization page, user has chance to change several settings of the visualization. Settings can be seen by clicking on the hamburger button in the top left corner of the page. By clicking on the robot button on its right, user is taken back to the index page where he or she can choose a new scenario. The opacity of the machines can be changed from 50% to 100% by moving the slider to left or right. The current opacity is shown under the slider. Default opacity is 75%. The purpose of this functionality is for the user to see the robots under the cars, if he or she feels the neccessity for that.

The theme of the page can be changed from dark to light by toggling the theme checkbox. Default value is light theme. This changes the background of the page, the parking lot image and the text color according to the theme settings. New themes can be easily made for the program doing minimal changes in the source code.

Furthermore, users have option to turn the grid on the parking lot on or off. By default, grid is turned on as it makes following the visualization easier.

When the user had selected the option to choose the speeds of the simulation, there is option to choose if the visualization is going to be made at original speed or two or four times the original speed. Once the visualization has started, it is not possible to choose the speed anymore.

Also, visualization start and end states are shown on the side of the vizaulisation canvas for better understanding what the algorithm has to do.

When visualization is running, the information about the movement on the current step is shown. The information shown is the ID of the machine, it's movement direction (north, west, east, south, lifting or dropping) and the speed or level of the machine. The speed of the machine shows how many steps it is needed for the machine to get from one space to the adjacent space. In case of vertical movement such as lifting and dropping, it shows the level of progress the machine is at. For example, „Machine R0 makes step L 1" means that Robot with ID „R0" is lifting and is on level 1.

Once the visualization has started, the frames per second are shown. The purpose of this is to indicate wether the machine running the visualization is up to the task – if the FPS drops under 24, user experience can suffer. The lower limits of the machine are determined in validation chapter.

To start automatic visualization, user has two options – either to click on the „Start automatic visualization" button or to click on any car on the parking lot. On user input, the visualization starts. The automatic visualization can be changed to step-by-step visualization by clicking on relevant button at any time during the visualization.

Starting step-by-step visualization is done by the „Step-by-step" visualization button. The button will be disabled, colored red and the text on the button is changed to „Active movement" until the step is finished. After the step is finished, it turns back to active button with text „Proceed to next step". Now user can watch the next step whenever it is convenient for him or her.

6 Validation

In this chapter, the application's working on different platforms, its efficiency and integrity are validated. As the application consists of client and server parts, then two of them are tested separately in all of the mentioned categories. In addition, the choices of presenting the machines' movement are validated by a survey.

6.1 Sanity testing

All application's core functions were tested on all the platforms and browsers specified in the requirements to ensure that application is working properly regardless of the operating system and browser user's device uses. In addition, the performance of the client application was verified by capturing the average framerate of the visualization.

Platforms on which server side of application was tested are as follows: Windows 10, Linux Mint 17.3 "Rosa" and Ubuntu 14.04.

Windows 10 was running on a physical PC with following specifications:

- Intel core i5-3570K @ 3.4 GHz
- 12 GB DDR3 RAM
- nVidia GeForce GTX 660 Ti
- 120 GB Samsung SSD 840 series

Linux Mint 17.3 "Rosa" was installed on a virtual machine with 4 GB of RAM and 128 MB 3D hardware graphics acceleration on the physical PC mentioned.

Ubuntu 14.04 is running as a virtual machine in Nitrous.io platform, which is a development environment in the cloud.

Client side web browsers used were as follows:

- Linux Mint
  - Firefox 42.0
- Windows 10
  - Google Chrome 50.0.2661.102 m
  - Firefox 46.0.1
  - Microsoft Edge 25.10586.0.0
- Mac OS X 10.11.4 (CPU – Intel core i7 @ 2.2 GHz, 16 GB RAM)
  - Safari
- Android (OnePlus 2, Android version 6.0.1)
  - Google Chrome 50.0.2661.89
- iOS (iPhone 4S, iOS 9.2)
  - Safari

By combining the server platforms with client platforms and web browsers, following test scenarios were created:

Table 1. Test scenarios for application

Test scenario number	**Server platform**	**Client operating system**	**Browser**
1	Linux Mint	Linux Mint	Firefox
2	Windows 10	Windows 10	Google Chrome

3	Windows 10	Windows 10	Firefox
4	Windows 10	Windows 10	Microsoft Edge
5	Ubuntu	Linux Mint	Firefox
6	Ubuntu	Windows 10	Chrome
7	Ubuntu	Android	Chrome
8	Ubuntu	iOS	Safari
9	Ubuntu	OS X	Safari

During the testing of scenarios, all of the functionality was tested and the behaviour of the application asserted. Furthermore, the average FPS was captured.

All the tests were successful and while the appearance of some HTML elements changed depending on the browser, everything worked as specified. In the beginning of testing, it came apparent that Safari and Microsoft Edge are not supporting default parameters in function arguments as described in ES6/ES2015[6]. Mentioned browsers raised an exception when parsing following code:

```
that.addPixel = function (dirX, dirY, updateShadow = true) {
   …
}
```

Code example 6. Parameter default value

To mitigate the problem, the functions with default parameter values had to be changed as follows:

```
that.addPixel = function (dirX, dirY, updateShadow) {
   updateShadow = typeof updateShadow !== 'undefined' ? updateShadow : true;
   …
}
```

Code example 7. Alternative to parameter default value.

The average FPS of the test scenarios can be found in table 2. The FPS in modern browsers is usually capped at 60 frames per second.

Table 2. Average FPS on visualization in test scenarios.

Test scenario number	**Average FPS**
1	58
2	60
3	60
4	60

[6] http://wiki.ecmascript.org/doku.php?id=harmony:parameter_default_values ECMAScript parameter default values

5	58
6	60
7	58
8	32
9	60

As can be seen from table 2, almost all the browsers were able to run the visualization with 60 FPS, where browsers limit the framerate. The only significant change in the framerate can be seen in test scenario 8 where iPhone 4S was used as a test device. Considering that the device was released in 2011, has dual-core 1 GHz processor and 512 MB RAM, this result was expected. In addition, 32 frames per second on average will not make a drastic difference in the user experience considering the nature of the application.

6.2 Efficiency of application

The efficiency of client application is partly covered in previous chapter showing the average framerate of the visualization. The application's network efficiency and CPU load are tested in this chapter. For server, the average time for serving JSON is tested.

Client side efficiency

For client side efficiency, Google Chrome developer tools are used to check the CPU load and network efficiency.

Network efficiency is measured by recording network log with Chrome developer tools. The recording is started when the index page of application is opened and ended after full visualization. In addition, the theme and grid settings are toggled during the time to ensure maximum network load. The results are as follows: 34 requests were made and 362 KB of data was transferred from server to the client. The content was loaded in 1.46 seconds. Those values are in limits with the requirements from chapter 3.

Measuring the load to the processor uses the same test scenario as network efficiency test. The test output has been taken from Google Chrome's task manager. Application's browser tab uses 34,468 MB of memory and when the visualization is ongoing, it takes 2-4% of the CPU time. Considering that most contemporary computers have at least 4 GB of RAM and smartphones 2 GB, the program is efficient in using the device's resources.

Server side efficiency is measured by finding the average time to get scenario list, layout and instructions as a response to requests. The percentage of CPU utilization by the server process is also captured. All of the test scenarios made 100 requests to the server. The test scenarios and results can be found in table 3.

Table 3. Server efficiency test results.

Test scenario number	Request	Average time in milliseconds	Responses under 250/300/350 ms	CPU utilization % of the process
1	Get list of scenarios	151 ms	77/97/100	0.2 %
2	Get ending state layout for first scenario	178 ms	62/86/91	9.8 %
3	Get starting state layout for first scenario	177 ms	67/90/93	10.5 %
4	Get realistic movement instructions for first scenario	236 ms	66/89/93	23.7 %
5	Get normal movement instructions for first scenario	235 ms	61/88/92	23.1 %

As seen from the test results, getting the list of scenarios took on average 151 milliseconds and 97% of requests were done under 250 milliseconds. Server used 0.2% of CPU time for this request on average.

There is minimal difference in requesting the starting and ending state layouts of the scenarios, with the average response time being 177 milliseconds, 86% of responses being under 250 milliseconds. The server was using roughly 10% of CPU time dealing with the requests.

Requesting the movement instructions took on average 235 milliseconds to respond with 88% of responses being under 250 milliseconds. Application utilized on average 24% of CPU while processing the requests.

Finally, all of the test scenarios were run sequentially 1000 times. The average response time was 231 milliseconds and the CPU utilization 11%.

6.3 Integrity

The integrity of the application is tested by using unit tests on server and visual confirmation on client side.

The unit tests on server side cover functions `get_parking_layout()`, `get_instructions()`, `get_realistic_instructions()` and `get_json_route_list()`. Unit tests are situated in tests.py file. Unit tests control if the functions return the proper Python type object and control if the length of different arrays and lists are what they should be. In addition, more detailed positive and negative test cases are performed to see that the application returns the

same correct object every time. All of the performed tests passed, so it can be said that the server part of the application is integral. Test coverage on fileReader.py is 96% when not taking into account non-essential helper functions that are never used by the client application. These functions are for converting computer generated scenario file's instructions to more human readable instructions for checking the compliance of the visualization to the original instructions.

Testing the client side of application was done as follows: Scenario 3 with realistic visualization was chosen and 10 steps were done with step-by-step visualization in Google Chrome, Mozilla Firefox and Microsoft Edge. The results were compared side by side.

Screenshot 3. Superimposed image of canvas elements from three browsers.

As browsers render default HTML elements differently, there was some differences between the pages on different browsers. Regardless, all the functionality worked as expected. Furthermore, on screenshot 3 there is superimposed image of the canvas element of the three outcomes of the test. As can be seen, the match is pixel perfect.

6.4 Validation of presentation choices

Validation of presentation choices was validated by a survey. People who answered to the survey were asked to start up the client application and start a visualization of a scenario of their choice and observe it. They were asked to rate the usefulness of the options to change the opacity and application's theme and the toggling of the grid on parking lot. Furthermore,

they were asked how easy it was to start a visualization, how useful were the instructions and how well they perceived horizontal and vertical movement.

The survey got answers from 8 persons. 50% of the people who answered to the survey were from age group 24 – 40 years, 37,5% from 19 – 24 years and 12,5 % from 40 – 60 years old. 62,5% of people who answered rate their computer literacy as 8 in a scale from 1 to 10, where 1 means that a person is computer illiterate and 10 that they know everything about computer. Average of all the people for computer literacy was 6,875.

All of the users used Chrome web browser to access the application, where 6 of the answered used PC with Windows 7, one MacBook running Mac OS X and one used Android device.

62,5% found the option to change the opacity of the machine helpful to perceive the movement of the machines. 75% of persons found the option to change the theme on website helpful and 100% found the option to display grid on parking lot helpful. From these results it can be concluded that the options do help most of people to perceive the movement of the machines better.

37,5% of the answered found the instructions helpful. The others found them somewhat helpful. One of the issues with the instructions was that they were too long. This issue was addressed after survey and the instructions of the application are now shorter and easier to read.

Regarding the fact that there were some issues with instructions, on scale 1 to 10 persons rated the easiness to start a visualization on average at 8,875 points.

Persons were asked to rate on scale 1 to 10 how well did they perceive the horizontal movement of the cars during the visualization, 1 being to lowest grade and 10 highest. The average grade was 8,75 while majority of persons rated their ability to perceive the movement at 10.

The same scale was used for people to rate how well they grasped the vertical movement of the vehicles. Here, the average grade was 7,5. This somewhat low rate can be addressed to the fact that at the time of the survey, dark theme was default in the application. Dark theme might be visually more appealing to some than light theme, but the disadvantage of dark theme is poor presentation of machines' shadows. To address the issue, light theme was made the default of the application.

Even when considering the fact that the number of people who answered the survey was relatively low, it can be concluded that majority of the people, who had no knowledge of the application prior to the survey, found it easy to use and machine's movement was perceived well.

7 Conclusions and future opportunities

In this thesis, two research questions were answered – what is the best approach and tool stack to satisfy the business needs of this particular project and what would be the best way to visualize the particular problem to potential customers. An application was made as a proof of concept based on the answers on the thesis and validated.

As validation confirmed, building and web application is the way to go. It is lightweight, can be made available from plethora of devices and it works also on low-end devices like iPhone 4S is nowadays without lowering the user experience. Web application can be used either by the presenter in front of the potential customers or the customers can try it themselves on their smartphones or PC. The decision to choose plain HTML5 canvas element and plain jQuery for AJAX calls and DOM element manipulations turned out to be a good idea. Using canvas element without any additional framework makes the code of the application more readable and lightweight. The JavaScript source code is 18 KB uncompressed and can be compressed up to 7.82 KB with JSCompress.com[7] tool.

Using Python 3.x with Bottle web framework turned out to be good idea. Everything works as needed, it can be run on all three big operating systems without a problem. Bottle framework is lightweight, responsive and uses little processing power. It is also compatible with all WSGI web servers.

The choice of having the visualization in 2 dimensional space with birds-eye view seems to be efficient approach presenting the movement of machines in a parking lot. The horizontal movement of the cars was clear to all interviewees, so was the vertical movement (robot lifting and dropping a car). The presentation of the vertical movement of the car was made more clear by applying the light theme as a default for the application after the survey.

In conclusion, the thesis answered the two research questions formed in the introduction of the thesis. An application was built as a proof of concept to confirm the findings in thesis. The questions were confirmed in validation part by efficiency tests and the survey amongst the users of the application.

In the future, as the underlying algorithm improves, it is possible to make the application interactive. That means that the user could add new car to the parking lot or retrieve a car by clicking on it. The function to identify the car that was clicked on already exists in the source code of the application and can be easily modified to make an AJAX call to server to retrieve new instructions. Furthermore, Python was chosen as a server side programming language partly because of the existence of Boost.Python[8] – a C++ library that allows seamless interoperability between Python and C++. This is relevant as the underlying algorithm is written in C++.

Furthermore, the application can be used by other upcoming researches to visualize new parking algorithms. The application's ability to work with implicit and explicit instructions is making it versatile to use with different types of algorithms. In addition, machine moving speeds and grid's width and height can be easily configured by changing the global variables in the client script.

[7] http://jscompress.com/ JavaScript compressor
[8] http://www.boost.org/doc/libs/1_61_0/libs/python/doc/html/index.html Boost.Python website

8 References

[1] S. S. McDonald, „Cars, Parking and Sustainability,“ *Transportation Research Forum,* 2012.

[2] Refsnes Data, „Browser Display Statistics,“ 2016. [Online]. Available: http://www.w3schools.com/browsers/browsers_display.asp. [Used 18 May 2016].

[3] Akamai Technologies, Inc, „akamai's [state of the internet],“ 2016. [Online]. Available: https://www.stateoftheinternet.com/downloads/pdfs/Q4-2015-SOTI-Connectivity-Executive-Summary.pdf. [Used 18 May 2016].

[4] Refsnes Data, „Browser statistics,“ 2016. [Online]. Available: http://www.w3schools.com/browsers/browsers_stats.asp. [Used 18 May 2016].

[5] IDC Research, Inc., „IDC: Smartphone OS Market Share 2015, 2014, 2013, and 2012,“ August 2015. [Online]. Available: https://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Used 18 May 2016].

[6] „Usage Statistics and Market Share of Operating Systems for Websites, May 2016,“ Q-Success, May 2016. [Online]. Available: http://w3techs.com/technologies/overview/operating_system/all. [Used 18 May 2016].

[7] N. Langley, „Write once, run anywhere?,“ May 2002. [Online]. Available: http://www.computerweekly.com/feature/Write-once-run-anywhere. [Used 14 April 2016].

[8] D. A. Botwe ja J. G. Davis, „A Comparative Study of Web Development Technologies Using Open Source and Proprietary Software,“ *International Journal of Computer Science and Mobile Computing,* kd. IV, nr 2, pp. 154-165, 2015.

[9] T. Peters, „PEP 20 -- The Zen of Python,“ 19 August 2004. [Online]. Available: https://www.python.org/dev/peps/pep-0020/. [Used 20 April 2016].

[10] P. J. Eby, „PEP 333 -- Python Web Server Gateway Interface v1.0,“ 07 December 2003. [Online]. Available: https://www.python.org/dev/peps/pep-0333/. [Used 20 April 2016].

[11] M. Hellkamp, „Bottle: Python Web Framework,“ 19 April 2016. [Online]. Available: http://bottlepy.org/docs/dev/index.html#. [Used 20 April 2016].

[12] The CherryPy team, „CherryPy -- A Minimalist Python Web Framework,“ 2015. [Online]. Available: http://www.cherrypy.org/. [Used 20 April 2016].

[13] A. Roacher, „Foreword -- Flask Documentation (0.10),“ 2013. [Online]. Available: http://flask.pocoo.org/docs/0.10/foreword/#what-does-micro-mean. [Used 20 April 2016].

[14] J. J. Garrett, *Ajax: A New Approach to Web Applications,* San Fransisco: Adaptive Path, 2005.

[15] N. Nurseitov, M. Paulson, R. Reynolds ja C. Izurieta, *Comparison of JSON and XML Data Interchange Formats: A Case Study,* Montana: Montana State University, 2009.

[16] N. Stewart ja S. Reimers, „Presentation and response timing accuracy in Adobe Flash and HTML5/JavaScript Web experiments,“ *Behavior Research Methods,* pp. 309-327, 2015.

[17] N. Ganesh, *Static Analysis of Malicious Java Applets,* San Jose: San Jose State University, 2015.

[18] P. Trivedy ja S. Harneja, „HTML5 – the new standard for Interactive Web," *International Journal of Research,* kd. 1, nr 10, pp. 39-42, 2014.

[19] AIR FORCE RESEARCH LAB ROME NY INFORMATION DIRECTORATE, *Advanced Visualization and Interactive Display Rapid Innovation and Discovery Evaluation Research Program task 8: Survey of WEBGL Graphics Engines,* New York: AIR FORCE RESEARCH LAB ROME NY INFORMATION DIRECTORATE, 2014.

Appendix

I. Glossary

RCPS Abbreviation of Robot Car Parking System	RCPS Robot Car Parking System lühend, robotiseeritud parkimissüsteem
robroute file The file with the description of the parking lot layout and the instruction steps, generated by C++ algorithm	robroute fail C++ algoritmi poolt genereeritud fail, mis sisaldab parkla plaani ning masinate instruktsioone
JSON JavaScript Open Notation – open-standard format to transmit data objects between systems.	JSON JavaScript Object Notation – lihtsustatud andmevahetusvorming, mida kasutatakse sageli andmete edastamisel ühest süsteemist teise
ASP.NET Server-side web application framework designed to produce dynamic web pages	ASP.NET Serveri poolne veebiraamistik, mis on disainitud dünaamiliste veebilehtede loomiseks
XML Extensible Markup Language – standard format that is human-readable and machine-readable. Is used to share information between systems.	XML Extensible Markup Language ehk laiendatav märgistuskeel – eesmärgiks on struktueeritud info jagamine eri süsteemide vahel
CSS3 Cascading Style Sheets level 3 – language used to describe the presentation of a document written in a markup language	CSS3 Cascading Style Sheets ehk kaskaadlaadistik tase 3 – keel mida kasutatakse märgistuskeeles kirjutatud dokumendi presentatsiooni kirjeldamiseks.
Sanity testing Testing technique where the functions of an application are tested without any scripts to find missing functionality	Sanitaarne testimine Testimistehnika kus rakenduse funktsioone testitakse ilma etteantud stsenaariumita leidmaks puuduolevat funktsionaalsust
FPS Frames per second – commonly used unit to measure frame rate – frequency at which imaging device is showing consecutive images	FPS Frames per second ehk kaadrit sekundis - sageli kasutatud kaadrisageduse mõõtühik.

II. Source code

Source code of the application can be acquired from https://github.com/dot-at/crobots/tree/master/Simulation. The content in the parent directory of Simulation is not created by the author of the thesis. In addition, source code of the application is bundled with the thesis.

III. Installation guide

Requirements for running the application's server:

- Windows, Linux or Mac OS X PC
- Python 3.x installed

To install the application's server, following steps are to be taken:

1. Download the source code of the application from https://github.com/dot-at/crobots/archive/master.zip
2. Unpack the archive
3. In UNIX environments it might be neccessary to make a virtual environment to the Python by using commands
   ```
   virtualenv -p /usr/bin/python2.7 py27env
   source py27env/bin/activate
   ```
   Make sure to be in the root directory of the downloaded unpacked archive before running the commands.
4. If not installed, install Jinja2 and bottle to the Python using commands
   ```
   pip install Jinja2
   pip install bottle
   ```
5. On command line, go to the Simulation directory in the unpacked archive
6. Run the server by using command `python start.py`
7. Open web browser and go to http://localhost:8080/

IV. User guide

These steps can be taken after installing the server on the PC.

- Go to page http://localhost:8080/ in the browser,
- Choose the scenario.
- Choose either realistic or speed choosing mode.
- Click „Choose scenario“ button.
- To see settings, press hamburger menu on the top left corner of the screen.
- Change opacity by clicking or dragging the opacity slider.
- Toggle the grid button for the application to show or hide the screen,
- If speed selection mode was selected, choose the speed by checking one of the checkboxes.
- Click the theme button to toggle between light and dark theme of the visualization,
- To start the automatic visualization, press either on „Start automatic visualization“ button or on any of the cars shown on the main visualization screen on the right of settings.
- To start step-by-step visualization, press on „Step-by-step visualization“. After every step has been finished, the button has to be pressed again for next visualization step.
- The visualization can be resumed automatically and vice versa at any time during visualization.
- To go back to the scenario selection, press on the robot icon on the top left of the screen.
- User guide can be accessed at any time by clicking on question mark on top right corner of the screen.

V. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Suido Valli**,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Visualization of simulations of a robot operated car park system,

(title of thesis)

supervised by Dirk Oliver Theis,

(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **19.05.2016**