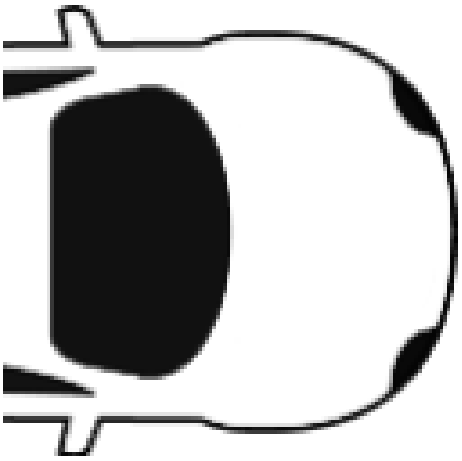
GIRAF

Cars Voice Game

SW613F14



Developing Complex Software Systems

Spring Semester 2014

AALBORG UNIVERSITET
STUDENT REPORT

Department of Computer Science
Selma Lagerløfs Vej 300
9220 Aalborg Ø
Phone (+45) 9940 9940
Fax (+45) 9940 9798
http://www.cs.aau.dk

Title:
GIRAF – Cars Voice Game

Subject:
Developing Complex Software Systems

Project period:
03-02-2014 – 28-05-2014

Project group:
sw613f1f

Participants:
Bruno Thalmann
Mikael E. Christensen
Mikkel S. Larsen
Stefan M. G. Micheelsen

Supervisor:
Thomas Pedersen

Printings: 6

Pages: 63

Appendices: 19

Total pages: 83

Source code:
http://cs-cust06-int.cs.aau.dk/git-ro/cars
commit: 77b33562d7cf0aa202ffe43c11f4f2fef63c9fbe

Abstract:

This project is a part of a multi-project, consisting of 16 groups, from Software-Engineering 6th semester at Aalborg University. The purpose of the multi-project is to develop a series of applications for autistic people.
The focus of the project-group is a game that should help in improving the auditory sense. In the game you must control a car, moving from the left to the right, while avoiding or collecting objects. The car is manoeuvred up and down by the volume of the speech produced by the player.
A speech therapist and two pedagogues from Aalborg Kommune are the main stakeholders of this product. Throughout the project there has been meeting with the stakeholders to match expectations and manufacture requirements.
To contribute to the multi-project process, the report also contains two chapters to be used by future multi-projects. The first chapter contains a description of the generic framework used to develop the game. The second chapter is about the usage of the Git version control system in the multi-project, with a detailed explanation of the usage and common mistakes.

Preface

This report has been made by 6th semester Software-Engineering students at Aalborg University, during the spring-semester of 2014. It is expected of the reader to have a background in IT/software, due to the technical content.

References and citations are done by the use of numeric notation, e.g. [1], which refers to the first item in the bibliography. Whenever the report refers to 'we', it is to be understood as the members of the project-group.

On the following page, a glossary can be found, containing words and terms that might not be apparent from their context when used in the report.

We would like to thank our lecturers, as well as our supervisor Thomas Pedersen, for his guidance throughout the semester.

Glossary

Citizen A person with autism.

Legal Guardian The legal guardian of a citizen.

Institutional Guardian An employee from an institution responsible for citizens.

Pictogram Is a symbol in the form of a simplified drawing, that is used to inform, instruct or caution.

Contents

Introduction

Throughout the entire course of this multi-project, the project-group worked on a project called The Voice Game (formerly Cars).

The overall structure and purpose of the project is explained in chapter 1, along with this project's place in multi-project.

The course was divided into 4 sprints, each with its own process and result. The result of each sprint is documented in its own chapter (see chapters 2-5).

As another result of the project, two chapters were formed, documenting our collaboration with the other groups in the multi project.

The first chapter contains a description of a generic Android Game Framework, which could be the foundation for new games. The second chapter contains a guide to the way version control was handled during the semester.

Each sprint is concluded by evaluation and future works chapters that assess the result of the sprint and recommend areas of the project to work on in a future sprint. Finally, some general advice will be given, to be used by next year's multi-project.

Chapter 1

Multi-project

1.1 GIRAF

Graphical Interface Resources for Autistic Folk (GIRAF) is a semester-wide multi-project collaboration between all Software 6th semester (SW6) groups at Aalborg University. The project started with the SW6 students of 2011 and has run every year since then (this being the 4th year).

The overall idea of the project is to create a multi-purpose Android application, which can be used to ease the lives of citizens and their institutional guardians. The main goal is to recreate already existing physical tools/methods in digitalized versions. Additionally, in order to make it an all-use-and-purposes application, it will also serve as an administration tool for each individual institution that will put it to use.

GIRAF is run in collaboration with Aalborg Municipality. This year there is a total of 6 stakeholders (all institutional guardians or otherwise linked to institutions) who will be available during the course of the project.

1.1.1 Launcher

The 'Launcher' application is the main application of GIRAF, from where you are able to reach all sub-applications. It is also to work as an independent home-screen, replacing the default home-screen of the Android tablet.

In order to start Launcher, which should only be done by an institutional guardian, you have to provide a QR-code. Each institutional guardian have their own QR-code, which is identified by Launcher using the tablet's camera.

Launcher can be in two modes; guardian or citizen mode. The mode is decided by the currently active profile (being either guardian or citizen), which is linked to the corresponding person at the institution. When in guardian mode, certain administrative options are available, that are not available in

citizen mode. It is only possible to select citizens who are associated with the logged in institutional guardian, as the currently active profile. In order to switch back to guardian mode, one only has to select the guardian's profile as the currently active.

Whenever an application is launched, it is provided the id of the logged in guardian. If in citizen mode, the citizen's id is also provided.

1.1.2 Applications

GIRAF consists of a total of 10 applications (excluding Launcher); Administration, The Category Game, The Category Tool, Lifestories, Schedule, PictoReader, PictoCreator, Sequence, The Voice Game, Timer. These applications are either of the following types; practical, administrative or assistive learning. The applications mostly have suggestive names. However, a few examples will follow below.

PictoReader enables a citizen to choose a sequence of pictograms, which can be read aloud, enabling a citizen with little or no speaking capability to communicate better.

Administration is exactly as it sounds. Here, institutional guardians can administrate their institution and associated citizens, legal guardians, and institutional guardians.

PictoCreator is a drawing program that makes it possible for institutional guardians to create new pictograms to be used by the other applications.

The Voice Game is, as mentioned in the introduction, the focus of this project, and will therefore be explained throughout the report. It falls under the category assistive learning.

1.2 Organization

The multi-project consists of 60 people split into 16 project-groups, each group consisting of 4 people (except for two groups, that was the split of a former 4-man group).

1.2.1 Method

Due to the high level of collaboration and the focus on getting out a usable and working product, an agile methodology was in mind when planning the process. The method chosen was Scrum of Scrums. Each project-group would work as a team, with their own day-to-day activities, focusing on their current project during the sprint. Then there would be common weekly status meetings during the sprints, each sprint concluded with sprint-review and sprint-planning meetings.

1.2.1.1 Sprints

The semester was split into 4 sprints, with the last three sprints being of same length (The first was a bit shorter, as this was meant as a sprint solely for getting to know the former projects). In order to take into consideration the courses that were running simultaneously with the multi-project, sprint lengths were based on available hours (with the subtraction of course hours).

At the end of each sprint, a sprint review meeting would be held, where all stakeholders were invited. Here the current products (if possible) would be presented by each group and critiqued by stakeholders and the other groups. Also, the product backlog would be presented for the stakeholders, in order to get some input on what to focus on in the following sprint.

Following the sprint review, and without the stakeholders, the next sprint would be planned. Here groups had a chance to switch focus, if their current project was finished, or if something more important had presented itself as a result of the sprint review.

1.2.1.2 Meetings

In order to stay on top of potential problems, a status meeting was held every week (with exception of sprint-end weeks). The meetings started out with a status report, where a representative from each group presented what they were currently working with, along with any potential problems that needed attention.

If there were problems, these were noted and the proper people informed, so that they could discuss these, without everyone needing to be part of the discussion. This saved a lot of time, as opposed to having everyone discussing every problem.

However, subjects that did need to be discussed with all groups present would be added to the meeting agenda and then discussed at the meeting.

1.2.1.3 Specialists

In order for people to concentrate on their respective projects, a few people were named specialists in different areas (such as server, contact with stakeholders, and version control). Each person were responsible for their area; that it was working as it should, as well as helping people with possible problems.

1.2.2 Stakeholders

Day-to-day contact with stakeholders went through a specialist (contact person). As mentioned, all stakeholders were invited to each sprint review, where they would give feedback to the presented work as well as available for questions. If any more contact was needed, such as arranging a meeting, this was conveyed by the contact person.

1.2.3 Shared Components

There were two main shared components; database and GUI components.

The database consisted of the storage of shared data, as well as providing an interface (`OasisLib`) for accessing this data.

The GUI components consisted of shared graphical components (`giraf-components`), such as backgrounds, buttons and other things that were commonly used and needed to look the same across applications.

As an addition to the shared components, whenever needed, groups would collaborate on minor shared components that could be used across applications.

1.2.4 Tools

In the beginning of the project two representatives from the previous semester told about the tools they used on their semester and their experience with them. Based on these experiences a couple of tools were chosen to aid the project. These tools will be described in the following sections.

1.2.4.1 Redmine

Redmine was used as a center of information, consisting of three main uses; forums, wiki-pages, and issue-trackers.

The forums were used for non-pressing matters. Here, updates to shared components were posted, along with possible instructions if changes needed to be applied to concerned projects.

Wiki-pages were used for static information. There was a main wiki-page, containing group contact informations, general guides, and meeting summaries. Additionally, each sub-project had the possibility for its own wiki-page, to contain whatever information about their project that needed to be shared with others.

Issue-tracking was a tool for tracking progress of a project, as well as the possibility to state issues for other projects that ones own project needed done. Issue-tracking also made it easier for a group to take over the projects if they changed focus after a sprint.

1.2.4.2 Git

Two tools were considered as content management tool for the GIRAF project; Git and SVN. The arguments for SVN were that it was the tool most students were accustomed to and thus would require little time to understand how to use. Git on the other hand had the benefit of having being set up by the previous students working on the GIRAF project. Git is also a superior version control tool as it performs branching and merging on a daily basis without conflicts, it allows for offline committing and, due to its distributed nature, automatically holds multiple backups of each project.

To handle management of the Git repositories and assist with difficulties regarding Git, a Git specialist was appointed. This specialist was part of the project-group. Because of this, an entire chapter focuses on the use of Git (see chapter 7).

1.2.4.3 Jenkins

On the same server that hosted both Redmine and Git, Jenkins was also installed. Jenkins is a continuous integration tool that performs builds of the code pushed to the Git repositories. The two main reasons for using Jenkins were:

- To easily keep track of which projects were building, as to detect if something was not working properly.
- To at all times keep a collection of recent builds.

As Android applications build to `.apk` files, these were to be stored and downloadable from an ftp server. This would ensure that the newest version of all applications were always available for installation.

Unfortunately, even though the build process was up and running, the `apk`s build by the server were not signed. This meant that they could not

be installed on the stakeholders' tablets. This issue was not resolved in the course of the semester.

The status of the builds performed by Jenkins were still monitored throughout the semester and helped in determining the state of each project.

1.3 Cars

Throughout the entire course of this project, the project-group has worked with an application called Stemmespillet (Translates to 'The Voice Game'). It was originally named 'Cars', but was changed to Stemmespillet midway through the multi-project. However, Cars was kept as a working title, and the project will be mentioned as Cars throughout the report. Cars is a sub-application of the GIRAF-application, with its own purpose; that of assisting citizens in learning to control their pitch of speech (This is explained in chapter 2).

1.3.1 Collaboration with stakeholders

In order to establish its overall purpose and to make it as useful as possible, there were three main stakeholders that were cooperated with during the project. Two are institutional guardians at 'Birken', a home for psychologically disabled children, and the third is a speech therapist, associated with Aalborg Municipality. These stakeholders were the most relevant for this sub-application, based on the citizens they worked with on a daily basis. These were also the stakeholders with the most relevant input, both during interviews and sprint reviews.

1.3.2 Collaboration with multi-project

Cars is a sub-application of the GIRAF-application which needs to follow certain project-standards. For Cars there were one overall standard; one for GUI.

Following GUI standards is necessary to give all the applications the same look and feel. Cars will be updated to fulfil these standards in section 4.4.

GIRAF also uses a common database to share data. Data is shared for two purposes; re-acquiring saved data in case of replacement of the tablet, and for multi-user tablets.

The project group also had two responsibilities; Git and Redmine Wiki. Each were assigned to one of the group members. The first one took up a lot

of time for the concerned person, which also led to the collaboration chapter on the subject of Git. The latter did not take up a lot of time.

Chapter 2

Sprint 1

In this sprint we focused on the 'Cars' project from last year's multi-project. The main purpose of the sprint was to get an understanding of the project, while making it usable by eliminating bugs and implementing missing features.

Cars is a small game, in which the user controls a car, moving from left to right on a three-lane road. The user must use voice input, based on pitch, to move the car up and down in order to dodge obstacles on the road and in the end manoeuvre the car into a garage. A screenshot from the Cars app can be seen in fig. 2.1.

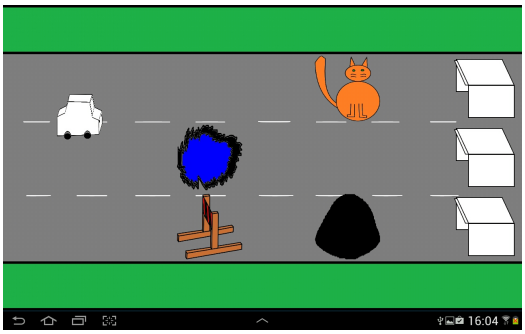


Figure 2.1: Screenshot from the 'Cars' app.

Sound input is an important element of Cars, as it is used to control the car. It was also a part of the game which did not work properly. The movements of the car did not correctly correspond to the voice input, seemingly causing the car to move at random, no matter what kind of pitch was used.

Using voice input based on pitch was an original requirement of the Cars project, worked out between the Cars group and the stakeholders. Its intention was to solve a particular problem concerning some autists, not being

able to control their pitch during speech. Based on its importance and that it does not work, this would be a natural major focus point during the first sprint.

However, we will argue that there are two overall problems with trying to improve the already existing Cars project. Firstly we will argue that it is very difficult to obtain frequency from sound input. Secondly we will argue that it would be easier to completely reboot the Cars project, with new framework and structure, rather than continuing on the existing code and structure.

We will instead argue for using volume instead of pitch, after confirming its validity with the stakeholders. Additionally we will propose a structure, based on the KiloBolt framework.

#	Requirement	Solved	Link
1.	The game must not be a side-scrolling game, because the citizen must be able to see the goal	×	-
2.	The game must make it possible for the citizen to control a car	×	-
3.	The game should contain stars as points	×	-
4.	The goal of the game is to reach a garage	×	-
5.	When the game is won a reward should be given	×	-
6.	It must be possible to save and load settings for a specific profile	×	-
7.	It must be possible to calibrate the microphone	×	-
8.	It must be possible for the user to change the difficulty of the game	×	-

Table 2.1: Requirements, sprint 1.

2.1 Requirements

The first step in continuing on the project, was to collect the requirements from the report written by Jensen, Sørensen, Petersen, and Albertsen [1]. These requirements can be seen in table 2.1. Some of the requirements were vaguely stated in the original report. The elaboration of the requirements will be considered a priority when preparing interviews with the stakeholder.

To keep track of requirements, we have chosen to use a table with the requirements expressed as user stories as well as if the requirement has been fulfilled and where this is described in the report. This makes it easier to manage the requirements when they change or get specified in details when communicating with the stakeholders.

2.2 Obtaining frequency from sound

The current method of controlling the car is based on sound input. When the game is running, voice input is continually recorded and analysed, in order to detect the pitch of the sound input, so that the direction in which to move the car can be decided. The current method of analysing this is by using Fast Fourier's Transformation (FFT) and subsequently choosing the loudest frequency.

In this section we will briefly explain what sound is, in order to understand how to obtain certain characteristics of sound. Based on this knowledge we will argue why it is difficult to obtain frequency from a sound input, especially

sound input with human voice. The technicalities in the section are based on the work of Burk, Polansky, Repetto, Roberts, and Rockmore [2].

2.2.1 Characteristics of sound

Sound is a physical phenomenon which acts like a wave and therefore has the same characteristics as a wave. So basically sound is a movement of air and the way we perceive sound is an interpretation of these movements.

Simply speaking, the volume (or loudness) of a sound, is the amplitude of the sound wave. The pitch of a sound, how high/low we perceive it, is the frequency of the sound wave. These two characteristics are independent, meaning that two sound waves with same frequency, but different amplitudes, will sound like the same pitch, but one being louder than the other. An example can be seen in fig. 2.2.

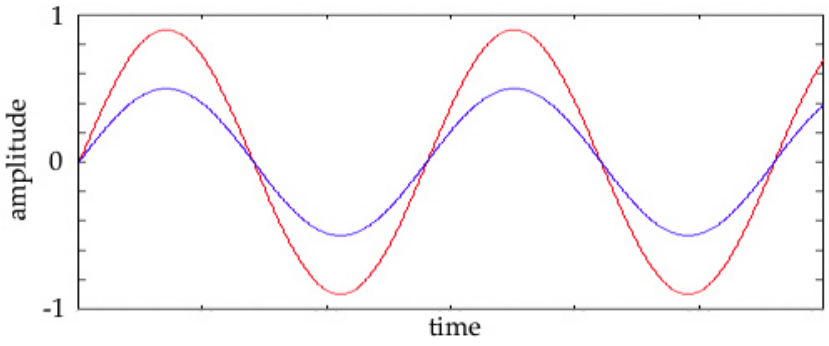


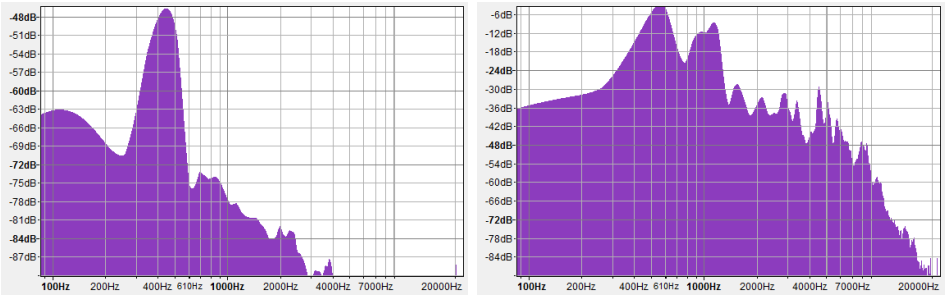
Figure 2.2: Two sound waves (represented as sine functions) with same frequency, but different amplitude.

There is also a third characteristic, which is more abstract and not directly a physical attribute, namely, timbre. Timbre is a composition of different sound waves (spectra), which makes up a specific sound (again, this is perceptual) and envelope (transients), which describe the different stages of a sound.

2.2.2 Obtaining frequency

The frequency of a sound can be obtained from a sound input interpreted as a wave (sine function). However, some sounds are simpler than others in their composition of waves. These simpler sounds could be considered ”cleaner” than others, meaning they consist of only a single or very few waves, where it is easy to identify a primary amplitude and frequency.

Human speech, among other sounds, consists of multiple sound waves, each with varying amplitude and frequency. In this case, it may be difficult to identify a single wave as the most important, and choose its frequency as the main frequency. A comparison between frequency spectra can be seen in fig. 2.3.



(a) Frequency spectrum for a 440 Hz generated tone

(b) Frequency spectrum for human voice input (in an attempt to reach 1200Hz)

Figure 2.3: Frequency analysis in Audacity

2.2.3 Using FFT

Fast Fouriers Transform (FFT) is a method of extracting the smaller components of a sound input. As mentioned previously, a sound is a composition of several sound waves with each their own frequency and amplitude. Therefore the result of the FFT on a sound input, is the existing frequencies and their amplitudes. The result is called a frequency spectrum and can be seen in fig. 2.3.

2.2.3.1 Cars and FFT

In Cars, part of the method of extracting a frequency from the sound recording, is by using FFT. How and why exactly the frequency spectrum is used for extracting a single frequency was not properly documented in the Cars report, it is only certain that the result from the FFT and the following manipulation leads to the loudest frequency value. This value is then used in comparison with the calibrated low and high threshold values, to determine in which direction to move the car.

This poses several problems. Firstly, human speech is complex and contains many different individual sound waves. What sounds to us like a high pitch voice, can easily peak at a much lower frequency, as higher amplitude

low frequency sounds will not always sound as loud as lower amplitude high frequency sounds. This is due to our perception of sound. An example of this can be seen in fig. 2.3b, where in an attempt to hit a high frequency of approximately 1200 Hz, the highest peak was actually at approximately 600 Hz.

Another problem is the method of grouping together frequencies linearly in the FFT method. Our hearing is logarithmic, meaning that a slight change in frequency is much more apparent to our perception at lower frequencies, than at higher frequencies. When analysing the input, frequencies are not registered as their exact frequency, but are grouped together in ranges, linearly. This means that lower frequencies that are very different in regards to pitch may be grouped together. This impacts the result of the analysis, where one group of frequencies will have a higher amplitude, without it being the one we thought was the most apparent to our perception.

2.3 Design Evaluation

The original goal of the first sprint was to refactor the Cars project. After using a few weeks reading and changing the code it turned out to be more complicated than initially assumed. The changes we made were small, but required extensive work because of the lack of structure in the project. It was decided that a total redesign of the application would be the better option. Instead of spending a lot of time trying to understand and modify the code, a redesign would not only take shorter time, but would also result in a better design.

The following sections will summarize the problems that made the task of refactoring the cars project near impossible.

2.3.1 Inheritance

It seems that the object oriented programming concept inheritance is not used in the project 'Cars' code. Inheritance in object oriented programming is when you have a class based on an other class. This prevents code duplication which improves maintainability.

An example from 'Cars' is the class `GameObject` which is an empty super class to the following classes:

- `Barricade`
- `Bump`
- `Car`
- `Cat`
- `Garage`
- `Rock`

The classes shown above all have a *draw* method which are identical. This is a classic example on a method which should have been in the super class. If it would have been in the super class all its derived classes could have used the method. This would then be good code reuse.

2.3.2 Methods

Understanding a method is difficult when it is big and has different unrelated operations. It would improve the readability of the code to split them up to small methods instead, for instance an operation per method.

An example of an operation could be to fill a whole array with a certain number. The code for this would be a for-loop filling an array with the number. Instead of having the code in the method, where you then will use it, it is better to extract it to a new method and call that. This also improves readability by providing a describing name for the method making it easier to tell the intention of the method. Keeping methods short makes the maintenance and refactoring of the code easier, because the code then have a higher readability.

The *Run* method (code example A.1) from 'Cars' from the class `RecorderThread` is an example of a method which is impossible to assess and should be divided into several sub methods.

2.3.3 Graphic

For the graphics in 'Cars' the authors have chosen to use OpenGL [3]. OpenGL is a tool for advanced graphics and it can be used for 3d graphics. For a simple game like 'Cars' it is simply overkill. As mentioned in the developer guide for Android [4], OpenGL should be used for advanced 3d graphics and if you need calculation power from the GPU. Instead the guide suggest that the standard tools in the Android framework would be sufficient.

2.3.4 Object Oriented Design

A problem throughout the code is that basic object oriented principles are not used or are not used properly. Classes do not have a clearly defined responsibility, and functionality is instead spread out on several classes. This breaks the concept of keeping high cohesion in object oriented design. Cohesion is a measure of how the responsibilities of a module are related. If the responsibilities are highly related, the module is said to have high cohesion. An advantage of high cohesion is that it improves maintainability of the system, because changes in one module requires fewer changes in other modules.[5]

An example of low cohesion in the cars project is the `Car` class(code example C.1). This class has the responsibility to draw itself, move itself according to pitch of the sound recorded by the microphone, calculate collisionboxes and determine if the car has collided with an obstacle and close the garage if the car has driven into the goal garage.

Another concept the existing code breaks is coupling. Coupling is the number of dependencies between modules. It is desirable to keep this number as low as possible, known as having low coupling. Low coupling has the advantage of making it easier to make changes in module, as they are not independent on each other.[5]

When trying to read and understand the existing implementation of the control of the car by pitch, it turned out to be very hard. Because the code responsible for handling the pitch is spread out on several classes, this relatively simple change becomes very complicated. Another example is the responsibility for drawing objects which is on the individual object instead of using inheritance as described in section 2.3.1. This makes it very hard to make consistent changes to the way objects are drawn.

2.4 Redesigning Cars

To best avoid ending in a situation similar to the above, it was deemed important that a framework was found on top of which the new game could be built. Such a framework could define a *proper* structure for game development, allowing the project group to focus solely on the game development without having to consider android.

Kilobolt In *"Beginning Android Games"*[6] an open source framework is suggested for game development. The framework is presented and described on kilobolt.com[7] using a simple example of a game. The framework gener-

alizes Java game development by defining a set of interfaces describing the structure of a game. How the framework is used can be seen in chapter 6.

These interfaces are then implemented in an Android version specifying how the generalized framework should be executed on Android devices. In the implementations presented on kilobolt.com there are some inconsistencies between the interfaces and Android implementations. An example of this is that in some cases the Android implementation defines more methods than the interface. Any such inconsistencies would have to be fixed if implementing for multiple platforms. Simultaneously any future expansion of the framework would require all methods to be defined in both interfaces and Android implementations.

Since the project for which the framework is used is Android-only, the generalized part of the framework was removed.

2.4.1 The framework

The following gives a description of the internals of the framework described above. Not every class in the framework will be described as not all are essential to its functionality.

The Game Activity A game in the framework is implemented as an activity, allowing for simple interoperability with other Android components. The game itself is primarily a shell, connecting the parts of the framework. Actually implementing a game is done via screens. A screen defines a *state* in a game, drawing context specific graphics. A game defines its initial screen, and screens are capable of replacing the active screen of a game. Using this method, parts of a game can be implemented separate of the remaining game, resulting in higher cohesion.

Rendering In order to handle continuous updates of a game (frames) a separate thread is started, which performs these updates. The thread uses a canvas for drawing, which is exposed only through a set of methods that draw on it. Using this method rendering can be handled directly in screens without access to the canvas itself. Ensuring that all drawing is done this way gives a low coupling of the rendering aspect of games.

Besides the canvas, the render-thread draws on a bitmap as a back buffer which the canvas draws when it is ready. This is known as double buffering. Whenever a new frame has been drawn on the back buffer, using the methods described above the content of the buffer is drawn onto the view associated with the activity. After the update a new frame-render is started.

As the rendering of frames is done continuously in a separate thread, there is no way to ensure a fixed framerate. In order to implement the updates frame independent the thread stores a timestamp whenever a frame has been rendered. This is then later used to determine the time the has passed since the last frame. This information can be used to properly update changes in a game, such as ensuring that a car moves at a constant speed.

2.4.2 Solution

2.4.2.1 Change from Frequency to Volume

Because of the problems obtaining the right frequency, explained in section 2.2.2, the idea was to change the control of the car to volume. The idea was proposed to one of the stakeholders *Tove Søby* (the speech therapist) so that there would not be worked on a solution which would not be used anyway. The idea was accepted, see appendix D for email correspondence.

2.4.3 New Prototype

Because of the decision to recreate the game from scratch, this sprint has been more like a pregame from SCRUM, and the actual product is therefore minimal. The framework that was suggested in section 2.4 has been used to recreate a minimal working example of the game. The result can be seen on fig. 2.4.

The car drives up and down and is controlled by the volume of the player's voice. Loud sounds will make the car drive up while quiet sounds makes it drive down. Only sounds over a certain threshold makes the car move, meaning that no sound causes the car to drive straight forward.

The squares in the right side of the screen represent garages, and the color represents its state. A green garage is open, a yellow garage is closing while a red garage is closed. It is possible to drive into a green garage which will then turn yellow for a short amount of time and then turning red. If the car drives into a red garage it 'crashes', resetting its position to the starting position in left of the map vertically in the middle of the screen.

The purple squares represent obstacles that have to be avoided. If the car drives into one of these it 'crashes' and is being reset.

2.5 Evaluation

This section contains an overview of the requirements and whether they have been fulfilled during this sprint. If they are fulfilled it is explained how.

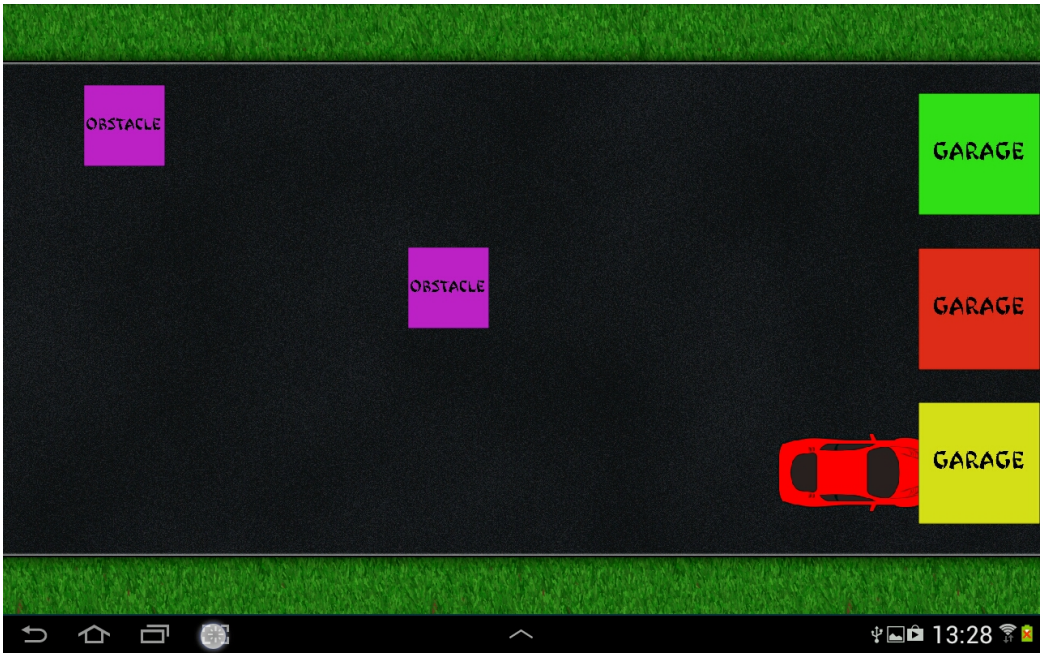


Figure 2.4: The product after sprint 1

#	Requirement	Solved	Link
1.	The game must not be a side-scrolling game, because the citizen must be able to see the goal	√	section 2.5
2.	The game must make it possible for the citizen to control a car	√	section 2.5
3.	The game should contain stars as points	×	-
4.	The goal of the game is to reach a garage	√	section 2.5
5.	When the game is won a reward should be given	×	-
6.	It must be possible to save and load settings for a specific profile	×	-
7.	It must be possible to calibrate the microphone	×	-
8.	It must be possible for the user to change the difficulty of the game	×	-

Table 2.2: Requirements fulfilled after sprint 1.

Requirement 1 Looking at fig. 2.4 the game starts at the left side of the screen and ends at the right side of the screen, without any scrolling.

Requirement 2 It is possible to control the car with volume, making the car move up when the input volume is high and down when it is low. The car stays at its position when there is no input. For more detail see section 2.4.3.

Requirement 4 Looking at fig. 2.4, on the right side there are three boxes labelled 'garage', these represent the place where the graphics for the garages should be. How they work is explained in section 2.4.3.

2.6 Future Works

This section contain the assignments which should be considered looking at in the next sprint.

In the next sprint the focus should be on looking at the rest of the requirements, but especially improve and finish the following requirements.

Requirement 2 The implementation of controlling the car right now only works using the microphone on the tablet used for the development process. So in the next sprint there should be looked at a solution for this problem.

Requirement 4 The boxes that are used as garages right now, should be replaced with some proper graphics.

Chapter 3

Sprint 2

In the previous sprint, a new framework suggested for the 'Cars' app and the implementation of it was begun. The main focus of this sprint is to continue this implementation, while getting some feedback from the stakeholders, to solve uncertainties regarding the old requirements.

The first part of this sprint chapter contains an interview with the stakeholders, from which new requirements were obtained. The result is a new list of requirements, based on both the old requirements and the newly obtained.

Following the interview, the design and implementation of the new requirements can be seen. This is done in an informal manner, explaining the overall idea, and results where appropriate.

The result of the sprint is a working application, which implements most of the requirements. However, not all of them were fully implemented, and some not at all.

3.1 Interview

A semi-structured interview [8] was carried out April 4th 2014 with two pedagogues from Birken; Mette Als Andreasen og Kristine Niss Henriksen.

3.1.1 Purpose

At the start of the sprint we were still solely working out from the requirements found in last year's Cars project report (see section 2.1). However, we found that some of these requirements were still not clear. Also, the overall purpose and application of the Cars app was not clear either.

The goal of the interview was then to establish the overall purpose of the Cars app; how it would be used in practice and how it would relate to real-life problems. Another important issue was to figure out how to adjust difficulty (see table 2.1, requirement 8) in order to make the game more usable, and not be too easy or difficult for some. Additionally there was mentioned points in the old requirements (see table 2.1, requirement 3), we would also like this clarified, as it was not implemented by the previous Car project group.

3.1.2 Planning

Prior to the interview we agreed upon topics we would like discussed with the stakeholders. We decided to stick to topics, and not specific questions, to allow a more open discussion. This was because we did not want to lock ourselves too much to the old requirements, in case there were some bigger and better changes available.

There were two main topics;

Overall purpose As of this point, we were still unclear on the overall purpose of Cars. So we wanted to the interviewees to explain this, along with how they imagined using it.

Difficulty customization We would like to discuss how to customize difficulty of the game. Variables like speed and size of the car, number and size of obstacles, and number of garages were examples of things to change in order to change the difficulty.

3.1.3 Result

The result of the interview can be seen here as a list of requirements. A resume of the interview can be seen in appendix E.

1. The car is controlled in such a way, that the vertical position of the car is relative to the current loudness of the player's voice.
2. There is a digit between 0 and 10 displayed on the car as well as obstacles, representing the loudness level, based on its vertical position.
3. Besides the scales from 0 to 10, both speed and loudness have pictograms illustrating some of the values on these scales.
4. It should be possible to pause the game. When the game is paused, a loudness-barometer is displayed next to the car, further visualizing the current loudness.
5. Speed is alterable. The speed level is represented as a digit between 0 and 10.
6. The placement and number of obstacles is alterable.
7. The placement of obstacles should be in such a way, that it is possible to adapt it to both citizen with tendency to speaking too loud as well as those speaking too low.
8. The graphics need to be simple, as some citizens have a low attention span and are easily distracted.
9. It should be possible, in settings, to switch between avoiding objects and picking objects up.
10. When picking objects up, this is linked to pictogram categories.
11. It is important that the pickup/category "mode" is optional, as not all development levels are capable of both sound mode and category mode.

3.2 Requirements

Initially the requirements were the same as last sprint. However, due to the interview carried out during the sprint, some new requirements were gathered, along with some of the old requirements being changed.

Here will be represented a new and complete list of requirements, as a merge between the initial requirements and the newly obtained.

Note: The following is based on table 2.1. Requirement 2 was vague so it was altered to requirement 2 on table 3.1 so it now is more precise. This also means that the old solution from sprint 1 has to be changed, so this

requirement is now unsolved. Requirement 3 was completely removed, as the focus should be on using the voice, not winning as such. Requirement 8 was vague so it was split into several new requirements regarding difficulty from the interview.

#	**Requirement**	**Solved**	**Link**
1.	The game must not be a side-scrolling game, because the citizen must be able to see the goal	√	section 2.5
2.	The car is controlled in such a way, that the vertical position of the car is relative to the current loudness of the player's voice.	×	-
3.	The goal of the game is to reach a garage	√	section 2.5
4.	When the game is won a reward should be given	×	-
5.	It must be possible to save and load settings for a specific profile	×	-
6.	It must be possible to calibrate the microphone	×	-
7.	There is a digit between 0 and 10 displayed on the car as well as obstacles, representing the loudness level, based on its vertical position.	×	-
8.	Besides the scales from 0 to 10, both speed and loudness have pictograms illustrating some of the values on these scales.	×	-
9.	It should be possible to pause the game. When the game is paused, a loudness-barometer is displayed next to the car, further visualizing the current loudness.	×	-
10.	Speed is alterable. The speed level is represented as a digit between 0 and 10.	×	-
11.	The placement and number of obstacles is alterable.	×	-
12.	The placement of obstacles should be in such a way, that it is possible to adapt it to both citizen with tendency to speaking too loud as well as those speaking too low.	×	-
13.	The graphics need to be simple, as some citizens have a low attention span and are easily distracted.	×	-
14.	It should be possible, in settings, to switch between avoiding objects and picking objects up.	×	-

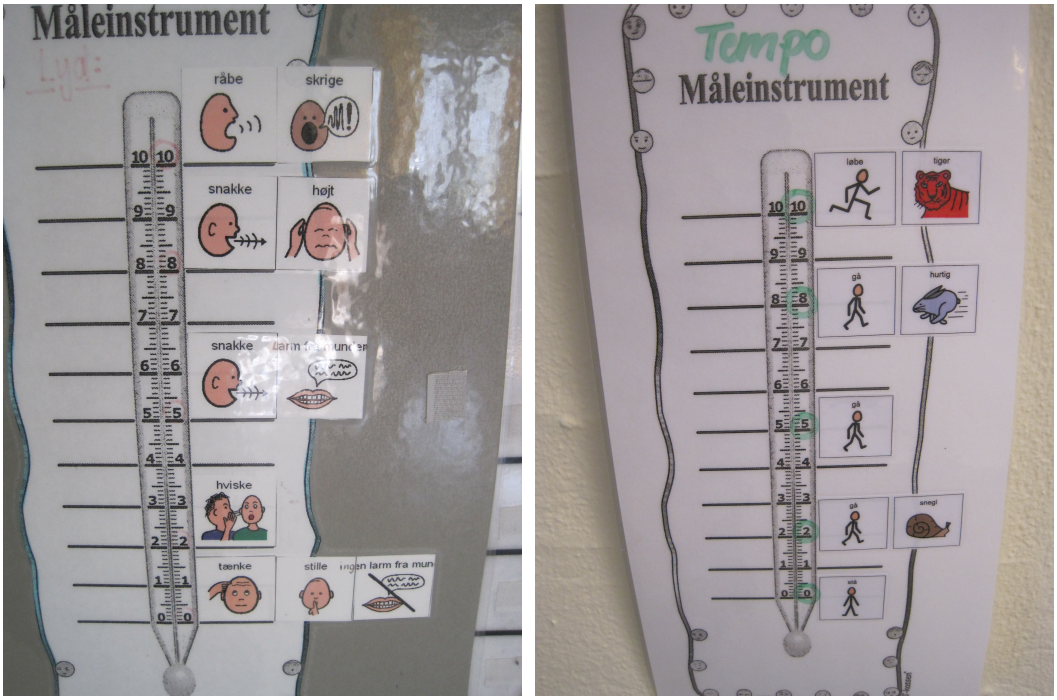
15.	When picking objects up, this is linked to pictogram categories.	×	-
16.	It is important that the pickup/category "mode" is optional, due to different capabilities for each citizen.	×	-

Table 3.1: Requirements fulfilled after sprint 2.

3.3 Design and Implementation

3.3.1 Sound and speed gauges

The interview with the stakeholders (described in section 3.1) revealed that they use a gauge system to illustrate speeds and volume levels at the institution. The gauges used are pictured in fig. 3.1. Each gauge is from 0 to 10, and some of the levels have a pictogram associated.



(a) The sound gauge **(b)** The speed gauge

Figure 3.1: The gauges used at 'Birken'

The requirements concerning getting the gauges incorporated in the game

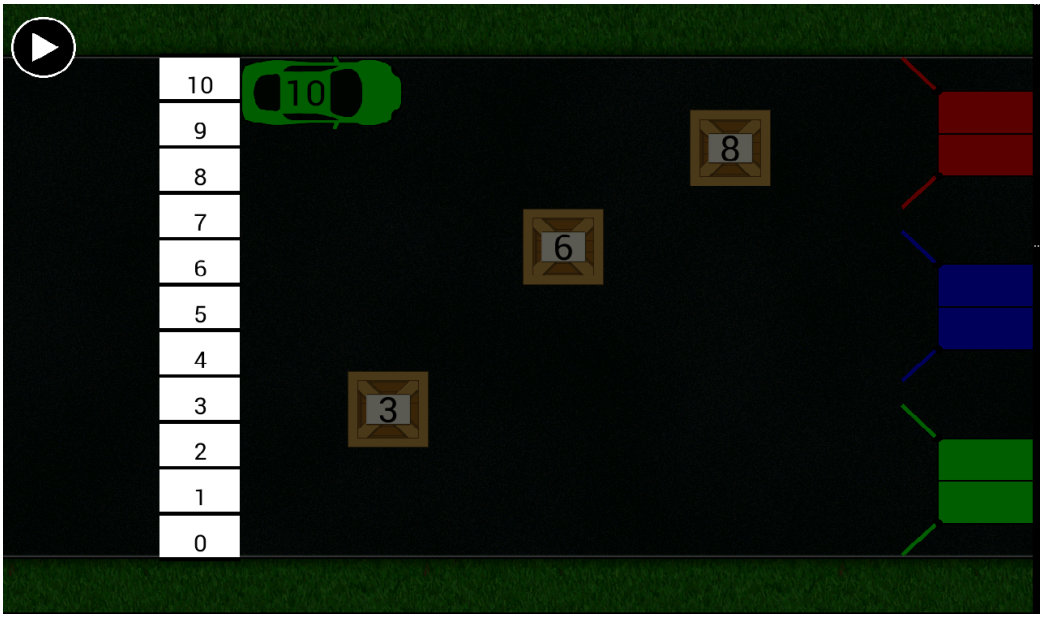


Figure 3.2: The game in a paused state. The game is seen in the background and a sound gauge is shown left of the car

are requirement 7, requirement 8, requirement 9 and requirement 10 in table 3.1.

Requirement 7 is met by drawing the number corresponding to the position of the object on the gauge. This can be seen on fig. 3.2.

Requirement 8 was not fulfilled.

Requirement 9 is partially fulfilled. The pause functionality is achieved by having a pause button in the upper left corner. When the game is paused the game is faded out, and a representation of the gauge is placed left of the car so it is possible to see the full scale in the game area. This can be seen on fig. 3.2. This placeholder scale is to be replaced with a gauge similar to the one seen on fig. 3.1a.

The fulfilment of these two requirements can be seen on fig. 3.2.

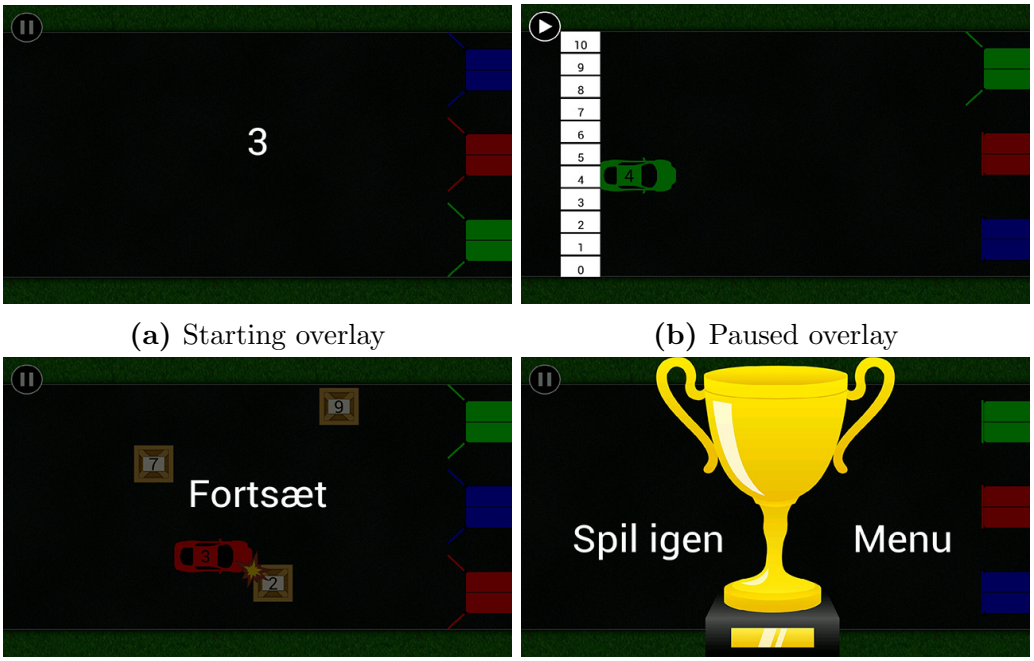
Requirement 10 was only partially fulfilled in this sprint, by making the speed alterable. The gauges and pictograms are not yet visible on the settings module, which can be seen on fig. 3.3. The shown number is the value of speed used internally. The speed is altered by pressing in the left or right

Figure 3.3: The temporary implementation of alteration of the speed

side, decreasing or increasing the speed respectively. These 'invisible' buttons are to be changed to pictograms from the gauge on fig. 3.1b.

3.3.2 Screen overlays

At certain stages of the game, we need to display overlays, containing important information and/or buttons for navigation. The game screen contains the four overlays; Starting, Paused, Crashed, and Won, which are described hereafter. All of the overlays can be seen in fig. 3.4.



(a) Starting overlay

(b) Paused overlay

(c) Crash overlay

(d) Won overlay

Figure 3.4: The four overlays for the game

Starting Overlay is displayed when a new game is started (from either the main menu or the Win Overlay). It is a short countdown, enabling the player to get ready, as opposed to starting the game straight away, or simply

waiting without any feedback for the player. It is a 3 second countdown (3.. 2.. 1.. Go), as can be seen in fig. 3.4a.

Paused overlay is linked to requirement 9 in table 3.1. At all times when in the game screen, a pause button is visible in the top left corner. When pressed, the game is paused (the car stops moving) and the loudness gauge is displayed (see fig. 3.4b).

Crashed overlay is for when the citizen hits an obstacle or a wrong garage. It is a simple screen displaying an explosion at the impact spot, stopping the game and awaiting the 'Fortsæt' (continue) button to be pressed. The crash overlay can be seen in fig. 3.4c.

Won overlay is displayed after all garages have been closed. This is linked to requirement 4 in table 3.1. A trophy is displayed along with two buttons; 'Spil igen' (Play again) and 'Menu'. The overlay can be seen in fig. 3.4d.

3.3.3 Graphics

According to requirement 13 in table 3.1, the graphics must be kept simple. To accommodate this both the background and the objects have been kept simple and two dimensional. The graphics used in the game can be seen on fig. 3.2. The background consists of a road which is a grey texture, and there is a border of grass in the top and the bottom of the screen, consisting of a green texture.

The individual objects can be seen on fig. 3.5. The objects are chosen so they are simple and yet recognizable.

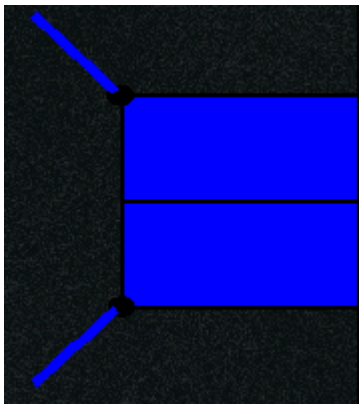
3.3.4 Control of the car

At the interview the stakeholders expressed that they needed to change how the car was controlled. They wanted the car to reflect the current sound level of the speech recorded by the microphone. This is expressed in requirement 2 in table 3.1.

This has been implemented in the game by mapping the values recorded by the microphone to a range that corresponds to the height of the road. This mapping is shown on fig. 3.6. The microphone records sounds and outputs values between 0 and some max value, shown as 1000 for simplicity. When calibrating the microphone two values are defined, the minimum value reflecting background noise caught by the microphone, and the maximum

(a) The car

(b) An obstacle



(c) One of the garages

Figure 3.5: The graphics used in the game

value determined by the sound level of the player's shouting. These two values are mapped to values between 0 and the height of the road, in the example 800.

The returned value is the position on the screen where the car needs to drive towards at the current sound level.

3.3.5 Map Editor

The map editor has been developed due to requirement 11 and requirement 12 in table 3.1. The map editor makes it possible for the user to make a map that fits to the citizen's needs. A screenshot can be seen on fig. 3.7, where there are added two obstacles. The map editor is a screen of the game with only garages. Users are able to add an obstacle by touching the screen - and the obstacle will be placed there. Users can remove an obstacle by touching it.

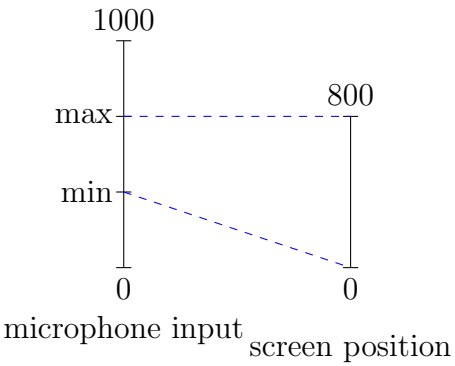


Figure 3.6: The mapping from microphone input to position on the screen

3.3.6 Settings

The customization (settings) of the game is linked to requirements 5, 6, 10, 14, and 16 (see table 3.1).

The settings activity handles three things; Altering the speed of the car, Changing the colors of the garages, and Calibrating the microphone. A screenshot of the settings activity can be seen in fig. 3.8.

Speed is changed by tapping the ends of a small fragment of the game, displaying only one lane and the moving car. By tapping the left end, the speed is decreased and by tapping the right, it is increased. Whenever a change to speed is made, it is immediately visible in the game fragment. The ability to alter the speed is directly related to requirement 10.

Color is as of now changed by picking a color from a dropdown box. Changing the color of the garages is not stated as a requirement, but is a feature that was implemented by the previous Cars project.

Microphone Calibration is done in a small game fragment, as can be seen in the bottom of the settings screenshot. This is in regards to requirement 6. The way it is done atm, is when holding either 'Høj' (high) or 'Lav' (low) down, during this time the microphone records and when they are released the average of all the recordings during the time are averaged and set as the appropriate threshold value.

Requirement 5 regarding loading and saving settings to a profile has not yet been implemented.

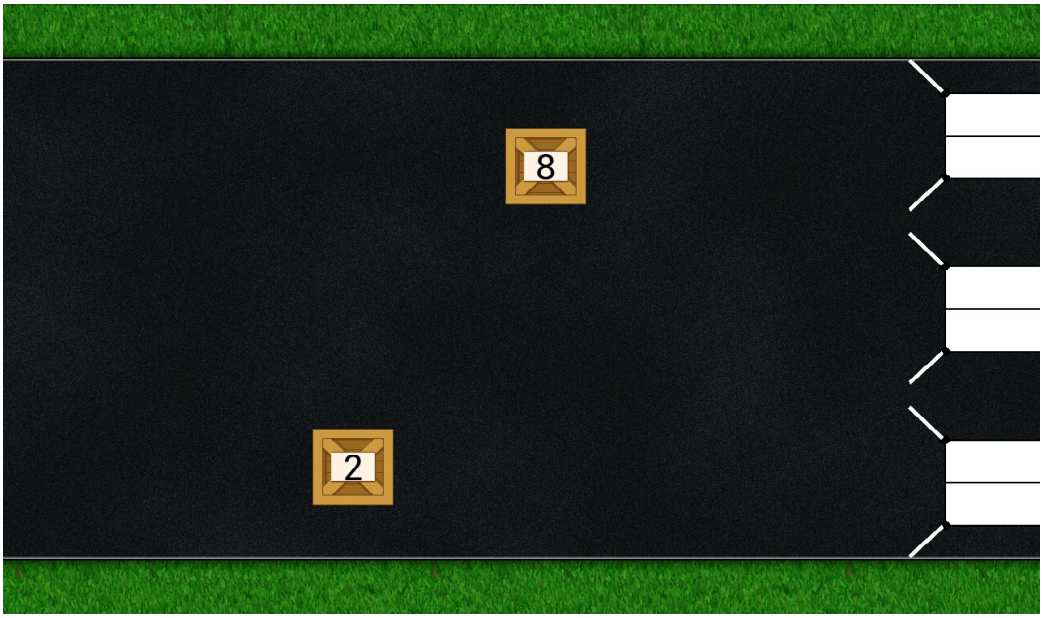


Figure 3.7: The map editor, which makes it possible to customize the obstacle placement.

Requirements 14 and 16 regarding picking up category pictograms is not yet implemented either.

3.4 Evaluation

This section contains an overview of the requirements and whether they have been fulfilled during this sprint. Comparing table 3.1 and table 3.2 it is possible to see the progress and what has been solved during sprint 2 compared to sprint 1. Each requirement, if fulfilled, has a link to where it is explained.

#	Requirement	Solved	Link
1.	The game must not be a side-scrolling game, because the citizen must be able to see the goal	√	section 2.5
2.	The car is controlled in such a way, that the vertical position of the car is relative to the current loudness of the player's voice.	√	section 3.3.4
3.	The goal of the game is to reach a garage	√	section 2.5
4.	When the game is won a reward should be given	√	section 3.3.2
5.	It must be possible to save and load settings for a specific profile	×	-

6.	It must be possible to calibrate the microphone	×	-
7.	There is a digit between 0 and 10 displayed on the car as well as obstacles, representing the loudness level, based on its vertical position.	√	section 3.3.1
8.	Besides the scales from 0 to 10, both speed and loudness have pictograms illustrating some of the values on these scales.	×	-
9.	It should be possible to pause the game. When the game is paused, a loudness-barometer is displayed next to the car, further visualizing the current loudness.	√	section 3.3.2
10.	Speed is alterable. The speed level is represented as a digit between 0 and 10.	×	-
11.	The placement and number of obstacles is alterable.	√	section 3.3.5
12.	The placement of obstacles should be in such a way, that it is possible to adapt it to both citizens with tendency to speaking too loud as well as those speaking too low.	√	section 3.3.5
13.	The graphics need to be simple, as some citizens have a low attention span and are easily distracted.	√	section 3.3.3
14.	It should be possible, in settings, to switch between avoiding objects and picking objects up.	×	-
15.	When picking objects up, this is linked to pictogram categories.	×	-
16.	It is important that the pickup/category ”mode” is optional, due to different capabilities for each citizen.	×	-

Table 3.2: Requirements fulfilled after sprint 2.

3.5 Future Works

This section contain the assignments which should be considered looking at in the next sprint. In the next sprint the focus should be on looking at the rest of the requirements, but especially improve and finish the settings and the control of the car. The specifics of these assignments will be discussed in the following.

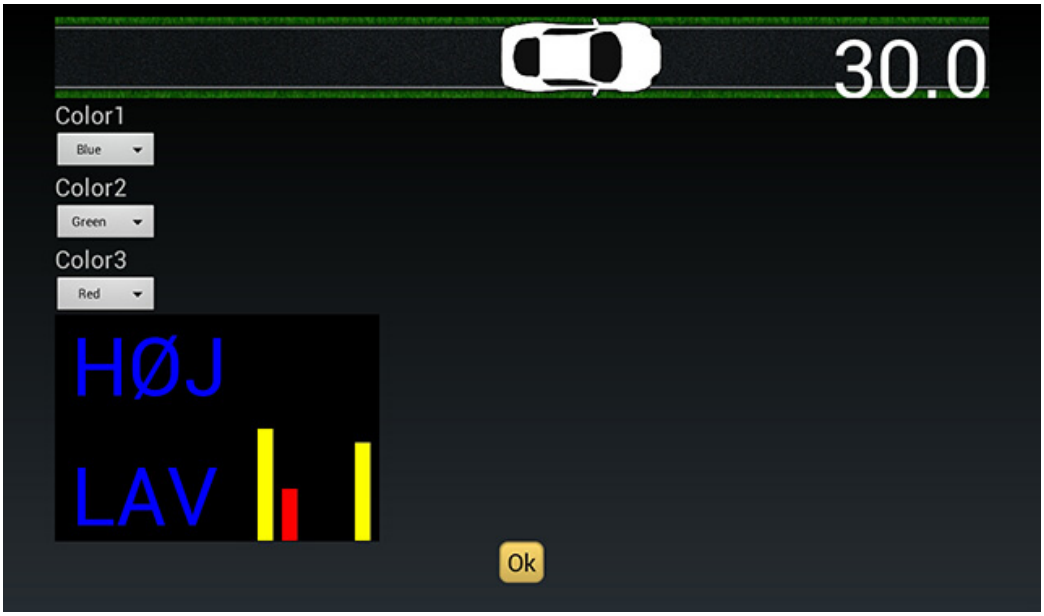


Figure 3.8: The settings activity

3.5.1 Settings

The main topic which has to be improved are the settings. The layout, fig. 3.8, has to be modified so that it is clear which settings group together and what it is the user is able to change. The three colours chosen for the garage should also be changed so the user is able to pick an arbitrary colour. The microphone calibration setting also has to get changed to some more suitable layout.

3.5.2 Control of the car

The control of the car does still not function properly with the calibration in the settings. It seems that when the car was calibrated it does not work properly in the game, but when using hardcoded values the game works quite okay. The problem therefore seems to be the calibration in the settings.

Chapter 4

Sprint 3

As suggested in previous sprint, the main focus of this sprint was to focus on the settings and the control of the car.

In the settings activity there was some major changes in the way the speed is set. A speed gauge was added, and the desired speed is set by pressing at the corresponding position on this gauge.

The selection of colors was changed to display a box filled with the selected color, and this box can be pressed to open up the color-picker.

The application was also hooked up to the common database so that the profile settings can be saved and synchronized.

At last some changes were made to the way the car is controlled, in order to prevent it from "jumping" when the sound level changes rapidly.

4.1 Settings

Since the requirements has not changed at all since last sprint, the ones regarding settings remain the same. During this sprint the following requirements were handled; 5, 6, 8, and 10 (see table 4.1).

The settings activity still handles the same three things as previously; speed of the car, garage colors, and calibration of the microphone. The method for changing the speed and garage colors was changed during this sprint. Calibration of the microphone was not changed, as no proper solution was found this sprint. A screenshot of the settings activity can be seen in fig. 4.1.

Altering of Speed was changed according to requirements 8 and 10. The speed is set by either pressing the gauge, changing the position of the gauge-bar, which indicates the currently set speed. Alternatively the pictograms

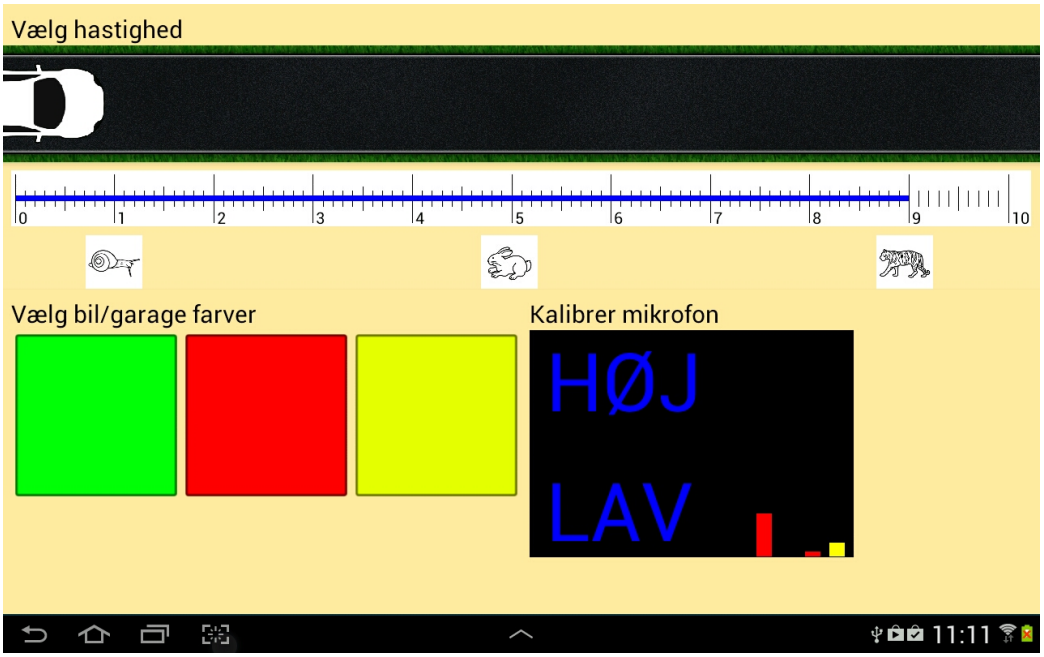


Figure 4.1: Screenshot of the Settings activity

(currently three; snail, rabbit, and tiger) can be pressed, moving the gauge-bar and setting the speed appropriately.

Choosing garage colors is now depicted as a colored box, where the box's color is corresponding to the selected garage color. A color can then be changed by pressing one of these boxes and choosing a new one from a color-picker (see section 4.4).

4.2 Connection to the database

As mentioned in section 1.2.3, apps within GIRAF can make use of a shared database, so that profiles and their settings can be saved.

Apps opened within the GIRAF launcher are supplied with two integers; `currentChildID` and `currentGuardianID`. The first integer identifies the currently active child, however, this value will be -1 if no child is active. The second integer identifies the current institutional guardian, and is always supplied.

Settings are loaded from the database whenever the settings, map-editor or main activities are started. Settings are saved whenever the settings or map-editor activities are terminated.

Accessing the database In order to access the local database (called LocalDB within GIRAF), a library `OasisLib` must be used. `OasisLib` makes available various controllers. The one used for saving and loading settings is called `ProfileApplicationController`. By supplying the current `Application` , which is a class that contains all identifying information about an application (name, package-name etc.) and the current profile id, a `ProfileApplication` can be obtained, containing the settings for the corresponding application and profile. `ProfileApplicationController` also contains methods for obtaining and modifying the loaded `ProfileApplication`.

Settings are saved within `ProfileApplication` as a map `Setting<String , String, String>`, where the strings represent key, type, and value accordingly. Cars has 3 setting keys; ”speed”, ”colors”, and ”calibration”. All values are therefore converted to strings and put in the map with their corresponding key. For colors and calibration, which contain more than one value, different types are also supplied.

4.3 Stabilizing the car

The vertical motion of the car is determined by voice input. Using the volume recorded by the microphone, a value in the interval $[0-1]$ is calculated, where 0 represents silence and 1 represents *maximum* volume. The value is then mapped linearly to a vertical position in the game. The car is then moved to this position on the game field.

This section will explain the different methods of stabilization of the car that was considered when trying to improve the motion of the car.

Positioning the car Given a vertical position where the car *should* be placed, the car could simple be moved to its destination on each frame update. This is the simplest solution as it requires no additional translation of the position input. Let $m_f \in [0 \dots 1]$ represent the translated measurement at frame f and $c_f \in [0 \dots 1]$ represent the position of the car at frame f.

$$\text{We then have that } c_f = m_f.$$

There is however a problem with this approach. The input from the microphone will sometimes yield spikes[1] in terms of the amplitude. The reason for these spikes has yet to be determined. The effect however is quite

[1] In this context *spikes* refers to any deviation in the measured amplitude. This includes any white noise recorded by the microphone.

clear. The car *jumps* whenever such a spike is recorded, rendering it difficult for the user to map the voice level to the cars position.

To resolve this problem, various methods for *moving* the car were attempted. In the following sections these methods will be discussed.

4.3.1 Linear motion

Instead of moving the car directly to its target location, it could be moved there *over time*. This naturally introduces a concept of speed, as to define how much time is required to move the car. Let s denote the vertical speed of the car and t_f the time at frame f. We then introduce the variable $\lambda_f = t_f - t_{f-1}$, representing the time difference between two frames. With this, the position of the car can be represented as:

$$c_f = \begin{cases} c_{f-1} + s \cdot \lambda_f & \text{if } c_{f-1} < m_f \\ c_{f-1} - s \cdot \lambda_f & \text{otherwise} \end{cases}$$

Using this approach the car no longer performs vertical leaps and will steadily move towards its current target.

The aforementioned spikes in volume are still present, as the same input is used. This means that while the car is moving downwards it might move upwards for a few frames before continuing its "correct" motion. The result is a car that jitters while it moves.

It is clear that while this approach fixes the issue of the car jumping from one position to another, there is still no clear mapping between the cars motion and the voice input.

4.3.2 Averaging the measurements

To reduce the effect of the spikes the average of the last x measurements could be used instead of the actual measurements. This would greatly reduce the jitter described above, as a single "up" measurement is outweighed by $x - 1$ "down" measurements (assuming the deviations are similar). Naturally we would still like the benefits of the linear motion (section 4.3.1) to keep the car from *"jumping"*. Let $m_f^{'}$ denote the average of the last x measurements:

$$m_f^{'} = \frac{m_f + m_{f-1} + \cdots + m_{f-x+2} + m_{f-x+1}}{x}$$

We can now define the cars position using this average:

$$c_f = \begin{cases} c_{f-1} + s \cdot \lambda_f & \text{if } c_{f-1} < m_f^{'} \\ c_{f-1} - s \cdot \lambda_f & \text{otherwise} \end{cases}$$

This method handles the spikes better than the strictly linear method. For larger spikes a bigger value of x is required to compensate. The result of this is a significant delay in the effect of raising the volume of ones voice.

4.3.3 Acceleration and deceleration

Another approach to managing the position is by having the car accelerate and decelerate when moving vertically. Spikes in volume would then be met with a deceleration in speed. But as they are often quite short-lived the car would quickly accelerate again. Thus the spike problem is solvable using this methodology. Meanwhile the delay associated with averaging the measurements (section 4.3.2) does not occur as the car always accelerates or decelerates depending on the actual input. Using this method for moving the car can be defined as below:

$$c_f = c_{f-1} + s_f \cdot \lambda_f$$

Where s_f refers to the cars vertical speed at frame f. We then define s_f in a way that involves acceleration and deceleration using s_{f-1} and λ_x as the only variables. For this purpose a function *speed* is introduced, yielding the following equation:

$$c_f = c_{f-1} + \text{speed}(s_{f-1}, \lambda_f) \cdot \lambda_f$$

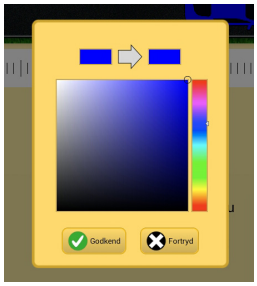
We will not explore the actual function that defines the acceleration of the car in the context of the cars vertical motion as it is the methodology of how the car is moved that is of interest. Allowing the car to accelerate solves the problem caused by spikes in the voice input and gives direct feedback for the user. This is therefore the chosen solution to control the car.

4.4 GUI Components

During this sprint we also implemented the use of the shared GUI components, through the library `giraf-components`. We used two components; `GButton` and `GColorPicker`. Additionally we used the standard background color, provided by the library. The button and colorpicker components can be seen in fig. 4.2.

Background color The main menu and setting activites were changed to use `giraf-component`'s default background color, through the `GetBackgroundColor()` method in the `GComponent` class.

(a) The `GButtons` in Cars main Activity



(b) `GColorPicker` in Cars settings Activity

Figure 4.2: The graphics used in the game

Buttons All menu-related buttons were changed to `GButton`, which overrides the standard Android `Button`. No functionality is any different for the button, only layout.

Colorpicker The colorpicker was used for selecting car/garage colors, as mentioned in section 4.1. `GComponent`'s `GColorPicker` is an extension of `GDialog`, which is an extension of Android's `Dialog`. The initial color of the dialog must be set by its `SetCurrColor(int color)` method. The new color, after choosing one in the dialog, is returned through the `OnOkListener` interface, which provides the method `OnOkClick(GColorPicker diag, int color)`.

4.5 Evaluation

This section contains an overview of the requirements and whether they have been fulfilled during this sprint. Comparing table 3.2 and table 4.1 it is possible to see the progress and what has been solved during sprint 3 compared to sprint 2. Each requirement, if fulfilled, has a link to where it is explained.

#	Requirement	Solved	Link
1.	The game must not be a side-scrolling game, because the citizen must be able to see the goal	√	section 2.5
2.	The car is controlled in such a way, that the vertical position of the car is relative to the current loudness of the player's voice.	√	section 3.3.4
3.	The goal of the game is to reach a garage	√	section 2.5
4.	When the game is won a reward should be given	√	section 3.3.2

5.	It must be possible to save and load settings for a specific profile	√	section 4.2
6.	It must be possible to calibrate the microphone	√	section 4.5.1
7.	There is a digit between 0 and 10 displayed on the car as well as obstacles, representing the loudness level, based on its vertical position.	√	section 3.3.1
8.	Besides the scales from 0 to 10, both speed and loudness have pictograms illustrating some of the values on these scales.	√	section 4.1
9.	It should be possible to pause the game. When the game is paused, a loudness-barometer is displayed next to the car, further visualizing the current loudness.	√	section 3.3.2
10.	Speed is alterable. The speed level is represented as a digit between 0 and 10.	√	section 4.1
11.	The placement and number of obstacles is alterable.	√	section 3.3.5
12.	The placement of obstacles should be in such a way, that it is possible to adapt it to both citizen with tendency to speaking too loud as well as those speaking too low.	√	section 3.3.5
13.	The graphics need to be simple, as some citizens have a low attention span and are easily distracted.	√	section 3.3.3
14.	It should be possible, in settings, to switch between avoiding objects and picking objects up.	×	-
15.	When picking objects up, this is linked to pictogram categories.	×	-
16.	It is important that the pickup/category ”mode” is optional, due to different capabilities for each citizen.	×	-

Table 4.1: Requirements fulfilled after sprint 3.

4.5.1 Control of the car

It was mentioned in section 3.5.2 that the control of the car did not work properly with the current calibration. This is regarding requirement 6. So

first the calibration process was examined but no problems were found. Then the control of the car was examined and a solution was found. The solution to this problem was to improve the control of the car described in section 4.3. So this indirectly solved requirement 6.

4.6 Future Works

This section contain the assignments which should be considered looking at in the next sprint.

In the next sprint the focus should be on looking at the rest of the requirements, but especially the look of the calibration in the settings.

In the next sprint it should be considered to implement a possible setting to pick up objects instead of avoiding them as stated in requirement 14. But this should be discussed with the stakeholders first, so no unnecessary features are added and to clarify the assignment.

Chapter 5

Sprint 4

As suggested in the previous sprint, the main focus of this sprint was to look at the calibration settings. Another focus was on whether it should be possible to pick up objects. This led to an interview with speech therapist Tove Søby, where several questions were answered.

After the interview, the requirements were altered and the changes applied and documented. Picking up objects was implemented as a game mode. The garages were removed, and a finish line was inserted instead. Sounds, to strengthen the auditory sense, were added, for when pressing buttons and picking up objects. Finally, the sprint is evaluated and it was found that all requirements were fulfilled.

5.1 Interview

A semi-structured interview [8] was carried out on the May 8th 2014 with the speech therapist Tove L. Søby. Tove was the contact person who was related to the original cars project [1]. We would have liked to have an interview with Tove much earlier, but communication problems throughout the project period (and a wrong email-address) caused the interview to be in the fourth sprint. A summary of the interview can be found in appendix F.

5.1.1 Purpose

The purpose of this interview was to get answers to some key questions about the fundamental purpose of the game. Tove is a speech therapist and therefore with the closest connection to the application and it is important that the product complies to her requirements.

5.1.2 Planning

Before the execution of the interview, a number of questions and doubts were collected.

Garages One doubt was whether the garages at the end of the track made sense, since the purpose of the game was to avoid obstacles in order to learn the citizen to talk at a specific sound level. The garages forced the citizen to use three levels in every game played.

Avoid/collect Is it preferred that the citizen has to avoid obstacles or is it better that they need to collect items.

Sounds Another question was if using sounds in the game was a good idea.

Specify input method A major doubt from the group prior to the interview was whether the citizen was supposed to produce a continuous sound or if it should be spoken words. This would have a tremendous impact on the way the control of the car would need to work.

Pictograms for buttons The last question was whether the text on the buttons needed to be changed to pictograms in order to make it easier for the citizen to understand.

5.1.3 Result

The result of the interview will be listed here as a list of requirements and changes to the application along with a short description of the reason to the changes.

1. The garages need to be removed and replaced with a simple finishing line
   The garages were removed because they did not fit with the purpose of training citizens particular needs given that every garage needed to be reached.

2. Button texts need to be renamed. “Fortsæt” (continue) will become “Ny tur” (new turn), and “Menu” will become “Færdig” (done).

3. The game objective needs to be to collect items instead of avoiding them. The old game mode can be a setting.
   Picking up object fit better to the purpose of teaching the citizens to hit a certain sound level.

4. Sounds at key events. Car revving at the countdown screen. Sound when collecting/colliding with things. A jingle when the game is won. This was in order to strengthen the citizen's auditory sense.

5. All buttons in the game need to say their text when pressed. This was also to strengthen the auditory sense.

6. The citizen is supposed to say short words when controlling the game such as "na - na - na". It is not supposed to be a continuous sound. This was to mimic the techniques already used by the speech therapist

5.1.4 Requirements

According to the information gathered during the interview the requirements have been updated and is available in table 5.2.

Changes have been made to requirement 3 while requirement 16 and requirement 17 has been added.

The old requirement "When picking objects up, this is linked to pictogram categories." was deleted. Objects that need to be picked up should use icons from a small selection of different icons (or even just one).

#	**Requirement**	**Solved**	**Link**
1.	The game must not be a side-scrolling game, because the citizen must be able to see the goal	√	section 2.5
2.	The car is controlled in such a way, that the vertical position of the car is relative to the current loudness of the player's voice.	√	section 3.3.4
3.	The goal of the game is to reach the finishing line after successfully collecting all items or avoiding all items (two different game modes)	×	-
4.	When the game is won a reward should be given	√	section 3.3.2
5.	It must be possible to save and load settings for a specific profile	√	section 4.2
6.	It must be possible to calibrate the microphone	√	section 4.5.1
7.	There is a digit between 0 and 10 displayed on the car as well as obstacles, representing the loudness level, based on its vertical position.	√	section 3.3.1
8.	Besides the scales from 0 to 10, both speed and loudness have pictograms illustrating some of the values on these scales.	√	section 4.1

9.	It should be possible to pause the game. When the game is paused, a loudness-barometer is displayed next to the car, further visualizing the current loudness.	√	section 3.3.2
10.	Speed is alterable. The speed level is represented as a digit between 0 and 10.	√	section 4.1
11.	The placement and number of obstacles is alterable.	√	section 3.3.5
12.	The placement of obstacles should be in such a way, that it is possible to adapt it to both citizen with tendency to speaking too loud as well as those speaking too low.	√	section 3.3.5
13.	The graphics need to be simple, as some citizens have a low attention span and are easily distracted.	√	section 3.3.3
14.	It should be possible, in settings, to switch between avoiding objects and picking objects up.	×	-
15.	It is important that the pickup/category ”mode” is optional, due to different capabilities for each citizen.	×	-
16.	The game must use sounds at key events, such as Car revving at the countdown screen. Sound when collecting/colliding with things. A jingle when the game is won.	×	-
17.	All buttons need to read their text when pressed.	×	-

Table 5.1: Requirements fulfilled after sprint 3.

5.2 Design and Implementation

After the interview with Tove there was some changes to be implemented. The new requirements are mentioned in table 5.2 and the changes made to satisfy these requirements will be described in this section. Additionally, due to a change in the way the active profile is changed, a new profile-selector button was added to the main menu.

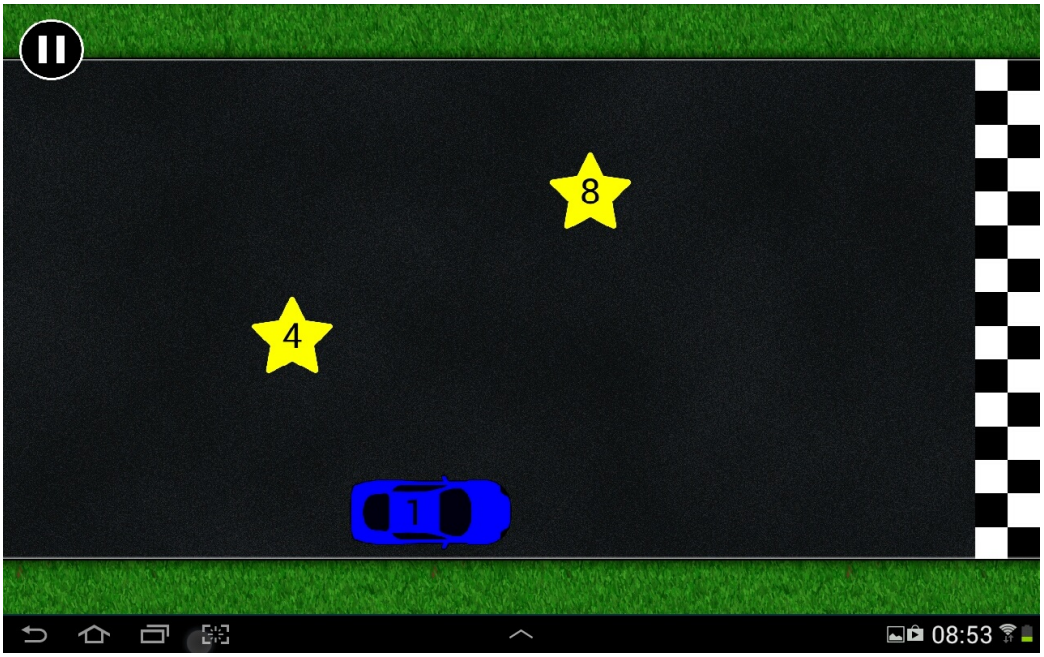


Figure 5.1: The game with a finishing line as goal

5.2.1 Game objective

According to requirement 3 (see table 5.2) the garages had to be removed from the game and replaced with a simple finishing line. The game also had to be changed so the items on the road had to be collected instead of avoided. These changes introduced some changes to the logic on how to determine when the player wins. Now it is necessary for the player to collect all items before reaching the finishing line. The new appearance can be seen on fig. 5.1, where the new star model for items is also depicted. This new model was introduced to be able to distinguish the two game modes from each other. Choosing between the two modes was implemented in settings as seen on fig. 5.2a

Because of the change away from garages it is now only necessary to choose one color for the game. This change was reflected in settings by only showing one colorpicker and shaping this like a car. This car changes color according to the picked color and can be seen on fig. 5.2b

The calibration fragment was changed slightly to blend more into the other components of the settings screen. A text was added to make the usage of the fragment more clear for the users. The changed fragment as well as the complete settings screen can be seen on fig. 5.3

(a) Choosing gamemode

(b) Changing the color of the car

Figure 5.2: Changes in settings

Figure 5.3: The complete settings window

(a) The old appearance of the button **(b)** The new appearance of the button

Figure 5.4: Changes in appearance of buttons

5.2.2 Buttons and sounds

Another requirement that introduced changes to the application was requirement 17. All buttons now have to say their text when pressed and events need to have a sounds associated.

It was also a request to make the buttons look more like buttons visually. The changes to the appearance can be seen on fig. 5.4, where fig. 5.4a shows the old appearance (where a button was only the text) and fig. 5.4b shows the new appearance (where a border and background color has been added), making the button resemble a traditional button more.

5.2.3 ProfileSelector

As of this sprint, `giraf-component` library now provides a `GButtonProfileSelect` class, to overcome a change in how profiles are switched. Previously, a profile was selected in launcher, whenever an app is opened. Now launcher has a currently active profile, which is the one provided when an app is opened. Due to this, all apps must now provide an option of selecting another profile, using `GButtonProfileSelect`.

5.2.4 Evaluation

This section will contain an evaluation on the fourth sprint, and how the requirements that changed after the interview (section 5.1) have been fulfilled.

#	**Requirement**	**Solved**	**Link**
1.	The game must not be a side-scrolling game, because the citizen must be able to see the goal	√	section 2.5
2.	The car is controlled in such a way, that the vertical position of the car is relative to the current loudness of the player's voice.	√	section 3.3.4

3.	The goal of the game is to reach the finishing line after successfully collecting all items or avoiding all items (two different game modes)	√	section 5.2.1
4.	When the game is won a reward should be given	√	section 3.3.2
5.	It must be possible to save and load settings for a specific profile	√	section 4.2
6.	It must be possible to calibrate the microphone	√	section 4.5.1
7.	There is a digit between 0 and 10 displayed on the car as well as obstacles, representing the loudness level, based on its vertical position.	√	section 3.3.1
8.	Besides the scales from 0 to 10, both speed and loudness have pictograms illustrating some of the values on these scales.	√	section 4.1
9.	It should be possible to pause the game. When the game is paused, a loudness-barometer is displayed next to the car, further visualizing the current loudness.	√	section 3.3.2
10.	Speed is alterable. The speed level is represented as a digit between 0 and 10.	√	section 4.1
11.	The placement and number of obstacles is alterable.	√	section 3.3.5
12.	The placement of obstacles should be in such a way, that it is possible to adapt it to both citizen with tendency to speaking too loud as well as those speaking too low.	√	section 3.3.5
13.	The graphics need to be simple, as some citizens have a low attention span and are easily distracted.	√	section 3.3.3
14.	It should be possible, in settings, to switch between avoiding objects and picking objects up.	√	section 4.1
15.	It is important that the pickup/category ”mode” is optional, due to different capabilities for each citizen.	√	section 4.1
16.	The game must use sounds at key events, such as Car revving at the countdown screen. Sound when collecting/colliding with things. A jingle when the game is won.	√	section 5.2.2

17.	All buttons need to read their text when pressed.	$\surd$	section 5.2.2

Table 5.2: Requirements fulfilled after sprint 4.

5.3 Future Works

As it can be seen on table 5.2 we have fulfilled all the formal requirements gathered in the four sprints. There is nevertheless still some things that can be improved on. These things will be discussed in the following.

Calibration The calibration fragment, used to calibrate the microphone, works as it is now, but its design is not as good as we would like. One suggestion has been to make it more like a wizard, where the fragment helps the user through the process, instead of having all the explaining text to the left of the fragment.

Layout in settings It was attempted to keep all settings on one screen to avoid the need to scroll through it This was in the end achieved as can be seen on fig. 5.3. Unfortunately it does not fit as well on smaller screens, and even though the institution use the 10 inch format as displayed in fig. 5.3, it would be a better solution if the layout would fit on all screens.

Chapter 6

Common Game Framework

The cars project has been built on an existing framework as mentioned shortly in section 2.4. This made the creation of the game easier because the basic problems of game development on the android platform were already taken care of. It was therefore relatively painless to get to the actual design of the game.

Some project proposals at the beginning of the semester suggested making more games for the citizens. It is therefore ideal to create a separate project that contains the framework so others can use it in future projects. Our modified version of the framework has been moved to a new repository for this to be possible. This repository can be a base for the development of future games in the GIRAF project.

In this chapter the usage of the framework will be described in order to make it easier for the students from next semester to create games in it.

6.1 Using the framework

The game framework[1] is built around using screens to encapsulate different game-states.

In Cars there is used a couple of different screens. The main game is in one screen, and the possible events, such as car crashed, game paused and game won, are contained in a screen each. This encapsulation makes it very easy to keep an overview over the game even if there are a lot of states.

[1] It can be found in the repository `game-framework` – http://cs-cust06-int.cs.aau.dk/git-ro/game-framework/

6.1.1 The GameActivity

The game is contained in an Android activity which holds the logic for switching between screens. An activity class that extends the `GameActivity` class from the framework needs to be created. This class will keep a reference to all screens and exposes a method, that makes it possible for any currently active screen to switch to another screen. This is done by using the `setScreen(Screen screen)` method from the `GameActivity` superclass.

The method `getInitScreen()` needs to be overridden in order to tell the activity which screen to show as the initial screen.

6.1.2 Screens

All screens need to override their `paint` and `update` methods, as these methods are continuously called while the screen is active. The `paint` method carries a reference to the `Graphics` object (see the following section). The `update` method carries a reference to `Input.TouchEvent[]` (see section 6.1.7). Both methods also have a floating point number `deltaTime` (see section 6.1.8).

All screens are created with a reference to the `GameActivity` that created it, making available the method `setScreen`, which changes the currently active screen.

Additionally, every object that needs to be updated, needs to implement the `Updatable` interface. This provides the concerned object with an `Update` method, which needs to be called in the concerned screen's `update` method for the object's logic to be updated. The same pattern applies to any object that needs to be drawn, but with the `Drawable` interface and the screen's `paint` method.

6.1.3 The Graphics Class

Drawing on the screen is handled by the `Graphics` class in the framework. This class exposes methods to draw basic shapes, as well as text and images. The screen's `paint` function carries a reference to the `Graphics` object, making available all functions needed for drawing onto a screen.

6.1.4 The Audio Class

Music and sounds are handled by the `Audio` class. It is responsible for loading sound-files into `Sound` and `Music` objects, where after these can be used to play the loaded music/sounds.

6.1.5 Loading assets

Images and audio need to be loaded before they can be displayed in the game. Images are loaded into `Image` objects, using `Graphic`'s `newImage` method. Audio is loaded using `Audio`'s `createMusic` or `createSound` methods.

6.1.6 Playing sounds

The framework contains a class used for playing sounds in the game. This class contains the methods `Play`, `Reset` and `PlayAndReset`. The sound class contains a boolean value representing if it has been played or not. This value is set to true every time `Play` is run and set to false every time `Reset` is run. This ensures that the sound is only played once, and that it can first be played again when needed.

6.1.7 Handling touch events

In order to handle input on the screen the framework contains a `SingleTouchHandler` class. This class handles input events and is sent to the `update` method of the screen as an array of `InputEvent`. Each `InputEvent` contains x and y coordinates; where the touch occured, along with its type; `TOUCH_DOWN, TOUCH_UP, TOUCH_DRAGGED` or `TOUCH_HOLD`.

6.1.8 Deltatime

Screens are also provided a floating point number `deltaTime`, which is based on the current system time in milliseconds. This makes it possible to determine how much time has gone by between frame-updates (as this won't always be the same). This makes it possible to make smooth animations over the course of time, despite having slightly time-differences between frames. It can also be used to time events, such as a countdown-timer.

Chapter 7

Git

The following chapter regards Git, the server system Gitolite and the use of the two. Thus a basic knowledge of Git is expected of the reader.

The code-base for the various applications and sub-projects, that are part of the GIRAF project, is quite big and requires that multiple people (possibly from different groups) can work on the same project at once. Thus a content management system is required to manage the code and its various revisions. It was unanimously decided by all project groups to use Git as this content management system. One of the reasons for using Git was that that an administration tool had already been installed on the GIRAF server. In section 7.1 this tool is described along with a guide to its usage.

Some of the students partaking in the project were new to the use of Git. Because of this a *"git specialist"* was chosen. This specialist was a member of the group developing *Cars*.

7.1 Gitolite

Gitolite is an administration tool for git repositories. Interaction with Gitolite is done through a git repository in which all repos are defined in one or more configuration files. Using hooks, Gitolite will create repositories when new ones are defined. The syntax of these configuration files is described in section 7.1.1.

Accessing the server Each GIRAF repository is exposed through two urls. One is read-only and is publicly available. This allows projects to have other repositories as submodules, but without having write-access. It also simplifies the server's automated build, as no authentication is required.

The second url is one, to which access is restricted (see section 7.1.1). The two urls are as follows:

`http://cs-cust06-int.cs.aau.dk/git-ro/` (read-only)
`http://cs-cust06-int.cs.aau.dk/git/` (read-write)

Each url lists a collection of repositories. The urls for a repo is equal to appending the repo name to either of the above urls. Connecting to these repositories can be done using LDAP authentication. This means that the standard @student.aau.dk logins used for AAUs systems will apply to Gitolite.

7.1.1 Configuration files

As mentioned above, repositories in Gitolite are configured through a Git repository. This repository can be found at the url below:

`http://cs-cust06-int.cs.aau.dk/git/gitolite-admin/`

In the `conf` directory in the repository, a collection of files define all the GIRAF repositories. To each repository a set of access rules apply. These are described throughout the following section. Note, however, that the access rules described below are only those applied in the course of the 2014 spring semester. Gitolite has an online manual[1] for its usage to which one should refer for any additional information.

7.1.1.1 Users and groups

In addition to defining repositories, Gitolite allows the definition of groups. A group can be either a group of repositories or a group of users. In the context of GIRAF only the latter is applied. In the two files `sw6f13-groups.conf` and `sw6f14-groups.conf` the project groups for each semester are defined. This scheme could be repeated for other semesters. A group is defined by prefixing a name with a @ symbol, as below:

```
@sw600f14 = user1@student.aau.dk user2@student.aau.dk
```

This creates a group called `@sw600f14` with two members. The use of AAU email addresses allows the usernames to correspond directly to the LDAP authenticated users. A group can also consist of other groups, such that a group can be created from the list of all semester groups.

[1] `http://gitolite.com/gitolite/master-toc.html`

7.1.1.2 Multiple configuration files

The `.conf` file loaded by Gitolite is the `gitolite.conf`. As the list of repositories grow, multiple configuration files can be created and included. To include a configuration file *repos.conf* Gitolite has an `include` command:

```
include "repos.conf"
```

This command can also be used in configuration files included using the command, and is useful when trying to separate various types of repositories.

7.1.1.3 Defining a repository

Defining a new repository in Gitolite is very simple. It must simply be defined, staged, commited and pushed (as any other content in git). When pushed to the server Gitolite will create the new repository. The following command defines a repository name newrepo:

```
repo newrepo
```

Note that this definition does not explicitly tell Gitolite to create the new repository, just that it should exist. Thus, the same repo can be defined multiple places with different sets of access rules. This allows for separating different types of access rules into different files (for instance a separate config file can hold any temporary access rules).

7.1.1.4 Access rules

Below in code example 7.1 is the definition of a fictitious repository `component`. Using this as an example, the various types of access rules will be described. There are four[2] types of access available in Gitolite. The terms *push* and *write* are used interchangeably, as is *pull*, *fetch* and *read*. The four access types are:

R Allows the users listed to read from the repository, but not write.

RW Allows the users listed to read from and write to the repository.

RW+ Similar to **RW**, but also allows deletion. This access type allows removal of remote branches and forced push. In the context of GIRAF, this type of access has been reserved for the git specialist.

[2]There are additional advanced access types, but these have not been used for the GIRAF project

- (dash) The users listed are not allowed to read from or write to the repository. This can be used to cancel parts of broadly defined access rights.

```
1 repo component
2     R   = @all
3     RW  = username1 username2
4     RW+ = @admin
5     RW  VREF/NAME/dir1/ = username1
6     RW  VREF/NAME/dir2/file = username1
7     -   VREF/NAME/ = username1
8     RW  branchname = @fixers
```

Code example 7.1: A Gitolite configuration file

In code example 7.1 we see examples of all the access types. Below is a list of how each of the lines in the example should be read.

- Everybody can read from the repository. The `@all` usergroup is predefined by Gitolite to be all users or all repositories, depending on the context.
- `username1` and `username2` have read and write access to the repository.
- Administrators can delete information from the repository. The `@admin` is not predefined by Gitolite.
- `username1` has read and write access to the directory `/dir1/` and the file `/dir2/file`, but does not have access to any other files. Notice that line 3 is required for `username1` to be able to write to the repository at all. The `VREF` rule is checked when the actual push is happening. There are additional rules available to Gitolite such as only allowing users to push a certain number of times or only at specific times of day.
- The `@fixers` usergroup can read and write to the `branchname` branch in the repo.

Note that the first rule specifies that everyone can read from the repository. This in turn means that the remaining rules only restrict write access to the repository.

7.2 Common Git difficulties

Git is a quite flexible tool for managing code. One of the reasons for this is the number of commands provided by Git. This was the main cause of problems with the usage of Git, as many of the student working on the GIRAF project were new to Git. Thus the git specialist spent a lot of time supporting other groups in the efforts to use Git. This included, in part, an introductory presentation to Git early in the semester. The following chapter will describe some of the typical issues with Git encountered during the project.

Git on Windows As many student were new to the use Git (and some to the use of a command-line tool) a guide for setting up Git on Windows was created by the Git specialist. This guide is include in appendix B on page 69.

7.2.1 Staging content

As most users had previously only been exposed to the SVN content management system, there were some confusion as to Gits additional *staging* and *pushing* steps in the commit process. Where SVN copies all changes to files from your local machine to a server in one step, Git allows local staging of content, local storing of commit before eventually pushing to the server.

This difference was not properly understood by many of the groups, as they continued in the SVN workflow of always sharing each change (in order to keep commits small), or only committed seldom yielding large commits that were harder to merge with the remaining group members.

Another problem that arose from this was the use of the `git add .` command, which stages all content in the current directory. This command (without the use of `git status` and `git diff`) renders the user blind to what they are actually committing. The result was that often undesired code wash pushed to the shared server. A special type of problem resulting from this misuse of Git is explained in section 7.2.3.1.

It is the view of the Git specialist that more time should be spent discussing the proper use of Git, in order to take advantage of the possibilities of git and avoid many of the problems associated with Git.

7.2.2 Branching and merging

The understanding of the difference between a local branch and remote branch was hard to grasp for many students. Similar to the problems described above, the typical issue was that a true copy of a centralized server

was expected - as is the case with SVN. The concept of what is stored locally and what is stored remotely was not properly explained and/or understood. The result of this was that many branches intended as local-only were pushed to the server and consequently created by others working on the same project.

Similarly, the idea of *"parallel"* work on multiple local branches seemed a scary thought to many. Some simply kept to the linear world they were accustomed to.

The confusion that arose from the multiple branches never lead to big problems, but mainly to insecurity due to the complexity of Git. Thus a lot of time was spent by the Git specialist in an effort to clarify these difficulties.

7.2.3 Submodules

The greatest cause for difficulty regarding Git was the use of submodules. In Git submodules represent the inclusion of one repository inside another. Using this kind of structure allows the nesting of shared components (such as `OasisLib` and `giraf-component`) inside other projects. Git will then allow its users to work with these components as separate repositories while tracking which version of the repositories are in use. The benefit of this tracking is that when a new version of these components is created, each project can upgrade to the newest version in its own pace.

Many groups had difficulties understanding how the submodules in Git worked. Asides from having to assist in the adding and removal of submodules, the Git specialist encountered two general problems in regards to Git submodules. These problems will be explained in the following two sections.

7.2.3.1 Referring local changes

As mentioned above, a Git submodule references the currently used version of each of its submodules. This is done by continuously updating a reference to the SHA of the used submodule commit. As in the example below:

> **Example**
> Let **A** and **B** be repositories, where **B** is a submodule in **A**. **A** keeps track of which version of **B** is in use by **A** by registering the SHA `2b8c053`[3]
>
> When a new commit 551b569 is created in **B**, **A** does not automatically use is. Only when a commit in **A** updates the SHA of **B** to 551b569, **A** will use the new commit.

[3]Git SHAs are 40 character strings, abbreviations are used in the examples.

This allows for content to be added to **B** independently of **A**, parallelizing the process.

As a commit in Git is only a local operation, nothing is shared with the remaining project groups before the changes are pushed to a server (or another member of the project group). If only the changes in **A** are pushed, then anyone trying to use this newest version of **A** will not be able to use the correct version of **B** as it only exists locally on one machine.

This problem has arisen, as members of some groups have committed content in repositories to which they have no write access. They are then not able to push their changes but only their reference to the content that they can not push. In particular this problem exists due to the wrongful use of `git add .` where all content is blindly staged without concern for what content should actually be committed.

7.2.3.2 Deep clone

The GIRAF repository `OasisLib` is referenced by multiple project. Amongst those are both `giraf-component` and `cars`. As the latter also includes `giraf-component` this infers a duplicate inclusion of `OasisLib` in `cars`. The situation has been illustrated in fig. 7.1. The figure illustrates the submodules

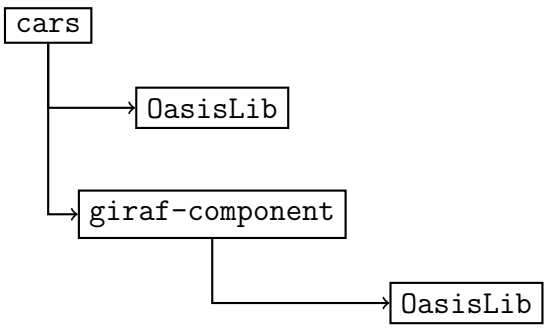


Figure 7.1: Deep clone of the `cars` repository.

associated with the `cars` repository (there were however other repositories with a much larger tree of submodules). In the development environment only the topmost level of submodules were used and thus the remaining were not to be cloned. However many groups performed a *deep clone* in which they cloned the entire structure. This is not directly a problem, but the cause of much confusion. Combined with the problems described in section 7.2.3.1 this caused quite a few problems for the groups with many submodules associated with their project. These problems were all attended to by the git specialist.

Chapter 8

Reflections

When we started the semester, we had a big planning meeting with all students present. Here we decided the overall method and structure of the project. We based our decisions on the experiences of those from the previous year (who in turn had also used the experiences of the year before that).

However, we did still have some things that could have been handled better, or remained problematic but without a better solution. We will present these problems, and possible solutions, so that next year's multi-project students can use our experience.

Early meetings with stakeholders is something we wish we had, but did not. We had our first meeting at the end of the second sprint and our second meeting in the beginning of the 4th sprint. This resulted in us establishing requirements way too late, and having to do several changes throughout the entire semester, as requirements changed drastically as a result of meetings with the stakeholders.

It would instead be a good idea to get a good overview of the project(s) and early on clarify any confusions, by meeting with the stakeholders.

Plan sprint-reviews early in the semester. Our first sprint-review had only 2 stakeholders present, and they had to leave early. This was due to them being invited only shortly before the sprint-review.

It would be a good idea to establish sprint-reviews early on and invite stakeholders as soon as these are planned. Additionally, depending on response, it could be a good idea to move sprint-reviews to fit the needs of the stakeholders, as it is really important that they can make the meeting, rather than every single person from the project-groups.

Requirements was something that was determined during the first sprint by 2 groups, however, these were mostly too general and therefore not very useful. Also, these requirements were not maintained or updated throughout the semester. Instead, informal rules formed as a result of status meetings, individual group's meetings with stakeholders, and from sprint-reviews.

It would be a good idea to establish concrete requirements early on, and then update these as new information is gained throughout the project.

Status meetings need to be short and to the point. At first our status meetings were long and, for some, pointless. This was due to everything being discussed by all present, even though some subjects only concerned some groups/individuals.

In the end the meetings worked a lot better, as they were very short, consisting of a short status report from each group, and if there were any problems, the concerned parties would meet or plan a meeting after the status meeting.

Working independently from GIRAF As cars is a game and does not use the database containing pictograms we were not depending as much on the overall project as most of the other groups. This made it easier to progress without disturbances, and this could also be seen at the sprint reviews as we were one of the few groups that had something to show at all four sprint reviews. On the other side it made it harder to have an influence on the project, because we did not have a direct dependence on the matters that were discussed on the status meetings.

Redmine Forums was meant as a source of discussions on non-pressing matters. It did however not work well for this.

Firstly, the forum itself works poorly. The layout is not very good, and its functionality very limited. A better way of viewing what is new since last time you checked is dearly needed.

Secondly, for it to work, people need to check the forums regularly, which far from everyone did.

Another option would be to activate notifications by email.

Issue-tracking is something that never quite had the effect that we wished. The intention was that everyone had their issue-tracker updated, so that it was possible to see what each group was currently working with. This is hard due to the different levels of details, and in some cases complete absence, of the issues.

Another problem, which we had, is that we also like to use a real-life scrum-board, with post-its as issues. This made it necessary to have both Redmine Issues as well as real-life tasks synchronized, which was a tedious task.

Report guidelines were a confusing matter throughout the entire semester. One thing we should have considered from the start, is the fact that they are only guidelines, not rules.

We thought the report structure described in the guidelines as confusing and far from what we were used to, so we had a hard time adapting to this.

Part I

Appendix

Appendix A

Code example: Run method from class RecorderThread

```
1  @Override
2  public void run() {
3
4  System.out.println("recorderThread started");
5  AudioRecord recorder;
6  int p;
7  short[] audioData;
8  int bufferSize;
9  int Samplerate = 44100;
10 bufferSize=AudioRecord.getMinBufferSize(Samplerate
    ,AudioFormat.CHANNEL_IN_MONO,
11 AudioFormat.ENCODING_PCM_16BIT)*2; //get the
    buffer size to use with this audio record
12
13 recorder = new AudioRecord (AudioSource.MIC,
    Samplerate,AudioFormat.CHANNEL_IN_MONO,
14 AudioFormat.ENCODING_PCM_16BIT,bufferSize); //
    instantiate the AudioRecorder
15
16
17 recording=true; //variable to use start or stop
    recording
18 audioData = new short [bufferSize]; //short array
    that pcm data is put into.
19
```

```
20 while (recording) {  //loop while recording is
    needed
21     if (recorder.getState()==android.media.
        AudioRecord.STATE_INITIALIZED){ // check to
        see if the recorder has initialized yet.
22     }
23     if (recorder.getRecordingState()==android.
        media.AudioRecord.RECORDSTATE_STOPPED){
24     recorder.startRecording();  //check to see if
        the Recorder has stopped or is not recording,
         and make it record.
25     }
26     else {
27     recorder.read(audioData,0,bufferSize); //read
        the PCM audio data into the audioData array
28     double[] endAudioData = new double [bufferSize
        *2];
29     //Now we need to decode the PCM data using the
         Zero Crossings Method
30     //FFT fft1 = new FFT();
31     short[] imgAudioData = new short [bufferSize];
32     endAudioData = FFT.fft(audioData,imgAudioData,
        true);
33
34     double[] magnitude = new double [bufferSize
        -1];
35     int highestMagnitude = 0;
36     for (p=2;p<(bufferSize-1)*2;p+=2){
37     magnitude[p/2-1] = Math.sqrt(endAudioData[p]*
        endAudioData[p] + endAudioData[p+1]*
        endAudioData[p+1]);
38     //System.out.println("magnitude = " +
        magnitude[p/2-1]);
39     if (magnitude[p/2-1] > magnitude[
        highestMagnitude]){
40     highestMagnitude = p/2-1;
41     }
42     }
43     int i;
44     double[] frequency = new double [bufferSize
        -1];
```

```
45      for (i=0 ; i<(bufferSize-1) ; i++){
46      frequency[i] = ((i+1) * Samplerate/2) / (
         bufferSize/2);
47      }
48
49      double total = 0;
50      double magnitudeTotal = 0;
51      int start;
52      if (highestMagnitude > 2) {
53      start = -2;
54      } else {
55      start = 0;
56      }
57      int end;
58      if (highestMagnitude + 2 < frequency.length) {
59      end = 2;
60      } else {
61      end = 0;
62      }
63      for (i=start ; i<end ; i++){
64      total += frequency[highestMagnitude+i]*
         magnitude[highestMagnitude+i];
65      magnitudeTotal += magnitude[highestMagnitude+i
         ];
66      }
67      double averageFreq = total/magnitudeTotal;
68      if (averageFreq<=highestHumanPitch &&
         magnitudeTotal>voiceSensetivity) {
69      currentFrequency = (int) averageFreq;
70      //System.out.println("average frequency = " +
         (int)averageFreq + " magnitude total = " + (
         int)magnitudeTotal);
71      } else {
72      currentFrequency = 0;
73      }
74      GameInfo.setCurrFreq(currentFrequency);
75      }//else recorder started
76      } //while recording
77
78      if (recorder.getState()== android.media.
         AudioRecord.RECORDSTATE_RECORDING) recorder.
```

```
        stop(); //stop the recorder before ending the
         thread
79      recorder.release(); //release the recorders
         resources
80      recorder=null; //set the recorder to be
         garbage collected.
81  }//run
```

Code example A.1: Run method from the original Cars project.

Appendix B

Git guide for Windows

The following section describes the setup of Git on a Windows system. This guide was written by the Git specialist and is included for additional reference on the use of Git. The guide was originally written in danish and has not been translated for this report, as it is not an essential part of this report.

B.1 Installation af Git

Git hentes på følgende url: `http://git-scm.com/download/win`.

Under installationen kan man vælge at installere Git Bash som terminal. Det vil jeg personligt ikke anbefale, da der findes en extension til PowerShell der laver *magi* og gør verden (på Windows) til et bedre sted. Opsætning af dette er forklaret i B.2 Opsætning af PowerShell. Man kan selvfølgelig også vælge at benytte en GUI til Git, omend det svækker mængden af kontrol man som bruger har over Git.

Test af Git installation Start den terminal du har mest lyst til at anvende (cmd.exe kan til nød anvendes her). Tjek herefter om git er blevet korrekt installeret og at git er tilføjet til din path ved at køre:

```
1 git --version
```

Terminalen skulle gerne svare med din version af git (fx `git version 1.8.3.msysgit.0`). Hvis det ikke sker skal du tjekke at stien til git.exe (fx C:\Program Files (x86)\Git\cmd\) er i din path[1].

[1] Se evt `http://www.computerhope.com/issues/ch000549.htm`

B.2 Opsætning af PowerShell

PowerShell skal have rettigheder til at hente informationer som en del af følgende opsætning. Derfor skal `ExecutionPolicy` ændres fra default (`Restricted`) til `RemoteSigned`. Start PowerShell (som administrator) og udfør følgende kommando:

```
1 Set-ExecutionPolicy RemoteSigned
```

Naviger herefter til en mappe hvor du gerne vil at Git gemmer posh-git. Når du har gjort dette kan posh-git hentes og installeres. Det gøres ved at ved at køre følgende kommandoer i PowerShell:

```
1 git clone https://github.com/dahlbyk/posh-git.git
2 cd posh-git
3 .\install.ps1
```

Følg herefter installationens vejledninger.

B.3 ConEmu

Console **Emu**lator er en wrapper til din konsol (uanset hvilken konsol du anvender) der gør din Windows konsol til alt den den skulle have været. Blandt funktionerne kan nævnes transparency, fullscreen, global hotkeys, tabs og quake mode. Du finder ConEmu på `http://sourceforge.net/projects/conemu/`.

ConEmu skal ikke installeres, men giver blot en `.7z` med executables. Når ConEmu er startet trykkes `Win+Alt+P` for at åbne settings. Alle indstillingsmuligheder vil ikke blive forklaret her - det er der ganske enkelt for mange muligheder til.

B.4 Ingen indtastning af username/password

Til sidst er her en vejledning i hvordan du slipper for at indtaste email og adgangskode hver gang du laver `push` eller `pull`.

Hent den seneste version af *Windows Credential Store for Git*[2] og start den. Det skulle gerne være nok. Hvis konsollen (som programmet starter) bliver hængende, så skal du i din terminal navigere til den mappe du har gemt programmet i og køre:

[2] `http://gitcredentialstore.codeplex.com/releases/view/103679`

```
git-credential-winstore -i "C:\Program Files (x86)
  \Git\cmd\git.exe"
```

Husk at rette `C:\Program Files (x86)\Git\cmd\git.exe` til stien hvor git.exe er installeret.

Appendix C

Code example: Car class from project 'Cars'

```
1  package dk.aau.cs.giraf.cars.gamecode.GameObjects;
2
3  import java.util.ArrayList;
4  import java.util.Random;
5
6  import javax.microedition.khronos.opengles.GL10;
7
8  import android.graphics.Color;
9  import android.graphics.Point;
10 import android.graphics.Rect;
11 import dk.aau.cs.giraf.cars.R;
12 import dk.aau.cs.giraf.cars.gamecode.GameInfo;
13 import dk.aau.cs.giraf.cars.gamecode.GameObject;
14 import dk.aau.cs.giraf.cars.gamecode.GameRenderer;
15 import dk.aau.cs.giraf.cars.gamecode.IDrawable;
16 import dk.aau.cs.giraf.cars.gamecode.IWorkable;
17 import dk.aau.cs.giraf.cars.gamecode.MapDivider;
18
19 public class Car extends GameObject implements
    IWorkable, IDrawable {
20 protected float xOffset = -MapDivider.
    obstacleWidth;
21 protected int yOffset = 0;
22 private final int mLowFreq;
23 private final int mHighFreq;
```

```
24 Point[] collisionBox;
25 private boolean updateCarCollisionBox = true;
26 private float carSpeedAsFloat;
27 public float carScaling = 0.7F;
28 private int halfObstacleHeight = (int)(MapDivider.
    obstacleHeight * carScaling) / 2;
29 private int[] colors;
30 private int[] bitmapIds;
31 private boolean[] closedColors;
32 private int currentColor;
33
34 public Car(int y, float carSpeed, int[] colors,
    int[] bitmapIds) {
35     carSpeedAsFloat = carSpeed;
36     this.yOffset = y;
37     mLowFreq = GameInfo.getLowFreq();
38     mHighFreq = GameInfo.getHighFreq();
39
40     collisionBox = new Point[4];
41     collisionBox[0] = new Point(0,0);
42     collisionBox[1] = new Point(0,0);
43     collisionBox[2] = new Point(0,0);
44     collisionBox[3] = new Point(0,0);
45
46     this.colors = colors;
47     this.bitmapIds = bitmapIds;
48     if (colors != null) {
49         closedColors = new boolean[colors.length];
50         Random rand = new Random();
51         currentColor = rand.nextInt(colors.length)
            ;
52     } else {
53         currentColor = 0;
54         this.colors = new int[] {Color.WHITE};
55     }
56 }
57
58 @Override
59 public void draw(GL10 gl, GameRenderer
    spriteBatcher) {
60     if (colors[currentColor] != Color.WHITE) {
```

```
61          spriteBatcher.draw(gl, bitmapIds[
             currentColor], new Rect(0, 0, 898, 348),
             new Rect( (int)xOffset, yOffset -
             halfObstacleHeight, (int)(MapDivider.
             obstacleWidth * carScaling) + (int)
             xOffset, yOffset + halfObstacleHeight));
62      }
63      else {
64          spriteBatcher.draw(gl, R.drawable.car, new
              Rect(0, 0, 898, 348), new Rect( (int)
             xOffset, yOffset - halfObstacleHeight, (
             int)(MapDivider.obstacleWidth *
             carScaling) + (int)xOffset, yOffset +
             halfObstacleHeight));
65      }
66  }
67
68  @Override
69  public void performWork() {
70      updateCarCollisionBox = true;
71      if (GameInfo.win == false && GameInfo.
         garageClosing == false && GameInfo.pause ==
         false){
72          xOffset = xOffset + carSpeedAsFloat;
73      }
74      int currFreq = GameInfo.getCurrFreq();
75
76      if (currFreq > 0 && xOffset > -(MapDivider.
         obstacleWidth / 2)) {
77          if (currFreq > mHighFreq && yOffset -
             halfObstacleHeight > MapDivider.mapYStart
             ) {
78              yOffset -= 2;
79          } else if (currFreq < mLowFreq && yOffset
             + halfObstacleHeight < MapDivider.mapYEnd
             ) {
80              yOffset += 2;
81              }
82      } else {
83          int closestLane = 0;
```

```
84          for (int i = 1; i < MapDivider.lanes; i++)
              {
85              if (Math.abs(MapDivider.laneCenters[i]
                  - yOffset) <
86              Math.abs(MapDivider.laneCenters[
                 closestLane] - yOffset)) {
87              closestLane = i;
88              }
89          }
90          int offset = MapDivider.laneCenters[
             closestLane] - yOffset;
91          if (offset != 0) {
92              if (offset > 0) {
93                  yOffset++;
94              } else if (offset < 0) {
95                  yOffset--;
96              }
97          }
98      }
99  }
100
101 public boolean CalculateCollisions(Point[] form) {
102     if (updateCarCollisionBox) {
103             int widthScaled = (int)(MapDivider.
                 obstacleWidth * carScaling);
104
105         collisionBox[0].x = (int)xOffset;
106         collisionBox[0].y = yOffset -
             halfObstacleHeight;
107         collisionBox[1].x = (int)xOffset;
108         collisionBox[1].y = yOffset +
             halfObstacleHeight;
109         collisionBox[2].x = widthScaled + (int)
             xOffset;
110         collisionBox[2].y = yOffset +
             halfObstacleHeight;
111         collisionBox[3].x = widthScaled + (int)
             xOffset;
112         collisionBox[3].y = yOffset -
             halfObstacleHeight;
113
```

```
114             updateCarCollisionBox = false;
115         }
116         if (collisionBox[0].y > form[2].y ||
             collisionBox[1].y < form[0].y) {
117             return false;
118         }
119         if (collisionBox[0].x > form[3].x ||
             collisionBox[2].x < form[1].x){
120             return false;
121         }
122         for (int i = 0; i < 3; i++) {
123             for (int j = 0; j < 3; j++) {
124                 if (doesLineCrossLine(collisionBox[i],
                      collisionBox[(i + 1) % 4],
125                             form[j], form[(j + 1) % 4]))
                              {
126                     return true;
127                 }
128             }
129         }
130         return false;
131     }
132     private boolean doesLineCrossLine(Point line1Start
        , Point line1End,
133                                       Point line2Start
                                           , Point
                                           line2End) {
134         float crossProduct1 = crossProduct(line1Start,
            line1End, line2Start);
135         float crossProduct2 = crossProduct(line1Start,
            line1End, line2End);
136         if ((crossProduct1 < 0 && crossProduct2 < 0)
             ||
137                 (crossProduct1 > 0 && crossProduct2 >
                     0)) {
138             return false;
139         }
140         else {
141             float crossProduct3 = crossProduct(
                 line2Start, line2End, line1Start);
```

```
142         float crossProduct4 = crossProduct(
             line2Start, line2End, line1End);
143
144         if ((crossProduct3 < 0 && crossProduct4 <
             0) ||
145                 (crossProduct3 > 0 &&
                  crossProduct4 > 0)) {
146             return false;
147         }
148     }
149     return true;
150 }
151 private float crossProduct(Point line1Start, Point
      line1End,
152                            Point line2Point) {
153     return (line1End.x - line1Start.x) * (
         line2Point.y - line1End.y) -
154             (line1End.y - line1Start.y) * (
                 line2Point.x - line1End.x);
155 }
156
157 public void resetPosition() {
158     yOffset = MapDivider.mapYStart + MapDivider.
         totalObstacleHeight / 2 + MapDivider.
         totalObstacleHeight;
159     xOffset = -MapDivider.obstacleWidth;
160 }
161
162 public int getColor() {
163     return colors[currentColor];
164 }
165
166 public void closeColor() {
167     closedColors[currentColor] = true;
168
169     newColor();
170 }
171
172 public void newColor() {
173     ArrayList<Integer> openColors = new ArrayList<
         Integer>();
```

```
174
175         if (closedColors[0] == false) {
176             openColors.add(0);
177         }
178         if (closedColors[1] == false) {
179             openColors.add(1);
180         }
181         if (closedColors[2] == false) {
182             openColors.add(2);
183         }
184
185         if (openColors.size() > 0) {
186             Random rand = new Random();
187             currentColor = penColors.get(rand.nextInt(
                 openColors.size()));
188         }
189     }
190 }
```

Code example C.1: Car class from the original Cars project.

Appendix D

Email correspondence about volume

Email containing question for Tove, redirected by the requirements group:

Hej sw604 aka. Requirements-gruppe, Vi har et enkelt spørgsmål til Tove i forbindelse med vores projekt (Cars Audio Game): "Hvis styringen af bilen i Cars var baseret på volume (dB) ville det så være brugbart?" Samt en tilknyttet kommentar: Vi tænker at dette kunne være relevant ift. det som Mette sagde med at nogle af børnene hviskede når der skulle råbes og omvendt. Desuden er det meget kompliceret at få styringen til at virke med stemme-/toneleje, så vi ser dette som et kompromis for at kunne få lavet noget der virker.

Email containing the answer from tove, from the requirements group:

Hej sw613, Vi har lige været ude of snakke med Tove, og hun er syntes at det er en rigtig god idé mvh. sw604f14

Appendix E

Interview 3/4/2014 - Summary

Summary written by the contact person: Mette Thomsen.
For birken er det nok for dem at bilen viser at stemmen er høj eller lav. De kunne godt lide at sætte et lyd barometer ind i spillet så børnene fx kan ses hvor højt de snakker på en skala fra 0-10. De vil gerne have et ikon der slår dette fra eller til. Det må gerne være muligt at stoppe bilen. De laver så selv koblingen med børnene mellem spil og virkeligheden. En ide de havde, var at forhindringerne var vurderet på denne skala, og at bilen så også havde et nummer der opdateret alt efter hvor bilen var. Opdeling så midten er 4-6 og alt andet er over eller under. De vil stadig gerne have mulighed for at stoppe spillet og vise lyd barometeret. De vil gerne kunne indstille antal forhindringer og hastighed. Hvis der er mange forhindringer skal bilen være mindre så der kan navigeres rundt. Lav det så simpelt som muligt! Ikke for mange detaljer på bilen eller andet da borgerne har problemer med de bliver optaget af detaljer. De øver både at borgerne skal snakke lavere og højere. Det må gerne kunne indstilles hvor forhindringerne er så et barn der snakker for højt er forhindringerne deroppe hvor bilen kører når man taler højt. På den måde bliver de tvunget til at ændre sig. De vil meget gerne kunne indstille om det handler om at undgå eller samle ting, og eventuelt begge dele. Men det skal kunne deles op. Hold alt enkelt og simpelt!

Appendix F

Interview 8/5/2014 - Summary:

Tove had used the application a couple of times since the last sprint review, to see how it would perform and what could be improved. Tove was satisfied with the settings, but the speed settings could be improved so that the number would make more sense. Tove had forgotten how to calibrate the microphone, but remembered when showed. Tove did find it that the children had to use too much power to steer the car, when trying to avoid the obstacles. She would like the car control too be more sensitive.

The current application is presented. She is impressed with the new speed settings and the way the speed barometer is used to show that visually. But she finds the pictographs a little bit too small. The calibration setting was not changed and is still confusing. Tove finds that the map editor is fine. The game gets started and Tove is impressed with the improvement of the control of the car. Tove plays around with the speed and obstacles and is satisfied with it. But she finds that the game gets difficult too fast when adding more obstacles.

The input volume from the player should not be a long and continuous. The input volume should be short iterations of syllables like ”da-da-da” or ”ba-ba-ba”. Tove finds the game is hard with more than two obstacles.

The new idea, with picking up objects instead of avoiding them is presented and Tove likes the idea. Tove also agrees to remove the garages, because the teacher should be in control when making the map. Instead of the garages their should be a finish line.

The application should have sounds to strengthen the citizens auditory sense. A sound when picking up an object. A car sound when the game starts. A voice which says the button text. When the player wins a voice should say ”godt gået”.

When missing an object and going through to the finish line, the player should get another chance but not win. The button text ”fortsæt” should be

changed to ”ny tur”. The button text ”Menu” should be changed to ”Færdig”.

Bibliography

[1] Anders Jensen, Daniel Bøcker Sørensen, Tom Petersen, and Jakob Albertsen. Cars - a voice controlled game. Technical report, Aalborg University - student report, 05 2013.

[2] Phil Burk, Larry Polansky, Doublas Repetto, Mary Roberts, and Dan Rockmore. Music and computers – a theoretical and historical approach. `http://music.columbia.edu/cmc/MusicAndComputers/`, 2011. [Online; Accessed 05-03-2014].

[3] Google Android. Opengl es. `http://developer.android.com/guide/topics/graphics/opengl.html`, 2014. [Online; Accessed 13-03-2014].

[4] Google Android. Displaying graphics with opengl es. `http://developer.android.com/training/graphics/opengl/index.html`, 2014. [Online; Accessed 13-03-2014].

[5] Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. *Objekt Orienteret Analyse og Design*. Forlaget Marko, Aalborg, Denmark, 3rd edition, 2001. ISBN 87-7751-153-0.

[6] Mario Zechner and Robert Green, editors. *Beginning Android Games, Second Edition*. Apress, 2012.

[7] Kilobolt. Android game development. `http://www.kilobolt.com/unit-4-android-game-development.html`, 2013.

[8] David Benyon. *Designing Interactive Systems*. Addison Wesley, second edition, 2010.