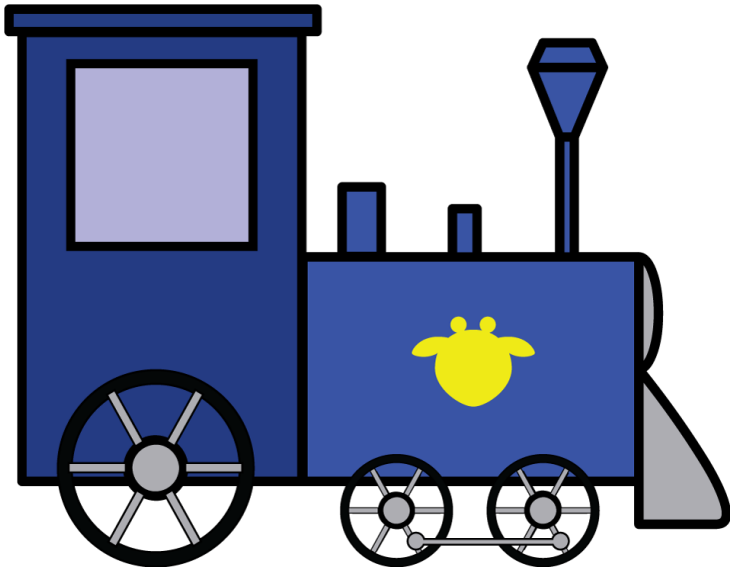
AALBORG UNIVERSITY

BACHELOR PROJECT

Interactive Learning Exercise for Children With Autism

- A Train Game



RESUMÉ

Vores applikation, Train, blev udviklet som en del af Graphical Interface Resources for Autistic Folk (GIRAF) systemet, som blev startet i 2011. GIRAF systemet er et multiprojekt som i 2013 bestod af otte grupper. Visionen for GIRAF er at lave en multifunktionel applikation til Android platformen som kan gøre livet nemmere for børn med autisme, deres pædagoger og forældre. Formålet er at erstatte fysiske genstande, som hjælper dem i dagligdagen, med digitaliserede udgaver. Ideen er at samle flere funktionaliteter et sted og tillade rig mulighed for konfiguration.

Vores gruppes fokusområde var at udvikle et spil til GIRAF systemet. Vi valgte at tage udgangspunkt i en øvelse som vores kontaktperson, Tove Søby, allerede udøver med børnene. Øvelsen går ud på at associere pictogrammer og/eller objekter med hinanden. For at gøre det til en leg havde hun et papir-tog til at køre rundt med objekterne.

Train applikationen består af to dele, en menu til at oprette spil-konfigurationer, og selve spillet.

Når applikationen starter, ser man menuen, her har man mulighed for at vælge det barn man ønsker at lave en spil-konfiguration for. En spil-konfiguration består af et antal stationer, som hver har en kategori og en mængde associerede pictogrammer. Når man har lavet en spil-konfiguration, kan man starte spillet.

I spillet følger man toget fra station til station og sætter de associerede pictogrammer af på hver station. Når der ikke er flere pictogrammer tilbage i toget så er spillet gennemført.

Den grafiske del af spillet er implementeret ved hjælp af OpenGL for Embedded Systems. Vi gør brug af drag-and-drop for at flytte pictogrammer til og fra toget.

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Software Engineering
Selma Lagerløfs Vej 300
Telephone +45 9940 9940
+45 9940 9798
http://www.cs.aau.dk

Title:
Interactive Learning Exercise for Children With Autism

Subject:
Developing Complex Software Systems.

Project period:
P6, Spring semester 2013

Project group:
SW606f13

Attendees:
Jacob Karstensen Wortmann
Jesper Riemer Andersen
Nicklas Andersen
Simon Reedtz Olesen

Supervisor:
Ulrik Mathias Nyman

Finished:

Number of pages: 95

Appendix pages: 7

Synopsis:

The GIRAF project is a multi-project consisting of eight groups each with their own sub-project. The GIRAF system is developed for the Android platform, and our focus is to develop a game.
The application is inspired by an exercise that our contact person, Tove Søby, practises with children with autism. The application makes use of drag and drop to drag pictograms to fulfil the games' criteria.
The application is split in two parts, the graphical part and a game customisation menu. The graphical part is developed using the OpenGL for Embedded Systems graphics API.
The customisation menu allows for easy creation, saving, and deleting of customised games, each individual child can have unique customised games saved to their profile.

Signatures

Jacob Karstensen Wortmann

Jesper Riemer Andersen

Nicklas Andersen

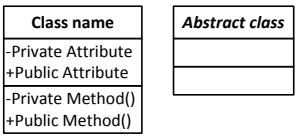
Simon Reedtz Olesen

PREFACE

This project was written as the Bachelor project by group SW606F13 - Software students from the Department of Computer Science at Aalborg University in the spring of 2013. The report documents the GIRAF project of 2013 and the implementation of the Train application. The application is developed for the Android platform. The reader is expected to be familiar with Java and UML. We have included our knowledge from all our previous semesters.

When reading the report, there are a few things the reader should be aware of:

- When a reference to a source of a section or paragraph is given, the number of the source is written inside square brackets [ ]. The number is a reference to the bibliography list on page 95.
- When "we/us/our" is mentioned in the report, it is a referral to the authors of the report
- In class diagrams a minus denotes a private attribute/method, and a plus denotes a public attribute/method. Italic class names are abstract classes.



We would like to thank our supervisor Ulrik Mathias Nyman for the feedback he has given throughout the project.

CONTENTS

Part I

GIRAF Introduction

CHAPTER 1

THE GIRAF PROJECT

Graphical Interface Resources for Autistic Folk (GIRAF) started out in 2011 as a semester project targeting children with autism and their guardians.

In the following chapter, the overall vision for the GIRAF project will be presented, the projects from previous years will be explained briefly along with the platform for the project. Lastly a section describing autism is included.

1.1 Vision for GIRAF

The vision for GIRAF is to create a multi-purpose application based on *Android* that can simplify and ease the lives of autistic children and their guardians.

The purpose of GIRAF is to replace physical items that are being used daily by the children and their guardians with digitized versions. The idea being to gather several functionalities in one object and allowing customization for each individual child.

This will also optimize work procedures on the individual institution in such a way, that guardians will save time doing repetitive tasks such as making pictograms. This time could be spent with the children instead.

As of the spring of 2013 three schools and institutions for children with autism in Northern Jutland are involved in the development, but the hope is that GIRAF will be distributed across all similar institutions in Denmark.

1.2 Previous Years

During the first year of development, four parts of the GIRAF project were developed. The four projects were developed during the spring semester of 2011 and included the projects:

Admin An administration interface used for administrating different aspects of the GIRAF system.

DigiPECS A digitized version of "Picture Exchange Communication System"[13] a system used as an aid for communication with people with special needs such as autism.

Launcher A home screen application and distribution platform for Android.

aSchedule A visual schedule for the Android platform.

During the spring semester of 2012, five new software groups continued development of the GIRAF project. The projects developed during 2012 were:

Launcher An enhancement of the launcher project developed during the spring semester of 2011.

Oasis An enhancement of the admin project from 2011. Furthermore the Oasis project developed a local database for the GIRAF system.

Parrot An enhancement of the DigiPECS project from 2011. The project was renamed because of trademark issues.

Savannah A server side database with web interface for the GIRAF system.

Wombat An Android application for measuring and visualizing time.

During the spring semester of 2012 two databases were developed, however synchronization between them was never achieved.

Problems with Initial Implementation

As the spring semester of 2013 started, an ”install party” for the students was held. The party was intended to help the students compile and deploy the projects from 2012.

Even though a representatives for each of the 2012 groups were present, some compilation problems still occurred.

The repository used for distributing in 2012, was disorganized and difficult to navigate, i.e. due to:

- Multiple copies of the same project.
- Unclear dependencies among the different project.
- Projects only meant to be compiled from Eclipse for Windows.

During the following week a working workspace was created and shared with the rest of the students, along with install instructions. The install instructions were later updated to a more clear edition.

1.3 Target Platform

Android is an open-source operating system originally developed by Android Inc, and later bought by Google Inc. The first release came in 2007, where it was launched by Google Inc. together with Open Handset Alliance (OHA), which includes companies such as Samsung, HTC, LG and Google.

Before the first students where involved in the project in the spring of 2011 Ulrik Nyman considered two platforms for the development of the project. The Android and iOS platforms. The Android platform was chosen for three main reasons:

- That the platform is open source.

- That in Android the developers can take control of the functionality of the home button.
- That distribution of the software is possible outside the official marketplace.

For the the two following years it has been chosen to stay on the Android platform. This is done both to be able to reuse the source code and because Android compatible hardware is available for the students. In the very long tern the system could support multiple platforms.

1.4 Autism

Autism is a spectrum disorder, meaning that it appears in different variants and not all people who are diagnosed have the same symptoms. The disorder can often be observed within the first three years of a child's life. Autism is a physical condition and is linked to abnormal chemistry in the brain, however the exact causes of these abnormalities are still unknown.[7]

Symptoms

Children with autism usually have difficulties understanding the concept of "play pretend", meaning that they have a hard time imitating the actions of others when playing and therefore prefer to play alone. Furthermore they have difficulties with social interaction and communication – verbally and non-verbally.

People diagnosed with autism may;

- Be very sensitive to light, noise, touch, and taste.
- Have a hard time adjusting to new and changing routines.
- Show unusual attachments to objects.

Autism diagnosed individuals may have a hard time starting and maintaining a conversation. They may communicate with gestures instead of words, develop language slower or faster than normal and some do not develop any language at all. Furthermore the lack of social interaction means they might have a hard time making friends, may be withdrawn and may avoid eye contact.[7]

Signs and tests

If a child fails to meet any of the following language milestones, it may be an indication that it needs to be tested for autism;

- Babbling by 12 months.
- Gesturing (such as pointing or waving goodbye) by 12 months.
- Saying single words by 16 months.

Children failing to meet any of the previous mentioned language milestones might receive a hearing evaluation, a blood test and a screening test for autism. Since autism covers a broad spectrum of symptoms, a single brief evaluation cannot predict what abilities the child has. Therefore a range of different skills are evaluated, such as:

- Communication
- Language
- Motor skills
- Speech
- Success at school
- Thinking abilities

Some parents might be scared of having their child diagnosed, however without a diagnosis, the child might not get the necessary help.[7]

Treatment

Autism cannot be cured, however an early diagnosis and treatment can greatly improve the child's quality of life. Different treatment programs usually build on the child's interests and are highly structured to their needs and routines.[7]

CHAPTER 2

THE GIRAF PROJECT 2013

When working in a multi-project consisting of eight groups, it is important to have a common goal for the project. This chapter describes this goal as a story. Furthermore the chapter includes description of the development process and the rules of conduct.

2.1 The Goals for 2013

Within the first couple of weeks, when all the groups had been assigned a project, a major story for the overall project was written.

The Major Story for 2013

> "The guardian arrives at the institution, and turns on the tablet. The guardian is aware of the arrival of a new child at the institution after lunch. The guardian sets up and customizes a profile for the child, this includes creation of new pictograms. Furthermore the guardian prepares games and a life story for the child.
>
> After lunch the new child and the guardian meet. The child is introduced to the communication tool Parrot. After some introduction they sit down to do some communication practice using the tool.
>
> Afterwards the child wants to go outside to see the rest of the institution, and needs to put on some outdoor clothes. The guardian introduces the child to the Zebra tool, and together they put on the child's outdoor clothes.
>
> When the child comes back in, the guardian and the child play the games prepared earlier by the guardian.
>
> When they are done playing the child and the guardian read the child's life story using Tortoise."

2.2 Definition of a Multi-Project

A multi-project is a project that includes multiple groups that each work on their own sub-project, which is part of a larger project. In this case, the larger project is the GIRAF system and each group works on a separate part of the system.

Compared to working on a single project in isolation, working together creates new challenges. The software produced by each group has to be integrated to ensure the entire system works properly. Some projects are more independent of the rest, while others depends heavily on some projects like the database project Wasteland described in Section 3.2.7. Groups have to be flexible and pass any requirements to other groups' projects early to prevent halts.

To ensure the project is successful and no misunderstandings occur, there must be good communication and cooperation between the groups. This requirement is amplified by the fact that there are no definitive authoritative figures, other than those chosen by project members.

2.3 Group and Work Structure

This section describes the development methods used during the spring semester of 2013, including stories and project management tools.

The section is rounded off by a description of the development tools used, including Redmine, Git, and Jenkins.

2.3.1 Development Method

Having a development method is one of the main ways to structure the work process of a project. A development method is a collection of methods and structures, from the way to have meetings, gathering requirements and structuring the development. There are many development methods, each is structured and handles issues differently,

however, it is rare that one fits a development problem perfectly. Different methods are often combined and customized to fit the problem at hand.

2.3.1.1 Implemented Development Methods

This project's nature calls for agile development, due to team collaboration, user feedback, product focus, and continuous integration. Agile development focuses on a flexible but structured work progress suited for projects with many unknown variables.

The agile development method has the ability to adapt to changing requirements throughout the project and focuses on having a shippable product at the end of each iteration.

2.3.1.2 Stories

User stories is one of the tools that helps streamline the work process, keeps focus on a shippable product and is the main component for management of the project. First of all the product story works as a common problem statement for all work groups. A product story is the agreement on what is necessary for the product to be finished. From product story each group can extract what is required of them to complete the story.

2.3.1.3 Management

The semester coordinator, Ulrik Nyman, has supervised the project since it's beginning. Ulrik Nyman himself has a child with autism and will continue being a part

of the project for the time to come, conveying his knowledge of the development process and the product.

To help fit the product to the needs of guardians, for which the product is intended, a number of representatives are included for more detailed feedback on the process and the product.

To keep as many work hours in development and to keep a good overall management, common meetings were held weekly. The common meetings had focus on sprints and team cooperation. Problems that needed further discussion and/or development were discussed by a committee consisting of a few representatives from each group.

The common meeting and committee meeting are further specified in sections Section 2.4.1 and Section 2.4.3.

2.3.2 Development Tools

A number of tools were used in order to optimize team collaboration and to make the projects more accessible. These tools will be further explained in the following sections.

A dedicated Linux server was commissioned for the entire GIRAF project and several services installed to facilitate collaboration and agile development. Common to all current services are their free, open-source nature and support of LDAP authentication, allowing all students and supervisors to log in using their AAU credentials.

2.3.2.1 Redmine

Several tools were audited for use in the project management aspect of development, including Trac, PivotalTracker and Github. Redmine, a Ruby-On-Rails web application, was selected owing primarily to its support of multiple projects and support features such as wikis, forums, milestones and various charts. The features most broadly used will briefly be described here.

Projects All projects live in a shared project space, and can be placed in a hierarchy under a super project. In this regard, the primary multi project served as the base of each of the eight groups' underlying projects.

Issue handling Redmine's primary feature is its issue handling. Project members can create and react to issues within custom-defined domains. For GIRAF, this was primarily development tasks, but could just as well be used for report-related tasks or general maintenance in an attempt to manage time usage.

Burndown Charts Redmine does not have native support for burndowns, but does support it through a Free and Open-Source Software (FOSS) third-party plugin. Burndowns are a visual aid of each subproject's progress throughout a sprint, giving quick summary of development speed and whether proactive action may need to be taken.

Milestones A generic milestone feature in Redmine is Versions. Versions are simply markers with a set date, and can be open or closed for attachment of issues. The burndown plugin couples a version's end date with attached issues and their progress to generate the related charts.

Wiki A per-project wiki module exists in Redmine. The basic wiki markup has been expanded to allow referencing of almost any other element in the project hierarchy, such as projects, issues, files and VCS revision.

Redmine has many more features not directly applied during this project period. However, many could be applied to create a more centralised and structured development experience in future projects. Examples include file and document hosting, advanced issue workflows, permission management and VCS integration. Future multiprojects may consider expanding into these fields if they feel proficient in Redmine's basic usage.

2.3.2.2 Version Control System

The university's IT services offers only a single version control system, Subversion. Although centrally supported and backed up regularly, Subversion's shortcomings were challenged before main development had begun. Most notably, the system's centralised workflow and high operation cost. Many of SVN's actions require access to the central server. Two alternatives without these issues were suggested: Git and Mercurial (Hg). The former was chosen as a general question of broad platform support and popularity. A primary strength of these systems is their support of separate branches of development without the constant need to connect to a central server. This allows developers of each project to synchronize with a main branch while maintaining several development branches on their own workstation.

Most groups used Github as hosting solution for development of their projects, as a git hosting solution was not immediately forthcoming (contrary to Subversion and Mercurial, Git does not have a default server implementation). At the conclusion of the project period, a solution was configured using Apache-based LDAP authentication, deferring authorisation and repository management to Gitolite, a low-footprint open-source offering.

In the interest of easier cross-project code contribution and inspection, an improved web solution may prove a better choice. Due to time constraints, a few solutions were briefly audited but ultimately discarded in preference of Gitolite. Gitlab should be mentioned as it featured an interface and features very close to those of Github itself, but proved difficult to install and maintain.

2.3.2.3 Jenkins

A principal element of agile development is continuous integration, the automated concurrent building of new code as it is pushed to central repositories which ensure constant availability of newest binary packages while catching coding errors before pushing them to the public. Jenkins, a fork of Oracle's Hudson, was suggested early and, given no proponents, was implemented. Build jobs were set up for each project, polling their origin repositories for new Git builds to main branches. If a repository has new code, it is downloaded and built. In case of build errors, the project developers are notified by email. To facilitate the deployment phase of each sprint, all projects are rebuilt every Thursday night and pushed to a public FTP server as well as making them publicly available by HTTP.

Git support is not part of Jenkins' core feature set, but is available as a plugin. During development, unhandled exceptions in the plugin code resulted in thousands of of superfluous builds as a failed build due to unexpected circumstances was not marked as failed.

2.4 Decision Making - The Process

The following section will describe the decision making process, set in place to ensure that everyone would be heard on an equal and democratic footing. The decision making process during this semester's multi-project consists of two different steps.

2.4.1 The Weekly Meeting

It was strongly recommended by the semester coordinator, Ulrik Nyman, to hold a weekly meeting for all software students on the bachelor semester of 2013. The meeting's agenda consists of a few points of formalism at the very beginning, in which a secretary and a moderator are chosen by means of voting. Candidates for these roles are entirely self-appointing and a vote is issued to pick one of the candidates.

Though the weekly meeting is established to ensure a higher level of communication between students, as well as ensure that decisions will be taken on a multi-project level scale, not all points are actually discussed at this meeting. Instead, a committee approach is agreed upon, see Section 2.4.3. The purpose of establishing committees is to ensure that relevant discussions to a given topic can be had, but within a smaller audience.

Committees are discussed at the weekly meeting where voting determines which committees are established. A chairman for a committee is self-appointed and a vote determines if there is consent to let the given person be chairman.

The meeting will then proceed and discuss the ideas and suggestions agreed upon within each committee from the previous week and at the multi-project level determine, by voting, which ideas are okay, or if any of the points concluded by one of the committees are subpar and should be reworked.

2.4.2 Rules of Conduct

During the first weekly meeting some general rules of conduct were established, including decisions on how voting should be done. A number of ways to do this were suggested. Ultimately it was decided that every person present at the meeting has an individual vote, and the idea of a group based voting system was therefore discarded. Furthermore in the event that there is a 50/50 split, the vote will have to be reissued. There must be majority 'for' or 'against' a decision. Guidelines for when a decision should be taken at the weekly meeting were established as well. If a decision involved only two or three groups, then it would not be necessary to discuss at the weekly meeting. If, however, the decision impacted everyone, a committee would be established to make these decisions.

During a committee meeting every group has a single vote. It is possible to send as many group members as is deemed necessary to the committee meetings, however, it does not increase the number of total votes a group has.

2.4.3 Committees

A committee ideally consists of a representative from each multi-project group and a chairman agreed upon at the weekly meeting. The chairman is responsible for

setting up the meeting, time, place, agenda as well as writing down the details of what is agreed upon during the committee meeting.

The resulting work product of the committee is a document, that potentially answers every question on the agenda, ready to be presented at the next multi-project meeting.

Important Committees

The following section describes an extract of some of the most important committees, that were established during one of the first weekly meetings.

- Wiki: *Ensures that the multi-project wiki page on Redmine is created in a uniform way by establishing guidelines for new articles.*
- Design Guidelines: *Ensures that the User Interface design of the GIRAF application is uniform (e.g. in regards to font, color scheme and various buttons - green for 'yes' and red for 'no').*
- Common Report: *This committee is responsible for the creation of the common-report chapters, which you are reading now, that are at the beginning of every semester report.*
- Pictogram Class: *Because every group requires a common pictogram class, it was decided to create a Pictogram Class committee to determine the functionality that this class needed.*
- GIT: *The GIT committee is responsible for working out a common structure across all repositories to create uniformity and make it easier to continuously integrate.*
- Public Pictogram: *Determines guidelines for how pictograms are handled in the database (e.g. who has access rights to what and why?).*
- Story: *The story committee is responsible for creating a story to follow every sprint. It puts the sprint's tasks into an overall context.*
- CI/Git: *This committee is responsible for coming up with solutions to potential issues that might occur as part of the Continuous Integration step when using GIT.*

CHAPTER 3

WHAT WAS DEVELOPED

This chapter describes the work done for GIRAF in the year 2013 and is rounded off by acknowledging the involved contacts and the semester coordinator.

3.1 Pictograms, Morgana, and Design Guidelines

In this section the notion of a pictogram will be presented followed by how pictograms are currently being used and why they should be digitized. Furthermore the section includes a description of the Morgana library.

The section will be rounded off with the overall design guidelines for the entire GIRAF system.

3.1.1 Pictogram

In the context of this report a pictogram is defined thus: *A pictogram is an image representing a living being, a physical object or some form of action.* Pictograms can contain a text-label, describing the respective images, for clarification. There is currently no standard for the layout or contents of pictograms, due to the specific needs and opinions of the users. User **A** might like to have black and white images with text labels whereas user **B** might want colorful images without text. The images can themselves vary from cartoons to photographic representations. Pictograms are commonly used as means of communication, especially by those requiring assistance with communicating, including but not limited to individuals with autism.

Current Use

During the spring semester of 2013, when this report was written, the use of pictograms is mostly in the form of physical images. The images need to be drawn and/or edited, printed, cut out and then laminated to extend their lifespan. After this process the pictograms are ready for use, generally for one individual, making this repetitive and tedious for the guardians.

When the required amount of pictograms have been created for an individual, they need to be organized and made accessible with the help of some sort of container. This container can be a folder with a pocket for the pictograms and a velcro-like strip for arranging the pictograms. For communication an individual

Figure 3.1: Pictograms in use 2013

can choose to form sentences by arranging the pictograms accordingly or use a single image to simply express needs and wants. Another purpose of the pictograms can be to graphically represent instructions for various tasks, in the form of "do **A**, followed by **B** and lastly do **C**" for individuals requiring special assistance.

Digitizing the Pictogram

The GIRAF project focuses on simplifying and digitizing a medium used by individuals with autism and their guardians. This includes digitizing the pictograms, making them available on devices running Android with added functionality. Added functionality includes the option to make the pictograms play a sound, dynamically change the layout of text-labels and editing images. Digitizing the pictogram also makes it possible to share them easily, carry them between devices and make backups of them. Previously, with the same idea in mind, it was attempted to digitize the pictogram. It was considered unsatisfactory (see section below) and therefore the re-implementation in this semester's project.

GIRAF Pictogram Design

The digitized pictogram consists of an image, text-label and a sound. With all elements included, it can be presented as each of the three, two parts combined or all three in union. This viewable container is designed as an extension of the *Android* view class, making it easy for developers to include and present in their applications. The idea is to have users sharing the same pictograms, with the option to customize their contents without affecting the pictogram itself. The previous GIRAF pictogram design lacked documentation, portability and functionality such

as text-labels. Therefore a new design was implemented, which hopefully fits the needs of both future GIRAF developers and GIRAF users.

3.1.2 Morgana

The Morgana library project was initially intended to make it possible for all the GIRAF applications to use both the Wasteland database, see Section 3.2.7, and the local Oasis database seamlessly, however in the time allotted it was not possible to finish this functionality, so the focus was shifted to making it parse and write JavaScript Object Notation (JSON) objects for use in calls to the Wasteland database.

The library implements a Java class for each value object documented in the Wasteland Application Programming Interface (API), each class parses a JSON object and turns it into an object which can be used by GIRAF applications, it is also able to create JSON objects from the stored Java object.

3.1.3 Design Guidelines

The purpose with the guidelines is to get a consistent look and feel across all of the different applications included in the GIRAF system. The design guidelines have been discussed among all of the project groups, and they are as follows:

- Keep the existing color palette
- Font: Helvetica
- Font size: use common sense. *Android* offers extra small/small-/medium/large/huge
- Minimize the use of text, use images instead of text
- Graphical User Interface (GUI) in vector graphics
- Green and red are universal colors for 'accept'/'cancel'
- Applications have animal icons
- Icons are non-customizable
- Every application should be locked in landscape mode

The color palette will be the same as in the 2012 version of GIRAF. With regards to font type and size, Helvetica has been chosen and developers need to keep in mind, that the text has to be readable on the tablet.

The aim is to use more images and less text as the target audience are mostly children, many of which have communication and/or reading difficulties and some have problems imagining objects purely from text.

The GUI will be in vector graphics, because it scales well, which makes it possible to reuse some of the images. Green and red are universal colors for 'accept'/'cancel'. It may sound obvious but other applications have been developed with different colors. Tool-applications should have animal icons.

Lastly everything will be in landscape mode as this eliminates additional implementation for responsive layout, when the tablet is rotated.

3.2 The Project of 2013

3.2.1 Admin

This project focuses on the creation of an administration interface for the GIRAF system. The Admin system consists of two parts, one for a desktop computer and one for *Android*. The desktop part will run on a Linux, Apache2, MySql and PHP (LAMP) stack and communicate with the database using the database API provided by the Wasteland (see Section 3.2.7) project. The *Android* part will run on the tablet using the same code base as the desktop part, using a web server application. The main focus of the project is for department managers and guardians to be able to administrate the GIRAF system.

3.2.2 Cars

The aim of the Cars project is to develop an application, which will help children with infantile autism to be more comfortable in using their voice. To ensure that the children learn to use their voice in creating different types of sounds, and not just speak in a monotone way, the application will require the children to create sounds covering different sides of the frequency spectrum.

Cars is a game in which the player has to lead a car through a street into a garage, controlling it with high or low frequency sounds. The car has a matching colored garage at the end, which when entered completes the game successfully.

Randomly placed obstacles are used to force the player to avoid them to reach the end.

3.2.3 Croc

The Croc project aims to create an application for creation of pictograms for use in the GIRAF system.

Pictograms can be created in a number of ways:

Camera take a picture with the camera and turn that picture into a pictogram.

Drawing draw a pictogram.

Audio record sounds to attach to pictograms.

3.2.4 Parrot

Parrot is an enhancement of the Parrot project of 2012 and is an application for communication between guardian and child. Its development is based around the currently used physical system Section 3.1.1. The original Parrot application from 2012 also included the administration of categories. It was therefore technically possible for a child using Parrot to access these administration tools, and it is for this reason, that the currently developed version has relocated the administration to a separate application named Category Adminitration Tool. The version developed during this project will focus on making improvements to the GUI design, adding subcategories (such as breakfast item under the food category) and handle the interaction with pictograms.

The primary focus for Parrot remains the same; providing an easier way for children to communicate with guardian in a way that they are familiar with.

Category Adminitration Tool

Category Adminitration Tool (CAT) focuses on administrating categories and subcategories. Currently CAT is also responsible for communicating with other applications that need specific pictograms, such as the Tortoise (Section 3.2.5) and Zebra (Section 3.2.8) applications, by providing search/deliver functionality.

3.2.5 Tortoise

The Tortoise application focuses on helping children learn about their own lives and strengthen their social skills. The hope is, that by letting the child interact with pictures and sentences, that are associated with their life, the child can develop an identity. By developing their own identity, the child will learn how to interact with other people by learning what kind of topics to talk about in a conversation with others.

3.2.6 Train

The inspiration for Train comes from an exercise, that one of the guardians practices with the children. The purpose of the game is to create a dialogue between the child and the guardian. The child has to drag pictograms from a train station onto the train wagons and make the train drive. When the train arrives at the next station, the child has to drag the correct pictograms from the train and onto the station. The correct pictograms are decided by the station category.

The category for each station is chosen by the guardians by clicking the category picture frame and browsing CAT (Section 3.2.4) for the picture they want to use. After selecting a category, they select which pictures they want associated with the station.

3.2.7 Wasteland

The purpose of the Wasteland project is to handle all of the data for the GIRAF system. In order to achieve this goal, a database will be implemented on a central server and a local database will be kept on the tablet. The two databases will synchronize data on a regular basis.

3.2.8 Zebra

The aim of the Zebra project is to create a software application aiding guardians in their work. The application should aid the guardian in situations where a child is to perform an ordered sequence of actions. These actions are typically represented by pictograms.

Zebra should replace the current paper based version of this system. The guardian should be able to create and manage digital versions of such sequences specific to each child. Upon selecting a sequence for the child to follow, the child should be able to mark actions as done when they are completed to illustrate their progress.

3.3 Acknowledgement

The group of students working with GIRAF during the spring semester of 2013, would like to thank the contacts, who were;

Tove Søby - speech therapist, and contact for three groups.

Mette Als Andreasen - kindergarten teacher at Birken Langholt, and contact for two groups.

Kristine Niss Henriksen - kindergarten teacher at Birken Vodskov, and contact for one group.

Drazenko Banjak - teacher at Egebakken Vodskov, and contact for one group.

Mette Frost - teacher at Egebakken Vodskov, and contact for one group.

In addition the group would like to thank Ulrik Nyman, semester coordinator, for his help, guidance and engagement during the project.

Part II

Train

CHAPTER 4

INTRODUCTION

We implement a game for children with autism on the Android platform, as a part of the GIRAF project.

The purpose of the game is to create a dialogue between the child and the pedagogue. The child has to drag pictograms from a train station onto the train wagons and make the train drive. When the train arrives at the next station, the child has to drag the correct pictograms from the train and onto the station. The correct pictograms are decided by the station category.

The category for each station is chosen by the pedagogues by clicking the category picture frame and browse the pictogram database and choose the picture they want to use. After selecting a category they select which pictures they want associated with this station, these are the pictograms the children have to drag onto this specific station.

Initial problem

Our initial problem was to create a game on the Android platform for children with autism.

During the first couple of meetings we came up with the theme for our game, we wanted to make a game involving a train and pictograms. The inspiration for this game came from one of the exercises that our contact person presented for us during the introduction week.

Problem statement

Based on our focus in the multiproject and our initial problem, we have come up with the following problem statement:

> In what ways can we aid the pedagogues in their work with children with autism, by digitalizing a physical exercise onto an Android tablet?

CHAPTER 5

CHRONOLOGICAL PROCESS

Week 1 - 18/02/2013 - 22/02/2013

After forming our groups we spent the first week brainstorming for ideas for a game. We had our first supervisor meeting this week as well, where we presented our initial idea and got some useful feedback

Week 2 - 18/02/2013 - 22/02/2013

After the brainstorm and discussions with our supervisor we came up with an idea for our game, the game idea takes inspiration from our contact person Tove Søby's own exercise with the children. In one of the introduction weeks of this semester, Tove Søby presented how she works with the children, improving their communication, categorization of objects, and social skills. In one of her exercises she tries to improve the child's skill to categorize objects. This exercise consist of making the child take the right object, placing it on a train, and drive the train to the correct train station that accepts this object. This exercise has proven to Tove Søby to be very powerful since it can be modified to help children categorize colors, sizes of objects, people of the child's social circle, and more. Though the exercise is good, the preparation time for each game can be very time consuming. She often has to make new pictograms or find objects of a certain criteria, so they can be categorized. Our game idea is to digitize the exercise and keep the principles of the exercises, by moving the exercise to a tablet, we also hope that the preparation time for each child is reduced.

We arranged a meeting with Tove Søby, so that we could present our idea and show her the paper prototype that we had made to demonstrate our idea.

Since this is a project that we are continuing from last year we had to set up Eclipse so that we could compile the old projects and run them on our tablet.

After setting up Eclipse we spent the rest of the time experimenting with Android code and make small runnable programs.

Week 3 - 25/02/2013 - 01/03/2013

Pictograms can now be dragged and dropped different containers. In case a container is filled or the pictogram is released outside a container, the dragged pictogram snaps back to its original container.

We also discussed how the layout of our game should be, in regards to where each pictogram container, train station, train etc. should be.

We also looked into different methods to make our background slide to create the illusion that our train was moving between stations.

Week 4 - 04/03/2013 - 08/03/2013

The sliding background that works, however we discovered that it is too slow, we discussed whether or not we should use OpenGL for Embedded Systems (OpenGL ES) instead.

We also started making the graphics we needed for our game. It was decided in the design committee that all graphics should be drawn in vector graphics, so that it easily could be resized without losing quality.

We also had a short discussion regarding how to make it look like the train was moving.

Week 5 - 11/03/2013 - 15/03/2013

This week we finished Sprint 1, which focused on getting our application into the existing GIRAF launcher. We achieved this.

We were also asked, by the Chief Integration Officer to make an ant script for our project so that he could set up continues integration for the multiproject. This was also achieved.

We have decided to use OpenGL ES, since the alternative was way too slow. We use OpenGL ES to render our graphics. We are able to draw objects / pictures on the screen using OpenGL ES.

We also created an icon for our game.

We had the second meeting with Tove Søby this week as well, she gave some comments on our graphic and some great ideas for improvement that we can use. She also suggested that there could be changing weather when the train moved from station to station.

Week 6, 7, 8 and 9 - 18/03/2013-12/04/2013

Since the last notes we did a lot of "under the hood" work:

- We decided to use OpenGL ES 1.x instead of the newer 2.0. We chose to use 1.x because it should be easier for OpenGL beginners.
- The class diagram (structure) for OpenGL ES drawing is implemented. It is now easy to draw and move objects across the screen.
- The wheels on the train and the smoke coming from the exhaust is now animated to create the illusion that the train is actually moving.
- The train can now smoothly accelerate and decelerate and can stop at exact coordinates at a station.
- Garbage collection does almost not run during the game any more.
- We implemented the Pictogram class.

- We sent an e-mail to Tove Søby with some prototypes of how the game looks. She has replied with some improvements.

Week 10, 11, 12 and 13 - 12/04/2013 - 10/05/2013

The game is almost finished, there are just a few bugs that needs to be fixed. Among the things we implemented are:

- We chose to use Three-dimensional space (3D) to create our game, even though the game is in Two-dimensional space (2D). When the train is driving the background have to seem like it is farther away than the train is by sliding slower than the train is driving. The easy way to achieve this is by putting the background farther away in the depth.
- There are hills in the background. They are made in sequences of four hills and are randomly picked throughout the game.
- There are clouds and a sun on the sky. The clouds are moving at a random speed and at a random height on the sky.
- There cows and trees on the hills. These come at random times throughout the game.
- There is a train depot to indicate that the game has finished.
- Pictograms are drawn on the station when the train begins to drive. This is done with OpenGL ES since we had some issues animating the layouts to follow the train.
- The game customisation menu is finished, it is now possible to create/save/delete game configurations.

During the debug sprint we finished the application. We also started writing on our report.

Week 14 - 13/05/2013 - 17/05/2013

We have been writing more report. We have also been making a few small fixes and tweaks to the code. We were visited by Mette Als Andreasen that wanted a demonstration of our application. Upon demonstrating it we saw for the first time that power-of-two (POT) texture was not supported on her hardware. However a fix for this was already implemented, but disabled at the time.

Week 15 - 20/05/2013 - 24/05/2013

We have been writing more report. We have also been trying to deal with a glitch for some weeks. The first pictogram being dragged at each station will make the station layouts disappear. We now have a workaround for this: Each time a drag event starts we draw all the layouts again.

Week 15 & 16 - 27/05/2013 - 04/06/2013

We gave the finishing touch to the report.

CHAPTER 6

OPENGL FOR EMBEDDED SYSTEMS

This chapter describes how we use OpenGL ES, it is included because we use it to draw the graphics for our game.

OpenGL is a 2D and 3D graphics API. It is a cross-platform API that specifies a standard for 3D graphics processing hardware. Android 1.0 and later versions have support for OpenGL ES 1.0 and 1.1 specifications. Android 2.2 added support for OpenGL ES 2.0. [3; 8]

Choice of OpenGL ES Version

When using OpenGL ES on the Android platform, the default OpenGL ES version is 1.*x*. If you want to use version 2.0 you need to explicitly write that you are using it. Version 2.0 should have better performance and more possibilities than the older version, but the Android developers guide says:

> *"Developers who are new to OpenGL may find coding for OpenGL ES 1.0/1.1 faster and more convenient."* [3]

Since we are beginners to OpenGL we have chosen version 1.*x*. One big difference between 1.*x* and 2.0 is that you have to write your own shaders in versions 2.0, this allows for easier effects customization. The Train game does not really require much special effects other than simple drawing of textures.

6.1 Design

In this section we show the class hierarchy that we have designed in order to use OpenGL ES within our game. The design description gets very close to an implementation description, but a few specific methods will be described in more detail in the next section.

6.1.1 Renderable Classes

Considering the game and its purpose, we came up with different objects that should be rendered. Figure 6.1 shows a simplified version of the different objects/classes that can be rendered. We needed to draw texture at different positions on the screen and we needed to be able to move texture around on the screen. This is the general idea and based on this, the following classes were created.

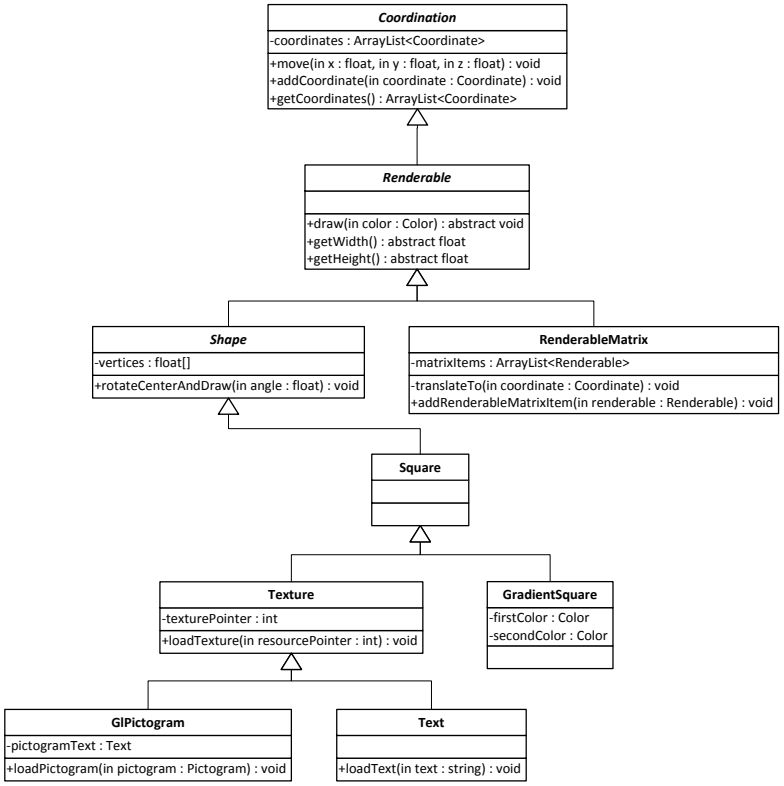


Figure 6.1: Class diagram of the objects that can be rendered.

Coordination/Renderable: In order to move objects around we have the abstract super class Coordination which defines the methods to move. The abstract class Renderable specifies that the objects needed to be rendered on the screen must implement the abstract methods: draw(), getWidth(), and getHeight(). If some Renderable object has to be be drawn more than once, then instead of making two identical Renderable objects with different coordinates, we have a list of coordinates in Coordination. The move method will then move all the coordinates by the specified amount. It was also a possibility to move only one specific coordinate by the specified amount, but this last scenario was never needed.

RenderableMatrix: Since almost everything is moved relative to the train's speed we needed some kind of grouping of Renderable objects, that were moved together. This is what RenderableMatrix handles. It is possible to add other Renderable objects to the RenderableMatrix, this means that it is also possible to add RenderableMatrix to RenderableMatrix. The addRenderMatrixItem() method also has a color as parameter to use when drawing the Renderable, this parameter is not shown on the figure to reduce the size of it. When calling draw() on this class, it iterates through all matrixItems and calls draw on them. The draw() method's color parameter for the RenderableMatrix is ignored, and instead

it uses the color specified for each matrix item.

The RenderableMatrix must also implement the abstract methods getWidth() and getHeight(). The width is calculated by the difference between the lowest x-coordinate to the greatest x-coordinate, and the same goes for the height with the y-coordinate instead. This feature was used at the time when this was designed, but has later become unnecessary.

The RenderableMatrix is made with a new OpenGL matrix, this will be explained in more detail in Section 6.1.3.

Shape: Everything that takes the form of a shape e.g. a square or a triangle, must inherit from the abstract class Shape. All shapes consists of an array of vertices. The vertices for the Shape determines how it is drawn relative to the coordinate.

Since we need wheels to rotate on the train, Shape implements a method to rotate the Shape around its center.

Square: The Square is initialised with a size in the constructor, and then vertices are created according to the specified size. We decided that the Square should be drawn from the top-left corner on its coordinates.

The color in the draw() method's parameter specify the color of the square.

GradientSquare: This is similar to the Square, instead this square is initialised with two colors to create either a horizontal or a vertical color gradient between the two colors.

Texture: This is the most important and most used class when creating our game. Since all pictures/images are 'square', it inherits from Square, and then the texture is mapped onto that Square.

This class must at some point load a resource pointer to a texture pointer. The loaded texture will always stretch to the size of the Square that it is drawn upon. The Texture class also implements ways to keep the original aspect ratio of the texture. Although not shown in the figure, the loadTexture() method also has an aspect ratio option that allows the size of the Square, containing the texture, to resize based on the size of the texture.

During loading of a piece of texture, the texture is converted to a power-of-two (POT) sized texture, Section 6.1.2 explains why. OpenGL ES has maximum dimensions for texture which is dependent on the device in use [4]. Each piece of texture is checked whether they respect the maximum dimensions, if it does not it is scaled down.

The draw() method must be overridden in order to draw the loaded texture. In this case it is possible to change the color that the Texture is drawn with. No matter what color the original Texture is, a color filter can be put on top of that on each call of the draw method.

Text: Adds the possibility to draw text. loadText() generates a bitmap with the specified text to use as texture. This is an easy way to draw text, but if you need to draw text that changes all the time, then loadText() is a slow way to do it. Fortunately we only need to load one time for each Text object.

GlPictogram: Adds the possibility to draw a pictogram. The method loadPictogram() generates a bitmap of the pictogram image, and then uses it as the texture for the object. To draw the pictogram text, a Text object is used.

6.1.2 Texture Power-of-two Support

Some OpenGL ES devices does not support texture with a non-power-of-two (NPOT) size, they only accept texture with a POT size, i.e. 2^x. Android developers reference states that you should perform a check whether the current context supports NPOT. [4]

You get an increase of performance when using POT, and at the time when the first version of OpenGL was created, the hardware needed this extra performance. Years later hardware only gained an insignificant increase of performance so they allowed NPOT. Some embedded systems using OpenGL ES still gain a significant increase in performance, and this is why some devices require texture to be POT. (We were not able to find official citation, the best we could find is this [10])

Instead of performing a check to see whether the current context supports NPOT, we choose to always use POT. Even if it is not necessary it will still give a small performance increase. The way we achieve this is to still use texture with a NPOT size, convert the texture to POT size, and then stretch it relative to the change in size. However this will also consume a lot more memory, but the train game does not require that much memory even with POT sized texture.

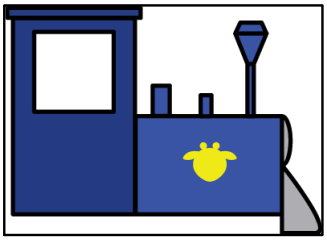


Figure 6.2: The NPOT train texture inside the Square container.

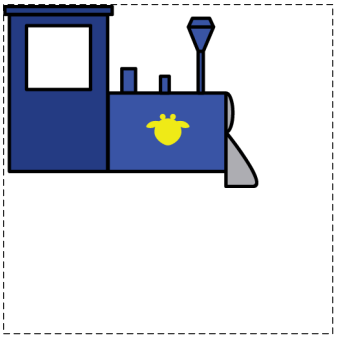


Figure 6.3: The POT train texture.

Figure 6.2 shows the train texture inside the Square container. The black line illustrates the Square, this is shown for the sake of the example and it is of course not shown in the actual application. The texture has a size of 396×284 pixels, and need to be converted to a POT size.

Figure 6.3 shows the train texture with alpha channels to the right and below the texture to obtain a POT texture with the size of 512×512 pixels.

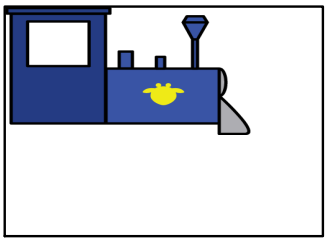


Figure 6.4: The POT train texture inside the Square container.

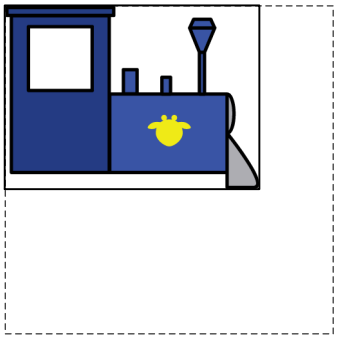


Figure 6.5: The POT train texture after stretching.

When we take the new POT sized texture and insert it into the Square the texture will still clamp to the edges. Figure 6.4 shows the clamped train texture.

We need to adjust the texture in order to fit the POT texture inside the Square. We stretch the texture by mapping the texture outside of the Square relative to the change in size from NPOT to POT. Since nothing outside the borders of the Square is drawn, then you could also say that the extra alpha channels are cropped away. Figure 6.5 shows the result of the texture conversion from NPOT to POT.

6.1.3 Game Rendering

To draw the game we are using an android.opengl.GLSurfaceView. To draw on the GLSurfaceView we are required to make our own class which implements android.opengl .GLSurfaceView.Renderer [3]. Figure 6.6 shows a simplified version of the different objects/classes that draws the game.

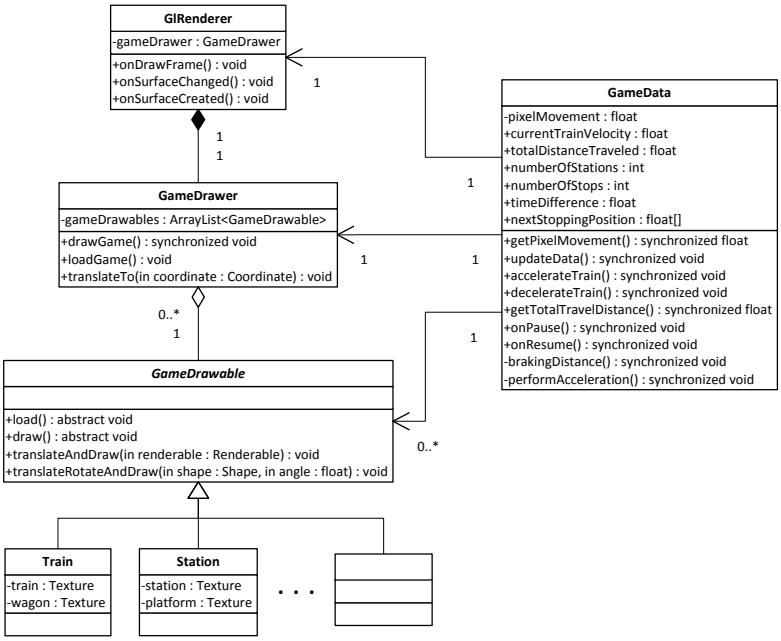


Figure 6.6: Class diagram of the objects involved with drawing the game.

GlRenderer This is our implementation of a Renderer, the class must override onDrawFrame(), onSurfaceChanged(), and onSurfaceCreated(). These three methods are all run on a thread that is automatically created for us when we implement the Renderer. When the Renderer is started, it runs the method onSurfaceCreated(). This is where we initialize the GameDrawer object. After the surface is created it runs onSurfaceChanged(), it is at this time the Renderer knows how big the surface is in pixels. We use this information to load all the texture in the game, by calling loadGame() in the GameDrawer. When the game is loaded it runs onDrawFrame() continuously, where we draw the game by calling drawGame() in the GameDrawer.

Even though the game is in 2D we choose to create it in a 3D environment. When onSurfaceChanged runs we create our main 3D matrix. OpenGL ES 1.x maintains a stack of matrices, the stack is required to contain at least one matrix. It will give an error if you pop all matrices. The RenderableMatrix is drawn by translating to its coordinates, and then pushing a new matrix on the stack. Then this new matrix can be translated in and the RenderableMatrix can draw its Renderable objects. When a matrix is popped from the stack you return to the same position as when you pushed a new matrix.[12]

GameDrawer When this object is initialized it creates all the GameDrawable objects, and add them to a list in the order they should be drawn. When the GlRenderer calls loadGame() or drawGame(), the GameDrawer simply iterates through the list of GameDrawable objects, and calls load() or draw() on them. Loading will only happen each time the surface is changed, i.e. when the activity is created.

An important thing the GameDrawer does, although not shown in the figure, is maintaining the current position in the frustum. translateTo() will translate the current position to the specified coordinate. RenderableMatrix also maintains the position of the matrix that it is working with. GameDrawer and RenderableMatrix is actually pretty much the same, the main difference is that GameDrawer draws GameDrawable objects and RenderableMatrix draws Renderable objects. It would be an elegant solution to refactor the code to get rid of either RenderableMatrix or GameDrawer instead, but we deem that we want to use both classes in order to separate the main 3D matrix from the others.

GameDrawable This is an abstract class. Classes inheriting from this must implement load() and draw(). In the load() method, drawables should add coordinates to Renderable objects, and call their respective load methods. In the draw() method, it should call the Renderable objects respective draw() methods, or use either translateAndDraw() or translateRotateAndDraw(). The two 'translate' methods both translate to a coordinate, by using translateTo() in the GameDrawer, and then draw the Renderable. Note that translateRotateAndDraw() will only rotate Shape objects. We do not allow rotation of RenderableMatrix.

GameData When the activity is created then we create one GameData object. This contains all the data needed to determine what has happened, what is happing, and what will happen later. When the object is initialized it will set the numberOfStations attribute according to the current game configuration.

pixelMovement is the number of pixels that the train has moved in the current frame. It is based on the currentTrainVelocity and the timeDifference between this frame and the last one. numberOfStops is how many times the train has stopped in this game session, and it is also used as an index in the nextStoppingPosition array which is an array with all the locations where the train should stop. The stopping positions are based on totalDistanceTraveled which is the total number of pixels the train has moved. Of all the attributes, it is only pixelMovement that is private. Some of the other attributes should also have protected access, but in this case it is better looking to access attributes directly instead of using get methods all the time.

The GameDrawer starts its drawGame() method by calling updateData() which updates the above attributes. It also checks whether the train should begin deceleration based on the method brakingDistance() and the stopping positions. In order to get the train started or stopped the method accelerateTrain() and decelerateTrain() are used. They set a flag that indicates the train is changing velocity and then set a positive or negative acceleration constant. performAcceleration() updates the train's velocity and makes sure that the train gets a nice smooth stop/start.

Do not mistake the method getTotalTravelDistance() and the attribute totalDistanceTraveled with each other. The method returns the total distance the train must travel for this game session plus the width of the screen, this is used to generate all of the background for the game session.

onPause() sets a flag indicating that the game is paused, it then saves the train's velocity and sets it to 0. The GameDrawable objects then know that the game is paused and can stop updating their position. onResume() will restore the train's velocity and the game continues.

6.1.4 Frustum

When our renderer is created it sets up 3D perspective. Figure 6.7 shows a perspective looking at the frustum between the near and far clipping planes.

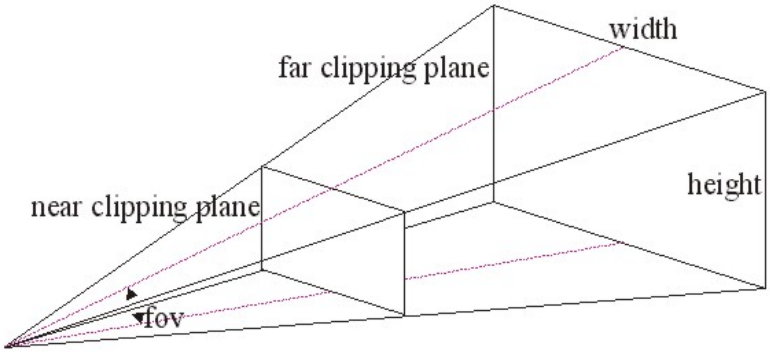


Figure 6.7: Perspective (Figure from [9]).

Inside the frustum is where you should draw, if you try to draw outside the bounds of the frustum, then nothing will be drawn and it will not consume resources. When setting up the game perspective we provide OpenGL ES with the current aspect ratio of the screen, a field of view angle (denoted 'fov' in Figure 6.7), and the depth of the near and far clipping planes. The width and height increases in size when we go deeper into the perspective, but the field of view angle always spans relative to the height of the screen. The width is always relative to the aspect ratio. This means that if we draw a square at a specific coordinate in the perspective, then the square will always have the same percent of space above and below it on the screen, but the space to the right and to the left of the square changes according to the aspect ratio of the screen.

The coordinate $(0,0,0)$ is where the perspective starts, or you could say it is where we are looking from. If you translate into the depth of the frustum e.g. $(0,0,50)$, then you are at the center of the perspective/screen.

The tablet has a resolution of 1280×800, but since it always displays a system bar with back-button, notifications, etc. we only have 1280×752 pixels of the screen available. All the graphics created for the game was made on a canvas with the same size as what we have available. In order to draw the graphics on the tablet screen we have to find the depth where the width and height in the frustum are the same as our screen size 1280×752.

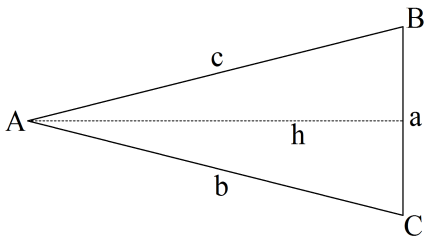


Figure 6.8: Perspective seen from the side.

Figure 6.8 shows the perspective seen from the side, we want to find the height h (depth) of the triangle when a is the same size as our tablet $a = 752$. We have chosen a field of view angle to be $A = 45°$. Since the angles B and C are the same size then

$$B = C = \frac{180° - A}{2} = 67.5°$$

Given a and all the angles of the triangle, then h is given by

$$h = \frac{a}{2} \times tan(B) = 907.7442$$

Now we have the depth of the perspective where one pixels of a texture corresponds to one pixels on the tablet screen. This will also make it possible to use pixels per millisecond as an actual unit of speed for our game.

6.2 Implementation

Some important code listing will be shown here. They will be simplified to make them easier to read. Each listing is followed by a description of it.

When using any of the Texture objects respective load methods, you must specify an aspect ratio option, the options are represented by an enum shown in Listing 6.1. The load methods must also create a pointer to the texture. Listing 6.2 shows how we generate a texture pointer.

```
1 public enum AspectRatio {
2     KeepWidth, KeepHeight, KeepBoth, BitmapOneToOne
3 }
```

Listing 6.1: Aspect ratio enum.

There are four aspect ratio options, which are all used to resize the Square according to the texture being loaded. Recall that texture will always stretch to the size of the Square. All Shape objects must specify a size in the constructor, but when loading texture onto the Square we want to be able to resize the Square to fit the aspect ratio of the texture.

KeepWidth will keep the width specified in the constructor, and resize the height of the Square to maintain the aspect ratio of the texture.

KeepHeight will keep the height specified in the constructor, and resize the width of the Square to maintain the aspect ratio of the texture.

KeepBoth does nothing, will keep both the width and height of the Square.

BitmapOneToOne will set the size of the Square to the same width and height of the texture.

```
1 protected void generateTexturePointer(GL10 gl, Bitmap bitmap, AspectRatio option) {
2     // Resizes the shape to fit the aspect ratio option
3     this.setAspectRatio(option);
4
5     // Resizes the bitmap to a power-of-two and crops the texture accordingly
6     bitmap = this.generatePowerOfTwoBitmap(bitmap);
7
8     // Resizes the bitmap if its size is not supported on this device
```

```
9       bitmap = this.maintainMaxTextureSize(gl, bitmap);
10
11      // Generate one texture pointer...
12      gl.glGenTextures(1, this.texture, 0);
13      // ...and bind it to our array
14      gl.glBindTexture(GL10.GL_TEXTURE_2D, this.texture[0]);
15
16      // Specify parameters for texture
17      gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
18      gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
19
20      // Use the Android GLUtils to specify a 2D texture image from our bitmap
21      GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap);
22  }
```

Listing 6.2: Generates an OpenGL ES texture pointer.

Line 3 Performs the resize of the Square according to the aspect ratio option as explained above.

Line 6 Convert the bitmap into a POT sized bitmap. It will be explained in Listing 6.3.

Line 9 Scales the bitmap down if its size is greater than what is supported on the device. It will be explained in Listing 6.4.

Line 12 This is where the texture pointer gets created. The arguments to the method are respectively: The number of texture pointers to be generated, this.texture is an array of integers, and the offset in the array.

Line 14 We bind our texture pointer to OpenGL ES in order to create texture at the pointed location.

Lines 17-18 Here we specify some texture parameters to be used when it is drawn. We specify that if the texture is upscaled or downscaled when drawn, we want to use LINEAR scaling. This should be the highest quality of scaling available to us.

Line 21 This is where we create the texture from the bitmap. The arguments to the method are respectively: Specify that it is a 2D texture, we are using the base image level 0 instead of the *n*th mipmap reduction level, and the bitmap to use.

```
1   protected Bitmap generatePowerOfTwoBitmap(Bitmap bitmap) {
2       int newWidth = this.getNextPowerOfTwo(bitmap.getWidth());
3       int newHeight = this.getNextPowerOfTwo(bitmap.getHeight());
4
5       // Check whether the bitmap wasn't already the correct size
6       if (newWidth != bitmap.getWidth() || newHeight != bitmap.getHeight()) {
7           // Create a new bitmap that is power-of-two sized
8           Bitmap newBitmap = Bitmap.createBitmap(newWidth, newHeight);
9           Canvas canvas = new Canvas(newBitmap);
10          canvas.drawBitmap(bitmap, 0f, 0f);
11
12          // Crop the texture to fit the original size
13          this.cropTexture(bitmap.getWidth(), bitmap.getHeight(), newWidth, newHeight);
14
```

```
15          return newBitmap;
16      }
17      else {
18          return bitmap;
19      }
20  }
```

Listing 6.3: How we generate POT sized texture.

Lines 2-3 Get the POT size for the width and the height. getNextPowerOfTwo returns a POT integer that is equal to the input or the next POT integer greater than the input.

Lines 6-16 If the bitmap was not already POT sized, then convert it to POT.

Lines 17-19 If the bitmap was already POT sized, then we can just return it.

Lines 8-10 We created a new POT bitmap with the calculated size, then we create a Canvas to draw on the POT bitmap, and then we draw the old NPOT bitmap on the POT bitmap in the top left corner.

Line 13 Here we change the mapping of the texture on the Square to stretch the bitmap outside the bounds of the Square and thereby cropping the extra alpha channels of the POT bitmap.

Line 15 Returns the newly created POT sized bitmap.

```
1  protected Bitmap maintainMaxTextureSize(GL10 gl, Bitmap bitmap) {
2      int[] maxSize = new int[1];
3      gl.glGetIntegerv(GL10.GL_MAX_TEXTURE_SIZE, maxSize, 0); //Get max supported texture size on
           this device
4
5      //If the width is to big and width is greater than the height
6      if(bitmap.getWidth() > maxSize[0] && bitmap.getWidth() >= bitmap.getHeight()) {
7          int height = (int) (bitmap.getHeight() * (maxSize[0] / (double) bitmap.getWidth()));
8          return Bitmap.createScaledBitmap(bitmap, maxSize[0], height, true);
9      }
10     //If the height is to big and height is greater than the width
11     else if(bitmap.getHeight() > maxSize[0] && bitmap.getHeight() >= bitmap.getWidth()) {
12         int width = (int) (bitmap.getWidth() * (maxSize[0]) / (double) bitmap.getHeight());
13         return Bitmap.createScaledBitmap(bitmap, width, maxSize[0], true);
14     }
15     else {
16         return bitmap;
17     }
18 }
```

Listing 6.4: Scales the bitmap if the size of it is not supported on the current device.

Lines 2-3 First we get the the max texture size supported on the current device. Do not question that we use an array to get this integer, OpenGL ES is made like the C language and some of the methods are characterised by this.

Lines 6-9 If the width is greater than the maximum allowed size, and the width is greater than the height, then we set the width to the maximum allowed size and decrease the height by the same percent as the width is decreased with.

Lines 11-14 If the height is greater than the maximum allowed size, and the height is greater than the width, then we set the height to the maximum allowed size and decrease the width by the same percent as the height is decreased with.

Lines 15-17 If the bitmap already respected the maximum texture size then we can return it.

CHAPTER 7

DESIGN

This Chapter describes the design of our application. It is split into two sections, one describing the design of our game and the other describing the design of menu.

7.1 Game Design

In this section we will discuss and explain our thoughts about the game design, we will also comment on the feedback given by our contact person Tove Søby. Finally we will show the result of the game design.

7.1.1 Game Idea/Description

In Chapter 5 it is explained how Tove Søby's version of the exercise work. We intend to keep the same principles of Tove Søby's exercise. We will start by given a short description of the important objects of the game.

Pictogram Is an object which contains a picture and is draggable.

Train Station Is a station where the train stops, and allows the child to drag and drop pictograms onto the station. Each station has a picture indicating the category of the station. The category of the station describes the pictograms that should be dropped onto the station.

Train whistle Is a button that the child can press if they think that the correct pictograms have been dropped onto the station. If however the child have made a mistake and dropped a pictogram not belonging to the station's category, then the train will not start. The train whistle was added by Tove Søby's request.

Train and wagons The train is the vehicle that drives to each station. The train has two wagons, each wagon can carry either two or three pictograms, depending on the total number of pictogram in the game.

Landscape Is a sequence of hills, trees, cows, and clouds. The landscape objects are placed in the background behind the stations.

Train depot When the train is driving from the last train station, the train will drive into the train depot to indicate the the game is over. The train depot was added by Tove Søby's request.

At the start of the game, the child must drag the pictograms off the start train station and drop them onto the train, the start station is different since it does not have a category. The start station is to help the child understand how to move the pictograms with drag and drop. When the child has moved all the pictures onto the train, it is allowed to drive towards the next station. Between the stations the train drives through a landscape sequence. The next train station has a category, which could be a picture of animals. It is now up to the child to drag all the animals off the train onto the station, before the train is allowed to continue. If all the right pictograms is on the station, the train will drive onto the next station. When all the pictograms is off the game will end.

7.1.2 Game Prototype

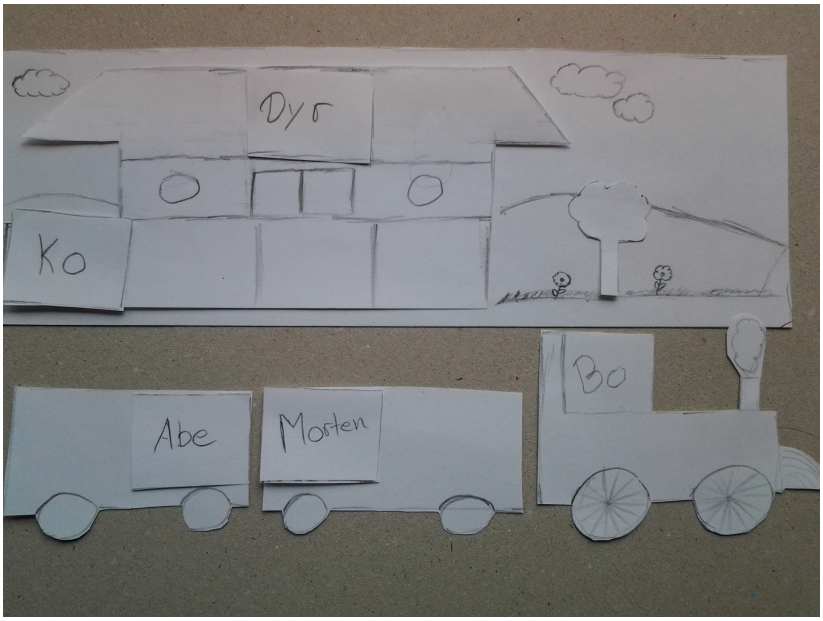


Figure 7.1: The paper prototype.

Figure 7.1 shows the paper prototype we showed Tove Søby at our first meeting, in which we explained our game idea. She liked the game idea and the user interface. Although she would like to add a button that checks if the correct pictograms have been dragged off the train, if this is case the button should start the train. She also mentioned that the game should have an ending, she suggested that the train could drive into a train depot. To accommodate these two features we added the train whistle and train depot to the game. Tove Søby mentioned that children like small surprises, but we should be careful that these surprises does not contain many colors and sounds, since this would cause visual and audible noise for the child. To accommodate these small surprises we added the decorative objects in the landscape sequence, and the whistle sound on the whistle.

7.1.3 Final Game Design

During the project we sent screen shots of the game interface to Tove Søby, to confirm that we were on the right track. Mostly she did not have much to add, only small graphical adjustments to stations.

In order to mimic Tove Søby's version of the exercise we had to make the game customizable, this was achieved by adding a menu where it is possible to customize the pictograms and the category for each station. The menu and its functionality is explained later in Section 7.3. During the design phase we agreed upon some constants of the game, the distance between the stations, the velocity of the train, etc. The constants in the game have all been approved of Tove Søby. The constants and their values are shown in Appendix A. To prevent that the game is the same every time it is played, we decided to add random elements to the game, e.g. the order of stations, the placement of the hills, and the decorative objects(cows, trees, clouds). All the textures of the game can be viewed in the Appendix B

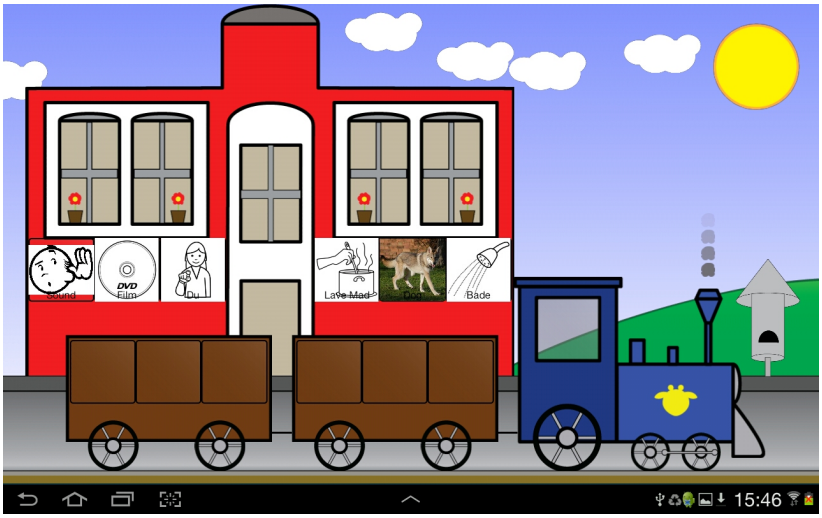


Figure 7.2: The final result.

Figure 7.3: Miniature game.

Figure 7.2 and Figure 7.3 shows the final result of the game design. Figure 7.2 shows the starting station, note that the station is is without a category in the top section of the station. The pictograms have also yet to be dragged off the station. Figure 7.3 shows a full game, though the game is simple it contains all the necessary objects. It should be noted that the game shown is missing the landscape sequence and all the decorative objects.

7.2 Game Configuration

We made two classes to hold the data of a game. GameConfiguration contains data about the game, such as the name of the game and who it is associated with. It also contains an ArrayList of StationConfiguration objects. A StationConfiguration contains the category of the station and an ArrayList of pictograms that will be accepted at that station. See Figure 7.4

Methods worth noticing are writeConfiguration() and writeStation(), these will write the object to a string, ready for saving to a file or in the database. writeToParcel will write the object to a parcel object which is *"designed as a high-performance inter-process communication transport"*[6]. The parcel object is used to pass the GameConfiguration between activities.

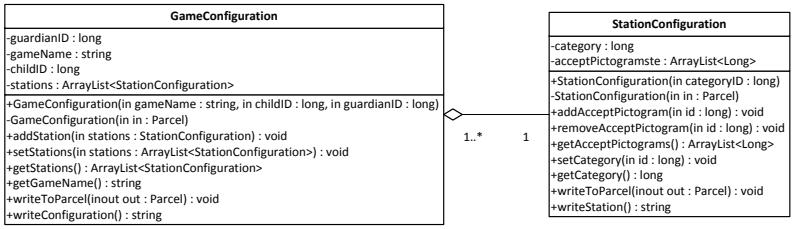


Figure 7.4: Class diagram of the GameConfiguration and StationConfiguration.

7.3 Menu Design

To keep the same look and feel throughout the whole Graphical Interface Resources for Autistic Folk (GIRAF) project, we looked for inspiration in the projects from 2012. The Way Of Measuring Basic Time (WOMBAT) project had the same kind of customization menu we were looking to make, it is shown in Appendix D. We followed the same basic design idea as in WOMBAT, but with a different customization part (the part at the very right). The left half of the screen contains a list of children associated with the current guardian and a list of already saved games. The right half is a customization part where you can add, remove, and edit stations. We chose a list approach for this as well, to keep the whole activity separated in three lists with roughly the same design.

When an application is run from the Launcher, the application receives a color that can be chosen in the Launcher. We use this color to set the overall color theme of the menu, meaning that the whole menu will show in the chosen color from the Launcher. To ease the process of finding the game that you are looking for, the saved games will have a picture attached to them. This picture represents the category of the first station in that game configuration.

7.3.1 Flow

To show the functionalities of the menu, the natural flow will be shown in this section.

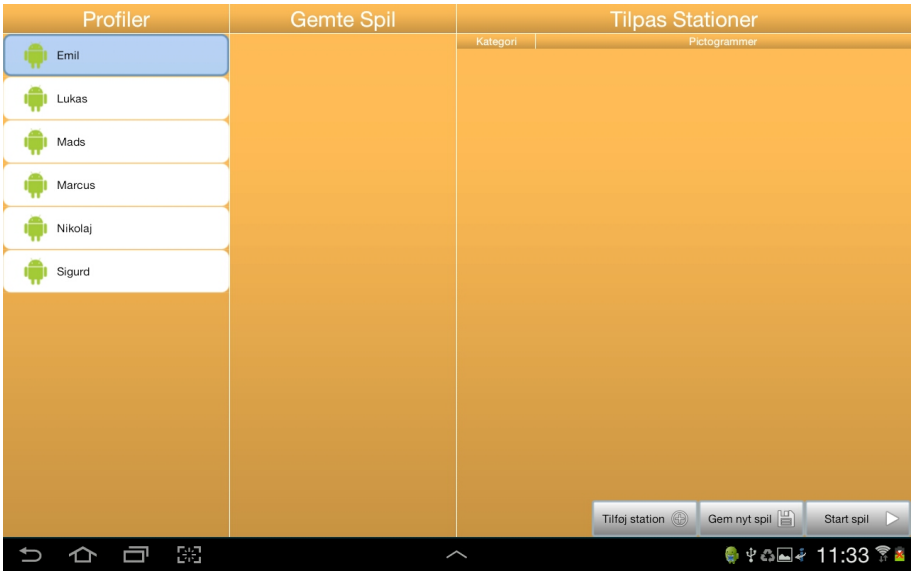


Figure 7.5: The main activity.

Figure 7.5 shows the general design of the menu. The overall color theme (Orange in this example) can be changed from the Launcher. The menu is divided into three parts; profiles, saved games, and game customization. When the application is launched the menu will appear and the profile list will be populated with the children associated to the current guardian. In this example "Emil" has been chosen, but there are no saved games related to that child, yet. We will now continue to push the "Tilføj station" (Add station) button three times.

Figure 7.6: The main activity with three stations.

The game customisation part of the menu has now been populated with three elements, each representing a station. An element/station consists of three buttons:

- **Add category:** Pressing the category pictogram will allow you to choose a different one. It is blank by default to show that no category has been selected.
- **Add pictograms:** Pressing the green "+" will allow you to select the pictograms you want to be associated to the category.
- **Delete station:** Pressing the red "x" will delete the station.

We will now continue to select a category for the first station by pressing the add category button.

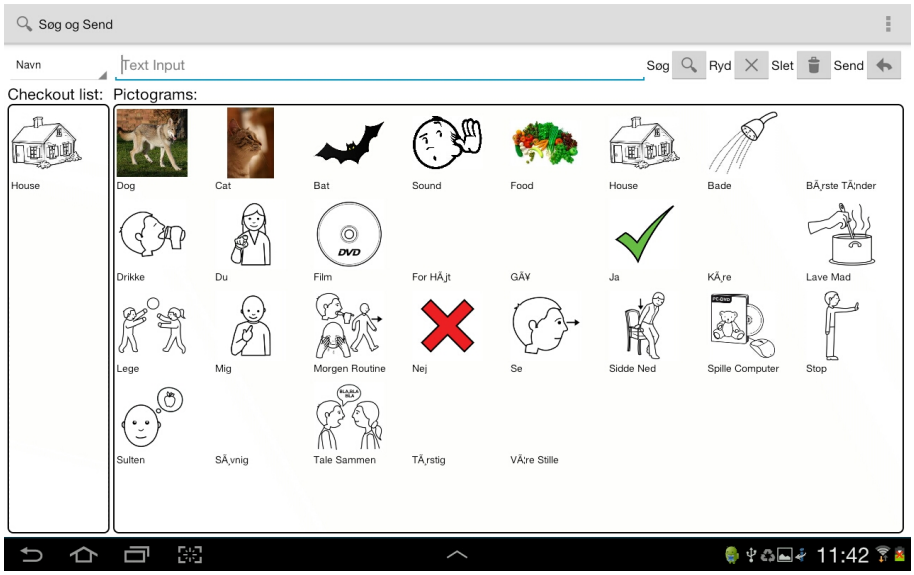


Figure 7.7: CAT with one pictogram selected.

Figure 7.7 shows that CAT opens to allow you to select a pictogram to use for the category. The pictogram "House" has been selected in this example, and we will now continue to press the "Send" button. This will send the pictogram back to the menu.

Figure 7.8: The main activity with one category chosen.

It can now be seen in Figure 7.8 that the pictogram we selected before has been set as the category for the first station. We will now press the green "+" to add the pictograms we want associated with this category.

Figure 7.9: The main activity with one complete station.

It can be seen in Figure 7.9 that two pictograms was selected, again via CAT We will proceed to add categories and pictograms to the remaining stations, using the same method.

Figure 7.10: The main activity with three complete station.

Figure 7.10 shows that the three stations have now been filled with pictograms. Note that the green "+" is gone, this is because we only allow a total of six pictograms. We will now press the "Gem nyt spil" button to save the game.

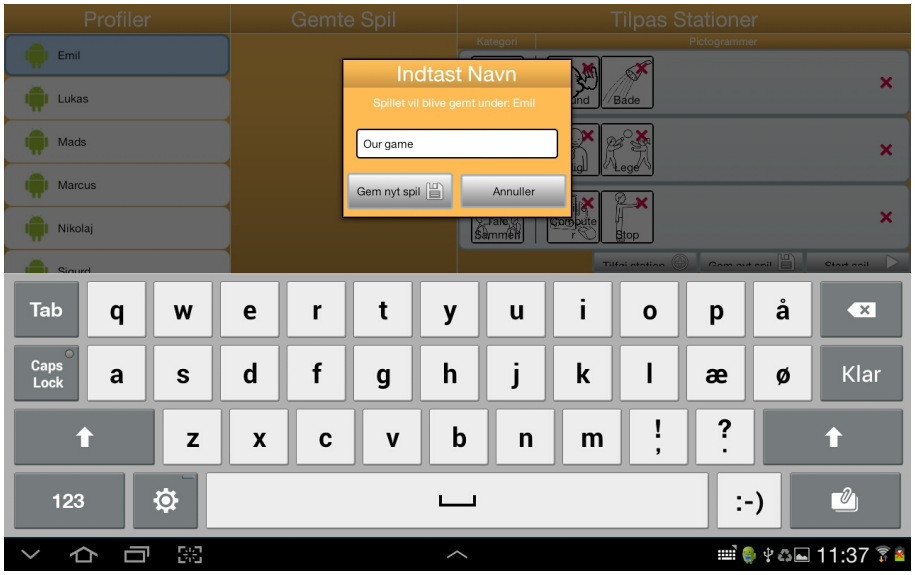


Figure 7.11: Save dialog.

Figure 7.11 shows that a dialog appears when the save game button has been pressed. This dialog allows you to enter a name for your game and save it. We will call our game "Our game" and press "Gem nyt spil" which will save the game.

Figure 7.12: Complete example of the main activity.

The game configuration has now been saved, and is shown in the list of saved games. See Figure 7.12. The newly saved game is linked to "Emil", and will be shown whenever he is selected.

To delete a saved game you will have to do a long click on the game you wish to delete. This will prompt you to whether you are sure or not. If deleted, the game will disappear from the list and the configuration will be deleted from the database.

Activity flow

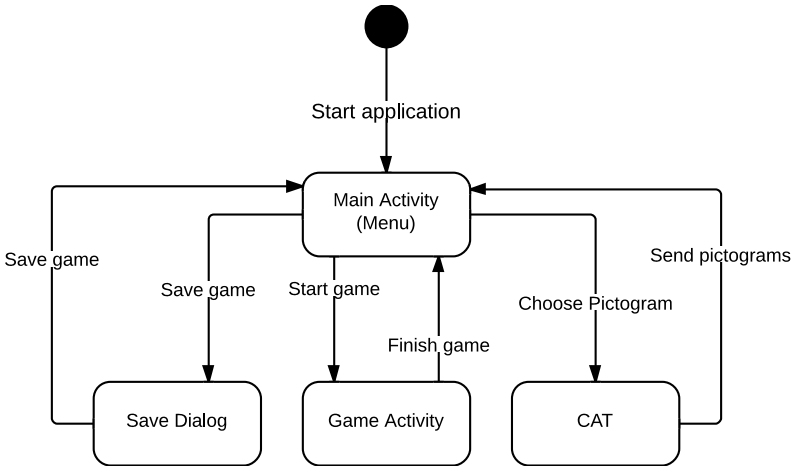


Figure 7.13: The flow of activities

Figure 7.13 shows the relation and flow between the activities in the application. The application will always start in the main activity which is the menu for cus-

tomising the game. From here you have the choice of either saving the game, starting the game, or selecting pictograms. Each of these actions will start another activity. It should be noted that whenever the "back" hardware button is pressed you will return to the previous activity. If you go back from the main activity, the application will terminate.

CHAPTER 8

ANDROID

This chapter explains how we used different components from the Android platform. It also explains how drag and drop works, and discusses the problem we had with ListView, including our solution to the problem.

8.1 Drag and Drop

This section will explain how drag and drop works in Android SDK 11 or greater[1]. It will explain how to start a drag, what happens when a view is dragged on the screen, and finally what happens when the view is dropped.

Overview

Before we begin explaining drag and drop we first need to introduce the terms Listener, View, and ViewGroup.

View is a base class for user interface components. A View represents a square on the device's screen, and is responsible for drawing and event handling. An example of the View class could be an ImageView, which is used to display an image.

ViewGroup is a subclass of the View class, and is a base class for all layouts, which are invisible containers for Views or other ViewGroups.

Listeners are attached to a View and are mostly used to receive and handle user input. Every Listener has a gesture method, this method is called when the system receive a gesture on the View. A simple example of a Listener is the OnTouchListener, The `OnTouch()` method is called every time the OnTouchListener's View receives a touch gesture.

Drag Shadow

Before we can make a drag we must first create something called a drag shadow. A drag shadow is basically a copy of the View you are trying to drag which follows the position of the finger. Listing 8.1 shows how to create a drag shadow and start dragging a View.

```
1  imageView.setOnTouchListener(new OnTouchListener(){
2      public boolean onTouch(View view, MotionEvent motionEvent){
3          if (motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
4              ClipData data = ClipData.newPlainText("", "");
5              DragShadowBuilder shadow = new View.DragShadowBuilder(view);
6              view.startDrag(data, shadow, null, 0);
7              return true;
8          }
9          else {
10             return false;
11         }
12     }
13 });
```

Listing 8.1: How to create a drag shadow, and start drag

Line 1 set an OnTouchListener on the imageView.

Lines 2-3 The OnTouch() method is called by the system when the imageView receives an input gesture, and if the gesture is a press motion it will continue.

Lines 4-5 ClipData is used to move text data. shadow is initialized with the touched imageView.

Line 6 The view.startDrag() starts the drag of the imageView. This will cause OnDragListeners of every View to be called.

Lines 7-10 The OnTouch() method returns either true or false based on if the motion event was handled or not.

Drag Events

An OnDragListener is like the OnTouchListener also attached to a View class. The drag of a View begins when the method view.startDrag() is called. This will cause the system to draw the drag shadow, and call every OnDragListener with an action. During a drag operation it is possible to enter these four states.

Started The OnDragListerner receives a drag event with the action ACTION_DRAG_STARTED. If the OnDragListener returns true it will continue to receive drag events, if it returns false, it will stop receiving drag events, and the View will not be able to accept dragged data.

Dragging If the user continues to drag the drag shadow and enters a View's boundaries, then the system will send a drag event to the View's OnDragListener with the action ACTION_DRAG_ENTERED. When the drag shadow exits a View's boundaries the system will send a drag event with the action ACTION_DRAG_EXITED.

Dropped If the user releases the drag shadow over a View, the View's OnDragListener receives a drag event with the action ACTION_DROP. However the drag event is not sent if the OnDragListener previously returned false. The OnDragListener is expected to return true if the drop was successfully processed, and otherwise false.

Ended After the user has released the drag shadow, all the OnDragListeners which still are registered to receive drag events, receives a drag event with the action ACTION_DRAG_ENDED. This event is done regardless of where the drag shadow has been released.

An implementation and use of drag and drop is shown later in Section 9.1.

8.2 Listview

The menu for customization in this project consists of three lists and it would thus be obvious to use the ListView class from the Android developers guide [2]. This proved to be quite the challenge, since the ListView does not seem to be made for adding and removing items since its initial population. It seems that it is build for showing a dataset in a list form, but if you want to change this dataset during runtime, you run into a lot of problems. There are plenty of examples of people who struggle to use the ListView class and what seemed to cause a lot of our problems is the NotifyDataSetChanged() method from the underlying adapter [11; 5]. The documentation of this method is very vague and we could not get our list to update with new data.

We chose to implement our own list classes for populating the saved game configurations list and the stations list. By using the LinearLayout class from the Android developers guide we are able to add and remove views during runtime [2]. When we put the LinearLayout inside a ScrollView what we get is almost similar to a ListView.

The first list in the menu contains the children, it is made with the ListView class. This could be done since it does not have to be manipulated after the initial load of children. The second list in the menu contains game configurations called GameLinearLayout, it is made with LinearLayout and some of it is described below. The third list in the menu contains station configurations called CustomiseLinearlayout, it is also made with a LinearLayout that does almost the same thing as the GameLinearLayout.

GameLinearLayout

The class we made to show a list of saved games it is called GameLinearLayout.

```
1  private void makeView(GameConfiguration gameConfiguration) {
2      LayoutInflater layoutInflater = (LayoutInflater) getContext().getSystemService(Context.
           LAYOUT_INFLATER_SERVICE);
3      // Use same layout style as the profile list
4      View gameListItem = layoutInflater.inflate(R.layout.game_list_item);
5
6      TextView gameNameTextView = (TextView) gameListItem.findViewById(R.id.gameName);
7      gameNameTextView.setText(gameConfiguration.getGameName());
8
9      ImageView gameIconImageView = (ImageView) gameListItem.findViewById(R.id.gameIcon);
10
11     Bitmap bitmap = BitmapFactory.decodeFile(PictoFactory.INSTANCE.getPictogram(super.
           getContext(),gameConfiguration.getStation(0).getCategory()).getImagePath());
12     gameIconImageView.setImageBitmap(bitmap);
13
14     gameListItem.setOnClickListener(new OnItemClickListener(gameConfiguration));
15     gameListItem.setOnLongClickListener(new OnItemLongClickListener(gameConfiguration));
16
17     // Add to the list of visible configurations
18     this.visibleGameConfigurations.add(gameConfiguration);
19     super.addView(gameListItem);
```

```
20 }
```

Listing 8.2: The method to create a list item.

Line 1 The method makeView() takes a GameConfiguration which contains all relevant information about the saved game.

Lines 2-4 Creates a new View for the GameConfiguration based on our layout for a game list item.

Lines 6-7 Gets the name of the game and sets it in the list item.

Lines 9-12 Gets the category pictogram of the first station in the game configuration and sets this as a game icon for the list item.

Lines 14-15 Set a LongClickListener and a ClickListener on the item. These are both listeners that we have made. The normal click will select the item and show its details in the customization list. The long click will show a dialog that asks you whether you want to delete it or not.

Lines 18-19 Add the item to the list of saved games, and add the view to the LinearLayout.

CHAPTER 9

GAME IMPLEMENTATION

This chapter describes how we have implemented the different elements of our game. It explains how we have implemented the drag and drop functionality. And how we load and draw the texture for the train, how we have implemented the wheels and train smoke in order to create the illusion that the train is moving between the stations, and how we randomly generate the background elements.

9.1 Drag and Drop Implementation

As Section 8.1 shows, the implementation of drag and drop has been simplified after SDK 11. This allows for an easy implementation of the drag and drop functionality. Before the explanation of drag and drop, we will first show the hierarchy of the layouts on the station and the wagons.

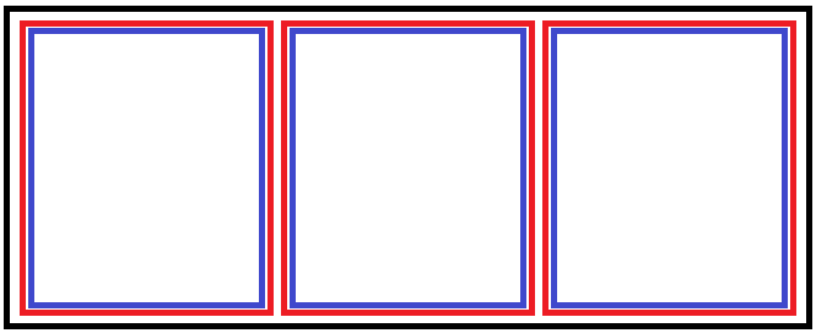


Figure 9.1: Example of a LinearLayout container.

There are two LinearLayout containers on each station, and there is a LinearLayout container on each wagon. An example of the LinearLayout container is shown in Figure 9.1. The black color represents the LinearLayout, the red color represents a Framelayout, and the blue color represents a pictogram. Each pictogram has an attached OnTouchListener, and each FrameLayout has an attached OnDragListener. Our TouchListener is shown below.

```
1 private final class TouchListener implements OnTouchListener {
2     @Override
3     public boolean onTouch(View view, MotionEvent motionEvent) {
```

```
4           if (motionEvent.getAction() == MotionEvent.ACTION_DOWN) {
5               ClipData data = ClipData.newPlainText("", "");
6               DragShadowBuilder shadowBuilder = new View.DragShadowBuilder(view);
7               view.startDrag(data, shadowBuilder, view, 0);
8               view.setVisibility(View.INVISIBLE);
9               return true;
10          }
11          else if (motionEvent.getAction() == MotionEvent.ACTION_UP) {// prevents that a pictogram
                disappears if only pressed and no drag
12              if(view != null && view.getVisibility() == View.INVISIBLE){
13                  view.setVisibility(View.VISIBLE);
14              }
15              return true;
16          }
17          else {
18              return false;
19          }
20      }
21  }
```

Listing 9.1: Our TouchListener

Line 3 The onTouch() method is called when the pictograms receives a touch input.

Lines 4-7 If the touch event is a press motion, make a drag shadow of the pictogram, and start the drag.

Line 8 Set the pictogram's visibility to invisible.

Lines 11-14 If the touch event is a release motion and the pictogram is invisible, then make the pictogram visible. This if sentence was added since it was possible to make the pictogram disappear with only a press motion. If the motion was handled it is expected to return true, otherwise false.

When the method view.startDrag() is called, the OnDragListener event is called. There is attached an OnDragListener to all FrameLayouts on the station and in the wagons. Our OnDragListener is shown below.

```
1   private class DragListener implements OnDragListener {
2       public boolean onDrag(View hoverView, DragEvent event) {
3           View draggedView = (View) event.getLocalState();
4           switch (event.getAction()) {
5               case DragEvent.ACTION_DROP:
6                   // DropAction, assigns the draggedview to the dropContainer if, the dropContainer
                        does not already contain a pictogram.
7                   ViewGroup ownerContainer = (ViewGroup) draggedView.getParent();
8                   PictoFrameLayout dropContainer = (PictoFrameLayout) hoverView;
9                   Object tag = hoverView.getTag();
10                  if (tag == null) {
11                      ownerContainer.removeView(draggedView);
12                      ownerContainer.setTag(null);
13                      dropContainer.addView(draggedView);
14                      dropContainer.setTag("filled");
15                  }
16                  draggedView.setVisibility(View.VISIBLE);
17                  break;
18
19              case DragEvent.ACTION_DRAG_ENDED:
20                  // Makes the draggedview visible again after the view has been moved or if drop wasn't
                        valid.
```

```
21                  if(event.getResult() == false){
22                      draggedView.setVisibility(View.VISIBLE);
23                  }
24                  break;
25              }
26              return true;
27          }
28      }
```

Listing 9.2: Our DragListener

Line 2 The onDrag() method is called when a pictogram starts being dragged.

Line 3 Get the current dragged pictogram and initialize the variable draggedView.

Line 4 Switch case of the current drag event.

Line 5 Case if the current drag event is dropping the pictogram on a FrameLayout.

Lines 7-8 Initialize the variable ownerContainer with the parent Framelayout of the dragged pictogram, and initialize variable dropContainer with the FrameLayout currently being hovered above.

Lines 9-10 Initialize the variable *tag* with the current status of the dropContainer, if the status is "filled" then the dropContainer is already containing a pictogram, and is therefore not eligible for the drop.

Lines 11-16 If the dropContainer does not contain a pictogram, then the pictogram can be moved from ownerContainer to dropContainer. The tag in ownerContainer is set to **null** and the tag in dropContainer is set to "filled", to indicate that it now contains a pictogram. The draggedView becomes visible in the dropContainer after a drop.

Lines 19-21 The current drag event is drag ended, and if no valid was performed the draggedView will become visible again. This will make it look like the pictogram snaps back to its original position.

Some cases has been excluded in Listing 9.2 because they are not relevant for the functionality of the drag and drop.

9.2 Train & Wagons

The train and wagons are stationary textures on the tablet screen, it is actually the background that moves in order to create the illusion that the train is driving. Listing 9.3 shows how we load the texture for our train and wagons.

```
1       // Initialize a train object and a wagon object
2       private final Texture train = new Texture(1.0f, 1.0f);
3       private final Texture wagon = new Texture(1.0f, 1.0f);
4
5       //Load the textures
6       this.wagon.loadTexture(R.drawable.texture_wagon, AspectRatio.BitmapOneToOne);
7       this.train.loadTexture(R.drawable.texture_train, AspectRatio.BitmapOneToOne);
```

Listing 9.3: Loading the texture for our train and wagons.

Lines 2-3 Initialising the train and wagon objects, with the size 1 × 1.

Line 6 This loads the wagon texture, R.drawable.texture_wagon defines which texture gets loaded, which in this case is the wagon and AspectRatio.BitmapOneToOne makes sure the texture keeps the original aspect ratio by resize the texture, as described in Section 6.2.

Line 7 This loads the train texture, and the original aspect ratio is kept again.

Now the texture is successfully loaded we have to place it on right location on the tablet screen. This is done using the coordinate system mentioned in Section 6.1.4. This is done as shown in Listing 9.4.

```
1  //Add coordnates to the renderables
2  this.wagon.addCoordinate(-542.32f, -142.72f, GameData.FOREGROUND);
3  this.wagon.addCoordinate(-187.45f, -142.72f, GameData.FOREGROUND);
4  this.train.addCoordinate(160.42f, -52.37f, GameData.FOREGROUND);
```

Listing 9.4: Placing the texture on the screen.

Lines 2-3 We have chosen to have two wagons on our train, but we only have one wagon object with two coordinates. GameData.Foreground determines the depth in the frustum where the objects are placed.

Line 4 Adds a coordinate for the placement of the train.

Now the textures have been loaded and given coordinates, they are ready to be drawn. This is done in Listing 9.5

```
1  //Drawing the textures
2  super.translateAndDraw(this.wagon);
3  super.translateAndDraw(this.train);
```

Listing 9.5: Drawing the texture on the screen.

Lines 2-3 This draws the wagon and *train* objects. The translateAndDraw() method is explained in Section 6.1.3

9.2.1 Wheels

To create the illusion that the train is moving, we had to make wheels rotate in order to make it look like it was actually driving.

The wheels are loaded and given a coordinate in the same way as the train that was just explained.

The difference comes when the wheels are drawn, we have to rotate the wheels so that it looks like the train moves, this is done by calculating the rotation using the function shown in Listing 9.6.

```
1  private float[] rotation = { 0f, 0f, 0f }; // rotation number for each wheel size
2  private final double[] wheelDiameter = {
3          106.39f, // large wheel
4          78.71f, // medium wheel
5          60.8f // small wheel
6  };
7
```

```
8     private final float calculateRotation(int wheelIndex) {
9         float circumference = this.wheelDiameter[wheelIndex] * Math.PI;
10        float degreePerPixel = 360.0 / circumference;
11        this.rotation[wheelIndex]+= degreePerPixel * super.gameData.getPixelMovement();
12        return this.rotation[wheelIndex];
13    }
```

Listing 9.6: Calculating the wheel rotation.

Line 1 This array has the rotation number for each wheel size.

Lines 2-6 This array has the diameter in pixels for each wheel size.

Line 8 The function takes a wheelIndex as parameter, this is to determine which the wheel size that is being used for calculations.

Line 9 The wheel's circumference is being calculated.

Line 10 Calculating the degreePerPixel by dividing 360 with the circumference

Line 11 Here the rotation for the specific wheel is calculated by taking the degreePerPixel we just found and multiplying it by the pixel movement. GameData .getPixelMovement is explained in Section 6.1.3. Please note that the rotation is not reset, it keeps getting added to, however is it is not necessary to reset it as the rotation will not reach an overflow.

Line 12 The specific wheel's rotation is returned.

This function is called each time a wheel is drawn. This is done by using translateRotateAndDraw() as can be seen in Listing 9.7

```
1 // Rotate and draw the wheels
2 super.translateRotateAndDraw(this.calculateRotation(this.mediumWheelIndex), this.mediumWheel);
3 super.translateRotateAndDraw(this.calculateRotation(this.largeWheelIndex), this.largeWheel);
4 super.translateRotateAndDraw(this.calculateRotation(this.smallWheelIndex), this.smallWheel);
```

Listing 9.7: Rotating and drawing the wheels.

Lines 2-4 Each time a wheel has to be drawn, calculateRotation() with the specific wheelIndex as parameter, is called and the rotation is calculated. The wheel is rotated and then drawn.

9.2.2 Train smoke

Another element in making it look like the train is moving is the train smoke. While the train is standing still at the station the smoke goes vertically up in the air, but when the train moves the smoke moves horizontally.

The smoke clouds are reset and updated based on time, as can be seen in Listing 9.8

```
1 //Reset one smoke cloud at the given interval
2 this.timeSinceLastReset += super.gameData.timeDifference;
3 if (this.timeSinceLastReset >= this.timeBetweenSmokeClouds) {
4     this.timeSinceLastReset = 0f;
5
6     if (!super.gameData.isPaused) {
```

```
7             this.resetOneSmokeCloud();
8         }
9     }
10
11    if (!super.gameData.isPaused) {
12        //Update position and alpha channels
13        this.updateSmokeClouds();
14    }
```

Listing 9.8: Smoke clouds getting reset based on time intervals.

Line 2 The timeSinceLastReset is the time since a smoke cloud was last reset and gameData.timeDifference is the time that has elapsed, in milliseconds.

Lines 2-9 If the time since last reset is higher or equal to the time between the smoke clouds (this is a variable we have pre-determined), then timeSinceLastReset is set to 0, and if the game is not paused, then the smoke cloud is reset back to its starting position.

Lines 11-14 If the game is not paused, then the smoke cloud's coordinates are updated, in order to move the smoke cloud.

Listing 9.8 showed two methods, resetOneSmokeCloud() and updateSmokeClouds(). These are the two methods in charge of updating and resetting the smoke cloud's positions, making it look like they move. They are explained in Listing 9.9 and Listing 9.10

```
1  private final void resetOneSmokeCloud() {
2      //Increment the reset index
3      this.resetIndex = ++this.resetIndex % this.numberOfSmokeClouds;
4
5      //Put cloud back to exhaust
6      this.coordinates[this.resetIndex].setCoordinate(this.startCoordinate.getX(), this.startCoordinate.
           getY());
7      this.colors[this.resetIndex].setColor(1f, 1f, 1f, 1f);
8  }
```

Listing 9.9: Function handling the resetting of a smoke cloud.

Line 3 This resets the index, helping to keep track of which cloud that has to be reset.

Lines 6-7 This resets the specific cloud back to its starting position and it also resets the color back to its original.

```
1  private final void updateSmokeClouds() {
2      //Updates position and alpha channels
3      for (int i = 0; i < this.numberOfSmokeClouds; i++) {
4          //Always move smoke vertically, move smoke horizontally relative to the train speed.
5          this.coordinates[i].moveX(super.gameData.getPixelMovement());
6          this.coordinates[i].moveY(this.ySpeed * super.gameData.timeDifference);
7
8          //Fade the smoke
9          this.colors[i].alpha -= (1f / (this.timeBetweenSmokeClouds * this.numberOfSmokeClouds)) *
               super.gameData.timeDifference;
10     }
11 }
```

Listing 9.10: Function handling the updating of a smoke cloud.

Lines 3-6 This is in charge of moving the clouds. The clouds are moved on both the x and y axis. The new x coordinate depends on gameData .getPixelMovement() which is explained in Section 6.1.3. The new y coordinate depends on a constant called ySpeed, and gameData.timeDifference, which is the time that has elapsed.

Line 9 This fades the smoke clouds. The cloud's original alpha channel is one, and each time the smoke cloud is updated and moved, the alpha color lowered, making the cloud more see-through, to make it look like they disappear up in the air.

9.3 Station

We have made three different station textures and to ensure an element of surprise in our game, each time we start the game we shuffle the list of stations so the order always will be random, however this only happens when the game is started, so the order within the same game is the same. Listing 9.11 shows this implementation.

```
1  Collections.shuffle(stations);
2  LinkedList<StationContainer> stationsQueue = this.getQueue(stations);
3
4  float xPosition = -364f; // first platform position
5
6  for (int i = 0; i < super.gameData.numberOfStations; i++) {
7      StationContainer nextStation = stationsQueue.pop();
8      stationsQueue.add(nextStation);
9      this.stationPlatformMatrix.addRenderableMatrixItem(nextStation.station, new Coordinate(
           xPosition + nextStation.xOffset, nextStation.yOffset, 0f));
10
11     xPosition += GameData.DISTANCE_BETWEEN_STATIONS;
12
13 }
```

Listing 9.11: Placing the station correctly.

Lines 1-2 The stations are shuffled and added to the queue.

Line 4 xPosition is the position in which the platform will be drawn.

Line 7-8 The next station is found and popped off the queue, added to the end of the queue again to ensure that there are always are stations in the queue.

Line 9 The station is added to the stationPlatformMatrix and drawn at the correct location. All stations have the same offset to ensure that they are all placed at the same location each time the train stops.

Line 11 xPosition is updated with the pre-determined distance between stations to ensure each station has the same distance between them. This is especially important when we have to calculate when the train has to stop.

9.3.1 Stopping position

In order for us to stop the train at the correct location each time, we have to calculate the stopping position, this is done as shown in Listing 9.12.

```
1  public final void calculateStoppingPositions() {
2      //Make new array
3      super.gameData.nextStoppingPosition = new float[super.gameData.numberOfStations + 1];
4
5      //Calculate all stopping positions
6      super.gameData.nextStoppingPosition[0] = GameData.DISTANCE_BETWEEN_STATIONS;
7      for (int i = 1; i < super.gameData.numberOfStations; i++) {
8          super.gameData.nextStoppingPosition[i] += super.gameData.nextStoppingPosition[i-1] +
               GameData.DISTANCE_BETWEEN_STATIONS;
9      }
10 }
```

Listing 9.12: Calculating the next stopping position.

Line 2 An array for the stopping positions is initialized. The array is +1 to ensure an ArrayIndexOutofBoundsException will not occur when calculating the stopping position for the train depot mentioned in Section 7.1.1.

Line 6 The first stopping position is equal to the pre-determined distance between stations.

Lines 7-9 We loop through the array and calculate the next stopping position for all stations. This is done by adding the distance to the next station to the stopping position of the previous station.

9.4 Game Background

The background for our game is randomly generated for each game session as mentioned in Section 7.1.3

9.4.1 Hills

We created four different sequences of hills, each sequence consists of a number of hills in a specific layout. Each sequence is static, so the layout of each of them will remain the same each time the game is played. These four sequences are randomly chosen throughout the game. It is important to note that every hill texture is only loaded once, even though it is used multiple times in the game. Figure 9.2 shows what the four different hills looks like in the game.

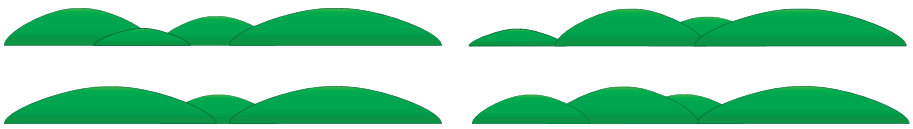


Figure 9.2: The four different hill sequences

HillItem is a class used for making hill sequences. Its constructor takes three parameters, an x-coordinate, a y-coordinate and a texture. The x- and y-coordinates are used for placement of the texture. Each HillItem is added to the renderablematrix before it is drawn. Listing 9.13 shows one of the hill sequences used in our game.

```
1 HillItem[] hillSequence1 = {
2     new HillItem(0f, this.hill_large.getHeight(), this.hill_large),
3     new HillItem(this.hill_large.getWidth()+11f, this.hill_medium.getHeight(), this.hill_medium),
4     new HillItem(this.hill_large.getWidth()/100f*60f, this.hill_small.getHeight(), this.hill_small),
5     new HillItem(this.hill_large.getWidth()+11f + this.hill_medium.getWidth()/100f*65f, this.
          hill_larger.getHeight(), this.hill_larger)
6 };
```

Listing 9.13: Hill sequence.

Line 2 As a starting point our hills are placed in the bottom left corner, so the first hill has the x-coordinate of 0, and because we draw the texture from the top left corner, we ensure that the hill is placed correctly by making the y-coordinate equal the height of the hill.

Line 3 The next hill is placed relative to the first hill, the x-coordinate is the width of the hill that came before it plus a constant. The y-coordinate equals the height of the hill again.

Line 4 The x-coordinate for the third hill is a percentage of the width of the first hill.

Line 5 The last hill is placed relative to the first two hills.

An important thing to note about the placement of the hills is the order in which they are added to the sequence, The hill furthest to the back has to be drawn first, then the second furthest and so on. This is also why the third hill in the sequence is drawn with a smaller x-coordinate than the second hill in the sequence.

9.4.2 Trees & Cows

The trees and cows are randomly generated as well, however they are implemented using more layers of randomization. Listing 9.14 shows the important elements of this.

```
1 private Texture[] decorationArray = {cow, tree, tree};
2
3 drawDecoration = super.gameDrawer.getRandomNumber(0, 2); // Randomly chosen which
      decoration to draw (cow = 0, tree = 1, tree = 2)
4 chooseHillSequence = super.gameDrawer.getRandomNumber(0, hillSequences.length-1); //
      randomly choose which hill sequence to draw
5 chooseHillToDrawOn = super.gameDrawer.getRandomNumber(1, 4); // randomly choose which hill to
       draw on (small_hill = 1, medium_hill = 2, large_hill = 3, larger_hill = 4)
6
7 for (HillItem hillItem : hillSequences[chooseHillSequence]){
8     this.sequence.addRenderableMatrixItem(hillItem.hill, new Coordinate(i + hillItem.x, hillItem.y, 0f)
          );
```

Listing 9.14: Selection of background.

Line 1 We have an array consisting of the cow texture and the tree texture. We chose to have one cow texture and two tree textures so that trees would have a chance of occurring more often. We have several random factors that play in before we find where to place it.

Line 3 First we randomly choose which of the four hill sequences we want to draw, this is because we do not want cows and trees on every hill as they are meant to be an element of surprise for the player.

Line 4 Now we know which hill sequence we draw, we have to find out what to draw, this is also chosen randomly. We randomly find a number between 0 and 2 and use this as index in our decorationArray.

Line 5 We now know what to draw and on which hill sequence to draw it, but we still need to find out which hill in that sequence to draw it on, because we do not want to draw on every hill in the specific sequence. This is again chosen with a random number, which is used to choose a random case.

Lines 7-8 We add each hill in the sequence to the matrix so they can be drawn in a loop.

Now it gets a little complicated, so far we have a random number which determines which hill sequence we have to draw, we also have a random number which determines whether a cow or trees should be drawn on this hill sequence and we have a random number deciding which specific hill in this sequence to draw on.

As explained before, each HillItem is added to the renderablematrix before being drawn, which allows us to check which hill is being drawn next. We set up four cases, one for each type of hill, hillItem.hill has to correspond to the case chosen by the random number, if this is the case, then there will be drawn on the hill, if not then nothing is drawn and the loop continues with the next iteration.

For example if the three numbers that have been chosen are one, one, and three. This means that the hill sequence we will draw is hill sequence one and we have to draw a cow on a large hill in this sequence.

The first hill in this sequence is added to the matrix, and then two scenarios can occur, either hillItem.hill is a large hill and the cow is drawn on the large hill, or hillItem .hill is not a large hill and nothing is drawn on the hill and new random numbers are chosen and everything is checked again. Appendix C shows different examples of a full backgrounds generated in a game.

9.4.3 Clouds & sun

The clouds are also implemented with a degree of randomness. There will always be clouds on the sky, however the speed and placement on the sky is random to a certain degree.

The clouds move on the x-axis with a random velocity between 0.07 and 0.18 pixels per millisecond, the clouds are also randomly placed on the top 1/4th of the screen.

The sun is a stationary texture in the top right corner of the screen.

CHAPTER 10

MULTI-PROJECT ASPECTS

Being a part of a multi-project was a new thing for us. The biggest difference was that we were depending on other groups, this has been a significant change compared to other projects. Due to the nature of our project, none of the other groups were dependant on us so the decisions we made did not effect anyone else but us. Some of the other groups were however depended on each other, so a form of controlled decision making process had to be established.

Agile development Throughout the project we have used an agile development method, which worked well for us. Always having a sprint deadline helped us focus and be on time.

As a part of the agile development method we had weekly meetings. They helped keeping track of what all the other groups were doing. The weekly meetings were also great for discussing different aspects of the project. At the start of the project, the meetings helped getting the project organised with the creation of committees for different assignments. However the further we got with the project, the more unnecessary the meetings became, since all the basic stuff was already determined. This resulted typically with same outcome as the previous meetings. We felt the meetings started to feel more like a chore than like a help.

With that being said, the meetings have to be there, they are a part of the agile development method and they are extremely helpful in the start-up phase of the project, however at the end of the project they might only have to be biweekly.

Committees The committees were a good idea, the subjects that concerned all groups were discussed with a representative from each group, and then presented at the next weekly meeting where everyone had to vote, to either accept or decline the idea. A list of the important committees can be found in Section 2.4.3

E-mails During the project we received a lot of different e-mails, and it was sometimes confusing to see if they had anything to do with the project. Later in the project it was suggested that every e-mail that concerned the project would have to be tagged with "GIRAF" in the subject, allowing us to filter the incoming e-mails. This can be highly recommended.

Roles In the beginning of the project several different roles were assigned to different people by volunteering. These roles came with an area of responsibility. They worked very well, but unfortunately some groups ended up having more than one role, which meant they had a lot of work which was focused on the multi-project and less on their own. It could be an idea to simply assign these roles to each of the projects before making the groups so that the workload was equally assigned.

Some advices for next year; Split all the roles up between all project groups so that the workload is equally assigned. Keep the meetings short and precise, too much unnecessary talk makes the meetings long and frustrating.

CHAPTER 11

CONCLUSION

Our application was developed as a part of the GIRAF project 2013, a multi-project consisting of eight project groups, each with their own sub-project. Our focus in this project was to develop a game for GIRAF.

The problem definition in Chapter 4 states:

> *"In what ways can we aid the pedagogues in their work with children with autism, by digitalizing a physical exercise onto an Android tablet?"*

The exercise we chose was an exercise that our contact person, Tove Søby, had presented during the first week of the project. The child had to place pictograms onto a train and unload them at the correct stations, this already allowed for great customization but required a lot of work from Tove Søby, so it was an ideal exercise for us to try and implement this on the Android platform.

The main focus with our application was to make it customizable for the guardian, so that they easily can create new and different games for each child.

We achieved this by creating a menu using LinearLayouts so they are possible to edit on runtime. The guardian is able to choose a specific child from the list and start creating a new game for that child. Using CAT they are able to choose the pictogram they want as category for each station and what pictograms they want associated with each station. This allows for customization with very limited effort from their side, which is what we wanted to achieve.

The different graphical elements in the graphical side of the application is drawn in vector graphics, this makes it possible to resize the graphics in the future without a loss of quality.

We have also implemented a whistle sound when the train starts driving and different random elements that can occur throughout the game, such as cows and trees on the hills in the background. This was done to create an element of surprise, which is something Tove Søby mentioned the children really liked.

Throughout the project we have communicated with Tove Søby for feedback and ideas and we have shown her and had her try out the application, however we do not have any official usability tests to document that our application is satisfactory, but we have managed to develop an application to aid the pedagogues in their work with autistic children, which was our goal for this project.

CHAPTER 12

FUTURE WORK

Unfortunately due to time limitations we have not been able to implement all the features that our contact person suggested. This Chapter describes some of the improvements that could be made on our application.

The sun The sun should have visible sunbeams, making it shine.

Overwrite games At the moment it is not possible to overwrite an old game, each time you click the "Gem nyt Spil" button, you save a new instance of a game even if you have changed an old game. There should be a "Gem Spil" button, which allows you to overwrite existing games in order to make it easier for the user to edit and update them.

Sounds on pictogram The pictograms should have sounds, this could either be when you move them around or when you press them.

Sound effects in game More sound effects in the game. This could be the cow mooing when the train passes it, wind blowing or the sound of the train accelerating when it starts from the station.

Customizable All colors in the game are pre-determined right now. A settings menu where it is possible to change the color of items in the game, for example the color of the train or station. In case more sounds are implemented it could be possible to mute and unmute the different sounds, in order to make it more customizable for the guardian.

Child as train driver Every child has an avatar picture, it could be funny to have the child's avatar as the train driver.

Make it clear how to delete a game It is possible to delete saved games by performing a long click, however this is not mentioned any where. Perhaps a little information button somewhere that explains this could be a good idea.

Greater variation of background features The selection of background features is very limited, it can only be cows or trees. A greater selection of items could be implemented, perhaps to include planes, different kinds of animals or even humans. It should still be kept simple, but it would create a greater variation in the background.

Complete a station in one go At the moment when creating new stations in the menu section, the user has to click the blank category box, open the CAT, choose one category pictogram and click the "Send" button, then click the green plus and choose which pictograms to associate with the station.

An easier way for this could be to allow the user to click the blank category box and open PictoSearch and then allow them to choose up to seven pictograms, where the first pictogram would count as the category and the rest would be the pictograms associated with this specific station.

Changing weather One of the suggestions made by our contact person was that the weather could change while the train moved from one station to another, for example the sun could shine from the first to the second station and it could rain or snow from the second to the third station.

Layouts disappear when drag starts At the beginning of each game, and each time you arrive at a station, the first pictogram you drag will make the layouts on the station disappear while the pictogram is being dragged. We have a workaround for this but it creates a lot of overhead. Each time we start a drag event, we draw all the layouts again.

Reduce drag events While dragging a pictogram the pictogram shakes. Too many dragging events happens while the pictogram is being dragged, this should be reduced. It also uses resources that OpenGL ES could use for higher frame rate.

Customization optimisation The lists for children, saved games, and stations were created under pressure of time. When you load a game, a lot of garbage is created from the game that was loaded before that.

Dragging of pictograms in OpenGL ES Instead of having regular views and OpenGL ES at the same time, it would be better if everything was implemented in OpenGL ES. There is some noticeable delay with the communication between the two, especially when pictograms are taken from the views and rendered in OpenGL ES.

Database The database was not fully implemented and because of that we save the game configurations to a file. When the database is implemented the following queries will be able to read and write data respectively.

```
{
    "auth": {
        "session": SESSION_STRING
    },
    "action": "read",
    "data": {
        "type": "application",
        "view": "list",
        "ids": null
    }
}
```

Listing 12.1: JSON query to read application data.

```
{
    "auth": {
        "session": SESSION_STRING
    },
    "action": "link",
    "data": {
        "profile'': PROFILE_ID,
        "link": [
            {
                "type": "application",
                "id": APPLICATION_ID,
                "settings": SETTINGS_STRING
            }
        ]
    }
}
```

Listing 12.2: JSON query to write application data.

Usability test Performing a usability test on our application would help spotting possible flaws in the design and help making it even easier and better for the guardians to use.

APPENDIX

GAME CONSTANTS

Constant	Value	
Foreground depth	907.7443	
Background depth	1800	
Max train velocity	0.35	p/ms
Distance between stations	12000	p
Distance to depot	5000	p
Acceleration time	5000	ms
Number of clouds	5	
Number of exhaust clouds	5	
Exhaust vertical velocity	0.15	p/ms
Large wheel diameter	106.39	p
Medium wheel diameter	78.71	p
Small wheel diameter	60.8	p

APPENDIX B

GAME TEXTURE

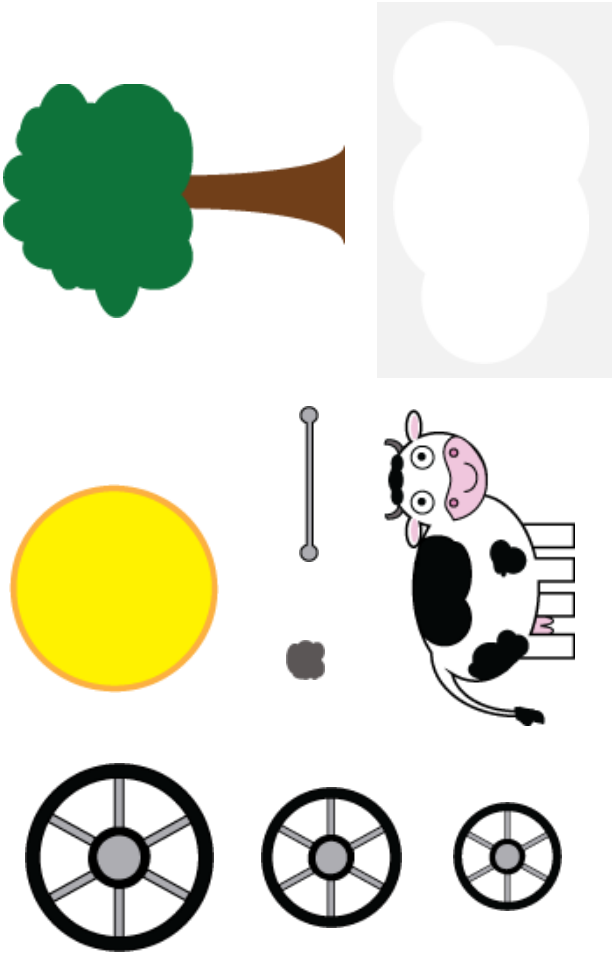


Figure B.1: Small texture.

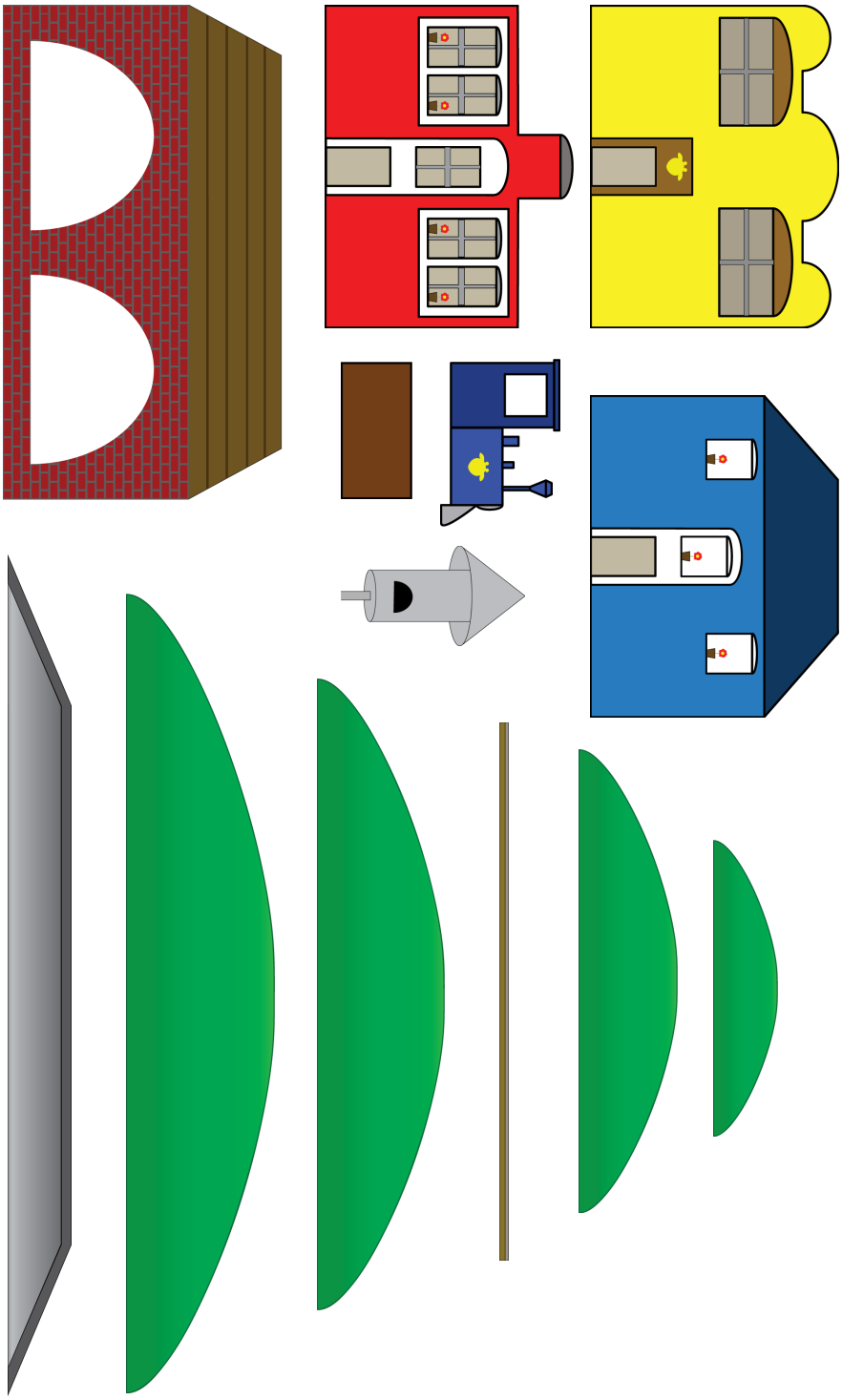


Figure B.2: Big texture.

APPENDIX C

GAME BACKGROUND

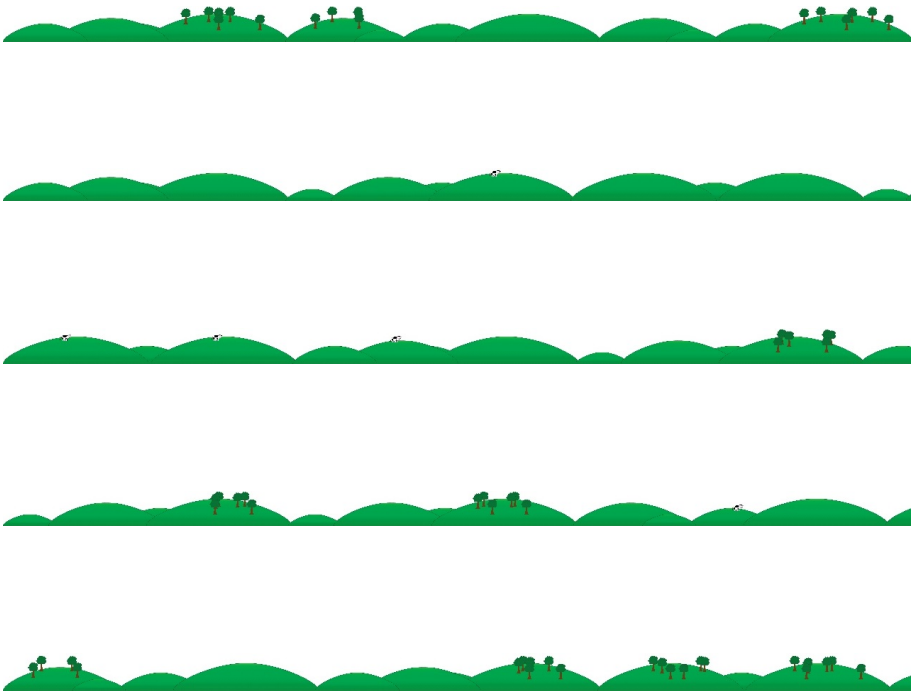


Figure C.1: Examples of different landscapes generated in our game.

APPENDIX D

WOMBAT DESIGN

Figure D.1: The wombat application.

LIST OF FIGURES

LIST OF LISTINGS

BIBLIOGRAPHY

[1] Android. Guide - drag and drop, 2013. http://developer.android.com/guide/topics/ui/drag-drop.html.

[2] Android. Layouts, 2013. http://developer.android.com/guide/topics/ui/declaring-layout.html.

[3] Android. Opengl, 2013. http://developer.android.com/guide/topics/graphics/opengl.html.

[4] Android. Glutils, 2013. http://developer.android.com/reference/android/opengl/GLUtils.html.

[5] Android. Notifydatasetchanged method, 2013. http://developer.android.com/reference/android/widget/ArrayAdapter.html#notifyDataSetChanged().

[6] Android. Parcel class overview, 2013. http://developer.android.com/reference/android/os/Parcel.html.

[7] American Accreditation HealthCare Commision. Autism, May 2012. http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0002494/.

[8] The Khronos Group. Opengl es - the standard for embedded accelerated 3d graphics, 2013. http://www.khronos.org/opengles/.

[9] NATUREWIZARD. Tutorial 1 - opengl fundamentals, 2013. http://www.naturewizard.com/tutorial0103.html.

[10] Sean O'Neil. Can opengl es render textures of non base 2 dimensions?, December 2010. http://stackoverflow.com/a/4329720.

[11] Stack Overflow. Listview on stack overflow, 2013. http://stackoverflow.com/questions/tagged/android+listview.

[12] Mark Segal and Kurt Akeley. The opengl graphics system, October 2004. http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf.

[13] Pyramid Droup Management Services. Picture your student learning. Website, Visited 21.03.2013. http://pecs.com.