P6 PROJECT:

Train and PictoSearch

In 2014 Multi-Project

Group: **sw611f14**

Date: **May 27th 2014**

AALBORG UNIVERSITET
Student Report

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
Telephone: +45 9940 9940
Telefax: +45 9940 9798
http://cs.aau.dk

Title:

Train and PictoSearch in 2014 Multi-Project

Theme:

Multi-Project

Project period:
P6, Spring term 2014

Project group:
sw611f14

Participants:
Minh Hieu Nguyen
Jonas Martin Kærlev
Tobias Frandsen

Supervisor:
Saulius Samulevicius

Page count: 40

Finished on May 27th 2014

Abstract:

This paper documents the intentions of creating an application-suite "multi-project" with the goal of aiding children with autism and their guardians. This paper describes the development of two applications within the multi-project: Train and PictoSearch. Train, an interactive game based on categorizing images "pictograms", was resumed from previous development, and was during the development of this paper expanded to include additional functionality. PictoSearch, an application meant to aid other applications in retrieving pictograms from the multi-project database, was significantly improved during the development. It went from a state of barely working to being near feature-complete.

Signatures

Minh Hieu Nguyen

Jonas Kærlev

Tobias Frandsen

Contents

Table of abbreviations

Abbreviations	**Significations of the abbreviations**
Tortoise	Life stories
GUI	Graphical user interface library
Timer	Stopwatch and countdown functionality
Lancher	Handles launching all other applications
Giraf Admin	Web administration tools
Sequences	Arranging pictograms to schedule and show a sequence of actions
CROC	Drawing new pictograms
Cat	Manages pictogram categories
PictoSearch	Searching for specific pictograms
RemoteDB	Online storage of pictograms and categories
Oasis	Pictogram manager
Parrot	Audio reader for pictograms
Cars	Visual voice-controlled game
Train	Visual game about matching pictograms and categories
Visual Scheduler on iOS	Identical to Sequences but for iOS
iOS Speak for Me	Identical to Parrot but for iOS

0

Introduction

0.1 6th Semester Multi-Project

As part of the 6th Software semester project of 2014 for Aalborg University, all of the semester Software students were to collaborate on a global project split into several applications, called the *Multi-Project*. The multi-project aimed to create a tool for faculties taking care of autistic children, and provide digital versions of existing methods for taking care of autistic children (such as schedule planning, communication, learning games etc.). For this project, students were divided into groups of three to four individuals and then assigned an application to work on. Some applications were intertwined, which required specific groups to collaborate. The multi-project was a project that had been in development for several years, and as such, the applications were not started from scratch. Instead, applications were partially or fully developed, and the work would be put into fixing issues or adding new functionality.

The key objective of this semester was having, at minimum, a product that could be used, as well as collaborating with other groups to ensure that all applications are working properly together. It is the desired goal that by the end of the semester, the product could be taken into use by the faculties.

0.1.1 Applications and projects of the multi-project

The applications and projects of the multi-project are listed below, and a small resume of each application or project is given to provide an overview of the multi-project.

Tortoise is an application used to create life stories. The idea of life stories is for the kids to create a story from pictograms, to show their parents at home what they have been doing or learned at school etc.

A GUI group is assigned to creating a graphical user interface library for all the other projects to use, so all the applications will have the same look and feel when being used.

Timer is an application where you can create timers used for managing a kids assigned play time etc. As an example you can set a timer for 20 minutes and start a game, then the game will shut down after 20 minutes.

Launcher is the Android launcher which is created specifically for this multi-project. The launcher will contain the login system, different settings like colors and you can access all applications from it.

Giraf Admin is the online administration tool. It will be used to create and manage profiles.

Sequences is a scheduling application where you can create a sequence of pictograms. A sequence is used to show the kids in what order to do things and work like a schedule for them.

CROC is the application you can use to create pictograms. You can draw a picture or use the camera and add text and sound to it. The pictogram is afterwards saved in the database so other applications can use it.

RemoteDB is the online database containing the full selection of pictograms and categories.

LocalDB is the local database which holds the offline data on the tablet. The localDB and remoteDB synchronizes the data betweem them.

PictoSearch is the application which is used whenever you need to find a pictogram or category in the database. You launch PictoSearch from other applications and search in the database, and afterwards return any chosen pictograms or categories.

Cat is an application used to create categories. A category contains multiple pictograms and can also contain subcatories. The categories are also save to the database for usage by other applications. This group also manages the category library named *cat-lib* which is used when communicating with LocalDB about categories.

Parrot is an application where you can find pictograms and get their attached sound played out loud. Many of the kids do not have the ability to communicate using words, so they use pictograms to show the teachers what they want to do. This group also have to manage the pictogram library *picto-lib*, which is used when communicating with LocalDB about pictograms.

Oasis is the main library used when having to communicate with the database. Both cat-lib and picto-lib goes through oasis-lib when communicating with LocalDB. Oasis also has an application to manage, which is used as a first time setup where all the data is added to the localDB.

Cars is a visual game where the players has to use the level of their voice to control a car through obstacles and get it parked correctly. The purpose is to learn the kids about controlling the level of their voice, to help them get a feeling of how loud they are talking.

Train is visual game where the purpose is to categorize pictograms under the correct topics. Train wagons are filled with pictograms and whenever the Train stops at a station the player has to load off the pictograms matching the topic of the station.

Visual scheduler on IOS is an application with the same functionality as Sequences but ported to run on an iPad device.

IOS speak for me is an application with the same functionality as Parrot but ported to run on an iPad device as well.

This paper will not go in-depth about each individual application, but will focus on the applications relevant for this project group. Train is elaborated in Chapter 1 on page 5 and PictoSearch is elaborated in Chapter 2 on page 10.

1 First Sprint - Train

1.1 Status of Train

In the first sprint of the multi-project, all semester groups were to select a project to work on. In this group, we prioritized getting access to one of the game applications (Train, Cars, etc.). This was because we had in previous semesters found it fun and interesting to work with interactive games. Several other groups had also shown interest in the few game applications. When the projects were handed out, this group was lucky enough to end up with the *Train* project.

Train was a project developed during the 2013 GIRAF multi-project as a digitalized version of an existing physical exercise [1].

When the project period ended in 2013, the Train application developed as part of the project was in a playable state. The application was tested at selected autism institutions and the feedback was positive with no major complaints. In the further development statement of the 2013 project, minor issues and improvements were mentioned. These issues were the focus of this first sprint.

In addition to these issues, new issues were found when trying to launch the application. As part of this 2014 project, the development environment was changed from *Eclipse* to *Android Studio*. This change was due to the fact that previous years had found it difficult to set up Eclipse for Android development, whereas Android Studio would allow for a seemingly easier setup. The migration of project files were performed by a few dedicated individuals.

This caused numerous issues, such as the *UI* (user interface) being misaligned, as well as the emulator being unable to execute the application (due to difference in Android SDK versions).

After the launch issues were resolved (mentioned further in the next section), it was possible to diagnose additional missing features.

At this point, the issues and features were prioritized into the following list:

- The Train application would be unable to start in Android Studio
- The minimum required Android SDK version should have been 15 instead of 17 (a requirement by the 2014 project)
- The application would halt with an uneven number of pictograms

- The number of pictograms was restricted to 6 but this was never communicated
- The UI was misaligned on some resolutions, notably in the emulator
- Game rules were ambiguous and a help feature was needed
- The pictograms were invisible at times

This sprint will aim to fix and implement as many of these as possible.

1.2 Implementation of first sprint

In the first implementation sprint of Train, the goal was to ensure that the application could be built in Android Studio and work without major errors. The implementation section is divided into two parts. The first part is about our problems with starting the Train application in Android Studio. The second part describes the errors that were solved and the functionalities that were added to the existing application.

1.2.1 Starting the Train application

In the early multi-project meetings, it was decided to port the project from Eclipse to Android Studio. At the start of the first sprint, we were presented with the problem of importing the Train project into Android Studio which resulted in a few problems. The Train project could not be ported to Android Studio as it was missing some Android files which were needed to build the project. We got these files from another project in the multi-project where things were already ported and working in Android Studio.

It was decided to change the minimum SDK to Android API level 15 from level 17, as we wanted to make sure it works on older versions of Android tablets. To do this we just needed to install the SDK for API level 15 and change the project to use that instead of level 17.

After running the application, we got stuck at the main menu of the application. The application had error messages which stated that the CAT application was not installed when we wanted to add categories and pictograms to train stations. This lead to a lot of work trying to import and install the CAT project and make it work with the Train application. The missing application was not the CAT application, but instead the PictoSearch application. With the PictoSearch application, we just needed to add the local database containing the categories and pictograms to get past the main menu of the Train application. The error message was changed to indicate that the PictoSearch application was missing.

1.2.2 Functionality

Many of the issues and features which was prioritized in section 1.1 have been addressed this sprint. The functionalities which was solved or implementation are listed below.

Number of Pictograms The restraint of only six pictograms was not communicated to the end user. In an attempt remove this restraint, we made the application allow 12 pictograms. This restraint was made because of limited space on the screen and the pictograms would have to be made smaller if we wanted more which will be harder to see and read.

Odd number of Pictograms After increasing the number of allowed pictograms, the application crashes when assigning an odd number of pictograms. This was due to memory allocation and was fixed by rounding up to even number.

UI on different resolutions The Train application had a working UI on the test tablet. When using an emulator the UI was misaligned as it used a different resolution (and as such, a different xml file called *layout_large*). This made it difficult to test the features of the application. The issue was resolved by making the *layout_large* file match what the tablet was using, which was the working layout *layout_xlarge*.

Added help instructions This was implemented by adding a help button at the top left corner of the screen, and when clicked a textbox with the instructions would appear. This can also be seen on figure 1.2.

Invisible pictograms This problem was investigated, however it was estimated that it would take more time to resolve than what would be available of time for the remaining duration of the first sprint. Initial findings show that it is likely an issue when sending Android UI to OpenGL.

Screenshots of the Train application on the test tablet can be on the following figures 1.1 and 1.2.

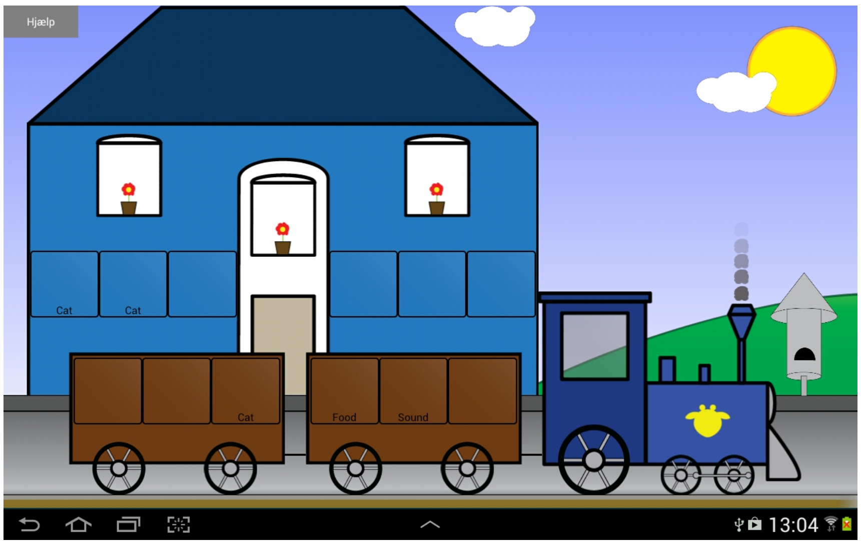
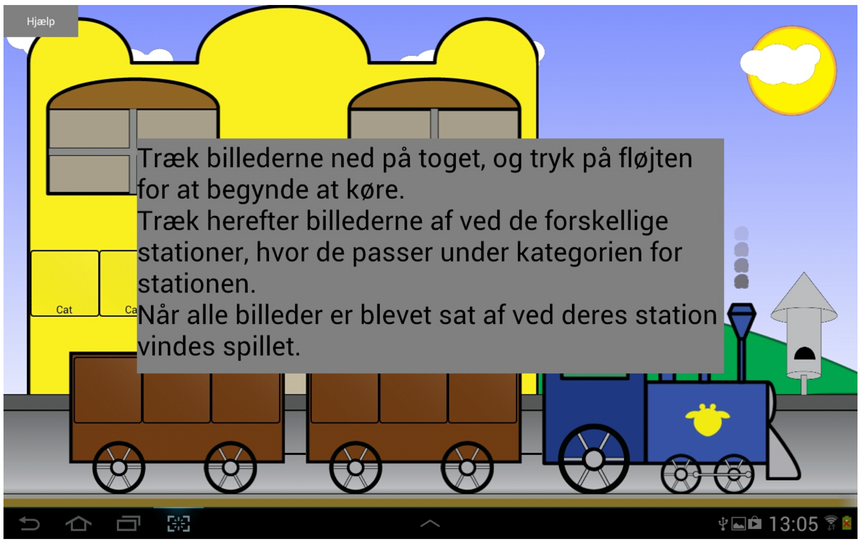


Figure 1.1: Screenshot of how the game looks now. Taken from a Samsung Galaxy Tab GT-P7510.



Træk billederne ned på toget, og tryk på fløjten for at begynde at køre.
Træk herefter billederne af ved de forskellige stationer, hvor de passer under kategorien for stationen.
Når alle billeder er blevet sat af ved deres station vindes spillet.

Figure 1.2: Screenshot of how the instructions look.

1.3 Sprint-end

For the first sprint-end, the Train application was at a presentable state. The customers were presented with the current version of Train and were satisfied with the bug fixes and the new changes. The concerns and wishes received from the customers were:

- The customers were concerned that the pictogram seizes might become too small if too many pictograms were used
- They would have liked the possibility of having more wagons
- They wanted to be able to set the time between each stations
- The customer mentioned re-implementing the help feature as an animation instead of text explanation

1.4 Conclusion

In this first sprint we have been working on the learning game Train, which was one of the projects the closest to being a finished project in the the previous 2013 multi project. However, we did find issues that we were able to fix properly, after getting everything to run in Android Studio instead of Eclipse. We made sure the game could run on Android API level 15, which was a requirement for all the products in the 2014 multi-project. In addition, issues regarding the UI were fixed where some pictures and buttons did not always scale and align properly. A few minor improvements were added to the game, such as making it possible to add more pictograms at a time, as well as adding instructions to game.

Train is what could be considered a working product and runs as intended. The only major issue is the lack of pictograms in the entire multi-project, however this is not an issue with Train. Rather, it is due to the fact that Train is using a test sample database, as no proper database has been set up for this sprint. When the full database of pictograms is ready for use, Train will be ready for testing in the institutions.

1.5 Further Development

In future sprints Train could be further developed by fixing any remaining issues, as well as adding any new features set up by a requirement specification from the customer. Examples of these could be resolving the minor invisible pictograms issue (mentioned previously in this chapter), or add gameplay elements that could be considered more engaging for all ages.

However, the main criteria for this 2014 multi-project was get everything working for customer usage. From the first sprint-end meeting, it was clear that the customer still had some wishes that could be fulfilled this semester. The Train application will still be worked on for the remaining three sprints as it will have to be updated when the dependency libraries changes during the semester.

2 Second Sprint - PictoSearch

2.1 Introduction

In the second sprint, the assignment of applications in the multi-project could be changed to new project groups. As we had found that the Train application was in a good state, we wanted to switch our development to any of the underdeveloped project, so that the multi-project could make progress towards being taken into use. As the other game applications were already taken by other groups, we decided to shift our attention to a project that no one had begun development on (not even in the first sprint); namely *PictoSearch*.

The Train application was left to a different group, and instead the PictoSearch application became the target for this group. PictoSearch is an application used by other applications to search the database and send requested pictograms to the requesting application. It was developed during the 2013 multi-project and came from the requirement made by the CAT application. The work environment for PictoSearch needed to be changed from the Eclipse development environment to Android Studio as PictoSearch had been left unchanged during the first sprint. This was a considerably easier task as this was the second migration the group had done, and as such will not be described in-depth.

The PictoSearch application had been worked on by a previous group in 2013. This group had also written a freely-available report, which gave an insight into some of the issues that had appeared with the application, as well as ideas for further development [2].

PictoSearch was in 2013 aiming to be capable of:

- Searching for pictogram using search term
- Sending one or more pictograms back
- Searching by different properties, such as by categories and/or sub-categories
- Should not be capable of launching on its own, but require a 3rd party application to launch it
- Should allow applications to specify whether a single or multiple pictograms are requested

It will be the goal of this and future sprints to complete the PictoSearch application so that it meets these requirements.

2.2 Status of PictoSearch

As described in 2.1, this group was assigned a new application, namely PictoSearch. Upon examining the application, it was found to be barely functional. The application was capable of generating a list of pictograms, and successfully select one or more pictograms to return to the previous application. However, the search box was found to have absolutely no functionality, rendering it pointless. The search filter was found to be equally dysfunctional. When the code was examined, it was found to contain a lot of unused or unfinished code, including classes with no functionality and UI elements with no underlying mechanics. It was immediately clear that the application would require a lot of work to meet the requirements set in 2013 (chapter 2.1). In addition, the sprint meeting for the multi-project added additional requirements for all applications. These requirements were:

- The minimum Android SDK version should be downgraded from 17 to 15
- The UI should to be updated to the new UI standards set by another group

With all these requirements in mind, this group decided to list the issues that would require immediate attention in a prioritized list:

- The search box had no functionality as it would show every pictograms independently of what was typed in the search box.
- There was no functionality to show pictograms for specific categories
- The minimum Android SDK version needed to be downgraded from 17 to 15
- The code was messy and requires refactoring
- The UI needed to be updated to the new UI standards made by another group
- The system would hang upon returning pictograms to requesting applications
- Categories would need to be selectable in the same manner as pictograms
- *Pictograms and categories should be delete-able*

Items listed in *italic* were added as additional functionalities that could be implemented if all other functionalities had been completed. It is not expected that all these functionalities were implemented this sprint, but the goal is to significantly improve the application, both the runtime and the code back-end.

2.3 Implementation of second sprint

In the second sprint we wanted to implement as many of the requirements listed in section 2.2 as possible. However mid-sprint, a lot of changes were made in the different libraries used to communicate with the database. Specifically, the Oasis grouped multiple times rolled out changes to the way pictograms and categories were handled. This resulted in mass confusion in all project groups, with a lot of applications being rendered unusable for weeks as a result. PictoSearch was also affected, and it became impossible to launch any of the application requiring PictoSearch, making it difficult to fully test the functionalities. It was only late in the sprint that some of the applications were up and running again.

The functionalities we were able to address are listed below. Items not listed were not addressed this sprint, but are planned for next sprint.

The search box had no functionality

The search function was previously non-existent, and as a result we implemented one of our own. An independent java class was created (named *SearchClass*) which aimed to filter and sort all pictograms and categories depending on input. This implementation would - at this time - not sort categories, as categories were yet to be implemented into PictoSearch.

Below we will go into more detail about how the searching works. In listing 2.1 we show a part of the code where we use from the Oasis library to communicate with the database and return a list of pictograms that we can work with. The *PictogramController* is a class we can use to get pictograms by using the *getPictograms()* function, which returns the list of pictograms that we need.

```
PictogramController pictogramController = new ↩
    PictogramController(this);
pictolist = pictogramController.getPictograms();
}
```

Listing 2.1: Getting a list of pictograms.

In listing 2.2 is a small outtake of code to show how we call the search function *DoSearch()*, where we send the spinner (drop-down box) info which contains whether you want to search for pictograms, categories, tags or all at once. We also send the *input* which is the string that the user has typed in, and last we send the list of all objects we have gathered from the database for the search. The function will return an updated list of the objects matching the search.

```
ArrayList<Object> searchlist = SearchClassInstance.↩
    DoSearch(spinner, input, allList);
```

Listing 2.2: Using the search class to filter list of all pictograms.

Here in listing 2.3 we show how we call the appropriate search functions in the search class. After calling DoSearch above we then check whether the *spinner* is equals to *"Pictograms"* which means we need to search for pictograms. After this we call the next search function . We have a check for all the different instances of the spinner as mentioned above.

```
if (spinner.equals("Pictograms"))
        {
            return DoSearch_Pictograms(input, ↩
                AllPictograms);
            }
```
Listing 2.3: Calling the appropriate search function

In the last listing 2.4 the *DoSearch_Pictograms* functions is shown. In this function we go through the *allList* list containing all objects, and because it is pictograms there are being searched for we check for *instanceof Pictogram*. If the object is a pictogram we instantiate a temporary pictogram to call *getName()*, which gives us the name of the specific pictogram. We check whether the name contains the input and adds the pictogram to a new list, which will contain the filtered objects. This will be the list returned.

```
private ArrayList<Object> DoSearch_Pictogram(String ↩
    input, ArrayList<Object> allList)
{
  for (Object o : allList)
  {
    if (o instanceof Pictogram)
    {
      Pictogram p = (Pictogram)o;
      if (p.getName().toLowerCase().contains(input))
      {
        lst.add(p);
```
Listing 2.4: Filtering the list Allpictograms with the search input

Downgrading Android SDK version from 17 to 15

We quickly implemented this as it was a small change in a few lines of code in the project. This was just to ensure that everything could run on the tablets that the customers have, as some of the tablets runs older versions of Android.

Updating UI

A group was assigned with the task of creating standard UI elements that every application should use, with the purpose of making everything looking consistent throughout the multi-project. The UI elements were still work in progress at the time, so only a few could be implemented. In addition, we had to make requests for buttons with centered images, as these were not available and our existing UI relied on this layout. Buttons that were updated to the new UI include 'delete' and 'confirm'.

Overall we were slightly frustrated with the progress in this sprint. The sudden changes made it very tedious to continue development because nothing worked for a significant amount of time.

2.4 Minor Collaborations

A significant part of the multi-project is to collaborate with different groups, as many application dependencies are intertwined. While a few major collaborations have been made with other groups (these are described further in chapter 5), it is also worthwhile to note some of the minor collaborations that have been made. By noting all collaborations, it is possible to analyse trends; which other groups experience problems with our application, and how can we prevent these issues from occurring in the future.

This section describes some of the discussions and interactions with the other groups to solve a minor problem between PictoSearch and their application. In this sprint there were issues with the dependency libraries, which caused a lot of groups to ask for help to solve a problem. The more important minor issues are prioritized in the following list:

2.4.1 Minor Collaborations

- Minor collaboration with the Train group to solve a crash problem after closing PictoSearch on a tablet.
- Minor collaboration with the CAT group on a bug when using PictoSearch with CAT.
- Minor collaboration with the Oasis group on rewriting parts of their GetAllPictograms feature.
- Minor collaboration with the GUI group on making buttons that were needed by PictoSearch.

For the minor collaboration which could not be solved at this sprint, meetings with the other groups were planned for the next sprint.

2.5 Sprint-end

For the second sprint-end, the PictoSearch application was not presented to customer at the meeting. This was due to PictoSearch having a lot of issues with the changes to the Oasis application, which rendered parts of PictoSearch non-functional (such as retrieving pictograms). In addition, there was little interest for PictoSearch to be demonstrated, seeing as it was not a standalone application, and did not provide any functionality of *direct* interest to the customer.

The only new feature added to PictoSearch during the second sprint that would be apparent to clients, was the addition of search functionality. Seeing as this functionality would not be demonstrable until pictograms were receivable again, it was decided not to showcase PictoSearch during this sprint-end meeting.

PictoSearch was not the only application not shown during the sprint-end. In fact, a lot of applications had yet to adapt to the sudden changes described in chapter 2.3 and were not ready to be showcased.

2.6 Conclusion

During this sprint, we managed to get a significant insight into PictoSearch. We were able to fulfill an additional requirement (as described in chapter 1.1) with the addition of searching functionality. This is a significant step in the right direction for PictoSearch, as this is one of the most important missing functionalities for the application. However, we were severely impacted by the sudden changes to the dependency libraries, and as a result we were unable to make progress for at least half the sprint. The groups responsible for the dependency libraries made major changes without discussing it with the multi-project groups. These changes did not only impact the PictoSearch application, but made every applications crash at launch as they were not updated to the new libraries. This became a huge problem for us, as we depended on the applications that utilize PictoSearch for testing and debugging purposes. Even after updating the source code to utilize the new libraries, many errors occurred and the applications would either crash or not work as intended. The major changes to the libraries should have been better discussed at the status meeting and tested before it was pushed to the repository.

In future sprints, it is desired to be prepared for sudden changes to libraries, so that these will not halt progress. This can be done by refactoring the code, so that the application can resume even without all the desired libraries.

In the following sprints, the goal is to fulfill the remaining requirements and get a fully functional application.

3 Third Sprint - PictoSearch Part 2

3.1 Status of PictoSearch

In the previous sprint, one of the major issues were the constantly-changing dependency libraries, resulting in a lot of applications having trouble working. At the start of this sprint, the dependencies had settled, and as such it was once again possible to compile and launch PictoSearch. However, as the libraries for retrieving pictograms and categories had changed drastically, parts of PictoSearch would need to be updated to reflect these changes.

Issues and functionality requests carried over from the previous sprint:

- Pictograms lack pictures
- Categories can not be searched
- The UI would need to be updated to use the new library
- The application hangs upon returning to Train (more on this in section 5.3)
- *Categories should be open-able, like a folder*
- *Pictograms and categories should be removable*

Additional issues added for this new sprint:

- Pictogram retrieval code needs to be updated
- Category retrieval code needs to be updated

In this sprint, the goal is to resolve all issues and feature requests, with the exception of those marked in italic, as these features would require more time. As such, they will likely be the focus of sprint 4 instead.

It should be noted that multiple applications were renamed during this sprint. However, we will continue to refer to them with their original names in this report to preserve consistency and lessen confusion.

3.2 Implementation of third sprint

In the third sprint, the dependency libraries were more stable which meant that we could resume resolving issues and adding features. The issues that were resolved and the features added are listed below.

Pictograms lack pictures When PictoSearch was updated to the new Oasis libraries, we had cut off several functions to get a barebone PictoSearch build up and running. Now that all dependency libraries seemed stable, we could look into re-implementing some of the lost functionality; one of them being pictures on pictograms. The implementation itself was minor and only required looking into the Oasis library to find the proper method to obtain the picture from the database. In listing 3.1 it can be seen how we now use Oasis-lib to get the pictures for the pictograms. Previously, the library picto-lib was being used for this with some similar functions for getting the pictures. However due to the changes in all the libraries we now had to re-implement this functionality with the functions from Oasis-lib. In the code outtake in listing 3.1 we use the function *getImage()* to get the image attached to a pictogram, and at the same time we also get the name of the pictogram with the *getName()* function, to show the name of the pictogram underneath the picture in our application.

```
pctNew = (Pictogram)pictograms.get(position);
if (pctNew != null) TextLabel = pctNew.getName();

Bitmap b = pctNew.getImage();
if (b != null)
{
  imageView.setImageBitmap(b);
}
```

Listing 3.1: Finding the attached picture and name of a pictogram

However, upon implementing the pictures, a new issue was introduced. The loaded pictogram images would crash when loaded repeatedly. This issue will be explained further in collaboration chapter 5.2.

A picture of the application can be seen in figure 3.2, where we have made a search for green.

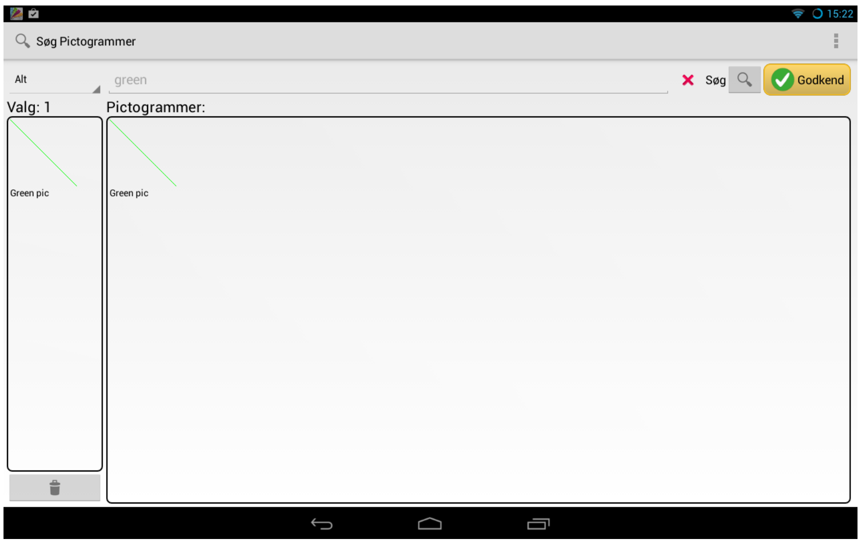


Figure 3.1: Screenshot of how PictoSearch looks now and displays a search for "green"

Pictogram and category retrieval code needs to be updated During the frantic week of dependency changes, it was decided that pictograms and categories were to be handled in Oasis instead of the now-deprecated pictogram library and category library. As a result, the code within PictoSearch for retrieving pictograms and categories was updated by removing the pictogram library and category library and replacing it with usage of the Oasis library. Pictograms and categories could now be retrieved through Oasis using the new *PictogramController* and *CategoryController* classes in Oasis. The code in listing 3.1 is an example of this. Everywhere in the code where we were working with pictograms or categories we needed to delete the functionality from the pictogram library and category library and re implement it with the functions we could find in Oasis. Some places this turned out to be a minor issue as many of the functions had the same functionality as the old libraries, however not always and more work was needed in these situations.

Categories can not be searched As described previously, categories were now managed using Oasis instead of the previous category library. This significantly changed the functionality to retrieve all the categories from the database and show it on the pictogram list of PictoSearch. The search class (described back in chapter 2.3) was extended with the functionality to search for categories and the functionality to use tags for searching. The only problem was that the database did not have any categories and the categories made with the CAT application could not be retrieve through Oasis at this stage.

The UI would need to be updated to use the new library In this sprint the UI went through some minor changes, such as adding a new dialog box when deleting a pictogram (more on this in a bit!) and additional text labels. The code for UI was updated to the newest GUI library, but other than these key points, the UI was not developed further.

The application hangs upon returning to Train The issue was tested on two tablets, but the problem was found on only one of the tablets. Meetings with the Train group were scheduled and after multiple discussions, the root of the problem was found to be in Train. More on this issue in detail in chapter 5.3.

Pictograms and categories should be removable The delete functionality was supposed to be the focus in a later sprint, but as additional time was available, it was developed this sprint.

The delete feature was implemented as a long click listener, which means that the application will show a confirmation dialog when a pictogram or category is held down for a couple of seconds. The dialog will prompt the user to approve the deletion or abort. The implementation utilizes the functionalities of the Oasis library to remove the chosen pictogram or category from the local database. An example of the code can be seen in listing 3.2 to see how we call our delete class. We call the *pictoDelete* function and send the pictogram that we want to delete in the parameter. The pictogram is then deleted in listing 3.3 where we use the *PictogramController* again, which has a function to remove a pictogram by giving the ID of the pictogram. Back in listing 3.2 we then use *getAllPictograms()* to update the UI to show that the pictogram has in fact been deleted.

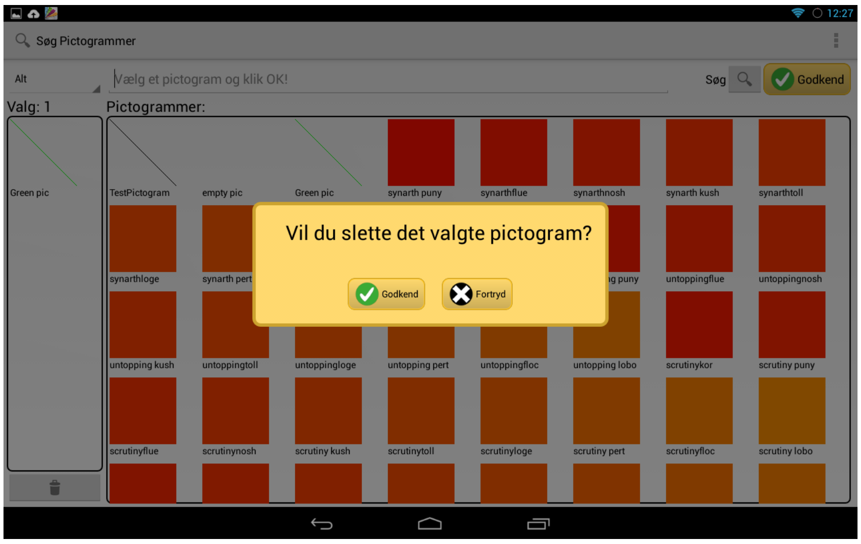
```
deleteClass.PictoDelete(view.getContext(), pictodelete);
getAllPictograms();
```

Listing 3.2: How we call the delete class

```
PictogramController pictogramController = new ↩
    PictogramController(context);
pictogramController.removePictogramById(pictogram.getId());
```

Listing 3.3: Code outtake on how we delete pictograms in the delete class

A picture of the application can be seen in figure 3.2, where the delete functionality is displayed.



Vil du slette det valgte pictogram?

Figure 3.2: Screenshot of how PictoSearch looks now and displays the delete functionality

3.3 Minor Collaborations

In the previous sprint there were some issue requests from the other groups that could not be taken care of at the time, as PictoSearch was unstable due to the changes in dependency libraries. These requests appeared again during the third sprint, and this time we were able to form minor collaborations with the groups to implement these new requests. The minor collaboration made during this sprint were:

- Minor collaboration with PictoCreator on utilizing PictoSearch in their own application.
- Minor collaboration with Tortoise on a crash in PictoSearch when launching the application in an incorrect manner.
- Minor collaboration with the CAT group on resolving a crash when launching CAT through the Launcher.
- Minor collaboration with the Sequence group that wanted the pictogram ID changed from long to int for consistency with the database.
- Minor collaboration with the Oasis on adding more pictograms and categories to the local database for testing purpose.
- Minor collaboration with the Launcher group on obtaining a QR code for logging in with the Launcher application.

The majority of the time, it is a simple problem of using a library incorrectly, or applying a minor change. These minor collaborations are often completed within an hour, and as such we will not be going in-depth with them here. The collaborations that we consider to be significant talking points are elaborated on in chapter 5.

3.4 Sprint-end

In the third sprint-end meeting, the PictoSearch application was at a presentable state. PictoSearch was not presented to the customer as a standalone application, but was shown through the presentation of the main applications in the multi project which utilizes PictoSearch. At the presentation, PictoSearch worked without any errors and the customer had no comments about changes and they were satisfied with the application.

3.5 Conclusion

In the third sprint, we were able to resolve the issues from the previous sprint as the new dependency libraries were stable. The pictograms in PictoSearch were implemented with pictures and a memory issue was found that we resolved in a collaboration with the Oasis group (described further in chapter 5.2). We were able to update the functionality of categories to have the same functionality as pictograms; the categories can also be retrieved, searched and sent back to the main application. However the requirement that the categories should be openable, like a folder were not implemented this sprint. Problems with returning to the Train application were found on one specific tablet in the first sprint, but could not be resolved due to the constantly-changing dependency libraries. This issue was investigated and resolved in a collaboration with the Train group (described further in chapter 5.3). Even though the UI has not changed much in this sprint, the code for the UI was updated with the newest GUI libraries. The GUI now provided sufficient functionalities to warrant further work on the UI. In the next sprint, the database should be filled with pictograms and categories which will help further test the PictoSearch application.

4 Fourth Sprint - PictoSearch Final Part

4.1 Status of PictoSearch

In the previous sprint we got a lot of work done on PictoSearch and at the start of this sprint we do not have many major requirements left to work on. PictoSearch was working as intended and had the required functionality needed by other applications. We did not get to work much on the UI, and further work was needed on the categories as the category libraries were still a work in progress. As a result it was decided to wait to this last sprint to finalize these parts of the application.

Issues and functionality requests carried over from previous sprints:

- The UI would need to be updated to use the new library
- *Categories should be open-able, like a folder*

Additional issues added for this new sprint:

- Everything needs to be working at sprint end

There were no new additional issues from the customers at the sprint end, but as this is the last sprint the application needs to be working without bugs or crashes.

In this sprint we will focus on getting the last two requirements done, as well as resolving any new issues that might show up and adding new required functionality from other groups. However if there is no time to add some new functionality, we will not add it as our main requirement is to deliver a working product without bugs or crashes.

4.2 Implementation

The focus of the fourth sprint is to finalize the PictoSearch application and the larger implementations will be left as ideas for further development.

Returning pictogram objects instead of ID Another group requested this functionality as they wanted to receive the pictograms directly from PictoSearch instead of using the IDs to call the pictograms from the database(which can be seen in minor collaboration section 4.3). This functionality was quickly implemented as it is almost the same as the implementation to return the pictogram IDs. However a problem occurred when trying to parse objects using the putExstra function in the Intent class, which caused an error in the implementation for returning pictogram IDs. Java does not seem to like parsing objects between activities and consequently it was decided to dismiss this functionality for further development due to lack of functionality importance and sprint time.

Finalizing PictoSearch The implementations needed to finalize PictoSearch were minor, such as removing auto-focus on the keyboard used for searching when starting the application. The functionalities of PictoSearch were polished and then tested, which is described further in chapter 4.4. Even though PictoSearch was finalized a week before the final upload date for PictoSearch, the Oasis group decided to make large changes two days before.

Sudden changes in dependencies ...again With the sudden changes to the dependencies made by the Oasis group, PictoSearch had to be changed to use the new dependencies. After using the new dependencies, a problem with retrieving pictograms and categories occurred. This was discussed with the Oasis group the day before the final upload date and they told us that the methods to retrieve pictograms and categories should not have changed. The issues were resolved at the last minute, and PictoSearch was uploaded on the final upload date.

4.3 Minor Collaborations

For the fourth sprint there was not much time left to implement major feature requests from the other groups. The focus of the minor collaborations with the other groups were to solve the minor errors before the last sprint end.

- Minor collaboration with Oasis on an issue on their end launching PictoSearch.
- Minor collaboration with the Parrot group requesting to receive pictogram objects from PictoSearch instead of ids.
- Minor collaboration with the Parrot group requesting that the keyboard does not pop up at launch of PictoSearch.

4.4 Testing

In this last sprint a series of tests were performed while PictoSearch was considered to be in a complete state. This was to ensure that all parts of the application were working as intended. The tests were performed using the *IEEE 829 Standard for Software and System Test Documentation* [3]. Even though the IEEE 829 features eight types of documents, our testing procedure will only utilize three, as these are sufficient to describe an application of a smaller size. The three documents are:

- Test Design Specification
- Test Procedure Specification

- Test Summary

As a big focus of this multi-project has been to deliver a working project, these tests will be testing towards a working application, not a flawless application. For that reason, minor warnings or quirks that do not prohibit the application from working as intended may be ignored. In addition, seeing as this paper is to be used as a reference for future developers, the tests will remain short and instead the focus will be on confirming that all components work as intended.

The feature tests that will be described in this section are:

- Searching for a pictogram by name
- Searching for a category by name
- Sending one or more pictograms to a parent application
- Deleting a pictogram
- Deleting a category

As these are the features considered essential for PictoSearch, these will be the features that are tested.

4.4.1 Searching for Pictograms and Categories

In this test, the searching functionality will be tested to ensure that the application is able to find, display and filter pictograms and categories.

Test Design Specification

The local database has been filled with 20 different pictograms and 20 different categories, all with different names of variable length (some containing symbol characters). These pictograms and categories are meant to simulate a real scenario.

The following expectations are made about how the program handles the data:

- When the search field is empty, we expect the program to display all pictograms and categories in the data list.
- When the search field is non-empty, we expect the program to display only pictograms and categories that have part of the search value in their name.

For all of these, the order in which pictograms and category are presented is irrelevant. In addition, all pictograms and categories must successfully display their name and icon, as well as be selectable. If the program displays pictograms or categories not intended to be present, then the test has failed. Similarly, if a pictogram or category does not show up when intended to be present, the test has also failed.

The search values used for the non-empty search values are as follow: **a**, **ab**, **short**, **loooooong**, **!**, **+**, **æ**, **@**, **1**, **123**, **a1**, **none**

As can be seen, a combination of a few letters, a lot of letters, symbols and numbers have been chosen. It has also been secured that there exists at least 1 pictogram for each search value, with the exception of *'none'*.

Test Procedure Specifications

The PictoSearch application was installed, and all dependency applications (local database, Oasis) were also made up to date. The pictograms and categories were added through a temporary feature in the Oasis application, which would insert the pictograms and categories. For this test, the CROC application was used as the entry-point to PictoSearch (since PictoSearch is not a standalone application).

When PictoSearch was loaded, the search field was left empty and the search button was clicked. All pictograms and categories would appear in the data list, as intended.

Next, the search words previously described were entered to see if the appropriate pictograms and categories would show up. For all letter, number and symbol combinations except 'none', the appropriate pictograms and categories would show up. For 'none', no pictograms or categories would show up, which was as intended.

Test Summary

All pictograms and categories appeared when intended, and there were no errors or halts in the application. For this reason, the PictoSearch application is considered to have passed the test successfully.

4.4.2 Deleting Pictograms and Categories

For this test, the deleting functionality of the application is tested to ensure that it can delete a pictogram or a category from the local database and remove it from the display list.

Test Design Specification

In chapter 4.4.1 the local database was filled and are ready to be tested. The expectation for the deleting functionality is that it removes the pictogram or category from the local database. When a pictogram or category has been deleted from the local database, it has to be removed from the retrieved list which are displayed in the application. The deleting function is expected to delete the pictogram or category completely, which means that it can never be restored again unless it is once again loaded into the database.

Test Procedure Specifications

Just like in chapter 4.4.1, the application and dependencies were made up to date. The first part of the test was to delete a pictogram and a category from the list in the PictoSearch application. To activate the delete message a pictogram or category was held down for a few seconds and a delete dialog appeared as intended. After clicking the accept button on the delete dialog, the list was updated and the deleted pictogram or category was gone. For the second part of the test, the application is rebooted to make sure that it was not just temporary deletion. All the deleted pictograms and categories were nowhere to be found as expected. Next part of the test was to create pictograms with the CROC application, to test that these pictograms can be deleted as well. These pictogram was deleted the same way as before, and no difference was found in deleting the pictograms.

Test Summary

All the pictograms and categories could be deleted as intended without any errors or halts in the application. The deleting functionality is considered to have passed the test successfully. The usability of the deleting method have not been tested (that is, whether it makes sense the way that users initiate deletion), but it can easily be changed later as the functionality works.

4.4.3 Sending Pictograms and Categories

One of the most significant features of PictoSearch, is being able to return pictogram IDs back to the parent application (this is referred to as *'sending'* the pictograms back). In the following test, it is ensured that this feature works as intended.

Test Design Specification

For this test, the 20 pictograms and 20 categories from the first test in chapter 4.4.1 still remain. However, seeing as name has no influence on the sending aspects of PictoSearch (as PictoSearch only returns and ID), the focus of this test will not be to send many pictograms with different names. Instead, the focus will be on sending various number of pictograms and categories, and under certain circumstances.

In this test, the following scenarios will be tested. Please note that when the list mentions *'pictogram / category'*, it means that the test is done twice, once with pictograms and once with categories.

- Sending a single pictogram / category
- Sending two pictograms / categories
- Sending five pictograms / categories
- Sending twenty pictograms / categories
- Sending a pictogram and a category
- Sending a pictogram and two categories
- Sending two pictograms and two categories
- Selecting a pictogram, deleting it, then sending it

It is expected that all pictograms and categories are able to be sent in the above scenarios, with the exception of the last scenario. In the last scenario, it is expected that it is unable to return the deleted pictogram. If any of the scenarios result in an incorrect pictogram ID being sent, or if any errors causing the application to halt occurs, it is considered failed. Otherwise, PictoSearch is considered to have successfully passed the test.

Test Procedure Specifications

The PictoSearch application was launched through Train, as Train were one of the few applications to request multiple pictograms. All applications were ensured to be up to date, the same way as in the first test in chapter 4.4.1. A pictogram is clicked, so that it is in the checkout (to-be-sent) queue. The OK button is then clicked, and the device returns to Train, where it is possible to see which pictograms were returned. This is repeated for all scenarios, and it is noted each time whether the pictograms popping up in Train match those that were selected in PictoSearch.

Test Summary

In all scenarios except the last, the test was passed. The pictograms / categories appearing in Train matched the pictograms / categories selected in PictoSearch. In the last test scenario, the application crashed upon returning to Train, meaning it has failed. This means that the overall test was not successful due to the last test scenario. This was unexpected, but it shows that some minor adjustments have to be made to the delete functionality, to prevent the deleted pictogram from being returned.

Post-Test

After the test was performed and the issue discovered, the application was fixed so that it would remove all pictograms from the checkout list that were deleted. The test was re-performed; this time it passed successfully.

4.4.4 Conclusion

After all the tests were performed and passed successfully, PictoSearch would now be ready for sprint end. At this point, no further development were made on PictoSearch, and the application is considered to be in it's final state for this multi-project.

4.5 Sprint-end

In the last sprint-end meeting, every applications were presented to the customer and PictoSearch was at a presentable state. The PictoSearch application was presented to the customer without any problems and the customers were satisfied. For the presentation, the search, sent, tags and delete functionalities were shown to the customer. The PictoSearch presentation were one of the application with the fewest errors and questions, which means that the presentation to the customer is considered a success.

4.6 Conclusion

In the fourth sprint, the goal was to finalized a version of PictoSearch which could be used by the customer without major errors. From the testing, it can be concluded that the application is stable enough to be used. At the sprint-end meeting, the customers were satisfied with the presentation and had no problems utilizing the application. PictoSearch still have functionalities that can be further developed, and the current status of PictoSearch is described in section 7.1.

5 Major Collaborations

5.1 Introduction

In the multi-project it's important to collaborate with other project groups, as a lot of applications require data or use libraries from one-another. By remaining in communication with other groups, it is possible to prepare for sudden changes made by an application or a dependency library.

In some cases (in particular with PictoSearch) technical issues appear that need to be solved across multiple applications, such as data transfer. In the cases where this has occurred, we have formed a collaborating subgroup with the other affected group, and tasked the subgroup with resolving the issue while documenting the process. In this chapter, some of the major collaborations are described in necessary detail. We hope that by describing these collaborations, future developers will be able to consult this chapter should the same issues occur again.

5.2 Memory Allocation - Collaboration with Oasis

In the third sprint, a Collaboration with Oasis was formed to solve an issue with memory allocation. The problem was found by the Train and Tortoise groups as their applications would crash after launching PictoSearch a couple of times on the tablet.

This occurred due to the fact that the Oasis library would allocate memory for pictogram images, every time the same pictogram was retrieved by another application. As PictoSearch uses the Oasis library extensively, this issue was most prominent in PictoSearch.

Several meetings with the Oasis group were arranged, and it was decided that a caching system would be the best solution to this issue. The caching system would store all retrieved images, and pull them from the cache upon request. This would avoid scenarios where a pictogram image would be allocated if it already existed in cache. As this was an implementation to be performed within Oasis, the Oasis group decided to continue development of this feature on their own.

5.3 Crash on Exit - Collaboration with Train

A continuous problem experienced on only our Android tablet, was that the application would hang upon exiting Train. This problem was initially detected in sprint 1, however it was determined at the time that the problem was occurring in Train (unrelated to PictoSearch).

In sprint 3, a collaboration with the Train group was formed, and it was through this collaboration that the issue was confirmed to be an issue with Train. Specifically, upon entering PictoSearch, the specific tablet would drop the Train state from memory, and as such would hang when attempting to return to Train from PictoSearch. This behavior was unlike any other tablet, which would otherwise keep Train in memory.

This issue was resolved by overriding default Android behavior (that is, refuse Android to determine on its own whether to drop Train from memory) and instead force Train to be saved in memory by overriding the application's onSaveInstanceState function. The basic concept laid out in the collaboration can be seen below:

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState)
{
  super.onSaveInstanceState(savedInstanceState);

  int[] train_stations = gameConfiguration.getStations();
    savedInstanceState.putIntArray("train_stations", ↩
        train_stations);
}

@Override
protected void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  if (savedInstanceState != NULL)
  {
    int[] train_stations = savedInstanceState.getIntArray("↩
        train_stations");
    gameConfiguration.setStations(train_stations);
  }
}
```

However, seeing as this implementation was primarily in Train, the Train group decided to extend this implementation to cover all of Train's state data on their own.

5.4 Code Review

Another collaboration performed with another group was code review. The point of this code review was to make our own code as easy to understand as possible, seeing as next year another group will have to work on the same application we have been working on. We asked another group who had not seen the code before, to take a look at it and provide feedback. We did this with group *SW404F14* who was working on the timer application, where we sent them our *SearchClass*, which has been mentioned previously in section 2.3. They returned the code with comments on how we could refactor the code to make it better. We performed the same for some of their code. They sent a class named *Overlay* which we examined and returned with comments to them. We will discuss the feedback received on the *SearchClass* here below.

General comments to the *SearchClass* *The code overall is pretty good, great seperation. But some of your comments are todo's and some variable names could be more describing. Other than that there is no more to say, the code is great.*

In listing 5.1 we show some of the specific comments we got from the other group. The code outtake is from our *DoSearch_Pictograms*. They recommend that we rename the variables *Object o* and *Pictogram p*, so it will be easier to understand for someone who has not written the code. They additionally recommend that *input* should also be lower case to avoid errors, however, we already do this conversion to lowercase before sending input to the *SearchClass*. As a result this will not be a problem, but the reviwer could not see this because he only had access to *SearchClass* by itself.

```
for (Object o : AllPictograms)//review: Bad naming of o, ↩
    instead use object
{
    if (o instanceof Pictogram)
    {
        Pictogram p = (Pictogram)o;//review: Bad naming of↩
            p, instead use pictogram
        if (p == null || p.getName() == null) continue;

        for(int i = 0; i < input.length; i++)
        {
            if (p.getName().toLowerCase().contains(input[↩
                i]))//review: should input not also be ↩
                toLowerCase?
```

Listing 5.1: Code with some of the returned comments from group sw404f14

These comments can be used to improve the readability of the code. We will look into improving our comments in the code as well as improving variable names.

6 Multi-project

6.1 Experiences with the multi-project

In this 6th semester working in a multi-project was a new experience for many of us. It was the first time that it was needed to work and communicate with people beside your own group. Even though Aalborg University promotes group work, there have not been any collaboration between the groups in the earlier semesters. Nonetheless in the 5th semester we had the course called Software Engineering, where we were taught the methods to work in a large software project. This semester was a great opportunity to really apply the methods learned on a larger project environment before graduating. To coordinate the work between the sixteen groups, it was decided to utilize the Scrum method. The method that was actually used was not 100 percent Scrum, but many of the principles were applied. At the start of the semester, the multi-project was divided into four sprints over the 6th semester. The length of each sprint was pre planned but could change if needed, however as a result it felt a bit random and as if there were no proper deadlines. This also made the sprint-end presentation feel like the same each time, and many of the changes that were presented to the customer at end was either minor changes or had the same functionality.

Once a week a status meeting was held to keep everyone informed about the changes in each application. This was a great way to maintain the communication between each groups and made it easier to find the groups that you needed to work with. It was shown during the sprints that communication between the groups was a key factor for the multi-project to become a success. At the first sprint of the multi-project, everyone seemed confused about what the goal of multi-project was and where to begin. This caused many groups to isolate themselves from the other groups and only work on their own issues like solving bugs and port the distributed applications to Android Studio.

We would have liked better management from the semester coordinator or at least have a group appointed to management. Even though the applications of the multi-project depended on each other, dysfunctional or major changes were made and published constantly without any warning or mutual agreement. The experience of sprint 2 was horrible as PictoSearch utilized all the dependencies that kept changing, which reduced the productivity of the group and the other groups.

The coordination between the groups could have been more structured, but we do not blame the people trying their best to keep structure in the multi-project. We take these problems as a part of the experience and we now know what can be done better in future projects we might participate in.

6.2 Tools

As part of the multi-project, several tools were introduced to ease the pains of developing in an environment where modules depend on one-another. These tools were introduced in the first few week of the multi-projects development by representatives from the previous year. They recommended a few tools which would help in a multi-project and these three were chosen as the tools for the multi-project of 2014.

The selected tools include:

Redmine

Project management and bug tracking software using a web interface. The Redmine tool is used to keep track of the different application and ease communication between the 16 project groups. It has a forum to discuss solutions, a functionality to keep track of issues and guides for the different application and tools can be written there. The most important feature in Redmine for this multi-project is the functionality to make a Gantt chart from the issues made in Redmine. This makes it easier to plan, structure and keep track of each sprint.

This worked well and provided the necessary communication between project groups.

Jenkins

Automatic build tool. This is used to automatically builds all the applications online, which makes it easier for the multiple project groups to download the applications they need. The Jenkins tool continuously build each time someone pushes code to the repository and gives feedback of whether the build is broken. It documents every changes and tell you when it failed and how to revert it.

Jenkins provided a great overview about which applications worked and which did not, however, it was not being used to its fullest extend and we question whether it is worth the upkeep.

GIT

Version control software, which is used to keep track of the changes made to the source codes. The GIT tool documents how each version was developed from each other and holds relevant information on the different versions. The positive effects of utilizing a Version control software, is the functionality of workspace and working in concurrence. It is possible with the GIT tool to retrieve a copy of the repository which can be worked on and updated at all times. When the changes are finished, it can be pushed to the repository and the tool helps control the multiple concurrent changes to the same source code.

> Syncing software is critical to this kind of development with a major amount of project groups, and we found GIT with its branching system to be surprisingly pleasant. SVN would also have been acceptable in this case.

We would recommend that the next multi-project developers consider using the same or similar tools as we found them to be a great aid.

7 Conclusion

7.1 Current Status of PictoSearch

To aid any future group who will work on PictoSearch in future multi-projects, we will in this section describe the current status of PictoSearch. This should provide a clear picture of how PictoSearch looks, and which changes need to be made to meet future requirements.

PictoSearch is currently working properly with pictograms, as you can search for and select a single or multiple pictograms in the database. The pictograms are displayed with a picture and the name underneath. It is also possible to delete pictograms from the database. In a similar fashion, categories can also be searched for and selected in the database. They are treated exactly like pictograms, however, when they are sent to the previous application, they are removed and replaced with the pictograms they contain. There is currently no support for opening categories and selecting specific pictograms from within a category. The application currently fulfills its purpose and any new additions are considered polish.

In the multi-project, the PictoSearch application depends on Oasis and the GUI libraries. If any of these change, PictoSearch needs to be updated accordingly. A frequent pitfall for us was to assume that Oasis always works and is operated in the same way. From experience, we have found that this is not true, in fact, Oasis would change very often. To counter this, PictoSearch should abstract its connection with Oasis, either through adapters, controllers or similar means. Currently, no such abstraction is implemented.

A lot of other applications also rely on PictoSearch. A few examples are CROC, Train, Tortoise and CAT. It is necessary for this reason that PictoSearch is stable, so that these applications can continue their development. Sometimes, these application may attempt to launch PictoSearch in an incorrect manner, or handle the returned data incorrect. This is often a communications issue; it is important to make information available on how exactly to launch and handle PictoSearch.

Overall, the PictoSearch application is in good standing. The source code for PictoSearch contains some unfinished or unused code which can be used for further development or polish the current functionalities of PictoSearch. In PictoSearch, the class for the search and delete functionalities could be more abstract, making it easier to add more functionalities for further development. The delete function is implemented with a on item long click listener and the usability of this methods should be discussed with the customer. Currently PictoSearch does not take profiles into consideration, as it retrieves all the pictograms and categories from the local database. This also happens when using the delete functionality, which means that a delete will remove the pictogram or category for all profiles. Out of all the applications available in the multi-project, it is one of the applications that are the most complete. For that reason, it might be worth turning the attention towards other applications that require more work. However, there are plenty of additions that could be considered for PictoSearch, these will be discussed further in chapter 7.3.

7.2 Conclusion

We have in this semester worked with the multi-project concerning learning games and applications aimed at helping kids with autism. We have been working on the multi-project over four sprints, where we have been developing two different applications.
In the first sprint we worked with the learning game Train, which was about categorizing objects correctly. Most of the time in this sprint was spent with getting the application up and running in Android Studio, rather than developing new features in the application. At first sprint-end we ended up with a working application, where some bugs had been fixed as well as minor features added to the game.
For the second sprint we were given the task of working with PictoSearch, which was the application used to search through the database. No group had worked with this application in the first sprint, so there was a lot of issues with this application. The code needed a lot of refactoring, the application was lacking features and there was issues with many of the features already implemented. We did not have time in this single sprint to work on all the requirements and issues, because of all the changes in database and libraries. We managed to add the necessary functionality to process pictograms, however the functionality for categories was still lacking at this time.
In the third sprint we continued our work on PictoSearch, where we still had to add the category functionality as well as other minor features and issues. We had a lot of minor collaborations with groups in this sprint, to ensure that PictoSearch was running as intended when opened from the other applications. We also implemented the requirement of deleting pictograms and categories from the database. PictoSearch was working as intended at the end of this sprint, so for the last sprint it would mainly be polishing the application and resolving new issues that would emerge.
For the last sprint we were again tasked with working on PictoSearch. We did not have many features to work on in this sprint, so we mainly just tried to resolve any issues and requirements from other groups, and we also tested the application . However due to big changes in the database and oasis just before we had to deliver PictoSearch, we ran into a lot of issues. But we delivered the application and it ran without issues at the sprint end presentation.

In conclusion, we were tasked with working on Train and PictoSearch, two applications in the 2014 Aalborg University multi-project. We provided additional functionality to Train, and managed to turn PictoSearch around from an unfinished product to a product in good standing.

7.3 Further Development

In chapter 7.1, it was discussed how PictoSearch is currently set up. In this section, we will be going through some of the shortcomings of PictoSearch, as well as recommended additional features. This section is ideal reading material for anyone interested in expanding PictoSearch.

Previously, it was mentioned how dependency libraries often became an obstruction in the development of PictoSearch. Instead, dependency libraries should be abstracted, so that the actual implementation of the libraries become less significant, and easily replaceable. Proposed solutions to this are controller, adapter or interface implementations, e.g. have a replaceable, modular class dedicated to handle the exchanges between PictoSearch and the dependency libraries, that is properly able to handle when the library is not working or has been changed.

While pictograms are well-implemented across the multi-project, categories are often ignored and discarded in many applications. A few example of this are the Train and CROC applications, as they are only able to accept pictograms, not categories. While this is not direct a problem with PictoSearch, it explains why PictoSearch has been developed in a way so that it only returns pictograms (and pictograms within categories), but not actual categories. Applying this significant change may require a change in the entire multi-project to make all applications accept both pictograms and categories. It would only require a very small change in PictoSearch to apply this transition, but the transformation would not be easy for all other applications.

In our design process, we considered categories to be similar to Microsoft Windows folders. Given this analogy, we felt it would be natural to allow the categories to be opened within PictoSearch. Opening a category would allow access to the individual pictograms within the category. That way, categories could become more prominent (such that all pictograms are organized and categorized), and users would be able to hand-pick pictograms from a category.

The development of PictoSearch in this multi-project was resumed from a previous multi-project. The majority of the application was at that time developed in a single class, which resulted in one, massive file that is hard to navigate. We put effort into refactoring this, by abstracting the different functions into several different utility classes (such as SearchClass, DeleteClass, etc.). However, chunks of code from the previous multi-project are still present, and some of it unused. For this reason, additional work could be put into refactoring.

The majority of these changes do not require that PictoSearch is changed in how it operates (with the exception of returning categories). As a result, PictoSearch can be developed without the fear of how these changes will affect other applications. This is unlike other applications (like Oasis), that other applications rely heavily on performing in a very specific manner (due to library dependencies).

Bibliography

[1] A. J. R. A. N. O. S. R. Wortmann, J. K., “Interactive learning exerise for children with autisim,” 09 2013.

[2] J. E. M. H. L. N. Anders Vinther, Christian Klim Hansen, “Communication application for children with autism, category administration and inter-application communication,” 06 2013.

[3] C. Consulting, “Ieee 829 documentation.” http://www.coleyconsulting.co.uk/IEEE829.htm. Date visited: 1st April 2014.