Android Timer Application

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Øst
Telephone +45 9940 9940
Telefax +45 9940 9798
http://cs.aau.dk

Title: Timer Application
Subject: Android Systems
Semester: Spring Semester 2012
Project group: sw602f12

Participants:
Kristian Kolding Foged-Ladefoged

Rasmus Hoppe Nesgaard Aaen

Simon Blaabjerg Frandsen

Supervisor:
Ulrik Mathias Nymann

Number of copies: X

Number of pages: X

Number of appendices: X Pages

Completed: X

Synopsis:

This project is about the development of...

Preface

This project has been produced by group SW602f12 at Aalborg University in the spring 2012 on the sixth semester of the software engineering study.

This project is written in light of developments of the timer application for the The GIRAF Project. The GIRAF Project is designed to support educators in schools for children with ASD, among these is the timer application that is built to replace the timers the institutions have provided.

The project describes the working process used to develop the product, included in this is the development method, the design, the implemented features, and the tests of the timer application.

The intent of the project is to explore the possibilities of combining tools used by educators for children with ASD, in a device which is portable and easy to use.

The project group would like to thank the educators whom helped developing and designing the timer application as well as our supervisor for the interest and cooperation in the project.

Contents

Special Words

- Time Timer - A visual clock. Used by parents and teachers as a tool to visualize time for children.
- Pictogram - A graphic symbol that conveys its meaning through its similarity to an object.
- Eclipse - An intelligent development environment for Java.
- ASD - Autism Spectrum Disorder.
- Children - Children refers to "Children with ASD".
- Guardians - parents, teachers, caretakers, and educators of children with ASD.
- MVC - **M**odel **V**iew **C**ontroller [**?**].
- Timer application - The end product of this individual project.
- WOMBAT - Name of the timer application, **W**ay **O**f **M**easuring **Ba**sic **T**ime.
- PARROT - Name of the pictogram application, **P**ictogram **A**ssisting with **R**hetoric **R**easoning **O**r **T**alking
- We - in the introduction, it refers to the whole multi project group. After the common report, it refers to the individual project group.

Part I

Introduction

In this part there will be an introduction to the project, including background knowledge about the target platform, and knowledge about ASD. There will be an analysis of the problem followed by a system definition of the whole multi project, and the specific group project.

CHAPTER 1

Analysis

Through meetings with an educator at Birken, a kindergarten for children with ASD, we have learned about the importance of having access to well-designed communication tools when working with children with ASD.

At Birken they often use hourglasses and time timers, in different sizes and colors to visualize the progression of time to the children. The children can then associate the color and size of an hourglass with the time it represents.
Our contact person explained how the educators at the institution use pictograms to communicate with the children. The children have a schedule of the day, where all their daily activities are listed in pictograms, such that the children can always go to their schedules and find out what they are going to do next. Also activity instructions are listed with pictograms, i.e. in the bathroom there is a scheme showing what to do when going to the bathroom. The pictograms used at Birken is from the software *Boardmaker*[5]. To use the pictograms from *Boardmaker*, the educators have to print, cut, and laminate them.

The educators have to take timers and pictograms with them, everywhere they go. Since there is supposed to be more than one specific timespan available, the timers can take up much space. Furthermore the pictograms are lightweight, which can make it difficult to use them outside when it is bad weather or windy.

1.1 System Definition

The timer application is targeted for Android tablets running Android 3.2. The use space is institutions and homes of children with ASD.
The application is a tool, such that parents and educators can visualize time in a way customized for each child by changing color schemes, symbols, forms, and save this information in profiles stored on a server. The visualization is formed as a full-screen timer, which can be customized to be shown as an hourglass, stopwatch, progressbar, or digital watch.
Furthermore the guardians should be able to add pictograms to the timer, to show them what they are going to do while the time is running, and what they are going to do when the time has expired.

Tail

Part II

Development

In this part we start with a section about project management, where we expand on the used development method and versioning. After that we describe the design, development process, implementation, and test of the WOMBAT application.

CHAPTER 2

Design

The design of WOMBAT has changed throughout the project. The design chapter presents the vision of the timer application and the tools used to design and develop the product.

2.1 Vision

Since the launcher had both a child-mode and a guardian-mode at the first period of the project, this vision relies on the initial design of the launcher project.

The idea of the WOMBAT application was, that is should be a tool for educators to illustrate time for the children.

One part is the timer application as it is implemented, from which it is possible to run timers, such as an hourglass or a digital watch. This part should be available only from the guardian-mode.

The second part of the application is a timer overlay, which is launched when other applications are launched through the GIRAF launcher. When such applications are launched through guardian mode, the user should be prompted to select if there is a time limit on the application, and what the limit should be. If a time limit is chosen, the given application, i.e. a game, is run with the timer overlay showing a custom timer with the time left (see

figure 2.1).

Figure 2.1: Example of how the timer overlay would look. Here there is an overlay on a game with about 40 minutes left.

If the launcher is in child-mode, and the child opens an application, the timer overlay is run according to settings chosen in the third and last part of the WOMBAT system; the settings application. The overlay can be used if a child is only allowed to play a game for 30 minutes a day, then the overlay can be customized to show the child how much time is left of the allowed time. When the time has expired, the application is automatically closed.

The third part of the application is only available from the guardian-mode in the launcher, and from this application it is possible to customize which applications should be run with the timer overlay. Also it is possible to define how the overlay for every child should look like, and what should happen when the time limit has been reached. Also it is also possible to set constrains for the applications, such as time constrains, i.e. certain applications can only be opened in a different timespan on the day, or when a specific application has been run for 30 minutes straight, the application is closed and cannot be opened before a certain "cooldown" has ended.

2.2 Stories

These stories are fictive, and are based on the vision of the timer system, together with an interview with our contact person.
We use a fictive person in the use cases, Trine, who acts as an educator in a special kindergarden for children with ASD.

Timer Application

Trine got an Android tablet from the kindergarden yesterday, and some of the other educators suggested that she should try out the WOMBAT timer application. Trine has planned a playing session with a few of the children, and decides to use the WOMBAT application to time the session, such that the children can see when they are done playing. They have about 30 minutes to play in, so Trine opens WOMBAT on the tablet and selects a predefined hourglass set to 30 minutes. When the children are in place and ready to play, she press the start button, and the hourglass starts. When the time runs out, a "Done" screen appear, and the children can see that they are done playing.

Timer Overlay

Trine walks in the playground while the children are playing. She sees one of the boys sitting on the ground in the corner of the playground. She walks over to him. It is Casper, he had tripped over his own feet, and hurt his knee. To get Caspers mind of his knee Trine lets Casper play his favorite computer game for the next 10 minutes. She starts the game, and gets prompted to choose a profile and the amount of time the game should allow to run. She selects Casper's profile and 10 minutes. The game starts with the timer overlay showing 10 minutes left of play time, with Casper's favorite green digital clock. When the time has run out, the game closes, and Casper is again ready to play with his friends.

2.3 Prototyping

Before the actual development started, we made a few drawings of our ideas (see figure 2.2).

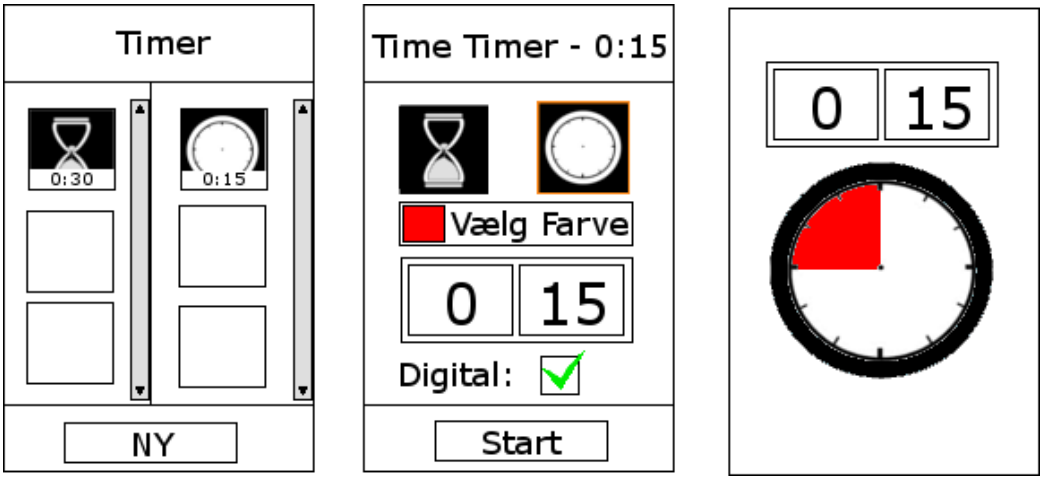


Figure 2.2: Drawing of the initial ideas of the timer application.

Paper Prototyping

Paper prototypes[1] has been used in the first iterations in the development process. The initial idea of the system design was drawn on paper, so that our contact person could give us some feedback on the design, before we started programming. In figure Figure 2.3 is a paper prototype of the menu in the timer application, and the rest of the prototypes can be found in appendix section 9.2.

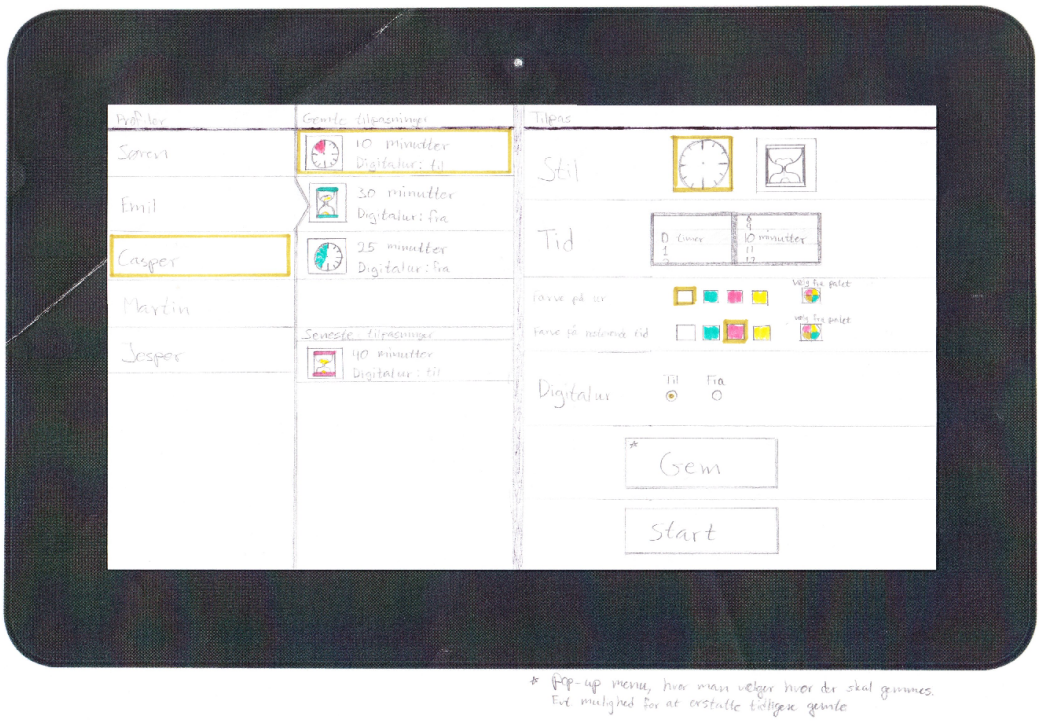


Figure 2.3: Scan of a paper prototype of the menu in the timer application.

Paper prototypes are produced quickly, and they capture early design ideas.

2.3.1 Metaphors

To enhance usability and learn-ability, we have used metaphors[1] on the buttons in the application. On the "Attach"-button, used to attach a second timer or one or two pictograms to the main timer, we have placed a paper-clip, which is known from the attach function in other programs, for example Microsoft Outlook Express. Furthermore we have used metaphors on the "Start Timer"-button, which looks like the "Play"-button known from various media players, and the "Save" and "Save As" buttons have a floppy disc icon, which is known from the save button in various word processing programs, for example Microsoft Office Word[1]. In figure 2.4 examples of the implemented metaphors are shown along with screenshots of other systems they are implemented in.

[1]Non-free word processor developed my Microsoft

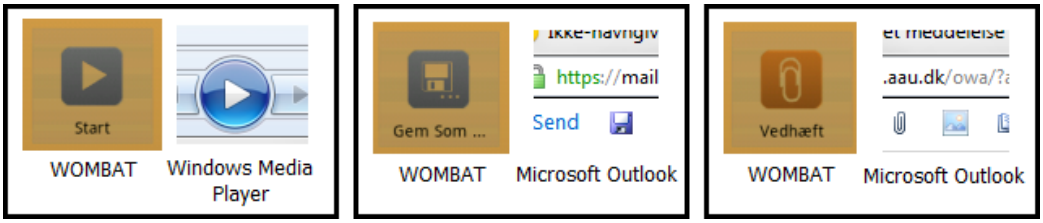


Figure 2.4: Metaphors implemented in WOMBAT and the original implementation.

CHAPTER 3

Development Process

As stated in chapter **??**, the development method used in this project is a modification of Scrum, which means that the development evolve through sprints. Here is a description of the six development sprints we had, and in appendix 9.3 all sprint backlogs can be found.

3.1 Sprint Walk-through

Because it took some time for all groups to agree on a development method, we started making some design and talking to our contact person before the actual sprints started, so when the first sprint was started, we already had an idea of the functionalities we wanted to end up with.

Sprint 1

19/03 - 23/03: In the first sprint, the main objective was to set up the android project in development environment, and get the development started. At this point we were expecting to end up implementing the whole vision (see section 2.1), and a natural place to start would be the timer. We had an idea of how the design should be, and we began to implement lists for holding children and configurations.

Sprint 2

26/03 - 04/04: In the middle of this sprint the launcher group stated that they would only be able to make the guardian mode in the launcher in this semester, so we decided in the timer group, that the two last parts of the timer application, the overlay and the settings, would not be developed in this semester either, since there will be no need for them if there is no child mode in the launcher.
We continued designing and developing, and in the end of this sprint, we had the design ready for implementation. We also started to make some OpenGL development[1], as we thought that OpenGL was the best way to implement the timers.
We had a meeting with the contact person, and she suggested that when the timer in the application has exceeded, it would be possible to show two custom pictograms on the screen. Also she suggested that two different timers could run in parallel on the screen, if a child was used to one specific timer, but was learning how to use another type of timer. We added these to suggestions to the product backlog.

Sprint 3

10/04 - 19/04: We found out during this sprint, that it was more convenient to implement the timers using canvas and 2D drawings. The progress bar had been implemented in both OpenGL and on canvas, and the canvas version was the easiest to implement, and required fewer lines of code.

Sprint 4

23/04 - 04/05: In this sprint we finished drawing the timers, and a lot of the most vital functionalities, such as loading and saving configurations, highlight on list items, and the "Done" screen, were implemented.
During this sprint we had a lot of contact with the admin group, because we had difficulties implementing functions to load and save data to the database. We also integrated the timer with the launcher, and did some testing on the interaction between them.

Sprint 5

07/05 - 11/05: This sprint was used to do some refactoring of the code, to make it more readable and understandable. We also implemented the last

[1] Widely used 2D/3D graphics API

things from the backlog, and did some polishing to the overall design. Furthermore we started making test design and test cases for the functionalities we wanted to test in the timer application.

Sprint 6

14/05 - 18/05: In this sprint we implemented the last critical functionality.

Besides getting done with the development, we started with the acceptance test, by letting our contact person use the timer application at the institution over a period of three days.

Sprint 7

21/05 - 25/05: This sprint was used to perform and evaluate on usability test, acceptance test, and black-box test.

CHAPTER 4

Development Tools

Beskrivelse af miniprojektet fra SOE kurset (om udviklings metoder etc)
Hvilke værktøjer har vi brugt fra SOE kurset - Beskrivelse af disse!

Fra soe...

CHAPTER 5

Implementation

Nice...

5.1 Architecture

5.1.1 Activity(android)

5.1.2 Architecture model

The WOMBAT application is depended on the Launcher project and Oasis project, but we choose to design WOMBAT as an independent application with features from Launcher and Oasis. We choose to design WOMBAT as an independent application, that way we never have to wait for the Launcher or Oasis to release features regarding WOMBAT. You can see a dependency diagram of WOMBAT architecture. The WOMBAT architecture is a five layer architecture which enhances how you can perform testing and do collaborative work. The five layers consists of:

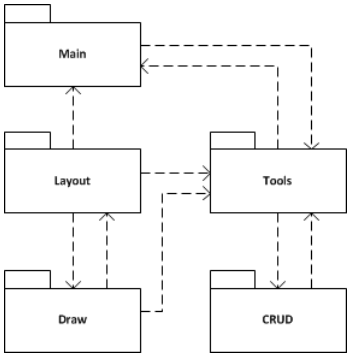


Figure 5.1: Dependecy diagram of WOMBAT architecture

Main

This layer is the main activity of WOMBAT which means that it initiate the whole WOMBAT application. The main layer is depended on the Tools layer since it contains all the initiating tools.

Layout

This layer cosists of the two fragments; Profile/SubProfile fragment and customize fragment, and the custom arrayadapters which WOMBAT uses. The layout layers is depended on the Tools since it contains all the objects and methods that is required for the layout to work properly. The layout layer is also depended on the main layer, without the main layer the layout layer would never be initiated. Final but no least the layout layer is depended on the draw layer, the draw layer delivers the methods that generates the views for the timers and pictograms.

Draw

This layer contains the methods that generate the views of timers and pictograms. Draw layer is depended on the tools layer since the tools layers contain all the different types of objects that the draw layer uses. The draw layer is also depended on the layout layer, the layout layer initiate the draw layer.

Tools

This layer cosists of all the types of objects and methods that WOMBAT uses. The tools layer is depended on the main layer to initiate the proper objects, you can read more about this in subsection: **??**. The tools layer is also depended on the CRUD layer which contains the connection to the Oasis library.

CRUD

This layer is responsible for saving and loading from the OasisLocal-

Database, it is depended on the tools layer because it uses the objects and method that the tool layer provides.

Library

Our original idea was the split the layers into two projects, that way it would be able to conduct tests outside the main layer. The first project would be an Android project with a main activity, WOMBAT. The second project would be an Android library which should contain all the backend functionality, TimerLib. This is how it the two projects would look like:

WOMBAT

- Main layer
- Layout layer

TimerLib

- Tools layer
- Draw layer
- CRUD layer

We later decided to redesign, and make three projects instead of two projects. We choose the redesign because that we are three developers in our project whom all want to conduct independent testing. The first project, WOMBAT, would stay as it original was. The Second project would be split into two projects, TimerLib and DrawLib. TimerLib would contain the tools layer and the CRUD layer. The DrawLib would contain the draw layer.

WOMBAT

- Main layer
- Layout layer

TimerLib

- Tools layer
- CRUD layer

DrawLib

- Draw layer

The dependency diagram of the WOMBAT projects can be see on figure: 5.2.

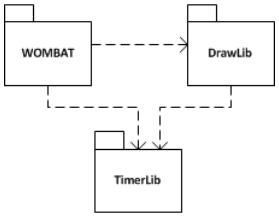
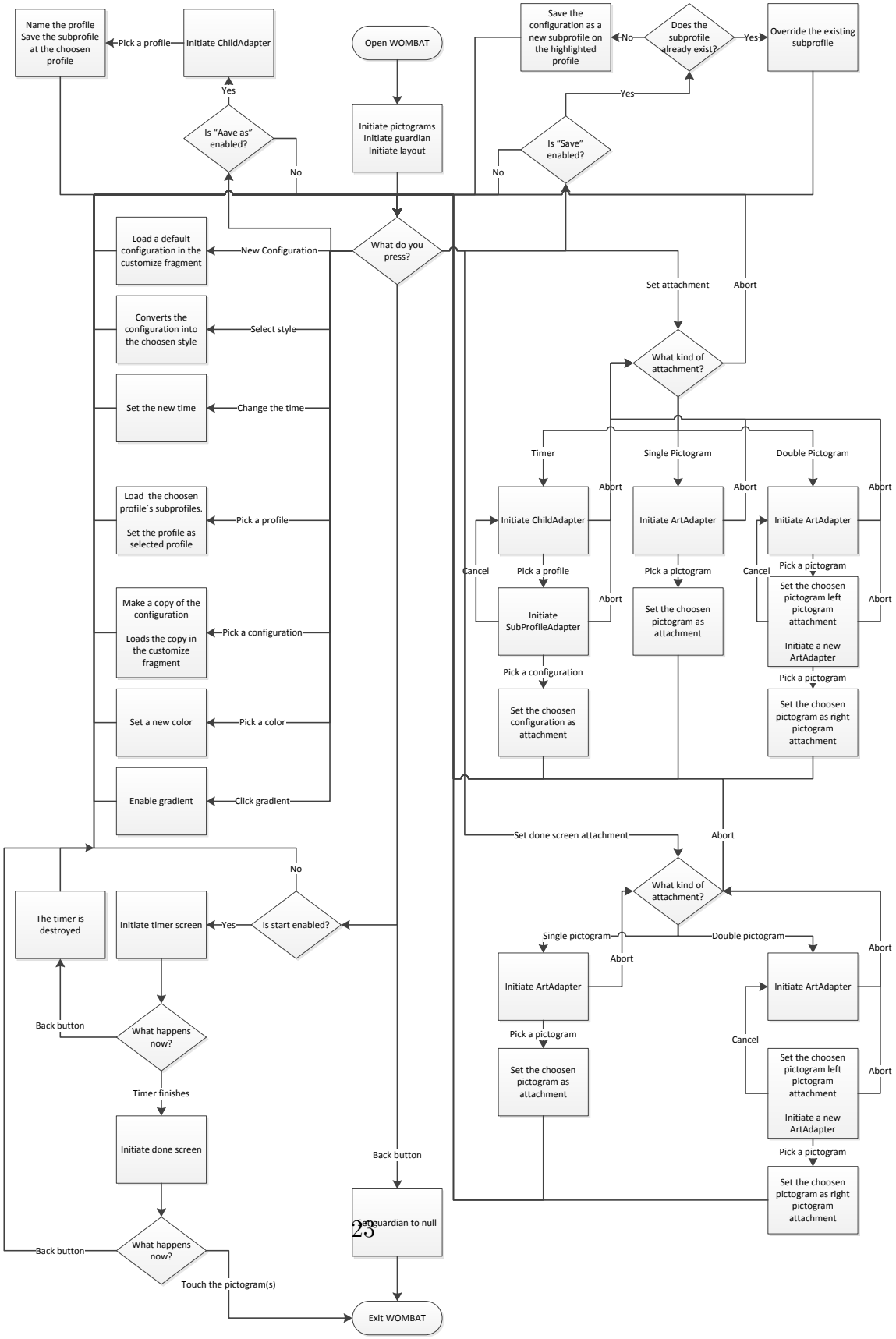


Figure 5.2: Dependecy diagram of WOMBAT projects

Wombat lifecycle

You can see a detailed flowchart over the WOMBAT lifecycle on figure:5.3. The flowchart descrips whatever that can occur while the WOMBAT application is running. The flowchart can help debug and understand the application if somebody chooses to develop futher on WOMBAT.


Open WOMBAT
Initiate pictograms
Initiate guardian
Initiate layout
What do you press?
Name the profile
Save the subprofile at the choosen profile
Pick a profile
Initiate ChildAdapter
Is "Aave as" enabled?
Yes
No
Save the configuration as a new subprofile on the highlighted profile
No
Does the subprofile already exist?
Yes
Override the existing subprofile
Yes
Is "Save" enabled?
No
Load a default configuration in the customize fragment
New Configuration
Converts the configuration into the choosen style
Select style
Set the new time
Change the time
Load the choosen profile´s subprofiles.
Set the profile as selected profile
Pick a profile
Make a copy of the configuration
Loads the copy in the customize fragment
Pick a configuration
Set a new color
Pick a color
Enable gradient
Click gradient
Set attachment
Abort
What kind of attachment?
Timer
Single Pictogram
Double Pictogram
Initiate ChildAdapter
Initiate ArtAdapter
Initiate ArtAdapter
Abort
Cancel
Pick a profile
Pick a pictogram
Initiate SubProfileAdapter
Set the choosen pictogram as attachment
Set the choosen pictogram left pictogram attachment
Initiate a new ArtAdapter
Pick a configuration
Set the choosen configuration as attachment
Set the choosen pictogram as right pictogram attachment
Set done screen attachment
Single pictogram
Double pictogram
Initiate ArtAdapter
Initiate ArtAdapter
Set the choosen pictogram as attachment
Set the choosen pictogram left pictogram attachment
Initiate a new ArtAdapter
Set the choosen pictogram as right pictogram attachment
No
Is start enabled?
Yes
Initiate timer screen
The timer is destroyed
Back button
What happens now?
Timer finishes
Initiate done screen
What happens now?
Back button
Touch the pictogram(s)
Back button
Set guardian to null
Exit WOMBAT

5.2 Fragments

In short a fragment in Android is functioning much like iFrames in HTML, meaning that a fragment is an embedded activity within another activity. According to Google the design philosophy of fragments are that they support a more dynamic and versatile design of applications on devices with larger screens, such as tablets. Google states that on these kind of devices there are more room to combine and interchange interface components, compared to a handset. An example of the use of multiple fragments on a large screen compared to activities on smaller handsets, is the Gmail application from Google:

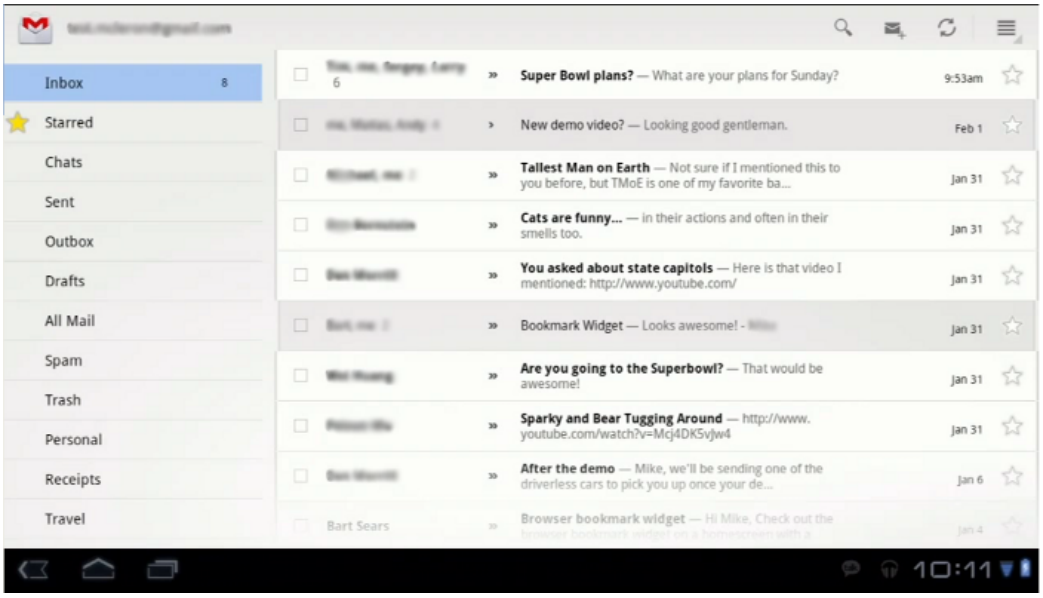


Figure 5.4: The Gmail application from Google on a tablet with fragment support.

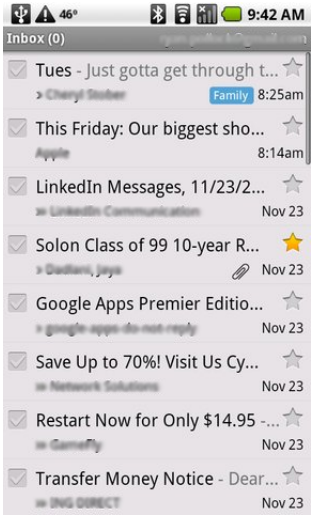


Figure 5.5: The Gmail application from Google on a handset without fragments

This and more information about fragments can be found at [4].

5.2.1 Benefits and Limitations

Fragments alone are not more powerful than activities and fragments can not exist without activities. When combining multiple fragments in one activity, the fragments are able to create dynamic interfaces which does not require the entire activity to be reloaded when changing an object on the screen.

The Gmail application utilizes this by having an overview of the email account on the left side of the device while showing specific mail content on the right side as seen in figure 5.4.

Fragments are built to be reused in multiple activities, which means that a single fragment can be used in several ways depending on the platform. The main activity only has to declare which fragments it holds in the layout.

Fragments were introduced the first time with the Android 3.0 platform (API version 11). But as of 3rd March 2011 Google has made the "Android Compatibility Package", a library which let Android 1.6 devices or newer support fragments [2]. `Mobile tuts+`, a website about tutorials for Android development, has tested the "Android Compatibility Package" and they successfully used fragments with the support library on the T-Mobile G1 (the first Android device) [3].

5.2.2 Fragments in WOMBAT

In WOMBAT we use fragments to give the user the ability to quickly switch between multiple children and configuration templates, when developing with fragments we are able to let each part of the screen update independently based on actions in the fragments. This means that we, much like the Gmail application, can keep an overview of all children and their personal configurations, while having a detailed configuration page open at the same time. Thus avoid requiring the user to switch screen to load previous defined configurations.

Having developed WOMBAT with fragments means that it is possible to create a handset version with few modifications, since it only requires the "'Android Compatibility Package"' library and some new activities to handle the fragments on a smaller screen.

5.3 Lists

In Android the philosophy behind lists is that the majority of all applications at some point will have to implement a list object. Therefore does Android ship with a build in list view, that supports a variety of list. This build in list view can be modified to match any list, that the developer wishes to create, as shown in 5.6.

Figure 5.6: Examples of how ListViews can be modified.

5.3.1 Benefits and Limitations

The advantage of a list view is, that we can modify the list to whatever we want it to look like without having to create any complicated programming to make it look good. This is done by making an adapter with the layout that the items of the list is going to look like as in figure 5.7, which is the adapter we use to show the children list.

```
View v = list_item_view;
  if (v == null) {
    LayoutInflater li = (LayoutInflater) getContext().
        getSystemService(
         Context.LAYOUT_INFLATER_SERVICE);
    v = li.inflate(R.layout.profile_list, null);
  }
  // FIXME: Insert pictures from admin here
  Child c = items.get(position);
  if (c != null) {
    ImageView iv = (ImageView) v.findViewById(R.id.profilePic);
    TextView tv = (TextView) v.findViewById(R.id.profileName);

    if (iv != null) {
      iv.setImageResource(R.drawable.default_profile);
    }
    if (tv != null) {
      if (c.name == "Last Used") {
        tv.setText(R.string.last_used);
      } else if (c.name == "Predefined Profiles") {
        tv.setText(R.string.predefined);
      } else {
        tv.setText(c.name);
      }
    }
  }
```

Figure 5.7: Example of an adapter to a list view, the actual adapter used in the children list

Creating lists in adapters and adding adapters to list views is alot easier than making custom lists. Because these built in list views are very versatile and can be configured to do exactly the same as any other view in Android.

Since lists wont be able to do anything but show a list of items without an on click listener does the list view have an interface which works exactly like any other clickable item in android. The on click listener in the list view is as easy to implement as a button or similar clickable object.

5.3.2 Lists in WOMBAT

Figure 5.7 is the actual implementation of the child list in WOMBAT, the only difference from the child list adapter and the configurations adapter is that the timer configurations require another text field with a description of the timer. Which means that the configurations list only requires another layout to be constructed, which holds the extra text view.

The on click listeners of both lists, ensures that the next fragments and Guardian object, described in **??**REF MANGLER , is updated figure 5.8 is an example of the child fragment listview.

```
1       public void onListItemClick(ListView lv, View view, int
            position, long id) {
          // Update the fragments
3         SubProfileFragment detf = (SubProfileFragment)
              getFragmentManager()
               .findFragmentById(R.id.subprofileFragment);
5         CustomizeFragment custF = (CustomizeFragment)
              getFragmentManager().findFragmentById(R.id.
              customizeFragment);
          custF.setDefaultProfile();
7
          if (detf != null) {
9           // Marks the selected profile in the guard singleton
            guard.profilePosition = position;
11          guard.publishList().get(position).select();
            guard.profileID = guard.publishList().get(position).
                getProfileId();
13          detf.loadSubProfiles();
          }
15      }
```

Figure 5.8: Example of the on click listener of the child fragment

5.4 Customize

The customize fragment is the heart of the application, this is where everything is generated and configured. The fragment can both create new and modify already existing timers, this is the key functionality of the customize fragment.

The timers that the guardians already use at the institutions comes in many different colors and timespans. Therefor is it possible to modify the look, time, and color of the individual timers in the customize fragment. Furthermore did the guardians state that a good feature would be if either pictograms or other timers were attachable to the timers.

Combined with the general idea of the application with a child and timer overview, is the requirements of the features in the customize fragment as follows:

- Change style of the timer
- Change the timespan of the timer
- Change the color of the timer and background
- Change the color of the timer to be changing gradiently
- Attach a pictogram or a timer to the timer
- Change the done picture of the timer
- Save the timer
- Start the timer

Det hører næsten mere til et design afsnit

5.4.1 Architecture of Customize

The customize fragment consists of four main elements, the first element is the style picker where the user can change the style of the timer. The second element is the time wheels where the user can change the timespan of the timer. The third is the color pickers where the user can change the colors of the time left, the frame, and the background of the timer. The last element is

the advanced element where the modifications like attachment can be found. Figure 5.9 show how the fragment has been outlined.

Figure 5.9: An outline of the elements of the customize fragment.

In the actual customize class has all buttons been initialized as global objects for quick reference. Since everyone on the WOMBAT project were new to Android programming when the structure of the class were stablished, is all functionality of the items in the customize fragment in this class. For

further development would a refactoring of the entire class be adviced such that all buttons in WOMBAT is handled like the WDialog.
To be able to get an overview of the items in the customize fragment has all items been assigned a method which initializes the item e.g. the style chooser in figure 5.10. These methods are then referenced from the onCreate method in figure 5.11.

```
1  private void initStyleChoser() {
     hourglassButton = (Button) getActivity().findViewById(
3        R.id.houglassButton);
     hourglassButton.setOnClickListener(new OnClickListener() {
5
       public void onClick(View v) {
7        selectStyle(formFactor.Hourglass);
       }
9    });

11   timetimerButton = (Button) getActivity().findViewById(
         R.id.timetimerButton);
13   timetimerButton.setOnClickListener(new OnClickListener() {

15     public void onClick(View v) {
         selectStyle(formFactor.TimeTimer);
17     }
     });
19
     progressbarButton = (Button) getActivity().findViewById(
21       R.id.progressbarButton);
     progressbarButton.setOnClickListener(new OnClickListener() {
23
       public void onClick(View v) {
25       selectStyle(formFactor.ProgressBar);
       }
27   });

29   digitalButton = (Button) getActivity().findViewById(R.id.
       digitalButton);
     digitalButton.setOnClickListener(new OnClickListener() {
31
       public void onClick(View v) {
33       selectStyle(formFactor.DigitalClock);
       }
35   });
}
```

Figure 5.10: The style choser initialization method, which utilizes the selectStyle that changes the style of the timer and highlights the button.

```
public void onActivityCreated(Bundle savedInstanceState) {
  super.onActivityCreated(savedInstanceState);
  currSubP = new SubProfile("", "", 0xff3D3D3D, 0xffFF0000, 0
      xffB8B8B8,
       0xff000000, 600, false);
  currSubP.save = false;
  currSubP.saveAs = false;

  /********* TIME CHOSER *********/
  initStyleChoser();

  /********* TIMEPICKER *********/
  initTimePicker();

  /********* COLORPICKER *********/
  initColorButtons();

  /********* ATTACHMENT PICKER *********/
  initAttachmentButton();

  /******** BOTTOM MENU **********/
  initBottomMenu();
}
```

Figure 5.11: The onCreate method, which calls the button initializers in the same order as they are shown in the layout.

For successors on the WOMBAT project does Eclipse have a keyboard shortcut (F3) which jumps to the method your cursor is at, that can be used to quickly find e.g. the color buttons in the customize class.

5.4.2 Buttons in Customize

There is roughly four kinds of buttons in the customize fragment:

- Start, Save, and style buttons
- Time picker wheels
- Color picker
- Attachment and Done picture button

Start Button

The style, save, start, and attachment buttons are ordinary Android buttons with a picture attached to the top side. The differences of these buttons is the onclick event handler which is slightly different from button to button, figure 5.12 is the source code of the start button.

```
private void initStartButton() {
 startButton = (Button) getActivity().findViewById(
     R.id.customize_start_button);
 Drawable d;
 if (currSubP.saveAs) {
   d = getResources().getDrawable(R.drawable.thumbnail_start);
   startButton.setOnClickListener(new OnClickListener() {

     public void onClick(View v) {
       currSubP.addLastUsed(preSubP);
       guard.saveGuardian(currSubP);
       currSubP.select();
       Intent i = new Intent(
           getActivity().getApplicationContext(),
           DrawLibActivity.class);
       startActivity(i);
     }
   });
 } else {
         ...
     }
   });
 }

 startButton
 .setCompoundDrawablesWithIntrinsicBounds(null, d, null, null);
}
```

Figure 5.12: The source code of the start button, which sets the top image of the button to the drawable `thumbnail_start_gray`

Time Picker Wheels

The time picker wheels is a wheel widget created by `yuri.kanivets@gmail.com` [7], these wheels can be customized to mimic the time pickers from the iPhone. All functionality of the wheel widget is handled by the widget itself and only requires the widget to be imported to Eclipse and added as a

library, which is a built in feature in the Android development environment. To avoid a situation where the user would try to set the time to more than 60 minutes, did we implement a functionality which sets the seconds wheel to zero whenever the minutes wheel is set to 60, this can be seen in figure 5.13.

```
private int previousMins;
private int previousSecs;
private void initTimePicker() {
  /* Create minute Wheel */
  mins = (WheelView) getActivity().findViewById(R.id.minPicker);
  mins.setViewAdapter(new NumericWheelAdapter(getActivity()
      .getApplicationContext(), 0, 60));
  mins.setCyclic(true);

  /* Add on change listeners for both wheels */
  mins.addChangingListener(new OnWheelChangedListener() {
    public void onChanged(WheelView wheel, int oldValue, int
        newValue) {
      updateTime(mins.getCurrentItem(), secs.getCurrentItem());

      if (mins.getCurrentItem() == 60) {
        previousMins = 60;
        previousSecs = secs.getCurrentItem();

        secs.setCurrentItem(0);
        secs.setViewAdapter(new NumericWheelAdapter(getActivity
            ()
            .getApplicationContext(), 0, 0));
        secs.setCyclic(false);
      } else if (previousMins == 60) {
        secs.setViewAdapter(new NumericWheelAdapter(getActivity
            ()
            .getApplicationContext(), 0, 60));

        secs.setCurrentItem(previousSecs);
        secs.setCyclic(true);
        previousMins = 0;
      }
    }
  });
}
```

Figure 5.13: The time picker wheels, with the functionality which ensures the time is never set to more than 60 minutes

Color Picker

Just like the time picker wheels did we utilize the widget functionality in Android to implement a widget called AmbilWarna (means Take Color in Indonesian), created by Yuku Sugianto [6], to handle the color picker. The color picker widget is a custom dialog which returns the color picked by the user in the widget, the implementation of the color picker widget can be found in figure 5.14.

```
colorGradientButton2 = (Button) getActivity().findViewById(
    R.id.gradientButton_2);
setColor(colorGradientButton2.getBackground(), currSubP.
    timeSpentColor);
colorGradientButton2.setOnClickListener(new OnClickListener() {
  public void onClick(View v) {
    AmbilWarnaDialog dialog = new AmbilWarnaDialog(getActivity()
        ,
        currSubP.timeSpentColor, new OnAmbilWarnaListener() {
      public void onCancel(AmbilWarnaDialog dialog) {
      }

      public void onOk(AmbilWarnaDialog dialog, int color) {
        currSubP.timeSpentColor = color;
        setColor(colorGradientButton2.getBackground(),
            currSubP.timeSpentColor);
      }
    });
    dialog.show();
  }
});
```

Figure 5.14: The color picker widget implement on the second "Time Left" button

Attachment Button

The attachment buttons are implemented like the other buttons, but they make use of a custom dialog in WOMBAT. We created this dialog, because the standard dialogs in Android were very different from the rest of the WOMBAT design. Also were we unable to define exactly what kind of buttons we wanted in our dialogs.
Therefor did we implement the custom dialog which would match the rest of our WOMBAT design and give the ability to control exactly what the but-

tons on the dialog should look like and what they should do when clicked. Figure 5.15 is an example of how a dialog could be specified.

```
final WDialog attachment1 = new WDialog(getActivity(),
  R.string.attachment_dialog_description);

ModeAdapter adapter = new ModeAdapter(getActivity(),
  android.R.layout.simple_list_item_1, mode);

attachment1.setAdapter(adapter);

attachment1.addButton(R.string.cancel, 1,
  new OnClickListener() {

  public void onClick(View arg0) {
    attachment1.cancel();
  }
});
```

Figure 5.15: Initialization of the dialog, that appears when the attachment button is clicked, the onItemClickListener is not shown here because of lack of space on the page.

5.5 Backend Library

The two libraries in WOMBAT each control a key functionality of the application, storage and drawing. These functionalities could have been implemented in the WOMBAT project, thereby avoiding situations where the project is compiled with an old library.
A problem with external libraries is that debugging can be difficult in Eclipse if the developer want to change directly in the library. This is because they require the external library to be imported in Eclipse and set as an external library in the Android properties of the project.
The benefits of using external libraries, is that they can be tested and debugged on an external test project instead of depending on a working main project. External libraries can be modelled by dummy functionalities in the main project untill they have been tested. Further, it becomes possible to change the library operation and thereby shifting entire modules in the application. Fx could the canvas drawings be changed to OpenGL drawings just by changing the canvas DrawLib in WOMBAT with a DrawLib that runs on OpenGL.

5.5.1 TimerLib

TimerLib containts the functionality that store and manage all the different objects that WOMBAT uses. The main goal behind TimerLib was the provide a set of simple methods and objects which the rest of WOMBAT could use without great knowledge how TimerLib works. This goal was achieved by being strict about visibility of attributes and methods. The public attributes and methods are designed so they can be used outside TimerLib without having knowledge of how TimerLib works.

Classes

Below is all the TimerLib classes described.

Guardian

The guardian class is an object that represent the guardian in our system. The idea behind the Giraf system is that only one guardian can be logged in at a given time, that is why the guardian class uses a singleton pattern to enforce WOMBAT never have more than one guardian logged into the application. When initiating the guardian class for the first time it is important to use the proper method. The method can be seen in code snippet: 5.16. No matter how many times you class this method, it will always return the same guardian due to the singleton pattern.

```
private static Guardian _instance = null;

public static Guardian getInstance(long m_childId, long
    m_guardianId, Context c, ArrayList<Art> artList){
     if(_instance == null){
       _instance = new Guardian();
       _instance.ArtList = artList;
       TimerHelper help = new TimerHelper();
       _instance.profileID = m_childId;
       _instance.guardianId = m_guardianId;
       _instance.m_context = c;
       _instance.oHelp = new Helper(c);
       long appId = _instance.findAppId();
       _instance.guardianId = _instance.findGuardianId();
       _instance.createChildren();
       _instance.crud = new CRUD(appId, c);
       _instance.crud.loadGuardian(_instance.guardianId);
       help.loadPredef();
       //_instance.crud.initLastUsed(_instance.m_oGuard.getId());
       _instance.publishList();
       //crud.retrieveLastUsed(m_guardianId);
     }
       return _instance;
   }
```

Figure 5.16: Code snippet how to initiate the guardian class.

Whenever the back button is pressed while WOMBAT is active, WOMBAT will call the reset method which will reset the guardian and exit the application. The reset method can be seen in code snippet: 5.17. The reset method only sets the guardian instance to null, so next time the getInstance method is called it will create a new guardian object, we are doing so because java has a garbage collector and we can therefore not manually destroy an object.

```
public void reset(){
     _instance = null;
   }
```

Figure 5.17: Code snippet how to reset the guardian class.

The guardian class will create a guardian if WOMBAT does not receive a valid guardian Id. This is to ensure that WOMBAT can run no matter what. In code snippet: 5.18 you can see the method which checks and creates a new guardian.

```
1  private long findGuardianId() {
     if(guardianId != -1){
3      // Does the original guard exist
       m_oGuard = oHelp.profilesHelper.getProfileById(guardianId)
           ;
5    } else {
       m_oGuard = null;
7    }
     // If not, try the default guard
9    if(m_oGuard == null){
       for (Profile p : oHelp.profilesHelper.getProfiles()) {
11       if(p.getFirstname().equals("Mette") && p.getSurname().
             equals("Als")){
           m_oGuard = p;
13         break;
         }
15     }
       // If thats not valid either, make the default guard
17     if(m_oGuard == null){
         m_oGuard = new Profile("Mette", "Als", null, 1,
             88888888, null, null);
19       m_oGuard.setId(oHelp.profilesHelper.insertProfile(
             m_oGuard));
         oHelp.appsHelper.attachAppToProfile(m_app, m_oGuard);
21       oHelp.profilesHelper.setCertificate("
             jkkxlagqyrztlrexhzofekyzrnppajeobqxcmunkqhsbrgpxdtqgygnmbhrgnpphaxsj
             ", m_oGuard);
       }
23   }
     return m_oGuard.getId();
25 }
```

Figure 5.18: Code snippet how to reset the guardian class.

The Guardian class will also create a set of children if the current guardian does not contain any children. The code snippet: **??** shows how this is done.

```
1  private void createChildren() {
     if(oHelp.profilesHelper.getChildrenByGuardian(m_oGuard).
        isEmpty()){
3      List<String> names = new ArrayList<String>();
       names.add("Sigurd");
5      names.add("Marcus");
       names.add("Emil");
7      names.add("Lukas");
       names.add("Mads");
9      names.add("Nikolaj");

11     for (String s : names) {
         Profile newProf = new Profile(s, " ", null, 3, 99999999,
             null, null);
13       newProf.setId(oHelp.profilesHelper.insertProfile(newProf
             ));
         oHelp.profilesHelper.attachChildToGuardian(newProf,
             m_oGuard);
15       oHelp.appsHelper.attachAppToProfile(m_app, newProf);
       }
17   }
   }
```

Figure 5.19: Code snippet how to reset the guardian class.

The guardian class generates an ArrayList of the object type Child. the ArrayList contains last used, predefined, and all the children that belong to the guardian. In code snippet: 5.20 you can see how this ArrayList is generated. It is important to know that any changes performed on this ArrayList will not be saved, the ArrayList is generated based on three other ArrayLists; lastUsed, predefined, and all the children. Last used is always in the top of the list and right after is the predefined and the children is then added alphabetically.

```
    public ArrayList<Child> publishList(){
2   if(_sortedList == null){
      _sortedList = new ArrayList<Child>();
4   }
    _sortedList.clear();
6   Child lastUsedChild = new Child("Last Used");
    lastUsedChild.setProfileId(-3);
8   lastUsedChild.SubProfiles().addAll(reverse(lastUsed()));
    lastUsedChild.setLock();
10  lastUsedChild.lockDelete();
    _sortedList.add(lastUsedChild);
12
    Child predefChild = new Child("Predefined Profiles");
14  predefChild.setProfileId(-2);
    Collections.sort(predefined());
16  predefChild.SubProfiles().addAll(predefined());
    predefChild.setLock();
18  predefChild.lockDelete();
    _sortedList.add(predefChild);
20  if(_guard != null){
      Collections.sort(_guard);
22    for(Child p : _guard){
        Collections.sort(p.SubProfiles());
24    }
      _sortedList.addAll(_guard);
26  }
    return _sortedList;
28 }
```

Figure 5.20: Code snippet how to reset the guardian class.

Child

The child class is an object that represent the either; last used, predefined, or a child. The child object no matter what it represent has a collection of SubProfiles, which is the variety of timers. You are not allowed to save or delete from the last used and predefined list, the child class therefore provides two kind of locks, a save lock and a delete lock. When the save lock is true, you cannot save the SubProfiles on the child object. When the delete lock is false, you cannot delete SubProfiles from the child object. Both delete and save lock is read only for projects outside TimerLib, this is because the WOMBAT should never lock any child objects that is loaded from the OasisLocalDatabase. The lock methods can be seen in code snippet: 5.22.

```
/**
 * Used to check if you can save on a specific Child
 * @return boolean, if true, you cannot save on the child. if
    false you can save.
 */
public boolean getLock(){
  return _lock;
}
/**
 * Enables the lock
 */
void setLock(){
  this._lock = true;
}
/**
 * Enable delete lock, so you cannot delete from a certain
    child.
 * This will always be used on lastUsed and predefined.
 */
void lockDelete(){
  _deleteCheck = false;
}
/**
 * Used to check if you can delete a subprofiles on a specific
     child
 * @return boolean, true = you can delete: false = you cannot
    delete.
 */
public boolean deleteCheck(){
  return _deleteCheck;
}
```

Figure 5.21: Code snippet how to reset the guardian class.

The child class contains a select method that selects a certain child so it can be called reused later on. The select method is called whenever you highlight a child, last used or predefined in the WOMBAT application. whenever you want to pick the selected child you simply call getChild from the guardian class. The idea behind this method is to make it easier to work with the same object across multiple projects that uses the same child object.

The child class provides methods for saving and removing. The save method saves a certain SubProfile either as a new SubProfile or over-

rides a old SubProfile, the save method calls saveChild from guardian to make sure it is saved in the OasisLocalDatabase. The remove method removes a certain SubProfile from the child and calls removeSubprofile-FromProfileId from guardian to delete the SubProfile from the Oasis-LocalDatabase. You can see save and remove method in code snippet:

```
  public SubProfile save(SubProfile p, boolean override){
2
    if(!override){
4     p.setDB_id(getNewId());
      p.setId(guard.getId());
6   }
      this.SubProfiles().add(p);
8     guard.saveChild(this, p);

10  return p;
  }
12
  public void remove(SubProfile p) {
14      _profileList.remove(p);
        guard.crud.removeSubprofileFromProfileId(p, this.
            getProfileId());
16}
```

Figure 5.22: Code snippet how to reset the guardian class.

SubProfile

The SubProfile class is a super class for all the different kind of timer which WOMBAT supports. The idea behind having a super class that represent the timers is so you can have a collection of all types of timers, without having to worry. It was original meant for the SubProfile and the classes which inherits from this class to have methods that should draw the timers, you can read more about it in this section: 5.1. Check code snippet: 5.23 for an example how SubProfile and classes which inherits from it was meant to work.

```
Hourglass hourglass = new Hourglass("Timeglas - 30 sek", "
    Timeglas - (0:30)", 0xff3D3D3D, 0xffffffff, 0xffB8B8B8, 0
    xff000000, 30, false);
    ProgressBar progressbar = new ProgressBar("ProgressBar - 30
        sek", "ProgressBar - (0:30)", 0xff3D3D3D, 0xffffffff, 0
        xffB8B8B8, 0xff000000, 30, false);
    DigitalClock digitalclock = new DigitalClock("DigitalClock -
         30 sek", "DigitalClock - (0:30)", 0xff3D3D3D, 0xffffffff
        , 0xffB8B8B8, 0xff000000, 30, false);
    TimeTimer timetimer = new TimeTimer("Ur - 30 sek", "Ur -
        (0:30)", 0xff3D3D3D, 0xffffffff, 0xffB8B8B8, 0xff000000,
        30, false);
    ArrayList<SubProfile> timers = new ArrayList<SubProfile>();
    timers.add(hourglass);
    timers.add(progressbar);
    timers.add(digitalclock);
    timers.add(timetimer);
    for(SubProfile sp : timers){
      View v = sp.draw();
    }
```

Figure 5.23: Code snippet how to reset the guardian class.

All SubProfiles got two different id's. The first id is used internal in the TimerLib, this id is used to check if the SubProfile already exists in the last used list and check if the SubProfile already exist on a certain child object and thereby overriding that SubProfile. The internal id is provided by the guardian class. The other id matches the SubProfile's id in the OasisLocalDatabase, the id is used for deleting a SubProfile in the OasisLocalDatabase.

WOMBAT never loads an original SubProfile into the customize fragment, WOMBAT instead uses the copy method which simply return a copy of the SubProfile. The idea behind using a copy is because, we want to allow the user to change options on a SubProfile and start the timer without saving the data. If the user saves the SubProfile, it will be replaced with the copy, and if the user use save as it will save it as a new SubProfile.

It is possible to convert a SubProfile into any other kind of SubProfile by using the four kind of convert methods. You can see these four methods in code snippet: 5.24.

```
public SubProfile toHourglass() {
  Hourglass form = new Hourglass(this.name, this.desc, this.
      bgcolor, this.timeLeftColor, this.timeSpentColor, this.
      frameColor, this._totalTime, this.gradient);
  form.setId(this.getId());
    form.setAttachment(this._attachment);
    form.setDoneArt(this._doneArt);
  return form;
}

public SubProfile toProgressBar() {
  ProgressBar form = new ProgressBar(this.name, this.desc,
      this.bgcolor, this.timeLeftColor, this.timeSpentColor,
      this.frameColor, this._totalTime, this.gradient);
  form.setId(this.getId());
    form.setAttachment(this._attachment);
    form.setDoneArt(this._doneArt);
  return form;
}

public SubProfile toTimeTimer() {
  TimeTimer form = new TimeTimer(this.name, this.desc, this.
      bgcolor, this.timeLeftColor, this.timeSpentColor, this.
      frameColor, this._totalTime, this.gradient);
  form.setId(this.getId());
    form.setAttachment(this._attachment);
    form.setDoneArt(this._doneArt);
  return form;
}

public SubProfile toDigitalClock() {
  DigitalClock form = new DigitalClock(this.name, this.desc,
      this.bgcolor, this.timeLeftColor, this.timeSpentColor,
      this.frameColor, this._totalTime, this.gradient);
  form.setId(this.getId());
    form.setAttachment(this._attachment);
    form.setDoneArt(this._doneArt);
  return form;
}
```

Figure 5.24: Code snippet how to reset the guardian class.

Whenever a SubProfile is started it will be added to the last used list, if the SubProfile already exists on the SubProfile it will be removed and added again, that way we ensure that the SubProfile on the last used

is the new version of the SubProfile and it is on top of the last used list. You can see the method for adding a SubProfile in code snippet: **??**.

```
1  public void addLastUsed(SubProfile oldProfile){
     if(oldProfile == null){
3      guard.addLastUsed(this);
     } else {
5    this._id = oldProfile._id;
     this.refPro = oldProfile.DB_id;
7    long ref = 0;
     for(Child c : guard.Children()){
9      for(SubProfile p : c.SubProfiles()){
         if(p.getId() == this.getId()){
11         ref = c.getProfileId();
         }
13     }
     }
15   for(SubProfile p : guard.predefined()){
       if(p.getId() == this.getId()){
17       ref = -2;
       }
19   }
     this.refChild = ref;
21   guard.addLastUsed(this);
     }
23 }
```

Figure 5.25: Code snippet how to reset the guardian class.

A SubProfile may contain two kind of attachments; an attachment which will be placed next to the running timer and an attachment which changes the done screen pictograms. You can read more about the Attachment class in this section.

OasisLocalDatabase saves the SubProfile as a hashmap, for that cause does SubProfile provide a method for generating a hashmap with all the needed attributes. You can see this method in code snippet: 5.26.

```
public HashMap<String, String> getHashMap(){
  HashMap<String, String> map = new HashMap<String, String>();
  map.put("db_id", String.valueOf(this.getDB_id()));
  map.put("type", this.formType().toString());
  map.put("Attachment", String.valueOf(this._AttaBool));
  map.put("Name", this.name);
  map.put("desc", this.desc);
  map.put("bgcolor", String.valueOf(this.bgcolor));
  map.put("timeLeftColor", String.valueOf(this.timeLeftColor)
      );
  map.put("timeSpentColor", String.valueOf(this.timeSpentColor
      ));
  map.put("frameColor", String.valueOf(this.frameColor));
  map.put("totalTime", String.valueOf(this.get_totalTime()));
  map.put("gradient", String.valueOf(this.gradient));
  map.put("save", String.valueOf(this.save));
  map.put("saveAs", String.valueOf(this.saveAs));
  map.put("refChild", String.valueOf(this.refChild));
  map.put("refPro", String.valueOf(this.refPro));
  map.put("timeKey", String.valueOf(this.timeKey));
  if(this._doneArt != null){
    map.put("doneArtType", String.valueOf(this._doneArt.
        getForm()));
    switch(this._doneArt.getForm()){
    case SingleImg:
      map.put("doneArtPic", String.valueOf(this._doneArt.
          getImg().getId()));
      map.put("doneArtLeftPic", String.valueOf(-1));
      map.put("doneArtRightPic", String.valueOf(-1));
      break;
    case SplitImg:
      map.put("doneArtPic", String.valueOf(-1));
      map.put("doneArtLeftPic", String.valueOf(this._doneArt.
          getLeftImg().getId()));
      map.put("doneArtRightPic", String.valueOf(this._doneArt.
          getRightImg().getId()));
      break;
    }
  } else {
    map.put("doneArtType", String.valueOf(formFactor.undefined
        ));
    map.put("doneArtPic", String.valueOf(-1));
    map.put("doneArtLeftPic", String.valueOf(-1));
    map.put("doneArtRightPic", String.valueOf(-1));
  }
  // TODO: Attachment this needs some testing and remember to
      add it to the crud!!!!
  if(this._AttaBool){
    map = this._attachment.getHashMap(map);
  } else {
    Attachment tempAttachment = new Attachment();
    map = tempAttachment.getHashMap(map);
  }
  return map;
}
```

Figure 5.26: Code snippet how to reset the guardian class.

TimeTimer - Hourglass - ProgressBar - DigitalClock

These four classes inherit from SubProfile, and it currently the only kind of timers WOMBAT supports.

Attachment

The attachment class is a super class and does only contain the base methods of the inherited classes. The idea behind having a super class is for making TimerLib more dynamic.

Timer

The timer class inherits from the attachment class. The timer class represent a SubProfile as an attachment, the timer class were made to simplify the attachment methods and only display those other projects might need. The timer class support all kind of SubProfile objects. The timer class contains a method which returns a hashmap, that is used when saving an SubProfile with an attachment into the OasisLocalDatabase. You can see the method in code snippet: 5.27. The timer class can only be set as an attachment next to a SubProfile timer.

```
1  public HashMap getHashMap(HashMap map){
     //Defines what kind of attachment it is
3    map.put("AttachmentForm", String.valueOf(this.getForm()));

5    //Timer
     map.put("timerForm", String.valueOf(this._form));
7    map.put("_bgColor", String.valueOf(this._bgColor));
     map.put("_frameColor", String.valueOf(this._frameColor));
9    map.put("_timeLeftColor", String.valueOf(this._timeLeftColor
         ));
     map.put("_timeSpentColor", String.valueOf(this.
         _timeSpentColor));
11   map.put("_gradient", String.valueOf(this._gradient));

13   //SingleImg
     map.put("singleImgId", String.valueOf(-1));
15
     //SplitImg
17   map.put("leftImgId", String.valueOf(-1));
     map.put("rightImgId", String.valueOf(-1));
19
     return map;
21 }
```

Figure 5.27: Code snippet how to reset the guardian class.

SingleImg

The SingleImg class inherit from Attachment, this class represent one object of the type Art, you can read more about the Art class later in this section. The SingleImg object can be set either as an attachment next to a SubProfile timer or as a pictogram for the done screen. SingleImg contains the a hashmap method with same structure as the Timer class has.

SplitImg

The SplitImg class inherit from Attachment, this class represent two object of the type Art, you can read more about the Art class later in this section. The SingleImg object can be set either as an attachment next to a SubProfile timer or as pictograms for the done screen. SplitImg contains the a hashmap method with same structure as the Timer class has.

Art

The art class is used for generating pictograms as objects in TimerLib. The art class constructor takes three arguments; The first argument is the path to the pictogram, the path is an int that basically is the id from the Resource class in Android. The second argument is a caption which is a string. The caption is used to add text to a pictogram. The last argument is an id as int, this an ad hoc solution and not meant to be working like it does. The id need to be the position it got in the ArtList from guardian. In future development, this id should be set by the TimerLib and not by the WOMBAT project. If one was to implement sounds into WOMBAT, the art class would be the place to do it.

formFactor

The formFactor class is a enum class which is used to manage object types. It contains nine enums which are; undefined, SubProfile, Hourglass, TimeTimer, ProgressBar, DigitalClock, Timer, SingleImg, and SplitImg. They all represent their own kind of object. The "'undefined"' enum is used by the super classes which should never be used.

TimeHelper

The TimerHelper class is contains two functions; the first function is its main function, to generate the predefined SubProfiles. The second function is a method which generates test data.

CRUD

The CRUD class responsibility is to create, retrieve, update, and delete

guardian, child and SubProfile objects in the OasisLocalDatabase. The crud class is only called from within the guardian class, this is done so it is easy to change the way WOMBAT create, retrieve, update, and delete objects if WOMBAT was ever to be released as a fully independent application.

Class diagram

You can see a simplified class diagram on figure: 5.28.

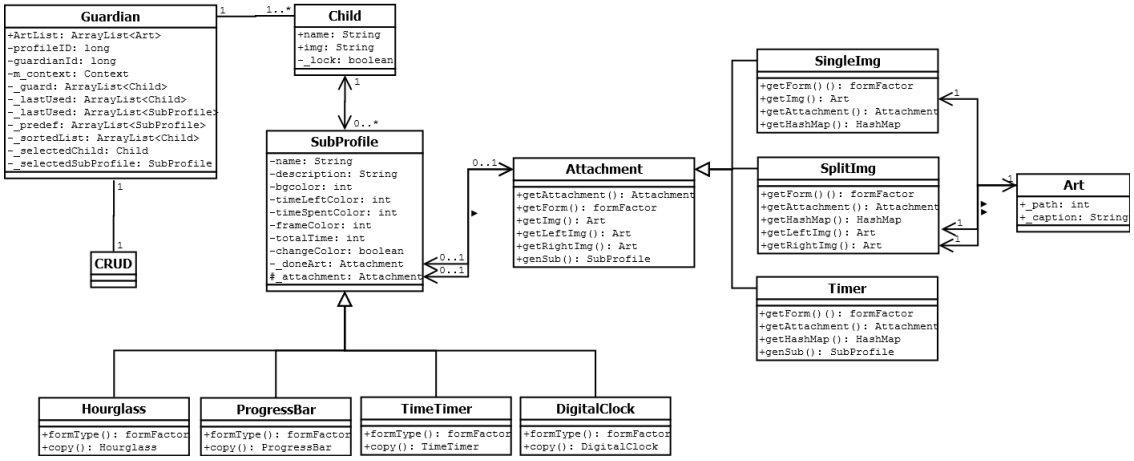


Figure 5.28: Dependecy diagram of WOMBAT projects

5.5.2 DrawLib

DrawLib contains the functionality that draws timers according to the settings set in the main activity. The main functionality in the DrawLib is that it can be either full screen, if there is no timer or pictogram attached, or it can be split screen if there is a timer or pictogram attached. This is done by creating the view layout programmatically instead of a static layout, figure 5.29 is an example of this functionality with a full screen timer, figure 5.30 is an example of the functionality with a split screen timer. In both figures the method genDrawView is a method which returns the draw view, that matches the timer specified in customization, figure 5.31 is a code example of genDrawView.

```
1  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
3    requestWindowFeature(Window.FEATURE_NO_TITLE);
     View main_layout = findViewById(android.R.id.content).
         getRootView();
5    main_layout.setSystemUiVisibility(View.STATUS_BAR_HIDDEN);
     Guardian guard = Guardian.getInstance();
7    SubProfile sub = guard.getSubProfile();

9    // Get display size and store it in static variables
     WindowManager wm = (WindowManager) getSystemService(Context.
         WINDOW_SERVICE);
11   Display disp = wm.getDefaultDisplay();
     frameHeight = disp.getHeight();
13   frameWidth = disp.getWidth();

15   if (sub.getAttachment() == null) {
       /* Set the drawing class (which extends View) as the content
           view */
17     View v = genDrawView(sub,frameWidth);
       v.setKeepScreenOn(true);
19     setContentView(v);
     }
21 ...
}
```

Figure 5.29: Example of how DrawLib sets just one timer view.

```
LinearLayout frame = new LinearLayout(this);
frame.setKeepScreenOn(true);
GradientDrawable gd = new GradientDrawable(GradientDrawable.
    Orientation.TOP_BOTTOM, new int[] {sub.bgcolor, 0xFF000000});

...

switch(sub.getAttachment().getForm()){
case Timer:
  frameWidth = frameWidth/2;

  firstView = genDrawView(sub, frameWidth);
  frame.addView(firstView, frameWidth, frameHeight);

  secondView = genDrawView(sub.getAttachment().genSub(),
      frameWidth);
  frame.addView(secondView, frameWidth, frameHeight);
  break;
case SingleImg:
  ...
case SplitImg:
  ...
}

setContentView(frame);
```

Figure 5.30: Example of how DrawLib sets two timers in one view

```
private View genDrawView(SubProfile sub, int frameWidth) {
  switch (sub.formType()) {
  case ProgressBar:
    return new DrawProgressBar(getApplicationContext(), sub,
        frameWidth);
  case Hourglass:
    return new DrawHourglass(getApplicationContext(), sub,
        frameWidth);
  case DigitalClock:
    return new DrawDigital(getApplicationContext(), sub,
        frameWidth);
  case TimeTimer:
    return new DrawWatch(getApplicationContext(), sub,
        frameWidth);
  default:
    return null;
  }
}
```

Figure 5.31: The genDrawView, which generates the draw view matching the timer

Canvas and OpenGL comparision

The timers in DrawLib are drawed with the Android canvas object, this could also have been done by creating and modelling an object in OpenGL. The major difference between the two approaches is that OpenGL is used to draw three dimensional objects and Android canvas can only draw in two dimensions. Drawing on a canvas is like drawing on coordinates, while drawing in OpenGL varies in the way that one will be drawing in triangles. Drawing in OpenGL is therefor much different from the way one would normally draw, thereby more time consuming. Since all timers in WOMBAT can be modelled in two dimensions, the only benefits of OpenGL compared to canvas is that one would be able to move the camera around or change the lighting in OpenGL.

All timers are classes which inherits the view class, the view class draws in a method called onDraw which is called the first time it is opened and whenever the method `invalidate()` is invoked. The timers draw inside the onDraw method, as seen in figure 5.32.

```
protected void onDraw(Canvas c) {
  super.onDraw(c);

  ... // Initialization of time + Draw Background and frame

  /* Draw the backgroundcolor inside the frame */
  paint.setColor(background);
  r.set(r.left + 2, r.top + 2, r.right - 2, r.bottom - 2);
  c.drawRect(r, paint);

  /* Draw the timespent color (on the right) on top of the
     timeleft */
  paint.setColor(timespent);
  r.set(left + 3, top + 3, left + width - 3, top + height - 3);
  c.drawRect(r, paint);

  if (endTime >= System.currentTimeMillis()) {
    timenow = endTime - System.currentTimeMillis();
    double percent = (timenow) / totalTime;

    paint.setColor(timeleft2);
    r.set((int) ((left + 3) + ((width - 5) * (1-percent))), top
       + 3, (left + 3) + width - 5, top
        + height - 3);
    c.drawRect(r, paint);

    /* Draw the gradient color */
    ...

    /*************** IMPORTANT **************/
    /* Recalls Draw! */
    invalidate();
  } else {
    paint.setColor(timespent);
    r.set(left + 3, top + 3, left + width - 3, top + height - 3)
        ;
    c.drawRect(r, paint);
  }
}
```

Figure 5.32: The onDraw method of the progress bar.

The progress bar and the hourglass both draws according to how many percent of the total time is left, while the digital watch and the time timer draws according to the exact time left. All timers evaluates the time left

according to the number of milliseconds elapsed, which is provided by the system timer.

Drawing can be time consuming and to optimize the timers, the background and frame is painted and stored as a bitmap file at the initialization of the view. The stored bitmap can then be redrawed on the canvas without having to be recalculated, this was especially useful when drawing the digital watch since all numbers has to be redrawn every time the view is reevaluated. With the bitmap it is possible to draw the numbers once and then blank out the lines not needed when reevaluating.

CHAPTER 6

Test

The WOMBAT application is tested through dynamic black box testing, which means that we run the application, and test the functionality through the user interface, without viewing the code. We have made test design and test cases for some of the most important functionality, so that we could outsource the test to one of the other project groups, if they had time to help us. This was not the case, so we did the testing ourselves.

6.1 Test Design

The test designs are split into four schemes, which are listed in this section.

Identifier Save and load.

Feature Saving and loading configurations.

Approach NB! Check if the configuration has changed in the database by closing the application and resetting the memory (RAM).

- Create a configuration in three ways:
  - Click "New Template" and click "Save As".

  - Edit a current configuration and click "Save As".
  - Check if it is possible to "Save As" a configuration in "Predefined" or "Last Used".
  - Check if loaded settings are the same as the when they were saved.
- Edit and save (by clicking "Save") each of the configuration:
  - Check if it is possible to save configurations in "Predefined" or "Last Used".
  - Check if they have the settings they were saved with.
  - Check if there is any duplicates of any of the configurations.
  - Check if any other configuration was changed while editing.

Test case ID

1. Check save as functionality - *saveAs#1*
2. Check save functionality - *save#1*

Pass/fail criteria Pass

- It is possible to create a new configuration by clicking "New Template" and "Save As".
- It is possible to make a new configuration by clicking "Save As" on any configuration.
- No profiles in "Last Used" or "Predefined" is editable.
- It is not possible to save new profiles to "Last Used" or "Predefined".

Fail

- If any of the above does not hold.
- When saving with "Save" the configuration is being duplicated.
- When saving another configuration is being altered.

Identifier Last used is updated correct

Feature When a timer has been used, it should lie in the top of the list of last used configurations.

Approach

1. Start any timer
    - Check if the timer has been added to the last used list
2. Repeat step [1] 7 times with different timers
3. Start any of the timers in "Last Used" and check if it is being moved to the top of the list.

Test case ID

1. Check "last used" functionality - *checkLastUsed#1*

Pass/fail criteria Pass

- Every time a timer used, it is saved on the top of the "Last Used" list.

Fail

- If, in any case, a timer is not saved in the "Last Used" list after it has been used.

Identifier Highlight is working

Feature When choosing a child or a configuration this should be highlighted, and when Wombat is started from the GIRAF launcher a child is chosen, this should be highlighted.

Approach

1. Test if children and configurations is being highlighted when they are chosen.
2. Test if the child chosen in the GIRAF launcher is being highlighted in Wombat.
3. Test if the child chosen is still highlighted after saving.

NOTE! An element in the child list is chosen if there is configurations in the configurations list, and an element in the configurations list is chosen if there is loaded a configuration in the edit screen.

Test case ID

1. Check if list elements is highlighted when clicked - *hOnClick#1*
2. Check if child is still highlighted after save - *stillHAfterSave#1*
3. Check if the right child is highlighted after launch through the GIRAF launcher - *hChildOnLaunch#1*

Pass/fail criteria Pass

- If list elements are always highlighted when selected.

Fail

- If a child is selected and it is not highlighted.
- If a configuration is selected and it is not highlighted.
- If nothing is highlighted when starting Wombat from the GIRAF Launcher

---

Identifier Deviation in time on done activity

Feature When the timer has run out, the "Done" screen will appear.

Approach

1. Run a timer with any timespan, and wait for the "Done" screen to appear
    - Use an independent stopwatch to verify the time it takes to show the "Done" screen.
    - Verify that there is no more than a 2 second deviation in time, from the time has run out until the "Done" screen appears.
2. Repeat step [1] 3 times with different timespans.
3. Do step [2] again with any timer with a "Digital Watch" attached.
4. Start a timer with any timespan, click the "back" button, and verify that the "Done" screen do not show up randomly.

Test case ID

1. Check if timer matches real-life time - *checkTimerTime#1*
2. Check if the done screen appears when it should - *checkDoneFunc#1*

Pass/fail criteria Pass

- If the "Done" screen appears no more than two seconds after the time has run out.
- If any timer is not deviating more than two seconds from real-world time.

Fail

- The "Done" screen appears even though the timer is not running anymore or the timer is not finished.

6.2 Test Cases

Identifier *saveAs#1*

Test item Functionality to save customized timers in specific lists.

Input spec.

1. Click "New Template", choose a timer type, click "Save As", and choose name and location. Check if the profile was saved in the chosen location and with the chosen name.
2. Choose any configuration, edit the settings, click "Save As", and choose name and location. Check if the profile was saved in the chosen location and with the chosen name.
3. Create a new configuration with random settings and use "Save As" to save it. Go to the saved configuration, and check if the settings has changed since the save.

4. Choose an existing configuration or create a new one, and do step 1. with "Predefined" and "Last Used" as save locations.

5. Do step 1-3 again, but clear the tablet memory before the correctness is checked.

Output spec.

1. Configurations is saved in the chosen locations, unless the chosen locations is "Predefined" or "Last Used".

2. Configurations is saved with the chosen name.

3. Configurations is saved with the chosen settings.

environmental needs

- Tablet running Android 3.2.
- Timer application installed.
- OasisLocalDatabase installed.
- One staff member to perform the test.

Identifier *save#1*

Test item Functionality to save customized or predefined timers in the highlighted child, without having to choose name and location.

Input spec.

1. Check if it is possible to save configurations in "Predefined" or "Last Used" by choosing one of the configurations in each list, edit some settings, and press "Save".

2. Select a child, edit the settings, and press "Save". Check if the chosen settings were saved.

3. Select a child, select a configuration among the child's configurations, edit the settings, and press "Save". Check if the chosen settings were saved in the same configuration.

4. Select a child, edit the settings, and press "Save" two times and see if two identical configurations are saved on the given child.
5. Highlight another configuration than the one you have just saved, then highlight the one you just saved and press "Save". Check if there is now saved a duplicate of the first saved configuration.
6. Do step 2-3 again and check if any other configuration was changed during the saving process.

Output spec.

1. Configurations is saved in the highlighted child.
2. When "Predefined" or "Last Used" is highlighted, nothing is saved when the "Save" is pressed.
3. New configurations are saved with the chosen settings.
4. When selecting and saving existing configurations, they are updated with the edited settings.

environmental needs

- Tablet running android 3.2.
- Timer application installed.
- OasisLocalDatabase installed.
- One staff member to perform the test.

Identifier *checkLastUsed#1*

Test item Functionality to save timers into the "Last Used" list every any timer has been run.

Input spec.

1. Run 3 different timers, and see if they were saved on top if the "Last Used" list.
2. Repeat step 1, but clear the tablet memory before "Last Used" is inspected.

Output spec.

1. Whenever a timer has been used, it is saved on top of the "Last Used" list.

environmental needs

- Tablet running android 3.2.
- Timer application installed.
- OasisLocalDatabase installed.
- One staff member to perform the test.

Identifier *hOnClick#1*

Test item Functionality to highlight list items when they are clicked.

Input spec.

1. Select three different list items in both the child list and the configuration list and see if they stay highlighted.

Output spec.

1. When a list item is selected, it is highlighted, and it stays highlighted until another list item is selected.

environmental needs

- Tablet running android 3.2.
- Timer application installed.
- One staff member to perform the test.

Special procedural requirements

- The configurations on every child will always be visible when a child list has been selected. Therefore, make sure that the highlighted child has at least one configuration before testing.
- There is no element in the configuration list if no element in the child list has been selected.

Identifier *stillHAfterSave#1*

Test item Functionality to highlight list items after a save procedure.

Input spec.

1. Select a child and a configuration, edit the settings for the configuration, and click "Save". See if the selected list items stay highlighted after it has been updated.

Output spec.

1. When a child and configuration is selected, and the settings for that configuration is changed and saved, the child list item and configuration list item is still highlighted.

environmental needs

- Tablet running android 3.2.
- Timer application installed.
- One staff member to perform the test.

Special procedural requirements

- The configurations on every child will always be visible when a child list has been selected. Therefore, make sure that the highlighted child has at least one configuration in the list before testing.
- There is no element in the configuration list if no element in the child list has been selected.
- When a configuration is selected, the settings for that configuration is always shown set in the "Customize" menu.

Intercase dependencies *save#1*

Identifier *hChildOnLaunch#1*

Test item Functionality to highlight list items according to the chosen child when the application is launched through the GIRAF launcher.

Input spec.

1. Start the GIRAF launcher and open the timer application.
2. Select a child and note the name of the child.

Output spec.

1. The child selected in the GIRAF launcher is highlighted and the configurations belonging to this child is loaded.

environmental needs

- Tablet running android 3.2.
- Timer application installed.
- GIRAF launcher installed.
- One staff member to perform the test.

Special procedural requirements

- The configurations on every child will always be visible when a child list has been selected. Therefore, make sure that the highlighted child has at least one configuration before testing.
- There is no element in the configuration list if no element in the child list has been selected.

Identifier *checkTimerTime#1*

Test item Functionality which draws and updates the timer according to the time left and ensures the timer ends when the time is up.

Input spec.

1. Run four different timer styles with a static timespan (fx 20 minutes).
2. Each time a timer is started, start a precise independent stopwatch.
3. When the timer reaches zero stop the independent stopwatch.

Output spec.

1. The stopwatch must deviate no more than two seconds from the time selected in **input spec.** step [1].

environmental needs

- Tablet running android 3.2.
- Timer application installed.
- Stopwatch.
- One staff member to perform the test.

Identifier *checkDoneFunc#1*

Test item The "Done" screen appearing when the time has run out.

Input spec.

1. Start a timer at any timespan and let the time run out. See if the "Done" screen appears within two seconds after the time has run out.
2. Start a timer at any timespan and click the "back" button, and wait at least the amount of time the timer would have run, to verify that the "Done" screen do not show up anyways, if the timer has been interrupted.

Output spec.

1. When a timer has been run, and not interrupted, the "Done" screen appears about two seconds after the time has run out.

2. The "Done" screen is only shown if the timer is not interrupted, and the time has run out.

environmental needs

- Tablet running android 3.2.
- Timer application installed.
- Stopwatch.
- One staff member to perform the test.

Intercase dependencies Test case: *checkTimerTime#1*

6.3 Test Results

Test Case ID	Pass	Fail	Notes
saveAs#1	X		
save#1	X		
checkLastUsed#1		X	When tablet memory is cleared, the last used timers disappears.
hOnClick#1	X		
stillHAfterSave#1	X		
hChildOnLaunch#1	X		
checkTimerTime#1	X		
checkDoneFunc#1	X		

Table 6.1: Test results of the black box testing of the WOMBAT application.

6.3.1 Reflections

We did not outsource the test, so we did the testing ourselves. This is not the best way to do testing, because we wrote the program, and the test design and test cases, and thereby have a deeper knowledge about the features we want to test, and therefore we may have done things different than persons with no knowledge about the application would have done.

The only test that did not pass was known beforehand (*checkLastUsed#1*). We had trouble implementing the "Last Used" together with the features of the database developed by the admin-group. This resulted in sporadic system failure, so we decided to only use the tablet memory to save the last used timers, and save the correct implementation of this feature to future development.

During the test we stumbled upon a few vague formulations, which could have lead to misunderstandings and different testing method, if the tests had been outsourced. We learned that it is a good idea to run the test ourselves at first, so that we could correct any formulation errors we found, and then the test could be outsourced. This would require even more time, and is not likely to be done in large scale projects, but could have been done in our case.

6.4 Usability Test

In this section we explore the results of the usability test, which was described in **??**. We focus only on the results for the Timer application, since we are not going to need the results for the other parts of the GIRAF system.

6.4.1 Results and Observations

Through the IDA method, we found six problems with the application, and they are categorized in table 6.2.

All the problems found, besides the color palette, are design related, and can be corrected without changing any of the implemented functionality. They could be solved by slightly changing the design of buttons and adding more confirmation messages.

To solve the color palette problem, a custom palette could be implemented instead of the imported Android palette, or there could be a number of predefined colors the user can choose from.

[H]

Category	Notes
Cosmetic	1: the users were not sure whether the attachments were selected, when they had chosen the attachments. 2: the user had difficulties finding the "Start"-button.
Serious	1: the user had difficulties using the imported Android color palette. 2: deletion by long-click is not intuitive. 3: the "Last Used" list was difficult to find.
Critical	1: when the background of the Timer application had a certain color, the "Gradient"-button was invisible.

Table 6.2: Problems found from the usability test of the WOMBAT Timer application.

6.5 Acceptance Test

The purpose of an acceptance test[1] is to test if the system fits into the context it is designed for. In this case the main objectives is to find out if it feels natural for the educators to use the timer application instead of their regular hourglasses and time timers, and to find out if the children understand the digital version of the different timers.

To test for acceptability, the system needs to be tested in the context it is developed for, and therefore the contact person borrowed the tablet with the timer application for a few days, so that she could use the system in real life scenarios.

6.5.1 Results and Observations

The contact person wrote a diary, to keep track of her experiences with the application. The contact person used the timer application herself, and she let some of the other educators try it as well. The diary can be found in appendix 9.4.

They wrote in the diary that the children understood the meaning of both the timers and the pictograms shown on the tablet. One of the children was more interested in watching the digital timer count down than in the activity he was meant to do, but beyond that there were no problems understanding the system.

Beside the diary, we can analyze on the timers they have used in the acceptance test by looking in the "Last Used" list (see appendix 9.4).

We can see in the list that they have used different timers with different "Done"-pictures and times, only one out of eight timers had a slightly different color. This could indicate that changing the colors on the timers is not very intuitive, or they do not need the functionality anyway.

Tail

Part III

Evaluation

Head

CHAPTER 7

Discussion

7.1 Evaluation of Development Method

In this section, we evaluate on the development method described in **??**.

Agile Development In the beginning of the project we worked towards the vision described in section 2.1. Because other parts of the multi project changed, we had to alter the focus of our project. These changes occurred when we had begun developing, and since we used an agile development method, it was possible for us to change direction instead of starting all over.

Meetings The meetings have been useful, since the individual projects changed when the development begun, and these changes was presented at the meetings. This helped the groups adapt their project to fit in the multi project.

Sprint Length Since meetings were only planned in the beginning and end of each sprint, and these meetings were used to present backlogs and discuss overlapping concerns, it was important that the sprint length was short (7-14 days). This was an appropriate length, because it was possible to stay updated on the progress of the multi project, and adapt the individual project to it.

Product Owner Since we had no product owner, we did not have one person to manage the multi project, and therefore all decisions were made democratically by the multi project groups. This lead to several time consuming discussions of minor details regarding the project.

7.2 Development Tools

As described in chapter 4, we have used pair programming and refactoring. We evaluated on our use of these methods as part of a mini project in the course *Software Engineering*, this evaluation is included here.

7.2.1 Pair programming

We did some pair programming, but we were unable to do all programming in pairs. Since we are three persons in the group, we would either have to work three around one computer, or exclude one person from the pair programming. We chose to exclude one person from the pair programming, also because it would be a waste of resources to work in groups of three since the efficiency does not match the cost, when adding the third person.

One of the downsides of the pair programming technique is that we have been unable to work in pairs outside the university, i.e. when two group members are doing pair programming on a specific part of the code, and one of them got sick, the programming would be continued by only one group member, the third group member would join the programming and thereby pause the tasks he was doing, or the programming of the specific part of code would be paused until both group members would be gathered again.

The code written in pairs, is of higher quality than the code written before we implemented the pair programming technique, and the higher quality code is more readable, and thereby has no need for refactoring. We therefore believe that the use of the pair programming technique has been beneficial. This technique could be useful in our future careers.

7.2.2 Refactoring

There are two main reasons for the need for the code to have high readability and understandability. One reason is that the project will be handed over to the students of next year for further development, so to help them we strive to write understandable code. The other reason is that we are three persons in the group, and we have not all been included in developing every part of

the program, so with understandable code it is easier for the other group members to understand the parts they did not participate in writing.

Some of the refactoring we have done is renaming variables and objects to match the content of them. Also all private variables and objects has been refactored to always start with either "m" or "_".

The use of this technique has helped ourselves to better understand the code, and we believe that when the project is passed on to the next project groups, it will be easier to familiarize themselves with it. We have learned that refactoring is very useful when several people are developing on the same product, and since we are very likely to work in groups later in our careers, we will probably use this technique again.

CHAPTER 8

Conclusion

Konklusionen paa projektet

CHAPTER 9

Perspective

9.1 Future Work

In this section we will list some ideas for further development on the WOMBAT application.

————————————-Test changes————————————-

9.1.1 Ideas based on the usability and acceptance test

In this subsection we will briefly describe changes we would like to implement based on the usability test and the acceptance test.

Color picking

WOMBAT is currently using a color palette which the guardians uses to picking colors with, we would like to change this into a list of predefined colors and an advanced button where they would be able to define new colors with a color palette. This change would make ease the color picking.

Gradient button

The gradient button is hard to spot, it is therefore necessary to improve the layout. It might help if the button layout looks the same as all the other buttons in WOMBAT.

Improve customize fragment buttons

The buttons in WOMBAT need to be redesigned to appear more like buttons. The attachment buttons visual effect after attaching an attachment needs to be improved so it is more clear for the user, when an attachment has been attached.

Delete

It is not intuitive how to delete a SubProfile. It might help to remove the long-click and make a delete button on each SubProfile. Figure: 9.1 illustrates how such a button might look like.

Figure 9.1: Illustration of possible delete button

Last used and Predefined layout changes

The usability test showed that the last used and predefined category were hard to find, these two categories need a design change to look more outstanding compared to the children. This could be achieved by changing the background color of these two categories.

Done screen picture lockdown

The acceptance test showed that it is necessary to improve the lock on the done screen picture. The done screen unlocks by a single touch on the screen, which is issue, should be changed to a long-click instead.

———————————-Ideas———————————-

9.1.2 Ideas

In this subsection we will briefly descrip ideas which can be used for future development of WOMBAT.

Handset Version

It is, at the moment, not possible to install and run the WOMBAT application on an Android handset, but there are several reasons for considering making a handset version of the application as well. This is some of the reasons:

- It might be more convenient to bring a handset than a tablet, when the guardians and children is on a trip.
- There are more smartphone users than tablet users.
- Der er flere damer? Vi er nok noed til at finde paa en grund mere.

Improved Graphics

WOMBAT is currently using canvas for drawing all the timers, which is simple 2d graphic. It is possible to improve the graphic by using OpelGL instead. OpelGL is used for drawing detailed 3d objects. The benefit of improving the graphic in WOMBAT is to make the timers more realistic and look like the timers which the children already uses. There have been conducted a comparison between canvas and OpelGL which can been read about in subsection: 5.5.2.

Overlay timers

This idea is based on the second part of our original idea, which can be seen in section: 2.1. When you exit the timer screen while a timer is running, that timer will be destroyed. Instead of destroying the timer, it should enable an overlay timer which should continue from where the timer left off. The overlay timer should always appear on top of all the other application that might running. This would make it possible for the guardian to conduct multiple tasks without without stopping a timer. When the guardian enters WOMBAT again, it should remove the overlay and continue the timer normally.

Code Architecture

The draw features which is a part of the DrawLib, were original designed to be a part of the TimerLib, the decision of this can be read in section: 5.1. The ideal architecture would be to import the draw methods into each timer class, that way you could call the method on a certain timer object and

return a view. By doing so, you would not have to check on which object you were handling and choose a specific. Check figure: 9.2 and figure: 9.3.

```
...
        switch(sub.getAttachment().getForm()){
        case Timer:
          frameWidth = frameWidth/2;
           v = genDrawView(sub,frameWidth);
          frame.addView(v, frameWidth, frameHeight);
          v2 = genDrawView(sub.getAttachment().genSub(),frameWidth
              );
          frame.addView(v2, frameWidth, frameHeight);
          break;
        case SingleImg:
          frameWidth = frameWidth/2;
          v = genDrawView(sub,frameWidth);
          frame.addView(v, frameWidth, frameHeight);
          i = new ImageView(this);
          i.setImageResource(sub.getAttachment().getImg().getPath
              ());
          i.setBackgroundDrawable(gd);
          frame.addView(i, frameWidth, frameHeight);
          break;
        case SplitImg:
          frameWidth = frameWidth/2;
          v = genDrawView(sub, frameWidth);
          frame.addView(v, frameWidth, frameHeight);
          frameWidth = frameWidth/2;
          frameWidth = frameWidth - 15;
          i = new ImageView(this);
          i.setImageResource(sub.getAttachment().getLeftImg().
              getPath());
          i.setBackgroundDrawable(gd);
          frame.addView(i, frameWidth, frameHeight);
          i2 = new ImageView(this);
          i2.setImageResource(sub.getAttachment().getRightImg().
              getPath());
          i2.setBackgroundDrawable(gd);
          frame.addView(i2, frameWidth, frameHeight);
          i3 = new ImageView(this);
          i3.setBackgroundDrawable(gd);
          frame.addView(i3, 30, frameHeight);
          break;
        }
        setContentView(frame);
        ...
```

Figure 9.2: Code snippet of how DrawLib checks for object form to generate the corresponding view.

```
LinearLayout frame = new LinearLayout(this);
Guardian guard = Guardian.getInstance();
frame.addView(guard.getSubProfile().draw());
setContentView(frame);
```

Figure 9.3: Example of how WOMBAT would generate timer view, if DrawLib was apart of TimerLib.

Last used

WOMBAT do not save the SubProfiles in the last used list, this is because it is not implemented to work together with OasisLocalDatabase. When the WOMBAT activity is destroyed or the memory of the device is cleared, the last used list will be empty. In the future this function should be reimplemented to work together with the OasisLocalDatabase.

Delete and save

There is a bug which occurs when you create three or more new timers on the same profile. When you try to delete these timers it will delete them randomly and might even force WOMBAT to crash due to it is trying to delete a non existing setting in the OasisLocalDatabase. This is a bug in the application that needs to be future developed on.

New Design Features

Noget med en skuffe?

Tail

Part IV

Appendix

Appendix

9.2 Paper Prototypes

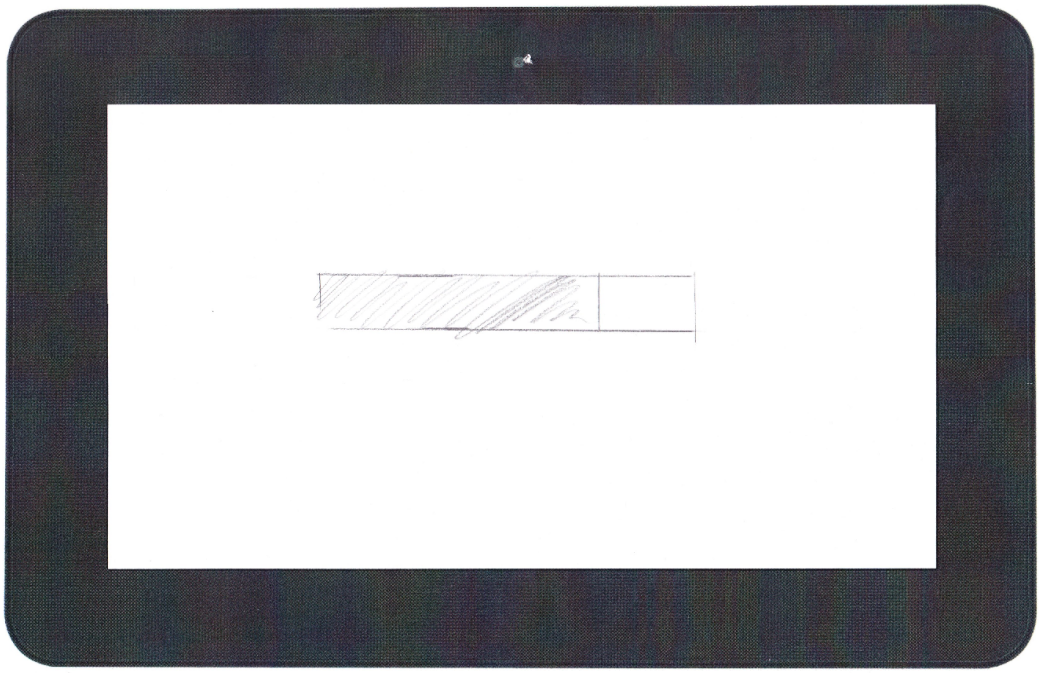


Figure 9.4: Scan of a paper prototype of a single progress bar timer.

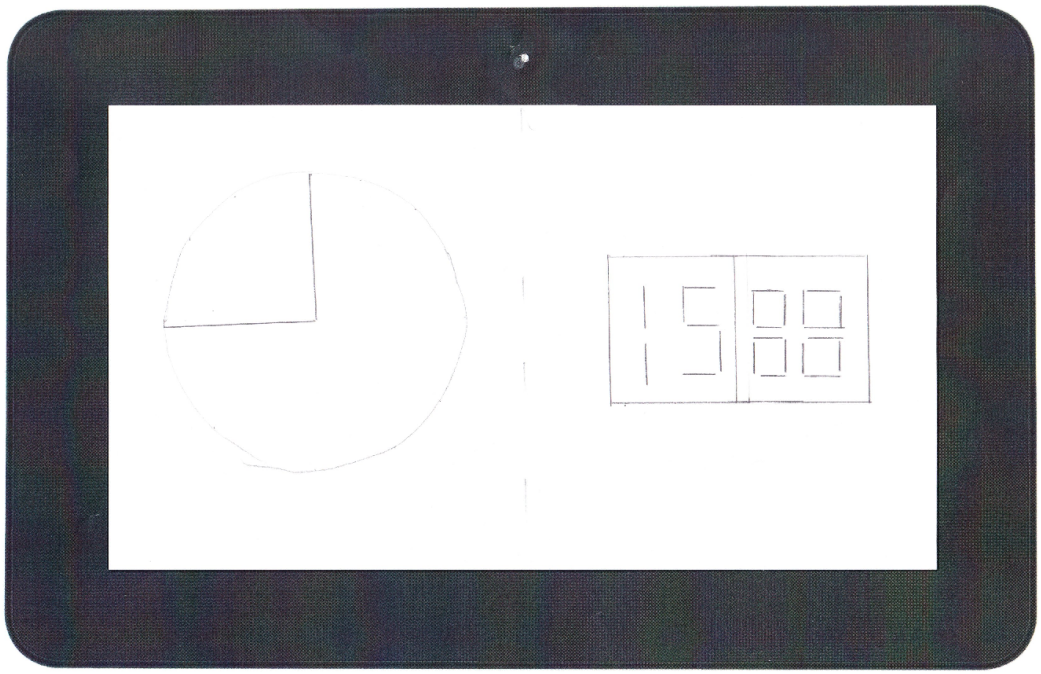


Figure 9.5: Scan of a paper prototype of a double timer.

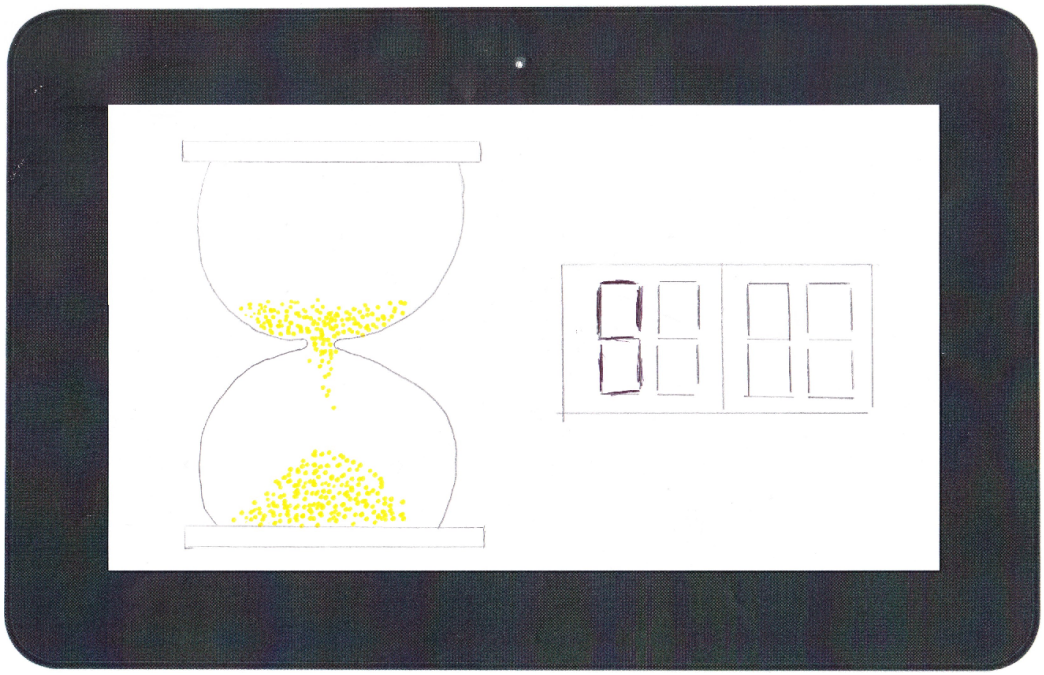


Figure 9.6: Scan of a paper prototype of another double timer.

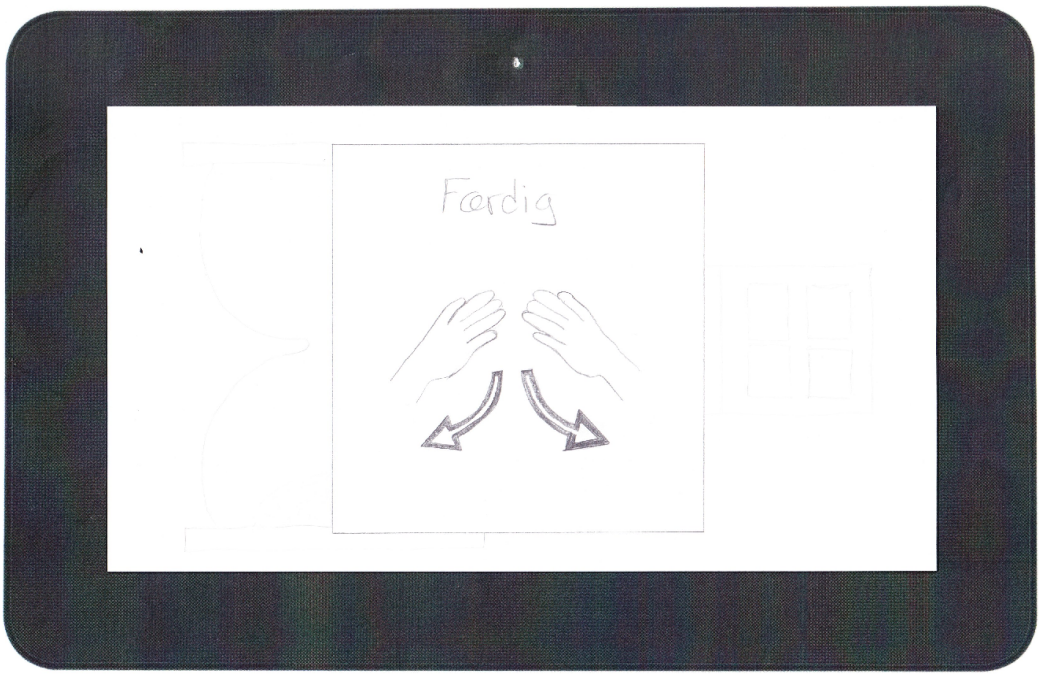
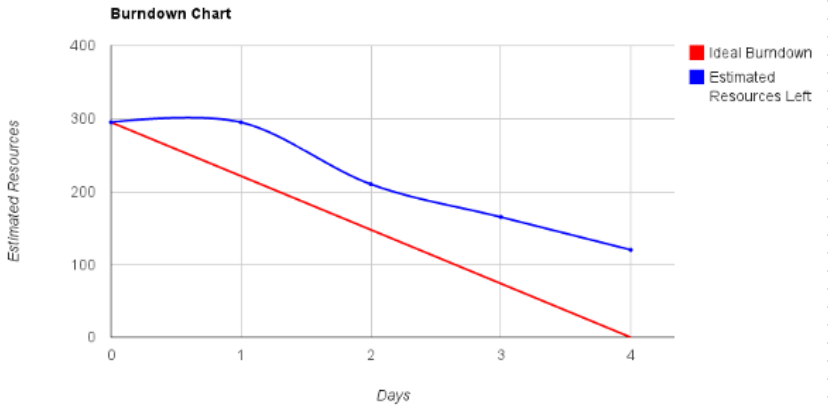


Figure 9.7: Scan of a paper prototype of the "Done" screen shown when the time has run out.

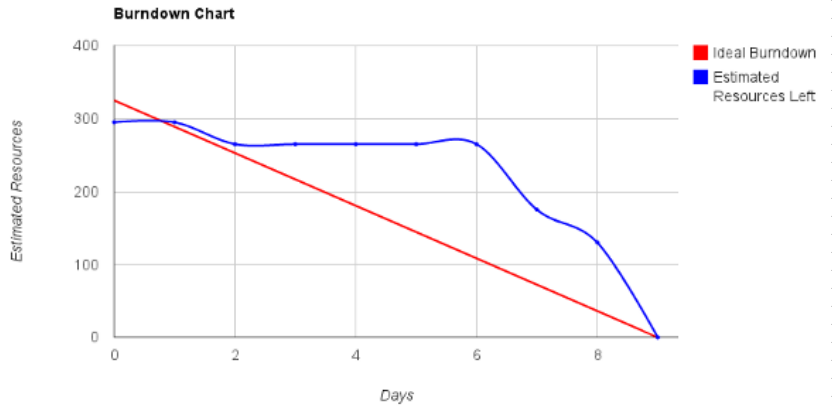
9.3 Sprint Burndown Charts and Backlogs

Timer (WOMBAT)



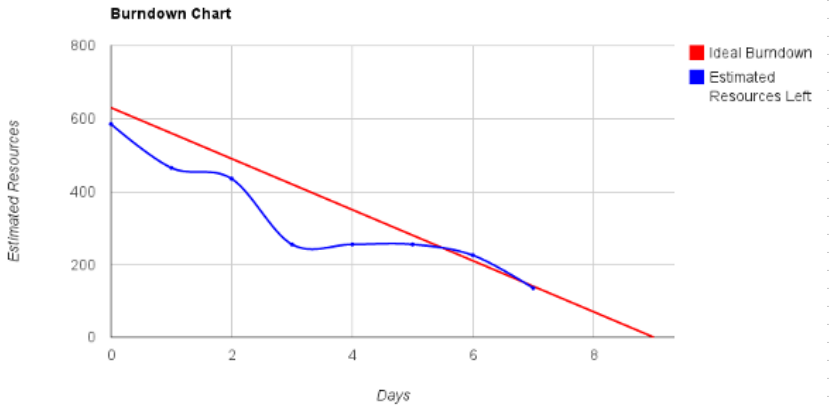
ID	Name	Priorite	Estima	How to demo	Note	Status
1	Design Start activity	14	25	Two lists, 1. = lists of list names, 2. list of profiles		*DONE*
10	Develop Start activity	13	60	According to the design [1], program an "input" for the two lists and make them clickable		*DONE*
5	Develop Load Settings	12	60	Load relevant attributes from admin	*Function at Admin*	*In progress*
14	Retrieve list of list profiles	11	45	Retrieve list of list of profiles from admin group	*Get from admin*	*Admin function not working*
11	Show lists	10	30	Get lists of list profiles from [14], function which returns to start activity [10]	*Missing show pictures*	*In progress*
12	Show profiles	9	30	Get list of profiles from [11], according to click from start activity, returns to start activity [10]	*Missing show pictures*	*In progress*
8	Select user profile	8	30	Open "Tilpas" activity when a profile is selected in [12]		*DONE*
2	Design Tilpas activity	7	15	Window with all possible settings, "Gem" prompts a profile chooser		*DONE*

Timer (WOMBAT)



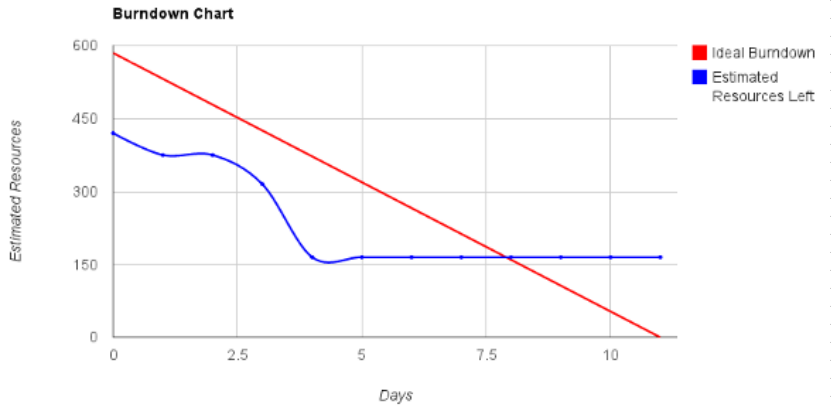
ID	Name	Priorite	Estima	How to demo	*Note*	*Status*
3	Design Vis activity	12	15	Digital Watch + the Timer according to the settings		*DONE*
4	Design "Færdig" activity	13	15	According to the "done" picture from the institution		*DONE*
7	Design Predefined Profiles	14	30	Design the predefined profiles	*Pictures Aquired*	*DONE*
23	Implement load/save	15	45	Implement load/save in the profile and template lists		*DONE*
15	Develop "tilpas" activity	16	90	According to [2], implement save settings		*DONE*
17	Develop "vis" activity	17	30	According to [3], link til OpenGL		*DONE*
25	OpenGL setup	18	100	General openGL settings e.g. camera distance		*DONE*

Timer (WOMBAT)



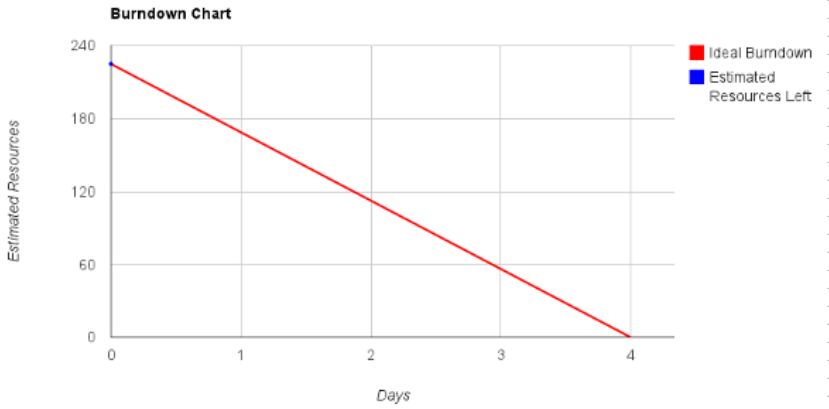
ID	Name	Priorite	Estima	How to demo	Note	Status
5	Develop Load Settings	19	60	Load relevant attributes from admin	*Function at Admin*	*In progress*
6	Develop Save Settings	21	30	Save relevant attributes to admin	*Function at Admin*	*In progress*
9	Last Used Algorithm	22	30	Design how the last used algorithm works		*DONE*
14	Retrieve list of list profiles	20	45	Retrieve list of list of profiles from admin group	*Function at admin*	*DONE*
22	Pictures in Lists	41	45	Implement pictures in the profile and template lists	*Missing how to pic from admin*	
26	Impelment Last Used	24	45	Implement last used algorithm in the profile list [11]		*DONE*
27	Attachment Functionality	40	90	Complete the attachment functionality according to the design	*SubProfile, digiPecs, missing pecs from admin*	*DONE*
28	Save/Save As error handling	26	60	Dont save if something is missing	*Hide "gem" and "gem som" and set lock on "ny subprofile" and "last used" and "predefined"*	*DONE*
30	Implement Predef profiles	29	30			*DONE*
31	1 min bug	28	15	Bug when opening and saving timer with set to 1 min		*DONE*
32	Highlight after save	27	60	Highlight the chosen subprofile after saving		*DONE*
33	Select subprofile functionality	25	30			*DONE*
34	Design icons for Last Used / Predef	30	30		*Not inplemented yet*	*DONE*
35	Long press delete	31	30		*Missing delete function*	*DONE*
36	"New" button	32	30			*DONE*

Timer (WOMBAT)



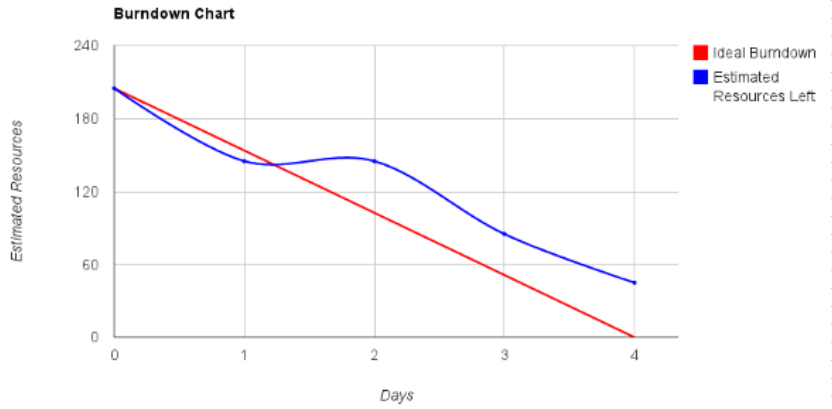
ID	Name	Priorite	Estima	How to demo	Note	Status
21	Digital Watch	44	30	According to [3]		*DONE*
38	Tekst på delete subprofile	46	15			*DONE*
39	Undim Screen	47	45			*DONE*
40	Gemte profil skal highlightes	48	30			*DONE*
18	Hour Glass		45	According to [3]		*DONE*
41	Vedhæftet skal ændre/update tid ved tryk på start/gem	49	45		*Bliver gemt når man opdaterer tiden på timeren*	*DONE*
5	Develop Load Settings	19	60	Load relevant attributes from admin	*Function at Admin*	*DONE*
6	Develop Save Settings	21	30	Save relevant attributes to admin	*Function at Admin*	*DONE*
16	Develop "færdig" activity	51	60	According to the "færdig" sign used in the institution		*DONE*
43	Develop "Vælg færdig billede"	52	60	Show a custom picture when the time has run out. Default is "Færdig billede"	*Mangler at kunne sætte færdig billede på subprofil*	
22	Pictures in Lists	43	45	Implement pictures in the profile and template lists	*Missing how to pic from admin*	
37	Vedhæftet ikon skal ændrer baggrund	45	60			*DONE*
42	Lås/Authenticate	50	60			

Timer (WOMBAT)



ID	Name	Priorite	Estima	How to demo	*Note*	*Status*
22	Pictures in Lists	43	45	Implement pictures in the profile and template lists	*Missing how to pic from admin*	*In progress*
43	Develop "vælg færdig-billede"	52	60	Show a custom picture when the time has run out. Default is "færdig-billedet" from Mette		*In progress*
44	Develop vedhæft pictogrammer	53	60		*Missing a part from admin*	*In progress*
45	Re-factoring	54	60	CustomizeFragment etc.		*DONE*

Timer (WOMBAT)



ID	Name	Priorite	Estima	How to demo	*Note*	*Status*
22	Pictures in Lists	43	45	Implement pictures in the profile and template lists	*Missing how to pic from admin*	*In progress*
42	Lås/Authenticate	50	60			*DONE*
43	Develop "vælg færdig-billede"	52	60	Show a custom picture when the time has run out. Default is "færdig-billedet" from Mette		*DONE*
44	Develop vedhæft pictogrammer	53	20		*Missing a part from admin*	*DONE*
45	Update færdig aktivitet	54	20	Update according to [43]		*DONE*
45	Update færdig aktivitet	54	20	Update according to [43]		*DONE*

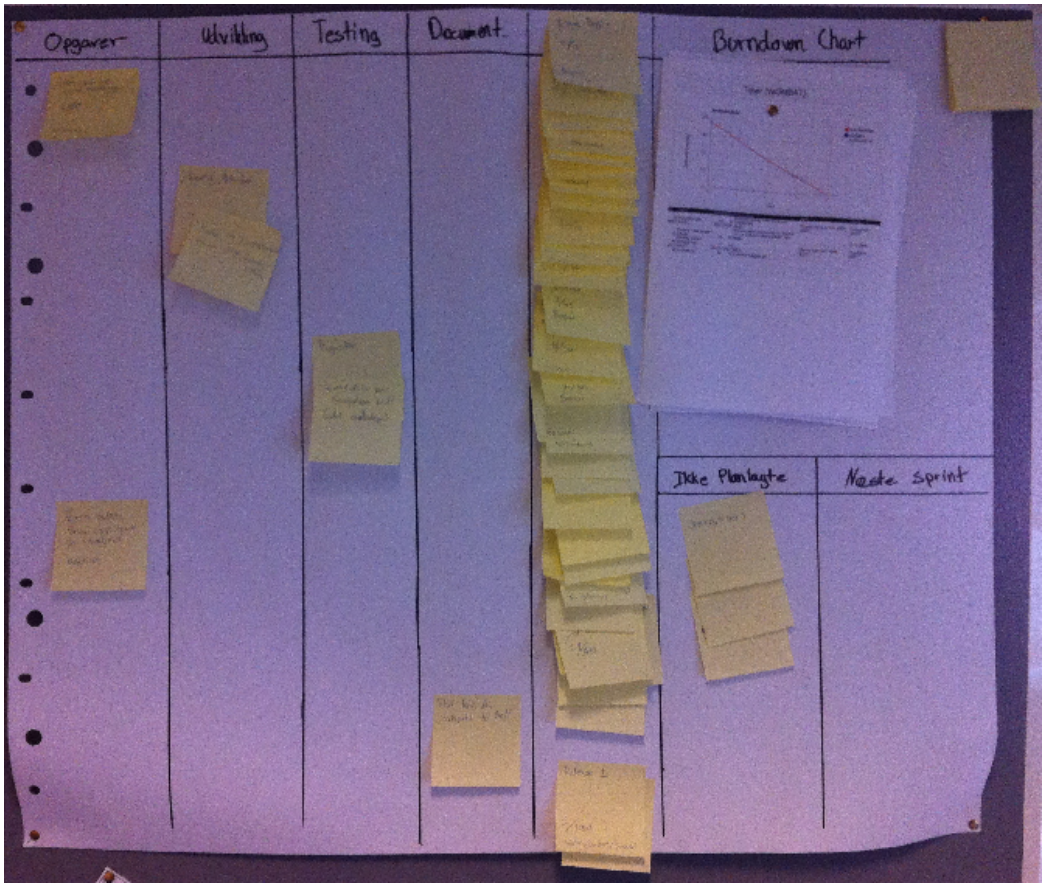


Figure 9.8: Picture of our burndownchart and list of assignments. Used to manage the total project backlog.

9.4 Acceptance Test Diary

Dato: 16/3 - 12

Mads

Virkede godt	Virkede knapt så godt/slet ikke
Var klart genkendelig for barnet (Saundur)	Slutbilledet var for hurtigt væk. Måske kunne det blive indtil jeg bestemmer andet.

Dato: 18/5 12

Virkede godt	Virkede knapt så godt/slet ikke
Drengene forstod straks meningen var. De gik til skema da tiden var udløbet. Saadur Lukas · Emil Lukas digetal Var mere optaget af tallene end aktiviteten – ellers virkede det efter hensigten.	"SAMSUNG" (Visuel stej)

Figure 9.9: Scan of a paper prototype of the "Done" screen shown when the time has run out.

9.5 WOMBAT Setup for Eclipse

To fetch the released code from SVN, make a checkout to `https://sw6-2012.googlecode.com/svn/tags/wombat/`, previous versions of the project, reports, and the wiki can be found by making a checkout to `https://sw6-2012.googlecode.com/svn/`. The folders on SVN contains the following:

- *trunk* - contains iteration releases of the code.
- *branches* - contains code under development.
- *tags* - contains the newest final release of the code.
- *common_report* - contains the common part of the report.
- *Reports* - contains all group reports.
- *wiki* - contains summaries from all group- and supervisor meetings, and guides/agreements.

When SVN checkout has finished downloading content from the SVN, import Wombat, AmbilWarna (the color picker), and wheel projects from svn to Eclipse, figure 9.10.

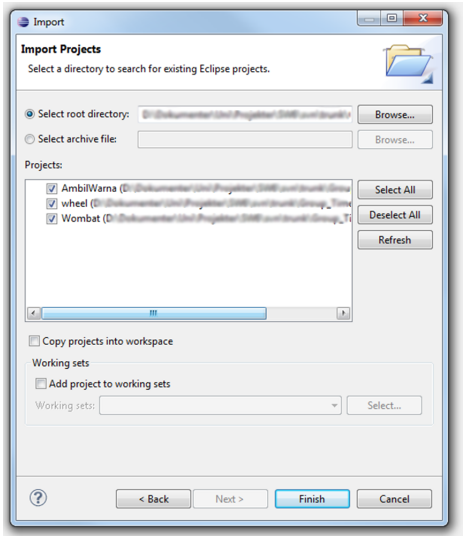


Figure 9.10: Import Wombat, AmbilWarna, and wheel.

Right click the Wombat project in Eclipse and click "Properties".
Under the "Android" pane, ensure that AmbilWarna and wheel are added as project libraries, figure 9.11.

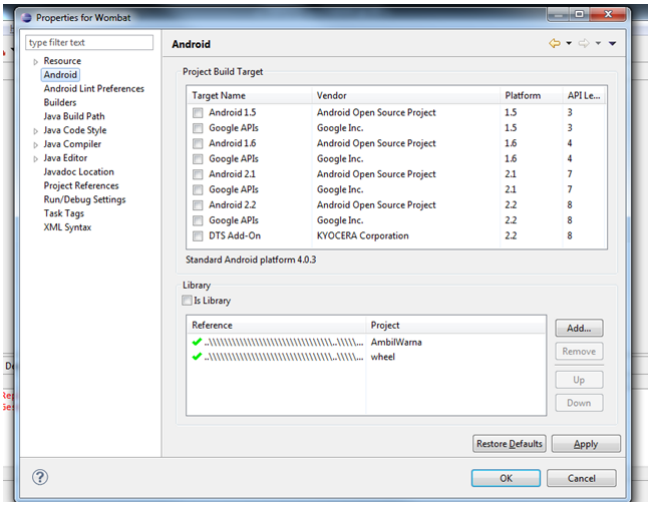


Figure 9.11: Add AmbilWarna and wheel as project libraries.

In case Eclipse fails to recognize the libraries in the libs folder, manually add the three JAR files in Wombat/libs on svn as external JARs, in Wombat - Properties, figure 9.12.

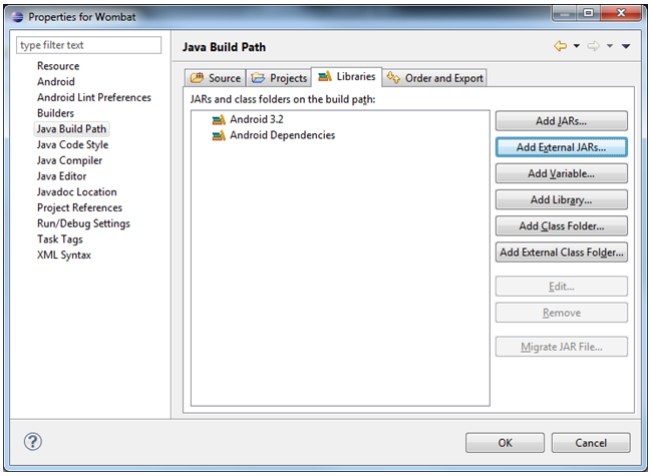


Figure 9.12: Add Oasis, TimerLib, and DrawLib as external JARs if Eclipse cant recognize them in the libs folder.

Extras

To edit the TimerLib and DrawLib the two projects has to be imported to the workspace in Eclipse, figure 9.13.

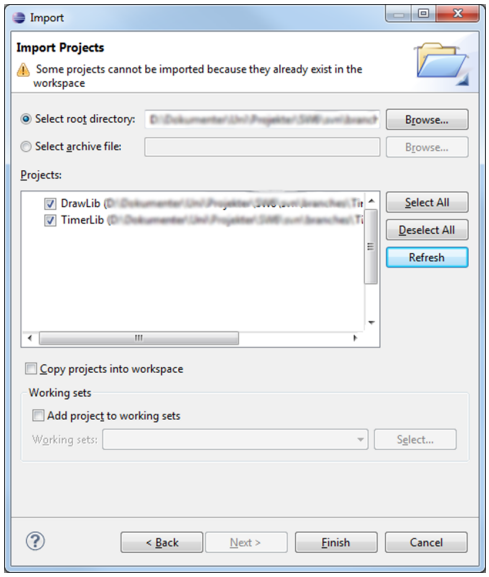


Figure 9.13: Import TimerLib and DrawLib to the workspace.

To debug the TimerLib and DrawLib in WOMBAT the projects has to be added as libraries in the Wombat properties, with AmbilWarna and wheel, figure 9.14. Then the TimerLib and DrawLib JARs has to be deleted from the libs folder or the Java Build Path library, to avoid a compile error.

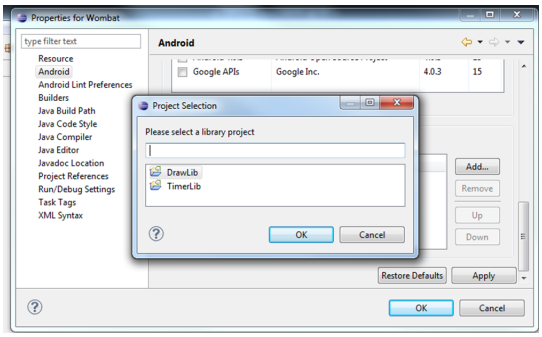


Figure 9.14: Add TimerLib and DrawLib as project libraries.

Bibliography

[1] David Benyon. Designing ineractive systems. Book, ? 2010. Pages: 88, 187, 203-207.

[2] Tim Bray. Fragments for all. http://android-developers.blogspot.com/2011/03/fragments-for-all.html, 2011. Last visit: 15-05-2012.

[3] Shane Conder and Lauren Darcey. Android compatibility: Working with fragments. http://mobile.tutsplus.com/tutorials/android/android-compatibility-working-with-fragments/, 2011. Last visit: 15-05-2012.

[4] Google. Fragments | android developers. http://developer.android.com/guide/topics/fundamentals/fragments.html, 2012. Last visit: 15-05-2012.

[5] DynaVox Mayer-Johnson. Boardmaker software. http://www.mayer-johnson.com/boardmaker-software/, 2011. Last visit: 03-05-2012.

[6] Yuku Sugianto. Android color picker. http://code.google.com/p/android-color-picker/, 2011. Last visit: 23-05-2012.

[7] yuri.kanivets. The wheel widget for android. http://code.google.com/p/android-wheel/, 2011. Last visit: 23-05-2012.