Assignment TDT4258 Assignment 3 - Group 1

Sondre Lefsaker André Philipp Håvard Wormdal Høiby

April 24, 2013

Abstract

Contents

1 Introduction

2 Setup

2.1 Board

To prepare the board for installing the drivers and the game the following must be done:

1. The jumpers and GPIO connectors must be placed properly.
2. The bootloader must be installed onto the board.
3. Linux must be installed onto a SD-card.
4. Linux must be booted with the help of uBoot and Minicom.

This section is a walkthrough of this process.

2.2 Game

To play the game it has to be compiled from the source, copied onto the board and started. This can be done from any of the lab pc's in the lab which contains the **avr32-linux-gcc** compiler.

2.2.1 Compilation

From the /game directory on the lab pc the following command will compile the game:

```
make
```

This produces the file /game/bin/game which is a executable that can be run under the linux distribution installed on the SD-card.

2.2.2 Copying the executable to the board

From the /game directory on the lab pc the following caommand copies the game executable to the board.

```
scp bin/game avr32@<IP address>:~/
```

Where *<IP address>* is the IP obtained in section ¡...¿.

2.2.3 Running the game

From the /home/avr32 directory on the board the game can now be started with:

```
./game
```

In order to play the game the user has to be root, this is obtained by running the **su** command and using password = roota.

3 Overview

3.1 Led Driver

3.2 Button Driver

3.3 Graphics

3.3.1 Main graphics

The graphics of the game is contained within the /graphics folder. Here the **Screen** object is responsible for communication with the framebuffer through the **/dev/fb0** file. The **Canvas** object holds an array of **Shape** objects. Its has a method called **CanvasPaint** where it renders the screen in an 320x240 **Pixel** array contained within the **Screen**. When the whole array has been updated the **Canvas** object ask the **Screen** object to copy the contents of its internal buffer to the framebuffer. This causes the LED screen on the device to be updated.

All graphical objects are inherits their main structure of the **Shape** object. The most importaint part is the **paint** method which **Canvas** uses to render all the objects on the internal buffer.

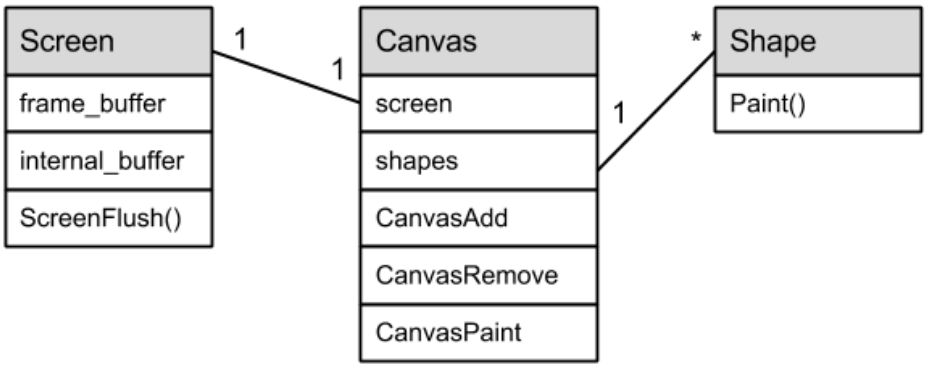


Figure 1: UML for the main part of graphics

3.3.2 Images

For all images the 24 bit version of the **BMP** (Bitmap file format) was implemented. This is done by the object **Bitmap** is hidden behind the **Image** object so that it is easy to add more file formats. The reason that bitmap was chosen is that the 24 bit version of it corresponds almost directly to the layout of the framebuffer, so the implementation was straight forward.

3.3.3 Graphical objects

Both objects for drawing lines and rectangles are where used only in the testing of the graphics module and are not used in the game. They serve as examples on how to develop more graphical objects.

3.4 Sound

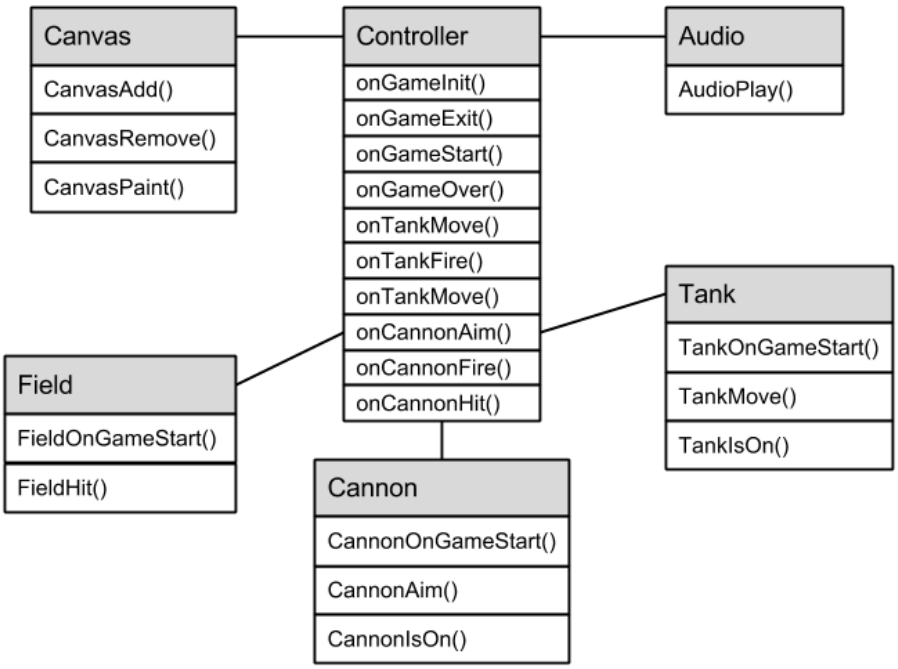
In the folder /sounds the object **Audio** takes care of playing sounds. It uses ordinary file operations to write 8 bit samples from a **wave** file to the file **/dev/dsp**.

3.5 Game

3.5.1 How to play

3.5.2 Design

The /game folder contains the objects used in the game. The most importaint are **Controller** which handles the communication between the modules in the game. The **Cannon**, **Tank** and **Field** are models for the objects in the game. The **Button** and **Led** objects are wrapper object around the char drivers, to make communication easier.



4 Solution

4.1 Led Driver

4.2 Button Driver

4.3 Graphics

4.4 Sound

4.5 Game

5 Test Report

6 Discussion

7 Conclusion