Mireia Llauradó Costa,

NFC extension for Catrobat

Master's Thesis

to achieve the university degree of

Master of Industrial Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl-Ing. Dr.techn. Slany, Wolfgang

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Slany, Wolfgang

Graz, May 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Abstract

"How does our daily routine look like nowadays? We wake up to an alarm set on Amazon Alexa, get a ride to work with Uber, message our team through Slack, buy a coffee and pay for it using Apple Pay, organize a to-do list on Trello, order food through Foodler, chat with employees via Google Hangouts, book our vacation through Kayak and Airbnb and load up Netflix and watch TV on demand before heading to bed."[1]

All those tasks didn't exist or would have been performed in a very different way ten years ago, and that's the best way to realise the world of software technology is growing really fast and so the need of people with programming skills and software developers. The aim of Catrobat project is to bring this programming first contact to kids and to make learning enjoyable for them.

The goals of this thesis are to widen the big amount of extensions that Catrobat already has, by adding the capabilities of Near Field Communication (NFC) so Catrobat users can learn something about NFC and its usage, and to ensure the automatise testing of the hardware features that cannot be tested on an emulated device. Within the implemented features, users should be able to read the content and the ID of NFC tags, as well as using them to handle conditions in the program logic and developers should be able to test these hardware features automatically.

[1] **https://www.forbes.com**.

Contents

List of Figures

1 Introduction

The number of smartphones used worldwide is increasing year by year[1]. Smartphones offer advantages as better mobility and easy access than laptops and the number or users is by far much higher than computer users, especially in developing countries. *Catrobat* project offers the opportunity to show teenagers how to take advantage of this situation by using smartphones not only for posting or gaming but also for learning in an enjoyable way.

Moreover, the need of people with developing skills is also increasing, and Catrobat offers a good oportunity for starters, as the user doesn't need to learn any language commands to start programming. On the contrary, it works with blocks, and the user just have to drag and drop these blocks to create their own application. Easy as it sounds, Catrobat can have a thousand of possibilities by using its extensions such as code blocks for the Lego robot NXT, Arduino or using many features that are found in the same smartphone.

1.1 Motivation

Catrobat philosophy is to make young people approach to programming knowledge and get some new skills by enjoying and having fun while learning. Motivation and enjoyment in the learning process is crucial to make it as much effective as possible, and this utopia became true through Catrobat project.

[1] **https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/**.

Among the many possibilities that Pocket Code offers to children, it is interesting to widen the programming skills by using extensions, the aim of this thesis is to make able to children to get to know and experiment with the Near Field Communication (NFC) technology. NFC is already integrated into the daily life, some examples would be mobile payments, wirelessly transferring data or quickly downloading information by approaching a phone instead of scanning a QR code.

As priorly introduced, testing is indispensable to ensure the quality of a software. Test-Driven Development (TDD) guidelines are a must when contributing to Pocket Code development. For this reason another big concern among the developers is how to be able to automatically test some hardware features that cannot be tested in an emulated device, so part of the time dedicated on this thesis will be focused on ensuring that these tests are currently available and operative for developers.

1.2 Problem statement

It can be considered that this thesis is divided in two main goals.

On one hand, one of them is that Pocket Code will be able to offer the possibility to scan NFC tags and subsequently use them to trigger any desired action as well as reading and writing into them, so users can get familiar with NFC technology.

On the other hand, the other goal of the thesis is to ensure that developers of Pocket Code are able to test some hardware features that are not able to be tested on an emulated device. For this aim it has been necessary to work with the sensor box designed by Lesser to automatize testing by using external hardware sensors.

1.3 Thesis outline

The aim of this chapter is to explain the structure of this thesis and make a short description of every chapter. All the background information and the

introduction to the topic can be found in Background. Its structure consists firstly to introduce the project itself called Catrobat, the Test-Driven Development method that is used by its developers followed by explaining which method is used for automatically testing hardware features that cannot be tested on an emulated device, therefore the Sensor Box is introduced . To finish this chapter Near Field Communication main features are provided.

Afterwards, the implementation process of the thesis is explained in Implementation. Information is mostly about the features concerned to NFC because there are very clear rules that need to be followed. When talking about the test related to the Sensor Box, accessing related information is given, as the tests were prior written and during this thesis just some mistakes have been corrected. However, the working methods and the implementation is exactly the same as the one implemented by Lesser in 2018, and his thesis can be checked for further details.

Finally and in Conclusion and Outlook, the lessons learned along this thesis and some ideas for possible future work are presented.

2 Background

Techniques and hardware information used in this master thesis are provided in this chapter. It describes the free open source software (FOSS) project Catrobat, the Test-Driven Development method and therefore, the hardware automatic testing method created by Lesser. Finally, technical details about the Near Field Communication are given.

2.1 Catrobat

Catrobat is a Free Open Source Software (FOSS) non-profit project that allows users to create and publish their own apps using only their smartphones. As explained in the abstract, the needed of computer science skills is everytime more demanded and Catrobat project aim is to introduce young people to the world of coding (Slany, 2014).[1]

Throughout the integrated development environment (IDE) Pocket Code the user can directly create programs on Android and iOS mobile devices, both phones or tablets.

The app is very intuitive and easy to work with but it has a lot of different possibilities and extensions to be able to develop more complex apps, using the same device sensors or using external hardware, such as Arduino, Raspberry Pi, Lego NXT or remote control flying drones, so it allows teenagers to make the most of their imagination.

Catrobat is strongly influenced by MIT's Scratch project[2] as it is a block based visual programming language where each block represents a separate

[1] **https://arxiv.org/ftp/arxiv/papers/1808/1808.06292.pdf**.
[2] **https://scratch.mit.edu/**.

functionality. These blogs are classified depending on their functionality and are easily distinguishable by colors as can be seen in Figure 2.1.

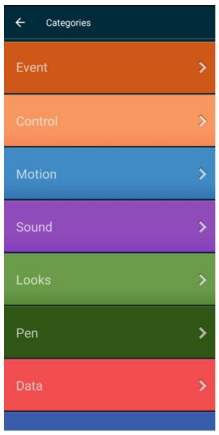

Figure 2.1: Example of categories in the 'add brick' menu distinguised by colors.

For developing of Catrobat project Test-Driven Development by Beck and Clean Code by Martin are strictly used, and every task, bug fix or other issues are stored as a ticket in Jira[3] and visualised in a Kanban board[4], ensuring a clear and up-to-date documentation.

2.2 Test-Driven Development

When talking about developing first thing anyone would do is go straight to coding, and that's a huge mistake. Test-Driven Development (TDD) is a software development process that requires writing very specific test cases before the code is improved, so the sequence would be the next one:

- Add a test: making a developer write a test before starting coding makes him focus on the requirements before writing the code.
- Write enough code to fail the test.
- Run all the tests and watch that the new test fails to make sure the test is going to be executed.

[3] **https://www.atlassian.com/software/jira**.
[4] **https://www.atlassian.com/agile/kanban**.

- Write the code to make the test pass.
- Run the tests and see that they all pass, if not, the new code must be adjusted until they do.
- Refractor code: the growing code base must be cleaned up regularly using Clean Code rules and of course re-running the test cases after every refractoring phase.
- Repeat the steps above until you can't find anymore tests that drive writing new code.

A visual scheme of the TDD cyclic method is represented in Figure 2.2.

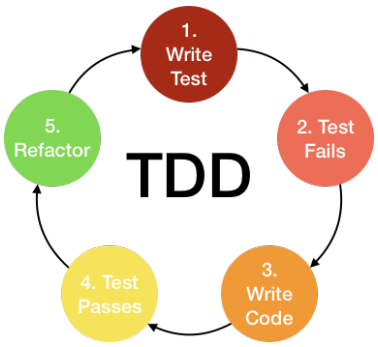

Figure 2.2: TDD cyclic process.

Although working with the Test Driven Development method can be a little time consuming at first, it has a lot of benefits, like allowing programmers to focus on the task as the first goal is to make the test pass or ensuring that all written code is covered by at least one test. This method gives the programming team a greater level of confidence in the code, reduces time spent on rework and in the debugger and forces the radical simplification of the code, in order to create a solid code.

2.3 Hardware automatic testing method

Emulated device testing is undoubtedly necessary for testing Android software through continuous integration method as the Jenkins servers have to be able to make multiple test code executions simultaneous when several

pull requests are made. When talking about hardware testing though, a real device comes to the need. One of the multiple possibilities that offer Pocket Code is to be able to use almost all the sensors and features of a smartphone, but of course they need to be tested. The solution for testing those features and sensors was made by Lesser ("NFC extension for Catrobat", May 2018) and it consists in building a SensorBox (see Figure 2.3) which currently can test five hardware features of a smartphone: vibration, audio signal via the auxiliary output, LED light, NFC and network access. All the details about the fabrication process can be found in Lesser Thesis and the details of every test can be found in *RasperIno: Testing Hardware Features of PocketCode using Arduino and some Sensors*.

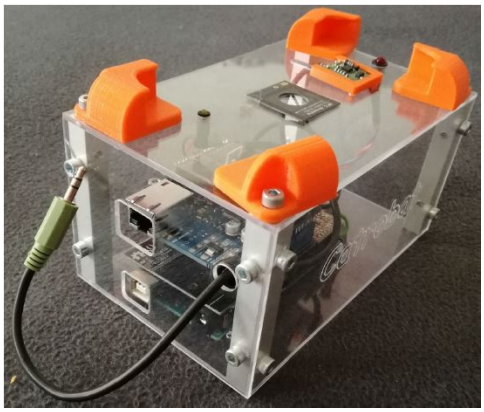

Figure 2.3: Sensor Box created by Joachim Lesser.

2.4 Near Field Communication

Near Field Communication is a bidirectional and short-range wireless communication technology that requires approaching two NFC-compatible devices together over a few centimeters and it's been designed for secure data transfer. However, it can be used to accomplish many other targets.

NFC technology uses the following smart devices:

- NFC-enabled mobile phone: the integration of NFC technology with mobile phones creates a big opportunity for the ease of use, acceptance, and spread of the NFC ecosystem.
- NFC reader: it is capable of data transfer with another NFC component.
- NFC tag: it is actually an RFID tag that has no integrated power source, each NFC tag has its unique ID (UID) which cannot be changed or deleted.

Depending on if the devices have a power source or not, we can call them active or passive. An initiator of the communication always needs to be an active device because it requires a power source to initiate the communication. On the contrary, the target can be both active or passive. In case the target is active, it is going to use its own power source to respond, while if the communication is being towards a passive device, it uses the energy created by the electromagnetic field, which is generated by the initiator.

Due to these three types of devices and depending on their interactions, there are three operating modes: reader/writer, peer-to-peer, and card emulation.

If the initiator of the communication is a mobile and the target is a tag, the reader/writer mode is used. Nevertheless, if the communication is between two mobiles, the operating mode is peer-to-peer, and finally, the card emulation mode is used when the initiator is a reader with a mobile as a target.

The main operating mode for this project is the reader/writer operating mode, as Pocket Code aim is that users can use NFC tag both to read and write their content or use the ID to be able to trigger actions, so as previously described, the user in our case is using their mobile phone devices to initiate the communication and a passive NFC tag as a target, and that's the reason why delve into this mode is necessary.

2.4.1 Reader/Writer Operating Mode

As described before, the reader/writer mode (see Figure 2.4) is about the communication of an NFC-enabled mobile phone with an NFC tag for the purpose of either reading data from or writing data to those tags. As the name indicates, there are two internal modes to make the communication, the reader and the writer.

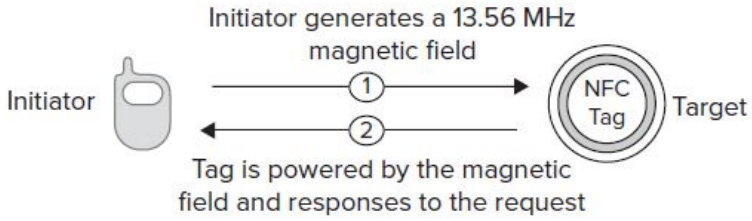

Figure 2.4: Reader/Writer operation mode.

In both modes the mobile acts as the initiator. Concerning on the reader mode, the mobile device reads data from a NFC tag, which consists of the requested data and the program that returns the requested data to the initiator. However, in the writer mode the mobile phone writes data to the tag and replies with the acknowledgment data, informing the user about the success of the operation. If the tag already contains any data prior to the writing process, it is overwritten.

The key point of this type of communication is that a mobile phone can perform several actions after it reads the data from the tag, and this provides many opportunities for both users and service providers as the applications are countless and can be very innovative.

3 Implementation

3.1 Testing Sensor Box

As mentioned in Hardware automatic testing method, the idea of the hardware testing box was created by Lesser and further details can be found in his thesis: "NFC extension for Catrobat, May 2018".

The requirements to run hardware tests can be found in Confluence[1] but the main are the following:

- Nexus must be on and connected to slave1
- Nexus must NOT be locked
- Nexus must be connected to hidden WLAN access point "roboarduino".

These tests can be run by `UiEspressoSensorboxTestSuite.java` which runs a total of seven tests:

- Light test named `FlashBrickStageTest.java` runs both `testActualFlashOnBrick` and `testActualFlashOffBrick`.
- Audio test named `PlaySoundBrickStageTest.java` runs both `testNoSoundPlayed` and `testSoundPlayedFromSoundBrick`.
- Vibration test named `VibrationBrickStageTest.java` runs both `testVibrationHardwareOn` and `testVibrationHardwareOff`.
- NFC test named `testWhenNfcHardware` testes WhenNfcBrick and the ability to read both the tag ID and the message.

[1] **https://confluence.catrob.at/display/JENKINS/SensorBox+Troubleshooting+101**.

By the end of this thesis, all of them except from the NFC test are proved available for testing those hardware features in two possible ways: on one hand it is possible to run these tests into the same local area network using the smartphone, the testing box and the computer. On the other hand, it is possible to execute them on the Jenkins server, which is permanently connected to one box.

3.2 NFC Programming: Reader/Writer Mode

The focus while working with an NFC extension in Pocket Code is to make a communication in between the mobile device and an NFC tag, and that's the reason why it is crucial to get in detail about the reader/writer operation mode.

3.2.1 NFC properties in the Android Manifest file

First of all, in order to enable NFC technology in an application, the required permissions should be given to the app to use the NFC hardware and to handle the intents. This can be done by declaring the following line in the Android manifest: `<uses-permission android:name=\android.permission.NFC" />`.

In addition, the minimum SDK version should be considered. Experts suggest that minimum API level 10 should be use and advise that in API 14 extra methods and Android Beam are introduced.

3.2.2 NFC packages in Android

Currently there are two packages for NFC application development in the Android platform. The first is the main package, `android.nfc`, which includes necessary classes to enable applications to read NDEF messages from tags and write to them. The second package is the `android.nfc.tech` package, which includes necessary classes to provide access to different tag technologies. For the developing of Pocket Code extension, the first package

is used, so it is important to know the six classes that it offers and that are explained in the following table.

Class Name	Description
Tag	Represents the discovered NFC tag
NfcAdapter	Represents the NFC adapter on the mobile phone
NfcManager	Obtains an Instance of the NFC adapter
NdefMessage	Represents an NDEF message
NdefRecord	Represents an NDEF record
NfcEvent	Wraps information associated with an NFC event

Table 3.1: Classes that can be found in android.nfc

3.2.3 Foreground dispatch system

There are actually two types of dispatch systems, which are the tag intent dispatch system and the foreground dispatch system. On the first one, the application registered to handle the tag is launched when scanning an NFC tag, and if more than one applications are registered, a pop-up to select the application is displayed. If the goal is to handle tags when the application is running the foreground dispatch system needs to be used and there is where our goal sticks to.

The foreground dispatch system gives priority to handle the tag to the application that is running and in order to implement it the next requirements must be met:

First thing that needs to be created is a `PendingIntent` so that Android can get details of the tag.

Then declaring the intent filters is needed to handle the tags. If the registered filters in the foreground dispatch system match with the tag, then the active application handles the intent. There are three intents: `ACTION_NDEF_DISCOVERED`, `ACTION_TECH_DISCOVERED` and `ACTION_TAG_DISCOVERED` and all three are declared in the Catrobat code.

Finally, it is needed to override the activity lifecycle callbacks and add logic to enable and disable the foreground dispatch on `onPause()` and

`onResume()`. The required code to process the data from the scanned NFC tag must be implemented in the `onNewIntent()` method.

3.2.4 Tag reading and writing

For both reading/writing actions the first steps are the same, they both start when the tag is discovered using one of the previous mentioned intents it is necessary to retrieve the extended data from the intent using the `getParcelableExtra` method.

When the aim is to read the message, the next step that should be done is to get the NDEF message from the tag, and it can be done through the `Ndefmessage` class from `android.nfc` package mentioned in NFC packages in Android. After the message is got and before being able to do any kind of action with the data it needs to be processed and the way of doing it depends on the contained record types.

On the other hand, when the intention is to write into the tag the procedure is a little bit different. After getting the instance of the detected tag, the NDEF message needs to be prepared based on the desired data type on the NFC Forum standards but once these data is ready, the writing operation is the same for all of them.

3.2.5 Implemented features

Through this thesis the user can now do the following activities while using the NFC extension:

- The first implemented feature is that the user is able to scan NFC tags, so they are registered in a list, where also the option to rename them is available, as it can be seen in Figure 3.1. This pre-activity is used to save tags for further uses while programming our application.
- Another feature that is available and showed in Figure 3.2 is to use the former scanned tags or any NFC tag to be able to trigger the desired following activity.

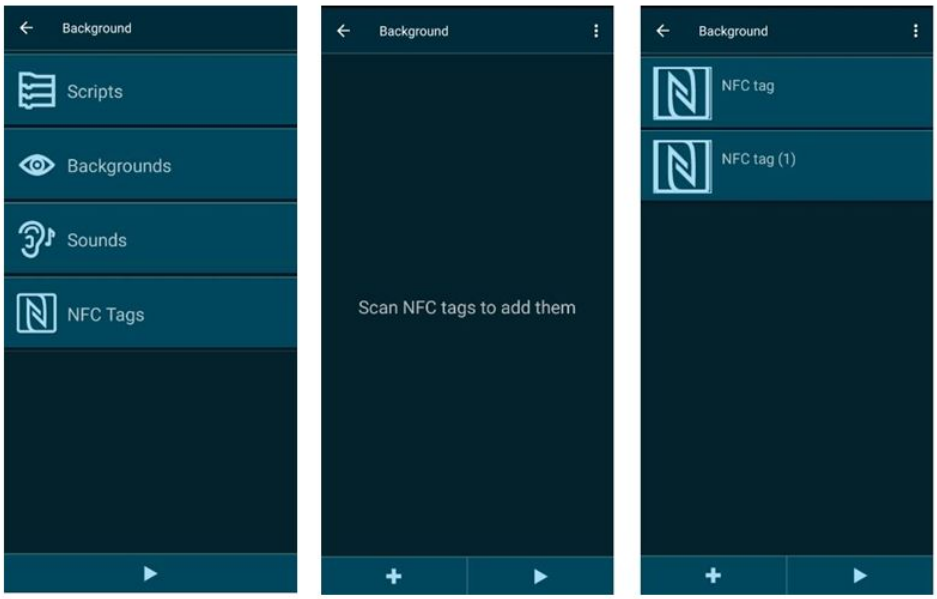

Figure 3.1: NFC tag adding process in Pocket Code.

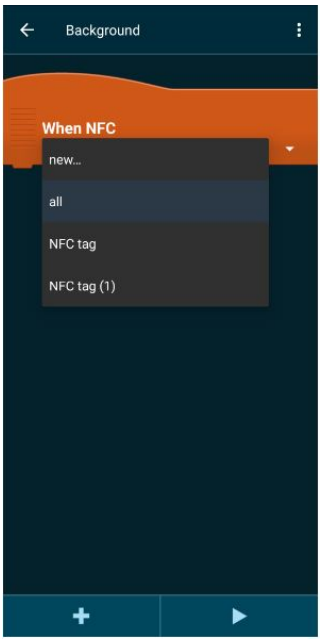

Figure 3.2: When NFC trigger brick.

3.2.6 Files of interest

In this chapter the information of the location of these implementations is given as it might be useful for future related work (see Future Work).

- Android Manifest permission: `AndroidManifest.xml`
- Pending Intent: `NfcTagListFragment.java` and `StageLifeCycleController.java`
- Declared intents: `NfcHandler.java`
- Override activity lifecycle callbacks and logic to enable/disable foreground dispatch: `NfcTagListFragment.java` and `StageLifeCycleController.java`
- Tag reading and writing: `NfcHandler.java`

If further details are needed in the NFC area, the book *Professional NFC application development for android* is a very useful resource and the base of the NFC related work along this thesis.

4 Conclusion and Outlook

In the following sections the learned lessons while working on this thesis and an outlook for possible future work are developed.

4.1 Lessons Learned

Prior to joining Catrobat team I had no experience at all with Java language and the app developing world. As an Industrial Engineer, I had been working with programming but never in this type of environment, and first of all I would like to thank my supervisor to give me this opportunity. The IDE Android Studio, Jenkins server and Github development platform are tools I wasn't familiar with.

The process at the first was pretty hard and with slow improvements but after some time I got used to the language and the methods used in the Catrobat project, achieving goals that at the beginning seemed impossible to me.

4.2 Future Work

This thesis was planned to complement Lesser master thesis to enhance the NFC extension for Pocket Code and the hardware tests run in the SensorBox to make it reliable for the developers of the Catrobat project.

The first goal was to make the hardware tests available so developers could rely on them. While working on those tests it came out that the NFC extension was not properly working. That's the reason why getting deeper

in the NFC requirements was necessary and as formerly mentioned, there are some activities that are currently working again and are described in Implemented features. However, there are still some functions that need to be tested and that may need some further modifications, such as writing messages on a tag. Of course, the NFC test from the Sensor Box did never pass because the function itself was not working properly, so the next step would be to check this test now that the NFC brick is operational and prove that the test is available to be used for the hardware automatic testing.

Bibliography

Association, International Catrobat (2010-2016). *Catrobat*. URL: https://www.catrobat.org/ (cit. on pp. 1, 4).

Baumann, Bernd (2019). *RasperIno: Testing Hardware Features of PocketCode using Arduino and some Sensors*. URL: https://confluence.catrob.at/pages/viewpage.action?pageId=4948088 (cit. on p. 7).

Beck, Kent (2000). *Extreme programming explained: embrace change*. addison-wesley professional (cit. on p. 5).

Coskun, Vedat, Kerem Ok, and Busra Ozdenizci (2013). *Professional NFC application development for android*. John Wiley & Sons (cit. on p. 15).

Lesser, Joachim (2018). *NFC extension for Catrobat*. URL: https://diglib.tugraz.at/download.php?id=5bebd9689533d&location=browse (cit. on pp. 2–4, 7, 10, 16).

Martin, Robert C (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education (cit. on p. 5).