Università degli Studi di Padova

Dipartimento di Ingegneria dell' Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

A Code Completion System for the Catrobat Visual Programming Language

Student

Marta Todeschini

Advisor

Prof. Carlo Fantozzi

February 26, 2018

Academic Year 2017/2018

Abstract

Code completion systems are increasingly popular in the computer programming world. This thesis describes the design of a code completion system within the visual programming language Catrobat, with the aim of making programming more accessible to younger and less experienced users. This system was subsequently implemented into an application developed for Android devices, Pocket Code.

Abstract

I sistemi di completamento di codice sono sempre più diffusi nel mondo dell'informatica. Questa tesi descrive il progetto di un sistema di completamento di codice per il linguaggio visuale di programmazione Catrobat, con lo scopo di rendere la programmazione più accessibile anche a utenti giovani e meno esperti. Questo sistema è stato successivamente implementato in un'applicazione sviluppata per dispositivi Android, Pocket Code.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The science that lives behind the concept of Information Technology has many names, such as "Informatics" , "Computer Science" and "Computing Science" .
Historically the birth of Informatics is associated to an article written by the British scientist Alan Turing in 1936. In this article, he talked about the concept of a *hypothetical computer*, underlying the presence of some important tools that we can still find in real computers.
Informatics concepts are at the root of the digital world. just think of the Google business model based on "Page Rank", to cryptographic algorithms that are used in the field of e-commerce, smartphones, a key everyday tool, Twitter, Facebook and other social networks and many others.
Nowadays there is a common need to include the study of of this topic in schools. IT is considered a multidisciplinary subject, it is essential in many disciplines such as physics, mechanics and even the humanities. It is often considered the key to innovation.
Computer programming is the part of the computer science that refers to the process of developing and implementing various sets of instructions to enable a computer to perform a certain task, solve problems, and provide human interactivity. These instructions (source codes which are written in a programming language) are considered computer programs and help the computer to operate smoothly.
Coding is becoming increasingly a key competence which will have to be acquired by all young students and increasingly by workers in a wide range of industries and professions.
It has therefore become increasingly important to be very familiar with computer science: this subject has been included in the official school program in secondary school or even in primary school. Before the children begin to interface with this discipline, the better they can learn the concepts that are

at the base.

1.1 Computer programming for education

The strength of teaching computer science is its power to stimulate students' creativity. Nowadays this discipline does not find obstacles in its diffusion, given that today every person has a personal computer. Anyone without distinctions based on sex, age and nationality, has the potential to start writing a program and making it available to many other people.

Children are those who have great imagination and creativity: although they may not have experience or knowledge of computer science concepts, they can express their ideas and creativity by developing their own applications with the help of some tools.
Thanks to their young age, it is easier for them to quickly learn new concepts and think about new features or ideas that could be implemented.

For children and people without any knowledge of programming languages, the need arises of devising tools to make the learning process easier and more understandable: *visual programming languages* are useful in this sense.

1.2 Visual Programming Language

In many countries [11, 12] programming was adopted as an official subject in primary primary education. Here *visual programming languages* are used to teach children how to program in a simple and fun way.
A visual programming language (VPL) [15, 16] is a programming language where the user simply manipulates the elements of the program in a graphical way, rather than specifying them through text statements.
Each component of the program is a sort of box or block that can contain some instructions, expressions or definitions of variables. These boxes are treated as entities and are linked by arrows or lines representing relations [13, 16].

This feature of VPLs makes them very suitable for educational planning for children in secondary or even primary schools: this is due to the fact that

VPLs do not require the knowledge of programming syntax and provide an environment without errors in the compile phase [12].
Using VPLs, children are more motivated because they can immediately see the results of the program they have developed [14] thanks to this high motivation, the rate of drop out of learning through these tools is kept low and children continue to study and they have fun at same time.

In a VPL the user simply has to manipulate cells or blocks from a list containing all the blocks of the program; sometimes she/he must complete or write the text formula in the object she/he is manipulating. This formula can include constants, references to other cells, or references to the cell value at an earlier time [17].

There are many examples of VPLs in different computing domains such as Scratch, Squeak-Etoys and LabVIEW (Laboratory Virtual Instrument Engineering Workbench, a system design platform and development environment for a National Instruments visual programming language).

1.2.1 Scratch

Scratch [18, 19] is a visual and block-based programming language designed to be used by children. It was first publicly launched in May 2007. Then Scratch 2 was released on May 9, 2013. With its introduction, custom blocks could be defined within the projects according to user needs . As of 2017, Scratch 2 is available online and as an application for Windows, MacOS, Linux (Adobe Air Required). There is also an unofficial version for Android as an apk file. The Scratch 2.0 offline editor can be downloaded for Windows, Mac and Linux directly from the Scratch website. However, the unofficial mobile version must be downloaded from Scratch forums.
Scratch 3 is under development and an alpha version is scheduled for the first quarter of 2018.

Figure 1.1 shows what the Scratch editor looks like. The editor is divided into three sections:

1. *Left section*: here the user can immediately have a visual feedback of the project she/he has developed up to that moment.

2. *Central section*: here are all the "blocks" available to the user, divided into categories (Motion, Looks, Sound, Pen, Data, Event, Control, Sensing, Operations and More blocks).

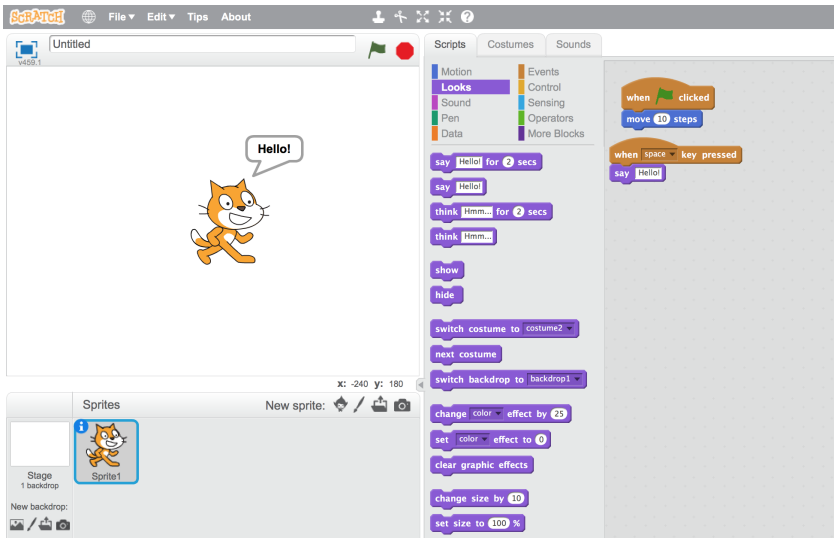


Figure 1.1: Scratch

3. *Right section*: to add a block into her/his program, the user drags the selected block within this part. Here there is the complete and ordered sequence of blocks of the program.

Scratch is designed to be highly interactive. The user just clicks on a stack of blocks and immediately starts executing her/his code. He can even make changes to a stack as it is running, so it is easy to experiment with new ideas incrementally and iteratively.

1.2.2 Squeak-Etoys

Figure 1.2: Logo of Squeak-Etoys

Etoys is a powerful learning tool to teach children powerful ideas about compilation. It provides an authoring environment that is rich in multimedia

content, and a visual programming system. It can also be used on almost all personal computers.
Through this tool, users (who are typically children) can draw their drafts and then write some "scripts" which tell such drawings what to do.

On the website `http://www.squeakland.org/resources/` users can find a lot of information about Etoys, some tutorials and a section called *Discuss* where users can share their problems or ask some questions.

1.3 Mobile applications and other tools

With the rapid development of technology, an increasing number of users use every day a mobile device such as a smartphone or a tablet.
These devices are considered essential today in our daily actions: we use the smartphone not only to make calls, send messages or surf the Web, but for many other activities such as taking notes, scheduling appointments and things to do, make payments, read books and other tasks.
For this reason and thanks to the portability of these devices, many mobile applications have been developed also in the field of education.
In the Play Store for Android or in the App Store for iPhone, we can find many applications of this type. In particular, for the purpose of this thesis, I will show a short list of applications developed for the purpose of teaching children and non-experts how to program.

1.3.1 Pocket Code

Pocket Code [4] is a mobile visual programming framework for smartphones and tablets. Pocket Code allows children and non-IT experts to develop their own programs or games in a quick and very intuitive way. They can create, remix and share their own games, applications, interactive music videos and many kind of other apps, directly on their smartphone or tablet.
The programs are developed and built by dragging some "elements" onto the screen used for the development of the program: Pocket Code provides a visual programming environment in "LEGO" style where users can drag blocks and add them to their programs.
Each block is associated with a code declaration, such as a conditional sentence, a declaration for defining variables, for checking certain conditions, and so on.

Blocks are grouped into categories and each category is associated with a different color in order to make their purpose clearer.
The user can add images and sounds to the program. She/he can also have immediate feedback on the program that has developed so far, by tapping the "*play*" button and testing the program itself.
There are also some tutorials to help the user in the development phase of her/his first program.

1.3.2 Kodable

Figure 1.3: Logo of Kodable

Kodable [20] is an application that can be used by teachers and parents to teach children the basis of computer science.
It presents the key contents of this subject in a funny and clear way.
It is organized in seventy lessons with both on-screen and off-screen components. Each lessons includes student materials, instruction guidance and vocabulary robotics.
In the web site `https://www.kodable.com` you can find the detailed programs year-per-year that teachers have to follow during the lessons.
As shown in Figure 1.4, for each lessons the teacher can find a summary of the lesson with the objectives that students have to reach, the instructions about what the teacher should do during the lesson, a sort of guide for practical activities and some examples of exercises to check and evaluate student preparation.

From the students' point of view, Kodable makes learning how to learn how to program fun. They just have to drag and drop commands to program their fuzzy character. In this way, they learn the concepts of problem solving, computational thinking, sequence, conditions, cycles, functions and debugging. These are skills necessary for learning any programming language.

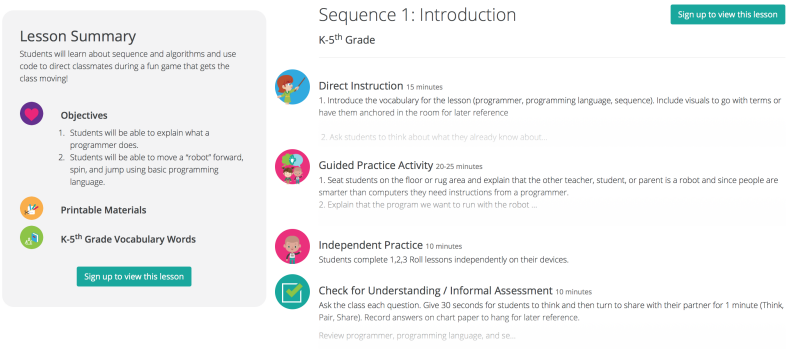


Figure 1.4: Lesson 1 in Kodable

1.3.3 Tynker

Tynker [21] is an online platform that allows children to learn to program intuitively.
It can be used as an educational tool that allows students to gradually learn how to create their own app or program robots and drones.
The key concept of Tynker is that learning to write code must seem fun and not so difficult for children: in this way, they can learn the basic concepts of computer science faster. Tynker follows the concept of "block programming" or "visual programming", so children do not need to have previous knowledge of programming languages.
Children can learn more efficiently and more effectively if they do something that is fun for them. They can also learn even faster if the tool they use follows their learning pace. This is Tynker's goal: the lessons are games, which are designed to handle forms of interactive learning, coding exercises and puzzles. Children learn to use the code as they are guided through interactive game-based courses to create projects and share their creations with friends and family.

There are no age limits for target users for Tynker. Even children under the age of seven can participate in Tynker: there is a tablet app for them.

Both children and teachers can register on this platform. For teachers, Tynker provides a set of tools to manage the lesson and track student progress.

As shown in Figure 1.5, Tynker's interface is similar to Scratch. But while Scratch was designed to program, Tynker was built to teach programming.
The app features initial lesson plans, class management tools and an on-line

Figure 1.5: Example of Tynker

showcase of programs created by students. The lessons are self-learning and simple for students to follow without assistance.

Every day, children have fun with the challenges of the learning path in their student bulletin board. When the kids complete the missions, they get XP, earn interesting trophies and unlock new characters to use in their projects. This is the dome to keep children's interest, attention and involvement high.

Tynker also provides a sort of "Parental Dashboard", in which parents and/or teachers can track the progress of children and students, manage the subscription and share children's creations.

1.3.4 Hopscotch

Hopscotch [22] looks a lot like Scratch and Tynker and uses similar controls to drag blocks into a workspace, but it only works on the iPad. The controls and characters are not as extensive as Scratch and Tynker, but Hopscotch is a great tool to start helping students with no programming experience learn the basics of programming, logical thinking and problem solving.

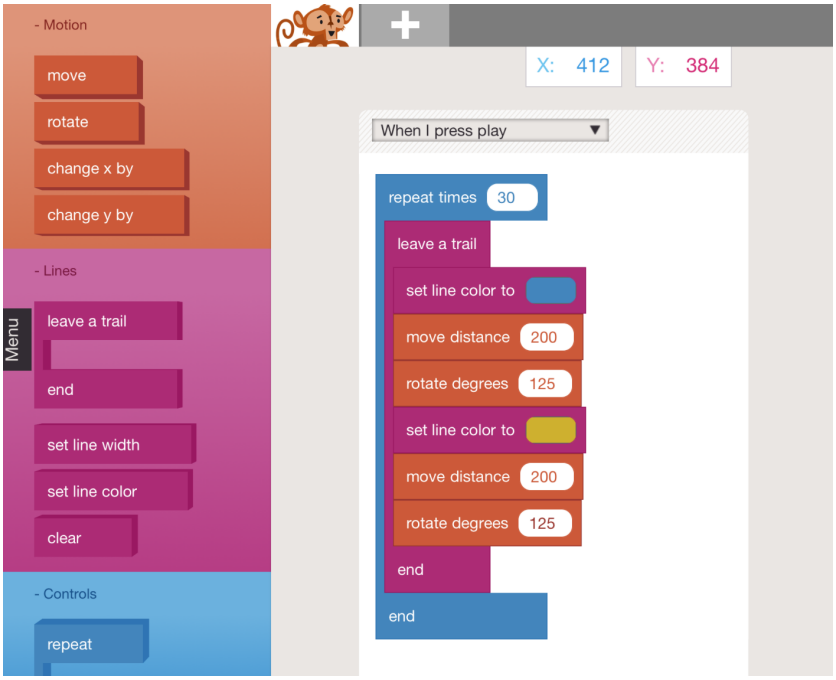


Figure 1.6: Example of Hopscotch

Figure 1.6 shows how the typical Hopscotch screen is. On the left side of the screen, there is a list of all the possible blocks that the user can choose to use. They are grouped into categories (movement or movement, line or drawing, control, looks, operators or variables and custom) and each category is associated with a different color. On the right side, the sequence of blocks chosen by the user for the development of her/his application is shown.

Children have complete freedom to do any type of application they want. The way they can do this is very intuitive and simple. They can also share the project and get immediate feedback on their programs or games.

1.4 Code completion

Code completion is a feature built into some programming environments to make the programming phase easier and faster.
A simple example is the IntelliJ IDEA integrated development environment (IDE) for the development of computer software. While the user is writing

her/his code, some popup windows appear on the screen to suggest a list of possible methods that can be called or what type of variable to use at that precise point in the code, given the lines of code written up to that moment .
This is useful not only to speed up the programming phase, but also to reduce typing errors and other common errors.
This feature can also be beneficial for applications such as Pocket Code, Kodable and the other applications I reviewed. The reason is that in general, users of such applications are children, who do not have a profound knowledge of programming languages. By integrating a code completion feature, we can help them develop their applications.
There can be essentially two types of code completion systems:

1. Statistical code completion systems [26]: they are built from a statistical analysis of many applications or code files. They can suggest what the user can write *next* given all the instructions, variables, classes and methods used previously. The weak point of these systems is due to their very nature. In fact, they are trained on a certain dataset: from this dataset they extract statistics which they then use to give the suggestion to the user. But if the dataset on which they were trained does not contain the particular sequence adopted by the user, then they are not effective: they will give a "null" response, since they have never "seen" that particular input, and they are not able to give a suggestion.

2. Contextual code completion systems [27]: they are statistical code completion systems that also take into account the grammar of the programming language, the code category and the context (i.e., the definition of the class, the definition of the function and the tabs and spaces).

1.5 Aim of this thesis

As the development of educational applications like Kodable, Hopscotch and Tynker is gaining momentum, I decided to focus on a specific application, Pocket Code, and to add some features to make the application easier to use. I wanted to design an efficient code completion system that could help the user during the development of her/his project (i.e., an application or a game).

In Chapter 2 I will explain in greater detail what Pocket Code is and its main components.

After the analysis, in Chapter 3, I will explain how I implemented some *suggestion-systems* for Pocket Code: during the development of the application, these systems can help the user by suggesting which block to use next, given the sequence of blocks that have been used up to that point.
In Chapter 4 I will describe in detail the three code completion systems I have implemented. All three share the same underlying data structure, but differ in the characteristics of the sequence of blocks they receive in input. I also realized a sort of fusion of these systems, to see if it would be more effective and efficient to combine the suggestions of one or two different systems.
In Chapter 5 I will show how I implemented the integration of this system into the application.
Finally, in Chapter 6 I will draw conclusions and sketch possible future developments.

Chapter 2

Pocket Code

2.1 What Pocket Code is

Catrobat [5, 6] is a free and open source visual programming language that allows young or inexperienced users to develop their own animations and games using their Android phones or tablets. The version of Catrobat developed for Android smartphones is called Catroid and is available on the Google Play Store as "Pocket Code". An interpreter for Catrobat and a mobile IDE are combined in the Pocket Code app.

Wolfgang Slany [1–3] is the head and founder of the Catrobat project and since 2010 has been developing Pocket Code. He is a full professor of computer science and head of the Institute for Software Technology at the University of Technology of Graz. His research topics include software quality, agile development and project management, computer science education, visual programming, interaction design and mobile development systems.

Catrobat wants to free the creativity of its users by encouraging them to create their own apps and content.

The Catrobat programs are written in a visual Lego-style, where individual commands are joined together by arranging them visually with the fingers.

One of the feature of Catrobat is that apps can be written solely using smartphone or tablet. The user does not need to use the keyboard or mouse, as for example in Scratch. Catrobat focuses on small devices where users interact through multi-touch sensitive screens.

As I wrote, Catrobat is an open source visual programming language: the system includes a community website where users can upload and share their projects with others. Every user must be registered to the Catrobat comunity (`https://share.catrob.at/pocketcode/login`). After this step, she/he can upload her/his project: her/his work is now open source and under a free software license. All (even non-registered users) can download and edit every projects found on the website: they can reorganize the code of the application, add new features and change the current behaviour of the project. This is called "remixing" and was a core idea behind the Scratch on line community.

2.2 The structure

2.2.1 Home Page

This (Figure 2.1) is how the home page of Pocket Code appears.

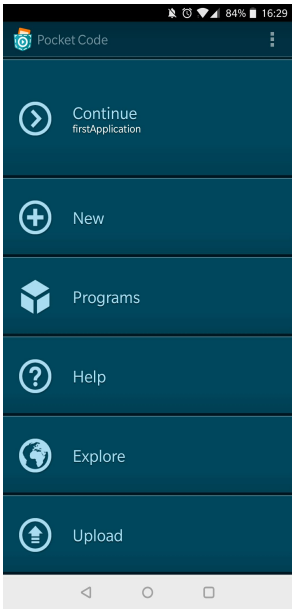
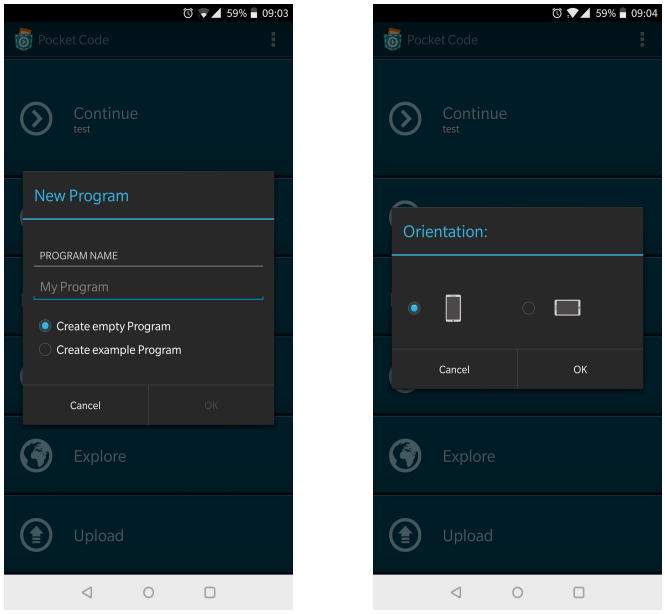


Figure 2.1: Pocket Code: home page

The first field is named Continue. It allows the user to continue developing the last application she/he was working on. The status of the last applica-

tion is saved and the user can add new features (that is, she/he can add one or more blocks), remove blocks, modify the parameters and everything she/he wants to change. After pressing this button, a new activity appears (see Figure 2.5) where she/he can find the list containing the blocks used in her/his work up to then.

The second field is New. After pressing this button,, the user can create a new application or a new game. As you can see from Figure 2.2a and Figure 2.2b, she/he can decide the name of the new project, whether it will be a new project or whether it will follow the tutorial project and the orientation of the project (landscape or portrait). Once these decisions have been made, the activity mentioned above appears (see Figure 2.5).



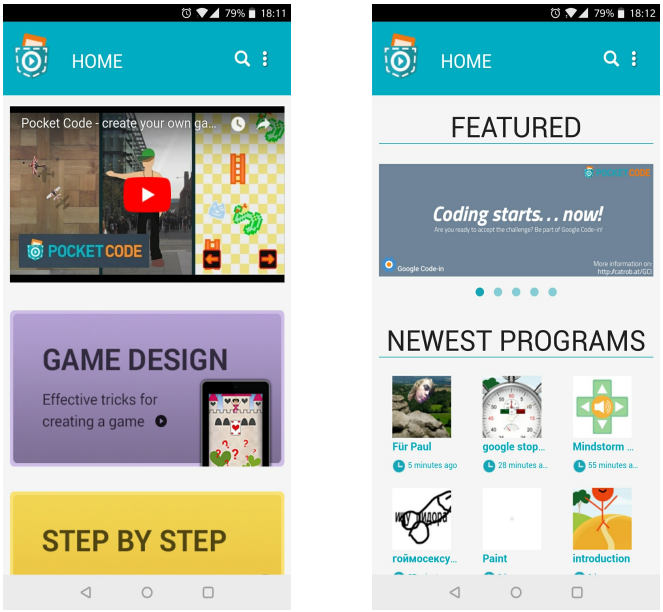
(a) Creation (b) Orientation

Figure 2.2: New project

The third field is Program. This button allows the user to see the structure and the blocks of the current application being developed. The Program button shows the list of blocks used by the user in her/his work based on what is written in Section 2.2.2.

The fourth field is Help. By tapping this button, the user goes directly to the page you can see in figure Figure 2.3a and which you can find at this

web site: `https://share.catrob.at/pocketcode/help`. Here the user can find a video gives an overview of what Pocket Code is and what users can do through it, some tricks to create a game, ten basic steps to make the user more familiar with the application, some tutorials and platforms where users can discuss together sharing ideas about developed applications or problems they have found.



(a) Help page (b) Explore page

Figure 2.3: Pocket Code

The fifth field is Explore. By tapping on it, the user is sent directly to the home page of the website of Pocket Code at `https://share.catrob.at/pocketcode/`. Here the user can "explore" the Pocket Code web world, seeing all the applications that have been developed and where she/he can find some useful links and some information on legalities and community statistics (see Figure 2.3b).

The last field is Upload. When the user finishes developing her/his application, she/he must upload all her/his work by tapping on this button. IThis pop-up window appears (see Figure 2.4) where, after logging in or registering, she/he can upload her/his work in the website of Catrobat. Once she/he has done this, the application or developed game becomes "open" and everyone can download or remix it.

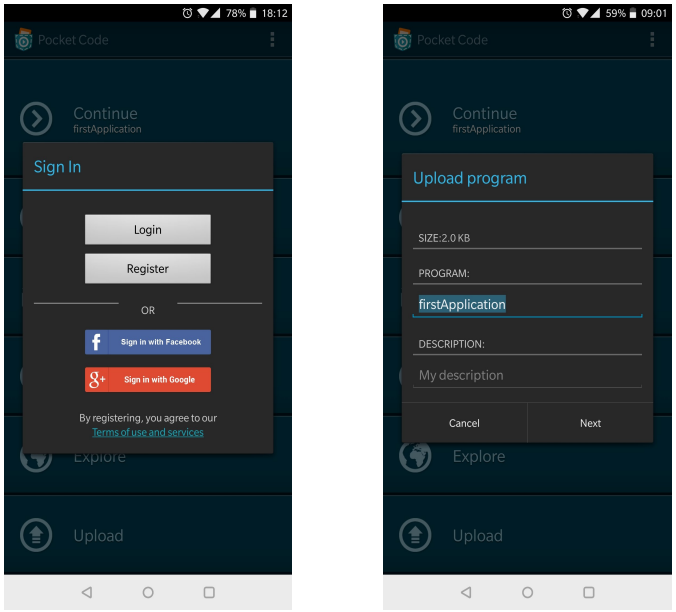


Figure 2.4: Pocket Code: upload pages

2.2.2 Structure of the application

The activity in Figure 2.5 shows the user the structure of the application that is developing so far. It is divided into two sections:

1. Background: once the user taps on this button, she/he can decide which type of object to add to her/his application (see Figure 2.6). It can be a script (see Section 2.3 to understand the key concepts of *brick* and of *script*), an image that the user wants to use as background or a sound.

2. Objects: the user can add some images taken from her/his gallery of her/his own device or she/he can draw the image she/he would like to use. The user must assign a name to this new object. Then, by tapping on the element, she/he can assign some *script*, a background or some sounds to the object: this item is treated as the *background* object above. We can say that the item *background* is the default object and then the user can customize her/his application by adding custom elements in the *Objects* section.

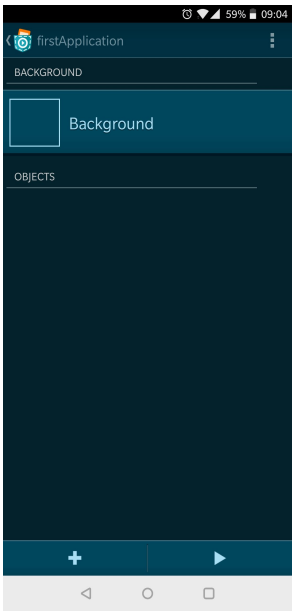


Figure 2.5: Pocket Code: structure of developed application

2.2.3 Categories of blocks

In Section 2.1 I said that the language used by Pocket Code is written in a visual Lego-style. The blocks which can be used by the user to built and create her/his application are divided into eight categories (see Figure 2.2.3).

Each block represents a declaration of a variable or any type of statement that can typically be found in a source code. However, the user ignores this: the user is not asked to have any knowledge of the programming languages and how they are structured and the rules that follow. The user simply has to choose which block to use and drag it to the box that contains the blocks of her/his program. The colors of the blocks and the words or the instruction written on them help the user during the development phase of the program. When some blocks require it, the user must specify or complete some fields written on them.

In the version of Pocket Code that I analysed there were, as I said before, eight categories: Events (dark orange), Control (light orange), Motion (blue), Sound (violet), Looks (light green), Pen (dark green), Data (red) and My Bricks (bright green).

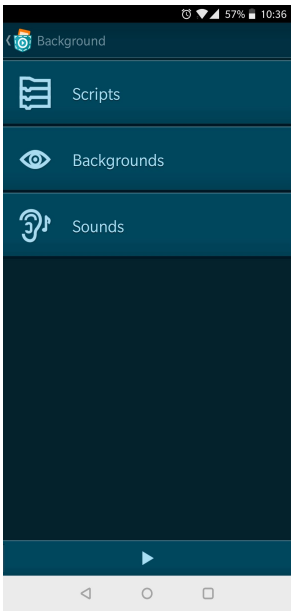


Figure 2.6: Pocket Code: background section

List of categories

Each category groups the blocks from a semantic point of view.

EVENT. Each block belonging to this category represents what could happen as a consequence of any type of *event*. An event can be generated when the user taps something on the screen, when the program starts, when there is some message received, when an internal variable of the program takes on a certain value, and many other, as you can see in Figure 2.8a.

- *When program started*: it runs the script when the program starts.
- *When screen is touched*: it runs a script when the screen is touched.
- *When tapped*: it runs a script when the related object is touched.
- *When I receive*: it runs a script when it receives specified broadcast message.
- *Broadcast and wait*: it sends a message to all sprites and waits.
- *Broadcast*: it sends a message to all objects.
- *When physical collision with (object)*: it runs a script when if physical collision with another physical object occurs.

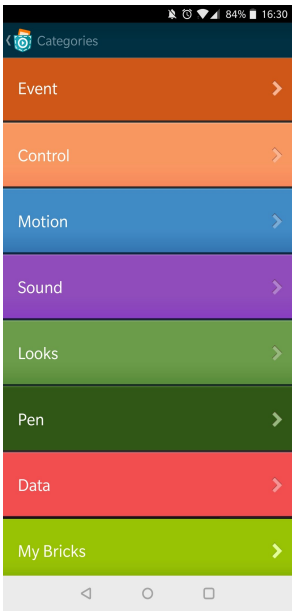


Figure 2.7: Pocket Code: categories

- *When (1<2) becomes true*: it runs a script when the given condition or value becomes true.
- *When background changes to (background)*: it runs a script when backdrop switches to a certain background.

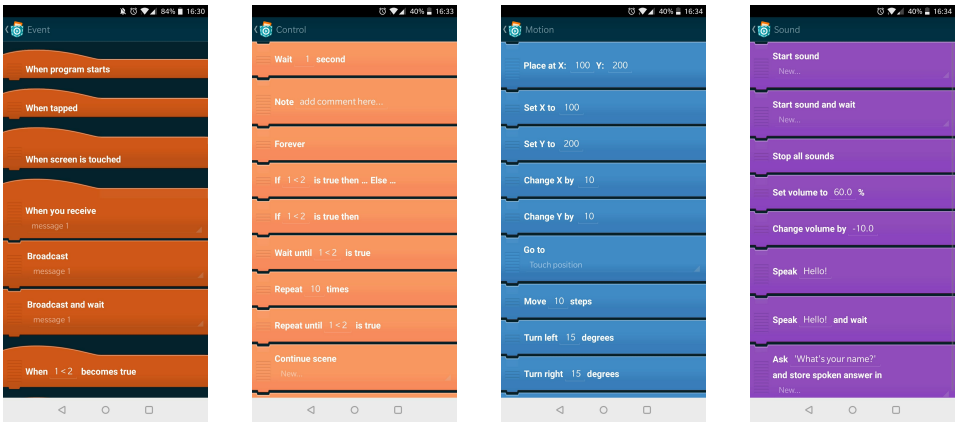
CONTROL. From Figure 2.8b you can see that each block in this category implies the need to carry out some kind of control during the execution of the application. These inspections cover the action of waiting some time, of checking if some variables assume certain values, of repeating the execution for a certain number of times and many other.

- *Wait (1) second*: it waits a specified number of seconds, then continues with next brick.
- *Note*: the user can add a comment to her/his Pocket Code program.
- *Forever*: it runs the bricks infinitely.
- *If (1<2) is true then ... else ...*: if the condition is true, it runs the bricks inside the if-area. If not, it runs the bricks inside the else-area.
- *If (1<2) is true then*: if the condition is true, it runs the bricks inside the if-area.

- *Wait until (1<2) is true*: it waits until condition is true, then runs the blocks below.
- *Repeat (10) times*: it runs the enclosed bricks a specified number of times.
- *Repeat until (1<2) is true*: it repeats blocks that follow until condition is true. It checks to see if condition is false; if so, it runs blocks inside and checks condition again. If condition is true, goes on to the blocks that follow.
- *Continue scene (scene)*: it jumps to the next scene as soon as the script is triggered and continue this scene.
- *Start scene*: it starts with the chosen scene from the beginning as soon as the script is triggered.
- *Create clone of (Oblect)*: it creates a clone (temporary duplicate) of the specified sprite. The user has to make sure she/he has chosen the sprite she/he wants to clone from the menu in the block.
- *Delete this clone*: it deletes the current clone.
- *When I start as a clone*: it tells a clone what to do once it is created. The attached script is triggered as soon as the clone is created.
- *Stop script/s (Stop this script / Stop all scripts / Stop other scripts of this object)*: it stops scripts within the object.

MOTION. As you can see from Figure 2.8c, the blocks in the Motion category can be used to reorganize images or other objects on the screen: the user can specify the value for the variables x and/or y (twith respect to the Cartesian coordinate system, where x = 0 and y = 0 are equivalent to the coordinates of the center of the screen), she/he can rotate the image/object to the right or left by a certain amount of degrees and many other things.

- *Place at X:(100) Y: (200)*: it place the sprite to the specified X and Y position.
- *Set X to (100)*: it sets the sprite's X coordinate.
- *Set Y to (200)*: it sets the sprite's Y coordinate.
- *Change X by*: it changes the sprite's X coordinate values by the given increment.



(a) Event (b) Control (c) Motion (d) Sound

Figure 2.8: Pocket Code: bricks in each category (part 1)

- *Change Y by*: it changes the sprite's Y coordinate values by the given increment
- *Go to (Touch position / Random position / object)*: the object goes either to the touch position, to a random position (any X/Y position on the screen) or to another chosen object.
- *Move (10) steps*: Move the sprite a certain number of steps (e.g.: 10).
- *Turn left (15) degrees*: it turns the sprite to the left (counter clockwise) by the specified degrees.
- *Turn right (15) degrees*: it turns the sprite to the right (clockwise) by the specified degrees.
- *Point in direction (90) degrees*: it sets the direction of the current sprite (in degrees).
- *Point towards* _ _ _: it sets the direction of the current sprite regarding another object.
- *Glide (1) second*: it glides within a certain time to the specified X,Y position.
- *Set rotation style*: it sets the way the sprite can rotate (left-right only, all-around, don't rotate).
- *Go to*: it brings the sprite to the given layer so it covers all other sprites with overlapping positions.

- *Vibrate for (1) second*: the user lets her/his device vibrate for a certain number of seconds.
- *Set motion type to*: the object is influenced by gravity, collisions, etc., e.g. a ping-pong ball collides with other dynamic and fixed sprites.
- *Set velocity to X: 0.0 Y:0.0 steps/second*: it sets the object's velocity along both X and Y axes.
- *Rotate left (15) degrees/second*: it sets the object's counter-clockwise rotational speed in degrees/second.
- *Rotate right (15) degrees/second*: it sets the object's clockwise rotational speed in degrees/second.
- *Set gravity for all objects to X: 0.0 Y: -10.0 steps/second*^{2}: it changes the physics world's gravity which affects all dynamic physics objects. Both positive and negative values are allowed for gravity on both X and Y axes.
- *Set mass to 1.0 kilogram*: it determines a object's mass. Accepted values are 0 and above. Note that increasing an object's mass will not increase the speed with which it will "fall" due to gravity.
- *Set bounce factor to (80.0) %*: it determines how much of an object's energy/velocity is lost (or gained) upon collision with another physics object. Both colliding objects' Bounce Factors are used to calculate how "violently" the objects bounce off of each other. Accepted values are 0 and above, factors greater than 1 are also supported. If both colliding objects have a Bounce Factor of 0 they do not bounce at all upon collision.
- *Set friction to 20.0 %*: it determines how fast/easily one physics object can glide along another. Accepted values are between 0 and 1, values greater than 1 are accepted as well. The higher the objects' friction values, the slower they will glide.

SOUND. Figure 2.8d shows the first blocks that belong to Sound category: by dragging them inside in its application, the user can play or stop any type of sound (musical media stored in her/his own device or the sound of her/his registered voice), she/he can make the application say something by specify what it should say, she/he can change the sound volume and she/he has also other options to choose from.

- *Start sound (Sound)*: it plays a sound and continues with the next brick immediately. The user can record a sound or choose one from the media library.
- *Start sound and wait*: it plays a sound and waits until the sound is finished before continuing to the next block.
- *Stops all sounds*: it stops all playing sounds.
- *Set volume to (60) %*: it sets the volume for sound replay to a certain value.
- *Change volume by (-10.0)*: it changes the volume for sound replay by a certain value.
- *Speak (Hello!)*: it speaks the phrase with the voice of the phone.
- *Speak and wait*: it speaks the phrase with the voice of the phone and waits until the sound is finished before continuing to the next block.
- *Ask (text) and store recorded answer in (variable)*: it asks a question and recorded your answer. The answer is stored in a variable and can be used.

LOOKS. Blocks belonging to this category (Figure 2.9a) allow the user to change the appearance of her/his game or application, changing the size or the position of objects on the screen or changing their colors and other their properties.

- *Set to look (look)*: the user can switch appearance to change the look of a object.
- *Next look*: it switches the sprite to its next look (the user can change the order of the different looks of one object).
- *Previous look*: it switches the sprite to its previous look (the user can change the order of the different looks of one object).
- *Set size to (60) %*: it sets the size of the current sprite.
- *Change size by (10)*: it change the size of the current sprite about the specified amount.
- *Hide*: it makes the sprite invisible.

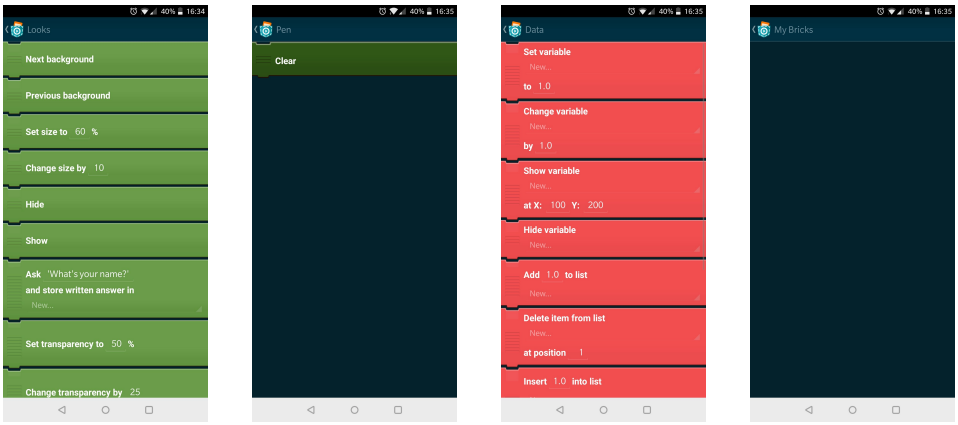
- *Show*: it makes the sprite visible on the screen.
- *Set transparency to (50) %*: it sets the sprite's transparency to a specific value.
- *Change transparency by (25)*: it changes the sprite's transparency by the specified amount.
- *Set brightness to (50) %*: it sets the sprite's brightness to a specific value.
- *Change brightness by (25)*: it changes the sprite's brightness by the specified amount.
- *Set color to (0.0)*: it sets the colour of the sprite to the given colour.
- *Change color by (25.0)*: it changes the colour of the sprite by the given amount.
- *Turn camera (on / off)*: it turns the devices camera on/off.
- *Use camera (back / front)*: the user can choose between the front or the back camera that should be used.
- *Turn flash-light (on/off)*: it turns the flash of the device either on or off.
- *Set background*: it switches to the specified background.
- *Set background and wait*: it switch to the specified background and wait for its scripts to finish.

PEN. In the current version of Pocket Code on my device, I have found only one block which belong to this category (Figure 2.9b).

- *Clear*: it clears all pen marks and stamps from the Stage.

There are other blocks which are not yet present in the version of Pocket Code that I analysed, but that we can find in some new applications and games (for example in the applications that I have used as "validation set" in Section 4)

- *Pen down*: it puts down sprite's pen, so the sprite will draw as it moves.



(a) Looks (b) Pen (c) Data (d) My Bricks

Figure 2.9: Pocket Code: bricks in each category (part 2)

- *Pen up*: it pulls up sprite's pen, so it won't draw as it moves.
- *Set pen size to (4)*: it sets pen's thickness.
- *Set pen color to Red (0.0) Green (0.0) Blue (255.0)*: it sets pen's color, based on choice from RGB values.
- *Stamp*: it stamps the sprite's image onto the Stage.

DATA. Using the blocks in this category (see Figure 2.9c), the user can manage variables and resources involved in the application she/he is developing.

- *Set variable (variable) to (1.0)*: the user can set the variable to a certain value.
- *Change variable (variable) by (1.0)*: the user can change the variable by a certain value.
- *Show variable (variable) at X: (100) Y: (200)*: it shows the value of the variable at a specific X and Y coordinate.
- *Hide variable (variable)*: it hides the variable so it is not visible on the stage.
- *Add (1.0) to list (list)*: the user can add a item with the given value to the list.

- *Delete item from list (list) at position (1)*: the user can delete the item at chosen position.
- *Insert (1.0) into list (list) at position (1)*: the user can insert a new item at the chosen position.
- *Replace item in list (list) at position (1) with (1.0)*: the user can replace the item at the chosen position with the new value.

MY BRICKS. In this category I have not found any block, perhaps because this category is a new feature and the code related to its blocks has not yet been loaded (see Figure 2.9d).

2.3 Basic components: bricks and scripts

All the blocks presented in Section 2.2.3 are divided into two groups: blocks called *script* and the others called *brick*.

2.3.1 Script

A *script* is a block which has a round shape from the visual point of view. From the semantic point of view, it has the task of grouping together some blocks that must be executed in sequence to realize a specific action to be performed following some event.
For example, the block *"When program started"* is considered a *script* : it groups all the bricks that need to be executed at the beginning of the program.
In this sense, we can use scripts to divide the applications developed by the usersby creating block sub-sequences: each application can be seen as a program consisting of a certain number of scripts.
Because scripts can not communicate directly, a broadcast transmission mechanism is used for communication between scripts.

2.3.2 Brick

From the visual point of view, a *brick* has a square shape.
A brick represents the atomic part of each script within the application. Each

block that does not belong to the set of script blocks is a brick: so it can have many different characteristics depending on the category it belongs to.

In the analysis of the applications, I found 148 different blocks, each identified by an unique integer number.
In Section 2.2.3 I showed a list of only 86 blocks: this depends on the different versions of the application available and installed on the physical smartphone or tablet and the on the version of the code data that we can find in the repository for the source code of Pocket Code. For example, I did not analyse the blocks belonging to the Arduino or Lego NXT categories. Now I will analyse them for a complete understanding of Pocket Code (see Figure 2.10).

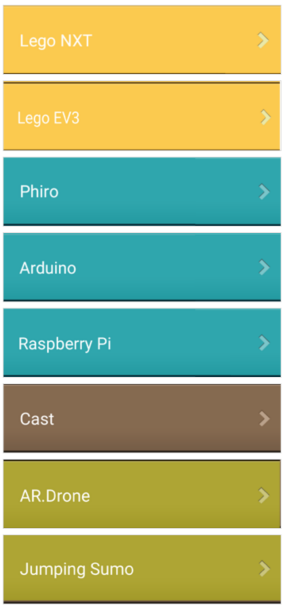


Figure 2.10: Pocket Code: new categories

LegoNXT.

- *Turn NXT Motor (A / B / C / B+C) by (*180°*)*: it activates the selected motor(s) and stops the motor(s) again after it spun for the entered angle. It is possible to select each output-port separately or use the two output-ports B and C at the same time. The direction the motor(s) will spin is controlled by positive/negative values of the entered angle.
- *Stop NXT Motor (A / B / C / B+C / All)*: it stops the movement of the selected motor(s). It is possible to select each output-port sepa-

rately or use the two output-ports B and C or all output-ports at the same time.

- *Set NXT Motor (A / B / C / B+C) to (100)% speed*: it activates the selected motor(s) and spins the motor(s) with the entered speed until a stop-command is issued to the motor(s). It is possible to select each output-port separately or use the two output-ports B and C at the same time. The direction the motor(s) will spin is controlled by positive/negative values of the entered speed.

- *Play NXT tone for (1.0)seconds with a frequency of (2) x 100Hz*: it plays a tone on the NXT brick for the entered duration in the entered frequency. The NXT brick supports frequencies between 200Hz and 14000Hz.

LegoEv3.

- *Set EV3 Motor (A / B / C / D / B+C) to (100)% speed*: it activates the selected motor(s) and spins the motor(s) with the entered speed until a stop-command is issued to the motor(s). It is possible to select each output-port separately or use the two output-ports B and C at the same time. The direction the motor(s) will spin is controlled by positive/negative values of the entered speed.

- *Stop EV3 Motor (A / B / C / D / B+C / All)*: it stops the movement of the selected motor(s). It is possible to select each output-port separately or use the two output-ports B and C or all output-ports at the same time.

- *Turn EV3 Motor (A / B / C / D / B+C) by (*180°*)*: it activates the selected motor(s) and stops the motor(s) again after it spun for the entered angle. It is possible to select each output-port separately or use the two output-ports B and C at the same time. The direction the motor(s) will spin is controlled by positive/negative values of the entered angle.

- *Play EV3 tone for (1.0)seconds with a frequency of (2) x 100Hz with (100)% Volume setting*: it plays a tone on the EV3 brick for the entered duration in the entered frequency. The EV3 brick supports frequencies between 200Hz and 10000Hz. The volume of the sound can be entered in percent.

The wiki web site of Catrobat states that the informations about bricks of categories Phiro, Arduino, Raspberry Pi and Cast will be available soon. Here I report only a list of possible blocks for each category.

Phiro.

- *PhiroMotorMoveForwardBrick*
- *PhiroMotorMoveBackwardBrick*
- *PhiroMotorStopBrick*
- *PhiroPlayToneBrick*
- *PhiroRGBLightBrick*
- *PhiroIfLogicBeginBrick*
- *SetVariableBrick*
- *new SetVariableBrick*

Arduino.

- *ArduinoSendDigitalValueBrick*
- *ArduinoSendPWMValueBrick*

Raspberry Pi.

- *WhenRaspiPinChangedBrick*
- *RaspiSendDigitalValueBrick*
- *RaspiPwmBrick*
- *RaspiIfLogicBeginBrick*

Cast.

- *WhenGamepadButtonBrick*

AR.Drone. For this category there are no available informations.

Jumping Sumo.

- *Move Jumping Sumo forward (1) seconds with (80) % power*: it moves the drone forward with the entered time and power. The percentage of the power with the value 100 is the maximum driving speed of the drone and the percentage downwards to the value 0 calculates the slower speed.
- *Move Jumping Sumo backward (1) seconds with (80) % power*: it moves the drone backward with the entered time and power. The percentage of the power with the value 100 is the maximum driving speed of the drone and the percentage downwards to the value 0 calculates the slower speed.
- *Animations Jumping Sumo (Spin)*: the drone supports many different funny animations, such as "Spin", "Slowshake", "Spinjump"
- *Sound (Normal) Volume (50) %*: different sounds can be selected, like "Normal", "Robot", "Insect" and "Monster". If the sound is activated, the drone continues every other brick with this chosen sound and volume.
- *No Jumping Sumo sound*: it turns off the sound.
- *Jump Jumping Sumo long*: the drone jumps forward and lands after a distance about one meter on the ground. In the upside down position, the drone can kick a ball.
- *Jump Jumping Sumo high*: the drone jumps up about eighty centimeter and lands on the ground. In the upside down position, the drone can kick a ball.
- *Rotate Jumping Sumo left (*90°*)*: it turns the drone to the left with the entered degree measure.
- *Rotate Jumping Sumo right (*90°*)*: it turns the drone to the left with the entered degree measure.
- *Turn Jumping Sumo*: it turns the drone around its own axis and stays in the upside down position. In this position, the drone can also do all other bricks. If the drone is in the upside down position, this brick turns the drone to the normal driving position.

- *Taking picture Jumping Sumo*: taking a picture with the drone camera and stores it to the internal storage on your own device. This brick creates a new folder "JumpingSumo" in the general photo directory and stores all further pictures from the drone there.

In total I found the documentation for 120 blocks.
For each block, I identified some features in the source code, which are listed in the Tables A.1 and A.2. in the appendix.

Chapter 3

Dataset and analysis

In this chapter I will talk about two basic phases: the collection of the dataset and then the analysis of it.

The purpose of this phase is to extract as much information as possible from the source code of the programs developed through Pocket Code. The information I got from the analysis concern:

- types of blocks used in the programs
- ordered sequences of blocks that make up each program
- ordered sub-sequences of blocks that make up each script
- check of the heterogeneity of the dataset.

Much attention was paid to the execution of these steps: the more detailed data I extracted, the more accurate the phase of implementation of the code completion system would have been.

3.1 Getting the dataset

3.1.1 Developed applications

My dataset consists of the source code of all applications that have been implemented through Pocket Code until 23 August 2017.

In Chapter 2 (Figure 2.4) I explained how the user can upload his application. Now I will explain how the program is displayed on the website

`https://share.catrob.at/pocketcode/`.

This web page is divided into six basic sections.

Featured

In this section there is a banner where some applications are showed in sequence.

Newest programs

Here the user can find the latest applications that have been developed or remixed.

Recommended programs

Here the user can find some sort of recommendation: more like (including "Thumbs up", "Laugh", "Love" and "Woow!") receives a program, higher up in this list .

Most downloaded

Applications that have been downloaded a higher number of times belong to this section.

Most viewed

Applications showed here are the programs that were most visited on the Pocket Code website.

Random programs

Here the user can find some randomly extracted programs, perhaps to have a suggestion on different types of applications available or that can potentially be developed through Pocket Code.

As an example, I chose a random program at this URL: `https://share.catrob.at/pocketcode/program/45045` [date: 2 January 2018] to show how an application appears on the web site.

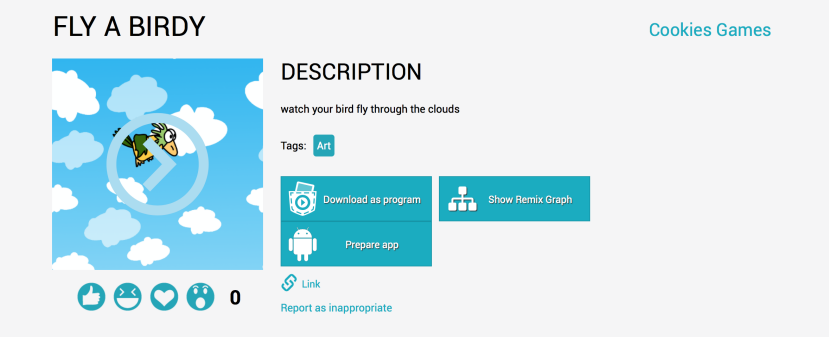


Figure 3.1: Pocket Code web site: program's general informations

As you can see in Figure 3.4 the user can find the name of the program, a brief description of what this application does, three buttons (one to download it as an application, one to download it as a program and the third to see the remixes of the program), the name of the author and the possibility to express some reactions about the program.

Each program developed by every user is identified by an integer number. This unique number is also used in the corresponding application URL on the Pocket Code website.

In the web address of each application the user can find further information about the application. In Figure 3.2 the user can find some numerical values about the program, such as the total number of bricks, the total number of scripts and the number of blocks for each brick category.
From the section showed in Figure 3.3, the user is able to immediately identify and understand the code structure thanks to a schema. The program is represented in the same way as it appears to the user during the development phase: the user sees the block structure of the bricks grouped by the scripts.

The last informations we can get from the web site about the developed programs are showed in Figure 3.4. In this box it is written the "age" of the program, its dimension and the number of downloads, views and remixes.

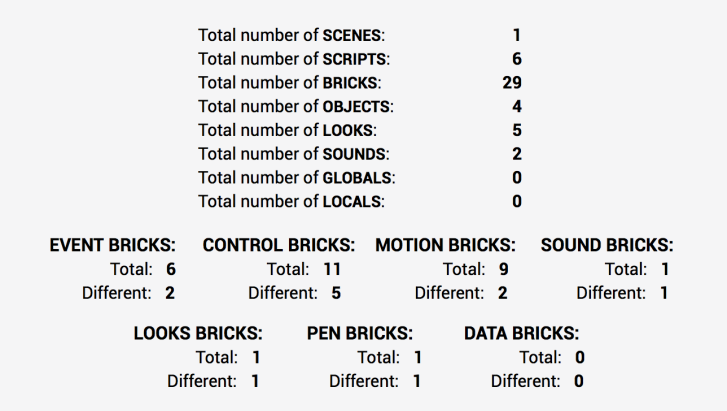


Figure 3.2: Pocket Code web site: program's general statistics

3.1.2 Size and permission

Before continuing with the download phase, I made an estimate of the size of the entire dataset to accept that it was contained and that it could be stored in my PC. To do this, I analysed the source code of each program's web page: from the *html* code files, see Figure 3.5, I collected all the tags that indicate the file size of the compressed program and sum all these values.
I wrote a program in Python to get all these values by browsing the web. The total size of the dataset I got was 88594.19 MB: it turned out to be a manageable size.

3.1.3 Download: crawler

To download the source code of each program, I implemented a sort of crawler in Java.
The structure of the URL lends itself well to an automatic analysis. The string that represents the URL of the *i-th* application is updated step by step by changing the value of i. Then a connection is created using the *Jsoup* library. After this step, the index i index is incremented by one unit and the process is restarted.
Since some URL do no exist (that is, not all i-th programs are still available on the Web site), the connection is established only after having verified the existence of that web address. If the website is not available, the connection is not requested, the i index is incremented and the process can continue.
Once connected to a URL, the crawler simulates a click on the "Download as program" button and the application starts the download phase.

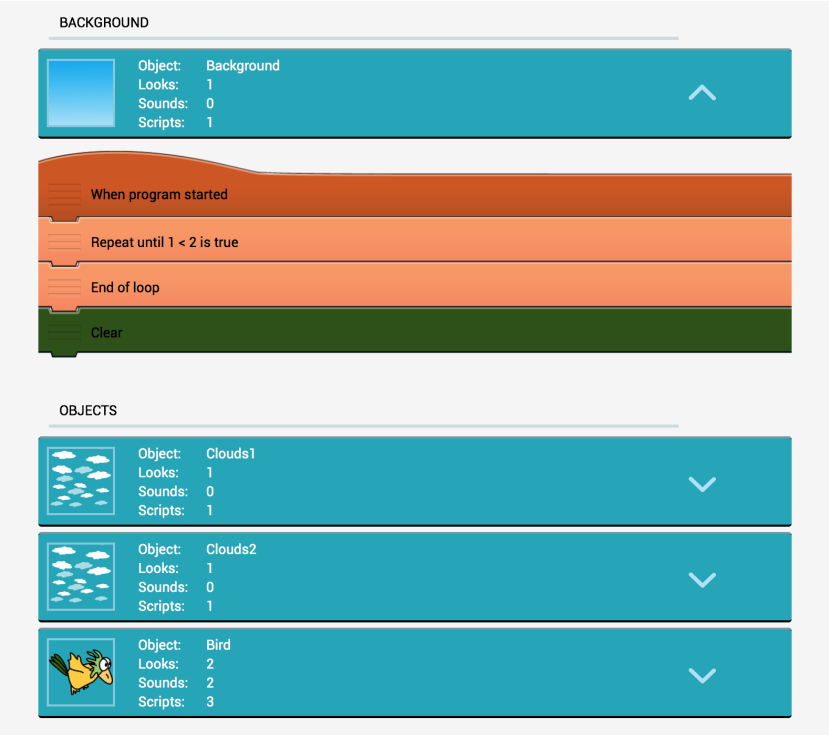


Figure 3.3: Pocket Code web site: program's code

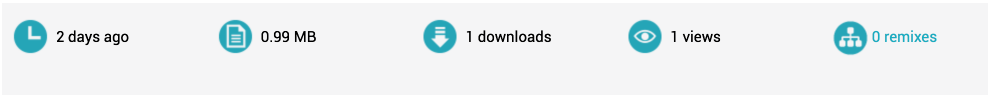


Figure 3.4: Pocket Code web site: program's basic info

3.1.4 Decompression and storage

After having obtained all the programs, they were decompressed to be subsequently analysed.

3.1.5 Data, number applications, size

The dataset acquisition process started on 23 August 2017 and all the programs available until that day have been downloaded.
The total number of programs downloaded is 31572, while the index that was used to set the work of the crawler was taken between 3299 and 36522. I

```
<div>
  <div class="img-size icon"></div>
  <span class="icon-text">0.99 MB</span>
</div>
```

Figure 3.5: Pocket Code *html* source code: program's dimension

started the download process from the value 3299 for the index and not from 0 for two reasons:

1. Before the identifier of *819-th* there are no available programs
2. From *820* to *3298* source files *xml* have a structure that is completely different from other programs, perhaps due to an update of the Pocket Code version.

The dataset size is the one that has been estimated in Section 3.1.2.

3.2 Analysis

Applications have been divided into subsets to improve efficiency and improve crawler performance.
The unpacked folder of each downloaded program contains data organized in other sub-folders.
I consider our example program again as I did in Section 3.1.1 for clarity. As you can see from Figure 3.6, in the folder you can find:

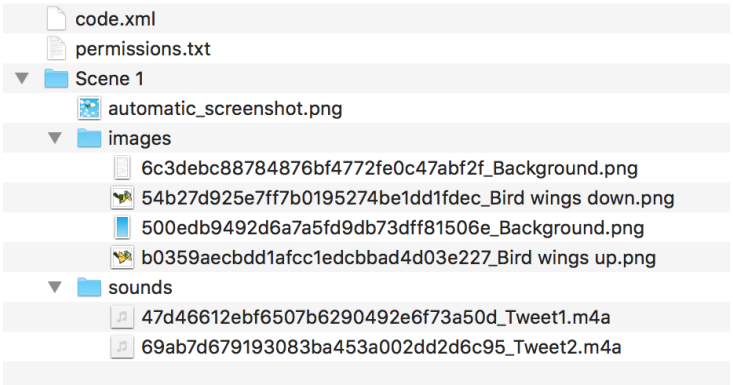


Figure 3.6: Program 45045's decompressed folder

1. *code.txt*: it is the most important file, it contains the entire structure consisting of scripts and bricks. This is the file that I analysed to extract informations and statistics on bricks and scripts.

2. *permission.txt*: it is usually an empty file.

3. other sub-folders, which contain the images and sounds used in the corresponding bricks.

Now we focus on the analysis of *code.txt*.

3.2.1 Structure of xml file

This file is well structured and lends itself well to an automated analysis.
The first part (Figure 3.7) is the *header* and it contains all the general infor-

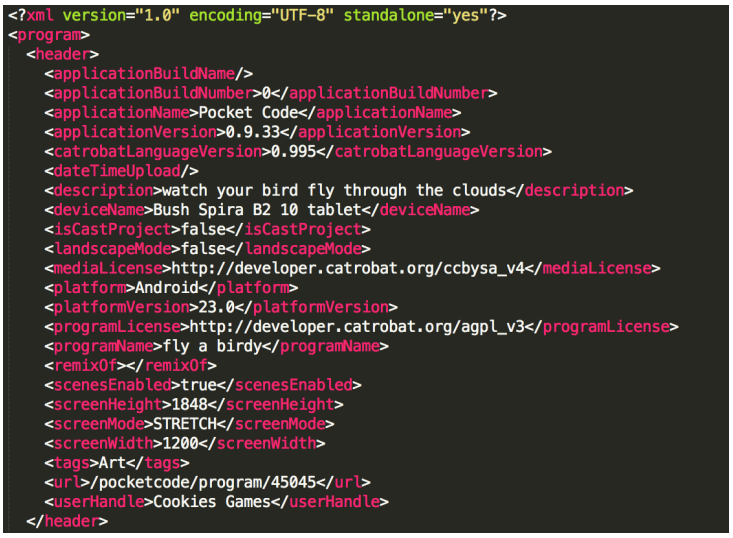


Figure 3.7: Program 45045's *xml* file: header

mation of which I have discussed in the previous sections such as the name of the programs, their description, the number of remixes, the version of the Catrobat language used and others.
The second part of this file is more interesting from the analysis point of view. It contains the list of each script used in the application, and for each script shows the name and characteristics of the bricks that belong to this script.
For example in Figure 3.8 you can read all the information on the script called

```
<scriptList>
  <script type="StartScript">
    <brickList>
      <brick type="RepeatUntilBrick">
        <commentedOut>false</commentedOut>
        <formulaList>
          <formula category="REPEAT_UNTIL_CONDITION">
            <leftChild>
              <type>NUMBER</type>
              <value>1</value>
            </leftChild>
            <rightChild>
              <type>NUMBER</type>
              <value>2</value>
            </rightChild>
            <type>OPERATOR</type>
            <value>SMALLER_THAN</value>
          </formula>
        </formulaList>
      </brick>
      <brick type="LoopEndBrick">
        <commentedOut>false</commentedOut>
      </brick>
      <brick type="ClearBackgroundBrick">
        <commentedOut>false</commentedOut>
      </brick>
    </brickList>
    <commentedOut>false</commentedOut>
    <isUserScript>false</isUserScript>
  </script>
</scriptList>
```

Figure 3.8: Program 45045's *xml* file: example script

"*StartScript*". It contains three bricks: "*RepeateUntilBrick*", "*LoopEndBrick*" and "*ClearBackgroundBrick*". For each brick, there are some additional tags that depend on the brick: for example, the tags enter values for the conditional statement written on the block (as seen in the brick "*RepeateUntilBrick*").

3.2.2 Analysis of xml file

I analysed all the *xml* file to extract as much information as possible. To do this, I wrote some programs in Python using the Integrated Development Environment (IDE) Pycharm [7]. Each program implemented has the function of analysing a specific aspect of the *xml* files.

Bricks identification

Once all the *xml* files for each application were obtained, Python was used to more thoroughly analyse the source code.
First of all, I wrote a program that counted the total number of occurrences of each script and every brick in the dataset.
At the beginning of the program, I defined a variable for each type of block: they are of type *int* and represent a counter. The name of each variable corresponds to the type of block to which it refers and all the values of the variables are initialized to zero.
Then the program is executed. The input shows the path where the dataset folders are stored. For each folder and for each *xml* file inside, the parser looks at the *"object"* tag, then the *"scriptList"* and *"brickList"* tags in the corresponding section .
Whenever you find a brick or script, call the *countElement* function on that object and increment the corresponding counter. If the parser does not find a variable with the same name as the block found in the *xml* file, then it creates a new variable with that name and assigns a value of one.

To make the analysis and comparison processes faster, I have identified each block with an integer.
Doing so, I discovered that the number of all possible blocks is 148. This number was also found looking at the source code of the Pocket Code's repository: I found classes written in Java for all the 148 blocks even if not all were still available for users. So they do not appear in the *xml* code, but they were in the source code.

List of all blocks

The second step of my analysis was to characterise each program by providing the list of blocks from which it was composed.

The first file I got was like the snippet of code you see in Figure 3.9.
For each program identified by an integer number (20001, 20002, ..., 20012 in the figure), there are the corresponding scripts and/or bricks inside it. Each block is identified by the corresponding integer number.
This identifier is associated with a number in square brackets that counts the number of occurrences of that block type in that specific program.
Bricks and scripts are presented in ascending order according to their id.

```
20001: 26[3],27[3],48[3],50[1],59[5],61[1],99[4],110[1],116[1],
20002: 26[3],27[3],48[3],50[1],59[5],61[1],99[4],110[1],116[1],
20003: 26[3],27[3],48[3],50[1],59[5],61[1],99[4],110[1],116[1],
20004: 13[1],26[1],48[1],59[3],89[1],95[1],99[2],110[3],116[1],
20005: 26[1],48[1],50[1],62[1],81[1],87[1],99[1],
20006: 61[1],74[1],75[1],99[1],100[1],110[2],116[1],
20007: 26[3],27[3],48[3],50[1],59[5],61[1],99[4],110[1],116[1],
20008: 3[1],26[2],48[2],50[2],59[1],61[1],62[1],99[2],110[1],116[3],
20009: 75[2],97[1],99[1],109[1],116[1],
20010: 20[1],72[1],99[1],101[1],110[1],
20011: 61[1],75[1],94[1],99[1],110[1],116[1],
20012: 18[1],50[1],99[1],
```

Figure 3.9: Fragment of list of all programs with blocks and frequencies

Then I created another hash map to associate the program with the bricks and scripts contained within it.
The process followed by this parser is divided into two phases.

1. A parser has been written in Python as before, but this time the parser collects the list of scripts and bricks found instead of just counting them. It writes the output of the calculations to a *txt* file.

2. Then a parser written in Java analyses the file returned from the previous step and builds an Hash Map: the keys are all the integer numbers that identify each program and the value are Array List containing integers representing scripts and bricks of that specific program.

After these two steps, I got four *txt* files. Each line represents the application, identified by the corresponding id, and the list of integer numbers that corresponds to the bricks and scripts used. The order in which the blocks appear in the program has also been maintained in the lists in these hash tables.
You can see a piece of one of the output files in Figure 3.10.

```
33453=[99, 61, 3, 116, 61, 61, 25, 8, 17, 113, 97]
33454=[99, 87, 59, 96, 110, 5, 145, 110, 96, 110, 96, 99, 87, 59, 145, 96, 5]
33455=[99, 75, 116, 61, 8, 17, 8, 110, 100]
33456=[99, 59, 96, 110, 96, 110, 27, 30]
33457=[99, 3, 117, 50]
33458=[145, 81, 145, 81, 99, 87, 59, 116, 5, 99, 87, 59, 116, 5]
33459=[99, 96, 110, 67, 81, 110, 81, 110, 47]
33460=[99, 86, 26, 50, 49, 110, 35, 48]
```

Figure 3.10: Fragment of list of all programs with blocks in order as they appear.

After this, the applications that do not contain any bricks or scripts (the corresponding Array List are equal in size to zero) were discarded because not relevant for the statistics.

List of scripts

From the analysis of the *xml* files and from the understanding of the structure of the program developed through Pocket Code, I noticed that much importance was given to the aggregation of bricks in a script.
With this in mind, a different type of analysis has been developed. Instead of finding a list of scripts and bricks for each applications, the list of bricks for each script was found.
This means that each script was identified by an integer number and the bricks associated with it were placed in a list of arrays.
The results of this analysis are showed in Figure 3.11.

```
2, 99, [ 81 89 89  ], 30002
3, 99, [ 87 26 59 110 94 81 27 81 110 30 110 48  ], 30002
4, 116, [ 61 81 110 30  ], 30002
5, 99, [ 87 26 59 110 94 81 27 81 110 30 110 48  ], 30002
6, 116, [ 61 81 110 30  ], 30002
7, 99, [ 87 26 59 110 94 81 27 81 110 30 110 48  ], 30002
8, 116, [ 61 81 110 30  ], 30002
9, 99, [ 87 26 59 110 94 81 27 81 110 30 110 48  ], 30002
10, 116, [ 61 81 110 30  ], 30002
```

Figure 3.11: Fragment of list of all scripts within programs with the corresponding bricks.

For each script, identified by an id, I keep track of the application ID I was considering and the corresponding list of brick arrays.
This type of analysis was very useful, because I discovered that scripts composed by the same bricks were used several times within the same application and also in different applications.

Types of programs

I analysed other information that the user can find on the web page of the developed programs. These were not numerical or statistical information, such as the number and type of bricks or scripts. Instead, they are fields that can be customized by the user, such as "title", "description" and "tags" that are entered by him during the loading phase of his program on the Pocket Code

```
root: /Users/Marta/Desktop/tesi/temp/25021
programName :SpaceInvaders by Egger
description :SchoolProject

root: /Users/Marta/Desktop/tesi/temp/25022
programName :dgg

root: /Users/Marta/Desktop/tesi/temp/25023
programName :njm
description :поцелуй это попа

root: /Users/Marta/Desktop/tesi/temp/25024
programName :Froger
description :unfinished-2

root: /Users/Marta/Desktop/tesi/temp/25025
programName :plane is hero

root: /Users/Marta/Desktop/tesi/temp/25026
programName :Cookie clicker Beta 1.0.
description :                  Beta 1.0.
                    News
- New menu
- Max 33 cookies
- Shop menu
```

Figure 3.12: Fragment of lists of descriptions and names of programs

website.
I wrote in Python a program that extracts these three informations from the *xml* of each program and I tried to cluster them in order to obtain a sort of semantic analysis of the applications in the dataset.

The result (a fragment is showed in Figure 3.12) was that there were too many variations for the same concept and often the content type of the program did not match what was indicated in the title or in the description.
So I looked for another way to create a sort of clustering of the dataset.

3.2.3 Validation of the dataset

The reason why I would like to make a sort of clustering of the programs in the dataset concerns the validation of the data I downloaded.

Just looking at the Pocket Code website home page (Figure 3.13) there are seven variations of the application "Flappy bird", where a bird is controlled by the user and must fly avoiding some obstacles that he can find during his flight.
Moreover, the Pocket Code tutorial teaches the user how to build a program for a compass: this means that almost every user, as his first program, would have built this kind of application.

Figure 3.13: Program Code's web site: random programs

To do a clustering process, I first normalized the dataset applications. Since I have identified 148 types of scripts and bricks, I rewrote each application as an array of 148 elements. The index i of each array position corresponds to the id of a block and the value associated to the i-*th* position indicates the number of occurrences of that block in the *xml* file (a fragment of the result of this analysis is showed in Figure 3.14).

```
0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 0 0 2 0 0 0 0 2 4 4 0 0 0 0 0 0 0 0 0 2 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 2 0 4 0 0
0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 1 0 0 0 0 2 4 4 0 0 0 0 0 0 0 0 0 4 2 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 2 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 2 2 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 3.14: *Normalisation* of the list of blocks for each program.

Then I used the *scikit* library and the k-means algorithm to find clusters from all these normalized arrays representing the various programs.
The number of clusters to use in the k-means algorithm was decided based on the number of applications considered and was fixed after considering the size of the clusters returned in output by the algorithm. For each cluster I found, I randomly chose some programs to check if they were really similar

or if they were variations of the same type of program.
In Figure 3.15 there is a fragment of the first ten clusters found in programs from id 20000 to 24999 using 1000 labels. For each cluster I indicated the number of elements of that cluster.

```
0=[24347, 24701, 24970, 20076, 20240, 20310, 20315, 20355, 20377, 20451, 20459, 20464, 20483, 20599,
20636, 20637, 20652, 20688, 20804, 20847, 20963, 20992, 21084, 21139, 21179, 21285, 21356, 21391,
21437, 21457, 21473, 21518, 21519, 22221, 22268, 22365, 22409, 22432, 22447, 22480, 22536, 22549,
22643, 22765, 22791, 23067, 23230, 23370, 23535, 23631, 23735, 24268] 52
1=[23726, 23611] 2
2=[21828, 24897, 20171, 21046, 21056] 5
3=[23285, 20137, 20269, 20272, 20941, 21000, 21277, 22169, 22195, 22239, 22250, 22819] 12
4=[24502, 24598, 24641, 24672, 24988, 20214, 20664, 20714, 20994, 21097, 21357, 21434, 21555, 21586,
21705, 22174, 22340, 22461, 22470, 22484, 22589, 22605, 22812, 22816, 22845, 22961, 23174, 23208,
23292, 23352, 23356, 23881, 23910, 23911, 24011, 24012, 24264, 24419] 38
5=[21423, 21422] 2
6=[23016] 1
7=[23000] 1
8=[22441, 21282, 21734, 21779] 4
9=[24448] 1
```

Figure 3.15: Fragment of programs' *Clustering*.

Out of a total of 31572 programs, 1428 are Flappy Bird variants, 136 tris, 113 minecraft, 109 galaxy war and then other clusters with far fewer elements.

From these analysis steps I get 31572 programs that represent a valid dataset to build the code completion system, as I have found many clusters that are composed of few elements. In the next chapter I will show how I implemented this system.

Chapter 4

Statistical automatic suggestions

In this chapter I will present the statistical systems I implemented for code completion in the Catroid visual language.
All these systems are written in Java to facilitate integration into the Pocket Code codebase.

The implemented systems share the main core of the code completion system's structure.

4.1 Key concepts

First of all it is important to underline the general considerations on the use of the statistics obtained in Chapter 2.
All the data collected during the analysis phase have been organised in a tree structure (see Section 4.2.1) to make data representation and exploration easier and faster.
For the construction of the *brick based system* (Section 4.3) the file containing the block sequences contained in each application was used, while for the *script system based* the file containing the brick sequences for each script was used.
The fundamental aspect on which the tree-structure is based is the *frequency* of blocks, that is, the number of times a given block is found in a sequence of bricks and scripts within a given application. The position of the block considered in the sequence is important and the system tracks it: for this reason, each sequence of blocks is represented as a path, a traversal, of the related tree. As long as a sequence has the same blocks as another sequence in the same positions, they share the same path in the tree. As soon as they show a different block in the same position, the paths diverges.

The main difference between the system based on bricks and the one based on scripts is the concept that each of them has of what a sequence is. For the system based on bricks, a sequence is the entire list of ids of each block contained in the application, while the second system considers the list of bricks for each script in the application as a sequence. This means that the tree obtained from the first system will have a depth greater than then the second and will be more sensitive to the position of the blocks within the application. On the other hand, the first system will be more precise in the suggestion to give to the user, since the "*history*" of all the blocks used before is kept in memory.

As you can see, these two systems have some pros and cons. I implemented both and then I compared their performance, trying to extract their positive aspects from both systems and merge them together to implement a better system.

The performance of the systems was evaluated as follows:

1. I downloaded the source code of other applications considering those with id that went from 36700 to 42000, that I did not use to train the systems.

2. I randomly chose a piece of the block sequence used by the user for that specific program.

3. I gave this piece of input sequence to the systems and got a suggested block.

4. I compared the suggestion provided by the systems with the *right* block used soon after in that specific program.

5. If the suggestion was the same as the block that actually was the next one in the program sequence, this meant that the system had been a success. Otherwise it had failed.

After all, the performance of these systems was not so good. That's why I implemented two other systems based on 3-gram. One was implemented only by considering sequences of three blocks contained within the complete list of blocks used by the user in the programs. In the other, a *smoothing* function has been added to take into account the possibility of "*null answer*" of the system.

These two other systems were evaluated as the previous systems.

In this chapter I will talk about the design and the implementation of the different systems, while in the next chapter I will talk about the integration of the best performing system in Pocket Code.
The systems are presented in this chapter according to the chronological order in which they were designed and implemented.

1. *Brick based system*: at the beginning I thought that keeping the order in which the blocks appeared within the programs was fundamental. The system is very precise from this point of view, but appears weak when a minimum variation is made along the sequence of blocks. This system will be described in the Section 4.3.

2. *Script based system*: this system is more stable than the previous one because it considers sub-sequences of blocks of a program, those contained in each script. Its weak point concerns the accuracy of the suggestion, as it does not take into account the complete chronology of the program. I will talk about this system in the the Section 4.4

3. *n-gram based system*: this system takes into account only a limited and pre-established number of blocks previously used by the user. I chose to use n = 3 and then to consider only the last two blocks present in the program created by the user. This system has the advantage of not needing much memory and is less inclined to give a null suggestion to the user. This system will be described in the Section 4.5.

To evaluate the performance of these systems, I initially considered the single block that the systems returned as output. Nowadays, however, many suggestion systems provide a list of possible aid to the user: examples can be in mobile devices when writing a message or in a text editor program like Intellij in which the user is suggested a possible list of methods to be applied to the object that is considered. So I decided to also consider the first three blocks returned by the systems to see if any of these was the correct one. As a number of blocks I chose three because it is the typical number of suggestions that are presented in a mobile device, given the limitation imposed by the size of the screen.

4.2 Core of the code completion system's structure

The core of the overall structure is composed of two elements: a "tree-structure" that has the function of organizing all the data obtained through the previous analysis phase and the way in which this tree is crossed.

4.2.1 The "tree-structure" of the code completion system

As I wrote in Chapter 1 the aim of this thesis is to help the user who is developing her/his own program through Pocket Code suggesting him what could be the most likely block that she/he would choose later in the realisation of the program.
With this in mind, I needed to construct a structure that kept track of all the possible block sequences that the user could have chosen to suggest the most likely sequence given the orderly sequence of blocks she/he had used up to that point.
The tree structure lends itself to be very useful to perform this task: a node of that tree can be associated to each block within a specific position in a given program.
Each nodes contains some useful and necessary information to the system, such as the name of the block and some kind of value that takes account of the "probability" of having that block in that program at that specific position.
Therefore the process of suggesting the most likely block can be reformulated. The system must traverse the tree structure based on the blocks used by the user. When it arrives to the node which correspond to the last block used by the user, she/he must suggest the block corresponding to the most probable child of the corresponding node in the tree.

Tree's nodes

Each node of the tree is an instance of the*MyNode.java* class.
The n node has the following attributes:

- *int nodeId*: it is an integer used to identify n.

- *int brickId*: is the iblock identifier associated with *n*. For example, a node that matches the "When program starts" block (see Table A.1 and Table A.2) has `brickId = 99`.
- *int occurence*: it indicates the number of occurrences of the block associated to *n*. The calculation of this value is an important and crucial aspect for the efficiency of the systems and it depends on whether the systems we are using are based on bricks or scripts. I will explain later in Section 4.3 and Section 4.4 how I calculated this value.
- *IntArrayList children*: it is an array list of integer numbers. Each integer represents the *nodeId* of each child of *n*.
- *int parent*: it is an integer number representing the *nodeId* of *n* parent in the tree-structure.
- *int level*: it is an integer number that indicates in which tree's level the node is contained.

Tree's levels

Each level of the tree is an instance of the *Level.java* class.
The level *l* has the following attributes:

- *ObjectArrayList<MyNode> list*: it is an array list of *MyNodes* elements belonging to *l*.
- *int size*: it is an integer number that indicates the number of elements belonging to *l*.

I implemented three different tree-structures: the first was filled by the lists of all blocks for each application (see Section 4.3), the second by the lists of all blocks for each script (see Section 4.4) and the third was constructed so that it can be used by the n-gram based system (see Section 4.5).

4.2.2 The tree's traversal

Even if the input given to the two systems is different, they follow the same strategy to cross the tree-structure.
The blocks used by the users are converted to the associated integer number, so all the numbers are saved as an *IntArrayList userDigit*, which here I will call `al` for simplicity.

If the size of `al` is equal to zero, this means that the user is about to add her/his first block to the program. The system then checks the first level of the tree and gives in output the block corresponding to the node that has the highest number of occurrences in that level.

Otherwise, the system starts crossing the tree. I will explain how the algorithm works by considering a generic position k in `al` .
So the system takes the k-th element of `al` and looks for a node in the k level of the tree with the same value in the `int brickId` parameter.
If it finds a node with this feature (I call it n^* for simplicity), then it looks at the value for the `children` attribute of n^*. It checks if any of them (we already know by construction that all these nodes belong to the k+1 level) corresponds to a node with the `int brickId` attribute equal to the($k+1$)-th element of `al`. If so, then this process is repeated considering that node that has just been found.
If this does not happen or if the system does not find a n^* node, then it is not able to suggest any help to the user.

Actually, the systems I implemented can be considered three and not only two: the one based on the bricks, the one based on the scripts and the one that merges the brick and/or script solution to limit the probability of having a "nothing" answer "to be returned to the user. The differences between the systems depend on the way the tree-structure has been filled.
In the next Section 4.3 and Section 4.4 I will explain how the two systems implemented individually and independently were constructed. To clarify the explanation, I take as an example these four array lists (Figure 4.1) : each one represents the id of a program with all block ids within the application. The scripts are the iwhole number written in bold, the others are the bricks and are attached to the related script.

30 = [**99**, {85}, **99**, {1}] ;
31 = [**35**, {85}] ;
32 = [**99**, {2}, **99**] ;
33 = [**99**, {85, 71}] ;

Figure 4.1: Toy example to explain tree-structure construction without end nodes.

30 = [**99**, {85}, **99**, {1}, -2] ;
31 = [**35**, {85}, -2] ;
32 = [**99**, {2}, **99,** -2] ;
33 = [**99**, {85, 71}, -2] ;

Figure 4.2: Toy example to explain tree-structure construction with end nodes.

At the end of each array list I added a fake number to indicate that the list of brick is finished: in this the way systems could also suggest the user not to add any other block to her/his program. So the arrays now become as showed in Figure 4.2.

4.3 Bricks-based system

This system builds the tree-structure with the sequences composed of block IDs for each application. Suppose I want to build the tree with these (Figure 4.2) lists of brick. Then I start from the first array: I create level 0 and I add to it a new node *n_first*. Then I set its attributes: nodeId=0, brickId=99, occurrence=1, children=new IntArrayList, parent=-1 (because it has no parent) and level=0. Then I go on with the other elements 85, 99, 1 and -2. Every time I add a node which is not in level 0, I have to update the attribute "children" of its father (adding the nodeId of the node I added) and I have to set the new node's attribute "parent" with its father's nodeId.
Now I consider the second application: I take the first element and check if it is already present in the structure (I did this check also previously, but I didn't mentioned them to make the explanation not too complicate). That integer number is not present, so I repeat the same procedure as before.
Now I consider the third array list. In the structure at level 0 I already have a node (which I call *n_found*) whose brickId is 95: so I update the value of "occurrence" setting it equal to two. Then I go ahead with the next value *temp* in the array list: first I check if one of *n_found*'s child has *temp* as "brickId" value: in this case I can not find a node with these characteristics, so I have to create a new one. Then I continue following the same procedure as before.
In the end I created this tree-structure:

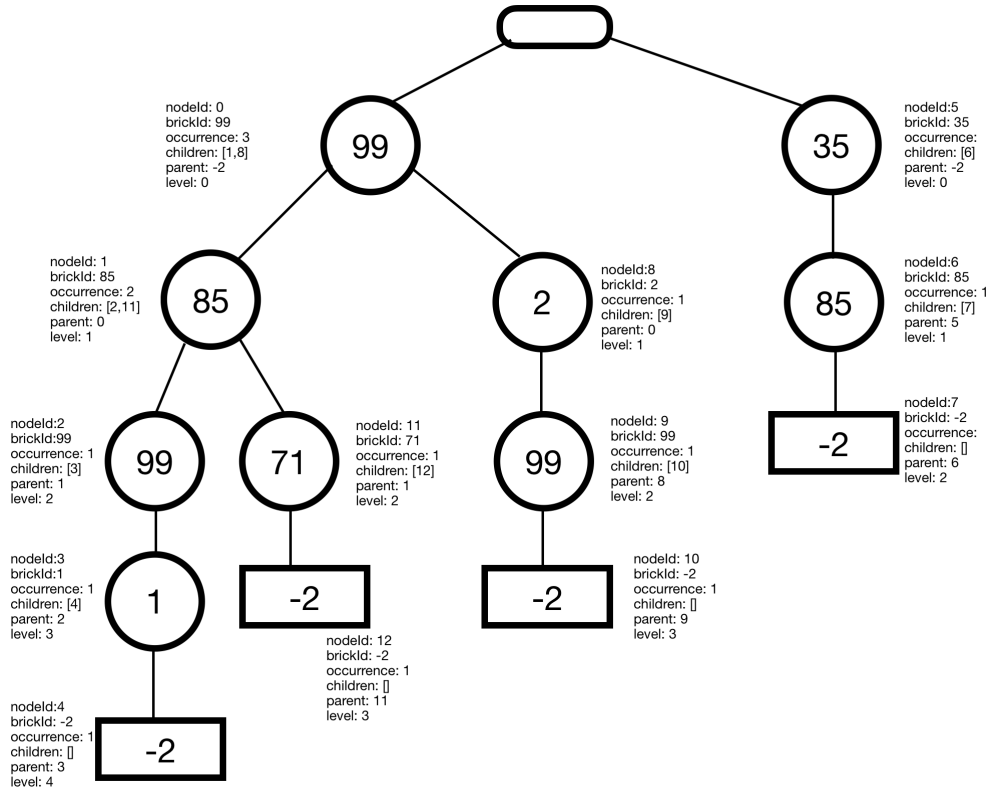


Figure 4.3: Bricks system structure.

4.4 Scripts-based system

The system based on the scripts is filled by the sequences obtained by combining the id of each script and the sequence of bricks within the corresponding script.
Now let's build the tree-structure for the toy example in Figure 4.2.
I started from the first application: it consists of two scripts. I considered the first script and I added the node corresponding to the id of the scripts to the tree, so I added one new node for each brick. Then I focused on the second script and I repeated the same procedure I followed in the construction of the system based on bricks.

Although I used some toy examples to explain how the tree structure is constructed, I can immediately underline some considerations on the results I have just obtained.
The tree-structure of the brick-based system has more levels than the one obtained from the other system: this is due to the fact that the first system

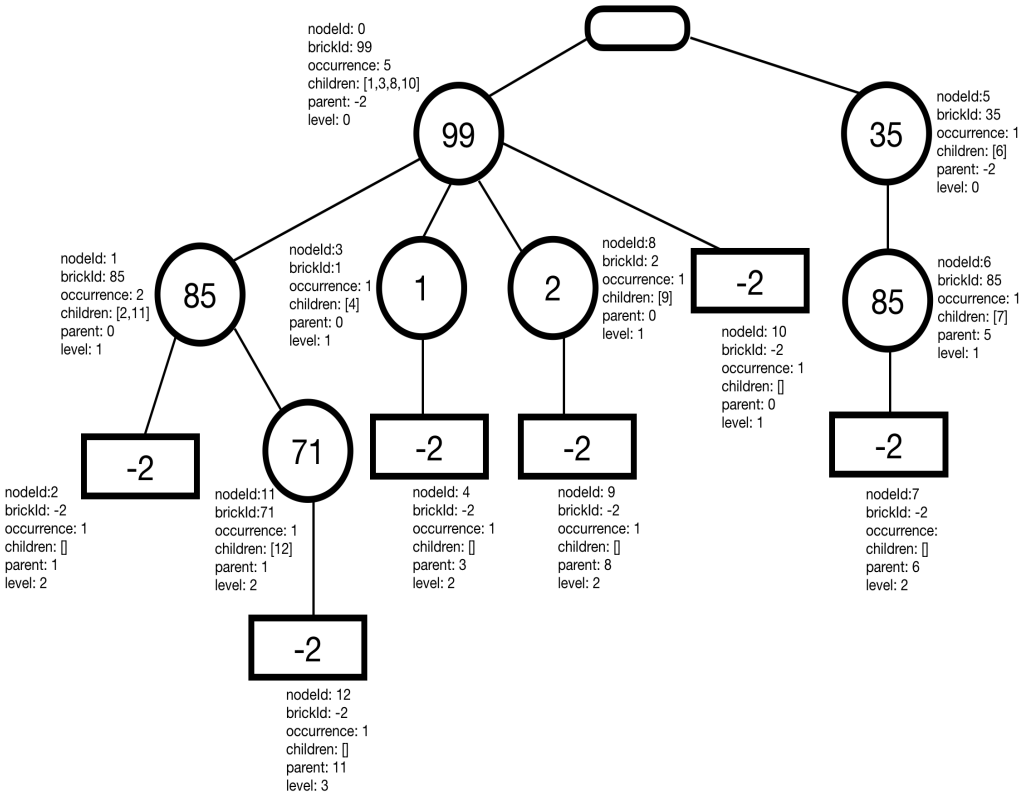


Figure 4.4: Scripts system structure.

keeps track of the entire sequence of blocks used in the application, while the system based on scripts divides the long sequence of id in shorter lists of integer numbers.

Another important difference between the two systems is the number of tree nodes. In this example it is not so immediate to understand, but in general in the tree-structure obtained from the system based on scripts, the total number of nodes is lower then the other system. The reason is again related to the compression performed by the second system of the length of the block sequences to be tracked.

I can point out another feature that will be examined in detail later: the system based on bricks is sensitive to the position of each block in the list of total elements used by the user. This means that two applications that differ only in one block or that have only two blocks inverted, will cover a different path in the tree-structure. Instead, the system based on scripts will almost

always consider sequences of blocks shorter than the bricks system, therefore if will suffer less from this problem.

On the other hand, the system based on bricks could give a more accurate suggestion: this is due to the fact that it is more "contextualised" in the developing of the programs, because it keeps track of a longer sequence of blocks.

As you can see, there are some advantages and disadvantages of these systems. For this reason, I implemented a sort of fusion of these two taking the positive aspects of both.
Before explaining how this fusion was made, let's examine how the two systems work separately.

4.5 Execution of bricks- and scripts-based systems

I wrote a Java program that uses the dataset obtained in the analysis phase to fill the two tree-structures. So I downloaded new data from Pocket Code's web site following the same procedure explained in Section 3.1.
To make the explanation clearer, I call *training set* the first corpus of data and *validation set* the second.

The validation set was used to verify the performances of the script-based system, the brick-based system, and possible merging of the two systems.

4.5.1 Merging of the two systems

First of all we need to specify what we mean by merging the two systems. From the data concerning the evaluation of the systems taken individually, it has been seen that the brick-based system was more effective than the one based on the scripts. The success rates were still low, so I wanted to make the two systems work together. If the first system (the one based on the brick) was not able to give any suggestion, then the system based on the scripts would have been used: this is what in the Table 4.1 is called "BRICK, ELSE SCRIPT".
As shown in Table 4.1, in the first line I obtained that system based on bricks gives a correct suggestion 28.328% of the time. But if the brick-system

fails and I consider the script-system, then I can give the correct suggestion 43.328% of the time.

4.5.2 Validation of systems: the most likely block

To simulate the user's development process of the program, I randomly collected data from this set and then I checked if the systems were able to suggest the correct block. I will call the simulated list of blocks *user_digit*

First I randomly chose an integer number inside the set of all programs identifiers (*first_int*). So I randomly chose a script id within the sequence of scripts contained in the selected program (*second_int*). Then I randomly chose another integer number in the range from 0 to the size of the brick sequence decremented by 1 within that specific script (*third_int*).

The way I built the sequence of blocks given as input was different in the two systems.
For the bricks-based system, I kept track of all the blocks used by the user: so the *user_digit* was calculated as the concatenation of all the IDs of the selected application block up to (*third_int*).
The system based on scripts tracks only the bricks within each script: so the *user_digit* was calculated taking the blocks within the (*second_int*) script up to (*third_int*).

Then I tested if the suggestions calculated by the two systems were correct: I let the systems cross the respective trees in order to find the most likely block. Then I made a comparison between the id of the suggested blocks and the right one. The results are shown in Table 4.1.

4.5.3 Validation of systems: the three most likely blocks

Instead of looking at the first suggestion that the systems can give, I can consider the first three suggestions, which I call in the table *3BB* (three-best-bricks) and *3BS* (three-best-scripts). In the table you can find the scores I got using the two systems individually and then merging them together.
The best score I get is when the systems give the user the first three best suggestions of the brick-systems and, if no one is correct, it gives the first three best suggestions coming from the other system. Doing so, 56.882% of the time I give the correct suggestion to the user.

System used	***Score (percentage)***
BRICK	28.328%
SCRIPT	27.732%
3B BRICKS	32.006%
3B SCRIPTS	48.798%
BRICK, ELSE SCRIPT	43.328%
3B BRICKS, ELSE SCRIPT	46.732%
3B BRICKS, ELSE 3B SCRIPTS	56.882%

Table 4.1: Results over 50000 tests.

4.6 Trigram-based system

Given the fact that the system based on scripts and the one based on bricks don't perform very well if taken individually, I tried to find better solutions. So I implemented a system based on another kind of models: I chose *Language Models* and in particular the *n-grams* with n=3.

4.6.1 Language Models with N-grams

First we define V as a finite set of words which belong to a specific language [25].
A *sentence* in the language is a sequence of words $\{x_1,x_2,\ldots,x_n\}$, where $n \geq 1$ and $x_i \in V$ for $i \in \{1,2,\ldots,n\text{-}1\}$.
Then we define $V\dagger$ as the set of all sentences with the vocabulary V: this is an infinite set, because sentences can be of any length.

Now we can define what a Language Model is. A *Language Model* consists of a finite set V, and a function $p(x_1,x_2,\ldots,x_n)$ such that:

1. For any $\langle\ x_1,x_2,\ldots,x_n\ \rangle \in V\dagger$, $p\ (x_1,\ x_2,\ldots,x_n) \geq 0$
2. $$\sum_{\langle x_1,\ldots,x_n\rangle \in V\dagger} p(x_1, x_2, \ldots, x_n) = 1$$

Hence $p(x_1,x_2,\ldots,x_n)$ is a probability distribution over the sentences in $V\dagger$.
Language Models are very useful in a broad range of applications, the most

obvious perhaps being speech recognition and machine translation. In many applications it is very useful to have a good "prior" distribution p(x_1,x_2,...,x_n) over which sentences are or aren't probable in a language.
To estimate the probability p, we usually use *Markov Models*.

Markov Model

Consider a sequence of random variables, X_1,X_2...,X_n.
Each random variable can take any value in a finite set V. For now we will assume that the length of the sequence, n, is some fixed number.
Our aim is to to model the probability of any sequence x_1,x_2,...,x_n, where $n \geq 1$ and $x_i \in V$ for $i \in \{1,2,...,n\}$, that is, to model the joint probability P(X_1 = x_1,X_2 = x_2,...,X_n = x_n).

The Markov Model makes some assumptions.

1. *First-order Markov assumption*: it says that the identity of the i-th word in the sequence depends only on the identity of the previous word, x_{i-1}. More formally, we have assumed that the value of X_i is conditionally independent of X_1,X_2,...,X_{i-1}, given the value for X_{i-1}.

2. *Second-order Markov assumption*: it says that each word depends on the previous two words in the sequence.

The second assumption form the basis of Trigram Language Models.

Trigram Language Model

As I wrote in Section 4.6.1 we model each sentence as a sequence of n random variables,X_1,X_2...,X_n.
A Trigram Language Model consists of a finite set V, and a parameter *q(w | u, v)* for each trigram u,v,w.
The value for *q(w | u, v)* can be interpreted as the probability of seeing the word w immediately after the bigram *(u, v)*. For any sentence x_1, x_2,...,x_n where $x_i \in V$ for $i \in \{1,2,...,n\text{-}1\}$, and the probability of the sentence under the Trigram Language Model is p(x_1,x_2,...,x_n) = q(x_i | x_{i-2},x_{i-1}) .

4.6.2 Building the *tree-structure*

To build the tree-structure, I consider again the *toy example* in Figure 4.2 as I did for the brick-based tree and for the script-based tree systems.
The tree-structure in Figure 4.5 is the one obtained from this trigram-based system.

I considered all the sub-sequences of three consecutive blocks in each sequence of bricks and scripts for each application. As you can see, the tree has depth equal to three. Whenever the system has to suggest the *next* block to the user, the system follows a sequence of three steps:

1. First, it considers the last two blocks used by the user.

2. It traverses the tree.

3. When it reaches the node corresponding to the last block adopted by the user, it suggests the most probable node, which is the node of its children with the maximum value for the field *occurrence*.

I used the same training and validation sets as for the previous systems based on bricks and scripts to train and to evaluate this new system.
Instead of focusing only on the most probable block, I considered again the three most probable blocks.
The results of the tests are shown in Table 4.2.

In order to improve the performance of the system, I tried to include in the trigram system a sort of *smoothing* to limit the negative effect of *null response* from the system.

4.6.3 Smoothing for n-grams

The key idea is to rely on lower-order statistical estimates to "smooth" the estimates based on trigrams.
The name smoothing [23] comes from the fact that these techniques tend to make distributions more uniform by adjusting low probabilities such as zero probabilities upward and high probabilities downward.
Furthermore, they attempt to improve the accuracy of the Model as a whole.

System used	***Score (percentage)***
Brick	28.227%
Script	28.167%
3B Bricks	31.823%
3B Scripts	49.185%
Brick, else Script	42.963%
3B Bricks, else Script	46.558%
3B Bricks, else 3B Scripts	56.132%
Trigram	30.550%
3B Trigram	54.162%
3B Brick, else 3B Trigram	53.873%
3B Bricks, else 3B Scripts, else 3B Trigram	67.178 %
3B Bigram	38.761%
3B Trigram, else 3B Bigram	54.326%
Smoothing	51.197%

Table 4.2: Results over 100000 tests.

Jelinek Mercer Smoothing

This kind of smoothing interpolates the Trigram Model with a Bigram Model. In Bigram Models, we make the approximation that the probability of a word depends only on the identity of the immediately preceding word.
Then this method interpolates linearly a Trigram Model and a Bigram Model as follows:

$$p_{interp}(w_i \mid x_{i-2}) = \lambda\ p_{ML}(x_i \mid x_{i-2}) + (1\text{-}\lambda)\ p_{ML}(x_i \mid x_{i-1}).$$

I built also the tree structure for the Bigram Model in order to use this kind of smoothing.
I followed the same strategies adopted during the building process of the other tree-structures. I considered the *three-most-probable* blocks suggested from both systems and the I used the J-M smoothing algorithm to get the score for each block.

In Table 4.2 you can find the results of the tests for the system based on trigram and for the one based on trigram plus the J-M smoothing algorithm.

In Chapter 5 I will explain how I integrated the *best* suggestion system into the Pocket Code application.

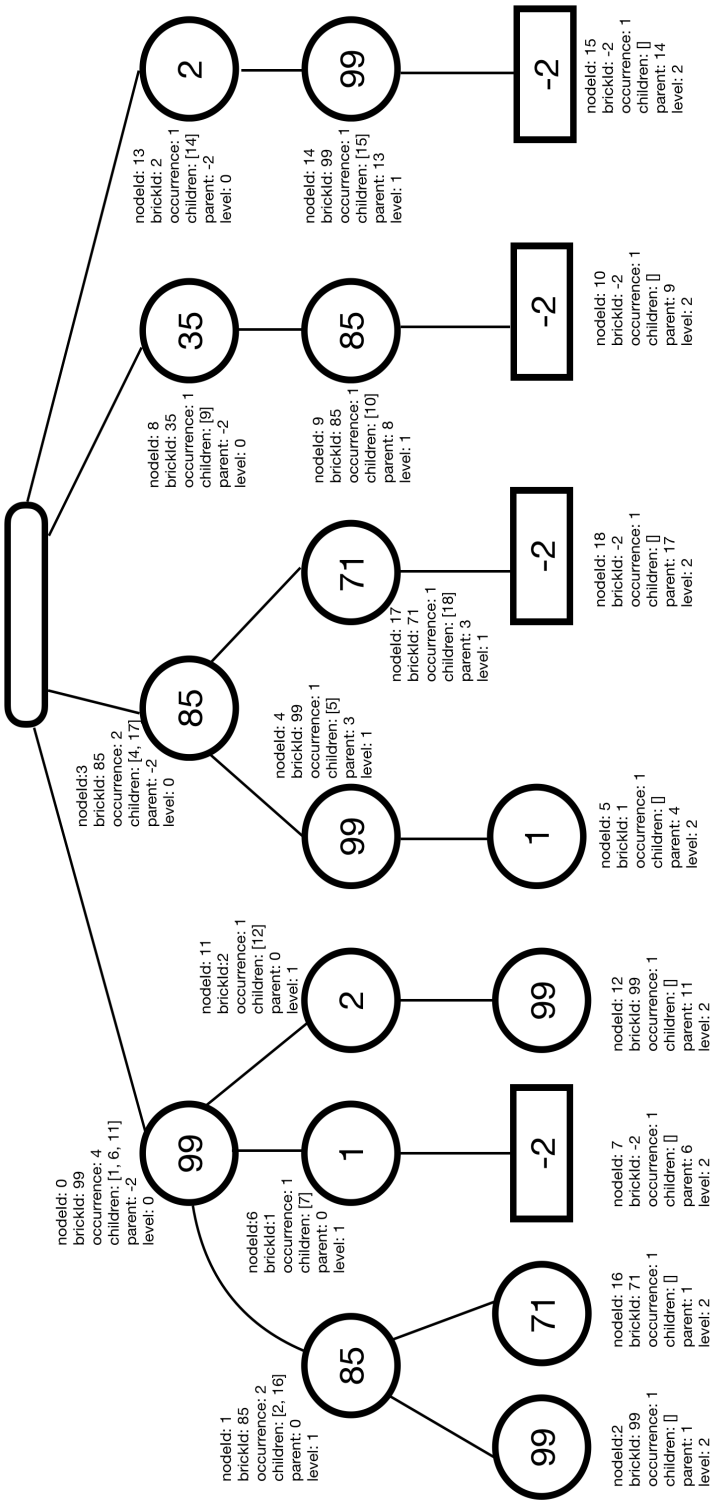


Figure 4.5: N-gram system structure.

Chapter 5

Integration into Pocket Code

Of all the code completion systems explained and implemented in Chapter 4, I chose to integrate into Pocket Code the system based on 3-grams essentially. There are essentially two reasons behind my choice:

1. It is the one that, taken individually, most often gives the *correct* suggestion to the user.

2. The file in which the corresponding tree structure is stored is not as large and can be easily stored in a mobile device.

In this chapter I will quickly explain how the Pocket Code source classes are organized and then how I integrated the suggestion system into the app.

5.1 Structure of Pocket Code's source code

The entire source code of Pocket Code is available at the URL `https://github.com/Catrobat/Catroid`.
In this section I will briefly show the classes on which I worked for the integration of the code completion system.
The classes in the `Catroid/catroid/src/main/java/org/ catrobat/catroid/` folder are structured in some sub-folders: *bluetooth*, *camera*, *cast*, *common*, *content*, *devices*, *drone*, *exceptions*, *facedetection*, *formulaeditor*, *io*, *merge*, *nfc*, *physics*, *pocketmusic*, *scratchconverter*, *sensing*, *soundrecorder*, *stage*, *transfers*, *ui*, *utils* and *web*.
The sub-folders which I analysed and modified are *content* and *ui*.

5.1.1 Content

This folder contains all the classes used to manage the functionality of the blocks. Here you can find a class for each kind of block, a class for each category of blocks and some classes used to give some functionality to the blocks enabling some actions on them.
In particular, in the *Sprite.java* Java class you can get all the blocks used by the user in his program.
There are also all the classes that explain the basic characteristics of a brick and of a script: the *Brick* and *ScriptBrick* interfaces, all the classes that implement these interfaces and all the extensions of these classes. A thorough analysis of this folder can be useful to get a more precise understanding of the key concepts on bricks and scripts.

5.1.2 UI

As the name suggests, this folder contains the classes which describe the Pocket Code user interface, which is how the app is displayed to the user.
In particular there is a sub-folder, *fragment*, which describes the details of each individual fragment of the app. The Java class *CategoryBricksFragment.java* is used to show the blocks for each category and show them to the user to allow them to decide which block to use for their program. It arranges all the blocks in a list when the user tap on the button of the corresponding category.
The are many other classes that describe how the fragment of each category should appear to the user.
There is another important sub-folder, *adapter*, which has the function of allowing the user to correctly drag the selected blocks and insert them into his program. It organises all the blocks selected by the user in the block list of the program and it allows the user to select the position in which to insert the block.
So the user can decide to remove some blocks or change their position.
The sub-folder *dialogs* manages all the pop-up messages shown to the user to interact with him and to give him a feedback of the operations he is doing while creating the project.

5.2 3-gram suggestion system integration

In order to integrate the code completion system in the app, I decided to add a new category of blocks which I called *Suggestion*.
First of all, I added a new TextView in the list of categories that appears when the user wants to add a new block to his application.

This TextView is places on the top of the categories' list and is coloured by grey (see Figure 5.1). To define this new TextView, I followed the sequent

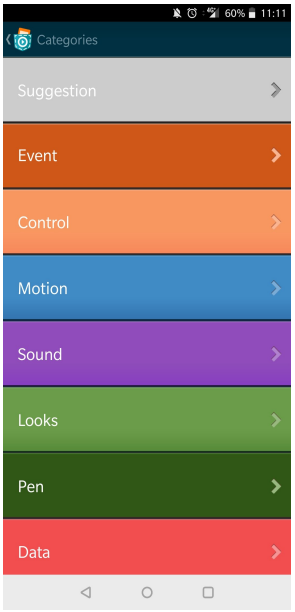


Figure 5.1: Suggestion's category in the categories' list.

steps.

1. In the folder `src/res/layout` I added the xml file *brick_ category_ suggestion.xml* in order to define a new linear layout in which there was a TextView for the category *Suggestion*.

2. The style of the TextView was declared in the `src/res/values` folder, where I added some code's lines to the file *styles.xml* (as you can see in Figure 5.2).

3. In the `src/res/drawable` folder I added the corresponding files *.png* to define the colour of the TextView for this category.

```
<style name="BrickCategoryText.Suggestion" >
    <item name="android:background" >@drawable/brick_selection_background_suggestion</item >
    <item name="android:drawableRight" >@drawable/main_menu_button_arrow_suggestion</item >
</style >
```

Figure 5.2: Suggestion's category: *style.xml*.

In the folder *ui* I created a new sub-folder which I called *suggestion*. Here I put all the Java classes for the code completion system: MyNode.java, Level.java, ReadFromTreeTxt.java, ConverterFromClassToId.java, NGramSuggestion.java and SystemComparrison.java.

5.2.1 MyNode

An instance of this class represents a node of the underlying tree-structure. Each node is uniquely identified by an integer number. It has other attributes, as the id of the block associated to it, the number of occurrences, the id of the father node and the list of children nodes' identifiers.

5.2.2 Level

This class has the function to represent each level of the tree-structure. It contains the list of all nodes at that specific level. It has also some specific methods, as the search for a brick or a script given the id of the node or the id of the block.

5.2.3 ReadFromTreeTxt

This class is fundamental for the code completion system, because an instance of this class has to read the `.txt` file containing the tree-structure. Its key function is to create the structure to be crossed to give the suggestion to the user.

5.2.4 ConverterFromClassToId

I implemented this class to have a faster correlation and translation from the class instance associated to a block and the corresponding id used to

identify that specific class. Given the list of blocks' instances, a *ConverterFromClassToId c* gives in output a list of integer representing the id of corresponding bricks and scripts.

5.2.5 NGramSuggestion

This is the focus of the system. It creates an instance of the class ReadFromFile to navigate through the tree-structure and return the list of possible suggestions to the user.

Tree-structure file

The file containing the data of the tree-structure is `tree.txt` and it is stored in the `/assets/` folder.

5.3 How it works

During the development phase of his program, the user can ask the system to help him in choosing the next block to use.
The user simply needs to tap the "Suggestion" button and he will receive the suggestion from the system.
As I explain in Chapter 4, the system based on the n-gram takes the last two blocks applied by the user to help him. If there are not blocks or if there is only one block, the system suggests the three most probable nodes in the first level of the tree structure respectively or considers only that block used as input.
In Figure 5.3 is shown how the list appears to the user.
The system can also suggest the user to stop and not add any other blocks. In this case it appears a *toast*, that is a message that appears on the screen and disappears after a time-out, which indicates to the user that he can stop.
If the system can not find any suggestions, a *toast* will appear telling the user that the system can not help him.

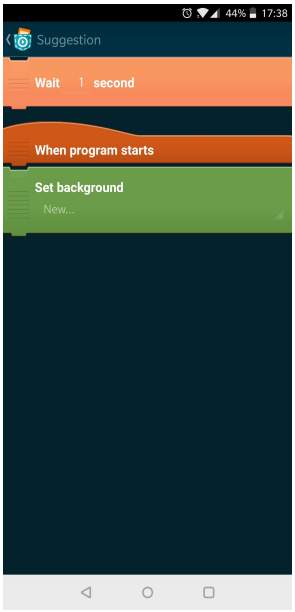


Figure 5.3: List of suggested blocks.

Chapter 6

Conclusions and future work

The aim of my work was the creation of a code completion system for the Catrobat visual programming language.
Catrobat is adopted in Pocket Code, a popular open-source application for Android devices designed to teach the principles of coding to children and non-experts. In Catrobat – hence, in Pocket Code – users select some coloured blocks and drag them into the application they are developing. Each block is associated with a program statement and has a specific function. Applications developed with Pocket Code can be published in the Pocket Code website.

During my work, I downloaded the source code of all the Pocket Code applications published up to August 23 2017, I analysed it, and I used statistics gathered from the code to design a block suggestion system that presents the user with the most probable blocks she/he would be using in the future development of the application given the blocks she/he has already used.
The blocks used in Catrobat are divided into bricks and scripts: I designed different code completion systems based on both bricks and scripts, then I benchmarked them to choose the most effective one.
A first system I developed is based on sequences of all blocks adopted by the user.
A second system considers the sequences of bricks used in each script.
A third system is based on 3-grams of blocks. In my tests, I also considered some combinations of such systems, in order to measure how the effectiveness of the completion system as a whole could be improved.

All the systems were trained on the whole set of downloaded applications. Performance measurements were performed on applications developed after August 23 2017, that is, on applications that had not been used in the

training phase. The performance metric was the number of times the correct block was presented among the top 3 suggestions of a given system. The most effective system turned out to be the one based on 3-grams: this system presents a good trade-off between performance and storage space needed by the data structure used by the code completion system. Out of a total of 100,000 tests performed, this system is 54.162% effective.

Finally, I integrated the most effective system into the Pocket Code codebase.

6.1 Future work

In my opinion, many features can be added into Pocket Code to improve the code completion system.
For instance, we could consider other statistical algorithms or other smoothing systems different from those considered in this work.
The user could also be asked to add additional information when she/he starts a new application project. In addition to the name of the program, a brief description, et cetera, the user could specify the category of her/his program choosing it from a finite set of possibilities. Such additional information could be used in the training phase of the system to improve the overall performance and to make the system more effective.
Moreover, once the code completion system knows the specific aim of the user, it can suggest her/him not only the most probable block or blocks: it can suggest some pre-defined sets of blocks implementing frequently-used features. This could be definitely helpful for the user.
It is also conceivable to integrate into Pocket Code a system which interacts with the user by asking, from time to time, questions related to the design process: the answers could be used by the system to provide more targeted suggestions.

Appendices

Appendix A

Table with the various blocks analysed

.

Table A.1: Table of all blocks (part 1).

ID	JAVA CLASS NAME	VISUALISED BLOCK NAME	BLOCK SHAPE	BLOCK COLOR	BLOCK CATEGORY
0	AddItemToUserListBrick	Add_to list	rectangular	red	Data
1	ArduinoSendDigitalValueBrick	Set Arduino digital pin_ to _	rectangular	teal blue	Arduino
2	ArduinoSendPWMValueBrick	Set Arduino PWM pin _ to_	rectangular	teal blue	Arduino
3	AskBrick	Ask _ and store written answer in_	rectangular	light green	Looks
4	AskSpeechBrick	Ask _ and store spoken answer in _	rectangular	violet	Sound
5	BroadcastBrick	Broadcast	rectangular	dark orange	Event
6	BroadcastReceiverBrick	When you receive	round	dark orange	Event
7	BroadcastWaitBrick	Broadcast and wait	rectangular	dark orange	Event
8	CameraBrick	Turn camera	rectangular	light green	Looks
9	ChangeBrightnessByNBrick	Change brightness by _	rectangular	light green	Looks
10	ChangeColorByNBrick	Change color by _	rectangular	light green	Looks

(Continues in the next page)

(Continues from the previous page)

11	ChangeSizeByNBrick	Change size by _	rectangular	light green	Looks
12	ChangeTransparencyByNBrick	Change transparency by _	rectangular	light green	Looks
13	ChangeVariableBrick	Change variable _ by _	rectangular	red	Data
14	ChangeVolumeByNBrick	Change volume by _	rectangular	violet	Sound
15	ChangeXByNBrick	Change X by _	rectangular	blue	Motion
16	ChangeYByNBrick	Change Y by _	rectangular	blue	Motion
17	ChooseCameraBrick	Use camera	rectangular	light green	Looks
18	ClearBackgroundBrick	Clear	rectangular	dark green	Data
19	ClearGraphicEffectBrick	Clear graphic effects	rectangular	light green	Looks
20	CloneBrick	Create clone of _	rectangular	light orange	Control
21	CollisionScript				
22	ComeToFrontBrick	Go to front	rectangular	blue	Motion
23	DeleteItemOfUserListBrick	Delete item from list _ at position_	rectangular	red	Data
24	DeleteThisCloneBrick	Delete this clone	rectangular	light orange	Control
25	FlashBrick	Turn flashlight	rectangular	light green	Looks

(Continues in the next page)

(Continues from the previous page)

26	ForeverBrick	Forever	rectangular	light orange	Control
27	GlideToBrick	Glide _ to X:_ Y:_	rectangular	blue	Motion
28	GoNStepsBackBrick	Go back	rectangular	blue	Motion
29	GoToBrick	Go to	rectangular	blue	Motion
30	HideBrick	Hide	rectangular	light green	Looks
31	HideTextBrick	Hide variable	rectangular	red	Data
32	IfLogicBeginBrick	If _ is true then ... Else ...	rectangular	light orange	Control
33	IfLogicElseBrick	Else	rectangular	light orange	Control
34	IfLogicEndBrick	End if	rectangular	light orange	Control
35	IfOnEdgeBounceBrick	If on edge, bounce	rectangular	blue	Motion
36	IfThenLogicBeginBrick				
37	IfThenLogicEndBrick	End if	rectangular	light orange	Control

(Continues in the next page)

(Continues from the previous page)

38	InsertItemIntoUserListBrick	Insert _ into list _ at position _	rectangular	red	Data
39	LegoEv3MotorMoveBrick	Set EV3 motor _ to _% speed	rectangular	yellow	Data
40	LegoEv3MotorStopBrick	Stop EV3 motor	rectangular	yellow	Lego EV3
41	LegoEv3PlayToneBrick	Play EV3 tone for _ seconds Frequency _ x100Hz Volume _%	rectangular	yellow	Lego EV3
42	LegoEv3SetLedBrick	Set EV3 LED status	rectangular	yellow	Lego EV3
43	LegoNxtMotorMoveBrick	Set NXT motor _ to _% speed	rectangular	yellow	Lego EV3
44	LegoNxtMotorStopBrick	Stop NXT motor	rectangular	yellow	Lego EV3
45	LegoNxtMotorTurnAngleBrick	Turn NXT motor _ by _	rectangular	yellow	Lego EV3
46	LegoNxtPlayToneBrick	Play NXT tone Duration _ seconds Frequency _x100Hz	rectangular	yellow	Lego EV3
47	LoopEndBrick	End of loop	rectangular	light orange	Control
48	LoopEndlessBrick	End of loop	rectangular	light orange	Control
49	MoveNStepsBrick	Move	rectangular	blue	Motion

(Continues in the next page)

(Continues from the previous page)

50	NextLookBrick	NextLook	rectangular	light green	Looks
51	NoteBrick	Note	rectangular	light orange	Control
52	PenDownBrick	Pen down	rectangular	dark green	Pen
53	PenUpBrick	Pen up	rectangular	dark green	Pen
54	PhiroMotorMoveBackwardBrick	Move Phiro motor backward _ Speed _%	rectangular	teal blue	Phiro
55	PhiroMotorMoveForwardBrick	Move Phiro motor forward _ Speed _%	rectangular	teal blue	Phiro
56	PhiroMotorStopBrick	Stop Phiro motor	rectangular	teal blue	Phiro
57	PhiroPlayToneBrick	Play Phiro music tone _ Duration _ seconds	rectangular	violet	Sound
58	PhiroRGBLightBrick	Set Phiro light _ Red _ Green _ Blue _	rectangular	light green	Looks
59	PlaceAtBrick	Place at X:_ Y:_	rectangular	blue	Motion
60	PlaySoundAndWaitBrick	Start sound and wait	rectangular	violet	Sound
61	PlaySoundBrick	Start sound	rectangular	violet	Sound
62	PointInDirectionBrick	Point in direction _ degrees	rectangular	blue	Motion

(Continues in the next page)

(Continues from the previous page)

63	PointToBrick	Point towards	rectangular	blue	Motion
64	PreviousLookBrick	Previous look	rectangular	light green	Looks
65	RaspiInterruptScript				
66	RaspiSendDigitalValueBrick	Set Raspberry Pi pin _ to _	rectangular	teal green	Raspberry Pi
67	RepeatBrick	Repeat	rectangular	light orange	Control
68	RepeatUntilBrick	Repeat until _ is true	rectangular	light orange	Control
69	ReplaceItemInUserListBrick	Replace item in list _ at position _ with _	rectangular	red	Data
70	SayBubbleBrick	Say	rectangular	light green	Looks
71	SayForBubbleBrick	Say _ for _ seconds	rectangular	light green	Looks
72	SceneStartBrick	Start scene	rectangular	light orange	Control
73	SceneTransitionBrick	Continue scene	rectangular	light orange	Control
74	SetBackgroundAndWaitBrick	Switch to look _ and wait	rectangular	light green	Looks
75	SetBackgroundBrick	Switch to look _	rectangular	light green	Looks

(Continues in the next page)

(Continues from the previous page)

76	SetBounceBrick	Set bounce factor to _%	rectangular	blue	Motion
77	SetBrightnessBrick	Set brightness to _%	rectangular	light green	Looks
78	SetColorBrick	Set color to _	rectangular	light green	Looks
79	SetFrictionBrick	Set friction to _%	rectangular	blue	Motion
80	SetGravityBrick	Set gravity for all objects to X:_ Y:_ steps/second	rectangular	blue	Motion
81	SetLookBrick	Switch to look _ and wait	rectangular	light green	Looks
82	SetMassBrick	Set mass to _ kilogram	rectangular	blue	Motion
83	SetPenColorBrick	Set pen color to Red_ Green_ Blue_	rectangular	dark green	Pen
84	SetPenSizeBrick	Set pen size to_	rectangular	dark green	Pen
85	SetPhysicsObjectTypeBrick	Set motion type to _	rectangular	blue	Motion
86	SetRotationStyleBrick	Set rotation style	rectangular	blue	Motion
87	SetSizeToBrick	Set size to _%	rectangular	light green	Looks
88	SetTransparencyBrick	Set transparency to _%	rectangular	light green	Looks
89	SetVariableBrick	Set variable _ to _	rectangular	red	Data
90	SetVelocityBrick	Set velocity to X:_ Y:_ steps/second	rectangular	blue	Motion

(Continues in the next page)

(Continues from the previous page)

91	SetVolumeToBrick	Set volume to _%	rectangular	violet	Sound
92	SetXBrick	Set X to _	rectangular	blue	Motion
93	SetYBrick	Set Y to _	rectangular	blue	Motion
94	ShowBrick	Show	rectangular	light green	Looks
95	ShowTextBrick	Show variable _ at X:_ Y:_	rectangular	red	Data
96	SpeakAndWaitBrick	Speak_ and wait	rectangular	violet	Sound
97	SpeakBrick	Speak _	rectangular	violet	Sound
98	StampBrick	Stamp	rectangular	dark green	Pen
99	StartScript				
100	StopAllSoundsBrick	Stop all sounds	rectangular	violet	Sound
101	StopScriptBrick	Stop	rectangular	light orange	Control
102	ThinkBubbleBrick	Think _	rectangular	light green	Looks
103	ThinkForBubbleBrick	Think _ for _ seconds	rectangular	light green	Looks
104	TurnLeftBrick	Turn left _ degrees	rectangular	blue	Motion
105	TurnLeftSpeedBrick	Rotate left _ degrees/second	rectangular	blue	Motion
106	TurnRightBrick	Turn right _ degrees	rectangular	blue	Motion

(Continues in the next page)

(Continues from the previous page)

107	TurnRightSpeedBrick	Rotate right _ degrees/second	rectangular	blue	Motion
108	UserBrickUserBrick		rectangular	bright green	My Bricks
109	VibrationBrick	Vibrate for _	rectangular	blue	Motion
110	WaitBrick	Wait	rectangular	light orange	Control
111	WaitUntilBrick	Wait until _ is true	rectangular	light orange	Control
112	WhenBackgroundChangesScript	When background changes to _	round	dark orange	Event
113	WhenClonedScript				
114	WhenConditionScript				
115	WhenNfcScript				
116	WhenScript				
117	WhenTouchDownScript				
118	PhiroIfLogicBeginBrick	If Phiro _ is activated	rectangular	light orange	Control

(Continues in the next page)

(Continues from the previous page)

119	RaspiIfLogicBeginBrick	If Raspberry Pi pin _ is true then	rectangular	light orange	Control
120	RaspiPwmBrick	SetRaspberry Pi PWM pin _ to _% _Hz	rectangular	teal blue	Raspberry Pi
121		Turn camera	rectangular	light green	Looks
122	WhenBrick	When tapped	round	dark orange	Event
123					
124					
125					
126	SetNfcTagBrick	Set next NFC tag to _ as NDEF record type	rectangular	light orange	Control
127	WhenGamepadButtonScript				
128	DroneBasicBrick	__	rectangular	blue	
129	DroneBasicControlBrick	__	rectangular	light orange	
130	DroneBasicLookBrick	__	rectangular	light green	
131	DroneMoveBrick	_ with _ power	rectangular	blue	

(Continues in the next page)

(Continues from the previous page)

132	DroneSpinnerBrick	__	rectangular	light orange	
133	LegoEv3MotorTurnAngleBrick	Turn EV3 motor _ by _	rectangular	yellow	Lego EV3
134	WhenClonedBrick	When you start as a clone	round	light orange	Control
135	WhenConditionBrick	When _ becomes true	round	light orange	Control
136	WhenGamepadButtonBrick	When gamed button _ tapped	round	light orange	Control
137	SetLookByIndexBrickAndWait	Switch to look with number _ and wait	rectangular	light green	Looks
138	SetLookByIndexBrick	Switch to look with number _	rectangular	light green	Looks
139	WhenNfcBrick	When NFC	round	dark orange	Event
140	SetTextBrick	Text_ X:_ Y:_	rectangular	blue	Motion
141	UserScriptDefinitionBrick		round	bright green	My Bricks
142	WhenRaspiPinChangedBrick	When Raspberry Pi pin _ changes to _	round	light orange	Control

(Continues in the next page)

(Continues from the previous page)

143	WhenStartedBrick	When program starts	round	dark orange	Event
144	WhenTouchDownBrick	When screen is touched	round	dark orange	Event
145	BroadcastScript				
146	SetBackgroundByIndexAndWaitBrick	Set background to number _ and wait	rectangular	light green	Looks
147	SetBackgroundByIndexBrick	Set background to number _	rectangular	light green	Looks

Table A.1: Table of all blocks (part 1).

Table A.2: Table of all blocks (part 2).

ID	XML NAME	TYPE	JAVA CLASS INTERFACE	JAVA FATHER CLASS	OCCURRENCES IN DATASET
0	AddItemToUserListBrick	brick	null	UserListBrick	2270
1	ArduinoSendDigitalValueBrick	brick	null	FormulaBrick	94
2	ArduinoSendPWMValueBrick	brick	null	FormulaBrick	661
3	AskBrick	brick	null	UserVariableBrick	3894
4	AskSpeechBrick	brick	null	UserVariableBrick	238
5	BroadcastBrick	brick	BroadcastMessage	BrickBaseType	123222
6	BroadcastReceiverBrick	script	ScriptBrick/ BroadcastMessage	BrickBaseType	0
7	BroadcastWaitBrick	brick	BroadcastMessage	BroadcastBrick	10414
8	CameraBrick	brick	null	BrickBaseType	1394
9	ChangeBrightnessByNBrick	brick	null	FormulaBrick	1802
10	ChangeColorByNBrick	brick	null	FormulaBrick	1865
11	ChangeSizeByNBrick	brick	null	FormulaBrick	9686
12	ChangeTransparencyByNBrick	brick	null	FormulaBrick	3766
13	ChangeVariableBrick	brick	null	UserVariableBrick	54667

(Continues in the next page)

(Continues from the previous page)

14	ChangeVolumeByNBrick	brick	null	FormulaBrick	367
15	ChangeXByNBrick	brick	null	FormulaBrick	12157
16	ChangeYByNBrick	brick	null	FormulaBrick	16300
17	ChooseCameraBrick	brick	null	BrickBaseType	1141
18	ClearBackgroundBrick	brick	null	BrickBaseType	622
19	ClearGraphicEffectBrick	brick	null	BrickBaseType	811
20	CloneBrick	brick	BrickWithSpriteReference	BrickBaseType	2773
21	CollisionScript	script	null	script	4361
22	ComeToFrontBrick	brick	null	BrickBaseType	14839
23	DeleteItemOfUserListBrick	brick	null	UserListBrick	1105
24	DeleteThisCloneBrick	brick	null	BrickBaseType	1351
25	FlashBrick	brick	null	BrickBaseType	3417
26	ForeverBrick	brick	LoopBeginBrick	BrickBaseType	89241
27	GlideToBrick	brick	null	FormulaBrick	80059
28	GoNStepsBackBrick	brick	null	FormulaBrick	5015
29	GoToBrick	brick	null	BrickBaseType	4453
30	HideBrick	brick	null	BrickBaseType	240010

(Continues in the next page)

(Continues from the previous page)

31	HideTextBrick	brick	null	UserVariableBrick	4942
32	IfLogicBeginBrick	brick	NestingBrick	FormulaBrick	169200
33	IfLogicElseBrick	brick	NestingBrick/ AllowedAfterDeadEndBrick	BrickBaseType	169214
34	IfLogicEndBrick	brick	NestingBrick/ AllowedAfterDeadEndBrick	BrickBaseType	169207
35	IfOnEdgeBounceBrick	brick	null	BrickBaseType	5891
36	IfThenLogicBeginBrick	brick	NestingBrick	IfLogicBeginBrick	37380
37	IfThenLogicEndBrick	brick	NestingBrick/ AllowedAfterDeadEndBrick	BrickBaseType	37389
38	InsertItemIntoUserListBrick	brick	null	UserListBrick	310
39	LegoEv3MotorMoveBrick	brick	null	FormulaBrick	48
40	LegoEv3MotorStopBrick	brick	OnItemSelectedListener	BrickBaseType	43
41	LegoEv3PlayToneBrick	brick	null	FormulaBrick	11
42	LegoEv3SetLedBrick	brick	OnItemSelectedListener	BrickBaseType	2
43	LegoNxtMotorMoveBrick	brick	null	FormulaBrick	197
44	LegoNxtMotorStopBrick	brick	OnItemSelectedListener	BrickBaseType	102
45	LegoNxtMotorTurnAngleBrick	brick	null	FormulaBrick	95

(Continues in the next page)

(Continues from the previous page)

46	LegoNxtPlayToneBrick	brick	null	FormulaBrick	32
47	LoopEndBrick	brick	NestingBrick/ AllowedAfterDeadEndBrick	BrickBaseType	22383
48	LoopEndlessBrick	brick	DeadAndBrick	LoopEndBrick	88617
49	MoveNStepsBrick	brick	null	FormulaBrick	6261
50	NextLookBrick	brick	null	BrickBaseType	28552
51	NoteBrick	brick	OnClickListener	FormulaBrick	17502
52	PenDownBrick	brick	null	BrickBaseType	811
53	PenUpBrick	brick	null	BrickBaseType	364
54	PhiroMotorMoveBackwardBrick	brick	null	FormulaBrick	97
55	PhiroMotorMoveForwardBrick	brick	null	FormulaBrick	199
56	PhiroMotorStopBrick	brick	OnItemSelectedListener	BrickBaseType	229
57	PhiroPlayToneBrick	brick	null	FormulaBrick	24
58	PhiroRGBLightBrick	brick	null	FormulaBrick	217
59	PlaceAtBrick	brick	null	FormulaBrick	172369
60	PlaySoundAndWaitBrick	brick	OnItemSelectedListener	BrickBaseType	3184
61	PlaySoundBrick	brick	OnItemSelectedListener	BrickBaseType	50724

(Continues in the next page)

(Continues from the previous page)

62	PointInDirectionBrick	brick	null	FormulaBrick	8324
63	PointToBrick	brick	BrickWithSpriteReference	BrickBaseType	2499
64	PreviousLookBrick	brick	null	BrickBaseType	1181
65	RaspiInterruptScript	script	null	Script	4
66	RaspiSendDigitalValueBrick	brick	null	FormulaBrick	14
67	RepeatBrick	brick	LoopBeginBrick	FormulaBrick	18431
68	RepeatUntilBrick	brick	LoopBeginBrick	FormulaBrick	3322
69	ReplaceItemInUserListBrick	brick	null	UserListBrick	1273
70	SayBubbleBrick	brick	null	ThinkBubbleBrick	856
71	SayForBubbleBrick	brick	null	ThinkForBubbleBrick	3876
72	SceneStartBrick	brick	NewSceneDialog. OnNewSceneListener	BrickBaseType	15017
73	SceneTransitionBrick	brick	NewSceneDialog. OnNewSceneListener	BrickBaseType	5879
74	SetBackgroundAndWaitBrick	brick	null	SetBackgroundBrick	175
75	SetBackgroundBrick	brick	null	SetLookBrick	12724
76	SetBounceBrick	brick	null	FormulaBrick	1114

(Continues in the next page)

(Continues from the previous page)

77	SetBrightnessBrick	brick	null	FormulaBrick	5764
78	SetColorBrick	brick	null	FormulaBrick	2816
79	SetFrictionBrick	brick	null	FormulaBrick	262
80	SetGravityBrick	brick	null	FormulaBrick	1216
81	SetLookBrick	brick	OnLookDataList ChangedAfterNewListener	BrickBaseType	153780
82	SetMassBrick	brick	null	FormulaBrick	572
83	SetPenColorBrick	brick	null	FormulaBrick	666
84	SetPenSizeBrick	brick	null	FormulaBrick	502
85	SetPhysicsObjectTypeBrick	brick	Cloneable	BrickBaseType	7812
86	SetRotationStyleBrick	brick	null	BrickBaseType	1079
87	SetSizeToBrick	brick	null	FormulaBrick	131822
88	SetTransparencyBrick	brick	null	FormulaBrick	23853
89	SetVariableBrick	brick	null	UserVariableBrick	209569
90	SetVelocityBrick	brick	null	FormulaBrick	6176
91	SetVolumeToBrick	brick	null	FormulaBrick	2705

(Continues in the next page)

(Continues from the previous page)

92	SetXBrick	brick	null	FormulaBrick	14702
93	SetYBrick	brick	null	FormulaBrick	18093
94	ShowBrick	brick	null	BrickBaseType	141376
95	ShowTextBrick	brick	null	UserVariableBrick	0
96	SpeakAndWaitBrick	brick	null	FormulaBrick	826
97	SpeakBrick	brick	null	FormulaBrick	10169
98	StampBrick	brick	null	BrickBaseType	222
99	StartScript	script	null	Script	232930
100	StopAllSoundsBrick	brick	null	BrickBaseType	5848
101	StopScriptBrick	brick	null	BrickBaseType	2765
102	ThinkBubbleBrick	brick	OnClickListener	FormulaBrick	164
103	ThinkForBubbleBrick	brick	null	FormulaBrick	396
104	TurnLeftBrick	brick	null	FormulaBrick	6552
105	TurnLeftSpeedBrick	brick	null	FormulaBrick	752
106	TurnRightBrick	brick	null	FormulaBrick	5044
107	TurnRightSpeedBrick	brick	null	FormulaBrick	521
108	UserBrick	brick	OnClickListener	BrickBaseType	668

(Continues in the next page)

(Continues from the previous page)

109	VibrationBrick	brick	null	FormulaBrick	3130
110	WaitBrick	brick	null	FormulaBrick	174474
111	WaitUntilBrick	brick	null	FormulaBrick	1959
112	WhenBackgroundChangesScript	script	ScriptBrick/OnLookDataListChangedAfterNewListener	BrickBaseType	8316
113	WhenClonedScript	script	rectangular	Script	2717
114	WhenConditionScript	script	rectangular	Script	15450
115	WhenNfcScript	script	rectangular	Script	27
116	WhenScript	script	rectangular	Script	110027
117	WhenTouchDownScript	script	rectangular	Script	5311
118	PhiroIfLogicBeginBrick	brick	rectangular	IfLogicBeginBrick	3
119	RaspiIfLogicBeginBrick	brick	rectangular	IfLogicBeginBrick	7
120	RaspiPwmBrick	brick	rectangular	FormulaBrick	7
121	VideoBrick	brick	rectangular		2
122	WhenBrick	brick	rectangular	BrickBaseType	167
123	IfLogicBeginSimpleBrick	brick	rectangular		0

(Continues in the next page)

(Continues from the previous page)

124	IfLogicElseSimpleBrick	brick	rectangular		0
125	IfLogicEndSimpleBrick	brick	rectangular		0
126	SetNfcTagBrick	brick	rectangular	FormulaBrick	0
127	WhenGamepadButtonScript	script	rectangular	Script	0
128		brick	rectangular	BrickBaseType	0
129		brick	rectangular	BrickBaseType	0
130		brick	rectangular	BrickBaseType	0
131		brick	rectangular	FormulaBrick	0
132		brick	rectangular	BrickBaseType	0
133		brick	rectangular	FormulaBrick	0
134	WhenClonedBrick	brick	rectangular	BrickBaseType	0
135	WhenConditionBrick	brick	rectangular	FormulaBrick	0
136	WhenGamepadButtonBrick	brick	rectangular	BrickBaseType	0
137	SetLookByIndexBrickAndWait	brick	rectangular	FormulaBrick	0
138	SetLookByIndexBrick	brick	rectangular	FormulaBrick	0
139	WhenNfcBrick	brick	rectangular	BrickBaseType	0
140	SetTextBrick	brick	rectangular	teal blue	0

(Continues in the next page)

(Continues from the previous page)

141	UserScriptDefinitionBrick	brick	rectangular	BrickBaseType	0
142	WhenRaspiPinChangedBrick	brick	rectangular	BrickBaseType	0
143	WhenStartedBrick	brick	rectangular	BrickBaseType	0
144	WhenTouchDownBrick	brick	rectangular	BrickBaseType	0
145	BroadcastScript	script	rectangular	Script	266858
146	SetBackgroundByIndex AndWaitBrick	brick	rectangular	SetBackgroundBy IndexBrick	0
147	SetBackgroundByIndexBrick	brick	rectangular	SetLookByIndexBrick	0

Table A.2: Table of all blocks (part 2)

Bibliography

[1] Wolfgang Slany (2012). *A Mobile Visual Programming System for Android Smartphones and Tablets*. Institute for Software Technology Graz University of Technology Inffeldgasse 16b, 8010 Graz, Austria.

[2] Wolfgang Slany (2014). *Tinkering with Pocket Code, a Scratch-like programming app for your smartphone*. Constructionism 2014, Wien, Austria.

[3] Wolfgang Slany (2014). *Pocket Code: A Scratch-like Integrated Development Environment for your Phone*. SPLASH '14 Companion, Oct 20-24 2014, Portland, OR, USA.

[4] `https://share.catrob.at/pocketcode/`: the official website of Pocket Code.

[5] `https://edu.catrob.at/`: resources and experiences around education combined with Pocket Code.

[6] `https://wiki.catrob.at/`: a Catrobat Mediawiki with a list of useful topics about Catrobat. [last modified on 2 November 2017, at 16:18]

[7] `https://www.jetbrains.com/pycharm/`: website of PyCharm.

[8] Digital literacy or computer science: Where do information technology related primary education models focus on? *S. Pasterk , A. Bollin*, (2017), 15th International Conference on Emerging eLearning Technologies and Applications (ICETA).

[9] Computing our future. Computer programming and coding. priorities, school curricula and initiatives across Europe *A. Balanskat, and K. Engelhardt.* (2015).

[10] Informatics education: Europe cannot afford to miss the boat. , ,Report of the joint Informatics Europe and ACM Europe Working Group on Informatics Education.

[11] Teaching Programming Concepts to Elementary Students *C. Williams, E. Alafghani, A. Daley Jr., K. Gregory, and M. Rydzewski*, 2015 Frontiers in Education Conference Proceedings (FIE 2015), 2015, pp. 706-714.

[12] Should Your 8-year-old Learn Coding? *C. Duncan, T. Bell, and S. Tanimoto*, ACM WiPSCE'14, 2014, pp. 60-69.

[13] Visual Programming for Smartphones , *Pavel Smutny'*, 2011 12th International Carpathian Control Conference (ICCC).

[14] Textual vs. visual programming languages in programming education for primary schoolchildren *Hidekuni Tsukamoto, Yasuhiro Takemura, Yasumasa Oomori, Isamu Ikeda, Hideo Nagumo, Akito Monden, Ken-ichi Matsumoto*, Frontiers in Education Conference (FIE), 2016 IEEE.

[15] Flow Experience Research of Sensing-Intuitive Dimension Learning Styles base on Visual Programming Language *Ching-Hung Yeh, Hung Hsiu-Yen*, 2015 Third International Conference on Robot, Vision and Signal Processing.

[16] Wikipedia contributors. "Visual programming language." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, [January 17th 2018].

[17] Integrating Algorithm Animation into a Declarative Visual Programming Language *Paul Carlson, Margaret M. Burnett*, Department of Computer Science, Oregon State University, 1995 IEEE.

[18] Scratch: Programming for Al *M. Resnick, B. Silverman, Kafai, J. Maloney, A. Monroy Hernandez, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver*, Communications of the ACM, vol. 52, p. 60, Nov. 2009. [Online]. Available: `http://portal.acm.org/citation.cfm?doid=1592761.1592779`.

[19] `http://scratch.mit.edu`: website of Scratch.

[20] `https://www.kodable.com`: website of Kodable.

[21] `https://www.tynker.com`: website of Tynker.

[22] `https://www.gethopscotch.com`: website of Hopscotch.

[23] An Empirical Study of Smoothing Techniques for Language Modeling *Stanley F Chen, Joshua Goodman*, TR-10-98, August 1998, Computer Science Group Harvard University Cambridge Massachusetts.

[24] Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition *Daniel Jurafsky, James H. Martin*, 1st Prentice Hall PTR Upper Saddle River, NJ, USA 2000.

[25] Language Modeling [Course notes for NLP] *Michael Collins*, Columbia University 2013.

[26] Code Completion with Statistical Language Models *Veselin Raychev, Martin Vechev and Eran Yahav*, PLDI '14 ACM SIGPLAN Conference on Programming Language Design and Implementation Edinburgh, United Kingdom.

[27] Graph-based pattern-oriented, context-sensitive source code completion *Anh Tuan Nguyen, Tung Thanh Nguyen and Hoan Anh Nguyen*, Software Engineering (ICSE), 2012 34th International Conference on Software Engineering (ICSE) Zurich, Switzerland.