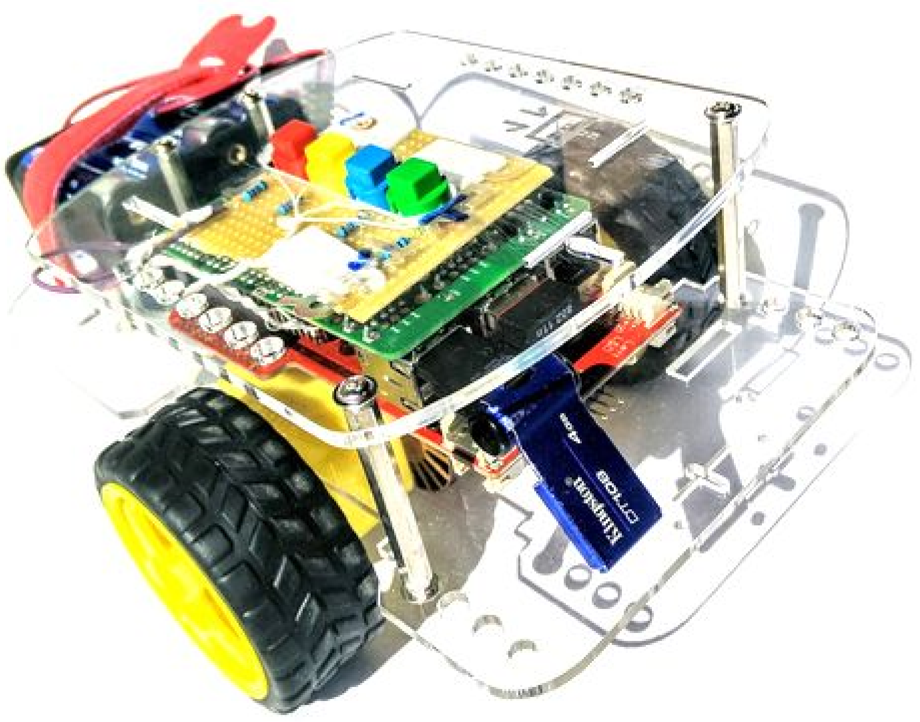


The PedaGoGo

Teaching children computer programming
Lära barn att programmera

Bachelor thesis in Computer Science and Engineering

Moa Persson
Henrik Steenari
Malin Thelin
Fredrik Thune

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY & UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

The Pedagogo

MOA PERSSON
HENRIK STEENARI
MALIN THELIN
FREDRIK THUNE

Supervisor: Patrizio Pelliccione
Examiner: Niklas Broberg

Chalmers University of Technology
University of Gothenburg

Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2016

Abstract

The aim of this report is to describe the development a programmable physical toy, designed to teach children, aged ten to twelve, computer programming concepts. As technology and computers play an increasingly large role in our society, the desire to teach young minds programming is growing. How then, do you teach children in a fun, and still pedagogical, way?

The work was divided into different phases, the first of which was a pre-study conducted to research the various ways in which children learn and how to keep them motivated when learning, particularly when learning to program. This pre-study mainly focuses on previous research and studies within this subject. The next phase was the implementation of the product; a modified version of the graphical programming language Scratch, incorporated into a webpage, and a modified version of the mobile robot GoPiGo. Based on results from two separate case studies involving users of the appropriate age, during the evaluation phase, a help panel was implemented in order to make the platform simple to use and to incorporate the pedagogical aspects of the project.

The report describes the phases of this work and the technical details of both the modifications made on the Scratch platform and the GoPiGo robot toy, which have both received a positive response from the target audience. The finished product is a robot on wheels that can successfully be programmed through the platform with a help panel, describing various programming concepts and with optional tasks, designed to motivate the users to learn. The results of the project also include observations and the results of two surveys from the conducted user evaluations.

Sammanfattning

Syftet med den här rapporten är att beskriva utvecklingen av en programmeringsbar fysisk leksak, utformad för att lära ut programmerings koncept till barn, mellan tio och tolv år gamla. Eftersom datorers och teknologins roll i samhället ökar, så gör även önskan att lära unga programmering. Frågan är då, hur lär man ut till barn på ett roligt men pedagogiskt sätt?

Arbetet delades upp i olika stadier, varav den första var en förstudie med syftet att efterforska de olika sätt som barn lär sig och hur de kan motiveras att utvecklas, särskilt då de lär sig att programmera. Den här förstudien fokuserar främst på forskning inom det här området. Nästa stadie var implementationen av produkten; en modifierad version av det visuella programmeringsspråket Scratch implementerat på en hemsida och en modifierad version av den mobila roboten GoPiGo. Baserat på resultat från två separata studier som involverar test personer i lämplig ålder, genomförda under utvärderingsfasen, implementerades en pedagogisk hjälppanel med syftet att förenkla användningen av plattformen och för att inkorporera de pedagogiska aspekter av projektet.

Rapporten beskriver de olika stadierna av arbetet och de tekniska detaljer samt beskrivningar av de modifikationer som gjorts både på Scratch plattformen och GoPiGo roboten. Både plattformen och roboten har mötts med en positiv respons från målgruppen och mött målen att vara roligt och öka intresset för programmering hos barn. Den färdiga produkten är en robot på tre hjul som kan programmeras med plattformen och tillsammans med pedagogiska handledningar, lära barnen olika programmeringskoncept samt motivera dem att fortsätta att lära sig. Projektets resultat inkluderar även observationer och resultat av två enkäter från de två utförda studierna.

Vocabulary

ICT - Information and Communications Technology
EU - European Union
Syntax (programming) - Set of rules that defines what the combinations of symbols means in a programming language
API - Application Program Interface: the specification of how the components of a software should interact
GUI - Graphical User Interface: graphical representation of an interface (comp. to text-based).
Egocentric - When a being cannot understand the viewpoints or feelings of others
CPU - Central Processing Unit: the component of a computer that performs most, if not all, basic logical and arithmetic computations
RAM - Random-Access Memory: a volatile memory device that functions as a computer's working memory; any stored information is lost if power is removed.
MicroSDHC - Micro Secure Digital High Capacity: a non-volatile memory card format
GPIO - General Purpose Input/Output
LED - Light-emitting diode
USB-port - Universal Serial Bus port: a port where one can insert USB devices
Ethernet-port - A port to where one can insert an ethernet-cable, either to connect to the internet or to a local area network (LAN).
Ssh - Secure Shell: a network protocol that allows one to connect and use a device through the terminal of another device.
IP-address - Internet Protocol Address
WiFi - technology that allows electronic devices to connect to a wireless local area network (WLAN).
Build Automation System - A system with the purpose of automating the creation of a software build.
Gradle - A build automation system.
Apache Ant - A build automation system.
ActionScript - Object-oriented programming language, primarily used to create software to be used with the Adobe Flash Player
SWF - Small Web Format, a file format for ActionScript
Adobe Flash Player - A player for Flash-projects like animations, games or other interactive works.
Flex SDK - Flex Software Development Kit; used to develop and build software for the Adobe Flash Player.
R/C car - Remote Controlled car
ZIP - A file format for compressed files or directories.
JSON - JavaScript Object Notation; a language independent, lightweight data-interchange format.

Foreword

This report describes a Bachelor's thesis project carried out by students from Chalmers Technical University and Gothenburg University during the spring of 2016.

We would like to thank our former group member Fredrik Holmberg for his contribution during the first half of the project. We would also like to thank our supervisor Patrizio Pelliccione for his help and advice during the course of the project, as well as his idea for the project. Furthermore, we would like to give an extra big thanks to our examiner Arne Linde for his additional help and advice, particularly about all things concerning hardware. Also, a special thanks to all the participants of our field studies at Kodcentrum and the elementary school here in Gothenburg.

Table of Contents

1

Introduction

Since the introduction of the modern computer, society and technology have moved forward at a great pace. Computers play a central role in basically all areas of society today. Therefore, it is obvious to identify the demand for many talented programmers, as well as for people in general to have a better knowledge of ICT. According to a report from the EU commission in 2015, Europe will be short of almost one million programmers by the year 2020 [1]. A matter that could have negative consequences, such as poor security and infrastructure of new software. So how do you meet the high demand for more programmers?

1.1 Background

An interest in teaching programming at earlier ages has grown more increasingly during the last few years. This interest mostly stems from the fact that computers and technology are playing an ever growing part in our society and basic knowledge in these areas, for a larger majority of the society, are becoming more and more important for a sustainable future.

With the hastily growing demand for more skilled programmers and engineers it is also important that all fields of technology begin to target more women - since they make up 50% of the world's population. Today, most engineering fields are dominated by men[2] and therefore appealing to girls when they are young may help to cancel out society's view on technology as being masculine and even out this gender gap.

The question then is how to teach younger minds to program computers. Both earlier and current research have been done on the subject and will be more thoroughly discussed in later chapters, mainly chapter 3 (Related Work & Research). Earlier attempts have shown that children found programming too difficult because they could not master the strange syntax. Also, the activities used to introduce programming were disconnected from the children's interests and was therefore unable to spark any further enthusiasm to study computer programming [3]. More recent research has resulted in new attempts to teach children the basics of computer programming and two examples here in Sweden are Kodcentrum (Code Centre) and Programming4Kids.

As the interest for introducing programming to younger audiences has increased, the UK [4] and New Zealand [5] are two countries that have introduced computer science topics in their primary school curriculum. Although there has been talk of teaching the basic skills of programming in the public school of Sweden, there are currently no existing plans to include it in the standard curriculum [6]. However, there are teachers and schools who on their own have decided to include some programming in the classroom[7], although this is often done in conjunction with other school subjects such as mathematics or technology.

1.2 Project Purpose & Goals

The purpose of this project is to further contribute to the efforts of teaching children programming and to help raise the level of basic knowledge about computer science in the coming generations.
The main goal is to teach children logical and systematic thinking and problem solving, but also to spark an interest in computer programming.

The goal is to develop the PedaGoGo to be a combination of the following three parts:

1. A software platform for programming
2. A programmable physical toy (robot) that can be controlled through usage of the software platform
3. Tutorials that will aid in the interaction between the platform and the robot, thus contributing positively to the learning process.

The hypothesis being that by combining the programming with a physical toy will make a child associate better with the learning process, thus leading to a better and more enjoyable experience. The ultimate goal will be to confirm this hypothesis.

The PedaGoGo will be aimed at children of any gender, 10-12 years old. The goal being to address the gender gap issue - described in the background - already at this younger age. Furthermore, the concentration on this age-span will be made more clear in chapter 3 (Related Work & Research).

1.3 Limitations

Due to the relatively large size of the project, several restrictive decisions had to be made. One of these decisions was to not write the programming platform from scratch, but to instead make use of an existing one. By choosing a language that already has a high level of pedagogy and that could be easily modified, the goals of the project would be more obtainable.

Due to the groups limited knowledge of hardware, it was also decided that the physical aid to not be built from scratch either, and that it needed to work in conjunction with some sort of microcomputer, enabling a larger focus on the software.

1.4 Outline

This report will handle a lot of different concepts and technologies. In order to provide the reader with the necessary insights to understand later chapters about implementation and results, the next chapter, Project Process, describes the methods and the outline for the project process. It describes the various parts of the project, such as planning and implementation, and the tools used to complete the project.

The following chapter, Related Work & Research, will provide the reader with details about previous research and projects that are relevant to this thesis and understanding certain decisions that were made. Chapter four, Technical Background, will describe the technical details about some of the technologies used for this project including *Scratch, Raspberry Pi* and the *GoPiGo*.

In chapter five, Implementation, a description of the details of the implementation such as modifications of Scratch is presented**.** This chapter will provide the reader with information on how Scratch was modified and also information about compiling and the elements of the interpreter.

Chapter six, User Testing, describes the process of testing the product through the involvement of users during two separate studies which are described here. Chapter seven, Results, will describe the results of the evaluations and the results of the project will be presented, with pictures of the final product. The results will then be discussed in chapter eight and the conclusions of this report will be presented in chapter nine.

2

Project Process

The project was planned into three main parts: a literature study, implementation and finally, user testing. This chapter will briefly describe these parts, as well as some of the different tools that were used during the implementation phase. This should provide an introduction to the coming chapters where these parts will be elaborated further.

2.1 Literature Study

To begin the project, a literary study was initiated and continued throughout the entire project process. Most of the extensive reading of larger works such as academic reports and articles were done primarily during the earlier stages of the project. The knowledge gained from these works served as the backbone as to why certain technologies were chosen and how the goals were defined.

The literature study focused mainly on pedagogical techniques but also on the different technologies like robots and programming languages that could be used for the project. The goal of the study was for the group as a whole to gain knowledge and a better understanding of how children learn and the best ways to teach them problem solving and logical thinking, both of which are important tools for computer programming. The goal was also to explore what previous attempts had been done to accomplish this and why they were, or were not, successful. These previous attempts would provide insight and serve as inspiration in order to try to reach the goals of the project.

Many of the more interesting results of the review can be found earlier in chapter three.

2.2 Implementation

The implementation was divided into three main parts, matching the goals of the project. These parts were; hardware, software and the help panel. The hardware and software parts were implemented side by side but as the help panel was designed based on the software, this was the last part to be implemented.

The software developed for this project was based on the visual programming language Scratch, a language that is designed for use of children and that will be described in more detail in the coming chapters. The software part of the implementation therefore involved modifying Scratch to fit the needs of the project and developing a way of transferring Scratch code to the robot toy. Another part of this implementation involved trying to develop a more child friendly GUI.

While the software was being implemented, so too was the hardware parts of the project. The physical aid chosen for this project was the GoPiGo, which will also be described further in the coming chapters. The implementation of the robot included getting it up and running, deciding which sensors, or other components, could be connected and how to make the robot sufficiently user friendly for children.

The last part of the implementation was developing a helpful and pedagogical panel, enabling the users to quickly get started programming with the PedaGoGo. This implementation also involved designing tasks for the users to complete in order to practise more complicated programming concepts.

2.3 User Testing

An important part of the project was evaluating the PedaGoGo by testing it with children of the appropriate age, ten to twelve years old. It was considered necessary to conduct user testing with children who had no previous experience, but also with children who had some programming experience. The reason for this was that even though the product is aimed at helping children learn and to spark an interest in computer science, it was also considered important that children who already possessed basic knowledge of programming would be able to continue their learning through the use of the PedaGoGo. These experienced programmers could also provide more significant feedback on the combination of the PedaGoGo platform and the robot, as opposed to programming without the physical aid. Therefore, two separate field studies were conducted in order to perform a proper evaluation of the PedaGoGo.

The first of the two studies was at an elementary school located in Gothenburg. This study involved users who had little, or no, previous programming experience, and the purpose was to investigate the perceived levels of difficulty and enjoyment of the PedaGoGo. This study also served as a way to investigate any differences between the girls and boys participating, as we aimed to create a gender neutral product. The participants were asked to fill in two different surveys, providing a strong base for the conclusions and results of this study.

The second study was as Kodcentrum (eng. Code Centre), a non-profit organisation that offers free lessons in programming for children, mainly using Scratch. This study was less scientific in nature and served more as a way to gather feedback on the product from more experienced programmers. There were no surveys involved in this study, instead the participants were asked orally for their opinions of the PedaGoGo and the robot.

These studies will be described in more detail in chapter 6 and the results can be found in chapter 7.

3

Related Work & Research

In order to provide the reader with more knowledge regarding the subject of pedagogy and teaching programming to children, this chapter will present relevant research and projects in the field. Furthermore, this should provide an understanding as to why certain decisions were made during the project process.

This chapter will introduce the reader to arguably some of the most important figures in the area and their research. The chapter begins with a description of the various phases of learning development, particularly for children, and moves on to describe a few previous projects with the aim of teaching programming to children using Scratch. For the reader to understand the reason to use a robot in the project there are some studies proving that robots are a helpful aid in reducing the gender gap when programming included in this chapter.

3.1 Jean Piaget

Jean Piaget was a psychologist (1896-1980) known especially for his extensive research about child development. He established the discipline known as genetic epistemology or “the study of the origins of knowledge” [8]. Piaget has influenced a lot of people with his research, including many educators and academics, for example professor Seymour Papert who will be discussed in the next section. Piaget’s work has allowed educators to improve on their curriculums in order to bring about a more student focused learning environment.

Piaget formulated his theory: Piaget’s theory of cognitive development in his book “The origins of intelligence in children” [9]. The theory talks about the four stages of cognitive development of a child from birth to adulthood. Table 3.1 below briefly describes the theory:

Table 3.1: Jean Piaget’s theory of cognitive development

Stage	Age	Description
1. Sensimotor	Birth - about age two	Children experience the world through movement and their five senses. They are extremely egocentric in this stage.
2. Preoperational	age two - age seven	Children are still egocentric. Playing and pretending, beliefs in magical things and the absence of concrete logical thinking.

3. Concrete operational	age seven - age eleven	Children are no longer egocentric. They can start to understand concepts like logic and conservation. Drastic improvements to classification skills.
4. Formal operational	age eleven - adulthood	Abstract thought. Can easily think logically. Major improvements to problem solving.

For the explicit purpose of teaching kids programming, stages three and four were deemed the most compelling, since at these stages the children are able to grasp logical thinking and more complex problem solving. These are considered important qualities when learning to program. The following subsections will therefore offer a deeper understanding of these two stages.

3.1.1 The Concrete Operational Stage (7 to 11 years old)

Piaget considered this stage a major turning point in the cognitive development of children [10].
It is considered a major turning point since at this stage children start to use and understand logical thinking. However, with concrete means that they can only apply it to physical objects. Hypothetical or abstract thinking is typically absent.

You can basically divide the concrete operational stage into two parts: onset and mature. At the onset of the concrete operational stage children begin to mature in the sense that they can start to grasp earlier foreign concepts like logic and conservation. A seven year old tends to understand the conservation of liquids, meaning that if you were to pour a glass of water into a differently shaped container the child would understand that it is still the same amount of liquid even if it appears to be less or more than before. A younger child not yet in the concrete operational stage would not understand this.

As they mature in this stage, previously mentioned skills, like logic and conservation, develop further and so do their classification skills. Until this stage most children have acted mostly on intuition and have been very egocentric. It is at this stage that they can start to understand that other people also have thoughts and feelings more or less like their own.

3.1.2 The Formal Operational Stage (11 to adulthood)

At the formal operational stage children are able to apply logical thinking to not only physical objects but also theoretical subjects. They can solve problems in their entirety all in their head without using any physical aids such as figures or models. Piaget has concluded that the principal characteristics of the formal operational stage is the ability to systematically solve a problem, abstract and logical thinking and to see the relationship between things [11].

However, a study showed that typically about 30-35% of adults do not fully reach this stage [12]. Biological maturation itself is not enough to allow one to reach the formal operational stage, a special environment e.g a good teaching environment is needed to attain this stage. Be it at home, at school or anywhere else, an individual actually needs to be offered challenges that train this kind of thinking and problem solving in order to reach this stage.

3.2 Seymour Papert

Seymour Papert (1928) is a mathematician and computer scientist at MIT [13]. He is considered one of the pioneers of artificial intelligence. He also considers himself to be a genetic epistemologist and one of the more dedicated followers of Jean Piaget's theories.

Combining his vast interest for math and computer science with the teachings of Jean Piaget he has written a number of books about teaching children by using a computer. Describing the computer as an incredibly powerful tool to improve children's learning.

In 1967, together with three other colleagues, he developed Logo; a programming language aimed at children. After using the language in schools and with kids for several years, Papert released his most famous work "*Mindstorms: Children, Computers and Powerful Ideas*"[14] in 1980. In this book he presents the reader many concrete examples of how the experiences the children have had with Logo actually improved their learning skills.

3.2.1 The Logo programming language

Logo was created in 1967. It was primarily developed and designed by Seymour Papert and Wally Feurzeig, another pioneer in artificial intelligence. The aim of this language was to be used as a tool by school educators in order to use the computer in a more efficient way when teaching children. Papert's view was that educators mainly used the computer to give children repetitive problems like flashcards, which in his opinion did not utilize the power of the computer well. The language was used and tested extensively by Papert, Feurzeig and their research team in several elementary-, middle- and high schools in the United States for several years [14].

The language and coding itself is text-based but relies on graphical components when run. In short, the language utilizes something called a "*turtle*" and "*turtle graphics*". The turtle is an entity presented as a triangle on the computer screen. It is this entity that the programmer is supposed to control. When the turtle moves according to the programmer's commands it draws a trail behind it (see figure 3.1), allowing the user to draw shapes or whatever comes to mind, only the imagination limits the possibilities [14].

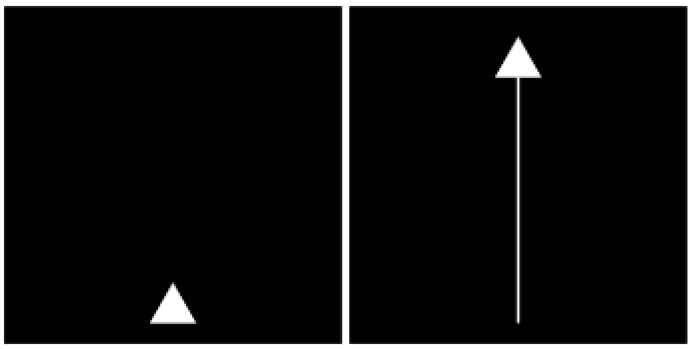


Figure 3.1: The "Logo turtle" before and after moving forward some steps.

This simple idea allowed children who before had trouble with mathematics to get more hands on with the subject. For example, in order to draw a square with the turtle you had to understand how turning a certain amount of degrees would affect the turtle. Children who before simply had found no meaning in memorizing 'boring' math rules, like the rule that one whole rotation is 360 degrees, could find themselves discovering these rules by themselves and then applying them in order to finish missions with the turtle.

The Logo programming language was one of the first attempts to teach kids programming, and also to educate them through programming. A lot of the following work and projects in the area have their roots in Logo, like Smalltalk, Squeak, Scratch and Lego Mindstorms.

3.2.2 Mindstorms: Children, Computers and Powerful Ideas

This book was written by Seymour Papert and released in 1980. Although the book is 36 years old the ideas, research and views presented by Papert still hold much relevance today. The book has inspired to a lot of further study and projects, as well as prompting educators to become more creative when combining teaching and computers. The book is arguably one of the most important works in the area.

As mentioned in the previous section, the book talks about the programming language Logo and Papert's experiments using it in American schools. He explains that the computer needs to get more integrated in schools and life in general. He also explains that people need to stop 'fearing' the computer, meaning that because some people see themselves as "non-technical people" they will stay away from computers. His aim is to create tools for these self-professed non-technical people and allow them to take part in the valuable learning process that comes as a result of using a computer.

Beyond the fact that people tend to classify themselves as being this or that, technical or non-technical et cetera. Many will also put up 'learning barriers' by classifying themselves as bad at something, like math or French. Papert argues that most children born in France will have no problem learning French, so maybe children in "Mathland" will have no problem learning math. It is all about creating the right environment for learning.

A child learning to speak sees the need to learn in order to be able to communicate with the world around it; the child can associate with the learning process. One of the things that Papert says in the book is:
"...learning languages is one of the things children do best. Every normal child learns to talk. Why then should a child not learn to 'talk' to a computer?" [14].

This quote represents Papert's view on that teaching programming to children should not be something to be feared, but rather something to embrace. That it is important to not put up these unnecessary learning barriers.

The book describes several concrete examples of how the children have improved their learning skill. By using Logo to solve problems many children learn that dividing a problem into several subproblems is a good way to make progress on the problem at large. What they then realize is that this technique of using subproblems can be utilized with other non-programming related problems, like the problem of writing an essay or learning to juggle. The book describes how hundreds of these students have easily learned to program despite the fact that 'most' people:

"...consider programming a complex skill acquired by some mathematically gifted adults…".[14]

In conclusion, the book is about what Papert calls 'powerful ideas' that will motivate children to learn. He argues that school is too much about memorizing facts and learning skills that will never be used. A lot of children ask why they should learn a certain skill, and receiving no good answer they will disassociate from the learning process. Finding no adequate meaning behind learning something will make it boring. Just "being good in school" is not enough to motivate most children. Papert also argues that

> *"...many children are held back in their learning because they have a model of learning in which you either 'got it' or 'got it wrong'. But when you learn to program a computer you almost never get it right the first time."* [14]

Meaning that being wrong is instead an integral part of the learning process; to debug your own program is a most valuable lesson. To provide children with a tool where they can try and fail, then try again, until they hopefully succeed, will build a certain confidence in their own ability.

3.3 Scratch

Scratch was first introduced in introduction, section 1.3, and technical background, section 4.1, will provide technical details about this visual programming language. In this section Scratch will be discussed on a more theoretical level and thus the aim is to provide a strong background to the decision of using this platform for the project.

Scratch is a combined programming language and editor that was first conceived in 2003 by the Lifelong Kindergarten group at the Massachusetts Institute of Technology Media Lab. It was designed by Mitchel Resnick, a professor from MIT and doctoral student of Seymour Papert, with the aim of helping children not just to learn programming but also to reason systematically and think creatively. All of which are important skills to have in the 21st century [15].

According to the developers of Scratch, the idea behind this special approach to programming was to appeal to people who had not previously considered themselves as programmers and to make it easy for everyone, no matter what age or background, to create programs and share them with other people [15]. The public launch of Scratch was in 2007 and is now used in more than 150 countries and available in more than 40 languages. According to the Scratch homepage they now have a learning community with 13 301 505 projects shared from all around the world. [16]

Unlike other high-level programming languages used today, Scratch makes it easy to understand and get the hang of programming compared to traditional text-based languages. Its visual style based on graphical programming blocks means that there is no difficult and obscure syntax involved. The developers of Scratch explain how Lego was one of the main inspirations and it is easy to see how the blocks could be compared to Lego bricks [17].

Scratch also offers the possibility to import photos and sounds, such as music or recorded voices, and create graphics making it easy for children to personalize their projects and making it more fun for them to experiment with. Scratch also supports a variety of different types of projects, such as games, stories, animations and tutorials, ensuring there is something for everyone to enjoy.

According to the article "Scratch for budding computer scientists" published by two professors from Harvard University, they view Scratch as a 'gateway' programming language [18]. They also write about other alternatives to Scratch - being visual and aimed at beginner programmers - but conclude that they come with a higher learning curve or are too restricted when compared to Scratch.

In a study done by the Weizmann Institute of Science in 2013 they found that beyond the use of Scratch for children, it could also be beneficial to use the language in introductory programming courses for college students [19]. Students that had programmed in Scratch before learning a text-based programming language learned that language more easily because they could recognise the fundamental concepts from Scratch. However, it should be noted that the study also found that at the end of the teaching process there was no significant difference in programming ability when comparing students who had worked with Scratch before and those who had not.

3.4 Scratch Projects

As mentioned earlier, Scratch already has a large community of young programmers and projects that take on forms such as games and animations. This section will provide examples of some of these projects in order to accommodate the reader with information about what can be accomplished when using Scratch.

3.4.1 Scratch and Mathematics

Students in grade 5-7 were participants in a study where they were being taught mathematics using Scratch [20]. They were shown different mathematical concepts, such as linear functions and algebra, and asked to create their own projects in Scratch, highlighting one of these concepts. One of the resulting programs made by the students was "Spider House". This was an animation explaining the concept of mathematical functions. It tells a story about how one spider helps his friend to build a solid house, a spider web, by explaining and using quadratic functions.

The results of the study showed that participants found it to be a fun way to learn mathematics since they now had some activity that made it easier to learn. Like Papert described in his book, the children could apply the mathematical concepts to something tangible that they wanted to create, rather than just learning mathematical rules for "no reason".

3.4.2 Creative Coding

Young artists have worked in Scratch to create projects that are closely connected to their interests. One girl, 13 year old Kaylee, created an animated dance video based on her favourite song "Hollaback Girl" by Gwen Stefani [21]. The user BalaBethany, also a 13-year old girl, used Scratch to create animated stories for the anime characters she enjoys to draw [3]. By announcing a contest on the Scratch web page to create a new character for her stories, she actively involved the Scratch community in her projects. As one of the participants in this contest did not know how to draw anime characters, so BalaBethany created a tutorial using Scratch, explaining the process of drawing and coloring anime characters.

Yet another Scratch project using creative coding is a simulation called Countries created by a 14 year old girl named Shannon [17]. Based on her love of the game SimCity and what she had been learning in her history class, she created this simulation about virtual countries and decisions that need to be made in response of economic, agricultural and political crises. This is also a great example of how schools could incorporate programming into other school subjects to make them more fun and easier to learn.

These are three examples of how one can connect one's personal interests to programming and create something fun and exciting. This connection makes it easier to understand concepts that can be hard for novice programmers, such as loops and conditionals [21].

3.5 Programming with Robots

There are also some existing robot toys that can be used with programming languages like Scratch[22]. Some examples are Lego WeDo[23], Lego Mindstorms[24] and Finch[25]. Lego WeDo have several ideas for programmable lego toys, a few examples are "Ball Kicker", "GoalKeeper" and "Cheering Fans" that could be used to program a football-inspired game [23].

Lego Mindstorms is another example of Scratch being used to program robot toys. Unlike the GoPiGo used for this project, Lego Mindstorms does not use a Raspberry Pi but instead uses a computer called "the NXT Intelligent Brick". This brick is then used to program and control the Lego Mindstorms EV3 robot that can be assembled in several different ways into a variety of robots [24].

Finch is a small robot inspired by the bird with the same name and is designed especially for Computer Science education. It is made by BirdBrain Technologies and supports over a dozen different programming languages including Scratch. Aside from being mobile on wheels, some other features include light, temperature and obstacle sensors, a penmount for drawing capabilities and a full-colour beak LED [25].

Another example of robot toys designed to teach children programming and to introduce them to technology is Dash and Dot. These are two small robot pets for children of all ages. They can be used with various different free apps such as "Wonder" and "Blockly" to create games and behaviours. The robots' features include microphones, a speaker, lights and distance sensors. The company behind these two robots, Wonder Workshop, have also created various snap-on accessories ranging from bunny ears to a xylophone that the robots can be programmed to play [26].

3.6 Gender, programming and robots

As mentioned previously in the report, the world of computer science is at present largely dominated by men. This can be seen clearly while looking at statistics of the applicants to the Information Technology and Computer Science bachelor programs at Chalmers University of Technology. In 2015 Only 12% of the applicants for Computer Science and 18% of the applicants for Information Technology were women [27]. This is an issue that undoubtedly deserves some more attention.

Although it is difficult to determine the exact cause of this gender gap, it is widely believed that the causes are mainly due to society's perception that computers and technology, in general, are masculine and meant for males. In the following studies, the use of robots or other physical tools, proved to be effective in helping women learn programming and in giving them the confidence to keep learning.

3.6.1 Learning to program with Personal Robots

An article written about a study conducted at Bradley University, discusses the motivational effects on students when using an IPRE (Institute for Personal Robots in Education) robot to teach an introductory programming course [28].

The study was conducted over a 15 week period at the Bradley University's Department of Interactive Media. In the results she notes three significant relationships between gender and the use of the robots as an educational tool in the course (Page 19-20).

1. The female participants had a stronger belief than the male participants, that the use of robots to learn how to program had helped them
2. The female participants found it somewhat intimidating to use the robots while the males did not
3. The majority of the female participants agreed that "It was a pleasure to work with the robot to learn how to program" while the majority of the male participants did not

After reviewing the results, McGill states that:

> *"Female participants were able to overcome their intimidation and possibly may have felt some accomplishment in overcoming their fears. This may be worthy of further investigation, as using the robots may give a sense of empowerment to women and girls and may give them more confidence in learning additional programming."* (Page 25).

3.6.2 Closing the gender gap in an introductory programming course

Another, similar, study was conducted at the University of Granada [29]. The purpose of this study was to determine whether there was a gender gap in a traditional introductory programming course taught at the university and if so, could this gap could be reduced using physical computing principles.

The study, conducted during a ten week period, looked specifically at differences between men and women with three main focus points

1. Perceived ease of programming
2. Perceived usefulness
3. Intention to program (Page 4)

The men and women participating were all 18 years old and had no previous programming experience. The participants were divided one control group and one experimental group. Both the groups consisted of 38 students with the same instructor teaching both groups. (Page 3).

The material and theoretical concepts used in the course were presented to the control group in the traditional way, with PowerPoint slides and multimedia material during the lectures.
For the experimental group however, the lecturer demonstrated examples using physical equipment, such as LEDs, sound and movement, which the students in the experimental group also had access to during lab sessions. (Page 3).

At the beginning of the course, all the participants in both groups were presented with questions on the three main focus points of the study, presented above. Results show that at this stage the difference between the male and female participants was small in both the control group and the experimental group.
When presented with the questions again at the end of the course, the differences between the genders in the experimental group remained small while in the control group they were significantly more noticeable. At this stage the perceived ease of programming as well as the intention to program was much higher in the male participants than the females in the control group. (Page 6)

Also at the end of the course, all participants of the study attended a written exam in order to observe any differences in the learning outcome between the genders. The results of the experimental group and the control group differ significantly as the failure rates of the women in the control group was much higher than in the experimental while the the failure rates of the men were almost identical. (Page 10).

The authors of the article came to this conclusion:

> "*The use of physical computing principles in combination with the traditional methodology reduced -actually closed- this gender gap."* (Page 9).

4

Technical Background

The project utilizes several different technologies and this chapter intends to provide more technical details about these. This is to create a clearer picture of these concepts when later reading about how and why these technologies were used for the project.

This chapter will mainly provide information about the programming language Scratch, the microcomputer Raspberry Pi and the GoPiGo robot.

4.1 Scratch

Scratch is a visual programming language intended to be used by children between the ages 8-16 [15]. The language can be used creatively to create games, animations and other interactive works, the aim being to teach younger or inexperienced people to program [15] [16].

There are several different versions of Scratch. The latest one being Scratch 2.0, available since 2013 [16]. Before 2.0 there were versions 1.0 up to 1.4. Scratch is available both as a web-application and as an executable program that you can download from the Scratch website. Since Scratch is open-source, there also exist multiple modified versions of Scratch.

This section will further explain the technical details of Scratch 2.0, since this was the version used in the project.

4.1.1 The Scratch 2.0 interface

The interface of Scratch 2.0 provides its users with some basic tools to quickly get started with computer programming as well as to see their progress easily. When creating the interface, the Scratch Team's main priority was to make the development environment intuitive for children who had no previous experience with programming [16]. The main parts of this interface are explained below in figure 4.1.

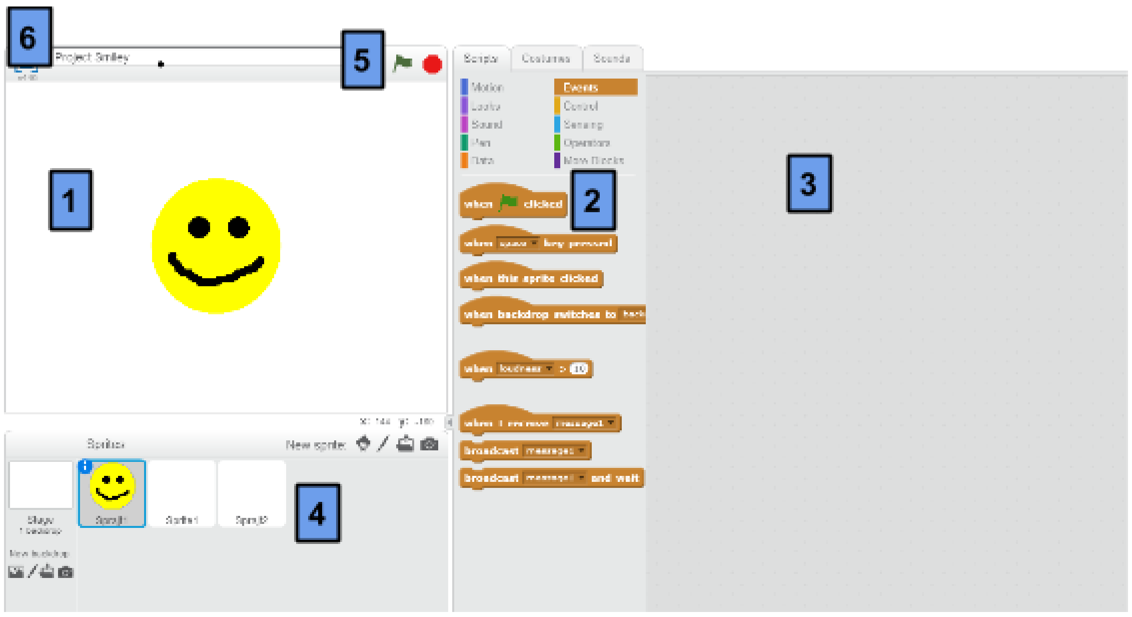


Figure 4.1: The Scratch 2.0 graphical user interface, with numbers marking the different parts.

1. The “stage” will allow the programmer to see the result of their coding
2. The “block palette” which contains the “blocks”
3. The “script area” is where the programmer performs the coding using blocks from the palette
4. The “sprite list” shows any active sprites

Worth mentioning is also:

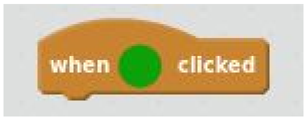
5. The start (green flag) and stop (red octagon) controllers
6. The project name

4.1.2 Blocks

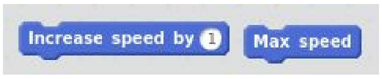
Coding in Scratch is done using “blocks”. They can be described as graphical representations of code. These blocks can be connected with each other, much like a jigsaw puzzle [30]. Several blocks connected with each other is called a script.

In order to prevent syntax errors, each block has one out of eight different shapes. The shape of a block will restrict it so it is only possible to connect it to certain other blocks [31]. All of these shapes are pictured in the Table 4.1 below.

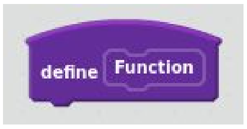
Table 4.1: All the different types of Scratch blocks.



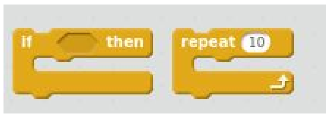
(a) Hat block



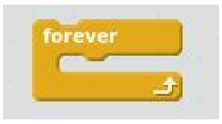
(b) Stack block



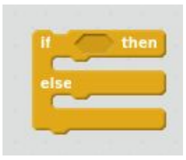
(c) Define block



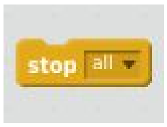
(d) C block



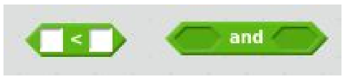
(e) Final C block



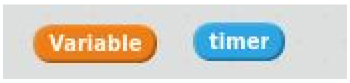
(f) E block



(g) Cap block



(h) Boolean block



(i) Reporter block

(a) The hat-block. The rounded shape at the top makes it possible to only attach other blocks beneath it, and none above it. This block is necessary at the beginning of a script.
(b) The stack-block. A notch above and a bump below allows it to be built on from both above and underneath. This is the most common type of block shape.
(c) The define-block is used when defining your own blocks directly in Scratch. Beneath it you can attach blocks to create a script. Doing this will in turn create a new stack-block called "Function". Then, whenever using "Function" the previously created script will be run.
(d) The C-block. These blocks run or loop the content that can be attached "inside" of the c.
(e) A final C-block is used when something will be done forever. Compared to a normal C-block there is no bump below, since there is no point in attaching further code - it will not be run anyway.
(f) An E-block is used much like a C-block except there is also an else command that can be run.
(g) A cap-block is used at the end of a script. Much like a final C-block, there is no bump on the bottom, thus preventing any further code to be attached.
(h) The boolean-block is used to handle boolean expressions.
(i) The reporter-block is used to handle certain data and variables.

In figure 4.2 below we can see an example of using four different blocks (of three different types) to form a script:

1. A hat-block to start off the script when the green flag is clicked
2. A C-block to initiate a loop that will repeat 7 times
3. A stack-block that tells a sprite to move 10 steps
4. A second stack-block that tells the sprite to turn 15 degrees

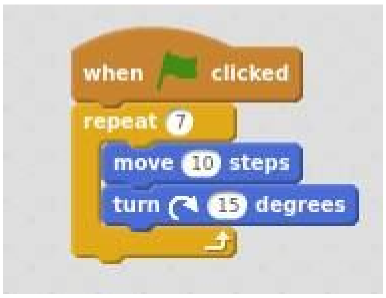


Figure 4.2: An example of a script in Scratch.

There are ten different categories of blocks: Motion, Looks, Sound, Pen, Data, Events, Control, Sensing, Operators and More Blocks. In total there are 145 different blocks to use.

4.1.3 Coding with Blocks

The Scratch coding process can be explained simply. A programmer will begin to choose blocks from the block palette. Then drag-and-drop them to the script area and connect them together to form scripts. When the programmer feels satisfied with their created scripts they can run them. By observing the stage they can see how their created scripts are affecting any sprites present there. This process is explained below in figure 4.3.

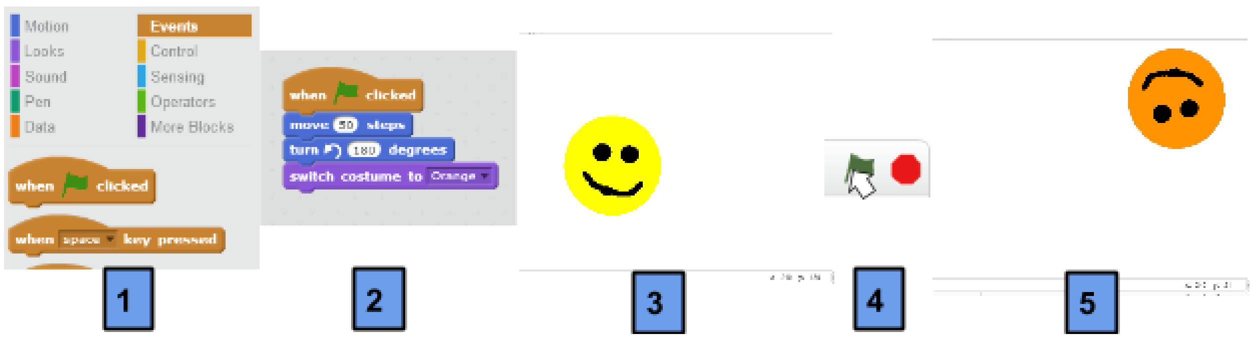


Figure 4.3: The step-by-step process of coding in Scratch.

1. Choosing blocks from the block palette.
2. Connecting blocks to form a script in the script area
3. Original state of the stage with a smiley sprite on it
4. Click the green flag (starting the script)
5. The resulting state of the stage after the script has been run

4.2 Raspberry Pi

The Raspberry Pi is a relatively powerful, creditcard-size microcomputer. The Pi works like any normal computer would, one of its limitations being its small size which makes it unable to contain more powerful components. The advantage of the Pi is its modification ability and its cheap price.

There are several versions of the Pi and they are much alike, but the following information will primarily concern the *Raspberry Pi 2 Model B* since this was the one used for the project.

4.2.1 The Hardware

To give a better understanding of what the Raspberry Pi is, and can be, capable of this subsection will describe some of the Pi's hardware. The Raspberry Pi can be seen in Figure 4.4 below.
Present on the card are several components[32]:

- CPU - 900MHz quad-core ARM Cortex-A7
- Memory - 1 GB RAM
- Storage - microSDHC slot
- 4 x USB slots
- 1 x HDMI slot
- 1 x 3.5mm Audio jack
- 1 x 10/100 Ethernet port
- 40 x GPIO (General Purpose Input/Output) pins

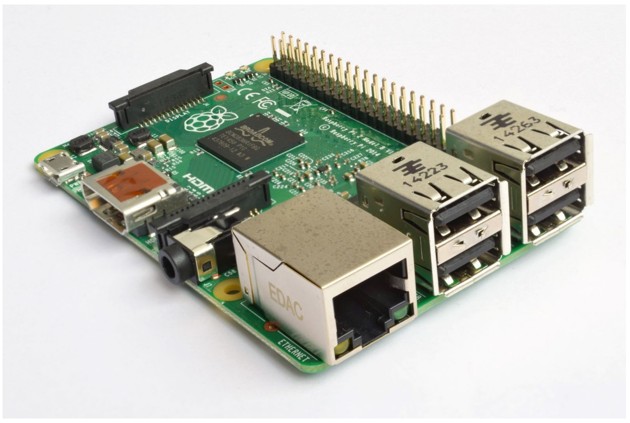


Figure 4.4: The Raspberry 2 Model B[33].

The ARM Cortex-A7 CPU is the most power-efficient multi-core processor [34], which makes it a good choice for the Raspberry, which usually needs to power several additional components; components that can be connected either to the different ports or to the GPIO pins. This processor is also used in phones and different wearables. The Raspberry Pi contains no hard drive for storage, instead it has a slot for a microSDHC card, which makes the Pi flexible to use but the lack of space that a microSDHC card means can be a problem. In conclusion, the Raspberry Pi is a powerful computer for its price, and allows a user to experiment with hardware and programming in a safe and cheap way.

4.2.2 The GPIO Pins

One of the more important aspects of the Raspberry Pi are the General Purpose Input/Output (GPIO) pins (Figure 4.5-6), all located on the right side of the card as seen in Figure 4.4. These pins make it possible to attach any desired hardware that is compatible with the Raspberry Pi, such as different sensors, LED-lights, buttons, or a GoPiGo.

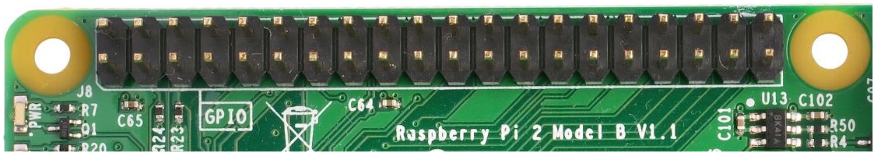


Figure 4.5: The GPIO pins on a Raspberry Pi 2 Model B.

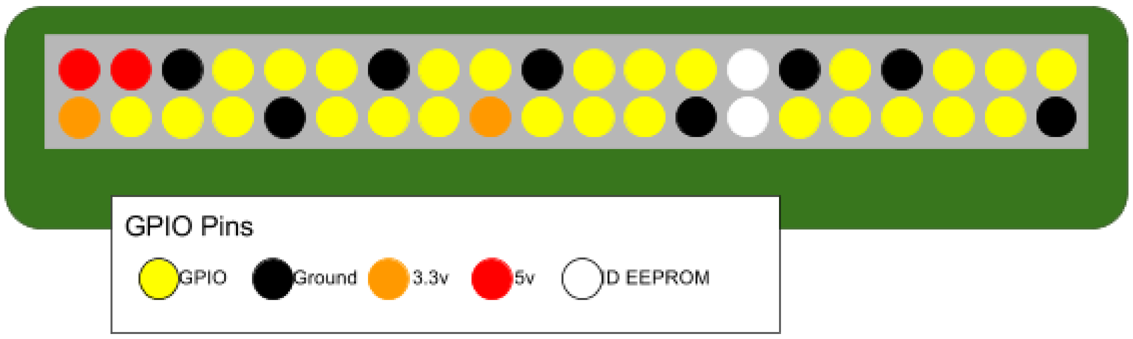
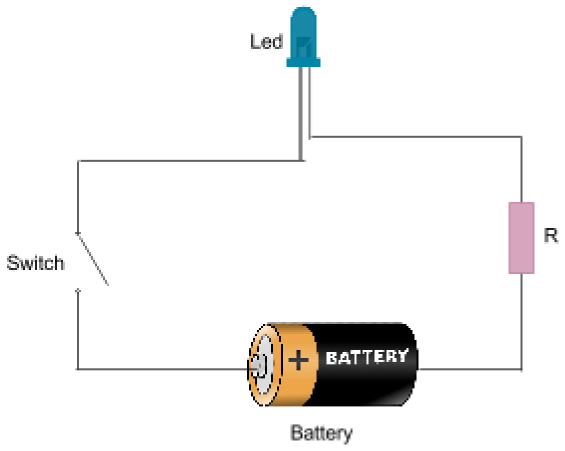


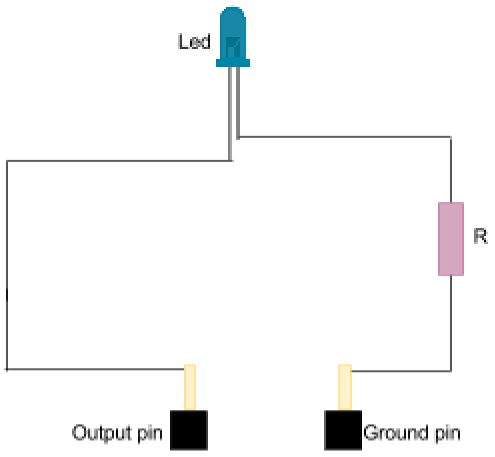
Figure 4.6: Graphical explanation of the GPIO pins

There are a total of 40 pins on the board, and 26 of them are GPIO [35]. The rest are: eight ground pins, four power output pins - two 5V and two 3.3V (Figure 4.6). There are also two pins called ID EEPROM pins, which we will not elaborate any further on in this report.

The GPIO pins can be used either as an input or an output. When a pin is used as an output the Pi works both as the power supply and as a switch to any connected component (Figure 4.7a-b); it can be programmed to go either high - output is 3.3V - or low - the pin is off.



(a): A simple circuit with a LED connected to a battery and a switch to turn it on and off.



(b): The battery is replaced by two GPIO pins on the Raspberry Pi. The output pin works both as the positive side of the battery and as the switch. The ground pin works as the negative side of the battery.

Figure 4.7: Two identically working circuits used to control a LED to turn on or off.

When a pin is used as an input - like when a button is connected to it - in order to work properly, the Pi needs a certain reference to know if the button is on or off. Otherwise it would:

"...be a bit like floating in deep space; without a reference it would be hard to tell if you're going up or down, or even what up or down meant!" [35]

These references are made by making use of electrical components known as 'pull-up' or 'pull-down' resistors present on the Pi (see figure 4.8a-b). These resistors make sure that the pin in question is either in a high (pull-up) or low (pull-down) logical state [36], and not floating in space.

4.2.3 Pull-up and Pull-down Resistors

A pull-up resistor is most commonly used in conjunction with buttons or switches. This resistor has the pin going high when the button is not pressed, then when the button is pressed the pin gets directly connected to ground and thus it goes low (figure 4.8a) [36].

Hence, a GPIO pin can be programmed to initially be pulled-up and a program can 'listen' for when it goes low and take appropriate action thereafter. In the programming language Python this is done with the following code:

```
GPIO.setup(number_of_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

The function "GPIO.setup()" takes three parameters:

1. The number of the pin to be manipulated,
2. If this pin should work as an input or an output, and,
3. If initially it should be in a pulled-up (high) or pulled-down (low) logical state.

"GPIO.IN" means that the pin should work as an input and defining the parameter "pull_up_down" to "GPIO.PUD_UP" tells the pin to be in a pulled-up state.

A pull-down resistor works similarly, but the other way around; the intended pin is programmed to initially be in a low state and if a button is pressed the pin will get connected to a powersource. If this source is powerful enough it will make the pin go high (figure 4.8b).

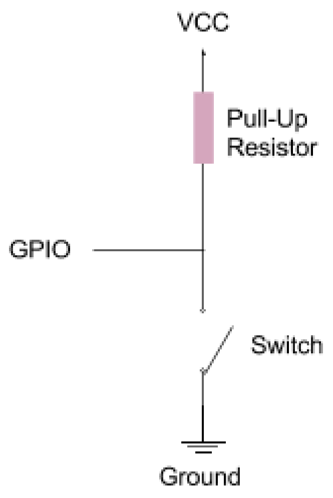
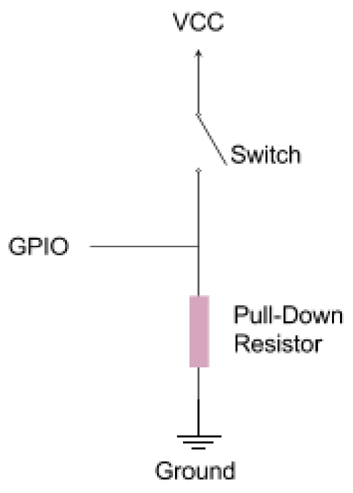
 **(a).** The GPIO pin is connected to the power source (VCC), thus being in a high logical state. If the switch is pressed the pin will get connected to ground bringing it into a low logical state.	 **(b).** The GPIO pin is connected to ground, thus being in a low logical state. If the switch is pressed the pin will connect to the power source bringing it into a high logical state.

Figure 4.8: The pull-up and pull-down resistors used in two slightly different circuits.

4.2.4 Python with Raspberry Pi

Python is a high-level computer programming language. The Raspberry offers the 'RPi.GPIO' library to control the GPIO pins using Python code. Python is used for its readability and for the ability to write the same code using less lines compared to languages like Java and C++ [37]. More about how Python was used will be elaborated on in chapter 5: Implementation.

4.3 GoPiGo

The GoPiGo is a robot built to work together with the Raspberry Pi, and it works with any version of the Pi. The GoPiGo is made to be attached to 26 of the 40 GPIO pins on the Raspberry.

The GoPiGo is made by Dexter Industries who are known for their robotic sensors and systems. Their idea behind making the GoPiGo was to create a simple robot platform that did not require the user to have a deeper understanding of hardware [38]. Thus, allowing people who want to focus on the software side of robots to get more quickly started.

The GoPiGo robot primarily consists of a board (the GoPiGo board), a battery pack (12V), two motorized wheels and a third support wheel. Some sensors, two LED-lights and pins for attaching additional components are also available to control.

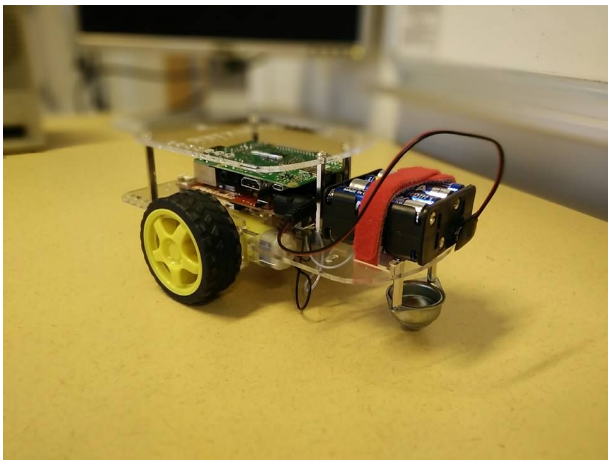


Figure 4.9: The assembled GoPiGo with the Raspberry Pi (green card) attached

The GoPiGo has compatibility for a variety of programming languages which include Python, Scratch, Java, NodeJS and C as well as C++ [39]. The robot comes with an API of how to control the different components which can be utilized when programming with these different languages.

4.3.1 The Functionalities of the GoPiGo

The GoPiGo, without any additional components attached, is capable of:

- Rotating one or both wheels any number of revolutions, either forward or backwards
- Turning on or turning off the right or left LED light
- Increase and decrease the speed, from 0 to 255

The speed varies depending on the current state of the batteries, so a definite max-speed is not set.

4.3.2 Raspbian for Robots

Dexter Industries has created their own version of the Raspbian OS called ‘Raspbian for Robots’ to be used specifically with the GoPiGo. The OS is an extension of Raspbian and works pretty much exactly like it. It comes with some improvements for robotics and comes with all of the Dexter Industries software preinstalled, as well as their own GoPiGo version of Scratch [40]. These factors make setup and connection to the Raspberry Pi easier.

4.3.3 Using the GoPiGo with the Raspberry Pi

In order to utilize the functionalities offered by the GoPiGo one must communicate with it by using the connected Raspberry Pi. This can be done by simply connecting a monitor with mouse and keyboard to the Pi and from there communicate with the GoPiGo through the Raspberry Pi OS. This is not generally a good idea for a robot considering that it is usually not stationary, thus being connected to something with a cable could hinder it.

One can also connect the Raspberry to another computer via an ethernet cable, thereby using that computer’s monitor, mouse and keyboard instead of connecting all of those devices directly to the Pi, but this basically provides the same issue with immobility. Instead, by connecting the Pi to the computer using WiFi it does not require any cables and is therefore the best solution for transferring information to a movable robot. Therefore, a mini WiFi-dongle, in the form of a USB-device that could be inserted into the Raspberry Pi, was utilized for the project.

By connecting to a WiFi network the Raspberry Pi receives an IP-address. This address is then used to connect to the Pi from another computer [41]. Dexter Industries offers to perform this connection through your browser, and one can choose to either see the desktop of the Raspberry Pi or to use the Raspbian terminal [42]. This works quite slowly and is prone to crashes or disconnections from the Pi. A faster solution is instead to connect to the Pi by using Secure Shell, more commonly known as ssh. When this connection is established one is able to communicate as one wishes with the Raspberry Pi and thus also the GoPiGo.

5

Project Development & Implementation

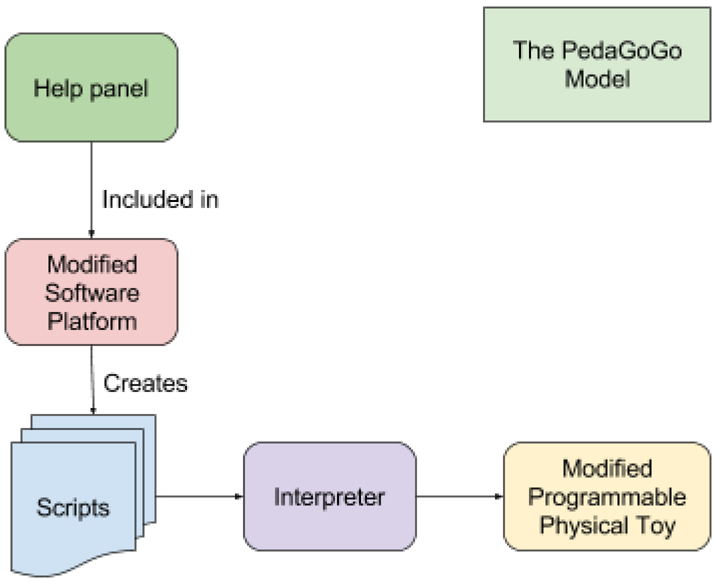


Figure 5.1: The PedaGoGo model, consisting of four different parts.

This chapter describes the iterative development process and implementation of the four different parts of the project. The four parts are described in Figure 5.1 above and include:

- The modification of Scratch 2.0, including its GUI
- The modification of the GoPiGo
- The creation of an interpreter between the Scratch project files and Python
- The creation of help panel

Due to these parts being developed simultaneously this chapter will attempt to avoid confusion by explaining the process in as linear a way as possible.

5.1 Choosing the Technologies for the PedaGoGo

In order to realize the PedaGoGo according to the goals specified in the introduction, the first part of the development process was to choose the appropriate technologies. Due to the fact that the user of the PedaGoGo would primarily be learning programming by using the software platform, and the robot mainly being used as a physical aid to that process, the platform was chosen first. Thus the robot would then have to be chosen based on what platform had been chosen, instead of the other way around.

5.1.1 Choosing the Software Platform

Since it was determined early on that the software platform to be developed for the project should be based on an already existing platform, the initial research in the project was focused on finding a proper platform. Ultimately, Scratch version 2.0 was chosen based on its establishment as a good, first programming language for children, the fact that it was open-source, and its compatibility with most platforms and devices.

5.1.2 Deciding to use the Raspberry Pi

Since the robot was supposed to be controlled by programs written in Scratch, it was determined that the robot had to be controlled by an external piece of hardware. This hardware should in a general sense: read the code and then send the corresponding commands to the robot. This way we would be able to have more control over both platform and robot. Rather than using a potential robot's pre-existing user interface that could offer unnecessary restrictions.

The two main options were to use either the microcontroller Arduino or the microcomputer Raspberry Pi. The main difference between the two is that the Raspberry is a fully functional computer, complete with an OS, while the Arduino is not. Instead, the Arduino is more hardware-oriented and it allows the user to control any connected devices more directly without going through an OS [43]. However, only one member of the group had significant hardware experience, so ultimately the Raspberry Pi was chosen over the Arduino for its greater user friendliness.

5.1.3 Choosing the Robot

By choosing the Raspberry Pi as our hardware platform the requirements we had for the potential robot were the following:

- It should be compatible with Scratch
- It should be compatible with the Raspberry Pi
- It should be able to move around
- It should be possible to attach more hardware like: sensors, lights or screens

From these requirements we quickly found the GoPiGo and chose it as the base for our robot.

5.2 Modification of Scratch 2.0

Primarily, since Scratch 2.0 is built to control images on a screen and not a GoPiGo robot, which would need custom blocks for controlling the robot hardware, it was deemed necessary to modify Scratch 2.0. Furthermore, some graphical user interface elements of Scratch 2.0 is not applicable to use with the robot and could therefore preferable be removed. The most noticeable element being the panel displaying the graphics of a running Scratch program.

Version 2.0 was chosen to be modified since this is the version available open-source; available in a repository at Github (see Appendix C for link to the repository). The source code is written in ActionScript which means that the compiled source code will be in the form of an *.swf* file and require the Adobe Flash Player to be run. How to compile the source code is described in Appendix D. The modification of Scratch will henceforth be referenced to as *the PedaGoGo platform*.

5.2.1 Theory Behind Adding and Removing Blocks

Adding blocks in Scratch is not as trivial as it could be but once you know the procedure it is relatively easy. First of all, every block is defined in the ‘Specs.as’ file, located in the root src folder, in an array of commands. For all blocks that are displayed to the user, there is a command located in this array. To remove a block you simply remove the corresponding command in the array.

Each command is in turn an array of strings, where each string define one of the following: block text, type of block, (which are both described in technical background, section 2.1.2) category, the id and finally optional default values, if the block should need any.

The second thing that needs to be defined to add a block is something called the “primitive” of each block. The primitive determines what is performed when a Scratch project is run in the Scratch player. What will happen during the execution of each block is defined by a function for each block in one of several files, located in the "primitives" folder, which are named by the different categories and the content accordingly. For the primitive “primExample”, a function is defined as follows:

```
private function primExample(b:Block):void {
     <code for block>
}
```

For the function to be run when it is supposed to, each function needs to be added to a dictionary, which in essence is a list containing keys and data. Given the key you can get the data. The key in this case is the id string mentioned earlier in this section.

The syntax for adding an element to the dictionary is:

```
primTable[‘exampleKey’] =  primExample;
```

This is done in the “addPrimsTo” function in the beginning of each file in the "primitives" folder mentioned earlier in this section. This is the final step in making your own blocks.

5.2.2 Adding and Removing Blocks

For the project, the only blocks needed were ones that could be used together with the GoPiGo, so a lot of blocks had to be removed, and new blocks had to be created.

First and foremost, everything that affected the graphics was removed, such as the ‘Pen’ blocks or blocks that affected the appearance of the sprite. Some motion blocks also had to be removed because of the physical limitations that apply when you no longer control an image on a computer screen but instead a physical object. Some examples of this is “Set x to y” or “Go to mouse pointer”. In the absence of a monitor for the GoPiGo, blocks for displaying and modifying text were also removed because they are superfluous.

New blocks were added that could control the functionalities of the GoPiGo (described in technical background, section 4.3.1). In summary, blocks to control: driving forwards and backwards, turning left or right, the LEDs and the speed.

5.2.3 Integrating PedaGoGo into a webpage

In order to increase the availability of the PedaGoGo platform and using web design to create tutorials to facilitate the early usage of the application, a website was made containing a Flash player running the PedaGoGo platform along with a tutorial panel. The original Scratch 2.0 had an online mode, which meant that Scratch exclusively showed the built in player with the workspace completely hidden and inaccessible. This online mode was disabled in our modified version which allows it to work exactly like running it locally on your machine.

5.3 Modification of the Scratch 2.0 GUI

The GUI consists of several panes all built in a Flash player that is displayed on a webpage (described in technical background, section 4.1.1). The initial plans for the GUI were quite ambitious. However, it proved to be an unreasonably difficult and time consuming task to make even the smallest changes in the Scratch GUI. The reason for this was primarily that the source code consisted of a lot of strange dependencies. This meant that changing one small aspect of the GUI consisted of going through a large amount of files that did not have a very logical connection. Beyond that, building with Gradle meant that no error messages would show if something did not work, instead the webpage just showed up blank when loading; in essence, debugging was impossible to do.

Due to these difficulties the changes made were either the removal or slight modification of something. The whole left ‘stage’ pane seen in figure 5.2 and ‘sprite list’ were removed, since the user is creating programs to control a robot and not sprites as stated in previous section. The sound and drawing tabs were also removed since the GoPiGo neither plays sound nor can display images.

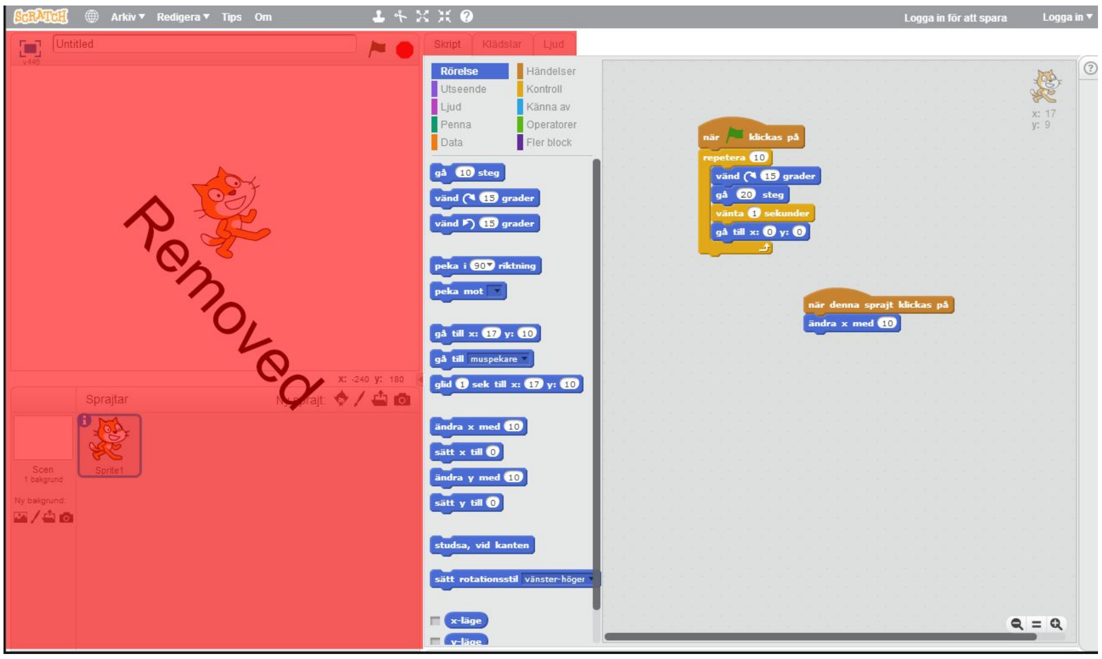


Figure 5.2: Original Scratch 2.0. The highlighted pane was removed in the PedaGoGo Platform.

5.4 The Robot

The robot's most important purpose is to provide physical aid to the programmer. A child 10-12 years old will be hovering between the concrete and formal operational stage of learning development (see related work & research, section 3.1), and most likely belong to the mature stage of the concrete operational stage. This greatly motivates the use for a physical aid since children at this stage have difficulty understanding abstract and theoretical concepts.

The goal is to make a child associate with the learning process and to learn programming easier by creating '*PedaGoGo-land*'. Meaning that by making as much of their environment - both on the computer and in the real world - as possible connected with programming they will naturally associate more with it. A child will find greater meaning in learning something if they can see a specific need to learn it. To be able to manipulate your own immediate environment the way you wish to is the perfect need. Although there does not seem to be any concrete evidence that prove that children will learn better through interaction with robots, many seem to agree that it is an excellent tool [28].

5.4.1 Setting up the robot

To assemble the robot, the first step was to put together the base, the GoPiGo (See Appendix C for a link to description). The next step was to connect the Raspberry Pi, this was done by inserting 26 of the GPIO pins on the Pi into the corresponding sockets present on the GoPiGo board. After these simple steps had been taken the connection between the GoPiGo and Raspberry Pi was established.

In order to control the GoPiGo, the 'Raspbian for Robots' OS (see technical background, section 4.3.2, for more information) was installed on the microSDHC card of the Raspberry Pi. A WiFi-dongle was inserted into one of the USB-ports on the Pi, and an ssh-connection was established between the team's laptops and the Pi. Since it was both convenient that the Raspberry would keep the same IP-address between connecting and disconnecting from the WiFi, and to keep it safe from possible harm like hackers or malware we wanted to establish a small, private WiFi-network. The dongle was supposed to be capable of creating this, but at first this worked very poorly and later not at all. The network was instead established by using a cellphone as a tethering hotspot.

From this connection, work could begin on the Raspberry. Simple control scripts in Python were initially programmed to maneuver the robot, these scripts could run sequences of commands to steer the robot like an R/C car. However, the robot was supposed to run PedaGoGo project files and thus there needed to be a Python control script that could read those project files and interpret the commands for the GoPiGo, also known as an "interpreter". The interpreter will be elaborated on later in this chapter, section 5.5.

5.4.2 Adding a Keypad

In order to facilitate an easy way to start and stop a script on the robot as well as allowing the user to be able to run different kinds of scripts, although not simultaneously, it was decided to add a keypad with different buttons to the robot. Initially, the plan was to have the keypad located on a touchscreen connected to the Raspberry Pi, but the screen required too many GPIO pins (26, there were 14 remaining pins on the Pi after connecting the GoPiGo). Although the amount of pins can be extended externally it required more advanced hardware components.

Instead it was decided to connect four differently colored *push switches*: red, blue, green and yellow. Where the blue, green and yellow ones should be able to run different scripts when pressed. Disparately from the rest, the red button should work as an interrupter and stop any running scripts, without turning off the robot. These hardware buttons needed to be manually built on the robot and connected to the Raspberry Pi.

One button uses up one I/O pin and each button had to be connected to a separate pin, thus using up a total of four additional I/O pins on the Pi. These pins were programmed to work as inputs and to use the *pull-up resistor* (see technical background, section 4.2.3 for additional information about the pull-up resistor). Furthermore, the pins were programmed to listen for a high to low change in voltage; pushing the button should bring about this change.

To get a default high signal, a 3.3V pin was connected through a 10kΩ resistor onto the GPIO pin. This circuit was then connected to a button which in turn was grounded, which will nullify the 3.3V when the button is pressed as seen below in figure 5.3.

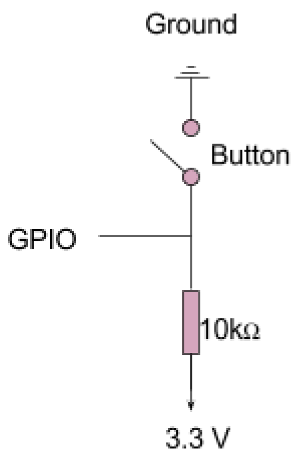


Figure 5.3: Circuit diagram for a button connected to one GPIO pin

It should be noted that the Raspberry Pi has the pull-up resistor hardware and functionality fully integrated, meaning that one could simply connect a button to only the GPIO pin and the ground pin and it would work. However, due to severe Electromagnetic Interference (EMI) in the circuit it was determined to include the connection to a power source via a resistance, with the intention of making the circuit more stable.

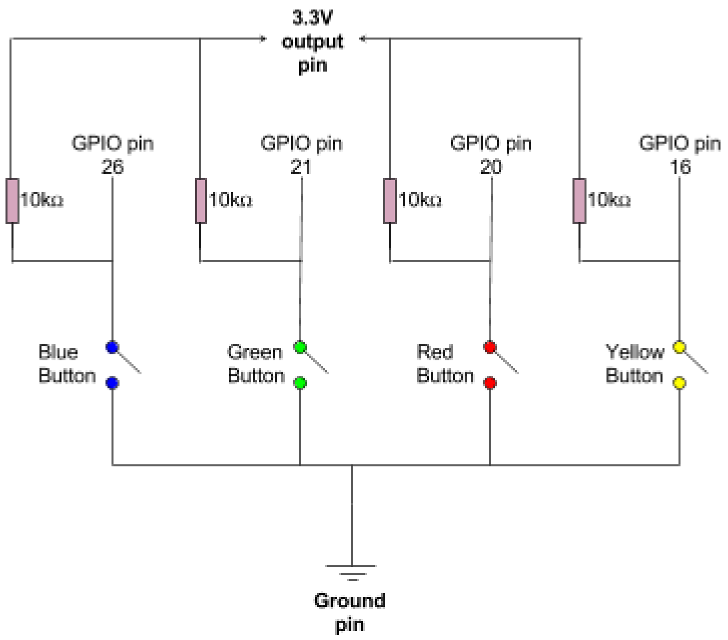


Figure 5.4: The circuit plan for the keypad.

This was all done by soldering the four buttons, following the circuit plan, shown in figure 5.4, on a breadboard. To further decrease EMI the wires to the ground pin and GPIO pins were intertwined as seen in figure 5.5.

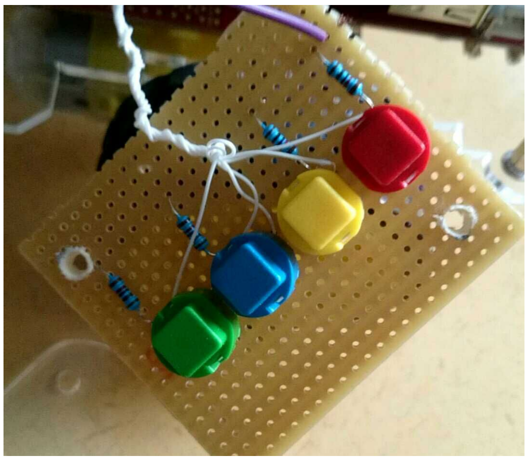


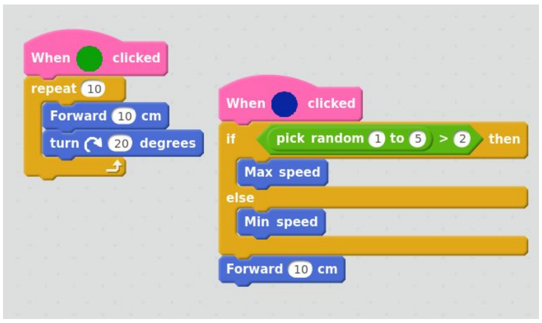
Figure 5.5: The four buttons soldered on a breadboard with intertwined wires (white).

5.5 Interpreter

To execute a PedaGoGo project you must interpret the data generated from the PedaGoGo platform and decide what should happen for each block. This is done single-handedly by a script written in the Python programming language. The reasoning why Python was chosen can be found in the Technical Background section 2.2.4.

5.5.1 PedaGoGo Project Save Files

When a project from the PedaGoGo platform is saved, the project is compressed into a ZIP archive. This archive consists mainly of a .json file that neatly contains all the necessary data needed to run the PedaGoGo project on the robot. For the PedaGoGo code represented in Figure 5.6a below, a part of the .json file where the blocks are defined is presented in Figure 5.6b.



(a) A PedaGoGo project in script area

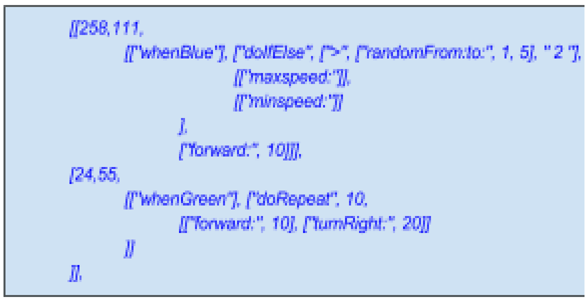

```
[[258,111,
        [["whenBlue"], ["doIfElse", [">", ["randomFrom:to:", 1, 5], "2 "],
                        [["maxspeed:"]],
                        [["minspeed:"]]
                ],
                ["forward:", 10]]],
[24,55,
        [["whenGreen"], ["doRepeat", 10,
                [["forward:", 10], ["turnRight:", 20]]
        ]]
]],
```
(b) The same PedaGoGo project as shown in (a) in raw form in the .json save file.
Every element represent a block and consist of x, y values and a script.

Figure 5.6: A PedaGoGo project in editor (a) and raw data (b).

Looking at the data in figure 5.6b seen above, it is clear that each chunk is composed of an x and a y value, representing where on the graphical workspace they are located. This is followed by a hierarchically structured set of commands starting with the block located first in the chain. Under normal circumstances, the first block is a so called “hat block”, which is one of the block types described in the technical background, section 4.1.2.

The hierarchical structure makes it very easy to access the various parts of the project. To locate a part that belongs to a button-click event, it is enough to go through the first element from each chunk of blocks and examine the "hat blocks” until you locate the right colored button, which will be the chunk you are looking for.

5.5.2 Elements of the Interpreter

The Python script that interpret the PedaGoGo project files consists of some key functions which break down the chunks of blocks and then executes each block in the correct order. The more important functions of the interpreter, that will be discussed in this section are: *buttonPress, executeChunkOfBlocks and executeBlock*

Running a chunk of blocks is done by the function called “executeChunkOfBlocks”, that goes through the chain of blocks derived from a button click. Each block is then sent to another function called “executeBlock” which reads the block signature and performs the defined action for the specified block.

The PedaGoGo project save file can also contain global variables and lists which are saved in the same manner as the chunks of blocks in the .json file. The Python script will in the initialization phase, set up any global variables or lists by adding them to dictionaries. Dictionaries are a data structure used to be able to have an unknown number of variables and lists without knowing their names in advance.

After this initialization is done the program simply waits for a button press. When a button press occurs, the program stops listening to other button presses to ensure that only one function will be performed at the same time, while it executes the function connected to the button press. However, it is always possible to press the red interrupt button in order to interrupt an ongoing script execution.

To give a concrete example on running a project on the GoPiGo interpreter, the program in figure 5.7 is executed as follows:
The procedure is started by pressing the green button on the GoPiGo. This event will in turn perform the function “buttonPress” which tries to finds the chunk of blocks that starts with the *green button hat-block*.

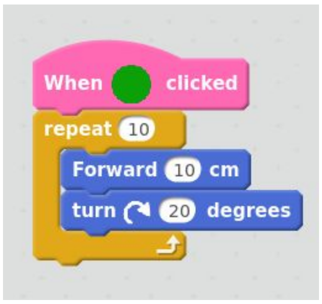


Figure 5.7: An example of a PedaGoGo script.

If this chunk is found, these blocks will be sent to the function “executeChunkOfBlocks”. This function then goes through each chunk of blocks and executes them with the function “executeBlock”. In this instance, the function “executeBlock” will receive a hat-block and a repeat-block, the repeat-block in turn contains a pair of stack-blocks. Since hat-blocks do not serve as anything but a startmarker, the hat-block will be “ignored”. However, the repeat-block yields the function “executeBlock” to recursively call the function “executeChunkOfBlocks” 10 times with the smaller chunk of blocks only containing the “forward” and “turnRight” blocks.

5.5.3 Running a PedaGoGo Project on the Robot

At first, running a PedaGoGo project on the robot had to be done manually by transferring the created .json project file over the ssh-connection and then using the interpreter on the code through commands in the terminal. This procedure was not very user friendly, especially not for a child, and it was decided that this process should be possible to complete without using the terminal or the Raspberry Pi interface in any way. Instead the process should seem almost completely hardware oriented to the user.

The process was planned and implemented to go as follows:

User	**System**
Saves their project on a USB flash drive as “drive.sb2”.	
Inserts the flash drive into the Raspberry Pi .	
Pushes the red button on the PedaGoGo robot.	
	Notices that a USB device has been connected.
	Run the interpreter script, the interpreter will attempt to read the “drive.sb2” file.
	When script is done: light up the two LEDs on the PedaGoGo robot, indicating it is ready to use.
	Waits for button press.
To run scripts: press the corresponding colored button.	

5.6 Help Panel

An important aspect to this project was to ensure that getting started with programming the GoPiGo would be easy and that the users would quickly see results as a way to prevent the loss of interest. In order to do this we created a host of tutorials to help get them started and a help panel to provide the users with enough basic knowledge to achieve their goals. This help panel was then implemented in HTML on the PedaGoGo webpage.

5.6.1 Project-Based Learning

In the article "Motivating Project-based learning: Sustaining the doing, supporting the learning", the authors argue that projects have the potential to help people learn [44]. They explain that by breaking down tasks to teach strategies for thinking...

> "*...learners are motivated to persist at authentic problems, meld prior knowledge and experience with new learning, and develop rich domain-specific knowledge and thinking strategies to apply to real-world problems.*" [44 (p. 371)]

According to the article, in order to ensure the students' motivation, certain factors must be considered when designing these projects or tasks. It is important, for instance, whether or not the students find the projects interesting, whether they feel that they have the competence to complete the tasks and whether they focus on learning rather than outcomes.

Some examples of ways to ensure that student interest is enhanced are to make sure that the tasks are varied and include novel elements, the problems are authentic and have value, and that the tasks are challenging and that there is closure so that an artifact is created.[44 (p. 375)]
The authors then explain that

> "*In doing projects, the students need access to information and examples or representations that will help them to understand and use central ideas." [44 (p. 378)]*

5.6.2 Designing the Help Panel

The information presented in the previous chapter was the base for the design process of the help panel for the PedaGoGo. The help panel is divided up into three sections; Getting Started with PedaGoGo, Uploading your Program and Tasks. The first section, Getting Started with PedaGoGo, describes how to start creating a program and gives a brief description of the different blocks and various programming concepts such as loops, if-statements and lists. The next section, Uploading your Program, gives a description of how to transfer the finished program to the GoPiGo and how to save the files correctly. The last section of the help panel is Tasks. This section, just as the title suggests, is a collection of tasks to complete. All together there are twelve tasks divided into two groups; Motion and Lights. The tasks start off very simple and gradually increase in complexity whilst making use of knowledge gained in previous tasks to make the tutorial as pedagogical as possible. These tasks are, of course, optional and do not necessarily need to be completed in order, since children who are already familiar with Scratch may find the earlier tasks too simple and thus, uninteresting. The intention of the tasks is, after all, to motivate the users to learn.

As mentioned above, the tasks section start off with very simple tasks in order to appeal to less experienced or novice programmers. The level of difficulty of these tasks were evaluated during a field study that will be described in the next chapter, Validation. It was after this field study that the perceived ease of the tasks could be noted and changed to match the level of knowledge for the specified age group. The tasks then become increasingly more difficult while repeating certain aspects of the previous tasks to keep the users motivated to learn more and more complicated concepts. The last task, in both Motion and in Lights, is a prespecified course to be completed, incorporating all the aspects of the previous five tasks.

6
User Testing

In this chapter we will describe the details of the two field studies, conducted in order to evaluate the product. The first study was at an elementary school in Gothenburg and the second was at the non-profit organisation Kodcentrum, also in Gothenburg. The studies were conducted in two different ways but with the same purpose, to gather feedback on the PedaGoGo platform, and robot, from users of the specified target age. The first study examined the perceived level of difficulty and enjoyment of our product for children who had little or no previous experience with Scratch, or computer programming in general. The second study examined the perceived levels for children who had more significant experience.

6.1 Field Study at Elementary School

This study took place at the elementary school during a two hour period. The participants were chosen by the class teacher prior to the study and were all eleven or twelve years old. We had requested a total of six students to participate and specified that three boys and three girls would be ideal, as we were interested in observing any potential differences between the genders during the study.

The aim was to investigate the perceived levels of difficulty and enjoyment of the PedaGoGo, but also to investigate the participants general perception and interest in programming. In order to properly evaluate the study, we wanted a way to compare what the students thought about programming before testing the PedaGoGo, with what they thought after testing. To this end, two surveys were prepared for the participants to fill in, one at the beginning of the study and the one at the end. The surveys will be described in more detail later on.

6.1.1 The Study

The study was conducted in pairs which the students chose themselves, with one girl and one boy per pair. As the participants had used Scratch once previously, a longer introduction on how to build scripts was not necessary, however a brief presentation of the PedaGoGo blocks and the GoPiGo was provided.

At this stage the help panel was not yet complete, and could therefore not be a part of the evaluation. However, the study was based on the existing plans for the tasks provided in the help panel (see Implementation, section 5.6), and so the results could still provide some feedback regarding the level of difficulty for the tasks.

The participants were first presented with a few relatively simple tasks to complete. These tasks were:

- Drive in a square shape
- Drive in a triangle shape
- Make both the lights blink four times
- Make the lights blink every other time, four times

After completing these tasks, the participants were asked to create a script for a prespecified track, incorporating aspects of the previous tasks as well as variations of speed. As the study was limited to two hours, we did not include more complicated concepts such as variables or if-statements. However, the participants were introduced to loops as they were instructed to limit the number of blocks by making use of this concept in their final scripts.

6.1.2 The Surveys

The first survey aimed to investigate what the participants general perception of programming was and what their expectations of the study were. Aside from filling in their age and gender, the participants were asked what they knew about programming, had they done it before, if they expected it to be easy or difficult, fun or boring and whether they thought that programming could be helpful when learning other school subjects. The first questionnaire consisted of the following questions:

- Are you a boy or a girl?
- How old are you?
- What do you know about programming? (How does it work? What do you do when you program et cetera)
- Do you have previous experience with programming?
- How difficult or easy do you think it is to program? Do you think it is fun or boring?
- Do you think, if you learn programming, that it can help you in other school subjects? If so, which?
- What do you think about robots?
- What do you feel about programming a robot(difficult / easy, fun / boring, et cetera)

The second survey was quite similar to the first, as one of the aims was to observe any changes in the results after testing the PedaGoGo. This time the participants were asked what they had learned, whether or not they had enjoyed programming, whether or not it had been difficult and what they thought of the GoPiGo. We also asked whether they would like to do more programming and whether they would like to learn it in school to see whether or not our product had managed to spark a greater interest. The second questionnaire consisted of the following questions:

- Are you a boy or a girl?
- How old are you?
- What have you learned about programming(how does it work? What do you do when you program? et cetera)
- How difficult/easy do you think it was to program? Was it is fun or boring?
- Do you think now that programming can help you in other school subjects? If so, which?
- How was it to program the robot? difficult / easy? fun / boring? et cetera¨
- Would you like to program again?

- Would you like to learn programming in school?

These surveys can be found in Appendix B and the results are presented in Results section 4.

6.2 Kodcentrum

The second study was conducted at Kodcentrum, a non-profit organization founded in 2014. Kodcentrum offers free lessons to children between nine and thirteen years of age in several different locations in Sweden, with the purpose to introduce children to computer programming [45].

Kodcentrums lessons are one hour long and so our study was conducted during this one hour period. Out of the eight children regularly attending these classes, five children were present the day of the study and three of these showed an interest in testing the PedaGoGo. This group consisted of one girl, aged nine, and two boys aged ten and twelve.

This study was less scientific in nature as the aim was to observe how more experienced programmers perceived the PedaGoGo. The focus was only on evaluating the product and therefore no surveys were prepared. Instead, the participants provided us with oral feedback at the end of the study. We were interested in the perceived level of enjoyment and also what the participants thoughts were on the help panel, which was completed at this stage.

6.2.1 The Study

As Scratch had been the main tool during Kodcentrums lessons, no introduction was needed other than to explain how the GoPiGo works with the PedaGoGo platform. Also due to the participants familiarity with Scratch, we did not provide any tasks for them to complete. Instead we were interested in observing how the children interacted with the PedaGoGo when being allowed to experiment freely, creating any scripts that came to mind.

At the end of the study, the participants were asked whether or not they had enjoyed the PedaGoGo, whether they had any ideas for improvement, how had they perceived the help panel and they were also asked what they thought of the experience compared to the programming they had done before.

The results of this study are presented in results, section 7.4**.**

7
Results

The final result includes a modification of Scratch called the PedaGoGo platform, the modified GoPiGo called the PedaGogo robot and the help panel for the PedaGoGo platform. These will be presented below along with the results of the validation.

7.1 The PedaGoGo Platform

The software used to program the PedaGoGo robot is a graphical user interface. The software can be found combined with a help panel as a web application online (see Appendix C for link).

7.1.1 The Interface

The interface used for programming the PedaGoGo consist of four main elements, seen in figure 7.1. Firstly there is a *block palette* containing the blocks used to program. Then there is the *script area*, where the programmer performs the coding by placing blocks from the palette. Another element in the interface is the *help panel* which is there to assist the user with getting started with the robot and the new language as well as to provide several challenging tasks. Lastly there is a *top bar* where the user can create, save and load projects as well as change the language and customize the block colors to your preference. The user can also use the cut and copy tools found here to manage the blocks in the *script area*.

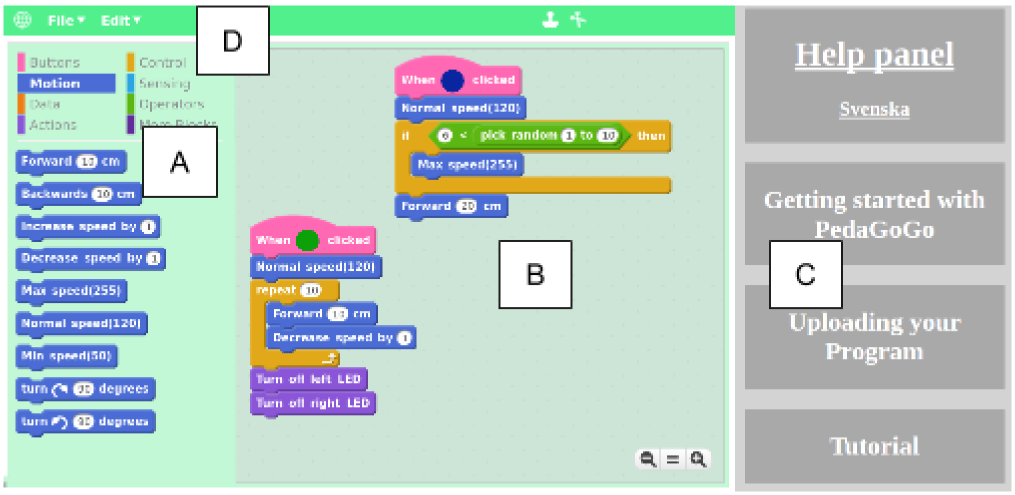
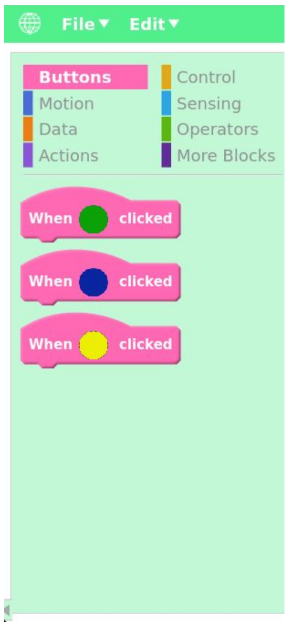


Figure 7.1: The PedaGoGo graphical user interface.
The letter A shows the *block palette*, B is the *script area*, C is the *help panel* and D is the *top bar*.

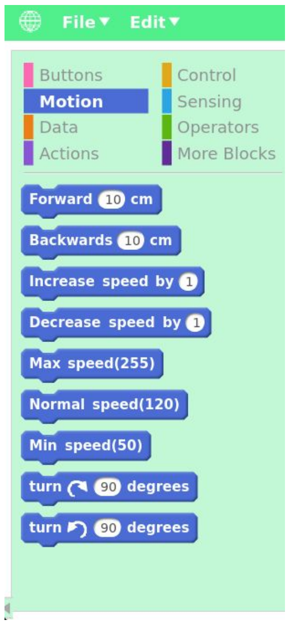
7.1.2 The Block Categories

The blocks used for programming are divided up into eight different categories, as shown in Table 7.1 below. These categories help to ease the finding of a particular block when scripting.

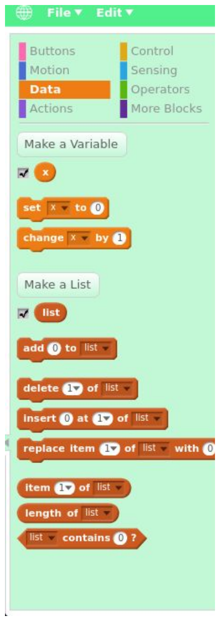
Table 7.1: The images a-h show the eight different categories with their blocks in The PedaGoGo Platform.



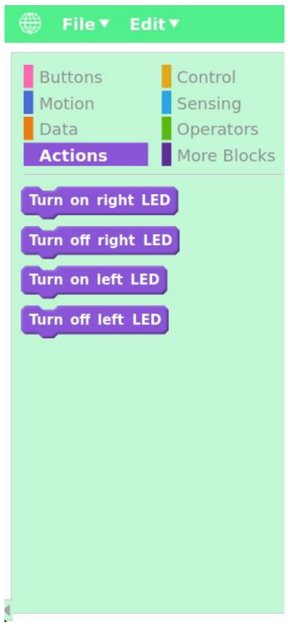
(a) Buttons category.



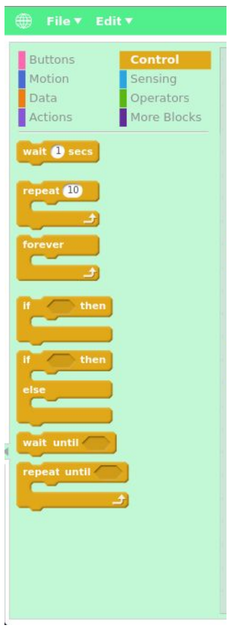
(b) Motion category.



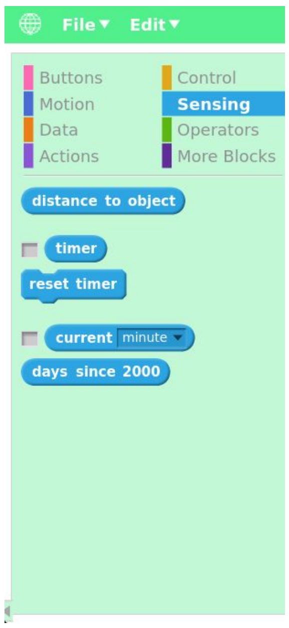
(c) Data category.



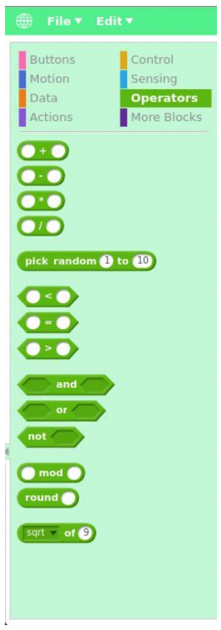
(d) Action category.



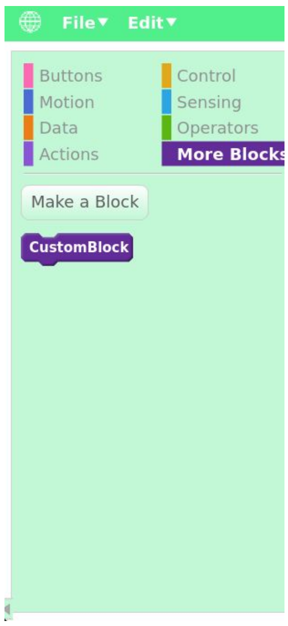
(e) Control category.



(f) Sensing category.



(g) Operators category.

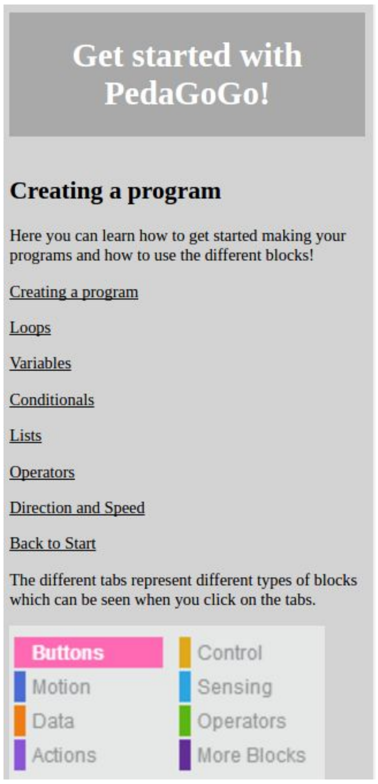


(h) More Blocks category.

7.1.3 Help Panel

The PedaGoGo platform has a built in help panel (Letter C in Table 7.2) which consist of three major subgroups show in Table 7.2. First there is a theoretical part on how to create a program and how to use the platform to control the robot. Secondly there a step by step guide on how to upload projects created by the software to the PedaGoGo robot. And lastly the tutorial, with a lot of varying tasks for the user to complete if one wishes so.

Table 7.2: The three different parts of the help panel



Get started with PedaGoGo!

Creating a program

Here you can learn how to get started making your programs and how to use the different blocks!

Creating a program

Loops

Variables

Conditionals

Lists

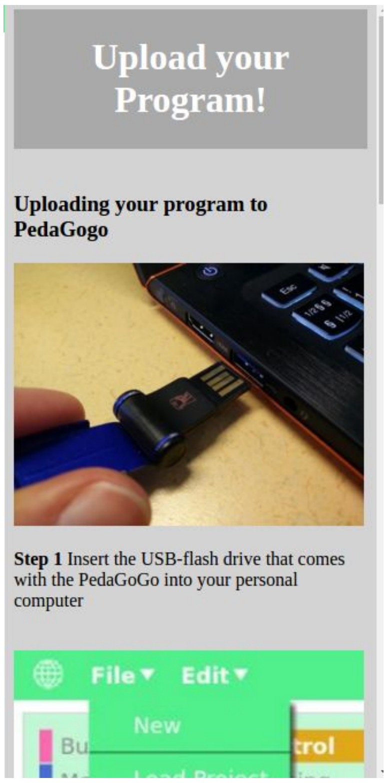
Operators

Direction and Speed

Back to Start

The different tabs represent different types of blocks which can be seen when you click on the tabs.

(a) Get started with PedaGoGo

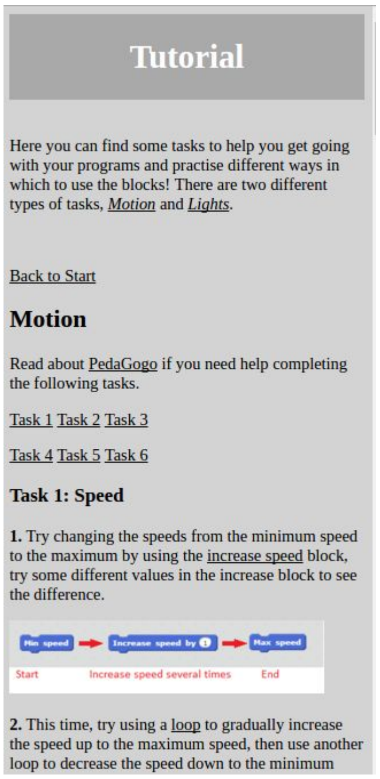


Upload your Program!

Uploading your program to PedaGogo

Step 1 Insert the USB-flash drive that comes with the PedaGoGo into your personal computer

(b) Upload your Program



Tutorial

Here you can find some tasks to help you get going with your programs and practise different ways in which to use the blocks! There are two different types of tasks, *Motion* and *Lights*.

Back to Start

Motion

Read about PedaGogo if you need help completing the following tasks.

Task 1 Task 2 Task 3

Task 4 Task 5 Task 6

Task 1: Speed

1. Try changing the speeds from the minimum speed to the maximum by using the increase speed block, try some different values in the increase block to see the difference.

2. This time, try using a loop to gradually increase the speed up to the maximum speed, then use another loop to decrease the speed down to the minimum

(c) Tutorial

7.2 The PedaGoGo Robot

The PedaGoGo robot is a programmable physical toy robot with a keypad for executing scripts written with the PedaGoGo platform. Presented in figure 7.2, the robot is composed of the Dexter Industries GoPiGo robot, a Raspberry Pi 2 Model B and four differently colored buttons. It is powered by 8 AA batteries and can run for several hours.

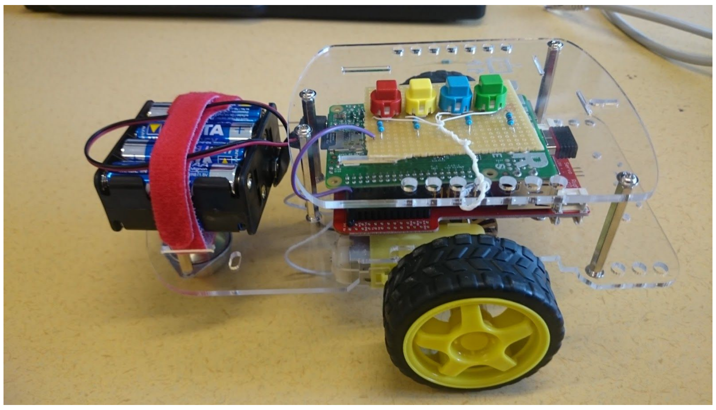
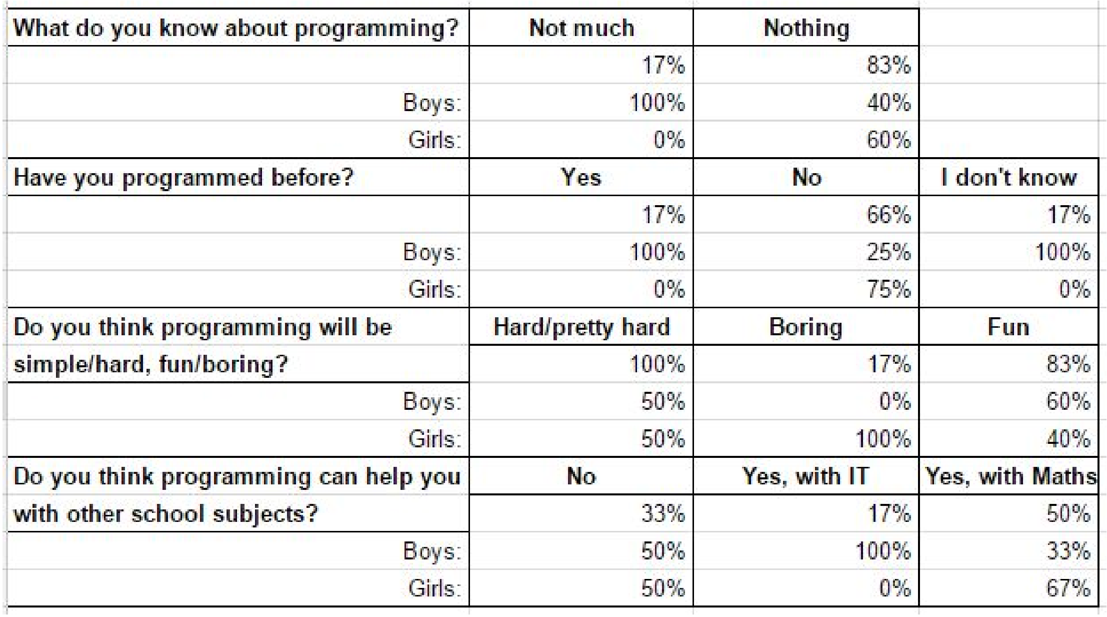


Figure 7.2: The PedaGoGo robot, motionless, waiting for input.

7.3 Field Study at Elementary School

As described in chapter 6, the participants of this study filled in two surveys, one before they tested the PedaGoGo platform and one after. The results of the surveys have been summarized below in Table 7.3 and 7.4 and the surveys can be found in full in **Appendix B.**

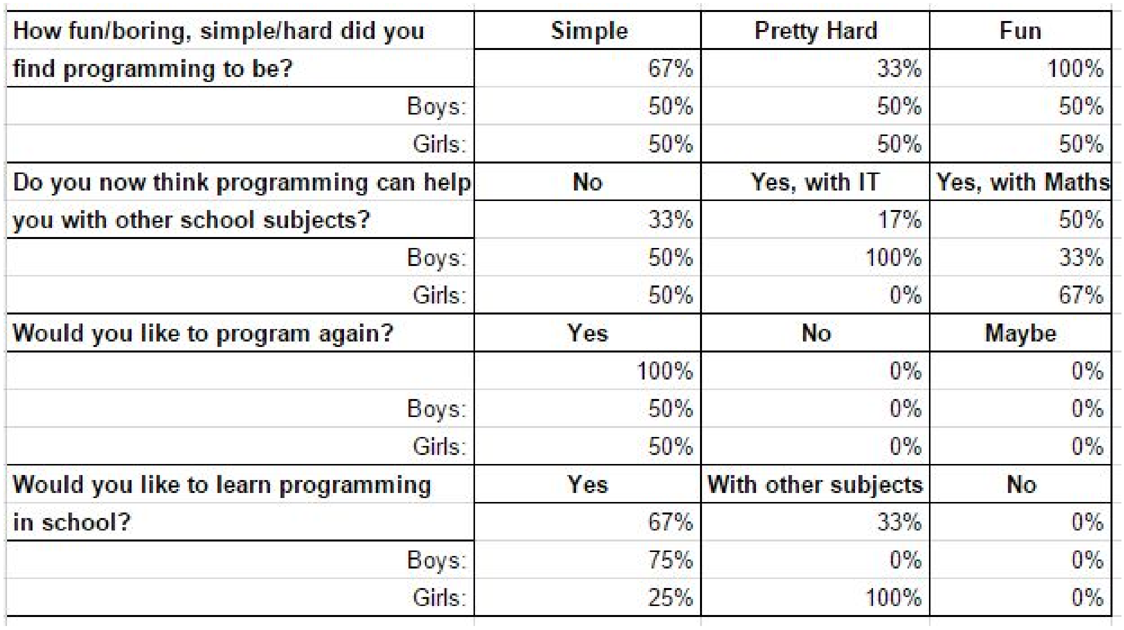
Table 7.3: Results of survey number 1. Boys: and Girls: refers to the percentage of the total that were boys or girls.



What do you know about programming?	Not much	Nothing	
	17%	83%	
Boys:	100%	40%	
Girls:	0%	60%	
Have you programmed before?	**Yes**	**No**	**I don't know**
	17%	66%	17%
Boys:	100%	25%	100%
Girls:	0%	75%	0%
Do you think programming will be	**Hard/pretty hard**	**Boring**	**Fun**
simple/hard, fun/boring?	100%	17%	83%
Boys:	50%	0%	60%
Girls:	50%	100%	40%
Do you think programming can help you	**No**	**Yes, with IT**	**Yes, with Maths**
with other school subjects?	33%	17%	50%
Boys:	50%	100%	33%
Girls:	50%	0%	67%

All of the participants stated that they knew nothing or very little about programming and only one boy was under the impression that he had programmed before. They all thought that it would be difficult but the majority believed it would also be enjoyable. A few of the participants recognized a connection to mathematics, even before testing the PedaGoGo, but some did not think programming could be connected to other school subjects. There were no significant differences between the genders.

Table 7.4: Results of survey number 2. Boys: and Girls: refers to the percentage of the total that were boys or girls



How fun/boring, simple/hard did you	Simple	Pretty Hard	Fun
find programming to be?	67%	33%	100%
Boys:	50%	50%	50%
Girls:	50%	50%	50%
Do you now think programming can help	No	Yes, with IT	Yes, with Maths
you with other school subjects?	33%	17%	50%
Boys:	50%	100%	33%
Girls:	50%	0%	67%
Would you like to program again?	Yes	No	Maybe
	100%	0%	0%
Boys:	50%	0%	0%
Girls:	50%	0%	0%
Would you like to learn programming	Yes	With other subjects	No
in school?	67%	33%	0%
Boys:	75%	0%	0%
Girls:	25%	100%	0%

After testing the PedaGoGo, all the participants stated that it had been enjoyable although the perceived level of difficulty varied slightly. They also all agreed that they would like to do more programming and to learn it in school. The opinions of the genders differ slightly on this matter as all the boys would like programming to be its own separate subject while the majority of the girls would like to learn it in conjunction with other school subjects.

7.4 Field Study at KodCentrum

As mentioned in chapter 6, the participants were briefly interviewed orally at the end of this study. When asked about how enjoyable the PedaGoGo had been, all three participants expressed that it had been very enjoyable. In fact, all three agreed that:

"It was a lot of fun, more fun than just using Scratch on it's own."

We also asked if they had any ideas of improvements for the PedaGoGo. One of the boys missed functions such as being able to measure the distance to other objects. The remaining two participants only missed a block for a "normal speed", somewhere in between the minimum and the maximum speed.

When we asked about the tutorial section the participants described it as confusing and that they had not understood the point of it. Only the eldest boy had looked at the tutorials to learn how to upload his finished scripts to the robot, the other two participants had dismissed the tutorials without any investigation.

Some pictures from this second field study are presented in Appendix B.

8

Discussion

This chapter will discuss the results of the project focusing on the results of the field studies, the use of Scratch for the PedaGoGo platform and the choice to use the GoPiGo as the physical aid. The discussion will end with a few suggestions of continuations of the project, in terms of future improvement.

8.1 Field Studies

One interesting observation made after the first field study was that even though most of the participants had encountered Scratch during a lesson, only one was under the impression that he had infact programmed before. This suggests that the others did not view Scratch as programming and as they all thought it would be difficult, this result serves to prove that society has a view of programming that needs to change. If we want more children to study programming or just gain some general knowledge of how technology works, they need to be introduced to a fun and creative way of learning to program. As all the participants of this study expressed a desire to learn more programming, we believe that the PedaGoGo was a successful tool in sparking this important interest in these children. Also, as no significant differences between the genders were observed, the PedaGoGo seems effective in attracting girls in an equal degree as boys.

However, only having one robot limited the field study somewhat and the relatively small number of participants may not be enough to prove that we have succeeded. Although, the previous studies mentioned in related work & research, section 3.5, together with our results, do suggest that using a physical aid is a successful approach to teaching programming to both boys and girls.

Another observation made was that the perceived level of difficulty varied somewhat. This could be due to a number of reasons. The first being the fact that the study was conducted in pairs. It was clear during the course of the study that in each pair, one member slightly dominated the process as only one person at a time can build the scripts. Had the participants programmed separately, perhaps the results would have been different as each participant would have had equal chance to try the PedaGoGo properly. The study was however, limited to a two hour period and so this was unfortunately not an option, as it would have taken too long to test all the scripts. Therefore a study conducted over a longer period of time, perhaps even during several separate occasions, would provide a more reliable result.

Another reason for this result could be due to the varying levels of mathematical knowledge of each participant, as a certain knowledge of geometry for instance, was needed to complete some of the tasks that they were presented with. Some of the participants struggled with the geometric aspects and needed a short introduction on calculating the angles of a triangle, while others managed this with ease. This could also be the reason why only a few thought that programming could be helpful when learning mathematics, as they learned to perform these calculations during the study while others, who did not think it would help them, already knew how to do this.

Overall, we feel that this was a positive evaluation as the PedaGoGo platform and robot were perceived as enjoyable. There was however, no measurement in gained knowledge noted since the surveys may not have been precise enough when asking the participants what they had learned. This question could have been more specific, perhaps asking the participants for example what a loop is and when do you use them. This type of question could have provided a better result in terms of the level of pedagogy of the product.

The second field study, at Kodcentrum, was a way of observing children who are experienced with Scratch and receiving their feedback. Because of their knowledge of Scratch it would have been difficult to say how much they learned from using the product and so this was not investigated. However, it was clear that even these experienced children enjoyed using the PedaGoGo as they did not stop testing new scripts during the entire study. The comments that it was more enjoyable to use the PedaGoGo and the robot, as opposed to programming with just Scratch, gives further proof that a physical aid is indeed a great way to learn at this age.

A first draft of the help panel was completed for this second study, however, since the participants already had the basic knowledge that the help panel provides, it was not evaluated properly. The children participating, did not give the panel much time, and described it as confusing since they did not understand why it was there. These comments helped shape the final design of the help panel in order to hopefully make it more clear and understandable.

8.2 The PedaGoGo Platform

Scratch was chosen because of its graphical nature and because a large amount of previous research involving Scratch for educational purposes, all yielded successful results. It was also chosen for the obvious practicality of it being open source, allowing us to modify the interface and the available blocks according to our needs. This made Scratch superior to other options in terms of us being able to implement the software in a way that would make our goals obtainable.

The creators of Scratch designed it for children between six and eighteen years of age, so theoretically, even younger children than our specified target group could learn from these the platform. However, during our study at Kodcentrum, one of the mentors explained how younger children often needed a finished script as a template in order to complete their given tasks. The reason for this is they do not understand the different concepts, such as variables and if-statements, needed to construct for example a game. Therefore, we believe that our target group, based on Jean Piaget's theories of cognitive development(described in related work & research, section 3.1), is the ideal starting age for learning to program, as the ability to think in more abstract ways is necessary for understanding the various concepts.

As Scratch is already broadly used by children of all ages, it makes it easy for young programmers to switch between Scratch and the PedaGoGo platform, combining knowledge of different types of projects and therefore contributing to a broader learning experience.

8.3 The GoPiGo

The decision to use the GoPiGo for the project was partly based on the fact that it interacts with a Raspberry Pi, which comes with extensive documentation and possibilities for new functions for the robot, and partly because one of the criteria we had for the robot toy was that it was able to move around. We feel confident in our decision to use the GoPiGo since it was so well received by the participants of our field studies. Although, not all of the functions we had planned for were implemented, leaving the robot with fewer programmable components than we had envisioned at the beginning of the project. These functions will be described in the following section.

Although we are overall pleased with the GoPiGo, one problem was discovered during the field studies; depending on which type of flooring the robot is tested on, its ability to turn is affected. Both studies were conducted on plastic flooring causing the wheels of the PedaGogo robot to glide and therefore not managing to complete the entire turns. To solve this problem, the robot needs to be heavier, and although placing a heavy object on top of the robot was not an ideal solution, it did solve the problem during the evaluations. This problem would of course need a better solution before further use of the robot.

8.4 Continuation of the Project

In the planning phase, several ideas were noted but unfortunately fell out of the scope of our project mainly due to limitation of time. One idea was to attach a touchscreen to the GoPiGo which would be used to personalize the robot and give it a face, almost like the small digital pet “Tamagotchi”. We had also planned to attach more lights, of different colours and sizes, to the GoPiGo, especially ones that can be viewed from above since the existing lights can only be viewed from behind and at a low angle. This would make it easier to observe the lights and note whether they are behaving as the user wishes. Another extension of the project that we did not have time to implement is the attachment of an ultrasonic sensor that measures the distance to various objects, allowing for programs that make the robot turn before hitting a wall for instance. Since this was one of our main ideas during the planning phase, we feel it is particularly unfortunate that we did not have time to implement any such sensors and that this would be a welcome extension of our project.

Another possible extension would be to improve the transfer of the PedaGoGo programs to the robot. Ideally this could be done using a bluetooth connection rather than with the USB stick. This would make it easier and faster to test the program in the robot, allowing for changes in the program without having to reload on the USB stick each time, which was confirmed by the participants as a slight hassle during the beginning of both of our field studies. However, this does help to ensure that only one program at a time can be transferred to the robot.

The GoPiGO is available in several different colours according to the user's preference. However, the relatively simple design of the robot could also be expanded, for instance, to look like an animal.

9

Conclusion

Computers are so widespread in today's society that almost everyone comes into contact with them several times a day. Due to the rapid growth of software development for all these computers, the need for skilled programmers is increasing at a dramatic speed. So how do you meet the high demand for more programmers?

We believe that programming must be introduced in a more user-friendly way to young people in order for more students to educate themselves in programming and thus meeting the demand. It is also important that people get some general knowledge of technology, as it continues to play a more important role in society. This is the purpose of the project, to be that first introduction that helps an individual, with little or no prior contact with programming, to grasp and find a meaning for programming with the aid of a physical toy.

The final result was a very well received robot and platform which fulfilled its purpose. All participants who tried it were pleasantly surprised and many thought it was much more fun than they had expected. The excitement over the robot was apparent, and at each of the field studies performed the participants played without breaks during the whole visit, which shows that our product generates interest. However, in order to prove that using the product is also a great way to learn, more extensive and thorough user testing needs to be conducted.

The participants without any prior programming experience did not give much proof to the theory that our product exceeded mere programming on a computer. The participants who had some prior programming experience on the other hand, appreciated the robot much more than their previous experiences programming without a physical aid, which indicates that the robot was a success.

References

[1] European Commission. (2015-09-25). “*Grand Coalition for Digital Jobs”* [Online]. (Date Accessed: 2016-02-22). Available: https://ec.europa.eu/digital-single-market/en/grand-coalition-digital-jobs

[2] Gender Disparity in STEM Disciplines: A Study of Faculty Attrition and Turnover Intentions, (Date accessed: 2016-05-27), Research in Higher Education, November 2008, Volume 49, Issue 7, pp 607-624, Available: http://link.springer.com/article/10.1007/s11162-008-9097-4

[3] M. Resnick *et al*. “Scratch: Programming for All”. *Communications of the ACM*, vol. 52, no. 11, pages. 60-67, November 2009, Available: http://web.media.mit.edu/~mres/papers/Scratch-CACM-final.pdf (Accessed: 2016-04-03)

[4] N. Brown *et al*. “Restart: The Resurgance of Computer Science in UK Schools”. *ACM Transactions on Computing Education,* vol.14, no. 9, pages. 9:1-9:22, June 2014 Available: http://dl.acm.org/citation.cfm?doid=2642651.2602484 (Accessed: 2016-04-19)

[5] T. Bell, P. Andreae and A. Robins. “A Case Study of the Introduction of Computer Science in NZ Schools”. *ACM Transactions on Computing Education,* vol. 14, no. 10, pages. 10:1-10:31, June 2014 Available: http://dl.acm.org/citation.cfm?doid=2642651.2602485 (Accessed: 2016-04-22)

[6] H.Knutsson. (2015-09-24). “*Uppdrag att föreslå nationella it-strategier för skolväsendet”* [Online].Utbildningsdepartementet. (Date Accessed: 2016-02-22). Available: http://www.skolverket.se/polopoly_fs/1.240546!/Menu/article/attachment/U2015-04666-S_Nationella_it-strategier.pdf

[7] Webbstjärnan. (2013-06-13). *“Ska programmering finnas på skolschemat?”*[Online]. (Date Accessed: 2016-02-22). Available: https://www.webbstjarnan.se/blogg/ska-programmering-finnas-pa-skolschemat/

[8] S. McLeod. (2015) “Jean Piaget,” *Developmental Psychology* [Online]. (Date Accessed: 2016-02-22). Available: http://www.simplypsychology.org/piaget.html

[9] J.Piaget, *“The Origins of Intelligence in the Child”* , New York, NY, International Universities Press, 1952

[10] S. McLeod. (2010) “Concrete Operational Stage,” *Developmental Psychology* [Online]. (Date Accessed: 2016-03-08). Available: http://www.simplypsychology.org/concrete-operational.html

[11] S. McLeod. (2010) “Formal Operational Stage,” *Developmental Psychology* [Online]. (Date Accessed: 2016-03-08). Available:
http://www.simplypsychology.org/formal-operational.html

[12] K.C. Powell and C.J. Kalina. “Cognitive and Social Constructivism: Developing Tools for an Effective Classroom”. *Education*, vol 130, no.2, 2009, pages 241-250 (Accessed 2016-04-26)
Available:
http://search.proquest.com/docview/196408006/fulltext/3262263C6F3F4816PQ/1?accountid=10041

[13] Seymour Papert’s Legacy: Thinking About Learning, and Learning About Thinking (Accessed: 2016-04-15)
https://tltl.stanford.edu/content/seymour-papert-s-legacy-thinking-about-learning-and-learning-about-thinking

[14] S.Papert, *“Mindstorms: Children, Computers and Powerful Ideas”*, New York, NY, Basic Books, Inc, 1980

[15] Scratch. *“About Scratch”* [Online]. (Date Accessed: 2016-03-15). Available:
https://scratch.mit.edu/about/

[16] Scratch [Online] (Date Accessed: 2016-02-10). Available: https://scratch.mit.edu/

[17] Brennan, K., & Resnick, M. (2012) “*New frameworks for studying and assessing the development of computational thinking”,* [Online] (Date Accessed: 2016-04-29). Available:
http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

[18] David J. Malan, Henry H. Leitner (2007) “*Scratch for budding computer scientists”* ACM SIGCSE Bulletin Volume 39 Issue 1, March 2007, Pages 223-227. Accessed: 2016-04-08.
Available: http://dl.acm.org/citation.cfm?id=1227388

[19] Orni Meerbaum-Salant, Michal Armoni & Mordechai (Moti) Ben-Ari (2013) “Learning computer science concepts with Scratch”, Computer Science Education
Volume 23, Issue 3, 2013, pages 239-264

[20] B. Choi, J. Jung & Y. Baek. In what way can technology enhance student learning? : A preliminary study of Technology Supported learning in Mathematics. In R. McBride & M. Searson (Eds.), Proceedings of Society for Information Technology & Teacher Education International Conference 2013 (pp. 3-9). Chesapeake, VA: Association for the Advancement of Computing in Education (AACE) (Date accessed: 2016-04-21) Available:
http://www.editlib.org/p/48061/proceeding_48061.pdf, http://www.editlib.org/p/48061/?nl=1

[21] Kylie A. Peppler, Yasmin B. Kafai (2005) “*Creative Coding: Programming for Personal Expression”* [Online] (Date Accessed: 2016-04-21). Available:
https://download.scratch.mit.edu/CreativeCoding.pdf

[22] Scratch Wiki "*Hardware that can connect to Scratch"* [Online] (Date Accessed: 2016-04-20)
Available: https://wiki.scratch.mit.edu/wiki/Hardware_That_Can_Connect_to_Scratch

[23] Barbara Ericson (2015) "*WeDo Activities with Scratch for Elementary and Middle School kids",* Scratched [Online] (Date Accessed: 2016-04-20) Available: http://scratched.gse.harvard.edu/resources/wedo-activities-scratch-elementary-and-middle-school-kids

[24] Lego Mindstorms Ev3. [Online] (Date Accessed: 2016-05-03) Available: http://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313

[25] FinchRobot [Online] (Date Accessed: 2016-05-09) Available: http://www.finchrobot.com/

[26] Wonder Workshop [Online] (Date Accessed: 2016-05-09) Available: https://www.makewonder.com/

[27] "Universitets- och högskolerådets antagningsstatistik" [Online] (Date Accessed: 2016-02-04) Search: HT2015, Chalmers Tekniska Högskola, Bara Program. Available: http://statistik.uhr.se/

[28] Monica M. McGill (2012) "Learning to Program with Personal Robots: Influences on Student Motivation", ACM Transactions on Computing Education, Volume 12 Issue 1, March 2012, Pages: 4:1 - 4.32, Available: http://dl.acm.org/citation.cfm?doid=2133797.2133801 (Accessed: 2016-04-19)

[29] Miguel A. Rubio, Rocio Romero-Zaliz, Carolina Mañoso, Angel P. de Madrid (2015) "Closing the gender gap in an introductory programming course", *Computers and Education,* Volume 82, Pages: 409-420 [Online]. (Date Accessed: 2016-04-11) Available: http://www.sciencedirect.com/science/article/pii/S0360131514002802

[30] Scratch Wiki. *"Scratch"* [Online]. (Date Accessed: 2016-03-15). Available: http://wiki.scratch.mit.edu/wiki/Scratch

[31] Scratch Wiki. (2016-02-12). *"Blocks"* [Online]. (Date Accessed: 2016-03-15). Available: http://wiki.scratch.mit.edu/wiki/Blocks

[32] Raspberry Pi 2 Model B. [Online]. (Date Accessed: 2016-05-30). Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

[33] Raspberry Pi 2, Available: https://upload.wikimedia.org/wikipedia/commons/0/0c/Raspberry_Pi_2_Model_B_v1.1_front_angle_new.jpg (Accessed 2016-04-11)

[34] ARM. *“Cortex A7 Processor”* [Online]. (Date Accessed: 2016-04-26). Available: http://www.arm.com/products/processors/cortex-a/cortex-a7.php

[35] Raspberry Pi Documentation. *“GPIO: Models A+, B+, Raspberry Pi 2 B and Raspberry Pi 3 B”* [Online]. (Date Accessed: 2016-03-21). Available: https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/

[36] Sparkfun. *“Pull-up Resistors”* [Online]. (Date Accessed: 2016-04-26). Available: https://learn.sparkfun.com/tutorials/pull-up-resistors

[37] Raspbian. *“Raspbian Faq”* [Online]. (Date Accessed: 2016-03-15). Available: https://www.raspbian.org/RaspbianFAQ

[38] Dexter Industries.(2015-02-11). GoPiGo Raspberry Pi Robot. [YouTube video]. (Date Accessed: 2016-05-09). Available: https://www.youtube.com/watch?v=RtVInMW0S8o

[39] GoPiGo *“GoPiGo Robot Base Kit”* [Online]. (Date Accessed: 2016-05-09). Available: http://www.dexterindustries.com/shop/gopigo-kit/

[40] Dexter Industries. *“Raspbian for Robots Update”* [Online]. (Date Accessed: 2016-05-10). Available: http://www.dexterindustries.com/raspbian-for-robots-update/

[41] Raspberry Pi Remote access. *“DOCUMENTATION > REMOTE-ACCESS”* [Online]. (Date Accessed: 2016-05-09). Available: https://www.raspberrypi.org/documentation/remote-access/

[42] Dexter Industries. (2015-11-27). GoPiGo2 Assembly PC 3 Setup Wifi.[YouTube video]. (Date Accessed: 2016-02-22). Available: https://www.youtube.com/watch?v=lWfJq0zrZUs

[43] Arduino vs. Raspberry Pi: Mortal enemies, or best friends?[Online] (Date Accessed: 2016-05-11) Available: http://www.digitaltrends.com/computing/arduino-vs-raspberry-pi/.

[44] Phyllis C. Blumenfeld, Elliot Soloway, Ronald W. Marx, Joseph S. Krajcik, Mark Guzdial & Annemarie Palincsar (2011): “Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning”, Educational Psychologist Volume 26, Issue 3-4, 1991, Pages: 369 - 398. Available: http://www.tandfonline.com/doi/abs/10.1080/00461520.1991.9653139 (Accessed: 2016-05-11)

[45] Kodcentrum [Online] (Date Accessed: 2016-05-12). Available: http://www.kodcentrum.se/

Appendix A

Project Description

Programmable Physical Toys

Projektkod: DATX02-15-17

Toys of the future will be basically small and “simple” robots (in the broad sense) that autonomously, integrated with various devices, like phones and tablets, and in collaboration with humans will entertain current and future generations of kids. Some examples of these toys are:

- https://www.play-i.com
-
- http://www.gosphero.com
-
- http://www.hawkin.com/rc-robo-fish
-
- http://makeymakey.com/
-

This project will focus on software aspects and on educational toys.

Projekt- /problembeskrivning

The objective of this project is to develop a software platform for programming robot toys. The software platform will create a learning path for children towards usage and programming of present and future devices. Through our platform children can enjoy a fun and active lifestyle while learning the principles of programming, become critical and active citizens and, why not, find an early interest for a future scientific career!

The software platform will allow kids to design missions that robot toys have to realize through their movements and by performing actions like turning on/off lights, reproducing a sound, taking a picture, recording a video, vibrating, etc. A fundamental aspect of the project and of its software platform is the language that has to be used by kids to program these toys. The language should be graphical and funny, especially conceived for kids. Gamification mechanisms will be suitably exploited to rewards kids once pre-defined (training) missions will be accomplished.

The definition of the programming language can take the move from Scratch (http://scratch.mit.edu), which is a language to create stories, games, and animations and to share them with others around the world. Scratch has been also integrated with Arduino, then a first step towards the integration of software with robot toys is already made: http://s4a.cat/. Other extensions of scratch might be found here: http://scratchx.org/. There are some attempts to integrate Scratch with Raspberry Pi, like http://www.dexterindustries.com/BrickPi/program-it/scratch/

The first phase of the project will be making a study of the state of the art. Then the project will provide a design of the solution including target users (e.g. age of the kids that we target), type of robot toy (e.g. a ball, a car, a cylinder, etc.), hardware needed (e.g. arduino, Raspberry Pi, etc.), integration plan with software. Then the project can focus on the software part. Scratch is only one possible way to proceed, however, we don’t want to constraint your creativity.

Litteraturförslag

Basically the links above and other related material you might find on the web.

Förkunskapskrav

Gruppstorlek

4-6

Målgrupp

D, DV, IT, Z and E

Handledare

Patrizio Pelliccione, Docent, Software Engineering division.

Email: patrizio@chalmers.se

Appendix B

Surveys

Two surveys were performed. One before using the PedaGoGo and one after.
Here are the results of the first survey:

Stäng fönster

easyQuest

Är du kille eller tjej?

[✓ Kille]

Tjej

Hur gammal är du?

12 år

Vad vet du om programmering? (Hur fungerar det? Vad gör man när man programmerar?, osv)

vet inte mycket

Har du programmerat tidigare?

Ja

[✓ Nej]

Jag vet inte

Hur svårt eller enkelt tror du att det är att programmera? Tror du det är roligt eller tråkigt?

svårt

Tror du, om du lär dig att programmera, att det kan hjälpa dig inom andra ämnen i skolan? I så fall vilket/vilka?

it

Vad tycker du om robotar?

coola!

Vad känner du inför att programmera en robot? (Svårt/enkelt, roligt/tråkigt, osv.)

roligt svårt

< Föregående | Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

Stäng fönster

Är du kille eller tjej?

[✓ Kille]

Tjej

Hur gammal är du?

11

Vad vet du om programmering? (Hur fungerar det? Vad gör man när man programmerar?, osv)

vet ingenting.

Har du programmerat tidigare?

Ja

Nej

[✓ Jag vet inte]

Hur svårt eller enkelt tror du att det är att programmera? Tror du det är roligt eller tråkigt?

ganska svårt kul

Tror du, om du lär dig att programmera, att det kan hjälpa dig inom andra ämnen i skolan? I så fall vilket/vilka?

matte ja

Vad tycker du om robotar?

coola

Vad känner du inför att programmera en robot? (Svårt/enkelt, roligt/tråkigt, osv.)

Svårt roligt

~~< Föregående~~ ~~Skicka in >~~

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Stäng fönster

Är du kille eller tjej?

[✓ Kille] Tjej

Hur gammal är du?

12

Vad vet du om programmering? (Hur fungerar det? Vad gör man när man programmerar?, osv)

Jag vet inget

Har du programmerat tidigare?

[✓ Ja] Nej Jag vet inte

Hur svårt eller enkelt tror du att det är att programmera? Tror du det är roligt eller tråkigt?

Jag trorat det är mitti mellan svårt
Jag trolaftde är roligt

Tror du, om du lär dig att programmera, att det kan hjälpa dig inom andra ämnen i skolan? I så fall vilket/vilka?

nej

Vad tycker du om robotar?

roliga

Vad känner du inför att programmera en robot? (Svårt/enkelt, roligt/tråkigt, osv.)

Svårt

< Föregående Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Stäng fönster

easyQuest

Är du kille eller tjej?

Kille [✓ Tjej]

Hur gammal är du?

11

Vad vet du om programmering? (Hur fungerar det? Vad gör man när man programmerar?, osv)

inget

Har du programmerat tidigare?

Ja [✓ Nej] Jag vet inte

Hur svårt eller enkelt tror du att det är att programmera? Tror du det är roligt eller tråkigt?

roligt, svårt

Tror du, om du lär dig att programmera, att det kan hjälpa dig inom andra ämnen i skolan? I så fall vilket/vilka?

Ja matte

Vad tycker du om robotar?

[illegible] Coola

Vad känner du inför att programmera en robot? (Svårt/enkelt, roligt/tråkigt, osv.)

roligt, svårt

< Föregående [✓ Skicka in >]

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

Stäng fönster

easyQuest

Är du kille eller tjej?

Kille [✓ Tjej]

Hur gammal är du?

12

Vad vet du om programmering? (Hur fungerar det? Vad gör man när man programmerar?, osv)

Inget.

Har du programmerat tidigare?

Ja [✓ Nej] Jag vet inte

Hur svårt eller enkelt tror du att det är att programmera? Tror du det är roligt eller tråkigt?

Väldigt svårt. Tråkigt.

Tror du, om du lär dig att programmera, att det kan hjälpa dig inom andra ämnen i skolan? I så fall vilket/vilka?

Ja.
No, Matte

Vad tycker du om robotar?

Kul tror jag.

Vad känner du inför att programmera en robot? (Svårt/enkelt, roligt/tråkigt, osv.)

Svårt, lite tråkigt.

< Föregående | Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Stäng fönster

easyQuest

Är du kille eller tjej?

Kille [✓ Tjej]

Hur gammal är du?

11

Vad vet du om programmering? (Hur fungerar det? Vad gör man när man programmerar?, osv)

Jag vet inget

Har du programmerat tidigare?

Ja [✓ Nej] Jag vet inte

Hur svårt eller enkelt tror du att det är att programmera? Tror du det är roligt eller tråkigt?

Jag tror det är ganska svårt men roligt.

Tror du, om du lär dig att programmera, att det kan hjälpa dig inom andra ämnen i skolan? I så fall vilket/vilka?

Näe, det tror jag inte.

Vad tycker du om robotar?

Dem är väl fina...

Vad känner du inför att programmera en robot? (Svårt/enkelt, roligt/tråkigt, osv.)

Jag tror det är svårt och roligt

< Föregående Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

Here are the answers to the second survey:

Stäng fönster

Undersökningen är i visningsläge.
…ren sparas inte.

easyQuest

Är du kille eller tjej?

Kille [✓ Tjej]

Hur gammal är du?

11

Vad har du lärt dig om programmering? (Hur fungerar det?, Vad gör man när man programmerar?, osv)

att det är svårt

Hur enkelt/svårt tyckte du att det var att programmera? Var det roligt eller tråkigt?

jag tyckte det var lätt

Tror du nu att programmering kan hjälpa dig att lära dig andra ämnen i skolan? I så fall vilka?

Ja Matte

Hur var det att programmera roboten? Var det roligt/tråkigt? Var det svårt/enkelt? osv

roligt , enkelt

Skulle du vilja programmera igen?

[✓ Ja!] Nej! Kanske

Undersökningen är i visningsläge.
Svaren sparas inte.

Skulle du vilja lära dig programmering i skolan?

- Ja, som ett helt eget ämne
- [✓ Ja, i samband med annat/andra ämnen]
- Nej
- Jag vet inte

< Föregående | Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

Stäng fönster

Undersökningen är i visningsläge. Svaren sparas inte.

easyQuest

Är du kille eller tjej?

- Kille
- [✓ Tjej]

Hur gammal är du?

12

Vad har du lärt dig om programmering? (Hur fungerar det?, Vad gör man när man programmerar?, osv)

[illegible] Ja man lägger in olika program osv

Hur enkelt/svårt tyckte du att det var att programmera? Var det roligt eller tråkigt?

Enkelt. Roligt.

Tror du nu att programmering kan hjälpa dig att lära dig andra ämnen i skolan? I så fall vilka?

No, Matte.

Hur var det att programmera roboten? Var det roligt/tråkigt? Var det svårt/enkelt? osv

[illegible] Roligt. Enkelt.

Skulle du vilja programmera igen?

- [✓ Ja!]
- Nej!
- Kanske

Skulle du vilja lära dig programmering i skolan?

Undersökningen är i visningsläge.
Svaren sparas inte.

- Ja, som ett helt eget ämne
- [✓ Ja, i samband med annat/andra ämnen]
- Nej
- Jag vet inte

< Föregående Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

Stäng fönster

Undersökningen är i visningsläge.
en sparas inte.

Är du kille eller tjej?

- [ ] Kille
- [✓ Tjej]

Hur gammal är du?

11

Vad har du lärt dig om programmering? (Hur fungerar det?, Vad gör man när man programmerar?, osv)

Jag har lärt mig att programera en robot.

Hur enkelt/svårt tyckte du att det var att programmera? Var det roligt eller tråkigt?

Det var hyfsat enkelet och roligt.

Tror du nu att programmering kan hjälpa dig att lära dig andra ämnen i skolan? I så fall vilka?

Näe, det tror jag inte.

Hur var det att programmera roboten? Var det roligt/tråkigt? Var det svårt/enkelt? osv

Det var ganska svårt men roligt.

Skulle du vilja programmera igen?

- [✓ Ja!]
- [ ] Nej!
- [ ] Kanske

Skulle du vilja lära dig programmering i skolan?

Undersökningen är i visningsläge.
Svaren sparas inte.

[✓ Ja, som ett helt eget ämne]

Ja, i samband med annat/andra ämnen

Nej

Jag vet inte

< Föregående Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

Stäng fönster

Undersökningen är i visningsläge.
Svaren sparas inte.

easyQuest

Är du kille eller tjej?

[✓ Kille]

Tjej

Hur gammal är du?

12 år

Vad har du lärt dig om programmering? (Hur fungerar det?, Vad gör man när man programmerar?, osv)

Jag har lätt mig man lär en robot att göra saker

Hur enkelt/svårt tyckte du att det var att programmera? Var det roligt eller tråkigt?

Det var enkelt att programera och roligt

Tror du nu att programmering kan hjälpa dig att lära dig andra ämnen i skolan? I så fall vilka?

Nej

Hur var det att programmera roboten? Var det roligt/tråkigt? Var det svårt/enkelt? osv

roligt enkelt

Skulle du vilja programmera igen?

[✓ Ja!]

Nej!

Kanske

Skulle du vilja lära dig programmering i skolan?

Undersökningen är i visningsläge.
Svaren sparas inte.

- [✓ Ja, som ett helt eget ämne]
- Ja, i samband med annat/andra ämnen
- Nej
- Jag vet inte

< Föregående   Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

Stäng fönster

Undersökningen är i visningsläge.
en sparas inte.

easyQuest

Är du kille eller tjej?

[✓ Kille]

Tjej

Hur gammal är du?

11

Vad har du lärt dig om programmering? (Hur fungerar det?, Vad gör man när man programmerar?, osv)

Ja. det fungerar bra.

Hur enkelt/svårt tyckte du att det var att programmera? Var det roligt eller tråkigt?

det var enkelt. roligt

Tror du nu att programmering kan hjälpa dig att lära dig andra ämnen i skolan? I så fall vilka?

~~Ne~~ Ja matte.

Hur var det att programmera roboten? Var det roligt/tråkigt? Var det svårt/enkelt? osv

roligt mitt i mellan.

Skulle du vilja programmera igen?

[✓ Ja!]

Nej!

Kanske

Undersökningen är i visningsläge.
Svaren sparas inte.

Skulle du vilja lära dig programmering i skolan?

- [✓ Ja, som ett helt eget ämne]
- Ja, i samband med annat/andra ämnen
- Nej
- Jag vet inte

< Föregående   Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

Stäng fönster

Undersökningen är i visningsläge.
en sparas inte.

Är du kille eller tjej?

[✓ Kille]

Tjej

Hur gammal är du?

12 år

Vad har du lärt dig om programmering? (Hur fungerar det?, Vad gör man när man programmerar?, osv)

man ger den kommandon på ett program och jag har lärt mig om prog-rammet

Hur enkelt/svårt tyckte du att det var att programmera? Var det roligt eller tråkigt?

svårt ganska

Tror du nu att programmering kan hjälpa dig att lära dig andra ämnen i skolan? I så fall vilka?

it

Hur var det att programmera roboten? Var det roligt/tråkigt? Var det svårt/enkelt? osv

roligt var ganska svårt

Skulle du vilja programmera igen?

[✓ Ja!]

Nej!

Kanske

Undersökningen är i visningsläge.
Svaren sparas inte.

Skulle du vilja lära dig programmering i skolan?

● Ja, som ett helt eget ämne [✓]

○ Ja, i samband med annat/andra ämnen

○ Nej

○ Jag vet inte

< Föregående | Skicka in >

Detta är ett anonymt svar. Det betyder att avsändaren inte kan koppla dina svar till din identitet.
Läs mer om personskydd och säkerhet här. (http://www.easyquest.com/se/privacy-statement/)

Den här undersökningen har skapats med EasyQuest (http://www.easyquest.se)

(a) Two participants trying out the PedaGoGo language for the first time.	(b) A participant creating a PedaGoGo project.
(c) A PedaGoGo project is loaded onto the robot.	(d) One of the participants trying out their program while being observed by the others.

Appendix C

Links

The whole project repository: https://github.com/thunef/DATX02-15-17/

The PedaGoGo language: http://thunef.github.io/DATX02-15-17/

Scratch 2.0 source code on Github: https://github.com/LLK/Scratch-flash

Description of how to assemble the GoPiGo:
http://www.dexterindustries.com/GoPiGo/getting-started-with-your-gopigo-raspberry-pi-robot-kit-2/1-assemble-the-gopigo-2/assemble-gopigo-raspberry-pi-robot/

Appendix D

Guides

A step by step guide in how to compile Scratch 2.0

The compilation of the Scratch 2.0 source code is most easily done with the build automation system Gradle. To use this system one must first navigate to the folder containing the source code using a terminal. If running Mac or Linux, Gradle builds the code with the command "./gradlew build", or if running Windows this is done by running the "gradlew.bat"-file located in the same folder. If this works without any errors, the compiled source code will be located in the "build/11.6/" folder as "Scratch.swf".

If this does not work, a much more complicated compilation needs to be done. On both Windows, Linux and Mac machines this is done using another build automation system called *Apache Ant* combined with the *Adobe Flex Software Development Kit* (Flex SDK) version 4.10 or higher. The Flex SDK needs two additional files in order to work:

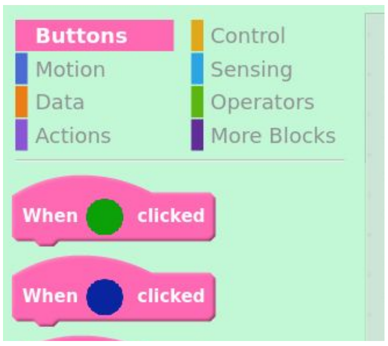
- playerglobal10_2.swc available at: http://fpdownload.macromedia.com/get/flashplayer/installers/archive/playerglobal/playerglobal10_2.swc
- playerglobal11_6.swc available at: http://fpdownload.macromedia.com/get/flashplayer/installers/archive/playerglobal/playerglobal11_6.swc

These files must be placed at:

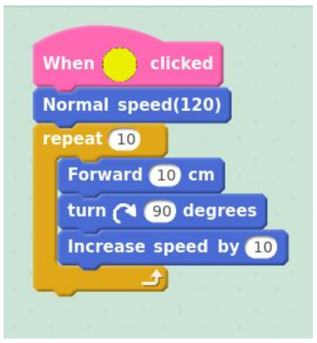
- <path to flex>\frameworks\libs\player\10.2\playerglobal.swc respectively
- <path to flex>\frameworks\libs\player\11.6\playerglobal.swc.

Then one has to navigate to the folder containing the Scratch source code and create a document named "local.properties" containing the path to the Flex SDK: "FLEX_HOME=<your path to flex>". Still located in the source code folder, by running the command "ant" in a terminal it should compile the source code and the "Scratch.swf" file will be found in the bin folder.

A step by step guide in how, in detail, to create a program and run it on the PedaGoGo robot.

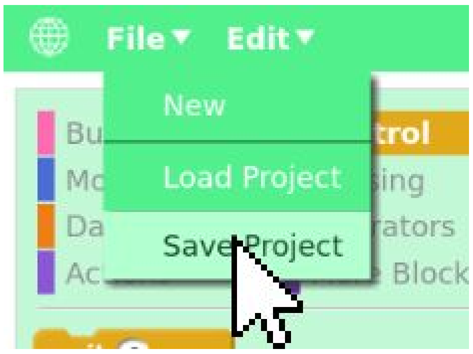


Step 1: Choose one or more of the blocks in the *Buttons* category, representing the three buttons (green, blue, yellow) on top of the robot seen in figure 7.3, and place in the script area.

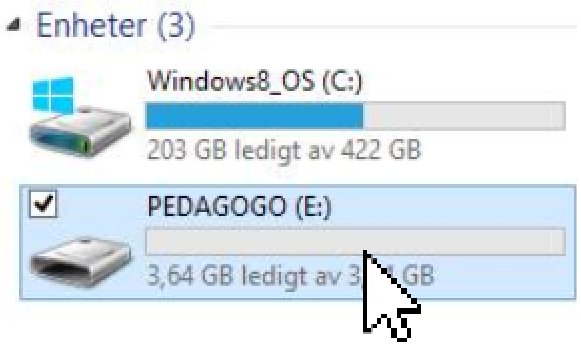


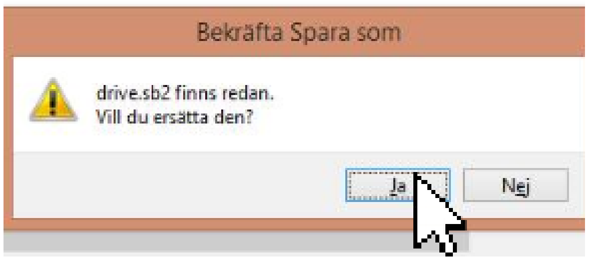
Step 2: Choose any combination from the remaining categories to build a script of your choosing.

Step 3: Insert the USB-flash drive that comes with the PedaGoGo into your personal computer.



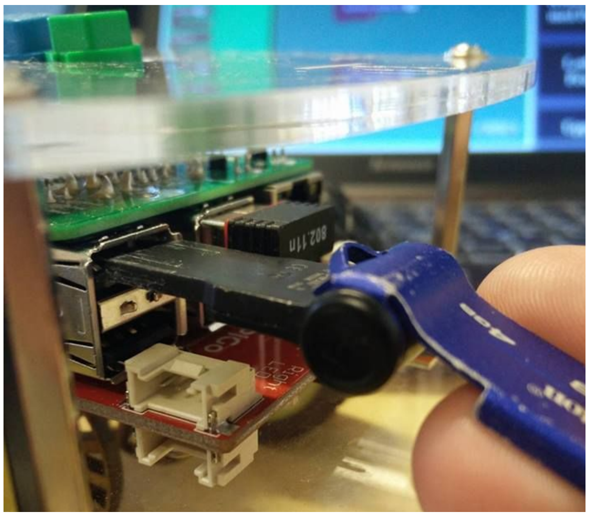
Step 4: Press "Save Project" from the top bar file menu.



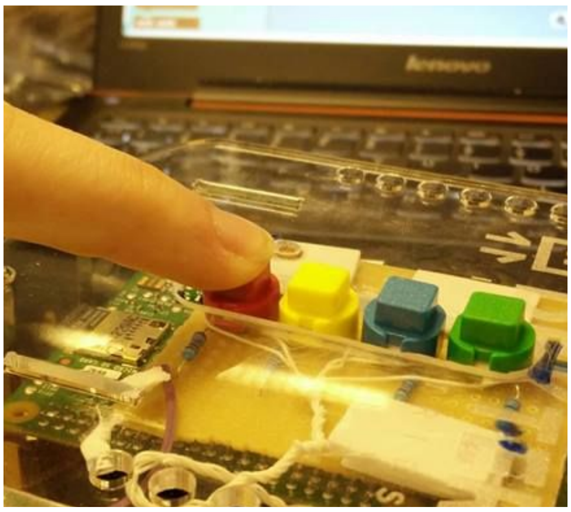


Step 5: Locate and enter the USB-flash drive (inserted in step 3) with the name of "PEDAGOGO"

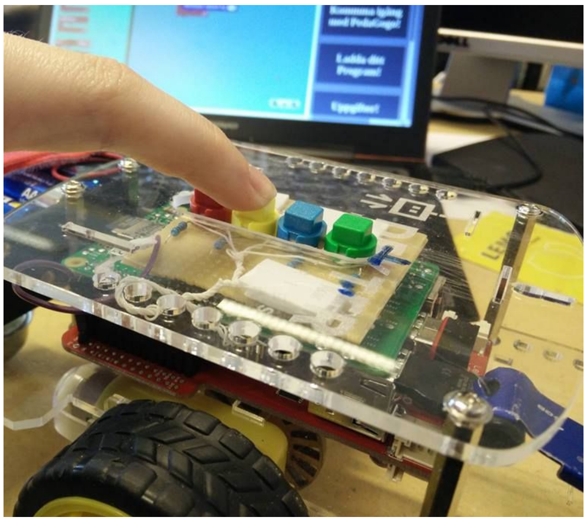
Step 6: Replace the "drive.sb2" project file located here with the new project file, also called "drive.sb2".



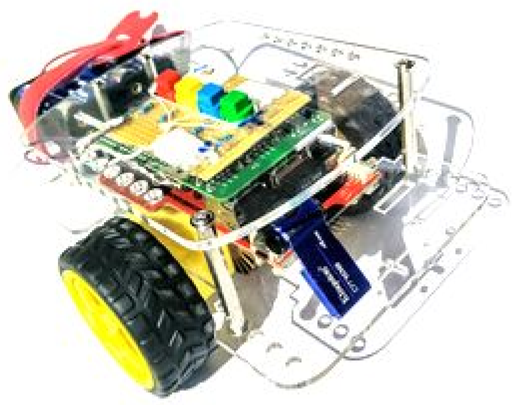
Step 7: Remove the USB-flash drive from the personal computer and insert it into one of the USB-hubs on the PedaGoGo robot.



Step 8: Press the red button to load your project on the PedaGoGo and stop any other scripts currently running on the robot.



Step 9: Press one of the button(s) with the same color of a button block used in your script.



Step 10: Observe the robot executing your script(s) and enjoy.