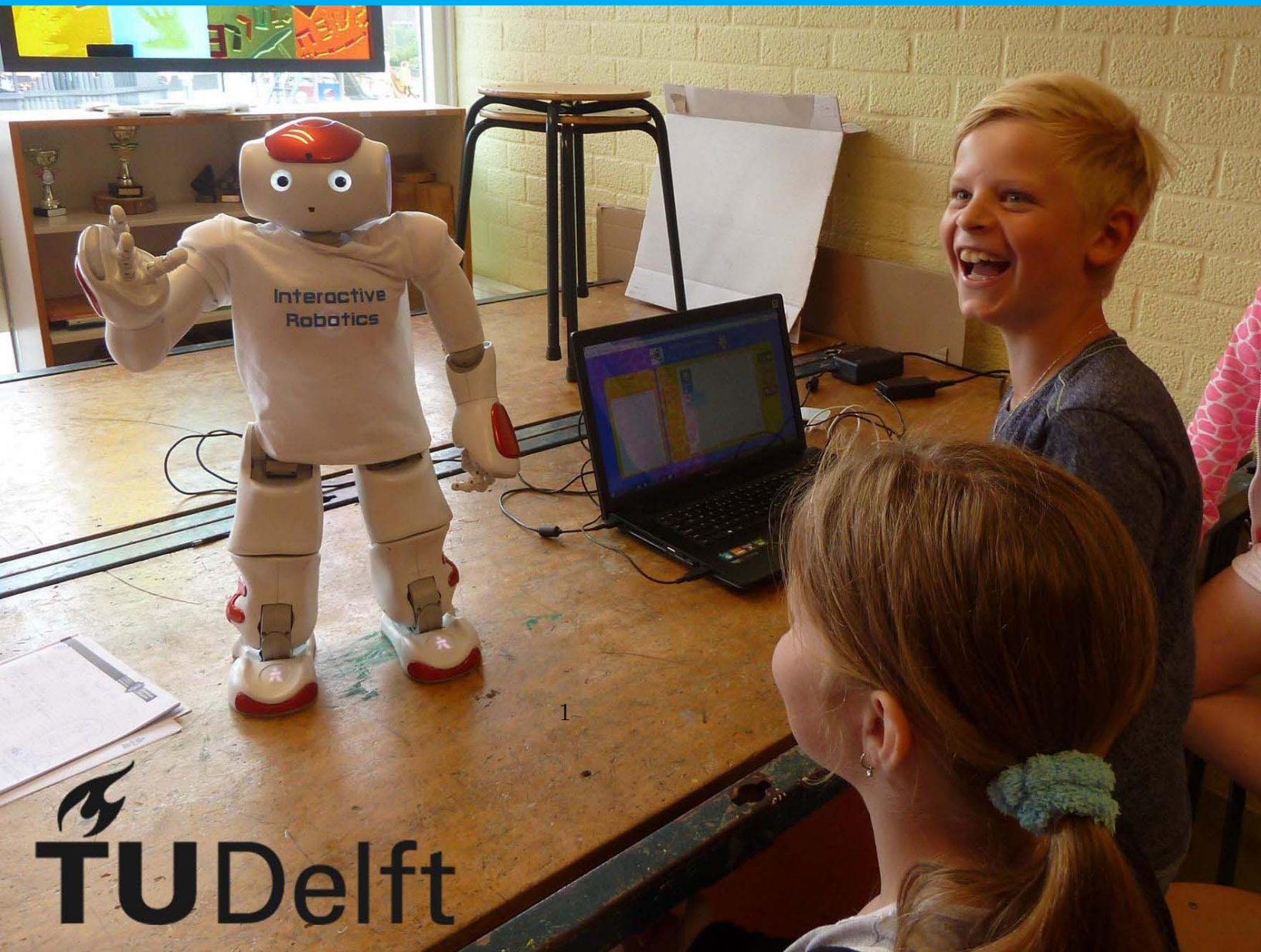


Robot Block-Based Programming

Teaching children how to program an interactive robot using a block-based programming language

Robin van der Wal
Jannelie de Vries
Luka Miljak
Marcel Kuipers

Bachelor's Thesis
Computer Science
Delft University of Technology



**This report is under embargo from July 2017 until
February 2018**

DELFT UNIVERSITY OF TECHNOLOGY

BACHELOR END PROJECT

ROBOT BLOCK-BASED PROGRAMMING

Final Report

Authors:

Robin van der Wal

Luka Miljak

Jannelie de Vries

Marcel Kuipers

July 5, 2017

Bachelor Project Committee

Coach name: Koen Hindriks

Client name: Joost Broekens

Cordinator name: Ir. O.W. Visser



Abstract

Robots play an increasingly large role in society and some material already exists that allows children to program robots in elementary school. However, this material often neglects the interactive capabilities of modern robots.

The aim of this project is to teach children how to write interactive programs for a robot. For this purpose, a NAO robot is used, which is a humanoid robot with advanced features. Children can use a web interface to create programs in a Block-Based Programming Language, which is then sent and processed by the robot in an intelligent manner, using an agent-based system.

Over the course of ten weeks, based on research done in the first two weeks, a web interface and an intelligent agent were developed. The BlocklyKids language implements many concepts you would expect from a programming language. Using these concepts, children can solve exercises that are presented to them in the web interface. Testing BlocklyKids in the classroom helped in the development of the product. The company Interactive Robotics, which commissioned this project, will further develop this project.

Keywords: block language, elementary school, primary school, robotic programming, interactive robotics, Blockly,

Preface

This report documents the BlocklyKids project, which was developed for the course *TI3806 - Bachelorproject*. This project aims to teach children about programming robots and the role of robots in society. This report gives an overview of the development process and the results that were achieved during the last ten weeks.

We would like to thank Koen Hindriks and Joost Broekens for their supervision, advice and enthusiasm. We would also like to thank Ruud de Jong for the technical support he provided. Finally, we would like to thank the teachers from de Meester van der Brugschool and OBS de Griffioen for letting us test the product at their schools.

Robin van der Wal
Jannelie de Vries
Luka Miljak
Marcel Kuipers
Delft, July 5, 2017

Contents

1	Introduction	6
1.1	Product description	6
1.2	Report overview	6
2	Problem Definition	8
2.1	Company: Interactive Robotics	8
2.2	General problem definition	8
2.3	Requirements	9
3	Problem Analysis	12
3.1	Blockly as a framework	12
3.1.1	Blockly composition	12
3.1.2	Similar use cases	13
3.2	Learning capabilities of elementary school children	14
3.2.1	Group 3 and 4	15
3.2.2	Group 5 and 6	15
3.2.3	Group 7 and 8	16
3.3	An intelligent and programmable robot	16
4	Software Design	18
4.1	Subsystem Decomposition	19
4.1.1	TECS Server	19
4.1.2	Web Client	19
4.1.3	NAO and NAOConnector	20
4.1.4	EIS Connector	20
4.1.5	Agent	20
4.2	Agent design	21
4.2.1	Eis-connector parser	21
4.2.2	Intelligent feedback system	21
4.3	Blockly Language design	22
4.3.1	Blocks	22
4.3.2	Canvas	23
4.3.3	Toolbox	23
4.3.4	Dutch naming conventions	23
4.4	Exercise examples	24
4.4.1	Example exercise for group 3 and 4	24
4.4.2	Example exercise for group 5 and 6	24
4.4.3	Example exercise for group 7 and 8	24

4.5	Web interface	25
4.5.1	Child interface	25
4.5.2	Teacher interface	26
4.6	Database	28
4.7	Message Design	29
4.7.1	Existing messages	29
4.7.2	New messages	30
4.7.3	Sequence diagram of a basic use case	31
5	Software Implementation	32
5.1	Roadmap	32
5.2	Scrum methodology	35
5.3	Development tools	35
5.4	Libraries	36
5.5	Quality Assurance	36
5.5.1	Scrum and User Tests	37
5.5.2	Code Tests	37
5.5.3	Regression Tests	38
5.5.4	Code Review	38
6	Evaluation & reflection	39
6.1	State of final product	39
6.2	SIG feedback	40
6.2.1	First feedback	40
6.2.2	Second feedback	41
6.3	Test coverage	42
6.3.1	JavaScript	42
6.3.2	EIS-Connector	43
6.3.3	GOAL	43
6.4	User test feedback	43
6.4.1	First user test	43
6.4.2	Second user test	44
6.4.3	Third user test	44
6.5	Ethical considerations	44
6.6	Reflection	45
6.6.1	Understanding of the RIE framework	45
6.6.2	Work rhetoric	46
6.6.3	Roadmap execution	46

6.6.4	Experience Gained	46
7	Conclusion	48
7.1	Conclusion on product	48
7.2	Conclusion on the project	48
7.3	Suggestions for future development of the BlocklyKids project	49
	Appendices	54
A	Glossary	55
B	Original BebSys product description	56
C	Executive infosheet	57
D	Research Report	59
E	Plan of Action	77
F	SIG Review	96
G	UserTest Results	98
H	Block overview	105

1

Introduction

Programming lessons are becoming more and more common in Dutch elementary schools. Children learn to write programs using a variety of different languages. Many of these languages are Block-Based Programming languages (BBPLs), which allow the user to create programs by connecting blocks. With robot technology becoming increasingly more advanced it is important that kids learn about robots and their role in society in the near future. BlocklyKids combines learning about programming and robots by allowing children to program the humanoid NAO robot using a BBPL.

This project was commissioned by the company Interactive Robotics, a Dutch company which aims to create a robotic interaction engine. Applications of this engine will be teaching children about interaction with robots, letting robots give interactive presentations and helping children practice in courses such as mathematics with robots.

1.1 Product description

The project is composed of a web interface and an intelligent agent.

In the web interface, children can choose an exercise that they need to solve. Using blocks, the child can create a solution to the exercise and send it. A teacher can create new exercises and make them available to his or her pupils.

The agent can interpret solutions in a smart way, let the NAO perform the requested actions and give feedback on the program.

1.2 Report overview

Chapter 2 gives an description of the problems that needed to be solved for this project, followed by a number of requirements that the solutions need to satisfy. In chapter 3 these problems are analyzed in more detail. Chapter

4 lists of all the different software components of the product. The software implementation process over the course of this project is given in chapter 5. A reflection on development process and the final product can be found in chapter 6. Finally, in chapter 7, a conclusion of the whole project is given together with recommendations for further development.

2

Problem Definition

This chapter describes the challenges that the team tried to solve during the project. The chapter starts with describing the company for which this project is done. Then the general problem definition is given to give an overview of the challenges. Lastly, the requirements for this project are stated using MoSCoW.

2.1 Company: Interactive Robotics

Nowadays, robots are quickly becoming more able to assist us. So human-robot interaction is the next challenge that needs to be solved. For robots to be able to assist humans in everyday life, they not only need to perform their task, but also do this in a way that makes sense to humans. Interactive Robotics is developing a Robot Interaction Engine (RIE), using cognitive technology to solve this problem.

In order to make this possible, robots need social intelligence to understand humans, natural interaction capabilities to talk with humans and robots should be able to adapt to humans. The Robot Interaction Engine enables users to quickly develop interactive scenarios for robot applications. Interactive Robotics' solutions strive optimal robot-human interaction. Whether it's a robot host, a care robot, a robot teacher or a robot teaching assistant.

2.2 General problem definition

Interactive Robotics wants a block-based programming environment for primary school children to program robots through the robot interaction platform. The aim is to use Blockly [1] as basis. In this project a web-based interface will be developed to be integrated in the existing platform. Within the web-based interface, Blockly is used to generate code for a robot pro-

gram. This project will also develop code for Prolog facts and rules that will be interpreted by GOAL [2]. Using this code and the web-based interface, children will be able to program a NAO robot. Key design criterion are that the project is easily extended to fit needs of different types of users to program the robot, ranging from young children up to adults and that the project can be easily integrated in the RIE system. The original project description is given in Appendix B.

2.3 Requirements

This section will describe the requirements of this project. The requirements were created in the first two weeks of the project in communication with Interactive Robotics. The state of completion at the end of the project on these requirements is described in section 6.1. The high-level features are defined using MoSCoW. MoSCoW uses four categories to separate the features by level of importance.

The categories are:

Must Have: Features that are of high importance. Without those features there isn't a product

Should Have: Features that are considered favorable. These features should be included, but are not necessary for the basic product.

Could Have: Features that are of low importance. When there will be enough time to implement, these features will be present.

Won't Have: Features that are of no importance or not doable. These won't be implemented. However, these features are free to be implemented by the company instead.

Must Have

- There has to be a Block based language for grades 3-4, 5-6 and 7-8 (Dutch school system). These languages have to be specially designed for these grades, based on what the children are able to understand.
- For each grade there has to be at least three exercises to program the NAO robot. These exercises teach programming in an interactive way.
- There has to be a web environment for children to program the NAO.

- The web environment has to contain a canvas where you can make your program.
 - the web environment has to contain a block library specially for the current assignment or grade level.
 - It has to be possible to select a block from the library and drag it toward the canvas.
 - It has to be possible to remove unused blocks from the canvas.
 - It has to be possible that the robot executes the program, so the children are able to see what their program does.
 - A child has to be able to go through the program step by step. So the robot executes a single step of the entire program. When doing this the current step of the program should be highlighted on the screen.
 - A child has to be able to pause the program.
 - A child has to be able to start the program.
 - A child has to be able to stop the program.
- There has to be a GOAL agent.
 - The GOAL agent has to be able to interpret a given program.
 - The GOAL agent has to be able to execute the program and send tasks to the robot.

Should Have

- GOAL agent
 - The Goal agent should be able to get sensor data from the robot. For example if a sensor is touched, the GOAL agent should receive this.
 - The GOAL agent should be able to process the received sensor data, either by sending something to the child, or something to the robot.
 - The GOAL agent should be able to execute the given program in a smart way, for example when it is interrupted, it should explain the cause to the child.

- the GOAL agent should be able to evaluate the solution the child gave for a certain exercise.
- There should be at least 5 exercises in total per grade (3-4,5-6,7-8).
- It should be possible to save and load partial solutions to exercises, so the children can continue their exercise another time.
- There should be an environment where teachers can create their own exercises.
 - There should be an environment where teachers choose the blocks needed for an exercise.
 - There should be an environment where teacher can create a possible solution for the exercise.
 - The exercise should be saved.
 - Teachers should be able to select exercises for children to work on.

Could Have

- There could be more exercises for each grade (3-4, 5-6, 7-8). In this way children have more exercises to learn from.
- The block language supports the facial recognition features of the NAO.
- There could be a virtual robot in the web-client. In this way, more children can learn to program, because less actual robots are needed.
- The Block language and the agent support recognition of objects.
- There could be a monitoring system, that monitors the behavior of the children and teachers when they use the web-client. The results could be saved in a database.

Won't Have

- There won't be a text editor to program the robot, because the idea of this project is to create a visual environment for pupils.
- There won't be any customizable shapes of blocks, because the basic Blockly blocks should be enough for the NAO actions and creating additional Blockly functionality and/or blocks is not the focus of this project.

3

Problem Analysis

The problem analysis contains a summary of the research that was done to create the software design. A more detailed report, that was made at the start of the project, can be found in appendix D: the research report. The main research question that is answered in that report is: “How do you teach children about robotics and programming using the NAO robot and a BBPL?”. Firstly, because the Blockly [1] library will be used, its capabilities are examined. The section after that is dedicated to outlining the capabilities of children from different grades. Lastly, a section is written about the design principles for an intelligent robot that will interpret the created BBPL.

3.1 Blockly as a framework

The Blockly framework is one of the two core components of the product, the other being the programmable robot. In this section, the capabilities and limitations of Blockly are explored. First, the framework itself is examined, then a few use cases similar to this product are looked at in order to get a grasp on possible competition and lastly a few limitation of Blockly are considered. In each category Blockly is evaluated in terms of this project.

3.1.1 Blockly composition

Blockly is an open-source JavaScript library that adds a block-based virtual code edit environment made by Google [1]. Blockly adds several different components that are useful for this project. Each component is briefly explored here.

Firstly, the main function of Blockly is to add a workspace for code editing. This workspace consists of a toolbox and a work area. The Toolbox contains the code blocks the user can use. The user can drag multiple blocks to the work area. blocks can fit into each other, and combining multiple blocks creates a “program.” There is usually a garbage can where the user

can drag blocks to delete them. Blockly allows restrictions on what blocks the user can use. Blockly already has some predefined blocks that allow for basic programming.

Secondly, Blockly allows developers to define custom blocks. These custom blocks can have multiple connections of different types, giving a lot of freedom in the creation process. The custom blocks can have their own text, images, and input types. Google has a web-page for creating custom blocks using the Blockly framework, which facilitates making new blocks [3]. Custom blocks make it possible to adapt the Blockly framework for robotic programming.

Thirdly, after a user has created a program, this program can be converted to code. Blockly itself supports multiple languages such as JavaScript, Python, Lua and Dart. Code generation is done by a generator script. This script has to be defined per code block, including for each new block. This allows Blockly to generate the type of output that the developer requires.

Finally, after a user has made some process with his or her code blocks, the code can be saved to XML output. This XML output can then be used to later reload the process of the user, or for future analysis.

All these features can be used for making a visual BBPL for programming robots. The workspace and toolbox allow for creating assignments for children. Next the custom blocks makes it possible to create blocks with images, so they are more suitable for young children that might have difficulty reading. Furthermore, the code generation can be made to match the robot programming interface. Lastly, the saving feature allows for users to keep track of their process while programming. All in all, these features make Blockly a suitable framework for this project.

3.1.2 Similar use cases

Blockly has been used before in programming robots in elementary school. In the article by M. Saleiro, et al. [4], the authors used Blockly in conjunction with small low-cost robots. These robots were constructed from simple micro-controllers, a small motor, some sensors and a sender. With the help of Blockly the authors taught 8 year old children how to program simple actions and movement for the robots. Using an exercise about robot movement and a map, the authors were able to successfully teach the children about robotics and geography.

In other research C. Martinez, et al. [5], examined whether it was possible to teach children of ages 3 to 11 basic programming functions such as conditionals and loops. Here Blockly was used as well to program a simple robot. They were successful in teaching the basic concepts in the age categories 5 to 11 years old, which is similar to the age range of 6 to 13 used in this project.

While these use cases are similar, they were both done with simple robots. In this project a NAO will be used. A NAO robot is a complex humanoid robot. There is an existing Blockly-based programming language for the NAO called Open Roberta [6]. Open Roberta has an implementation of all basic features that the NAO comes with. It has blocks for each sensor and multiple handy blocks such as the “Wait for” block. The programs can be exported to a NAO for execution.

Open Roberta is excellent for users who are familiar with robots and programming. But using it as an educational tool for primary school won’t work with a robot such as a NAO [7]. Open Roberta is a sandbox that gives all of the options of the NAO. Options like “get electric current of *head yaw*” won’t be used in a classroom. Someone who never programmed before will likely get lost. Though, there are good parts too. The command blocks are generic and precise. And the extra image guide for all the NAO sensors clears up where each sensor is. This makes Open Roberta useful to look at when examining the capabilities of NAO programming.

Thus there have been multiple successful attempts on teaching robotic programming to children of elementary school with Blockly. However, using a humanoid robot to achieve this task hasn’t been done before in elementary school.

3.2 Learning capabilities of elementary school children

In 2006 the Dutch government decided on several core goals primary school students should have reached at the end of grade 8. Next some examples are given. Dutch language: children should be able to get information from spoken language. English: students should be able to write simple daily used English words. Math: students should be able to add, subtract and multiply. All other goals the government decided primary school student should have are stated in “Het kerndoelenBoekje” [8]. Each school is able to decide how they teach these goals to their student, but the basic

curriculum per school is the same.

However, this means that there is no precise definition of what children should learn in each grade in elementary school. Interactive robotics requested that this project focused on primary school students and suggested splitting the primary school grades into three groups based on average age and capabilities of those groups. In this section, each group is examined on their learning capabilities and a suggestion on their programming capabilities is given.

3.2.1 Group 3 and 4

In grades 3 and 4 the students get the basic knowledge needed to learn more advanced information in the following classes. In grade 3, kids start learning. They have to sit still and concentrate. The first steps of reading and writing are made. Kids are able to read one line sentences. They also learn math. They are able to add and subtract numbers to 20. In grade 4, kids learn to read consecutive sentences, furthermore they learn basic grammar. Kids learn multiplication for the numbers below 6.

Thus, children from grade 3 and 4 are able to read simple and small sentences and will program based on images. A language for grade 3 and 4 should only contain simple blocks with images. These simple blocks are the basic movements the robot can make, for example, walking, waving, dancing, standing, sitting etc. It also contains a “for-loop” block for repeating actions, but no other advanced blocks.

3.2.2 Group 5 and 6

In grade 5, kids learn to write and read longer sentences. Math exercises become a bit more difficult. In grade 6, fractions and divisions are introduced. Language skills are improved. A programming curriculum [9] is proposed for Dutch schools. This curriculum gives an insight into what kinds of programming skills should be expected from Grade 5 and 6. Focus is put on abstract reasoning and problem solving. By now, kids are ready to learn more complex programming concepts. Kids should be able to work with if-statements in a program.

The basis for grades 3 and 4 is expanded on with new blocks for grades 5 and 6. Most importantly, a switch block is introduced. The previously mentioned curriculum says that children can be expected to work with if-

statements, but in this project the children will work with the sensors of a robot. The switch block can take different routes depending on what sensor was touched. Each sensor will be included in another block. This seems to offer more functionality than a simply if-statement and makes it more clear that the switch block is for sensors.

The idea of the switch block also allows of the introduction for the “wait for block”. In the “wait for” block, the robot waits till a condition becomes true, such as sensor that is touched. The inclusion of the switch and “wait for” block comes paired with the introduction of boolean expressions. “And”, “not” and “or” blocks are implemented, but the exercises will not require extensive usage of these blocks.

3.2.3 Group 7 and 8

In the 7th grade, reading and the Dutch language is even further explored. Children learn to sum numbers up to 1,000,000. Fractions, decimals, percentages, averages and ratios are introduced. The metric system of length, area, volume and weight is also explained. The 8th grade continues on this; learning the language becomes more difficult and they explore the bigger areas in math such as the circumference of a circle.

While grade 3 through 6 children only have the standard “loop x times” block, grade 7 and 8 should be able to handle the two more complicated blocks: the “while” block and “until” block. The “while” loop keeps repeating an action while a certain condition is true, the other keeps repeating an action until a certain condition is true. The idea is that these children should be able to create conditional loops as explained in section children learning capabilities from the research report. Furthermore, the “time” blocks are introduced. The “time” blocks can be used in a similar manner as sensors in conditional blocks such as the switch block or “wait for” block.

3.3 An intelligent and programmable robot

When working with interactive robots, it is necessary to design some sort of “intelligent” behavior. This project uses the NAO, which is a infant-sized humanoid robot. The NAO will fulfill the roll of the interactive robot. In this section, interactive robots in the context of working with children are explored and suggestion is made for the robot.

Robots will play an interactive role in society. Exposing children to robots early will help prepare them for this future. Two robots that were used in this context are examined.

One of the first robots that were used in an interactive roll with children was the Robovie [10]. The Robovie was used in children-robot interaction research at a Japanese elementary school. The robot had basic behavior such as shaking hands and giving hugs. The robot could also speak, but only speak and recognize English. The goal of the robot was to teach children some basic English words. Two exams at the begin and after the two week test period with the robot, revealed that it did have a positive impact on the English understanding. The research did not include a control group, but it showed the possibility of positive effects. The researchers noted that the interaction and idle movements the robot did were main reasons for children paying attention to the robot [11].

A more recent example of a robot in an elementary classroom was the DragonBot. This bot was designed to teach children about nutrition. Over a 3 week period, children had to socially interact with the robot and present food items to it. The DragonBot would respond and would prefer healthy foods in its interactions. After the test period, children took more time to consider what food to choose and it was shown that the children kept showing an extremely positive attitude towards the robot [12].

Based on the research above, NAO should be interactive for children to keep them engaged. There should be a set of idle actions and movement that occur when the robot is not busy with execution a BPPL program. Furthermore, the NAO should greet children when starting up, be interactive during the assignments and say goodbye when shutting down. Both works weren't able to show how effectively the robots were able to teach the children. Thus when designing a robot programming teaching system, it is important to keep testing the system with the users and keep track of progress.

4

Software Design

This chapter gives a global overview of the implementation of the product. The first section gives the subsystem decomposition, giving a global outline of the different system the product uses. The section “Agent design” goes into more detail on the EIS-Connector and the GOAL agent, which are mentioned in the subsystem decomposition. Afterwards, “Blockly Language Design” gives an overview of some design choices made for creating the BlocklyKids language. The next section gives a few examples of programming exercises for the children. Section “Web Interface” gives an overview of how the interface for both the child and the teacher looks like. “Database” gives a short overview of what client data needs to be saved in the server database, so that it can be loaded another time. Finally, “Message Design” goes into more detail on what, when and how messages are sent throughout the system.

4.1 Subsystem Decomposition

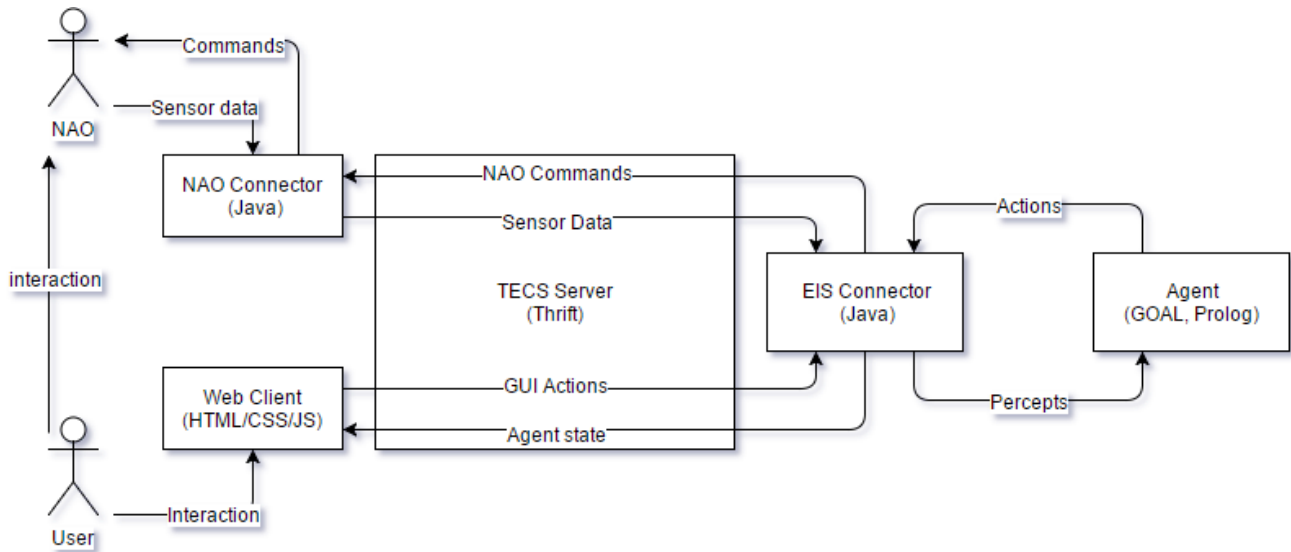


Figure 1: System decomposition of the product.

This section gives a global overview of the underlying system of the product. The figure above shows how the system is split up into smaller subsystems and how these subsystems interact with each other. The subsections below explain the purpose of each of these subsystem.

4.1.1 TECS Server

The TECS Server mainly acts as messaging system between the different modules of the product. The messages sent and received are Thrift based [13].

4.1.2 Web Client

The user can interact directly with the web client. The client sends "GUI Actions" to the server. Meaning that buttons being pressed or blocks moved on the screen are immediately sent to the server. Meanwhile, the client also receives the state of the agent. This way it knows for example what block is being executed and whether the program has finished. The web client can

react to these messages by highlighting the block being executed, locking or unlocking certain buttons on the screen. The button locking makes it more clear what actions can be taken during program execution, as examples: a paused program can't be paused again and if no program is running the stopping the program has no effect.

4.1.3 NAO and NAOConnector

The NAO robot can be interacted with by the user. The user can do this by for instance pressing buttons and activating sensors. The NAO sends this data to the NAO connector (located on the NAO itself). A few examples of data being sent is that of the foot bumpers that are pressed or that the posture of the robot has changed. Some types of sensor data that are performance-intensive, require that the data is being request. This is included in the NAOCommand messages. However, NAOCommand messages are mostly used to make the robot do certain movements or say a particular text.

4.1.4 EIS Connector

The EIS connector [14] mainly acts as an intermediate between the GOAL agent and other modules that want to send message to the agent. In the scope of the BlocklyKids product, the most complicated operation is parsing a blockly string that have been sent from the web client. This parsing is not done in the agent as it's simply easier to do in Java than in Prolog.

The EIS connector sends messages to the agent as percepts and receives messages as actions. Actions are usually immediately forwarded to either the NAO Connector or the Web Client.

4.1.5 Agent

The GOAL agent [2] handles all the complicated logic required to control the robot. Block programs that have been sent by the user are interpreted here. Whether actions performed in the agents are forwarded to the NAO or the Web Client is decided by the EIS Connector.

There are two main reasons why a GOAL agent is used to control the NAO, rather than just 'pumping' Javascript block code into the NAO Connector. First of all, most projects in the Interactive Robotics company use GOAL as their way of controlling robots. Some projects might even be

merged together into one by merging their GOAL agents. Therefore, to ensure a product that is easily integrated into the existing RIE framework, GOAL is a solid choice to run the robot on.

Secondly and more importantly, GOAL is a language specifically designed for logic based Artificial Intelligence. While it is true that a Blockly program created by the user is simply executed sequentially, which can easily be done in a language like Java, the robot always needs to give some form of intelligent feedback. Examples of such intelligent feedback can be found in section 4.2.2.

4.2 Agent design

This section explains the real purpose of the EIS Connector and why GOAL has been used instead of a traditional programming language like Java.

4.2.1 Eis-connector parser

As the Subsystem Decomposition section showed, between the Web client and the GOAL agent there is a Thrift-based server and an EIS Connector. Because Thrift doesn't support real complex object, the easiest way to send a block program to the agent is as a string. This string can be parsed inside of the EIS (Java) and can then be sent to the GOAL agent. The string sent from the web client should follow a certain syntax.

Apart from just the block program, other simpler messages are also sent. The content of these messages can be in JSON and if the EIS detects that the format of the content is in JSON, this is automatically parsed and sent to the GOAL agent. This allows for some general messages that don't need any additional code in the EIS.

4.2.2 Intelligent feedback system

In this subsection, several points are listed which outline the intelligent feedback system of the NAO.

To start with, the NAO shouldn't always immediately start executing a program it has received from a child. When a human receives an assignment, it will first get ready to execute it. So the NAO should only start executing the program when it is ready and should give feedback before doing so.

Another example is giving reactions to the success of an exercise after running a program. The robot could give positive feedback when the children

succeed in finishing an exercise. The NAO has no facial expressions, but with the help of its LEDs and body posture it can give a sense of emotion, as can be read in this paper by M.A. Miskam [15].

Negative feedback when failing an exercise should be avoided and perhaps an explanation could be given by the NAO. An explanation would mean that the NAO tries to analyze the reason why it couldn't continue with a program. For example, if a program is missing a condition in a "wait for" block, the NAO will stop executing the program and tell the user that it should fill in a wait condition in this loop. This allows children to learn how they can create complete programs that work.

4.3 Blockly Language design

In this section, an overview is given of the design choices for the Blockly language that were made. These choices are based on the research done in the research report, this report can be found in appendix D. The design consists of four parts: the design of the blocks, the design of the canvas in which the blocks are placed, the design of the toolbox that contains the blocks and the (Dutch) naming conventions that are used in the language. The total overview of all blocks created during this project is given in Appendix H.

4.3.1 Blocks

Like with most other BBPLs, blocks have a jigsaw puzzle shape, which is provided by the Blockly framework [16]. When two blocks fit together, their connections light up, making it clear that they can connect. Most blocks contain an image, which shows what the block does. These images were created using a Photoshop template made in this project. Hovering over the block will show a tooltip, which helps when the image is not clear enough. For some blocks, it was not possible to find images that showed the function of the block, for example the "switch"-block. For these blocks, a brief description is given on the block, consisting of a few words. Some blocks can contain multiple other actions, such as the "for-loop". These blocks are C-shaped, making clear that blocks can be inserted. Each block is part of a specific category, which decides its color.

4.3.2 Canvas

Blockly allows making some modifications to the canvas on which the blocks are placed. This canvas contains grid points, allowing blocks to snap to a location. Dragging the canvas is possible, but it is ensured that the blocks will not leave the canvas because of this. A garbage can is present, which allows for the deletion of blocks.

4.3.3 Toolbox

The toolbox allows for the selection of the blocks that need to be dragged onto the canvas. Because the language contains a large number of blocks, it was necessary to store the blocks in multiple entries. At first, they were separated by category, but this still resulted in entries that contained too many blocks. Because of this, some categories were spread out over multiple entries. For example, the “action” category is spread out over the “walk action”, “arm action” and “body action” entry. An entry has the same color as the blocks it contains.

4.3.4 Dutch naming conventions

The naming used on blocks, toolboxes and tooltips needed to be clear and understandable for Dutch children. A few other BBPLs exist that support the Dutch language, some examples can be found at [17]. These language were used for comparison to see whether terminology exists that is standard. Since BlocklyKids contains some features that cannot be found in other BBPLs this could not be done for every piece of text. To make sure that everything is still clear, children were asked during user tests whether they understood the more difficult terminology. Because of this feedback, some difficult words were switched for words that are more understandable. For example, the name “functioneel” (“functional”) was switched out for “herhalen” (“repetition”) because it turned out to be more intuitive after comparison with other languages. User tests showed that “sensoren” (“sensors”) was not a good name, because children did not know what a sensor is. For this reason, this name was changed to “aanraken” (“touching”).

4.4 Exercise examples

One of the deliverables was a set of exercises to use in the system that was build. This section will show some examples of exercises that were created for each group. Also an explanation is given for why this exercise was made.

4.4.1 Example exercise for group 3 and 4

For group 3 and 4 an exercise was made called “Head, shoulders, knee and toe”. In this exercise, the children will need to find the right blocks. Also they need a for-loop in order to repeat the knee, toe actions. In the song, this will be repeated too. By using the well-known song as an exercise, the children know how the end result should look like. Children only need to link the blocks to the parts of the song. Another way to solve the exercise is by placing another knee and toe block below the program. Both solutions are good solutions.

4.4.2 Example exercise for group 5 and 6

For group 5 and 6 an exercise was made called “Sprinkle chocolate sprinkles”. In this exercise, students need to make a slice of bread and let the robot sprinkle chocolate sprinkles, a well known Dutch bread spread, over it. This is an interactive exercise, because the students make something with the robot together. The robot needs to put its hand forward, the open is hand and wait for the sprinkles to be put in its hand. When the student put chocolate sprinkles in its hand, the student should touch the robot hand so it closes and the the robot will sprinkle the chocolate. This exercise was created because it is fun, interactive and the students learn to make their own lunch.

4.4.3 Example exercise for group 7 and 8

For group 7 and 8 an exercise was made called “command the robot”. In this exercise students need to command the robot with voice commands. The robot will walk until it hears something. When it is the word left, it will turn left, when it is the word right, it will turn right. When he hears the word stop, it will stop. This exercise was created, because it is interactive and the students are able to command the robot, not only by using blocks, but also by using their voice.

4.5 Web interface

The web interface that was made for this project is primarily for feature development and user testing. The focus of this project was to build a programmable robot and an interface to use it. To add to that, the RIE system was in heavy development at the time of this project. Thus, the focus of this project was not to integrate it into the existing RIE system. Instead a temporary web interface build.

This temporary web interface was the basis of the entire project and the primary testing platform. The website only uses HTML, CSS and JavaScript as development languages. The style used is a really basic and colorful idea of how the interface could look like and not much time was invested in the appearance because the website is only temporary. Nonetheless, it is made with children in mind as at the start of the project it was planned to have multiple user tests, thus it should look “fun” for children between ages 6 and 14. The code behind the website is as modular as possible and well documented, so that integrating the website into the RISE-platform is going to be easier.

The website contains two main components, the child interface and the teacher interface, which will be discussed further below.

4.5.1 Child interface

The idea in the entire RIE platform of using it in the classroom is that the teacher has control over the exercises that the students see. In other words, the teacher can set an exercise ready and then the students can do the exercise. This means that the only page students see is the exercise page itself.

The exercise page contains all the information for a single exercise. It displays the name of the exercise, its difficulty and a description of what the student should do. The exercise page only contains relevant blocks to the exercise, though sometimes it can contain blocks that might not be required for the correct solution to increase difficulty. Lastly, the page contains a toolbox with blocks and the workspace where blocks can be created.

The exercise page is the page that sends block programs and to GOAL agent. The exercise is also responsible for sending the solution of an exercise and the checking mode for the exercise. The exercise page contains a start, step, pause and stop button. All buttons do exactly what their name

describes, as an example the step button steps the program by 1 block, executing a single movement for the NAO. The exercise page can be seen in the following image.



Figure 2: The exercise page for students. This page contains all the information for exercises and a workspace to do them in. It also contains start, step, pause and stop buttons to let the GOAL agent know to execute the program. The language is in Dutch because the user tests were conducted with Dutch children. The text is also not the focus here as the image serves to give an idea of the structure of the exercise page only.

4.5.2 Teacher interface

The teacher web-interface, as said earlier, is for creating and setting up exercises for students. It consist out of three pages: the “grade / interface” page, the “select exercise” page and the “create exercise” page. The “grade / interface” page holds links to each grade and the “create exercise” page. The links of grades go the “select exercise” page which loads in the correct exercises per grade. There are only three HTML documents in total, one for each page, and loading in exercises is done dynamically. In the next image a visual representation can be found.

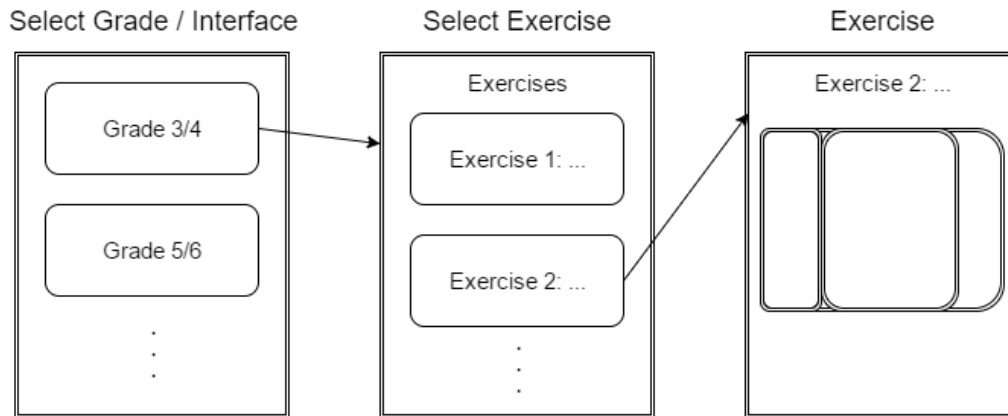


Figure 3: The flow diagram of the web-interface. First the teacher can select a grade level, then the teacher can select exercises for that grade level. After that the exercise is ready for the students.

The “create exercise” page contains more functions than the other two pages. At the moment any exercise created is transformed to a JSON object in string format. This object can later be saved in a database.

The page has the following fields: Blockly workspace, teaching goals, exercise name, exercise description, exercise checking method and exercise difficulty. A teacher can use these fields to create an exercise by simply creating the solution block program in the Blockly workspace. Additional blocks can be added by dragging them into the workspace and not adding them to the start block. The exercise page can be seen in figure 4.

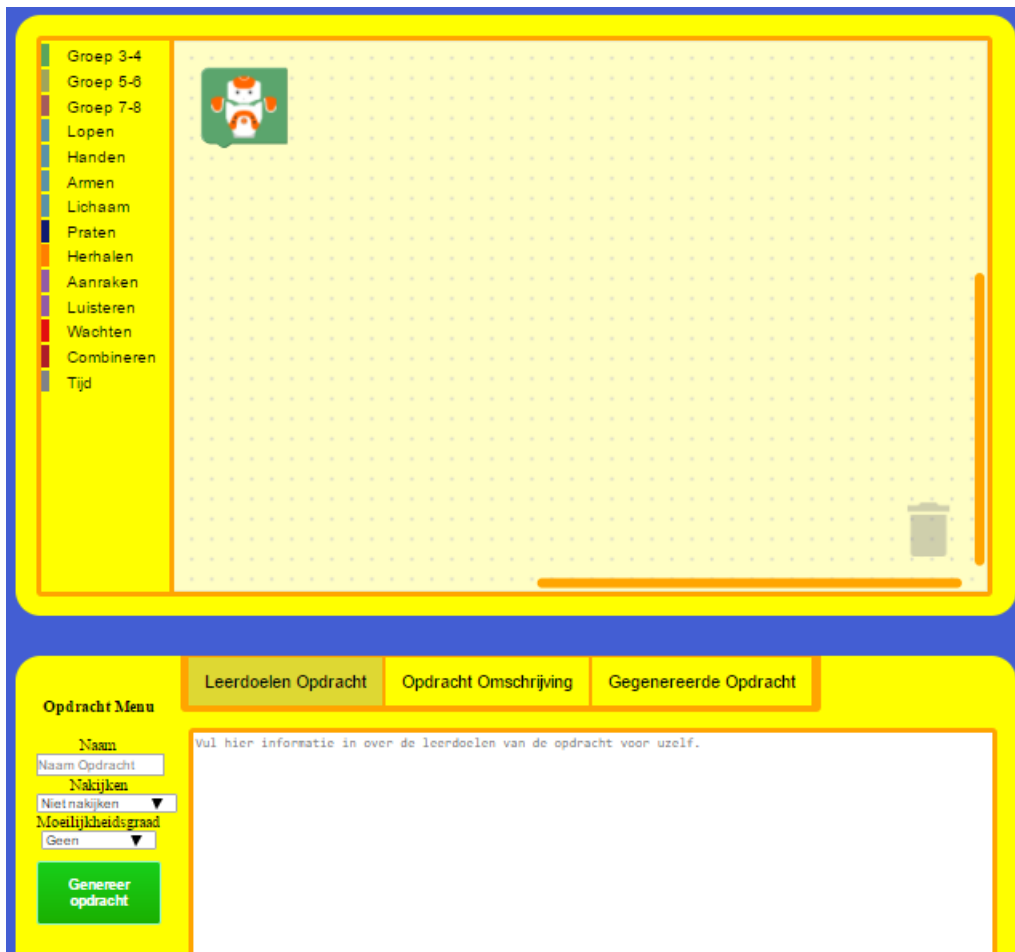


Figure 4: The “create exercise” page. This page contains all the fields necessary for creating a new exercise. Just as in figure 3 the text of this image is in Dutch because Dutch teachers used this page in user tests.

4.6 Database

There are several items that require saving inside of the server database. To make it simple, these items are all compiled to a JSON object. So that it can easily be inserted and retrieved.

The most complex objects that requires to be saved are Exercises. Exercises consist of a name, difficulty, appropriate grade, available blocks, the solution checking mode (either “none”, or “exact”, this decides how the ex-

ercise should be compared to the solution), and finally the solution to the exercise. Apart from exercises, a list of all the blocks needs to be saved as well.

4.7 Message Design

The BlocklyKids project requires messages in order to implement all the required functionality. There are 3 objects that require state updating: the NAO, the GOAL agent and the client-side web interface. Each object can send its own messages. There are two lanes of communication, the first one is from the web interface, to the GOAL agent and back, and the second one is from the GOAL agent to the NAO and back. Thus, the NAO and web interface do not directly communicate with each other. The state of the entire system is kept within the GOAL agent. This means that the GOAL agent is the primary entity that sends commands to the web interface and NAO, the other entities can send requests and precept to the GOAL agent, which the GOAL agent processes. The GOAL was chosen as centerpiece because it allows the agent to always hold the true state of the system, which makes it easier to react to and synchronize state changes from the NAO and the web interface.

This sub section contains all message related data for extending or maintaining the BlocklyKids message system. Firstly, messages that already were created by Interactive Robotics and used in this project are discussed. Secondly, the new messages are justified. Lastly, a sequence diagram is given of a basic use case of the Blocky-kids project.

4.7.1 Existing messages

These messages were already in use in other projects at Interactive Robotics. The two most important of these messages are the *gesture* message and the *low-level* message. The first message is used by the GOAL agent to send gestures to the NAO. The second message is used to signal the GOAL agent that it is done with completion. Other messages that were used in this project were for letting the NAO listen to voice commands.

4.7.2 New messages

The new messages can roughly be split into three categories. The first category is that of communication between the web client and GOAL agent. Here are messages for sending over the user created block programs and solutions to exercises to the goal agent. There is also a message for state updating. This state message is primarily used by the GOAL agent to let the client know it should highlight a block or that the agent is done with executing a program.

The second category of new messages are messages send from the NAO to the GOAL agent. For the project sensor input and posture information from the NAO is necessary. The additional information allows programs with booleans to be created and allow the NAO to stand up and react when it falls. The touch message covers all contact and tactile sensors of the NAO [18]. The posture message was only used in this project for when the NAO is lying on its back or belly, so that the NAO can stand up again [19].

The last category is a bit more special and is only a single message. During the project, it was noted by Interactive Robotics and their active development teams that the GOAL agent needs a lot of information from the outside to function. This includes everything from web client commands to voice commands. Thus instead of having to listen to all these different messages a single message was created, this message can be send by different sources to the GOAL agent. Touch input from the NAO is an example of such a message. However, because this message was created in the last weeks before the deadline, it was not feasible to convert all the GOAL agent input of the BlocklyKids project into this message.

So instead of converting everything last minute, only three messages were converted to this input message: start, pause and stop. These messages are sent from the web-client to the GOAL agent and are about the program execution of user-created programs. The start message starts a program and the other two messages speak for themselves. These messages are an example of how the new input message acts, and can be used to implement the other GOAL input messages similarly.

4.7.3 Sequence diagram of a basic use case

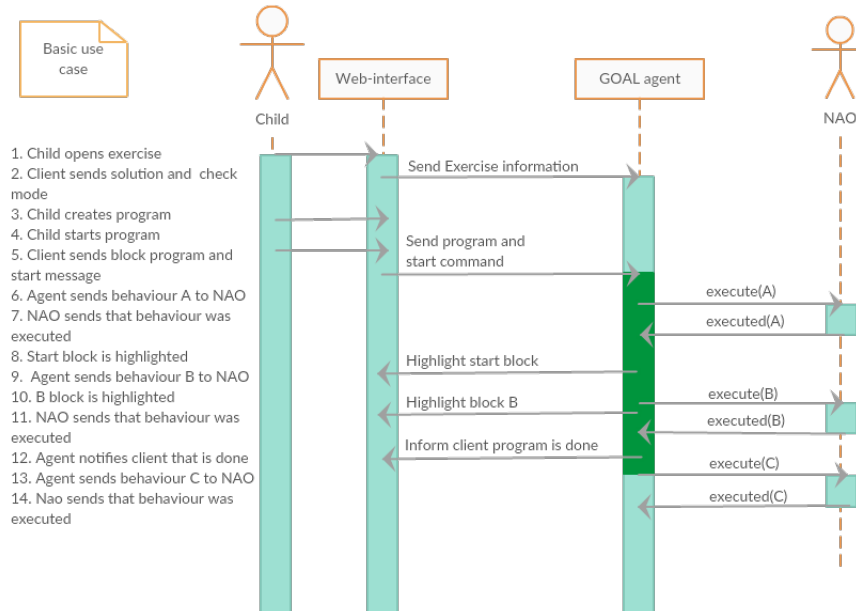


Figure 5: A sequence diagram of a basic use case of the Blockly project. In the image the bright green part is where the agent is executing the program. Behaviour A is a speech command that tells the user that it received a program. Behaviour B is the “wave” command, which corresponds to block B here. Behaviour C is an speech command that acknowledges that the program has finished executing. When the Highlight messages and the done message arrive, there is a visual change in the web interface notifying the user. While this is a very simply program it should still give a clear image on the messages that are passed around during run time.

This basic use-case considers a user that makes a simple block program of the block “wave.” The user then executes the program. The user and the NAO are physical objects here. The web interface and GOAL agent are already connected and running on a server. The GOAL agent normally also sends idle gestures to the NAO when the NAO is not executing any user created program, however these are left out for clarity.

5

Software Implementation

In this chapter the roadmap is given that was used during the project. Furthermore, the scrum methodology that was used will be explained. Next the used development tools and libraries will be given. Finally, the section Quality assurance will describe how the project teams assures the quality of the product.

5.1 Roadmap

This section describes a high level planning for items stated in the MoSCoW.

The process is split into several phases. The first phase is the **concept** phase, during the first two weeks. During this phase research is done and a plan of action for this project is created.

Then comes the **basic features implementation** phase in week 3 to 5. Basic features will be implemented, such as sending the code towards the GOAL agent and the GOAL agent should be able to send the task to the NAO. The aim is to have at least finished the *must haves* from the deliverables.

Next a more advanced agent is needed, so week 6 to 8 will be the **advanced features implementation** phase. In this phase the more advanced features will be implemented, like dealing with feedback from the NAO and testing the given solution. The aim is to have at least finished the *should haves* from the deliverables and possibly some *could haves*. In this phase the final report or in other words thesis is started on.

The last phase is the **finalization** phase in week 9 to 11. In this phase the final product needs to be delivered, as well as the final report. Lastly, a presentation has to be given on the project as a whole.

week 1 (24 April - 28 April) [Concept]
Research report.

Research learning capabilities for group 3-4.
Research learning capabilities for group 5-6.
Research learning capabilities for group 7-8.
Create a toolbox to select blocks from.
Create a canvas to make your program.
Implement block deleting from the canvas.

week 2 (1 May - 5 May) [Concept]

Research report.
Plan of action.
Create a Block based language for group 3-4.
Create a Block based language for group 5-6.
Create a Block based language for group 7-8.
3 exercises for group 3-4.
3 exercises for group 5-6.
3 exercises for group 7-8.

week 3 (8 May - 12 May) [Basic features implementation]

GOAL agent which is able to send the actions to the robot.
Basic Ontology based on the blocks available.
Design of basic pipeline with TECS server.
Design template for the robot images used in the blocks in the Blockly interface.

week 4 (15 May - 19 May) [Basic features implementation]

Basic pipeline set up with TECS server:
- GOAL agent which is able to send the actions to the robot.
- GOAL agent which is able to receive feedback from the robot.
- Web client that is able to sent basic programs, single moves, to the GOAL agent.
- GOAL agent able to sent feedback to the web client.
2 extra exercises for group 3-4.
2 extra exercises for group 5-6.
2 extra exercises for group 7-8.

week 5 (22 May - 26 May) [Basic features implementation]

GOAL agent which is able to send the actions to the robot such as idle actions, start up and shut down actions.

GOAL agent which is able to receive feedback from the robot based on execution of the robot moves as percepts.
Implement step by step program execution.

week 6 (29 May - 2 June) [Advanced features implementation]

Upload code to SIG.

GOAL agent which is able to send the actions to the robot when a program is interrupted for basic reasons such as waiting to long on an input task.

GOAL agent which is able to process the sensory data from the robot used in the program exercises.

Design teacher interface system together with company.

First field test with basic features of robot in a school.

Create proposal for content of the final report.

week 7 (5 June - 9 June) [Advanced features implementation]

Implement teacher interface.

Implement testing solutions given by the child.

Start writing final report, incorporating existing documentation.

week 8 (12 June - 16 June) [Advanced features implementation]

GOAL agent is able to provide feedback on incorrect solutions.

Write content for final report, based on new sections.

Second field test with advanced features such as program evaluation with robot in a school.

week 9 (19 June - 23 June) [Finalization]

Last field test with finished advanced features such as an feedback system with robot in a school.

Finalize all code implementations.

Finalize final report.

week 10 (26 June - 30 June) [Finalization]

Deadline final report.

Deadline info sheet.

Upload final code to SIG.

Prepare presentation and demo of presentation.

Week 11 (3 July - 7 July) [Finalization]

Presentation.

Upload thesis to repository.
Put library link on BEPsys.

5.2 Scrum methodology

The process of creating the product followed the scrum methodology. Meaning the project is split up into sprints of fixed time, in our case: a week per sprint. A sprint starts with a sprint planning, where every team member is assigned a certain set of tasks to fulfill by the end of the week, and ends with a sprint retrospective, where the team reflects on the progression that week, in order to improve the next. In between sprints there are also sprint reviews, where the team meets with the stakeholders in order to receive feedback on the product. A stand-up meeting happens daily (daily scrum), where each team member is expected to have an answer to what they have done since the previous meeting, what they're planning to do until the next and if they need any help.

The purpose of scrum is mainly to quickly be able to react to problems and stay in contact with the stakeholders of the product. For this reason, it is expected that by the end of each sprint the team can present a working product, that is an improvement of the product delivered the sprint before.

5.3 Development tools

The following are tools used to aid the team in writing good code throughout the project:

- NAO - The humanoid robot itself [20].
- NAOqi API - Java API to connect to the NAO [21].
- RIE Framework - Framework specifically made for the Interactive Robotics company.
- Eclipse - IDE used to write Java and GOAL code [22].
- GOAL - An Eclipse plugin that allows writing code in the GOAL language. GOAL is useful as it is a cognitive programming language specifically designed for logic based AI [2].

- QUnit - A Javascript framework used to test Javascript code [23]. Much of the web client contains complicated Javascript code with many potential bugs. Using a testing framework such as QUnit helps bug-proofing the code.
- Blanket.js - A QUnit extension that shows code coverage for JavaScript code [24]. The percentage of code covered in the test gives a decent indication of how extensive those written tests were.
- JIRA Software - Site used to create scrum boards [25].
- Gitlab/Github - Sites that use Git for version control [26] [27].
- Confluence - Site used to organize work documentation [28].
- Maven - A build automation tool primarily used for projects [29].

5.4 Libraries

The following tools are libraries that are mainly used to lower the code volume. These libraries contain some general functions that then don't have to be written again by the development team.

- Blockly - Used to create Blockly programming languages [1].
- jQuery - Library designed to simplify the client-side scripting of HTML [30]. Throughout this project it has mostly been used to load JSON files.
- Socket.io - Used to easily send messages to a server [31].
- org.json - A Java library used to parse text in JSON format. This was mainly used to handle incoming messages with JSON content [32].

5.5 Quality Assurance

This section describes how the project team assures the quality of the product. The first section will describe how scrum and user tests will add to this quality. Second the rules kept during the project about code testing. Then a part is written about regression test. The final section will describe how code review was done.

5.5.1 Scrum and User Tests

In order to get a high quality product that the stakeholders want, the process of creating the product follows the Scrum paradigm. This means that at the end of every week, the team is expected to have a working product to show, as soon as the implementation phase starts. By doing this, the consumer can immediately try out the product and provide feedback. This feedback can then be used to improve the product even further in order to completely satisfy the customer.

5.5.2 Code Tests

To achieve robust and well-working code, the project team makes use of automated Unit Testing; meaning that each function will be tested for correct and incorrect results.

It should be noted that a lot of code can't be tested for various reasons. First of all, the complete code base makes use of two different programming languages: Javascript and GOAL. The latter one, which is used to send certain instructions to the NAO Robot, is impossible to unit test. Furthermore, for a lot of parts of the Javascript code, unit tests would be meaningless. An example would be the interface of the web application, which is mostly written in Javascript.

For the parts of the Javascript code that are testable, the aim is to achieve 80% of line coverage. This means that when running the tests, at least 80% of the code lines will be executed. The tool used for unit testing Javascript code will be the QUnit framework. Code coverage is checked using Blanket.js.

Another type of code tests are end-to-end tests. Rather than testing separate units of the code, these automated tests are directly based on user stories and look of the code has the expected behavior of what the original task was.

Unlike with unit tests, GOAL does support end-to-end tests using test2g files. These test files have simple looking and easy to write tests like "when the agent thinks it hits a wall, it should stop trying to walk". In Javascript, end-to-end tests can be created the same way as unit tests are created, using the QUnit framework. There exists no way of getting the code coverage of GOAL programs, because there is currently no tool available for this.

5.5.3 Regression Tests

To ensure that the product doesn't get broken after a week through as an example old functionality that suddenly doesn't work anymore, regression tests are used. The idea is that whenever new functionality is added, included with unit and end-2-end tests for that functionality, all of the tests of the old functionality should be run with it as well. This allows the developers to see if old code broke down due to the new changes and speeds up the identification of errors in the code base.

5.5.4 Code Review

Automated tests are usually not enough to bug-proof code. One way to ensure maintainability and reliability of the code, is to do code reviews. Every time new code is added, it should be reviewed by *at least* two other team embers, before it can be merged with the current code base.

6

Evaluation & reflection

This chapter will evaluate and reflect on this project. First the state of the final product is given. Then the feedback gotten from SIG is discussed. Next the final test coverage of the product is given and explained. The fourth section will show the user tests results and discuss them. Section five will consider some ethical issues for this project. In the final section the reflection on the project is given.

6.1 State of final product

In section 2.3 an overview of all features is given that must, should, could or wont be implemented. This section will describe the features that are implemented, are not implemented and which features are added to the project.

During this project a Block based language was developed for grades 3-4, 5-6 and 7-8. In order to create this language, new behaviors for the NAO-robot was created. The blocks that are used in the language contain images to show what the block does. These images were created with the help of a Photoshop template created during this project.

With this block language, a total of 5 exercises were created for each group. The exercises consist of a Word template containing an explanation and an overview of blocks used. Also the exercises are implemented and shown on the student interface.

The student interface was also created during the project. Students can select an exercise, make it, send it to the robot. A student can go through the program step by step. It is possible to stop a running program. Besides the student interface, an teacher interface was created. With this interface, teachers can create an exercise. They can select which blocks should be visible for this exercise, what the solution should be, create the name of the exercise. They can enter a student description and teaching goals of the exercise. The difficulty of the exercises can be indicated. And lastly,

the method of solution checking can be selected, which can be exact or no checking at all.

When a program is send, the generator will generate code for the program. This code will be send via messages to the EIS-Connector. The messages were developed during this project. The way these messages are processed by the EIS-Connector is also implemented during this project. When the connector processes the code, it will send the blocks and state updates as percepts to the GOAL agent.

The GOAL-agent developed during this project processes the percepts, executes the programs by sending behaviors to the robot and gives feedback when something is wrong within the code. For example, when a block is missing, the GOAL-agent, will send a feedback message to the robot, so the robot can give the feedback to the student.

It is also possible for the robot to send feedback to the GOAL agent, for example, when it falls or when it is waiting too long. Also it can send a message when a sensor is touched, or a word is recognized.

As described above, a lot of work is done during the project. To link the work to the MoSCoW, described in section 2.3, This part will give an overview of what is done from this list.

All the Must haves are implemented in this project. From the should haves, everything is implemented except evaluating solutions in a smart way. It is possible to check the solution exact, but there was no time left to create a smart way of checking a given program with a solution. Also it is not possible to load partial solutions, because the exercises are small, so saving and loading is not necessary. None of the could haves are implemented due to lack of time.

6.2 SIG feedback

In this section the SIG feedback is given and an explanation is given on how this feedback is processed in the project. The complete and original feedback is given in Appendix F.

6.2.1 First feedback

The first feedback from SIG gave this project three stars out of five on their maintenance model. This meant that our code was average maintainable.

this project didn't get an higher score because it scored low on unit size and unit complexity.

A low score on unit size, meant that a high percentage of the code was above average long. So to solve this, it was needed to split these part of code into multiple parts. This would make the code easier to understand and easier to test. In order to improve the score in unit size, the length of the methods was reduced by splitting it into multiple methods.

A low score on unit complexity, meant that a high percentage of the code was above average complex. To solve this problem, also splitting the code could be a solution. By making the methods smaller, the complexity will decrease. In order to improve the score in unit complexity, the methods were split into multiple methods and where possible, we reduced the complexity.

The project also got some positive feedback. The test-code was promising sufficient, as when the code grows, the test-code should grow with it.

6.2.2 Second feedback

In the second feedback, it was noticed that the score for unit complexity and the score for unit size increased greatly. So the project learned from the first feedback. However, the product also got the feedback that the attention for increasing the maintainability overlooked the testing. So the testing score did not increase. This feedback was not expected, because the tests were adapted too. Also a lot of new tests were written and the end test coverage is over 95%, which is higher than most projects. Due to the refactoring some code was removed together with the belonging tests. This code was really complex, so it contained a lot of tests. Due to the refactoring, this code wasn't needed, so the tests were removed too.

Moreover the development team did not get the impression that SIG really looked at the test coverage of the JavaScript code, nor looked at the GOAL code as the team received no feedback on these parts both evaluation rounds. The team would have liked feedback on these parts of the product as well, so that these parts could have been improved, if necessary.

Though, at the end of the second SIG evaluation, SIG did concluded that the team took into account the feedback of the first round. All in all, the second feedback round was positive.

6.3 Test coverage

This section will give the test coverage for the JavaScript files, the EIS-Connector and the GOAL agent at the end of the project. Also it will describe why some parts are not tested.

6.3.1 JavaScript

A big part of the JavaScript files are tested. As seen in figure 6, the total coverage is 97.45%. There are a few files left out in the coverage, because they are too closely linked to the HTML pages and the Document Object Model (DOM) of those pages. Moreover, these files are only temporary and will probably not be used after integration in the RIE system, thus it was decided to not test these files. There are other ways to automate testing of websites, however because the website will be changed during integration in the RIE system, it would have been wasted work.

It was possible to test some parts of the navigate file. The rest of this file, however, could not be tested as it too interacted with the DOM. This project had a goal of at least 80% coverage, for the JavaScript files, and this goal was reached, even with the files that couldn't be tested.



QUnit 2.3.2; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36		
76 tests completed in 50 milliseconds, with 0 failed, 0 skipped, and 0 todo. 76 assertions of 76 passed, 0 failed.		
Blanket.js results	Covered/Total Smts.	Coverage (%)
1. http://172.16.80.1:8080/js/toolboxGenerator.js	20/20	100 %
2. http://172.16.80.1:8080/js/frontpageToolboxGenerator.js	12/12	100 %
3. http://172.16.80.1:8080/js/creatorToolboxGenerator.js	8/8	100 %
4. http://172.16.80.1:8080/js/customgenerator.js	252/253	99.6 %
5. http://172.16.80.1:8080/js/navigate.js	31/49	63.27 %
6. http://172.16.80.1:8080/js/createExercise.js	9/9	100 %
7. http://172.16.80.1:8080/js/customBlocks.js	393/393	100 %
Global total	725/744	97.45 %

Figure 6: Test coverage of JavaScript files

6.3.2 EIS-Connector

The EIS-connector is almost fully tested. The total amount of instructions are 1271. The amount of instructions that are covered are 1249, the total amount of missed instructions are 22. So the total coverage is 98.27%. For this project, a total of at least 80% test coverage was needed. With 98.26% this goal was reached.

6.3.3 GOAL

In this project, there were no test2g files generated for the GOAL agent. This was because, the GOAL agent needs the client side and the robot side to work with it. In order to test the GOAL agent, smoke tests were created on the client side. These tests contained all behaviors, sensors, timers, logical blocks and functional blocks. By sending these to the GOAL agent, it was possible to check if the blocks work properly, if the messages were sent correctly, if the GOAL agent processes the blocks in the right way and if the robot gives the right feedback.

Besides these smoke tests, the GOAL agent was also tested manually by sending some programs to the GOAL agent. Also during this project several user tests were done. Not only to test if the product was what the students wanted, but also to check if everything works properly. The results of these user tests can be found in the next section.

6.4 User test feedback

In this section, the feedback and notes from the user tests are summarized. The complete results are shown in appendix G.

6.4.1 First user test

During the first user test, a lot of students asked if the robot was able to do the dab. Sadly the robot couldn't do this, so this was implemented afterwards. Also it was noticed that group 3/4 should have access to the speech block, because they are able to type some words and they really like it when the robot pronounces their names. Even when it didn't pronounce it the way it should. In the stay alive module, the robot sometimes scratches his head. This was a weird movement, and the students asked a lot of questions about this. So maybe this behavior should be removed from the stay alive module.

Finally during this test it was noticed that the students don't even care about the exercise explanations, the students don't look at it and just start trying some blocks.

6.4.2 Second user test

During the second user test it was noticed that students still don't read the exercise explanation even when it is on the screen they are working on. Next time, there shouldn't be any explanation, so the students have to read the exercise. Also the menu item 'sensoren' is not clear for the students. This should be changed to 'aanraken', which is the Dutch word for touching. This name was chosen because all the sensors used are touch-based.

6.4.3 Third user test

During the third user test, it was noticed that if you don't give any explanation, the students will read the the exercise. But for group 3, the text was too small. This should be made bigger. Also some bugs were found. Listening to a word, when no word is given, will cause the agent to keep listening to nothing. Another bug that was found, is that the agent sometimes terminates randomly. These bugs should be fixed.

6.5 Ethical considerations

When teaching kids there is always a number of ethical considerations. In this subchapter three ethical aspects relevant to this project are outlined: pedagogical responsibility, security and privacy.

Though it can be achieved in a multitude of ways, a teacher always needs to fulfill its pedagogical responsibility. Curriculum design plays a very important role in this [33]. The material developed for this project prescribes a programming curriculum and therefore puts a restriction on what material is taught and how material is taught. Thus, it is important that the product provides enough flexibility for the teacher and has the approval of teachers.

Flexibility is guaranteed by allowing teachers to create their own exercises. For this purpose, a teacher interface was created with the wishes of teachers in mind. Testing the teacher interface was also part of the user tests. Lastly, adding new blocks and exercises is possible.

Ensuring that the product is in line with pedagogical principles was achieved by actively involving teachers in the user tests and talking with them about the product. Furthermore, other teaching materials were used as a reference.

Another ethical consideration is safety. The moving parts of the NAO might pinch a finger and might leave a mark. To add to that, the NAO is quite heavy, the danger exists that it falls off a table onto a child, which could cause injuries.

To prevent children's fingers getting pinched exercises that require a lot physical contact with the NAO are intentionally avoided. For instance, no exercises exist that require the child to pick up or hug the robot. Teachers should be informed of these hazards and will need to take them into account when designing exercises. Reading the safety guide [34] should be done as well.

Teachers need to be instructed not to put the NAO on a (small) table when exercises that require walk actions are done.

The issue of privacy is also relevant in this project. Though it is not yet implemented, eventually data will be stored regarding progress and other statistics. When eventually these functionalities are implemented it will need to be ensured that the personal data is encrypted.

6.6 Reflection

This subsection reflects on the project in the ten weeks that the group has worked on it. The aspects that are reflected one, are the company's RIE framework, the work rhetoric, the execution of the roadmap, and finally, the experience gained from this project.

6.6.1 Understanding of the RIE framework

During the start of the project, the RIE framework was a big unknown to all team members and was largely undocumented. The third and fourth week of the project was mainly used to get an understanding of the framework. During these weeks, no team member managed to fully understand the required knowledge. However, as a collective group, this was no problem. Some members knew how to connect to the robot, while others knew how to send messages from the web client to the server. It's safe to say that the group

was very dependent on each other. As the project progressed, everyone got on an understanding of the most important parts of the framework.

6.6.2 Work rhetoric

Thanks to the scrum methodology, all the group member always knew what to do and what the state of the project was at the time. The Sprint Planning however always caused large, but mutual discussions on what to do, what not to do, and especially how to do it. While the discussion sometimes became a little heated, each of these discussions resulted in a good compromise, where the best ideas of each member always got incorporated.

Looking at each individual's work, there is barely anything to complain about. Sprint items always got done on time. If not, there was always a good explanation and usually there was always another team member to take over a task of someone who can't finish it. Everybody contributed roughly the same amount and there were no issues with the amount of work somebody did within the group.

6.6.3 Roadmap execution

The roadmap, as defined in the previous section, was mostly followed as planned. Obviously it cannot be expected from any group that their project time-line completely matches their planned roadmap. During the first few weeks, the project proceeded almost exactly as planned. The few weeks after that, some tasks that had been planned for the weeks after, have been done in previous weeks. During the final weeks, the process slowed down a little as the result from user tests came in and the planning had to be adjusted accordingly. Thus, apart from the tasks that already had been specified in those week, some extra tasks came in. This was not problem however, as it compensated for the previous weeks going so fast.

6.6.4 Experience Gained

Working full-time on a project for ten weeks gave the group members quite a lot of experience. This will be divided into technical and social experience below.

Technical

There are many different programming languages that have been used in the project: JavaScript, HTML, CSS, Java, Prolog and GOAL. Of these programming languages, only Java has already been used a lot by the group members, the other languages were only briefly touched upon in a one or two courses before. Everyone gained some form of experience in all of the used languages.

TECS and Thrift messaging are also new concepts, these messages follow the publisher-subscribe design pattern. This pattern was new to the team, but this pattern certainly showed that it's very practical and easy to use in future projects.

This project also gave a lot of insight in how to make software for robots. The NAO robot in particular. Two ways used to program the NAO were Choreograph (an IDE specially developed for the NAO) and using the NAOqi API with Java. Choreograph was mostly used to define behaviors for the NAO. The NAOqi API was used to send and receive information from the server and get the NAO to do something with them. The experience gained here will certainly help with any future development for the NAO.

Google's Blockly framework was also new to all the team members. This framework shows great promise in any product requiring the user to program for the first time. Creating blocks and using these blocks to generate code is an experience of its own. Interpreting this generated code is also its own thing that relates much to concepts of programming languages.

Social

While all group members have done projects in the past for Bachelor Computer Science, none of these project were full-time. Scrum has been used in the past projects, but its was never as useful as in this project. Apart from scrum, not much more social experience has been gained, that hasn't been done so already in previous projects such as working together in a team and having fruitful discussions.

7

Conclusion

In this section, a conclusion is given on the product and the project. After that, a number of suggestions is given on how the product could be improved.

7.1 Conclusion on product

The results of this project are quite satisfactory. Functionality-wise the delivered product far exceeds the project description that was posted on BepSys. All must-haves and nearly all should-haves that were devised using MoSCoW have been implemented in the product. A lot of feedback from the user tests was processed and integrated into the project. A number of improvements can still be made, which will be discussed later in this section. All in all, the product is ready for integration into the RIE system.

7.2 Conclusion on the project

The goal of this project was to create a platform with which children in elementary school can write interactive programs for an intelligent robot. The project has achieved this goal by providing a web interface for creating and solving exercises, as well as an intelligent agent that can interpret these programs. Over the course of ten weeks a lot was learned. Two weeks of research resulted in a lot knowledge and allowed for the planning of the rest of the project. During the following eight weeks the product was developed. Some changes were made in the RIE platform to implement all requested features, which required independence and taking the initiative. The delivered product met the specified requirements and even exceeded them. The delivered product met and exceeded the requirements specified in initial product description. In conclusion, the project was very successful and will be a valuable addition to the RIE platform.

7.3 Suggestions for future development of the BlocklyKids project

In the last section of this report, the BlocklyKids team will give a few points where future development can be done on the system. In the first two points, the GOAL agent of the BlocklyKids team is considered, in the third point variable blocks are discussed and finally a suggestion is made on expanding the functionality of the project to high school students.

Firstly, integration into the RIE platform always has been a key design criterion and the GOAL agent is no exception. The team was challenged multiple times to come up with some ideas for connecting the different goal agents such as the arithmetic robot, the presentation bot and the programmable robot. One of the team's idea is listed here. The basic gist is that there should be one main GOAL agent that can do the basic things such as talk and do gestures. Then for each functionality there should be a module, or a group of modules, that will be loaded when a specific goal agent is requested. Creating this main GOAL agent, or loading in different modules was outside of the scope of the project and not the main focus. Though, in the future, it would prevent having a separate GOAL agent for each functionality that is added to the RIE platform, increasing maintainability for the RIE platform

Secondly, the presentation bot agent is very similar to the BlocklyKids agent. The BlocklyKids agent misses the noise detection and polling modules, but these can easily be added again. In another project at Interactive Robotics a Blockly based programming interface was made for the presentation bot. This interface already contains all the blocks needed for the presentation bot. To add to that the presentation bot at the moment needs an extra parser and interpreter step, that could be avoided using the agent that was created in this project.

Thus the presentation agent could be converted to the programming agent. This would be done in the following steps: 1) adopt the BlocklyKids generator in the web interface of the presentation bot, 2) write custom generator code for a Blockly block in the presentation interface and 3) handling the blocks generated in the BlocklyKids GOAL agent during block execution the same way that it is done now in the presentation bot. Doing this for every block in the presentation interface would be enough. And an added advantage is that the presentation bot can then use all actions and function-

ality of the BlocklyKids bot.

Thirdly, according to research done in the research report, children of in grade 7 and 8 of primary school are able to understand variables with types such as boolean, integers and doubles. This subject of variables was discussed many times during the project, but there was no time in the end to implement this feature correctly and robustly in the project. The team estimated that it would require 2 weeks of development time to get this feature in. This would have taken away from other features like sensor input, intelligent feedback on user created programs and system robustness. Adding this feature would not require any large changes to the GOAL agent, as it was constructed from the ground up with variables in mind. And so, for future development this should probably be the first feature that developers should concentrate on.

Lastly, the programmable robot was made for primary school, however the robot could perhaps also be used in high school. The entire suite of blocks made in this project should be used together with new blocks that will be created specifically for high school students. Developers of this feature should definitely include variables. To expand on that, the NAO has many advanced sensors not used for primary school, such as the sonar, facial recognition and electric current of different NAO parts. These sensors are implemented in Open Roberta and developers could take a look at that software to gain an idea on how to implement the advanced sensors. With variables and all the capabilities of a the NAO at hand, the programmable robot should be ready for high school students.

References

- [1] Apache Software Foundation, “Blockly,” <https://developers.google.com/blockly/>, [Software].
- [2] goal, “The goal agent programming language,” <https://goalapl.atlassian.net/wiki/display/GOAL/>, accessed: 2017-06-26.
- [3] P. in het PO, “Blockly developer tools,” <https://blockly-demo.appspot.com/static/demos/blockfactory/index.html>, 2017, accessed: 2017-05-01.
- [4] M. Saleiro, B. Carmo, J. M. Rodrigues, and J. H. du Buf, “A low-cost classroom-oriented educational robotics system,” in *International Conference on Social Robotics*. Springer, 2013, pp. 74–83.
- [5] C. Martinez, M. J. Gomez, and L. Benotti, “A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform,” in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 2015, pp. 159–164.
- [6] Fraunhofer IAIS, “open-roberta,” <https://lab.open-roberta.org/>, [Software].
- [7] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, “Graphical programming environments for educational robots: Open roberta-yet another one?” in *Multimedia (ISM), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 381–386.
- [8] ministerie van Onderwijs Cultuur en Wetenschap, “Kerndoelenboekje,” apr 2016.
- [9] P. in het PO, “Programmeren in het po,” kn.nu/leerlijnprogrammeren, 2016, accessed: 2017-04-26.
- [10] H. Ishiguro, T. Ono, M. Imai, T. Maeda, T. Kanda, and R. Nakatsu, “Robovie: an interactive humanoid robot,” *Industrial robot: An international journal*, vol. 28, no. 6, pp. 498–504, 2001.
- [11] T. Kanda, T. Hirano, D. Eaton, and H. Ishiguro, “Interactive robots as social partners and peer tutors for children: A field trial,” *Human-computer interaction*, vol. 19, no. 1, pp. 61–84, 2004.

- [12] E. Short, K. Swift-Spong, J. Greczek, A. Ramachandran, A. Litoiu, E. C. Grigore, D. Feil-Seifer, S. Shuster, J. J. Lee, S. Huang *et al.*, “How to train your dragonbot: Socially assistive robots for teaching children about nutrition through play,” in *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*. IEEE, 2014, pp. 924–929.
- [13] C. Buerckert, “Thrift eventbased communications system,” <http://tecs.dfki.de/tecs/>, accessed: 2017-06-26.
- [14] T. E. team, “Environment interface standard,” <https://goalapl.atlassian.net/wiki/display/EIS/>, accessed: 2017-06-26.
- [15] M. A. Miskam, S. Shamsuddin, H. Yussof, A. R. Omar, and M. Z. Muda, “Programming platform for nao robot in cognitive interaction applications,” in *Robotics and Manufacturing Automation (ROMA), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 141–146.
- [16] N. Fraser, “Ten things we’ve learned from blockly,” in *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*. IEEE, 2015, pp. 49–50.
- [17] Codekinderen, “Codekinderen,” www.codekinderen.nl, accessed: 2017-06-22.
- [18] “Nao software 1.14.5 documentation,” 2012, accessed: 2017-05-01. [Online]. Available: http://doc.aldebaran.com/1-14/family/nao_h25/contact-sensors_h25.html
- [19] “Aldebaran documentation alrobotposture,” 2012, accessed: 2017-05-01. [Online]. Available: <http://doc.aldebaran.com/2-1/naoqi/motion/alrobotposture.html>
- [20] S. Robotics, “Nao,” <https://www.ald.softbankrobotics.com/en/cool-robots/nao>, accessed: 2017-06-26.
- [21] A. Robotics, “Naoqi,” <http://doc.aldebaran.com/2-1/naoqi/>, accessed: 2017-06-26.
- [22] Eclipse, “Eclipse,” <https://eclipse.org/>, accessed: 2017-06-26.
- [23] J. Resig, “Qunit,” <https://qunitjs.com/>, accessed: 2017-06-26.

- [24] A. Seville, “Blanket.js,” <http://blanketjs.org/>, accessed: 2017-06-26.
- [25] Atlassian, “Jira,” <https://www.atlassian.com/software/jira>, accessed: 2017-06-26.
- [26] GitHub, “Github,” <https://github.com/>, accessed: 2017-06-26.
- [27] GitLab, “Gitlab,” <https://about.gitlab.com/>, accessed: 2017-06-26.
- [28] Atlassian, “Confluence,” <https://www.atlassian.com/software/confluence>, accessed: 2017-06-26.
- [29] Apache, “Maven,” <https://maven.apache.org/>, accessed: 2017-06-26.
- [30] jQuery, “jquery,” <https://jquery.com/>, accessed: 2017-06-26.
- [31] socket.io, “socket.io,” <https://socket.io/>, accessed: 2017-06-26.
- [32] org.json, “org.json,” <https://mvnrepository.com/artifact/org.json/json>, accessed: 2017-06-26.
- [33] G. J. Posner and A. N. Rudnitsky, *Course design: A guide to curriculum development for teachers*. ERIC, 1994.
- [34] *Safety Guide*, http://doc.aldebaran.com/2-1/_downloads/nao_safetyguide_2017_en_fr_sp_pt_de_it_nl.pdf, 2017 (accessed June 26, 2017).

Appendices



Glossary

This is a glossary of all the used terms within the report that might require additional explanation.

BBPL: Block-Based programming language

GOAL: Programming language for programming rational agents

Scrum: An iterative and incremental software development framework

RIE: Robot Interaction Engine. A system that is developed by Interactive Robotics.

BlocklyKids: Name of the product that was developed for this project.

TECS: Thrift Eventbased Communication System. System that used for message passing.

Interactive Robotics: The company that commissioned this project.

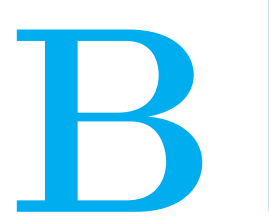
NAO: Autonomous, humanoid, programmable robot developed by Aldebaran Robotics.

Agent: Autonomous entity that senses and act upon its environment.

Elementary school: Dutch elementary schools were visited during the user tests and the product is aimed at Dutch children. All references to grades and schools are specifically about the Dutch school system.

Prolog: General-purpose logical programming language.

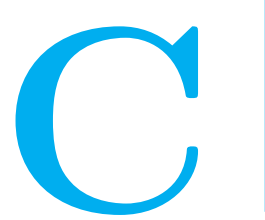
Blockly: Client-side JavaScript library for creating visual block programming languages and editors.



Original BebSys product description

Below is the original product description that was on BEPsys, the bachelor end product website for Computer Science.

We are developing a block-based programming environment for children to program robots through our own robot interaction platform. We aim to use blockly as basis. Students will develop a web-based interface to be integrated in our existing platform and within that web-based interface use blockly to generate code for our propitiatory robot script. If time permist, students will also develop the code generation for prolog facts that will be interpreted by GOAL (interpretation by GOAL not part of the assingment). Key design criterion is that we can easily extend the work of the students to fit needs of different types of users to program the robot, ranging from young to old children and even up to adults.



Executive infosheet

On the next page the executive infosheet can be found. This infosheet is an extremely compact representation of the entire project. The infosheet also contains the roles of the members within the project and contact information.

Executive Infosheet

Title of the project: Robot Block-Based Programming

Name of the client organization: Interactive Robotics

Date of the final presentation: 28-06-2017

Description: Robots play an increasingly important role in society. To teach children in elementary school about programming and robots, a Block-Based Programming Language was developed that allows children to write interactive programs and send them to a NAO robot. This robot then interprets these programs in an intelligent manner.

Since Interactive Robotics is still a young company, working in its code base was quite challenging. Good time management and communication with the client was important. Some independence was also required when it was necessary to extend the functionality of the system.

The Scrum methodology was used for the development process. A clear role distribution helped pull the project forward. Holding daily meetings kept all members on the same wavelength and allowed for quick detection and resolution of problems.

The main deliverables are a web interface and an intelligent agent. The web interface allows you to create and solve programming exercises, while the agent interprets the program. Children can use the product to write interactive programs for the NAO robot. The agent has some intelligent behavior and can, for instance, give feedback on weird or incorrect code.

The product is ready for integration within Interactive Robotics' system, the RIE-system. The implemented agent can be extended so that it can be combined with other RIE-projects, requiring only one agent to be maintained. Moreover, the language can be extended to also create exercises for high school students.

Members of the project team:

Name: Jannelie de Vries

Interests: Robotics, Teaching

Roles & contribution: Lead Tester, Developer

Name: Marcel Kuipers

Interests: Robotics, Agent Design

Roles & contribution: Product Owner, Developer

Name: Luka Miljak

Interests: Logic-based Programming, Agent Design

Roles & contribution: Lead Programmer,
Lead Presenter, Developer

Name: Robin van der Wal

Interests: Architecture Design, Front-end Development

Roles & contribution: Scrum Master, Report Master,
developer

Client and coach information

Client name: Joost Broekens

Client affiliation: Interactive Robotics

Coach name: Koen Hindriks

Coach affiliation: TU Delft, Department of Intelligent Systems, Interactive Intelligence

Contact

Contact person 1: Koen Hindriks, k.v.hindriks@tudelft.nl

Contact person 2: Marcel Kuipers, m.e.kuipers@tudelft.nl

The final report for this project can be found at: <http://repository.tudelft.nl>



Research Report

On the next few pages the research report can be found. The research report is a summarization of all the research that was done in the first two weeks of the project.

DELFT UNIVERSITY OF TECHNOLOGY

BACHELOR END PROJECT

BLOCK BASED PROGRAMMING LANGUAGE FOR THE NAO

Research Report Robot Block-based Programming

Authors

Robin van der Wal

Luka Miljak

Jannelie de Vries

Marcel Kuipers

June 22, 2017



Contents

1	Introduction	2
2	Environments for teaching robotics and/or programming	2
2.1	scratchJr	3
2.2	Scratch	3
2.3	Alice	3
2.4	Choreographe	3
2.5	LEGO Mindstorms	3
2.6	Open Roberta	4
3	Capabilities of Blockly	4
3.1	Blockly composition	4
3.2	Similar use cases	5
3.3	Working with Blockly	6
4	Design choices of block based languages	7
4.1	Blocks	7
4.2	Layout	8
5	Children learning capabilities	9
5.1	Children of grade 3 and 4	9
5.2	Children of grade 5 and 6	10
5.3	Children of grade 7 and 8	10
6	Design of an intelligent agent	11
6.1	Intelligent agents in the context of children	12
6.2	Translating BBPL programs into intelligent behavior	13
7	Conclusion	13

1 Introduction

This report aims to investigate what is important when designing a Block-based programming language (BBPL) for children. The BBPL that will be created for this project will allow users to create interactive programs for the NAO robot. Not only the design of the language, but also the behavior of the robot is considered. In the last 15 years have elementary schools started to introduce concepts of programming and computer science to their students and a need for teaching methods exists. Since children that are learning to program have very different needs than adults [1], creating a programming language must be done with their needs in mind.

This project was commissioned by the company Interactive Robotics, a Dutch company which aims to create a robotic interaction engine. One of the applications of this engine will be teaching children about interaction with robots. For this, it is necessary to provide a framework in which this interaction is intuitive and easy to learn. Many important aspects of this topic are addressed in this paper.

The main question that is answered is: "How do you teach children about robotics and programming using the NAO robot and a BBPL?". This question will be answered by first looking at previous work done in the field of teaching robotics and/or programming. Then, because the Blockly library will be used, its capabilities are examined. After that, design principles for BBPLs are considered. The section after that is dedicated to outlining the capabilities of children from different grades. Then, a section is written about the design principles for an intelligent agent that will interpret the created BBPL. Finally, the conclusion of the paper is given.

2 Environments for teaching robotics and/or programming

It is useful to take a look at previously made efforts to teach robotics and programming using BBPLs. Many such projects exist, with different purposes aimed at different age categories.

2.1 scratchJr

This language aims to teach kids age 5-7 to program. Program code is written from left to right rather than from top to bottom [2]. Text blocks contain an image and optionally a number. The blocks do not contain any text. The most “advanced” action is a for-loop. The language contains specific “start blocks” that decide when a script is executed. This can happen when a sprite is touched or when the block itself is pressed.

2.2 Scratch

This is an extended version of the scratchJr language. It is aimed at kids of age 8 and up [3]. Unlike scratchJr does execute from top to bottom. It contains most of the mathematical and logical operations you would expect from a programming language. It is used in elementary schools in the United States. The environment is quite limited, with a focus on logic. The usage of multiple objects is introduced by allowing the starting character, a cat, to clone itself.

2.3 Alice

This language is quite advanced and is aimed at high school students that will soon start their first Computer Science classes [4]. It allows users to create videos and games. It allows the creation of multiple items of different types, making the language sort of an introduction to Object Oriented Programming.

2.4 Choreographe

This is a programming environment for the NAO robot [5]. Its complexity makes it difficult to use as a learning tool for kids. It is a useful language to look at, as it shows the capabilities of the NAO robot. Its design, with blocks and lines that represent connections between blocks, allow for very complex applications, which are outside of the scope of this project.

2.5 LEGO Mindstorms

LEGO Mindstorms allows the building of robots from LEGO blocks and allows you to program them. Multiple languages exist for this purpose, includ-

ing some graphical ones, like RXC Code and ROBOLAB. LEGO Mindstorms is a very popular tool in schools to teach about programming [6].

2.6 Open Roberta

This is a Blockly-based language to program the NAO robot in, that aims to teach children around age 10 to 16 [7]. It has some advanced functionalities, such as function specification and variables. Also, there are extensive possibilities to detect sensor input.

3 Capabilities of Blockly

The Blockly framework is one of the two core components of the product. In this chapter, the capabilities and limitations of Blockly are explored. First, the framework itself is examined, then a few use cases similar to this product are looked at in order to get a grasp on possible competition and lastly a few limitation of Blockly are considered. In each category Blockly is evaluated in terms of this project.

3.1 Blockly composition

Blockly is an open-source JavaScript library that adds a block-based virtual code edit environment made by Google [8]. Blockly adds several different components that are useful for this project. Each component is briefly explored here.

Firstly, the main function of Blockly is to add a workspace for code editing. This workspace consists of a toolbox and a work area. The Toolbox contains the code Blocks the user can use. The user can drag multiple blocks to the work area. Blocks can fit into each other, and combining multiple blocks creates a “program.” There is usually a bin where the user can drag blocks to delete them. Blockly allows restrictions on what blocks the user can use. Blockly already has some predefined blocks that allow for basic programming.

Secondly, Blockly allows developers to define custom Blocks. These custom blocks can have multiple connections of different types, giving a lot of freedom in the creation process. The custom blocks can have their own text, images, and input types. Google has a web-page for creating custom

Blocks using the Blockly framework, which facilitates making new blocks [9]. Custom blocks make it possible to adapt the Blockly framework in robotic programming.

Thirdly, after a user has created a program, this program can be converted to code. Blockly itself supports multiple languages such as JavaScript, Python, Lua and Dart. Code generation is done by a generator script. This script has to be defined per code Block, including for each new Block. This allows Blockly to generate the type of output that the developer requires.

Finally, after a user has made some process with his or her code Blocks, the code can be saved to an XML output. This XML output can then be used to later reload the process of the user, or for future analysis.

All these features can be used for making a visual BBPL for programming robots. The workspace and toolbox allow for creating assignments for children. Next the custom blocks makes it possible to create blocks with images, so they are more suitable for young children that might have difficulty reading. Furthermore, the code generation can be made to match the robot programming interface. Lastly, the saving feature allows for users to keep track of their process while programming. All in all, these features make Blockly a suitable framework for this project.

3.2 Similar use cases

Blockly has been used before in programming robots in elementary school. In the article by M. Saleiro, et al. [10], the authors used Blockly in conjunction with small low-cost robots. These robots were constructed from simple micro-controllers, a small motor, some sensors and a sender. With the help of Blockly the authors taught 8 year old children how to program simple actions and movement for the robots. Using an exercise about robot movement and a map, the authors were able to successfully teach the children about robotics and geography.

In other research C. Martinez, et al. [11], examined whether it was possible to teach children of ages 3 to 11 basic programming functions such as conditionals and loops. Here Blockly was used as well to program a simple robot. They were successful in teaching the basic concepts in the age categories 5 to 11 years old, which is similar to the age range of 6 to 13 used in this project.

While these use cases are similar, they were both done with simple robots. In this project a NAO will be used. A NAO robot is a complex humanoid robot. There is a Blockly-based programming language for the NAO: Open Roberta. Open Roberta has an implementation of all basic features that the NAO comes with. It has blocks for each sensor and multiple handy blocks such as the “Wait for” block. The programs can be exported to a NAO for execution.

Open Roberta is very good for users who are familiar with robots and programming. But using it as an educational tool for primary school won’t work. Open Roberta is a sandbox that gives all of the options of the NAO. Options like “get electric current of *head yaw*” won’t be used in a classroom. Someone who never programmed before will likely get lost. Though, there are good parts too. The command blocks are generic and precise. And the extra image guide for all the NAO sensors clears up where each sensor is. This makes Open Roberta useful to look at when examining the capabilities of NAO programming.

Thus there have been multiple successful attempts on teaching robotic programming to children of elementary school with Blockly. There However, using a humanoid robot to achieve this task hasn’t been done before in elementary school.

3.3 Working with Blockly

Blockly is highly customizable, but there are a few notes one must pay attention to when using framework. The most important one is that the framework itself is not a language, as it is stated on the Blockly website [12]. As a result, just using the Blockly framework is not enough. The robotic programming language has to be created separately from Blockly and only then Blockly can be used to create robot programs. The extra time this process takes should be accounted for.

Furthermore, a feature that is often missed in Blockly is the ability to customize Block shapes. The shapes are predefined in the Blockly library as Scalable Vector Graphics. It is possible to edit these SVG definitions. But, due to time constraints, this project will employ the standard Blockly shapes as the focus of this project is teaching children robotic programming and not creating the perfect looking Blocks. Nonetheless, these two restrictions do not prevent the Blockly framework from being useful.

4 Design choices of block based languages

There are already a lot of BBPLs in existence. A few examples of these languages have been mentioned in section 2. This section researches the design choices that are made for several BBPL. This section considers the following languages: Scratch [13], scratchJr [14] and open-roberta [15]. Blockly is also considered, although it is not a language, but a framework to create languages. Since all the languages above are all based on the Blockly framework, some general guidelines are still included. This section considers the blocks used for the developer tool, to create new blocks [12]. Each language made choices in their design. This section will describe some different design choices that these languages made.

4.1 Blocks

Languages that use the Blockly framework have different design principles. But one thing all these languages have in common, is that the blocks are based on jigsaw puzzles so that the blocks fit nicely together. This makes it clear for users how the different blocks fit together. Some languages have an extra way of showing which blocks fit together, by highlighting the ends from other blocks to which the selected block can connect to. This is an effective way of showing the possibilities of how the blocks can be connected [16].

Images are sufficient enough in explaining the behavior of simple blocks. This helps young children that might not be able to read yet. When you hover over the blocks a textual explanation is given. In this way, an extra explanation is given, when the image is not clear enough for the user. For more advanced blocks, additional text is displayed on the block itself when an image is not sufficient enough to deduce the meaning of the block. Blocks such as if-then-else statements or loops always have a C-shape. An example of such a block as used by Blockly [12] is given in figure 1. In this way it is clear that blocks inside this shape belong to the block around it. The inside of these advanced blocks should have a puzzle shape, in order to make clear what other blocks are available to be put inside it. A study done by Blockly researchers [16] showed that there is a trade off to be made: the lower part of the C-shape can either have a puzzle shape or not. When it has such a shape, it becomes more clear which blocks fit, but it is not immediately clear that multiple blocks can fit between the C-shape. When the lower part doesn't



Figure 1: C-shape block from Blockly [12].

have such a shape, it may not be as clear which blocks fit, but it is clear that multiple blocks can put in between the C-shape. scratchJr [14] chose to have a puzzle shape at the end, but Blockly chose to not have such a shape and solve the vagueness by making the ends highlighted when a shape can fit under it.

There are multiple ways for a block to ask for input. You can ask input next to the block or inside the block. An example of these blocks is given in figure 2. On the left a block is shown where the input is expected outside the block, while on the right a block is shown where the input is expected inside the block. Both blocks are from open-Roberta.



Figure 2: left is a block asking for input next to the block, right is a block asking for input inside the block. Both blocks are from open-Roberta [15].

4.2 Layout

The layouts of the above mentioned languages are different, but also have some similarities. Every language has a toolbox and when clicked the blocks available will show up, either above, below or next to this toolbox. The available blocks, are split into categories and with the use of the toolbox, you can choose the different categories. Each category has its own color and the blocks have the color matching their category.

Next to the toolbox is the canvas where the different blocks can be dragged, dropped and connected to each other. An important difference between

scratchJr. and Scratch, is that scratchJr connects the blocks in a horizontal way, while Scratch connects the block vertically [14]. Because the blocks can be quite wide, a vertical approach seems better. However it was shown that young beginning programmers find it easier to understand horizontal programming. Once they have a little experience, the vertical programming way is not a problem [17].

When programming, sometimes the wrong block is placed and it should be removed. In scratchJr there is no way of deleting the block, other than dragging it out of the screen, while Blockly [8] has a garbage can where you can drag your blocks into.

5 Children learning capabilities

In 2006 the Dutch government decided on several core goals primary school students should have reached at the end of grade 8. Some example are, For The Dutch language: children should be able to get information from spoken language. For English, they should be able to write simple daily used English words. For math they should be able to add, subtract and multiply. All other goals the government decided primary school student student should have are stated in "Het kerndoelenBoekje" [18]. Each school is able to decide how they teach these goals to their student, but the basic curriculum per school is the same.

5.1 Children of grade 3 and 4

In grade 3 and 4 the students get the basic knowledge needed to learn more advanced information in the following classes.

In grade 3, kids start learning. They have to sit still and concentrate. The first steps of reading and writing are made. Kids are able to read one line sentences. They also learn math. They are able to add and subtract numbers to 20. In grade 4, kids learn to read consecutive sentences, furthermore they learn basic grammar. Kids learn multiplication for the numbers below 6.

There is already a curriculum for primary school children in the Netherlands for programming [19]. Per grade, there are multiple lessons available, for either unplugged exercises, where no robot is used and exercises where a robot is used. These exercise have simple sentences explaining the task. The

exercises with a robot have simple task, like go one step forward, turn to the left etc. The block and or buttons don't show any text, but have a picture explaining what it means.

5.2 Children of grade 5 and 6

In grade 5, kids learn to write and read longer sentences. Math exercises become a bit more difficult. In grade 6, fractions and divisions are introduced. Language skills are improved.

In [20], a programming curriculum is proposed for Dutch schools. This curriculum gives an insight into what kinds of programming skills should be expected from Grade 5 and 6. Focus is put on abstract reasoning and problem solving. By now, kids are ready to learn more complex programming concepts. Kids should be able to work with if-statements in a program. Also, they need to be aware of the concepts of "functions" and "variables".

Programming languages such as Scratch become accessible to kids, though some of its advanced features might still be too complex. These languages allow complex behavior without the definition of functions and variables. Scratch also introduces more complex concept, such as concurrency, in a soft and safe manner. The design of the language still needs to prevent the occurrence of error messages. Early forms of boolean logic are understood. Actions that make use of text fields can be used. Typing numbers is also possible, making arithmetic expressions possible.

5.3 Children of grade 7 and 8

In the 7th grade, reading and the Dutch language is even further explored. Children learn to sum numbers up to 1,000,000. Fractions, decimals, percentages, averages and ratios are introduced. The metric system of length, area, volume and weight is also explained. The 8th grade continues on this; learning the language becomes more difficult and they explore the bigger areas in math such as the circumference of a circle.

In a paper Maya Sartazemi, et al, compared two programming languages for a LEGO robot and did experiments with 3rd year junior high school students in Greece [21]. These students are approximately 13-14 years old, slightly

older than the target group discussed in this section. However, using interpolation with the information from the experiment and with what is stated above on grade 5 and 6 can give insight on the difficulty level for grade 7 and 8. The experiments made use of the programming languages ROBOLAB and Not Quite C. Since this paper is mainly in the context of block-based programming, it would be fitting to only include the results of ROBOLAB. While ROBOLAB isn't exactly Block-Based like Blockly, it has a similar graphical concepts.

One of the first exercises in the experiment was to program the Lego vehicle to move forward for 4 seconds, to stop for 2 seconds and then to move in the opposite direction for 4 seconds. 83% of the students had a correct answer (of the 91% who actually tried the exercise). This implies that these student understand simple actions like moving forward, rotating and using time as parameter. Another exercise that was done very well by most student (92% of 96%) is to produce an audio signal every second, for a total of ten times. It can be concluded that loops are also a concept that is understood. The other exercise shows competence towards if, then, else statements.

The proposed programming curriculum [20] suggests that for grade 7 and 8, children learn how to use repetitions in combination with conditions. Children should also learn to come up with conditions (if, then, else statements) themselves. Functions are learned to be combined with parameters and variables. Variables should be learned to be used in abstract situations.

This curriculum should however not be used as a standard for what the children know. First of all, the children may just have started learning about these things, so they may not know things like functions with parameters yet. Secondly, programming is not an actual course in the primary education yet. So most children who will start playing with the robot, are programming for the first time. The proposed curriculum assumes that children from grade 7 and 8 can do the things children from 5 and 6 learned, which is not true if the school just started adopting the curriculum.

6 Design of an intelligent agent

When working with interactive robots, it is necessary to design some sort of "intelligent" behavior. This project will use the NAO, which is a infant-sized humanoid robot. The NAO will fulfill the roll of the interactive robot. First,

through other examples with interactive robots with children, some base line actions for the NAO are proposed. Secondly, it is examined how the NAO should interpret the BBPL programs the children create.

6.1 Intelligent agents in the context of children

Robots will play an interactive role in society. Exposing children to robots early will help prepare them for this future. Two robots that were used in this context are examined.

One of the first robots that were used in an interactive roll with children was the Robovie [22]. The Robovie was used in children-robot interaction research at a Japanese elementary school. The robot had basic behavior such as shaking hands and giving hugs. The robot could also speak, but only speak and recognize English. The goal of the robot was to teach children some basic English words. Two exams at the begin and after the two week test period with the robot, revealed that it did have a positive impact on the English understanding. The research did not include a control group, but it showed the possibility of positive effects. The researchers noted that the interaction and idle movements the robot did were main reasons for children paying attention to the robot [23].

A more recent example of a robot in an elementary classroom was the DragonBot. This bot was designed to teach children about nutrition. Over a 3 week period, children had to socially interact with the robot and present food items to it. The DragonBot would respond and would prefer healthy foods in its interactions. After the test period, children took more time to consider what food to choose and it was shown that the children kept showing an extremely positive attitude towards the robot [24].

Based on the research above, NAO should be interactive for children to keep them engaged. There should be a set of idle actions and movement that occur when the robot is not busy with execution a BPPL program. Furthermore, the NAO should greet children when starting up, be interactive during the assignments and say goodbye when shutting down. Both works weren't able to show how effectively the robots were able to teach the children. Thus when designing a robot programming teaching system, it is important to keep testing the system with the users and keep track of progress.

6.2 Translating BBPL programs into intelligent behavior

In this project, the programs created using the BBPL will not be directly translated into machine code. Instead, an intelligent agent needs to interpret it and perform actions in a human-like manner. The works in the previous section both mentioned that it is important that for the robot to be interactive in order to keep the children interested. An agent is better suited for this as it can adapt autonomously based on the situation, see the following scenarios for examples:

Firstly, the NAO shouldn't always immediately start executing a program it has received from a child. When a human receives an assignment, it will first get ready to execute it. So the NAO should only start executing the program when it is ready and should give feedback before doing so.

Another example is giving reactions to the success of an exercise after running a program. The robot could give positive feedback when the children succeed in finishing an exercise. The NAO has no facial expressions, but with the help of its LEDs and body posture it can give a sense of emotion, as can be read in this paper by M.A. Miskam [25].

Negative feedback when failing an exercise should be avoided and perhaps an explanation could be given by the NAO. An explanation would mean that the NAO tries to analyze the reason why it couldn't continue with a program. For example, if the program said that the NAO has to move forward, but something is in the way, then the robot stops the program and tells the child.

7 Conclusion

Many BBPLs already exist, for different purposes and age categories. Some are aimed at young children and some are used for programming robots. Blockly provides a lot of possibilities for creating a BBPL in the context of this project. This is mainly due to its high customizability, although some changes are very difficult to implement. Many different design considerations exist, such as how blocks connect, the shapes of blocks, the usage of colors and the layout of the menu. The learning capabilities of children are documented, allowing for the categorization of programming concepts that they can be brought into contact with. Research on Human-Robot Interaction provides some ideas on how to design an intelligent agent. All in all, this report

provides enough information to design a BBPL and a NAO robot agent in order to teach children about robotics and programming.

References

- [1] V. Barr and C. Stephenson, “Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community?” *Acm Inroads*, vol. 2, no. 1, pp. 48–54, 2011.
- [2] A. Strawhacker, M. Lee, C. Caine, and M. Bers, “Scratchjr demo: A coding language for kindergarten,” in *Proceedings of the 14th International Conference on Interaction Design and Children*. ACM, 2015, pp. 414–417.
- [3] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, “Scratch: programming for all,” *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.
- [4] S. Cooper, “The design of alice,” *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 15, 2010.
- [5] S. Krivanec, A. Petrácková, T. Thi, P. Linh, and D. Pruša, “Nao robot applications developed in choregraphe environment,” *Czech Technical University Faculty of Electrical Engineering Department of Cybernetics*, 2010.
- [6] A. J. Hirst, J. Johnson, M. Petre, B. A. Price, and M. Richards, “What is the best programming environment/language for teaching robotics using lego mindstorms?” *Artificial Life and Robotics*, vol. 7, no. 3, pp. 124–131, 2003.
- [7] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, “Graphical programming environments for educational robots: Open roberta-yet another one?” in *Multimedia (ISM), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 381–386.
- [8] Apache Software Foundation, “Blockly,” <https://developers.google.com/blockly/>, [Software].

- [9] P. in het PO, “Blockly developer tools,” <https://blockly-demo.appspot.com/static/demos/blockfactory/index.html>, 2017, accessed: 2017-05-01.
- [10] M. Saleiro, B. Carmo, J. M. Rodrigues, and J. H. du Buf, “A low-cost classroom-oriented educational robotics system,” in *International Conference on Social Robotics*. Springer, 2013, pp. 74–83.
- [11] C. Martinez, M. J. Gomez, and L. Benotti, “A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform,” in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 2015, pp. 159–164.
- [12] Google, “Blockly for developers,” <https://developers.google.com/blockly/>, 2016, accessed: 2017-05-02.
- [13] Lifelong Kindergarten Group, “Scratch,” <https://scratch.mit.edu/>, [Software].
- [14] Code-to-Learn Foundation, “Scratch-jr,” <http://www.scratchjr.org/index.html>, [Software].
- [15] Fraunhofer IAIS, “open-roberta,” <https://lab.open-roberta.org/>, [Software].
- [16] N. Fraser, “Ten things we’ve learned from blockly,” in *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*. IEEE, 2015, pp. 49–50.
- [17] M. M. Lab, “Scratch google = next generation of programming blocks for kids,” *Medium*, May 2016.
- [18] ministerie van Onderwijs Cultuur en Wetenschap, “Kerndoelenboekje,” apr 2016.
- [19] K. redactie, “Codekinderen,” <http://www.codekinderen.nl/>, 2015, accessed on: 26-04-2017.
- [20] P. in het PO, “Programmeren in het po,” kn.nu/leerlijnprogrammeren, 2016, accessed: 2017-04-26.

- [21] M. Sartatzemi, V. Dagdilelis, and K. Kagani, “Teaching introductory programming concepts with lego mindstorms in greek high schools: a two-year experience,” in *Service Robot Applications*. InTech, 2008.
- [22] H. Ishiguro, T. Ono, M. Imai, T. Maeda, T. Kanda, and R. Nakatsu, “Robovie: an interactive humanoid robot,” *Industrial robot: An international journal*, vol. 28, no. 6, pp. 498–504, 2001.
- [23] T. Kanda, T. Hirano, D. Eaton, and H. Ishiguro, “Interactive robots as social partners and peer tutors for children: A field trial,” *Human-computer interaction*, vol. 19, no. 1, pp. 61–84, 2004.
- [24] E. Short, K. Swift-Spong, J. Greczek, A. Ramachandran, A. Litoiu, E. C. Grigore, D. Feil-Seifer, S. Shuster, J. J. Lee, S. Huang *et al.*, “How to train your dragonbot: Socially assistive robots for teaching children about nutrition through play,” in *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*. IEEE, 2014, pp. 924–929.
- [25] M. A. Miskam, S. Shamsuddin, H. Yussof, A. R. Omar, and M. Z. Muda, “Programming platform for nao robot in cognitive interaction applications,” in *Robotics and Manufacturing Automation (ROMA), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 141–146.



Plan of Action

On the next few pages the plan of action can be found. The plan of action is, as the name says, a proposal on how to complete the project on time, with as many features as possible. The plan of action is based on the research report found in appendix D.

DELFT UNIVERSITY OF TECHNOLOGY

BACHELOR END PROJECT

BLOCK BASED PROGRAMMING LANGUAGE FOR THE NAO

Plan of Action Robot Block-based Programming

Authors

Robin van der Wal

Luka Miljak

Jannelie de Vries

Marcel Kuipers

June 22, 2017



Contents

1	Introduction	2
2	Deliverables	2
3	Road map project	5
4	Design guidelines of block based languages	8
4.1	Blocks	8
4.2	Layout	9
4.3	Creating exercises	9
5	Proposal of Block based language for Robot programming	10
5.1	Children of grade 3 and 4	10
5.2	Children of grade 5 and 6	10
5.3	Children of grade 7 and 8	11
6	Provisionary System Decomposition	12
6.1	Subsystem Decomposition	12
6.2	This Project's Focus	14
7	Quality Assurance	14
7.1	Scrum and User Tests	15
7.2	Code Tests	15
7.3	Regression Tests	16
7.4	Code Review	16
8	Definition of Done	16
8.1	Backlog Items	16
8.2	Sprint	17
8.3	Releases	17

1 Introduction

This document contains the plan of action for the BEP (bachelor end project). In this project, a Block-Based Programming Language (BBPL) for the NAO robot will be created with the intention of teaching children about robotics and programming. The NAO robot is different from most other robots in that it has a humanoid shape and has many possibilities for interaction. Therefore, the design of interaction and intelligent behavior play a big role in this project.

This project was commissioned by the company Interactive Robotics, a Dutch company which aims to create a robotic interaction engine. One of the applications of this engine will be teaching children about interaction with robots. For this, it is necessary to provide a framework in which this interaction is intuitive and easy to learn. In this document it is outlined how this will be done for this particular project.

Firstly, a section is designated to giving an overview of all the deliverables of the project, which will be presented using the MoSCoW method. After that, the design decisions for the BBPL are discussed. This is followed by a proposal of the contents of the language, per grade. For grade 3-4, 5-6 and 7-8, it is explained and justified what programming concepts will be introduced and in what sorts of contexts they will be used. Then, a road map will be drawn, showing the planning of the project. A provisional system decomposition can be found after that. This section contains a small explanation of each component in the project. The document finishes with a section on Quality Assurance.

2 Deliverables

This section will describe the deliverables of this project. The high-level features will be defined using MoSCoW. MoSCoW uses four categories to separate the features by level of importance. The categories are:

Must Have: Features that are of high importance. With those features there isn't a product

Should Have: Features that are considered favorable. These features should be included, but are not necessary for the basic product.

Could Have: Features that are of low importance. When there will be enough time to implement, these features will be present.

Won't Have: Features that are of no importance or not doable. These won't be implemented. However, these features are free to be implemented by the company instead. If such an item is created by the company then it *should* be integrated in the project.

Must Have

- There has to be a Block based language for grades 3-4, 5-6 and 7-8 (Dutch school system). These languages have to be specially designed for these grades, based on what the children are able to understand.
- For each grade there has to be at least three exercises to program the NAO robot. These exercises teach programming in an interactive way.
- There has to be a web environment for children to program the NAO.
 - The web environment has to contain a canvas where you can make your program.
 - the web environment has to contain a block library specially for the current assignment or grade level.
 - It has to be possible to select a block from the library and drag it toward the canvas.
 - It has to be possible to remove unused blocks from the canvas.
 - It has to be possible that the robot executes the program, so the children are able to see what their program does.
 - A child has to be able to go through the program step by step. So the robot executes a single step of the entire program. When doing this the current step of the program should be highlighted on the screen.
 - A child has to be able to pause the program.
 - A child has to be able to start the program.
 - A child has to be able to stop the program.
- There has to be a GOAL agent.
 - The GOAL agent has to be able to interpret a given program.

- The GOAL agent has to be able to execute the program and send tasks to the robot.

Should Have

- GOAL agent
 - The Goal agent should be able to get sensor data from the robot. For example if a sensor is touched, the GOAL agent should receive this.
 - The GOAL agent should be able to process the received sensor data, either by sending something to the child, or something to the robot.
 - The GOAL agent should be able to execute the given program in a smart way, for example when it is interrupted, it should explain the cause to the child.
 - the GOAL agent should be able to evaluate the solution the child gave for a certain exercise.
- There should be at least 5 exercises in total per grade (3-4,5-6,7-8).
- It should be possible to save and load partial solutions to exercises, so the children can continue their exercise another time.
- There should be an environment where teachers can create their own exercises.
 - There should be an environment where teachers choose the blocks needed for an exercise.
 - There should be an environment where teacher can create a possible solution for the exercise.
 - The exercise should be saved.
 - Teachers should be able to select exercises for children to work on.

Could Have

- There could be more exercises for each grade (3-4, 5-6, 7-8). In this way children have more exercises to learn from.

- The block language supports the facial recognition features of the NAO.
- There could be a virtual robot in the web-client. In this way, more children can learn to program, because less actual robots are needed.
- The Block language and the agent support recognition of objects.
- There could be a monitoring system, that monitors the behavior of the children and teachers when they use the web-client. The results could be saved in a database.

Won't Have

- There won't be new moves for the NAO, because there is no access to the right tools or the cloud system.
- There won't be a text editor to program the robot, because the idea of this project is to create a visual environment for pupils.
- There won't be any customizable shapes of blocks, because the basic Blockly blocks should be enough for the NAO actions and creating additional Blockly functionality and/or blocks is not the focus of this project.

3 Road map project

This section describes a high level planning for items stated in section 2.

The process is split into several phases. The first phase is the **concept** phase, during the first two weeks. During this phase research is done and a plan of action for this project is created.

Then comes the **basic features implementation** phase in week 3 to 5. Basic features will be implemented, such as sending the code towards the GOAL agent and the GOAL agent should be able to send the task to the NAO. The aim is to have at least finished the *must haves* from the deliverables.

Next a more advanced agent is needed, so week 6 to 8 will be the **advanced features implementation** phase. In this phase the more advanced

features will be implemented, like dealing with feedback from the NAO and testing the given solution. The aim is to have at least finished the *should haves* from the deliverables and possibly some *could haves*. In this phase the final report or in other words thesis is started on.

The last phase is the **finalization** phase in week 9 to 11. In this phase the final product needs to be delivered, as well as the final report. Lastly, a presentation has to be given on the project as a whole.

week 1 (24 April - 28 April) [Concept]

Research report.

Research learning capabilities for group 3-4.

Research learning capabilities for group 5-6.

Research learning capabilities for group 7-8.

Create a toolbox to select blocks from.

Create a canvas to make your program.

Implement block deleting from the canvas.

week 2 (1 May - 5 May) [Concept]

Research report.

Plan of action.

Create a Block based language for group 3-4.

Create a Block based language for group 5-6.

Create a Block based language for group 7-8.

3 exercises for group 3-4.

3 exercises for group 5-6.

3 exercises for group 7-8.

week 3 (8 May - 12 May) [Basic features implementation]

GOAL agent which is able to send the actions to the robot.

Basic Ontology based on the blocks available.

Design of basic pipeline with TECS server.

Design template for the robot images used in the blocks in the Blockly interface.

week 4 (15 May - 19 May) [Basic features implementation]

Basic pipeline set up with TECS server:

- GOAL agent which is able to send the actions to the robot.

- GOAL agent which is able to receive feedback from the robot.
 - Web client that is able to sent basic programs, single moves, to the GOAL agent.
 - GOAL agent able to sent feedback to the web client.
- 2 extra exercises for group 3-4.
 2 extra exercises for group 5-6.
 2 extra exercises for group 7-8.

week 5 (22 May - 26 May) [Basic features implementation]

GOAL agent which is able to send the actions to the robot such as idle actions, start up and shut down actions.
 GOAL agent which is able to receive feedback from the robot based on execution of the robot moves as percepts.
 Implement step by step program execution.

week 6 (29 May - 2 June) [Advanced features implementation]

Upload code to SIG.
 GOAL agent which is able to send the actions to the robot when a program is interrupted for basic reasons such as waiting to long on an input task.
 GOAL agent which is able to process the sensory data from the robot used in the program exercises.
 Design teacher interface system together with company.
 First field test with basic features of robot in a school.
 Create proposal for content of the final report.

week 7 (5 June - 9 June) [Advanced features implementation]

Implement teacher interface.
 Implement testing solutions given by the child.
 Start writing final report, incorporating existing documentation.

week 8 (12 June - 16 June) [Advanced features implementation]

GOAL agent is able to provide feedback on incorrect solutions.
 Write content for final report, based on new sections.
 Second field test with advanced features such as program evaluation with robot in a school.

week 9 (19 June - 23 June) [Finalization]

Last field test with finished advanced features such as an feedback system with robot in a school.

Finalize all code implementations.
Finalize final report.

week 10 (26 June - 30 June) [Finalization]

Deadline final report.
Deadline info sheet.
Upload final code to SIG.
Prepare presentation and demo of presentation.

Week 11 (3 July - 7 July) [Finalization]

Presentation.
Upload thesis to repository.
Put library link on BEPsys.

4 Design guidelines of block based languages

As shown in the research report in section design choices of block based languages, there are multiple block based languages and each has made their own design choices. This section will combine those design choices into the design guidelines that will be used for the project.

4.1 Blocks

The languages described in section design choices of block based languages from the research report all had jigsaw puzzle shaped blocks in order to make it clear the pieces fit together. This will be the same in this design. To make it even more clear, the design of this project will contain highlighted ends for the pieces where the block fits when the block is nearby.

For a simple block, the design of this project will contain images, so children who are not able to read, still can understand the meaning of a block. When hovering over a block a textual explanation will be given. For more advanced blocks, text is displayed on it when an image is not sufficient enough. Blocks such as “if, else” statements or loops will use the C-shape as used by Blockly [1], without the puzzle shape at the bottom of the statement input.

This design, when an input is needed, will have the input inside the shape

as used by Open-Roberta [2]. In this way it is clear for which the input is needed.

4.2 Layout

The layout of this design will contain a toolbox with the blocks on the side of the canvas. The toolbox will contain different categories depending on the type of blocks used. For example, an action block or logic block. The blocks have the same color as the toolbox item it belongs to.

From the toolbox you can drag and drop your block onto the canvas. the blocks are connected vertically. This design does contain horizontal connection for inputs needed for certain blocks, such as the “if” block. There will be an trashcan in the right bottom corner, so blocks can be deleted if needed. The canvas will be partly fixed, so it is not possible to drag blocks outside the canvas. In this way children cannot lose their blocks. The most important part of the design of our canvas is that it will always contain a start block. So children know where to start programming. It is possible to move this block, but it can’t be removed.

4.3 Creating exercises

A teacher might want to create his or her own exercises, in order to make the allow changes to the proposed curriculum. In order to make this possible, the web-page should contain a special page for teachers, where they can create there own exercises. It should be possible to select the blocks available during the exercise and a possible solution should be given to check whether or not the given solution is correct. These exercises should be savable.

An exercise should contain a grade label. This label shows if this exercise is meant for children from grade 3-4 or grade 5-6 or grade 7-8. The exercise should contain a title, explaining the goal of the exercise. It should contain the blocks needed for the exercise and it should contain an explanation or step by step guide to execute the task. The most important part is that there should be at least one solution in order to check if the given solution is correct.

5 Proposal of Block based language for Robot programming

Based on sections 4 and from the research report section children learning capabilities, a proposal is made for block based programming languages for grades 3-4, 5-6 and 7-8.

5.1 Children of grade 3 and 4

In the research report section children learning capabilities it was shown that children from grade 3 and 4 are able to read simple and small sentences and will program based on images. In this report in section 4, certain design guidelines were made. In this section it was decided to split the blocks based on their category and give them colors accordingly.

A language for grade 3 and 4 should only contain simple blocks with images. These simple blocks are the basic movements the robot can make, for example, walking, waving, dancing, standing, sitting etc. It also contains a block for looping certain block, but no other advanced blocks.

An example of an exercise could be to program the robot to dance the “head, shoulders, knee and toe” dance. the child can do this exercise by simply putting the blocks needed for these actions after each other. If the exercise is finished, the child can dance with the robot.

5.2 Children of grade 5 and 6

Reading, writing and problem solving skills have improved greatly. New blocks for grades 5 and 6 are introduced. Most importantly, the if-statement is introduced (but not the if-else-statement).

The inclusion of the if-statement comes paired with the introduction of boolean expressions. “And”, “not” and “or” blocks are implemented, but the exercises will not require extensive usage of these blocks. Example usage would be a simple exercise such as “If the left AND right shoulder are touched, let NAO walk forward for 2 seconds.”

Movements made by the robot become customizable. Where grades 3 and 4

would have separate actions “raise left hand” and “raise right hand”, grades 5 and 6 get one action called “raise hand”. This action will have a drop-down menu that allows you to choose the hand that needs to be raised. For the action “move forward”, it will be possible to define how much the robot will walk forward.

A bigger focus is put on interactivity. Blocks that make use of the sensors of the robot make this possible. An example exercise for this age category would be a clapping game. In this exercise, the robot needs to first put up his left hand and wait until it touched. After that, he needs to put up his right hand and wait until that one is touched. This process can then repeat.

5.3 Children of grade 7 and 8

It’s safe to say that everything that the grade 5 and 6 children get, the grade 7 and 8 children will also get, plus some more. While grade 5 and 6 and 3 and 4 children have an easy to understand block like “loop x times,” the grade 7 and 8 will get two more complicated blocks: the *while* block and *until* block. The while loop keeps repeating an action *while* a certain condition is true, the other keeps repeating an action *until* a certain condition is true. The idea is that these children should be able to create conditional loops as explained in section children learning capabilities from the research report.

Another block is the “if-then-else block.” While 5 and 6 graders do get access to “if, then blocks,” the *else* statement is something new that will be included and allows for more interesting exercises.

An exercise could be to program the NAO to keep walking forward and using voice recognition to listen to the words *left*, *right*, or *stop*. This requires the student to make use of the *until* loop block.

Variables have been considered as a possibility to grades 7 and 8 because the proposed programming curriculum [3] wants these groups to use variables in abstract situations. However, since this curriculum assumes that the idea of variables have already been taught in grades 5 and 6, which isn’t currently the case. Therefore variables would have to be explained from scratch and unlike most other blocks, they aren’t self-explanatory. Thus the proposal for grades 7 and 8 is that there won’t be any variables.

Furthermore, the Blockly language should be mostly self-explanatory,

and variables is not a concept that can be used without prior instructions. Therefore variables are not included in this language. Because of the exclusion of variables, every arithmetic operation becomes obsolete as well. This is a shame, as seventh and eighth graders are actually able to do basic math. In the future, this project might be extended with variables. Thus the pipeline design will include the ability to use at least global variables.

6 Provisionary System Decomposition

This section gives an overview of the design goals and the software architecture. As the project goes forward and the code base gets larger, this section will expand over time.

6.1 Subsystem Decomposition

The system has been divided into several subsystems. These systems are shown in figure 1.

Please note that is a *current in place* system decomposition. It may change if required.

- **Child Web Interface**

The web Interface of the child is the client-side part of the web application. From here the child can send information to the server (such as their created Blockly program).

- **Teacher Web Interface**

The web interface of the teacher will be kept separate from the child interface. This is done so that the teacher can do things like create their own exercises. Like the child interface, the teacher interface will send information to the server.

- **TECS Server**

The server receives information from the clients through the internet. Information sent by children can be directly be sent to the GOAL Agent through an EIS connector. The server itself uses a Thrift Event-based Communication System (TECS). Furthermore, the server is responsible for connecting the GOAL agent to the NAO Robot.

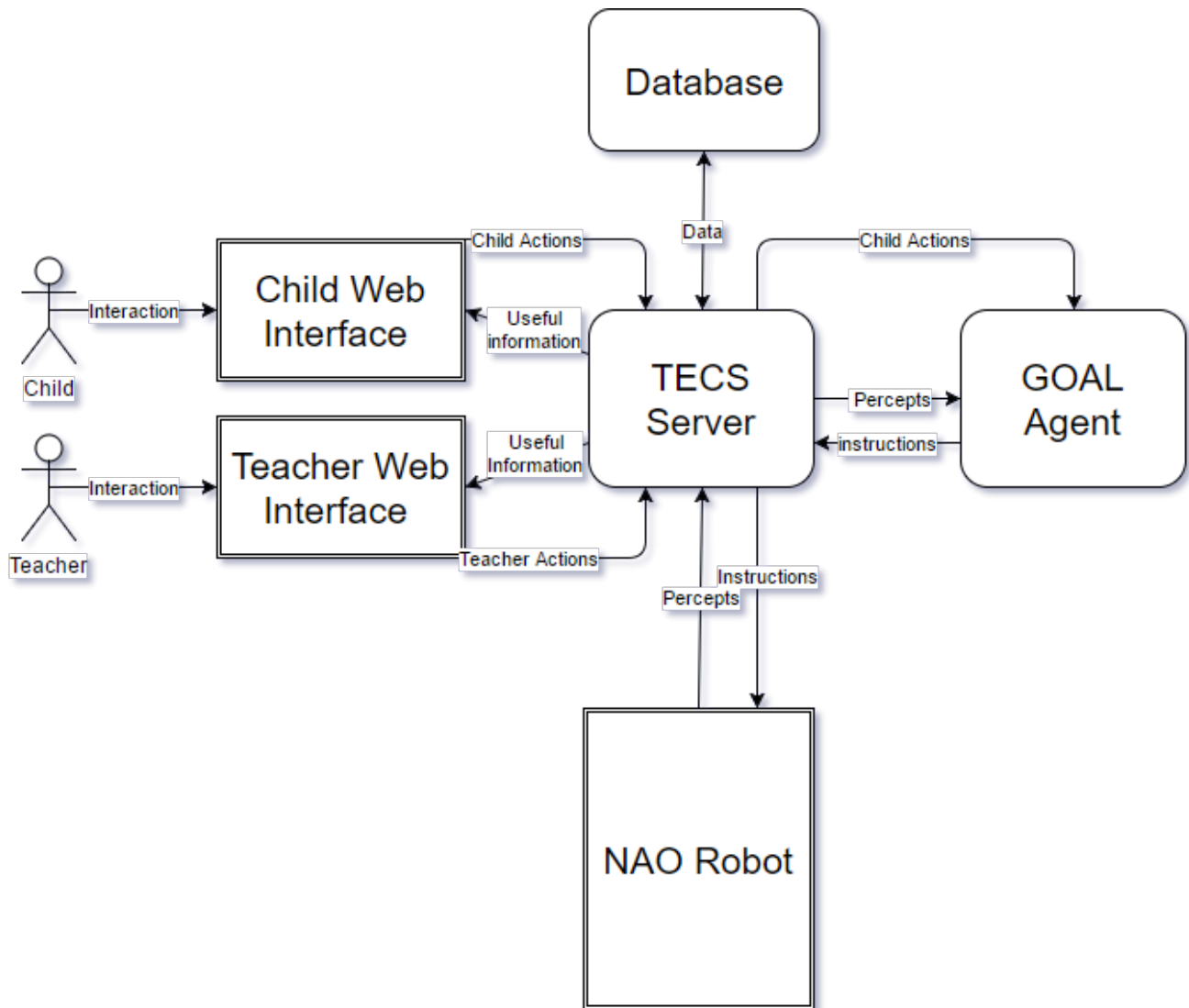


Figure 1: System decomposition of the project. The users interact with a web interface which send information to the server. Actions from the child propagate to the GOAL agent, which sends instruction to the NAO robot through the server. The robot in turn sends percepts from the environment to the GOAL agent. Teachers can create new exercises which are saved in a database with the help of the TECS server.

- **GOAL Agent**

The GOAL agent is responsible for deciding what instructions to give the NAO Robot (usually based on information received by the client and robot). The agent sends instruction to the robot (through the server) which then sends instruction to the NAO. The agent runs on the server itself and communicates with it through an EIS connector.

- **NAO Robot**

The NAO is in direct contact with the clients (children and/or teacher). It receives instructions from the server (originating from the GOAL agent), which it executes and in turn sends perceptions from the environment back to the server (which will continue towards the agent).

6.2 This Project's Focus

The system architecture is complicated, but this project won't focus on all of the parts. The following are the parts of the system that are a main focus in this project:

- Child Web Interface
- Teacher Web Interface
- GOAL Agent
- Messages sent between the web interface and the server.
- Instructions sent by the GOAL agent to the server.

Any other items depicted in figure 1 are not goals of this project. Yet they're still included in the system decomposition, as they still play an essential part in the project.

7 Quality Assurance

This section describes how the project team assures the quality of the product.

7.1 Scrum and User Tests

In order to get a high quality product that the consumer wants, the process of creating the product follows the Scrum paradigm. This means that at the end of every week, the team is expected to have a working product to show, as soon as the implementation phase starts. By doing this, the consumer can immediately try out the product and provide feedback. This feedback can then be used to improve the product even further in order to completely satisfy the customer.

7.2 Code Tests

To achieve robust and well-working code, the project team makes use of automated Unit Testing; meaning that functions of the code will be kept separate and then those functions will be tested for correct results.

It should be noted that a lot of code can't be tested for various reasons. First of all, the complete code base makes use of two different programming languages: Javascript and GOAL. The latter one, which is used to send certain instructions to the NAO Robot, is impossible to unit test. Furthermore, for a lot of parts of the Javascript code, unit tests would be meaningless. An example would be the interface of the web application, which is mostly written in Javascript.

For the parts of the Javascript code that are testable, the aim is to achieve 80% of line coverage. This means that when running the tests, at least 80% of the code lines will be executed. The tool used for unit testing Javascript code will be the Jasmine framework. Code coverage is checked using Blanket.js.

Another type of code tests are end-to-end tests. Rather than testing separate units of the code, these automated tests are directly based on user stories and look of the code has the expected behavior of what the original task was.

Unlike with unit tests, GOAL does support end-to-end tests using test2g files. These test files have simple looking and easy to write tests like "when the agent thinks it hits a wall, it should stop trying to walk". In Javascript, end-to-end tests can be created the same way as unit tests are created, using the Jasmine framework. There exists no way of getting the code coverage of GOAL programs, simply because the language is not built for it.

7.3 Regression Tests

To ensure that the product doesn't get broken after a week because of old functionality that suddenly doesn't work anymore, regression tests are used. The idea is that whenever new functionality is added included with unit and e2e tests for that functionality, all of the tests of the old functionality should be run with it as well.

7.4 Code Review

Automated tests are usually not enough to bug-proof code. One way to ensure maintainability and reliability of the code, is to do code reviews. Every time new code is added, it should be reviewed by *at least* two other team members, before it can be merged with the current code base.

8 Definition of Done

This section shows all items that need to be checked before a backlog item, sprint or release can be considered done. These checks are mostly based on the quality assurance.

8.1 Backlog Items

A backlog item is done when:

- Code properly satisfies the user story defined in the Backlog item.
- Code has been tested appropriately. In the case of Javascript, appropriate e2e and unit tests should be written in QUnit. The code coverage, generated by Blanket.js should be at least 80% unless the code isn't testable. In the case of GOAL, appropriate e2e tests should be written in a test2g file.
- Code should contain clear comments, clear enough so other team members reading it will quickly understand what the code does.
- Code may not contain any warnings caused by any software engineering tool disclosed in the group agreements.

- Code has been reviewed by at least two other team members. Reviewers should keep all the above items in mind when reviewing and request changes to the code if there is a flaw.
- Large non-code tasks such as external documentation should be reviewed by all team members.
- Small non-controversial tasks with an estimated time smaller than a hour don't have to be reviewed. Unless they require a pull request.

8.2 Sprint

A sprint is done when:

- When the deadline of that sprint has been reached. All tasks and user stories from the finished sprint will be added to the next sprint during the sprint planning meeting.
- A new release has been created.

8.3 Releases

A release is done when:

- All features that should be present in that release have been tested to be working.
- All relevant documentation has been added to the release.

References

- [1] Apache Software Foundation, “Blockly,” <https://developers.google.com/blockly/>, [Software].
- [2] Fraunhofer IAIS, “open-roberta,” <https://lab.open-roberta.org/>, [Software].
- [3] P. in het PO, “Programmeren in het po,” kn.nu/leerlijnprogrammeren, 2016, accessed: 2017-04-26.



SIG Review

This section shows the feedback the team got from SIG. The feedback is given in Dutch, this is not translated in order to keep the original feedback intact.

F.1 Review 1

De code van het systeem scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere scores voor Unit Size en Unit Complexity.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt.

In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen. BlockParser.java scoort op beide gebieden vrij laag. Dit is natuurlijk ook vrij complexe, low-level logica, waardoor je altijd enige mate van complexiteit zal hebben. Het valt echter op dat jullie zelf "from scratch" een parser hebben geschreven, terwijl te hier ook een library voor zou kunnen gebruiken. Als je besluit het zelf te doen is het zaak om de complexiteit onder controle te houden. In de methode parseArgument doen jullie bijvoorbeeld een lookup, maar omdat dit nu als if/else boom is gementeed ontstaat onnodige complexiteit.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code dus gemiddeld, hopelijk lukt het om dit niveau nog wat te laten stijgen tijdens de rest van de ontwikkelfase.

F.2 Review 2

In de tweede upload zien we dat zowel de omvang van het systeem als de score voor onderhoudbaarheid is gestegen. Bij Unit Size en Unit Complexity zien we een sterke verbetering ten opzichte van de eerste upload.

Helaas is deze aandacht voor onderhoudbaarheid ten koste gegaan van het schrijven van nieuwe testcode. Juist bij refactoring is het belangrijk om tests te hebben, om te voorkomen dat er dingen stuk gaan tijdens het aanpassen.

Uit deze observaties kunnen we concluderen dat een deel van de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.



UserTest Results

This section shows the results from the three user tests that were done during this project.

G.1 UserTest 1

The first user test was in Dordrecht at OBS de Griffioen on Tuesday 30 may 2017. We were able to test students in group 6, group 3/4 and group 3.

G.1.1 Results group 6

Group 6 had 33 students. In groups of 3, they were able to try the exercise 'Voor, achter, links en rechts' and 'Applaus'. When the exercise went well, they were allowed to let the robot say something. After the test, the students were asked if they liked it and if they thought it was hard to do the exercise.

The students all really liked doing the test. The biggest part of the students (25/33) found it the most fun to let the robot say something. The rest of the students (8/33) found it the most fun to let the robot move.

Some (5/33) students did think it was hard to do, but after the explanation they found it easy. The rest of the students (28/33) thought that it was easy to program the robot.

It was noticed, that the students were so eager to let the robot move, that they didn't read the exercise. This caused the exercise to sometimes fail. Next time there should be more focus on the exercise, so it will be read properly. It was also noticed that the students really liked the sandbox exercise, because the students were able to see that the robot can do a lot of movements. Finally a thing that was noticeable was, that not for every student it was clear that the text on the speech block was click-able and adjustable.

G.1.2 Results group 3/4

Group 3/4 had 25 students and in groups of 3 they were allowed to do the exercise 'Hoofd, schouders, knie en teen'. If this succeeded, they were allowed to let the robot say their name. After the test, the students were asked if they liked it and if they thought it was hard to do the exercise.

All the students liked the exercise, also they all liked it the most when the robot said their name. Even when the robot didn't pronounce it right. Some students (5/25) did think it was a little hard to connect the blocks, but the rest of the students (20/25) thought it was easy.

It was noticed that some students wanted to give commands to the robot by voice, when they saw the robot. Also some students had a hard time handling the mouse. This caused the exercise to be a little harder. Because sometimes the blocks were dragged over the whole screen, before it was connected. But this didn't cause the children to think that the exercise was harder. Finally it was noticeable that the students were very happy when the robot did the exercise right.

G.1.3 Results group 3

Group 3 had 25 students and in groups of 3 they were allowed to do the exercise 'hoofd, schouders, knie en teen'. When this succeeded, they were allowed to let the robot say their name. After the test, the students were asked if they liked it and if they thought it was hard to do the exercise.

The students really liked letting the robot say their name. Also each student (25/25) thought it was easy to program the robot.

Also in this group it was noticeable that some students had a hard time handling the mouse. Which caused the blocks to go all over the screen.

G.1.4 Overall noticeable things

There were some things, that were noticeable in every group. A lot of students asked if the robot could do the dab, which was not possible yet. Also the students asked why the robot was scratching his head every time he was in the stay alive module. Apparently this was a weird movement to do when

waiting. The students were very excited to program the robot, even before it was explained what they had to do.

When teachers were asked for something they would like to see, they wanted to know if the robot could speak multiple languages to make the language lessons more interesting.

G.1.5 Conclusion

In conclusion, a dab action should be interpreted. Also it should be possible for students in group 3/4 to use the speech block. They are able to type some words, and really love it when the robot pronounces their name. The movement where the robot scratches his head during the stay alive module should be removed. This movement is weird to have in an idle state and causes students to get distracted.

During the next user test, the focus should be more on the exercise, to make sure the students read it and the exercise is executed properly.

G.2 UserTest 2

The second user test was in Woudsend at the Meester van der Brugschool on Monday 12 June 2017. We were able to test students in group 5/6, group 7/8. Also the teachers asked if there could be a demonstration in group 0, 1/2 and 3/4. So in these classes, a small demonstration is given.

G.2.1 Results group 5/6

Group 5/6 had 24 students and in a group of 3 they were allowed to try the exercise 'Handje klap'. Afterwards they were able to do the sandbox exercise and create an own program. Afterwards it was asked if the students liked the exercise and if they thought it was hard to do.

The students all really liked it and the biggest part of the group (18/24) liked it the most to create an own program for the robot to execute. The rest of the students (6/24) liked everything the same.

There was an own group of students (3/24) who thought it was hard to think of a new program, but the exercise 'handje klap' was easy. There were two groups of students (6/24) who didn't think it was hard, once it was explained what to do. The rest of the students (3/24) didn't think it was hard at all.

It was noticed that the students really liked the dab movement. Each group of students used this block in the sandbox exercise and they were enthusiastic when the robot executed this block. Also the sit down action and the stand up action was interesting. The students thought it was awesome that the robot could sit down and stand up all by itself. Finally it was noticed that a lot of students, when they learned the basics, were trying out a lot in the sandbox exercise. For example, they started using sensors and they were looking through the whole menu to see what the robot could do.

G.2.2 Results group 7/8

Group 7/8 had 20 students and in a group of 3 they were allowed to try the exercise 'Sporten'. Afterwards they were able to do the sandbox exercise and create an own program. Afterwards it was asked if the students liked the exercise and if they thought it was hard to do.

The students again all liked it really much. Some students (6/20) liked creating a new program the most. One group (3/20) liked it the most to see what the robot did when he got the program. The rest of the students (11/20) liked everything the same.

A big part of the group (12/20) thought it was easy, but it should be explained first. The rest (8/20) thought it was easy.

It was noticeable that students from group 7/8 were faster with trying new things. After a short explanation and one exercise, they went to the sandbox and were looking what was possible. They didn't use sensors in the exercise, but they did use this in the sandbox exercise. They did ask what 'sensoren' meant before they started using this, so maybe another name should be chosen for sensors. After explaining the sensors, the children were able to use it.

G.2.3 overall noticeable things

The school asked if it was possible to give a demonstration for group 0 1/2 and 3/4. The robot was standing in the middle of the group with the students around it. A small program was written and the robot started executing. All the students were happy to see the robot move. In group 3/4 we also showed how the program was created, but due to lack of time the students were not able to program the robot themselves.

The dab movement is a movement that should stay, every student loves it. Also sitting down and standing up is appreciated. But after the sit down action, we should make sure that the robot stands up. Otherwise it falls down. Still the exercise is not read properly. Maybe this is caused due to the lack of information, so the students listen to the explanation and then forget to read. It should be tested during the next user test. This time, the students did look at the block explanation. Creating an own exercise was really appreciated, so the sandbox exercise should stay. Finally the movements in the stay alive module were now more naturally and the students didn't get distracted anymore.

G.2.4 Conclusion

During next user test, we need to let the students read the exercise. The next user test is at the previous school, so the students know the basics. Maybe now they read the exercise. The name 'sensoren' is not clear, so another name should be chosen. Maybe 'aanraken' is more clear for the students.

G.3 UserTest 3

The third user test was in Dordrecht at OBS de Griffioen on Thursday 15 June 2017. We were able to test students in group 7, and group 3.

G.3.1 Results group 7

Group 7 had 46 students. In groups of 4, they were able to try the exercise 'Sporten'. When the exercise went well, they were allowed to create an own program in the sandbox exercise for group 7/8. After the test, the students were asked if they liked it and if the though it was hard to do the exercise. Because we wanted the students to read the exercise, we tried to not explain everything and let them read. This caused the students to read the exercise.

All students liked programming the robot. Most students (32/46) found it the most fun when the robot did the dab movement. a few students (9/46) found it the most fun to create a program on their own. The rest of the students (5/46) liked everything the same. There were some students (12/46) Found it easy, once they found out where to look. The rest of the students (24/46) thought the exercise was easy. There were a few students (16/46) who really read the exercise precise and used this to find out how to program. Some

other students (23/46) did read the exercise, but were so enthusiastic, that they didn't read it well and made a different solution. The rest (7/46) didn't read the exercise at all and just started trying something to let the robot do.

It was noticed that the students really liked the dab action. Every group used this block during the sandbox exercise. Also the sit down action was used a lot. The children were often so enthusiastic, that they forgot to read the exercise properly. But when they read it properly, they were able to execute it very well.

G.3.2 Results group 3

Group 3 had 23 students. In groups of 4, they were able to try the exercise 'Loop een rondje'. When the exercise went well, they were allowed to create an own program in the sandbox exercise for group 3/4. After the test, the students were asked if they liked it and if they thought it was hard to do the exercise. Because this group already used the robot before, we tried to let them read the exercise instead of explaining the exercise.

All students really liked to program the robot once again. Most students (16/23) liked the dab behavior the most. Last time they asked if the robot was able to do this and now they were very happy to see that the robot did the dab. the rest of the students (7/23) liked everything the same. Every student (23/23) thought it was easy to program the robot.

It was noticed that when we said that the students needed to read the exercise, they had a hard time reading it. This was because the letters on the screen were too small for students from group 3. The names and the layout of the menu were changed since the last time the students used the robots, but they were able to find every block. Finally it was noticed that using the mouse sometimes still was hard for the students, but this didn't make the exercise harder according to the students.

G.3.3 Overall noticeable things

There were some things, that were noticeable in every group. The dab action is really appreciated. Letting the robot listen to voice is also fun to do, but the robot is not able to recognize every voice. So the children had to work together to let the robot listen. The text in the exercise explanation is too

small for students from group 3.

Also some bugs were found during testing. Sometimes the goal agent terminates randomly. This needs to be fixed. When a free walk block is used and the robot should listen to a word, but no word is given. The robot will walk forever. So we need to catch the edge case where there is a block inside, but no word to listen to.

G.3.4 Conclusion







In conclusion, We need to make sure that the text for the exercises will become bigger, so it is readable for everyone. The bugs that were found should be fixed.









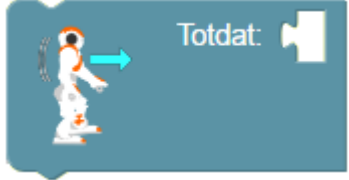



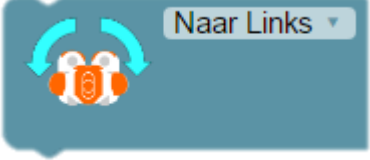

Block overview







This section contains an overview of all blocks in the toolbox. Currently, blocks are split up into five categories: Basic Action, Sensor Blocks, Functional Blocks, Logical blocks and Timers.




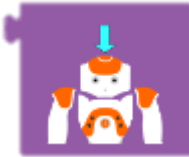
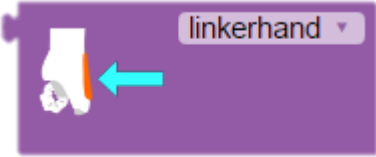
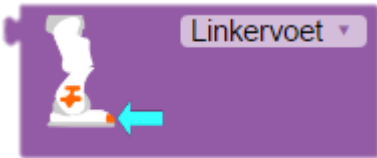
Apart from these categories, it's important to differentiate between vertical blocks and horizontal blocks. Vertical blocks can be placed below and above each other, they can be seen as instruction blocks. The two categories corresponding to vertical blocks are basic action blocks and functional blocks, where functional blocks are usually more complicated and contain more blocks in itself. Horizontal blocks don't execute an action, but rather return a value. All sensor blocks belong to this category. For example, a headTouched block returns a Boolean. Logical blocks also return booleans. Also timers will return a boolean.



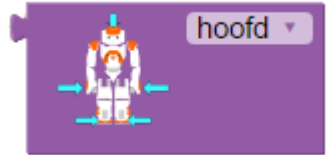
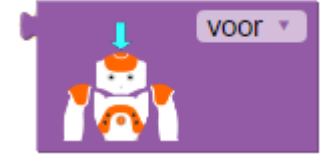


Basic Action	Block
<p>Makes NAO wave its right arm.</p>	
<p>Makes NAO touch its head with both hands.</p>	
<p>Makes NAO touch its shoulders.</p>	
<p>Makes NAO touch its knees.</p>	
<p>Makes NAO touch its toes.</p>	
<p>Makes NAO raise the given arm.</p>	


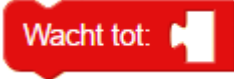


<p>Makes NAO raise its left arm.</p>	
<p>Makes NAO raise its right arm.</p>	
<p>Makes NAO put both arms forward.</p>	
<p>Makes NAO put both arms backwards.</p>	
<p>Makes NAO point the left arm to its side.</p>	
<p>Makes NAO point the right arm to its side.</p>	





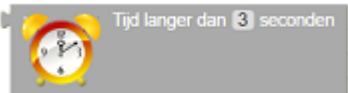
<p>Makes NAO walk forward until the condition is true.</p>	
<p>Makes NAO walk forward one step.</p>	
<p>Makes NAO rotate left (90 degrees).</p>	
<p>Makes NAO rotate right (90 degrees).</p>	
<p>Makes NAO rotate in the given direction, (90, -90 or 180 degrees respectively).</p>	
<p>Makes NAO clap its hands once.</p>	

<p>Makes NAO open the given Hand.</p>	
<p>Makes NAO close the given Hand.</p>	
<p>Makes NAO say the given Text.</p>	
<p>Makes NAO rotate its hand and move the hand back and forth as to sprinkle some chocolate.</p>	
<p>Makes NAO bow.</p>	
<p>Makes NAO stand up (its default position).</p>	

Makes NAO duck (by bending its knees).	
Makes NAO sit down.	
Makes NAO dab (moving his arms and head) and the robot plays some sound	
Sensors	Blocks
Evaluates to true if the head of the NAO has been touched.	
Evaluates to true if the given hand has been touched.	
Evaluates to true if the bumper of the given foot has been pressed.	

<p>Evaluates to true if the NAO has heard someone speak the given text.</p>	
<p>Evaluates to true if the NAO has any of its sensors touched.</p>	
<p>Evaluates to true if the NAO has its sensor touched</p>	
<p>Evaluates to true if the NAO has one of its head sensor touched</p>	
<p>Functional</p>	<p>Blocks</p>
<p>Executes all the blocks inside the block exactly the given amount of times.</p>	
<p>If the condition evaluates to true, then all statement actions are executed.</p>	

<p>If the condition evaluates to true, then all statement actions are executed.</p>	 <p>Wachten op invoer Als</p> <p>Dan</p> <p>Als</p> <p>Dan</p> <p>Als</p> <p>Dan</p>
<p>Makes the NAO idle until the given condition evaluates to true.</p>	 <p>Wacht tot:</p>
<p>Keeps executing all blocks inside this block as long as the condition evaluates to true.</p>	 <p>Zolang:</p>
<p>Keeps executing all blocks inside this block until the condition evaluates to true.</p>	 <p>Totdat:</p>

Logical	Blocks
Evaluates to true if the condition evaluates to false.	
Evaluates to true if both the left part as well as the right part evaluate to true.	
Evaluates to true if both the left part or the right part evaluate to true.	
Timers	Blocks
Evaluates to true if the timer variable is less than the given amount of seconds.	
Evaluates to true if the timer variable is greater than the given amount of seconds.	
Special	Blocks
The start block indicates where the block program starts. It contains the order of execution.	