14 October 2023

# Wearable technology software toolkit

## THE DEVELOPMENT OF A VISUAL PROGRAMMING ENVIRONMENT FOR ARDUINO

BY: KEVIN SMID

SUPERVISOR: EDWIN DERTIEN

CRITICAL OBSERVER: ANGELIKA MADER

# Abstract

Not everyone can participate in DIY-focused workshops. One of the main reasons for this is a lack of programming experience, which makes the participant unable to work independently. As there is no time to guide these participants to all the steps required to create working code, this research proposes a more visual approach to programming. For this purpose, a tool has been created that can aid users to program their own prototypes.

The developed tool uses a flow-based programming language which is in turn converted to Arduino code before being uploaded to the device. To achieve this the tool implements simple Arduino functions and links them to matching blocks, these blocks can then be connected in a sort of a flow chart to create code.

A user test was held with ten participants in which they were asked to create increasingly difficult code for the components supplied. The results show that this lowered the threshold for beginners while still allowing for a lot of creativity/flexibility in what can be created. For advanced programmers however it is inconclusive whether or not this tool decreases the possible complexity of what can be created. The results are promising, but further development of the tool and further testing is required to conclude whether a non-limiting tool can be created.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Thanks to advances in hardware and software, low-power devices such as IOT sensors or BLE devices are becoming increasingly available to a wider public. In addition, these advancements enable countless applications of technology. One area in which these affordable and smaller technologies can be used is wearables. Wearable technology or wearables refer to technology that is designed to be worn throughout the day, such as watches, wristbands, glasses, and clothing. For example, wearables in clothing provide many different possibilities and functions, such as posture correction, a turn signal for cyclists or integrated party lights. There is a lot of potential to improve wearables through technology, but in the current products extraordinarily little wearable technology is used. Smart wearables have many other applications across multiple industries, such as in the medical industry, where wearables can reduce costs and improve the quality of healthcare by remotely monitoring body functions [1].

## 1.1 Problem statement

To familiarize people with this technology, a wearable technology toolkit has been created, which is used during the workshops. This toolkit allows users to prototype and design wearable technology without requiring extensive knowledge of electronics. Currently the code is written in Arduino, but users who do not have a coding background often struggle with writing Arduino code. To make the prototypes within the timeframe of the workshop, you need a technical coding background, allowing fewer designers to participate in this workshop.

Helping the user in programming his or her prototype idea, would aid in the ideation process. Creating the tool(s) needed to enable designers to make wearable technology is the focus of this project. This leads to the following research question: “How do we make programming accessible to designers in a short (half-day) workshop?”

## 1.2 Research questions

As discussed in the problem statement the main research question is defined as:

“How do we make programming accessible to designers in a short (half-day) workshop?”

In order to answer this question, this question will need to be split into smaller parts, this resulted in the following sub-questions:

- How can a different approach in programming help shortening the time it takes to code, and reduce the amount of experience needed?
- Does a programming tool still allow the user to be creative and experiment with all the options the toolkit provides?
- Is using a visual programming tool a hinder for users that are comfortable with programming?

## 1.3 Report outline

In the first chapter the goal and research questions of this research are explained. After that in the second chapter a description of the creative technology design process and how it is used during this project can be found. Chapter three contains the state of the art detailing the current contents of the workshop and a description of how other workshops deal with lacking programming skills. In chapter four visual programming is explained along with how it will be used in this project. Chapter five defines the possibilities for the programming tool and explains choices made to get requirements. These requirements are then used in chapter six to create a design for the programming tool. Chapter seven explains how the programming tool was created and a in depth look at its features. After that chapter eight contains the materials and setup used for the user tests followed by the results of this user test. Chapter nine will be the conclusion of this research, while chapter ten is the discussion related to this project.

# 2 Method

This chapter discusses the steps that were taken to get to the end result of this graduation project. First there is a description of how the Creative Technology design process is structured, which will be used to create a prototype to answer the research questions. After that there is an overview of what was done in each of these steps during this project.

## 2.1 Creative Technology design process

The Creative Technology design process consists of the following four phases: ideation, specification, realisation, and reflection. Each phase has its own set of intermediate results as seen on the left side of Figure 1. Within each phase a cyclic workflow is used, which allows the designer to iterate over the nested problems that are often found in designing a product.

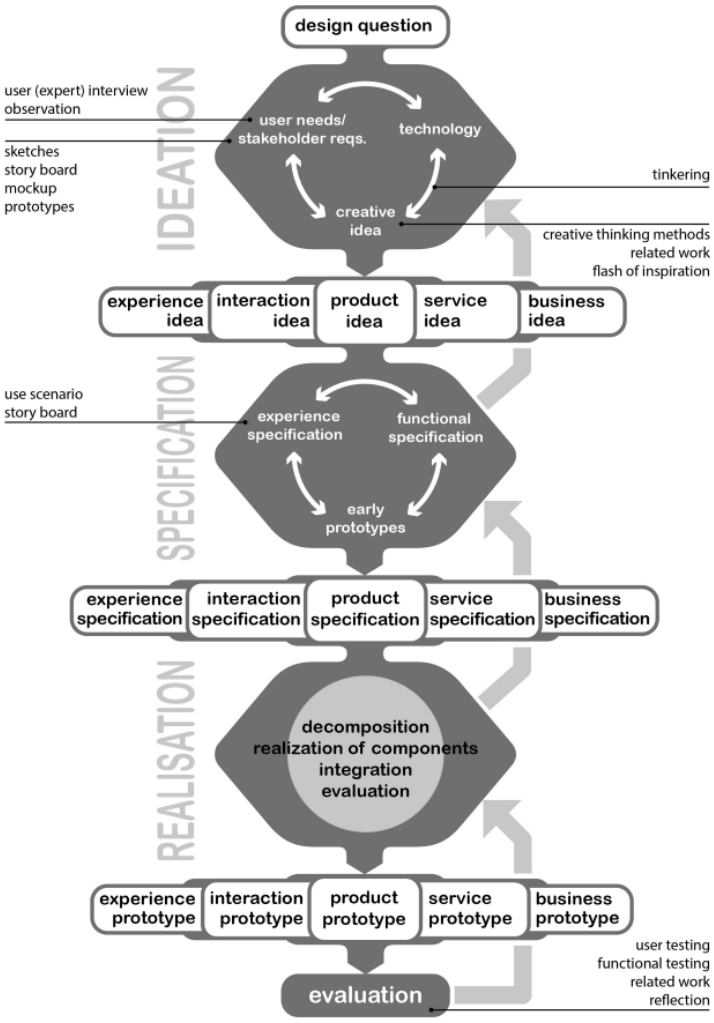



*Figure 1 Create Design Process Phases [2]*

### 2.1.1 Ideation

During this phase the focus is on defining the problem and gaining requirements that would solve the problem. For this project this started with an interview to look at how the workshop was structured. After this background research was done into related works for inspiration. One of the creative ideas was to use the whiteboard flow chart as code. After that this idea was used to look more closely at one interesting possibility namely visual programming, when looking at this possibility it was selected as the solution that is going to be used. To see if this was doable a brainstorming session was held with another student to generate as many ideas as possible that would be possible using the toolkit.

### 2.1.2 Specification

The results from the brainstorm session were then used to determine the functions that need to be included in the prototype. After this several personas are made to better understand the programs the target audience uses often. In addition, some small prototypes were made to investigate how the code generation could work.

### 2.1.3 Realisation

In this phase the prototype that is described in the specification phase is created. To do this first the requirements were converted to descriptions of the classes. These classes were then developed without making all the specific subclasses for the different blocks.

### 2.1.4 Evaluation

During the evaluation phase the prototype is tested, to do this multiple user tests are performed. The first user test was a pilot test to test for any mistakes in the assignment and bugs in the code that would prevent other test users from completing the test. The information gathered from the user tests is then used to verify if the prototype is a solution to enable all the participants to generate their own code without being too limited in the possibilities provide by the toolkit. Additionally, the areas that still need further improvement in the future are defined in this phase.

# 3 State of the art

As previously stated, a toolkit was created for the wearable technology workshop to familiarize designers with the possibilities of wearable technology. In order to achieve the intended improvements in accessibility for designers, as indicated in the research question, it is important to first describe the current situation.

This chapter starts with an interview with an important stakeholder who provides a brief description of the current workshop and its origins. Subsequently, a detailed description of the wearable technology toolkit is given. Since we want to make programming more accessible to a specific target group in a short workshop, a further description is given of the participants in the workshop. How the current workshop and other workshops and toolkits in the same field deal with the missing programming skills of the participants is another important element of this chapter. During the expert interview, three important criteria for developing a toolkit were identified. This chapter about the state of the art concludes with a number of requirements for both the workshop and the toolkit.

## 3.1 The current wearable technology workshop and its origins

Edwin Dertien is a lecturer and researcher at the Robotics and Mechatronics department at the University of Twente. In an interview, he gave the following information about the origin of the wearable technology workshop. At first, a short introduction to programming was given during a one-day workshop on interactive inflatables which allowed all group members to understand what other group members had programmed. This workshop developed into a broader wearables themed workshop, not just about inflatables. Subsequently, Angelika Mader (lecturer Creative Technology, University of Twente) and Edwin Dertien were asked to give a half-day wearable technology workshop in a couple of other libraries.

Edwin indicated that in addition to the technical parts of the toolkit, instruction cards were developed (see also 2.2). These instruction cards were intended to give structure to the workshop. For example, three cards were drawn, and the participants were challenged to discover what they could make with the items on the cards. While brainstorming, all kinds of ideas arose. Discovering possibilities by using a random set of components was an important learning objective of this workshop. This ensured that the participant became acquainted with the possibilities of the toolkit before moving on to the reverse process, namely from a desired outcome to the required components.

The core activity of the workshop is designing and tinkering, to figure out all the possibilities that wearables have to offer. For structuring coding process during the workshop sense-think-act is used as a guideline. So first you focus on measuring the sensor that you want to use, then you try to translate those values into something sensible, and finally you output those variables to your actuator(s). To do this in a sensible way you first make a flowchart of the behaviour that you want your program to have before converting that into code.

In the next section, we examined the current toolkit in detail.

## 3.2 Description of the current toolkit

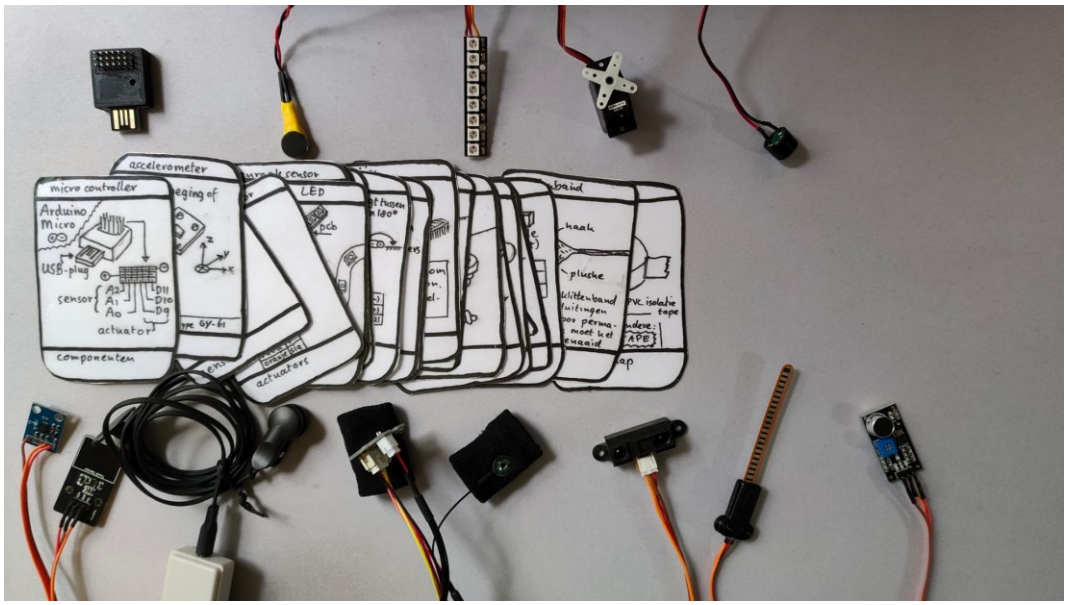



*Figure 2 An overview of the toolkit*

Developing and evaluating a toolkit requires a thorough understanding of its content and capabilities. Therefore, this section provides more detailed information about the current toolkit. As Figure 2 shows, the toolkit consists of a number of items, namely instruction cards, a controller, sensors, actuators, and starters.

### *Instruction cards*

The instruction cards consist of a small deck of 21 cards. Each card provides a brief overview of a relevant item. The cards are divided into five groups: components, tools, materials, sensors, and actuators. During the workshop, these cards can be used as quick cheat sheets. Another important function is its use as a way to ideate; if a user picks a few random cards and combines them, a new product quickly emerges. For example, the micro controller card as seen in Figure 3 shows essential information such as the pinout and the possible connections, allowing the card to be used as a quick reference during prototyping.

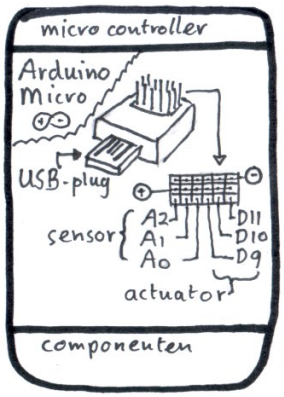



*Figure 3 Micro controller card source: wiki.edwindertien.nl [3].*

*Controller*

When using a toolkit to create wearable technology, a controller is always needed to plug the other components into. Therefore, the controller has three analog inputs and three digital outputs to which the sensors and actuators can be connected, respectively. The controller, the Beetle, is based on Arduino micro which means the controller is based on the ATmega32u4 chip. The controller can be plugged into a computer using the USB-plug. When connected, the user can program the controller using Arduino Integrated Development Environment (Arduino IDE). In this environment Arduino code can be created (C++ with additional methods and functions) that determines the behaviour of the controller after uploading. Since Arduino is a widely used programming language, the user can easily find information about it. However, coding can be challenging for a user who has no prior programming experience. For this reason, the controller uses a premade code that works for all starter cards.

*The sensors*

The toolkit contains six sensors that can be connected to the controller with color-coded cables. This modular design allows the user to instantly connect a different sensor. The foregoing in combination with the premade code enables quick prototyping; the user only needs to connect the sensor to the correct port. The included sensors are a touch sensor, an accelerometer, a flex sensor, a sound sensor (microphone), a galvanic skin response (GSR) sensor, and a heart rate sensor. By using these sensors, the prototype can easily react to the environment.

*The actuators*

There are four kinds of actuators: a buzzer, a servo motor, a vibration motor, and a LED strip. These allow for a large variety of output signals. The actuators can be connected to the following digital ports: D9, D10 and D11, which means that no more than three actuators can be used at the same time.

*Starters*

Included in the kit are some “starters”. These starters are meant to start building in a simple yet inspiring way [3]. Figure 4 shows an example of such a starter, the turn signal card. This card describes the necessary components to create a turn signal using a LED strip on your back and touch sensors in your sleeve. When a cyclist presses the button, a turn signal will blink indicating the direction the cyclist wants to go.


*Figure 4 The turn signal starter card source: wiki.edwindertien.nl [3].*

In order to develop a tool that fits well with potential users, more insight is provided into the target group, which consists of both students and designers. Since the tool should be especially suitable for people who have less programming skills, the designers, we then provide a user scenario for this.

## 3.3 Target audience

The wearable technology workshop is given to designers and students to boost interest in wearable technology and open their eyes to the possibilities. What this means for the workshop itself is discussed here. The participants do not need any special skills to start with the workshop. As it is a technology-based workshop, a technological interest can nevertheless be expected. During the workshop, the audience should get an understanding of the possibilities and impossibilities of technology in their future designs. As Durling states, in fashion design being different can be a goal on its own, not following a path that has been followed before is important in fashion [4]. This aiming at being different is often seen in the designers' approach. They intuitively feel what is right and aim for it. This often results in taking another path than expected, creating something new and interesting in the process. The following table shows a brief overview of what the target audience could contain.

| | Students | Designers |
|---|---|---|
| Age | 18-26 [5] | 18-65 (working age) |
| Possible background knowledge/experience | • Programming experience<br>• Uses apps frequently<br>• Experience with Simulink | • Designing of clothing<br>• Uses apps frequently<br>• Experience with graphical design programs |
| Motivation | Investigate whether programming is an interesting field of study.<br>Learn about the creative process involved in producing a new product, with hands on experience. | Discover the possibilities of wearable technology and get inspiration to use it in their designs. How easy is it to get started with wearable technology, can I think of wearable technology that adds value to my design? |
| Related personal interests | • Internet of Things (IOT)<br>• Smart watches<br>• Fashion<br>• Health monitoring | • Fashion<br>• Technology |

*Table 1 average end-user specification*

### 3.3.1 User scenario

James, 19, a first-year industrial design student, is interested in discovering more about wearables. He recently acquired a smart watch and now wants to know more about wearable technology and its possibilities. He joins the workshop expecting to experiment with and discuss the future of wearables. However, during his studies he has never programmed before, so as the workshop progresses, he struggles with the coding of his prototype. He is too proud to constantly ask for help and wants to impress others.

He needs something that enables him to make a prototype independently or with minimal help from the people who assist in the workshop. He wants to create a heart rate monitor that shows a visual representation of his heart rate using a circle of LEDs attached to his t-shirt. He wants to be able to make this monitor without having to learn Arduino syntax that is currently used in the workshop. He is used to programs with an intuitive user interface and wonders if such an interface for creating code exists.

Julie, 37, a sportswear designer, wants to keep up with the current trend of devices that track the user's performance. For that reason, she wants to learn more about how sensors and wearable technology work. With the help of the assistants during the workshop, she manages to make a nice prototype of shorts that measure how someone runs. However, when she wants to continue at home, she cannot get her programs to work. She does not have time to immerse herself into the world of programming.

She wants something that will allow her to continue experimenting with the sensors and create smart wearables at home, without spending time and effort learning to code on top of her busy job. At work, she often uses a program that quickly reveals what a design idea would look like by dragging patterns and letters onto digital previews of T-shirts without having to draw it from scratch.

## 3.4 Current workarounds for the missing programming skills

In this section, we explore how missing programming skills are addressed in the current wearable technology workshop. During the wearable technology workshop, the participants will receive sample code which they have to adapt to their needs in Arduino IDE. If the sample code provided matches what they intend to create, users can quickly prototype as they do not have to code themselves. Not having to code is both an advantage and a disadvantage, because the user does not tinker with the code or only to a limited extend, so that not all possibilities of the hardware are explored. Since a designer likes to make something new and creative, this is a rather large handicap of the toolkit. In order to make full use of the possibilities, it would be necessary to teach the designers to write Arduino code themselves. However, this is a very time-consuming process that cannot be accomplished in one session. Another possibility is that the coder extensively guides the designer in creating the necessary code for the behaviour intended by the designer.

## 3.5 Other ways of addressing the lack of programming skills

This section discusses workshops in the same field, focusing on how these workshops deal with a lack of programming skills during the workshop. Subsequently, we describe two toolkits that could be used for people with minor programming skills.

### 3.5.1 In other workshops

*Interactive inflatables studio workshop*

During this six-hour workshop, participants create an inflatable device that allows you to notice certain human behaviour that is otherwise difficult to observe, such as excitement or attraction [6]. This device gives a physical interpretation of certain behaviours or emotions by, for example, inflating or moving. To create such an inflatable device, the user is equipped with a simple kit of components and is assisted by instructors. Before programming the Arduino, the participants receive a short workshop on how to work with an Arduino. In addition to other skills, the learning objectives indicate that the user acquires new programming skills, so there definitely is a focus on educating the user in the field of programming. The setup of this workshop bears a lot of resemblance to the wearable technology workshop, so Arduino IDE could also be used during the current workshop. However, this is already being done and takes a lot of time. Time that could be better used for the creative process.

*Kobakant: Soft Interactive Technology 1*

During this adult-oriented workshop, the participants will explore what textile sensors can do and how users can interact with digital technology [7]. The user chooses an app and then analyses how to interact with this app, do you swipe or click? Once it is clear how the app works, the user starts to think about how to modify this interaction. To investigate this, many prototypes are created over the course of two days. An advantage of this workshop over other workshops is that it is a two-day workshop, so there is a lot of time to learn and tinker with code. During the workshop, the user makes his/her code in Arduino. To guide the user through the programming, links are provided to Arduino reference pages often used on previous prototypes. With this guidance and the allotted time, the user can program it from scratch.

### 3.5.2 In other toolkits

*MakerWear*

MakerWear is a toolkit that uses physically connecting hexagonal modules instead of code. One of these modules can be a sensor or an actuator, but more interestingly it can also be an action such as a threshold or a counter. When connecting these modules, the user creates ‘code’ directly in the physical world. While this approach does not require actual coding, it does present some drawbacks, including the following: “the sole reliance on a tangible, modular approach limits designs to available modules as opposed to completely open kits” [8]. This means that full physical programming limits the freedom of end users significantly, but since the toolkit is aimed at children in primary schools, this is justified.

*BYOR – Build Your Own Robot*

This kit consists of a printed circuit board and various sensors and actuators, all of which have a Jack connector. The board has six inputs and six outputs. When connecting a sensor to the first input, the same signal is applied to the first pin on the output side [9]. This results in direct feedback, plug in a sensor and actuator and you immediately have a working device.

When a micro:bit is inserted in the BYOR Easyboard it is possible to create your own code for the BYOR kit. The main advantage of this kit is that it is completely plug and play; the micro:bit adds a lot of functionality using a block-based visual programming environment. One downside of the kit is the nonstandard connector.

### 3.5.3 Applicability to the current workshop

During the background research, different ways were found to address a lack of programming skills. One way consisted of just getting started with Arduino by providing lots of examples or premade code (for example the Interactive Inflatables Studio Workshop). Other more interesting routes were to help the user with visual programming (BYOR) or not to use programming at all (MakerWear and BYOR). The sub research question “Does a programming tool still allow the user to be creative and experiment with all the options the toolkit provides?” states that the tool should not be a restriction to what the user can make. In the two cases where no code was used, the user was limited to either a direct translation of input to output or very limited processing of the signal was found. This means that the user cannot program everything he or she wants (MakerWear) or in the case of BYOR, that all current starters must be included in the kit. The Soft Interactive Technology allows the user to learn a lot about programming, but since this is a two-day workshop, there is also more room for this. For the wearable technology workshop simply telling the user where to find the information may not be enough, but proper help files or a wiki should be created.

## 3.6 Exploring possibilities with the current toolkit

### *What is possible?*

The toolkit contains a variety of sensors and actuators, but what can be created with this? To what extent do these creations depend on different programming principles? To investigate this further, a list of code examples has been made (see Appendix A1). This list includes a brief description of each program and uses keywords to describe the programming requirements. This list was created as follows. First the researcher sat down with another student to generate ideas of what they thought could be made using the hardware provided in the toolkit. When this was done each of the generated example uses got assigned code keywords based on what would be needed to create them.

### *Code keywords for the programs*

The keywords were counted and then ranked based on this count, this was done to get insight on what the most important features to implement are. Appendix A2 shows how often a keyword occurs. There are a few things worth noting. Some keywords occur very often and are used in almost every program. For instance, the “Read values” keyword appeared directly in at least eight different programs. The actual usage is even higher, because reading the value of a pin is used in every program that uses a sensor. It is also noteworthy that “Custom calculation” occurs five times, for these programs a simple if-statement or threshold would not be enough, for example the program that describes using an accelerometer to measure the power at which a soccer ball is kicked cannot be created using just simple condition.

| **Keyword** | **Count** |
|---|---|
| Read values | 8 |
| Threshold | 8 |
| Timer | 6 |
| Conditional statement | 6 |
| Alarm | 5 |
| Custom calculation | 5 |
| … | … |

*Table 2 Top six coding keywords*

## 3.7 Criteria for developing a toolkit

During the expert interview with Edwin Dertien, three important criteria for developing a toolkit were identified, namely wide walls, high ceiling, and low threshold. A brief explanation is given of what they entail.

*Wide walls*

This relates to how many different uses something has. When saying that a tool has wide walls this means that the tool can be used to create a wide variety of behaviours. This is important to not hinder the creativity of the end-user.

*High ceiling*

The ceiling in this case is the limit of what can be achieved with a tool, it is the opportunity for the user to make and learn advanced things that are beyond the average user's skill level.

*Low threshold*

This entails how easy it is to get started with using something. If the threshold to using is low, it is quick to pick up without requiring a lot of effort before the first results can be achieved.

## 3.8 Requirements for the workshop setting and toolkit

A number of things can be concluded of the different wearable technology workshops/toolkits and the target audience analysis that support the development of code for smart wearables:

- No technological skills should be expected when participating in the workshop. It is preferable that the tool can be used with a short explanation or no explanation at all. The tool should make it possible to start up quickly and to create a simple program within half an hour.
- Easy-to-read documentation for the user, this allows the user to be more independent, especially after learning the basics.
- The tool should not hinder creativity, it is the task of a designer to produce new innovative uses. Limiting the freedom of the designer in this process is not desirable for the workshop.
- Technological details do not need to be visible; the designer sees technology as a way to achieve the goal.
- All sensors and actuators should be supported, as the toolkit expands new components should be easy to implement.
- All functions from the examples should be present, prioritizing the code constructs that occurred in most projects. In this way, even if a newly developed tool is not yet finished, it can still be used and tested.

From these requirements it can be inferred that the tool must be very versatile, in order to be able to use it for many applications. In addition, the tool must be easily adaptable by the user, because it is not feasible to make all code for every possible prototype in advance, as it uses custom calculations and other “fancy” programming features. And last but not least, the tool must not hinder the user’s creativity. The next chapter explores an alternative to text-based programming.

# 4 Visual programming

## 4.1 Visual programming or regular programming

From the previous chapter, it became clear that not programming at all is not a valid option for the smart wearable toolkit as that would hinder the wide walls the toolkits hardware has to offer. This is why visual programming was chosen, as when implemented properly the designer should not be hindered in their creativity. To aid in programming, many alternatives have been created over the years of which programming visually is the most exciting. In theory, learning a programming language is replaced by using intuitive visual representations of code. To investigate whether visual representation of code can also be used in the smart wearable toolkit, a literature study on visual programming was performed, the results of which are presented in this chapter.

### 4.1.1 The reduction of required skills by visual programming

Scratch and other block-based visual programming languages are designed to be a first language learned, meaning they are created for novice programmers [10]. Block-based visual programming languages use several ways to facilitate code creation, such as drag and drop blocks; using shapes to show what kind of block is required; c-shaped control structures indicating that the body is missing and many more [10] [11]. These pieces that only fit together in a correct way prevent syntax errors from being made. However, this does not guarantee that the program will behave as intended when the pieces fit. Weintrop [11] examined student perceptions and discovered that a majority found block-based programming tools easier to use than text-based programming. Booth and Stumpf found that end users using a block-based visual programming environment for Arduino were more likely to complete the tasks given to them than those using a traditional programming environment. In addition, the end users with whom they performed the tests reported that the task was less mentally demanding and less frustrating [12]. One of the participants in Booth and Stumpf's research stated that it was difficult to remember which special characters to use whilst coding in the textual programming environment [12]. This clearly shows that the syntax used in textual programming, which is so familiar and easy to the advanced programmer, can be quite daunting at first for a new user. In addition to the block-based approach from scratch and many other visual programming languages, there is also another way of visual programming, namely dataflow-based visual programming (DFP). As explained by Sousa, DFP is based on nodes/blocks with lines connecting them [13]. An advantage of DFP is that it enables extremely fast prototyping of code and implementation of certain systems, allowing DFP to be used by experienced programmers as well as less technical users [13]. Visual programming languages require barely any programming knowledge to get started with. The building blocks are quite literally provided for the end user, which results in less learning required for the user to get started [10]. In summary, visual programming reduces the required knowledge of a specific programming language and prevents the user from making syntactic errors. This allows the novice user to quickly get started with programming and to create a program with relative ease straight away.

### 4.1.2 The limits of visual programming

Putting the code in premade blocks might seem very restrictive. However, this is not the case in the popular block-based languages such as Scratch, Snap and Blockly. As explained by Weintrop [11] abstract syntax trees are used, meaning that the program's primitives are represented visually and once all primitives are implemented, no functionality is lost. Technically, there is no limit to what can be created in these. However, as Schaefer clearly states, visual programming is very convenient for smaller programs, but when you increase the scale of the program, it often becomes very unclear [14]. Resnick et al. describe C-shaped building blocks in which the other code is contained [10]. When nested, the horizontal screen space used increases rapidly, causing the code to no longer fit into the space of the window [14]. The downside of block-based visual programming can create a struggle for the user to find what he/she wanted to look at, making you really search for what you want to change and making it more likely that you lose oversight over the code as a whole. (DFP does not use blocks that are inside other blocks, so larger programs might seem like a great plan here. However, if a user creates many nodes that are interconnected, the screen can quickly become filled with the nodes and the edges connecting the nodes. This can be difficult to interpret, as it is quite hard to find all relationships between all nodes [13]. According to Sousa a possible solution to this cluttering could be the ability to group nodes effectively creating your own bigger node [13]. The main problem for visual programming environments is not what an end user could technically create, it is keeping track of what has been created and fitting it onto the screen.

### 4.1.3 Usefulness for wearable technology toolkit

The previous sections discussed some advantages and disadvantages of visual programming. This section discusses its usefulness for the wearable technology workshop. During the wearable technology workshop, it is important that a novice coder can quickly create code. Visual programming can be a great tool for this, as it is designed to reduce prior knowledge to create code. The main disadvantage of visual programming is that it quickly becomes cluttered when creating large programs. This would not be a problem for the wearable technology workshop, as the amount of code created for one wearable is not that massive. Therefore, it can be concluded that visual programming is a possible solution to the programming problem in the workshop.

## 4.2 Existing visual programming tools

Since there are already many visual programming tools, these tools have been thoroughly researched for user-friendliness. A brief overview of the results can be found in this section.

### 4.2.1 Search strategy

To investigate the most searched/most popular tools, a clean browser session was used in combination with DuckDuckGo. This ensured that the previous browsing history did not influence the search. The keywords searched were “visual programming”. Subsequently, the results were screened on relevance, including only those results that were made for Arduino or other microcontrollers.

### 4.2.2 A brief overview of the most popular tools

*Scratch for Arduino*

Scratch for Arduino or S4A is a visual programming environment based on Scratch [15]. Just like Scratch, S4A has draggable blocks that are located in a menu on the left. These blocks only fit together in certain ways, as shown in Figure 5.

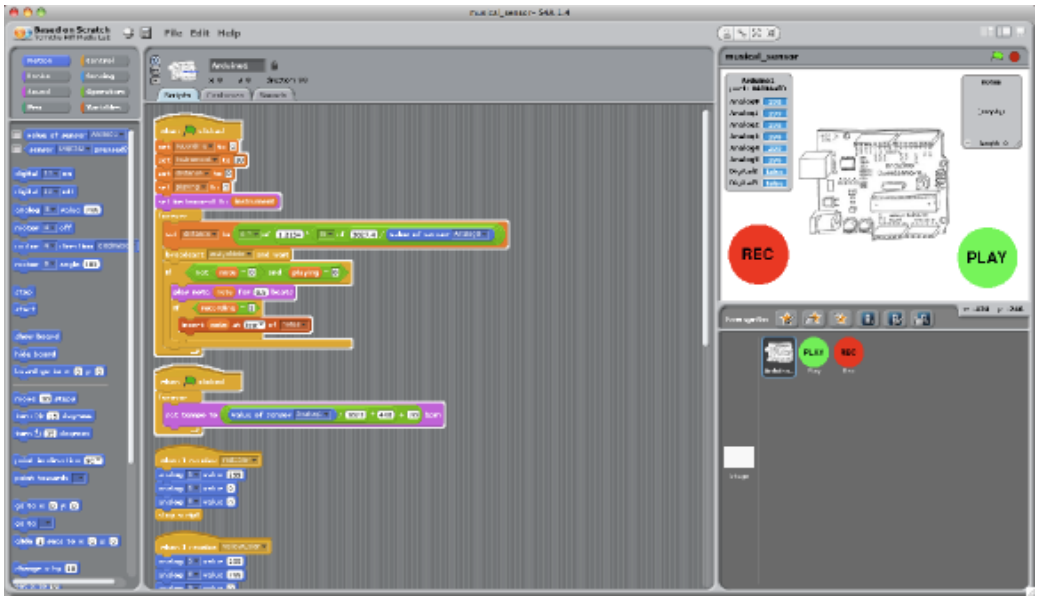



*Figure 5 A Scratch for Arduino program [15]*

The brightly coloured blocks represent basic structures and datatypes, which can be used to create programs that can be uploaded to the Arduino. Scratch was targeted to teach children how to code [10] and the same holds true for S4A. It is not created to be the fasted way to prototype an Arduino project. For the wearable technology toolkit workshops, software aimed at rapid prototyping for wearable technology is better suited. In addition, since the software will be used for adults, the user interface should be more aimed at users with more life experience.

*TinkerCAD ® Circuits*

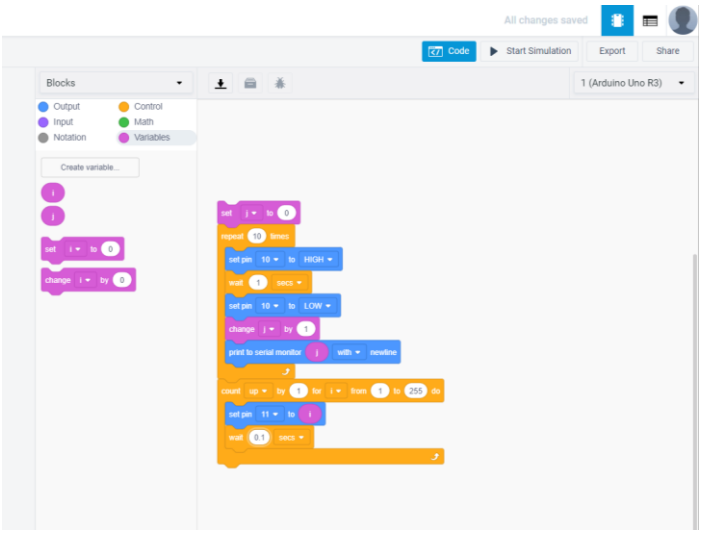



*Figure 6 Creating a program in TinkerCAD [16]*

TinkerCAD Circuits [16] has a block-based visual programming environment. With a similar drag and drop-based programming interface, code for the Arduino can be created. The number of blocks is extremely limited, allowing the user to create only simple programs. As shown in Figure 6, the "language" is remarkably similar to Scratch for Arduino. The blocks have bright colours and fit together like a jigsaw puzzle. This is sufficient when used with the library of components provided in the simulation section of the editor. For creating prototypes with more advanced sensors, like those provided for the toolkit, this would be less than ideal.

*Arduviz*

As Pratomo and Perdana state [17], Arduviz also uses a block-based approach to visual programming, minimizing errors by limiting how the blocks can be connected. They also clearly state that Arduviz is designed as a tool that teaches about Arduino code and coding in general.

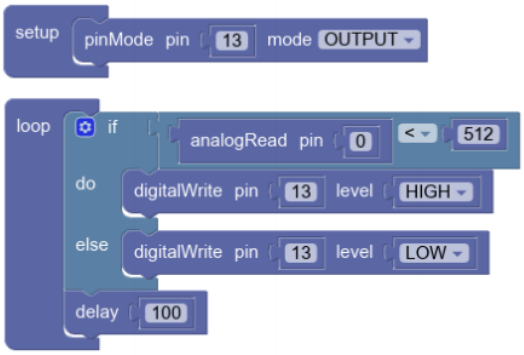



*Figure 7 Arduviz: example of a Light sensor program [17]*

As Figure 5 shows, Arduviz uses a design similar to both TinkerCAD and Scratch for Arduino. Arduviz has the advantage of having a rather large library of blocks, which makes it easy for the user to create code for various applications.

Due to the lack of ready-to-use, still-maintained flow-based visual programming tools for Arduino, a broader view of similar tools is needed to determine whether block-based is indeed the best option for the wearable technology toolkit. To this end, programming tools for similar hardware platforms have been researched in this chapter.

### *Node-RED*

Node-RED is an example of a modern flow-based programming tool, working with premade code blocks with titles such as “Light Sensor” [18]. These blocks can be intuitively connected together to create programs without knowing the code behind the blocks. An example of this ‘code’ can be seen in Figure 6. These blocks with their easy-to-interpret titles allow for quick development of a program.

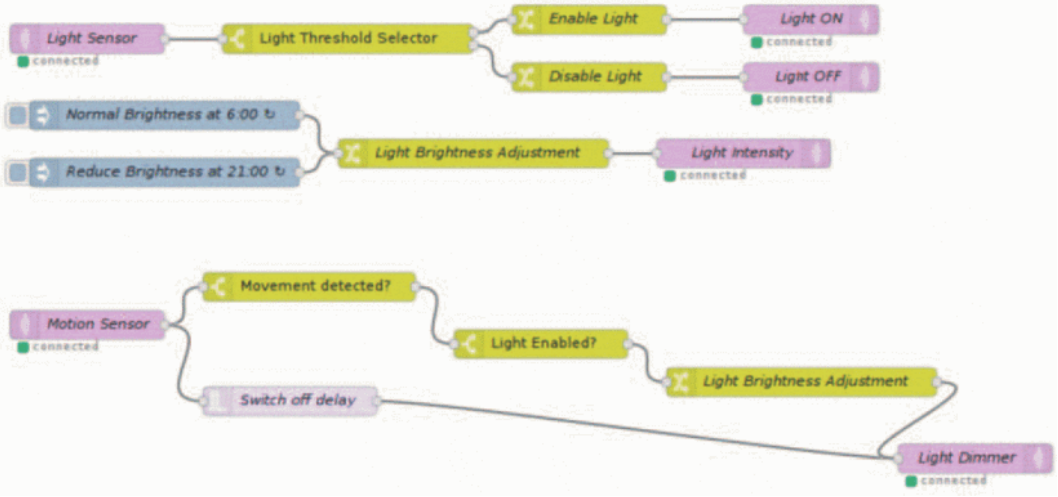



*Figure 8 Node-RED controlling Lights [18]*

### 4.2.3 Visual programming in other areas

To the researcher's knowledge, there is barely any scientific research to be found on the effectiveness of flow-based tools for educational purposes, whereas flow-based programs are widely used in professional applications. This means that no scientific publications could be found that discuss the advantages and disadvantages. For that reason, this section will discuss some of the existing visual programming tools in a non-educational setting.

*Unity visual scripting*

This tool is made to make scripting more accessible for artists and designers, who can work together with programmers to make developing games easier. It is stated that programmers can create custom nodes with visual scripting that the other members of the team can then use to make things regardless of their programming knowledge [19].

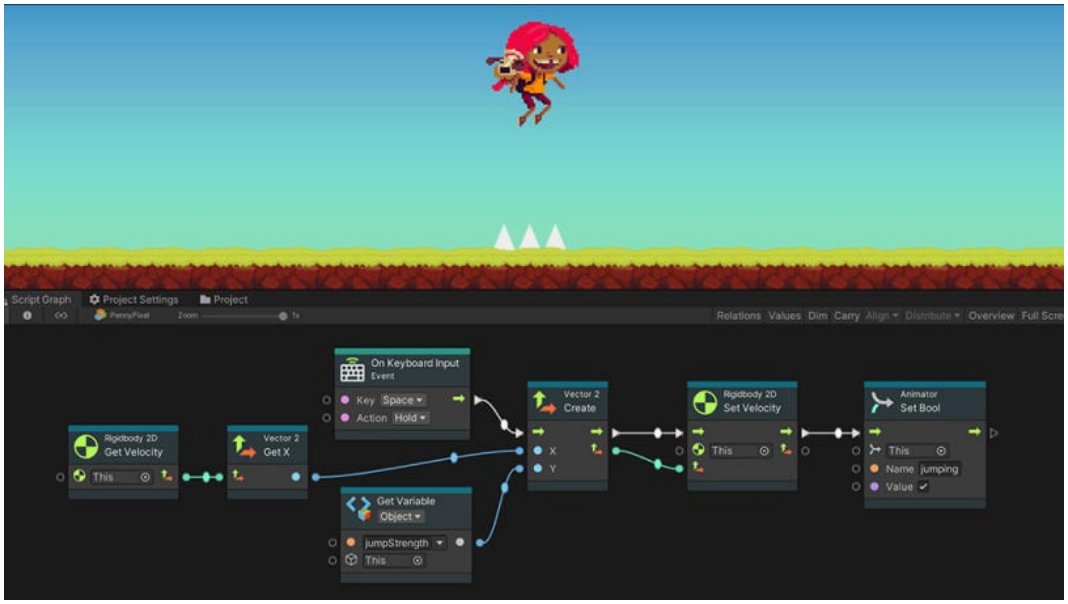



*Figure 9 Unity visual scripting Source: [19]*

In the figure the tool is shown, from this you can get a feel of what using the tool is like. There are nodes that are connected together or have a value set with a box on the same spot as where one could connect a node.

*Unreal engine Blueprint Visual Scripting*

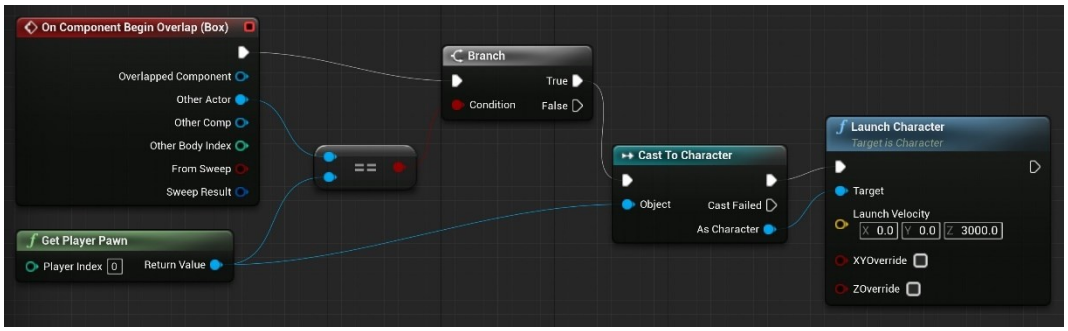



*Figure 10 Unreal engine Blueprint Source: [20]*

This visual scripting system uses a node-based interface. The system is designed to be flexible so that designers can create anything that normally only a programmer could

create. In addition to this it is made in such a way that a programmer can create the underlying systems on which de designer can work using this visual scripting language. When looking at the tool's user interface it is clear that the compile/save and other functions are on the top in a toolbar, there is an overview of the components used and add component button on the left while the right of the screen has a "Details" window. With the middle of the screen filled up with a large window in which the graph editor is located. This is where connecting the blocks takes place.

*Simulink*

To quote the website of the maker of Simulink: "Simulink is a block diagram environment used to design systems with multidomain models, simulate before moving to hardware, and deploy without writing code. [21]" This tool is often used both in education and professionally, where it is used to model real world behaviour in a simulation. This allows for rapid prototyping and makes it easy to test your system before making it.

### 4.2.4 Block-based or flow-based

When considering a visual programming language as a possible solution for the wearable technology workshop, it should be specified what kind of visual programming language will be created. During the theoretical research on visual programming languages, two main types were specified namely block-based and dataflow-based visual programming. The brief overview of the most popular visual programming tools for Arduino and other microcontrollers Appendix B – Current language types shows that only one of these two main types was commonly used, namely block-based visual programming. To further investigate whether this was correct for Arduino, a Google search was performed on 'Arduino visual programming'. The first fifty records of this search query were examined to determine which tools were involved. After removing duplicates, the search yielded 12 unique results. The following categories were identified: block-based, flow-based, and other. Table 3 shows the totals for the different categories. This table generates the following question: Why is block-based hugely more popular than flow-based?

| Block-based | Flow-based | Other |
|---|---|---|
| 9 | 2 (1 of which is no longer updated) | 1 |

*Table 3 Number of visual programming tools by category for Arduino*

### 4.2.5 Suitability of both styles

#### *Block-based*

A block-based tool such as Scratch aims to make code easier to learn. To achieve this, the code has been converted to a visual representation that closely matches the actual code. Due to the close resemblance to the original code, the user still needs to understand many programming principles. Using this tool is a great way to learn the train of thought behind a program. This is why this tool is widely used for educational purposes [11]. For the workshop this style is less well suited as it has a relatively steep learning curve to get started with.

#### *Flow-based*

Visual programming with a flow-based language is different from traditional programming because there is no direct translation to code. The user creates flowcharts to model the desired behaviour. As a result, flow-based languages have a higher level of abstraction than other paradigms. A node can consist of all the code required for the sensor with only the necessary pins as input and the value as output, which makes coding quite easy as almost all coding is done for the user.

# 5 Defining possibilities for the visual programming tool

First, this chapter provides an overview of the basic requirements for the programming tool of the wearable technology workshop. Subsequently, the choice for the type of programming tool is explained. The preliminary requirements are then discussed in more detail.

## 5.1 Choices made for the programming tool

Based on the previously mentioned knowledge in combination with the limitation of a half-day workshop, a number of basic requirements for the tool can be derived. These requirements consist of:

- Creativity is especially important for fashion designers; the tool should allow for a lot of customization using the components in the toolkit.
- The tool should not hinder creativity, it is the task of a designer to produce new innovative uses. Restricting the creativity of the designer in this process is not desirable for the workshop, as this is important for tinkering.
- No technological skills are expected when participating in the workshop, the tool is preferably used with only a short explanation or no explanation at all.
- All sensors and actuators should be supported. As the toolkit expands, new components should be easy to implement.
- All functions from the examples should be present, prioritizing the code constructs that occurred in most projects. In this way, even if a newly developed tool is not yet finished, it can still be used and tested.
- The visual programming tool should allow for rapid prototyping, since the time for creating a program is limited.
- The learning curve should be as small as possible, a user should be able to start using the tool intuitively. This could be done by hiding more advanced options behind a menu, which ensures that the user only sees simple building blocks at the start.
- Familiar interface for designers, based on tools they have used before.
- Easy-to-read documentation for the user allowing the user to be more independent, especially after learning the basics.

The aforementioned basic requirements result in the choice for a flow-based visual programming tool, as it is very similar to the flowchart that people make during the workshop this makes it very easy to quickly go from the flowchart to the corresponding code blocks. A flow-based visual programming tool creates an easy environment to start with, because the number of blocks can be limited to not overwhelm the user. A single block can contain all the code required for reading a sensors value. Connecting these large blocks together also aids in faster prototyping, as explained in 4.1.1. One of the downsides of using this data flow-based representation is that it does not have the same educational value as a block-based language, making the workshop even more focused on generating interest rather than teaching programming principles.

### 5.1.1 Preliminary requirements

For each of the following main requirement's ideas were generated as to what they would mean for a possible software prototype. Some of them might not be feasible later on in the design process due to either time constraints or because they are to complex, but the aim is to at least consider them when making a prototype.

The requirements in 1) are based on the stakeholder interview. During this interview it was clear that in a workshop there is not a lot of time for coding which means it is important to get started as quickly as possible. Part 2) consists of requirements that were gotten by taking the keywords analysis from the brainstorm session. For generating the requirements for the sensors and actuators the capabilities of each of the components was looked at along with how they were used in the brainstorm session and the examples Edwin Dertien provided.

1) Low threshold, high ceiling
    a) Quick installation and setup before being able to start coding
    b) User can start coding with a 5-minute introduction and a simple instruction card
    c) Possible to create a simple program within the first half hour after starting
    d) Output to serial for debugging
    e) Provide proper documentation on all the functions
2) Non-restrictive
    a) Support all keywords from appendix A2
        i) Get values from each sensor as detailed in 3) Sensors
        ii) Control all actuators that are in the toolkit see 4) for more details
        iii) Timers and loops
        iv) Threshold on value of the sensor
        v) Conditional statements, preferably full Boolean logic support
        vi) Counters to keep track of previous events
        vii) Custom calculations/algorithms
3) Sensors
    a) Accelerometer
        i) Acceleration in each axis
        ii) Speed in each axis
        iii) Rotation over each axis
    b) Flex sensor
        i) Threshold -> Boolean
        ii) Raw value
        iii) Value in degrees
    c) GSR sensor
        i) Raw value
        ii) Threshold -> Boolean
        iii) Calibration
    d) Heartrate sensor
        i) Time between beats
        ii) Beats per minute
        iii) Average heartrate
        iv) Logging

e) Sound sensor
    i) Value
f) Touch sensor
    i) Pressed
    ii) Released
    iii) Clicked

4) Actuators
a) Buzzer
    i) Tone control
    ii) Duration control
    iii) Musical notes
    iv) Patterns
b) LEDs
    i) Multiple configurations
    ii) Per LED or as a whole
    iii) RGB
    iv) Effects (optional, could also be created by the user)
c) Servo
    i) Set angle
    ii) Move to angle with certain speed
d) Vibration motor
    i) PWM

# 6 Design of the tool

From the previously mentioned tools a couple of design and interaction elements can be found that are used a lot more than alternatives. Most notably, menu bars are always on the top of the screen. Since a user is used to this location, the menu bar will be located on the top. When looking at unity, Unreal Engine, Maya, and a lot of other (programming) tools, there is almost always an inspector on the right side. This inspector contains additional parameters of whatever is selected, as this tool requires some simple parameters such as: the pin it is connected to and the initial values of the output, the inspector will be implemented in a similar fashion. All the tools that used blocks had the library which contained a list of blocks on the left side where the user could just drag and drop them to the middle of the screen to use them in his/her program.

## 6.1 What should one block look like

Since working from left to right feels more natural as we are used to reading and writing this way, the blocks will be a rectangle with inputs on the left and outputs on the right. When determining the shape of the blocks, the main goal was for them to look inviting to touch. To achieve this the number of sharp corners were minimized as this can impose a sense of threat in users [22]. However, to accommodate multiple inputs and/or outputs in an organised way a straight edge was more convenient. This resulted in the use of rectangles with rounded corners instead of a pill shape.

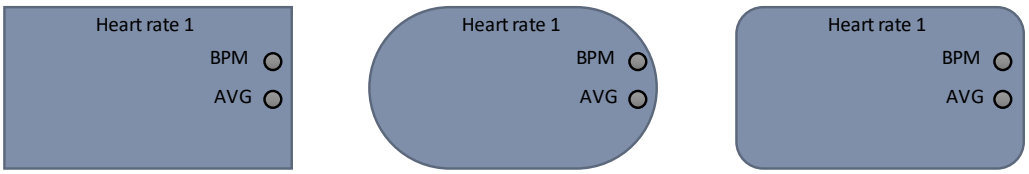



*Figure 11 Block design options*

## 6.2 The interface as a whole

The interface of the program will consist of the following parts, each of these parts is picked to be familiar to programs that are used in the creative industry.

### *Menu bar*

This component is based on the arduino interface but most desktop applications have something similar. A bar at the top from where you have access to all the settings and other commonly used functions like saving, selecting a file and uploading. This was included in this project as a location for the upload, save, settings, serial monitor and a link to help texts.

### *Component picker*

To store a list of blocks for the user to drag into the canvas a component picker is needed, this will sit on the left side of the screen where the blocks are easily accessible for the user. This approach is much like the block palette from scratch or the toolbox from maya.

### *Canvas*

To place and connect the blocks a blank canvas area is required, this is in the middle of the screen as this is where the user will spend most of their attention whilst coding. The canvas area is often seen in other tools such as: Gimp, scratch, unity, photoshop.

### *Inspector*

The blocks have a lot of parameters that you can set, to make this easier the choice was made to not have this under a menu but appear as soon as you click on a block. In a lot of programs this can be done by either single or double clicking on a item. Programs that use an inspector window/settings window are unity, blender, gimp, and photoshop.

Using 6.1 and the choices made in chapter 5 the following first sketch could be created, this sketch was created in PowerPoint to serve as a visual guide when creating the program. As you can see the interface has simple blocks with connection points that link together with a line. To create this line the user has to drag from one connection point to the other.

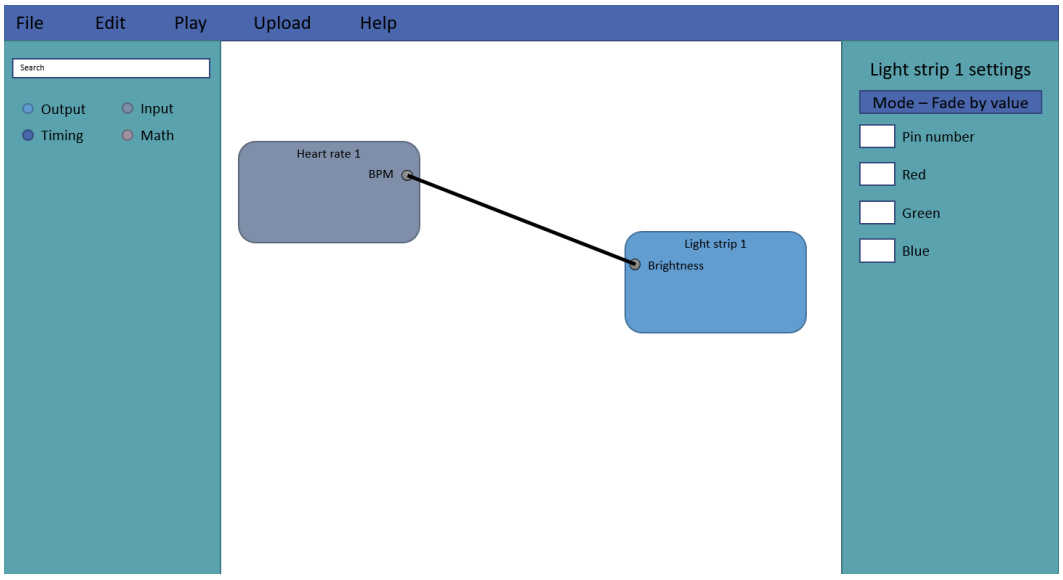



*Figure 12 The design of the GUI*

## 6.3 Refining colour choices

Looking at the colours of this first prototype, you cannot distinguish an actuator from a sensor at a glance. To improve this the colours were changed to red for actuators, green for sensors, and blue for other assisting blocks. These colours ended up looking very out of place on the blue background, so we switched over to a dark theme. This dark theme also feels more familiar to experienced programmers. This resulted in a more uniform look for the software as a whole and made the type of the blocks more significant in the way they appeared.

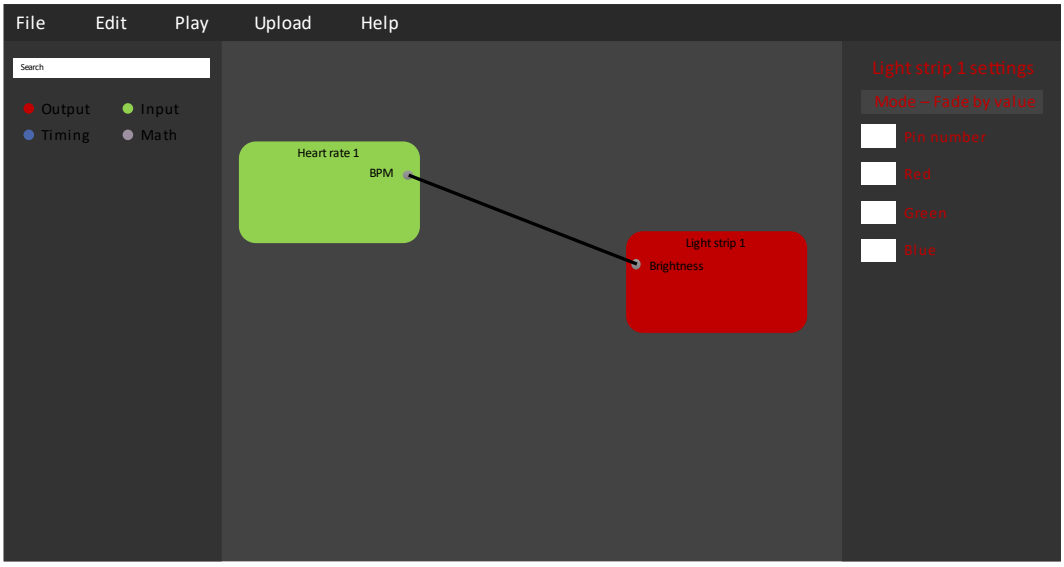



*Figure 13 The GUI after the makeover*

## 6.4 Technical implications of this design

From the design that was made a lot of additional requirements about how the user will interact with the prototype became apparent. The following GUI related requirements for the application were used when creating the application:

- The layout of the window should look as similar to the sketch as possible:
  - Menu bar on the top where all the general settings are located, these are not used for creating the code but the buttons here are used for saving, uploading, help and changing user preferences.
  - Inspector window on the right where the user can set certain parameters of the blocks.
  - Library window where the user can easily distinguish the block that he/she needs for creating a prototype.
  - Canvas area where the user can drag and drop the blocks to create the actual flow of the code.
- The code blocks have to be draggable so that the user can arrange them in a way that makes sense to him/her. When dragging the blocks of the screen into the library the block should be put back in the library/removed from the code that the user created.
- There could be a way to upload the code to the microcontroller that is easily accessible within the application, preferably the user does not need to copy paste text into the 'normal' Arduino program, an upload button that handles detecting

the Arduino and uploading the code to the Arduino is desired. If this is not feasible it should at least not take a lot of effort to upload the code using the traditional Arduino ide.

## 6.5 Priorities for design

To gain more insight in which features should be developed first for a functioning prototype the MoSCoW [23] method was used. MoSCoW has four categories for the features that can be developed: Must, Should, Could and Would. The functions that are in must are needed for even the most basic user test, even without any other functionality the user interface of the tool and its effectiveness could be assessed without any actual code generation, this is why this category contains most of the functions related to basic user interaction. Should contains all the functions that are needed to generate Arduino code, making the prototype functional even though it might not contain all the desired functions that make it easy to work with, this allows testing the feasibility of this tool for and allows for testing if it is possible to make it function. Could contains uploading directly and saving/loading of the workspace, this would allow the user to work with the tool completely independently without the researcher needing to upload the code for them, thus leading to less interruptions in the user's experience. In Would there are some features that would have been implemented if the tool was actually developed fully.

| MoSCoW | Function |
|---|---|
| Must | Have blocks that represent sensors/actuators/other |
| | Be able to spawn new blocks |
| | Linking the flow blocks |
| | Easy to implement new sensors/actuators by creating different blocks |
| Should | Set parameters of the flow blocks |
| | Create Arduino code |
| Could | Upload the Arduino code directly |
| | Save and load the workspace to continue with a project later |
| Would | Resizable windows |
| | Live mode, where the code is directly executed without uploading to the microcontroller |

*Table 4 MoSCoW for the prototype*

# 7 Implementation

## 7.1 Processing

To create the prototype as proposed in the previous chapter, a program needs to be created. For this the language Processing was chosen because it is easy to program visual prototypes in Processing. The language is based on Java but has a lot of functions that make drawing simple shapes to the screen a lot easier.

## 7.2 General graphical interface structure

To structure the creation of the classes a class diagram was made that shows how the main components interact. The following structure diagram is only for the graphical user interface, more on the other classes will follow in subsequent sections.

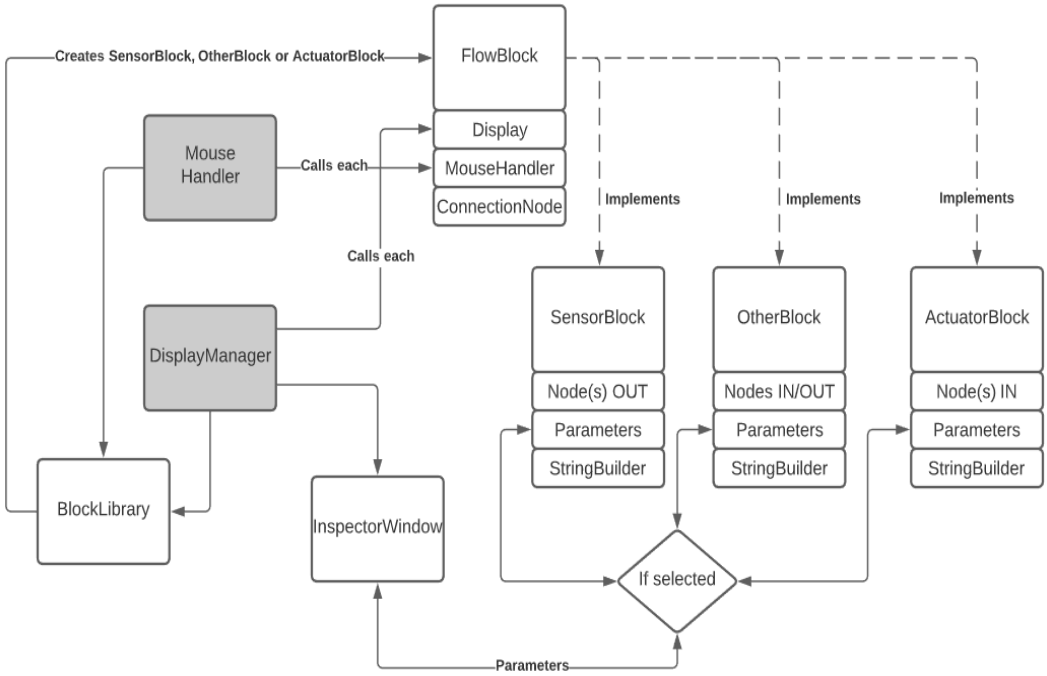



*Figure 14 General structure of the program*

Figure 14 General structure of the program shows, there are multiple classes that make up the interaction and display part of the code: display manager, mouse handler, block library this allows the user to spawn in new flowblocks, inspector window a way to set the parameters of the blocks that are used, and flow blocks a super class that has the common behaviour of all other blocks (sensor block, other block, and actuator block). Each of these will be discussed in the following subsections.

## 7.3 The resulting interface of the program

Before discussing the content of all the classes that were created, we will briefly discuss the programs interface. The program has a simple dark theme as shown in Figure 15 each of the block types has its own unique colour to make it easier to distinguish the type of the block at a glance. What is missing from the design sketches made previously is the collapsible menu on the left. Whilst adding all the blocks it was found that they all fit comfortably on the screen of the user, this meant that the menu was no longer necessary and was left out for simplicity.

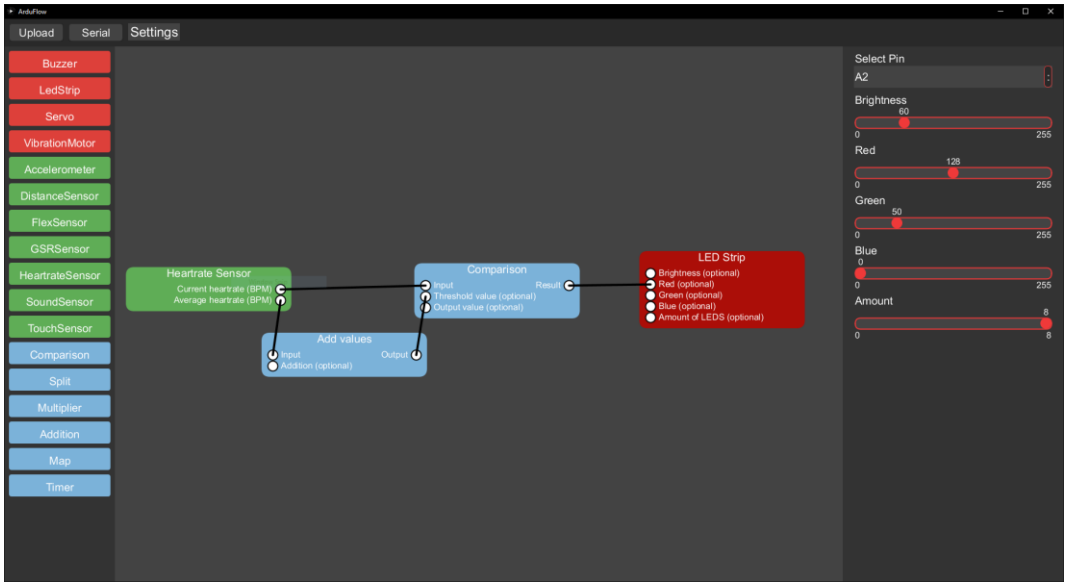



*Figure 15 An example of a heartrate sensor that goes red when the heartrate is above the average heartrate*

## 7.4 The classes in detail

In this section there is a brief description of how the main classes work.

### 7.4.1 Display manager

This class handles all the drawing and displaying that needs to be done for the user interface. It does not contain a lot of functionality, what it does is create a structured way of calling all display functions of the flowblocks, menu bar, block library, and inspector window. The only function we will discuss here is the *drawBlocks()* function.

#### *drawBlocks()*

This function iterates over all flowblocks that are currently on the screen and calls their individual display() functions. After drawing all the flowblocks, the function iterates over all flowblocks again, now drawing their connectionNodes and the lines connecting the node, this to make sure that all connection lines are always drawn on top of the actual blocks.

### 7.4.2 Mouse handler

Contains methods for *mousePressed()*, *mouseDragged()*, *mouseReleased()*, and *mouseClicked()*. This class handles all things mouse related basically driving all interaction that the program contains.

#### *mousePressed()*

Checks if there currently is a locked block (the block that is being dragged) and if there is no locked block, loops through all blocks and determines if they are being pressed. If a block is detected at the mouse position this block is now the locked block.

#### *mouseDragged()*

If there is a locked block execute the *mouseDrag()* of this block, meaning that this block will either move with the mouse or create a connection between two nodes.

#### *mouseReleased()*

Set the locked parameter to false, unlocking interaction for all blocks to be interacted with again. Additionally, call the locked blocks *mouseUnpressed()* function.

#### *mouseClicked()*

Handle short mouse clicks currently only used for the block library to spawn new blocks.

### 7.4.3 Block library

A class that handles the creation of new blocks. This class contains a list of blocks which are displayed as seen in the following figure. The blocks are coloured according to their type to help the user understand if it is a sensor, actuator, or other block.

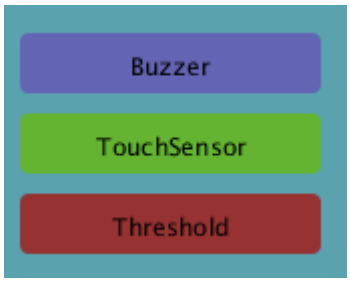



*Figure 16 Minified version of the block library.*

If any of these buttons are clicked a new block is spawned of the type matching the text on the button. For example, if the Buzzer is clicked a buzzer block is spawned.

When looking at the full list of components and their coding block counterparts that are included in the current version of the programming tool it was deemed unnecessary to make a menu inside this block library as all the buttons could comfortably fit in the menu on an average sized laptop display.

### 7.4.4 Inspector window

The class named SettingsMenu handles the inspector window, when the active block is changed it removes the current items displayed on it, changes the theme to match the colour scheme for newly selected blocks type, then it generates all the menu items (such as sliders and dropdown lists) that should be included in the menu for the current block. To do this it has the following methods:

- activeBlockChanged
- closeSettingsMenu
- display
- createLabel
- createSliderWithText
- createSlider
- createList

### 7.4.5 Flowblock

The superclass from which all other blocks inherit the following functions: *display()*, *mouseDown()*¸ *mouseDrag()*, *mouseUnpressed()*, and *checkPosition()*. This allows all blocks to be displayed and the mouse to be handled. The subclasses from the flowblocks class are kept as minimal as possible to allow for easy creation of new block types.

*Sensor block*

Only contains the colour of the flowblock, and the pin variable, this class is mainly used for organization purposes. By having the other sensor classes be a subclass of this it is easy to detect that they are sensor blocks. The following TouchSensor class is a subclass from the sensorblock class, to save space not all other subclasses will be discussed, for more specifics on each subclass look at the supporting files containing the full code.

*TouchSensor*

This class contains all code required for the touchsensor block; this is a nice example of how the sensor classes work. It contains a constructor that calls the SensorBlock constructor and adds a connection with the label Measured Value. The stringBuilder is explained more elaborately in the next subsection but this returns the string containing the function call that is needed in Arduino to get the value of this sensor.

```
public class TouchSensor extends SensorBlock

{
  TouchSensor(float _xPos, float _yPos)
  {
    super(_xPos, _yPos, 300, 60, "Touch Sensor");
    Connections.add(new Output(BLOCKWIDTH - 30, 40, "Measured
Value"));
  }

  String stringBuilder()
  {
    return "touchSensor(" + String.valueOf(pin) + ")";
  }
}
```

*Actuator block*

Same as the sensor block this class only contains a colour and pin number and is mainly used for organization of the code.

*Buzzer*

```
public class Buzzer extends ActuatorBlock

{
  float tone = 440; //default frequency in hz

  Buzzer(float _xPos, float _yPos)
  {
    super(_xPos, _yPos, BLOCKWIDTH, 80, "Buzzer");
    Connections.add(new Input(this, 20, 40, "Volume"));
    Connections.add(new Input(this, 20, 60, "Tone
(optional)"));
  }

  String stringBuilder()
  {
    String toneString = String.valueOf(tone);
    String volumeString = "";
    for (ConnectionPoint c : Connections)
    {
      if (c.label == "Tone (optional)" && c.connectedTo !=
null)
      {
        toneString = c.connectedTo.stringBuilder();
      } else if (c.label == "Volume" && c.connectedTo != null)
      {
        volumeString = c.connectedTo.stringBuilder();
      }
    }
    if (volumeString == "")
    {
      return "";
    }
    return "buzzer(" + String.valueOf(pin) + "," + volumeString
+ "," + toneString;
  }
}
```

This class functions much the same as the touch sensor example, only the *stringBuilder()* is a lot more elaborate. The *stringBuilder()* returns a string when it is called that contains its parameters and Arduino function call, in order to do this, it checks the label of each connected connection and if this matches one of the expected parameters save this in a string, this string is then combined with the other parameters into the Arduino function call and returned.

*Other block*

In this class all blocks that are not actuators or sensors are gathered, they serve as a way to manipulate the signal before sending it to the actuator.

*Invert*

```
public class Split extends OtherBlock
{
    Split(float _xPos, float _yPos)
        {
        super(_xPos, _yPos, 300, 80, "Split");
        Connections.add(new Input(this, 20, 40, "Input"));
        Connections.add(new Output(this, 300 - 30, 40, "Output
1"));
        Connections.add(new Output(this, 300 - 30, 60, "Output
2"));
    }

    String loopBuilder(ConnectionPoint output)
        {
        for (ConnectionPoint c : Connections)
            {
            if (c.label == "Input" && c.connected())
                {
                return c.connectedTo.loopBuilder();
            }
        }
        return "";
    }
}
```

Here you can see an example of the split build block, this block does not have any Arduino code as all it does is split the signal in two. So, when it is called by an actuator it just returns the loopBuilder of whatever is on the input side.

### 7.4.6 Connection point

Handles all the connections between the blocks. All flow blocks have one or multiple connection points. The connecting point contains a reference to the block that it is on and a reference to a different connection point when connected.

## 7.5 Code creation

All the arduino code is contained in functions, each flowblock has a stringbuilder that contains all the code to generate a string that is a valid call of this function. This string/function call contains one or more parameters, these parameters can either be set in the inspector window or by connecting a flowblock to the input of this parameter. If a flowblock is connected to this its stringbuilder will be called and the resulting string is then inserted into the spot of the parameter. The advantage of putting most of the code into the premade arduino functions and then only generating the function call is that it is easy to generate code and it remains readable. One downside is that it requires the programmer to always write an Arduino function even for something that could have been a single line of code. Another downside is that this does limit the code that can be created as not every single code concept has a corresponding arduino function and flowblock, as this could be a limiting factor the high ceiling of the tool this is evaluated during the user tests.

### 7.5.1 Arduino functions

To allow for all of the possibilities that can be made with the prototype the Arduino code needed to be modular and all functions should be as self-contained as possible, this makes generating the code a lot easier. Because of this the following Arduino function design was made:

```
int sensorName(int pin, int parameter, int parameter2, etc.)
{
    return value;
}


void actuatorName(int pin, int input, int parameter, etc.)
{
    //actual code that does something based on the input
}


int other(int value, int parameter, etc.)
{
    return newValue;
}
```

By only using int as a datatype the value does not need to be changed for a LED strip or a servo, both will just scale the value internally. A on/off signal or a Boolean is thus represented with the value 0 or 1023, these values were chosen as they are also the maximum value of *analogRead()* on an Arduino, therefore there would not be a difference between the brightness of a LED strip when turned on with a Boolean or when the max input is reached on the analog input.

### 7.5.2 Building and uploading the code

To go from these building blocks to an actual Arduino program the following setup was used:

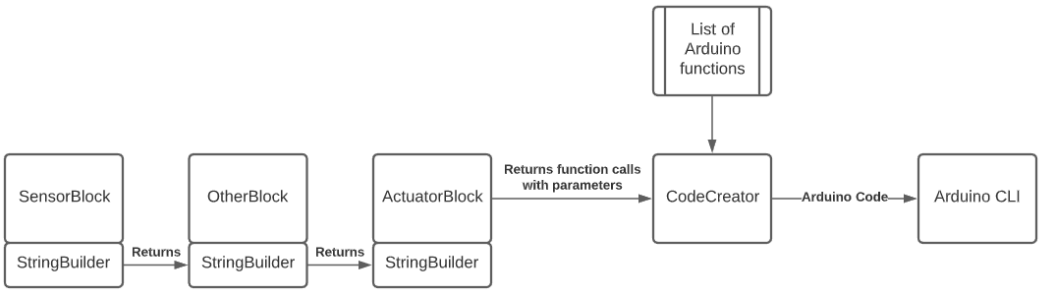



*Figure 17 The structure of the string builder and Arduino Command Line Interface*

#### *CodeCreator*

When the user presses the upload button, first the CodeCreator class is called. This class creates the header, adds the variables used, adds the import statement for the libraries and finally creates the main loop of the code, all that is needed in the loop are nested function calls with their respective parameters for most blocks. For this the *create()* function calls all actuators *stringBuilder()* functions and puts these into a string separated with a newline after each actuator. This actuator then calls the stringBuilder() function of what the blocks that it is connected to and so on until we reach a sensor or dead-end. An example of the following example would like both in code and visually in the programming tool is this:

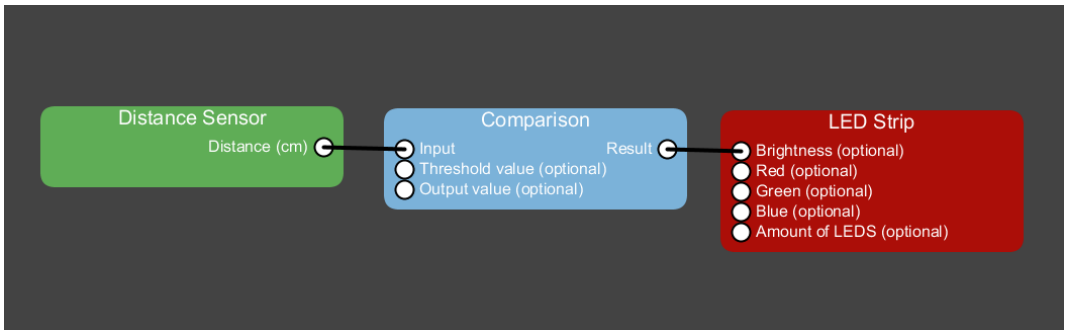



*Figure 18 A LED strip that turns on when the distance is greater than 50cm*

The code:

```
ledStrip(A2,comparison(distanceSensor(A2),">",50,255),128,0,0,8);
```


*Figure 19 Example of the hover over text in Arduino IDE*

The resulting code is compact and still readable for any user that wants to use the functions or tweak the parameters without using the tool, especially if you use any IDE that has hover over text that shows you information on the method. When all the code creation is done it is saved in a folder that has the same file as the .ino sketch alongside the file containing all the Arduino functions that were created.

### 7.5.3 Arduino CLI

The Arduino command line interface is a powerful tool that allows users to access Arduino boards through the command line or a daemon [24]. Using this tool, it is possible to detect the connected board, compile Arduino code for it and then upload to that Arduino compatible board. This is currently not implemented as for testing the concept it was not required but could definitely be a great opportunity later on to make the interaction more seamless.

## 7.6 Functional evaluation

To determine if the tool meets the minimum requirements to use during the user tests, the tool was analysed using MoSCoW and the general requirements that were determined. This was done multiple times until most of the important features where implemented.

### 7.6.1 MoSCoW

During this project it was not feasible to implement all features that Could or Would have been in the programming interface given more time. The following MoSCoW table shows what currently has been implemented.

| MoSCoW | Function | Implemented |
|---|---|---|
| Must | Have blocks that represent sensors/actuators/other | Yes |
| | Be able to spawn new blocks | Yes |
| | Linking the flow blocks | Yes |
| | Easy to implement new sensors/actuators by creating different blocks | Mostly |
| Should | Set parameters of the flow blocks | Yes |
| | Create Arduino code | Yes |
| Could | Upload the Arduino code directly | No |
| | Save and load the workspace to continue with a project later | No |
| Would | Resizable windows | Yes |
| | Live mode, where the code is directly executed without uploading to the microcontroller | No |

*Table 5 MoSCoW - Implementation state*

When looking at Table 5 all the Musts but one and all the Shoulds are implemented, this suggests that the program that was created has all the minimal features needed to be used, from the features that are lacking the highest priority one is “Easy to implement new sensors/actuators by creating different blocks”, a brief explanation of why it is classified as mostly is the following. While it only takes a small amount of time to go from the Arduino function to the block that represents it and the code that generates its code this could be made simpler. A great way to do this would be to make the interface read the Arduino return type, name, and parameters. When this is done a quick menu should pop up that asks for if the parameter needs to have an input connection, as slider, and what the minimum and maximum values are. Then if the function has a return type that is not void an output connector could be added to the block. This would greatly improve the speed at which new functionality/hardware can be added to the visual programming language.

# 8 Evaluation

When the final prototype was realised, the prototype was used in a user test with potential end users. The aim of this evaluation is to show whether the tool was effective, and to define areas that need more research/work before the tool can be used. After conducting the test, a questionnaire is conducted focussing on low threshold, wide walls, and high ceiling.

## 8.1 Design of the workshop

To test the effectiveness of the tool that was created a short workshop/assignment was created for the test subjects to follow. The contents of the workshop are briefly described here, you can find the assignments as they were used in Appendix C – The workshop. During this workshop the user used the toolkit as discussed in **Error! Reference source not found.**. The assignment consisted of three parts that progressively increased in difficulty and freedom. First the user was given an information sheet that contained a general description of how-to code in the interface, how to connect sensors/actuators to the Beetle, and a description of all the available blocks and their properties.

*Part 1 – Getting started*

The first assignment was to follow the steps described in the workshop. Here, the user made a heartrate sensor connected to the number of LEDs of the LED strip using a mapping function. The main goal of this assignment was to familiarize the user with the interface and the hardware. To do this effectively the workshop contained simple instructions to guide through each step of connecting the hardware, using the interface of the tool, and coding.

*Part 2 – Time to tinker*

The second assignment was about tinkering with the previously made code. First the user got to change something about the code, for example: make the LEDs go red if the heartrate is above 120 bpm. After this, the user got to add a sensor or actuator of choice. If the user had no idea what to connect, one of the cards could be used for inspiration. This part forces the user to slowly make more substantial changes to the code and program big parts of the system at once.

*Part 3 – All the freedom*

The third assignment was not really a traditional assignment. The question to the users was to think of something they could do with the given hardware, write down what they were going to do and finally build it both in code and physically. Here the user was given all possible freedom to create something. This caused the user to explore more of the toolkit's options allowing them to explore as much of the needed programming as possible.

## 8.2 Questionnaire

The questionnaire consists of three main parts, namely:

- Questions to gain insight into their previous programming experience. (Questions 1 to 6)
- Questions about the tool and the test workshop. (Question 7 to 20)
- Questions specific to programmers who used the tool. (Questions 21 and 22)

Most of the questions were formulated in such a way that they could be answered with yes/no/something in between, but with space to fill in a further explanation if the user wanted to. This resulted in data that was easy to analyse using a three-point Likert scale, but still gave qualitative feedback on the tool as well. The qualitative information was mainly used to determine where possible shortcomings were in the tool and to validate that the answer was about visual coding style and not the specific implementation. Let us say for example that a user decides that there are technical problems that a user encounters with the code. This could be that a certain button does not work. This is not necessarily related to the general idea of visual coding but is a purely technical/ implementation problem. Quantitative research cannot discern the difference between implementation and ideation problems, which motivates the need for further elaboration by the participant.

The survey will be conducted on location, as the tool can only be used with certain hardware that the participants do not have. Therefore, the users need to be present on location. The three sub-research questions each focus on another aspect of what makes the tool successful, therefore the questions about the tool (questions 7 to 20) were chosen to match with these aspects. The resulting questionnaire can be found in Appendix D – Questionnaire.

## 8.3 Ethics

Before a user test can take place, the participant needs to be informed and consent to the use of the gathered data. To inform the participant the information document as seen in Appendix E – Information document was used. They also received two consent forms (as seen in Appendix F – Consent form) to sign of which they could keep one. The data gathered during the user test was anonymized to ensure privacy.

## 8.4 Test setup

For both the pilot test and the full user test the setup was as described in this section. The total time of each test ended up from 30 to 50 mins including filling out the questionnaire in the end. Upon arrival of the participant, they were first handed an information document about what they were going to do and an ethics permission form. When this was read and any possible questions where answered the test began. They were asked to go to a desktop where the tool was already open. Then they received the following: the assignments, a cheat sheet containing some information on how to use the program, some paper to make notes, and the toolkit. The goal of the test setup was for it to be as close to a workshop as possible, this is why the participant is left to work on the assignments themselves. Any questions regarding the tool could be asked and the researcher did check on their progress at times, but the researcher was not constantly working with them/looking at what they were doing. This approach where there was help available, but the participant works independently is a lot like a full

workshop with limited assistance per participant. When the participant was content with the last assignment, they are asked to fill in questionnaire, during this the researcher sits the other way or leaves to room to make sure it was anonymous.

## 8.5 Pilot test

First a pilot test was conducted, which had a sample size of 1. The pilot test took roughly one hour. The pilot test differed from the previous test setup slightly, the researcher was observing more closely and asking questions more often when compared to the more independent tests that were held later. The goal was to determine whether there where flaws in the assignments or in the tool that prohibited proper testing. During this pilot test a few errors in the prototype where found, mainly the Arduino code not working if there was no LED strip defined and other small issues. Also, the test user had some remarks about the naming of the blocks which was subsequently modified, for example having an addition and multiplication block made little sense as it could be combined into one simple math block. Finally, the assignments did not mention that the user should upload the code after each assignment. After the pilot test the following changes were made small bug fixes, combining math into a single block type, and adding a sentence telling the user to upload their code after/during each assignment.

## 8.6 Results

### 8.6.1 Defining the participant groups

Each of the participants was asked how comfortable they are with programming. Based on this, the participants were assigned to two groups for analysis of the questionnaire:

- Not comfortable with programming: participants with a programming comfort level of 0 to 5. This group consisted of three participants, one of which never programmed before.
- Comfortable with programming: a programming comfort level of 6 to 10. This group consisted of 7 participants.

The overview of each participant's programming comfortability can be seen bellow.

| ID | Group | Programming comfortability |
|---|---|---|
| A | Comfortable | 7 |
| B | Not comfortable | 3 |
| C | Comfortable | 8 |
| D | Comfortable | 7 |
| E | Comfortable | 8 |
| F | Not comfortable | 4 |
| G | Comfortable | 6 |
| H | Not comfortable | 0 |
| I | Comfortable | 6 |
| J | Comfortable | 6 |

*Table 6 Programming comfortability of the participants*

### 8.6.2 Low threshold

*Likert score*

To provide a better overview of the data a three-point Likert scale was used. If the answer of the participant was in line with the tool having a low threshold the question got a +2, if the answer of the participant was neutral the question got a score of +1, if the answer opposes the tool having a low threshold the answer got a score of +0.

Seven questions related to the low threshold of the tool to get started, after coding the answers to these questions the following box plot was created. The maximum score of the Likert scale with seven questions is 14 (seven questions multiplied by +2).

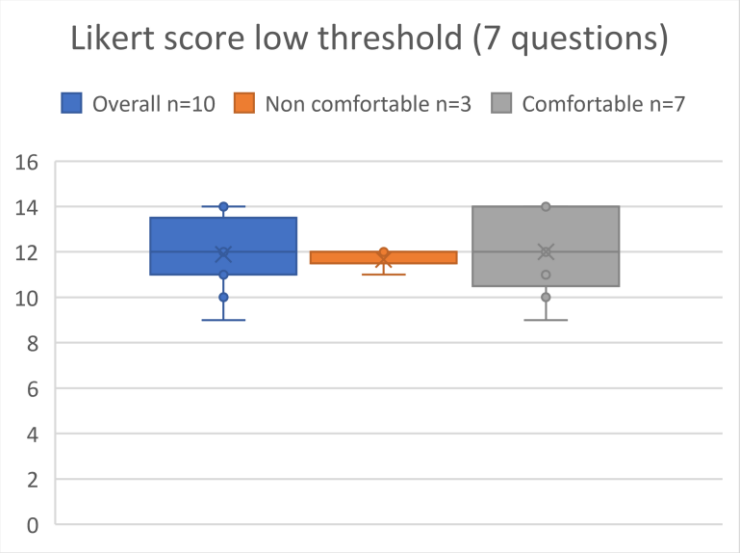



*Figure 20 Low threshold Likert scores*

As any score above 7 indicates that the user was at least slightly positive about the low threshold of the tool, the scores in the boxplot give a clear indication that the tool does have a low threshold. To explore the specific aspects of the low threshold, the questions themselves are now examined separately.

*Usage without any guidance*

Most users indicated that they could have used the tool without any guidance, however five out of the ten users stated that the guidance they received during the workshop significantly reduced the amount of time it took to get started with the tool.

*Time required to get started*

The users indicated that it did not take long to get started with the programming even though participant H did mention that the assignment contained a lot of text.

*Experienced difficulty*

This aspect consisted of three questions relating to the increasing difficulty of the three assignments. Most of the users did not experience the tool as being difficult to program with, however participant F noted that the programming of the last part was difficult for them, but also stated that they were giving themselves a significant challenge.

*Helpfulness structuring idea*

All the users but user A stated that the tool was helpful in giving structure to their idea. Participant E stated that the fact that the tool was limited was an advantage here as it made brainstorming a lot easier and the tool is more straightforward.

*Required effort turning idea into code*

Most of the users found the tool to require surprisingly little effort to get their ideas converted into code or as participant J said: “Surprisingly little, I usually struggle with coding in Arduino”.

### 8.6.3 Wide walls

*Likert score*

There were three questions that directly corresponded to wide walls, after coding the answers to these questions the following box plot was created. The maximum score of the Likert scale with three questions is 6 (or three questions multiplied by +2).

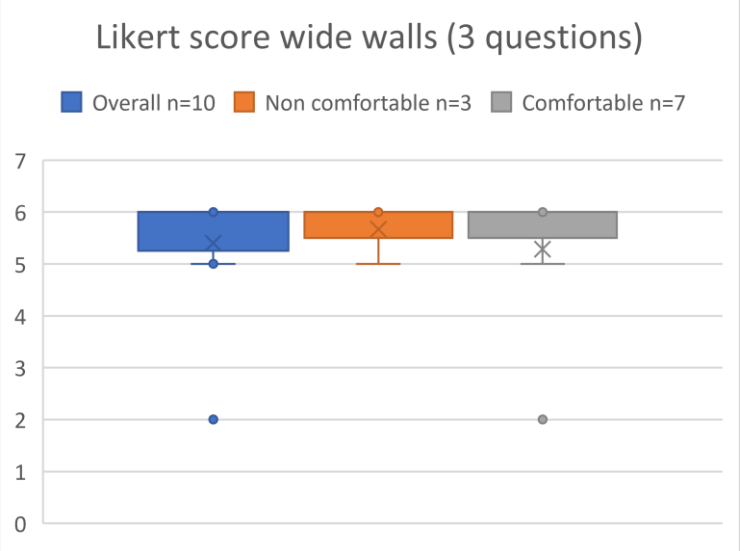



*Figure 21 Wide walls Likert scores*

Looking at the box plot it is clear that using a three-point scale for three questions had a ceiling effect on the possible outcomes of the score, as for seven of the participants the resulting score was six. This still indicates that all participants were positive about the tool in terms of the diversity of things you can do with it. The only outlier is participant A who felt that the tool limited the coding style.

*Enabling converting idea into code*

Seven of the users stated that the tool did allow them to convert their ideas into code. Two stated that they got close to their idea, however both of them felt that they could have solved their issues if they had been given more time. “At first it seemed like there was no function for what I wanted, but that could be solved by using other functions.” One user stated that the tool did not enable them to make what they wanted.

*Motivating to experiment with the components*

All ten of the participants agree that the tool is motivating to experiment with the components used during the workshop. Some quotes that support this are: “As I said, I don't have prior experience with the hardware components, but I feel that I could work with them nevertheless”, “It was very easy to swap out and made me curious to try other sensors and actuators”, and “It was very rewarding and fast to make a new component work, so I wanted to keep adding new things.”.

*Tool enabling to create other things*

Eight of the participants thought they could make other things that they did not create during the workshop. Some examples of their answers are: “I think plenty is possible.” and “The programming language would not be a restriction to let my creativity flow”. These answers clearly state that the users had confidence in being able to create different things using the toolkit and the software tool.

### 8.6.4 High ceiling

*Likert score*

There were three questions that explored the tool having a high ceiling. The resulting box plot can be seen below. The maximum score that can be achieved is six. Since in answering these questions, users were asked to compare with normal coding, the user that never programmed before was excluded for this criterion. Since a high ceiling usually relates to the experienced user, this was not a problem.

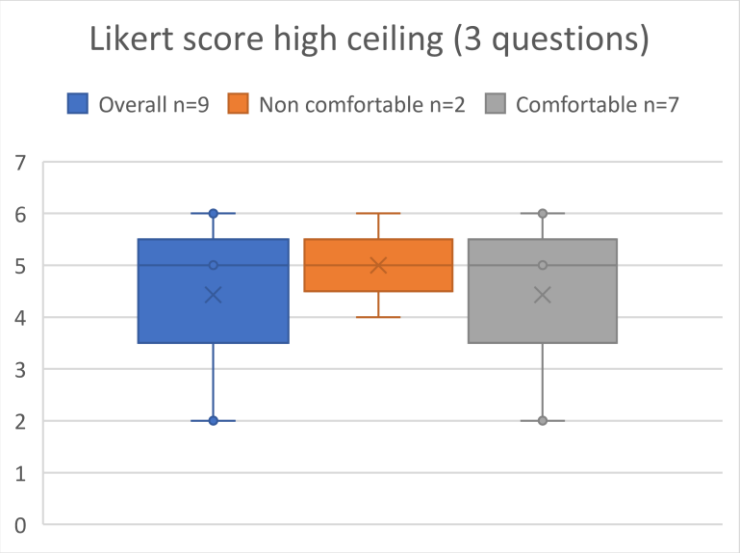



*Figure 22 High ceiling Likert scores*

Figure 22 shows that the scores fall into neutral to positive. To determine whether the users were actually positive about the tool, we need to investigate what their exact answers are, as it is likely that only some parts of having a high ceiling were achieved while others could be missing.

*Confidence in prototyping more complex projects*

When asking the users if they had confidence in the possibility of using the tool for creating more complex projects, they answered the following: “Yes, although it might get very messy, and it might become more cluttered for larger projects” and “Sometimes the blocks are limiting as I would have more control coding stuff myself.” It became clear from these responses that whilst it might be possible to use the blocks to create more complex programs, it could become cluttered or confusing for the user.

*Time saving compared to normal coding*

Six users stated that the tool saved them time compared to coding similar projects; two users stated that it saved time for simple projects and one stated that they could not compare it, because they had not done similar projects before. Explaining why it saved time, one of the participants stated: “It prevented many syntax errors and minor mistakes I usually make when coding. It also made it easier to change orders of operations, change parts and keep a clear overview of the program.” Another stated that “it might be 10 times faster”. So even if it limited the user with advanced projects, it would not hinder the simple prototypes created during the workshop.

*Some things were more difficult than normal coding*

Three users found doing something with the tool to be more difficult than with traditional coding. The examples they mentioned are math (especially more advanced things like sine waves), logic gates, and making your own classes/functions.

### 8.6.5 General results of the questionnaire

*The tool feeling natural to use*

The participants of the workshop got the following question: "Did using the tool feel natural to you? (Where 1 is not at all and 10 is extremely natural)". The result was then plotted into a box plot. From this plot we can tell that the users found the tool natural to use to at least a certain degree.

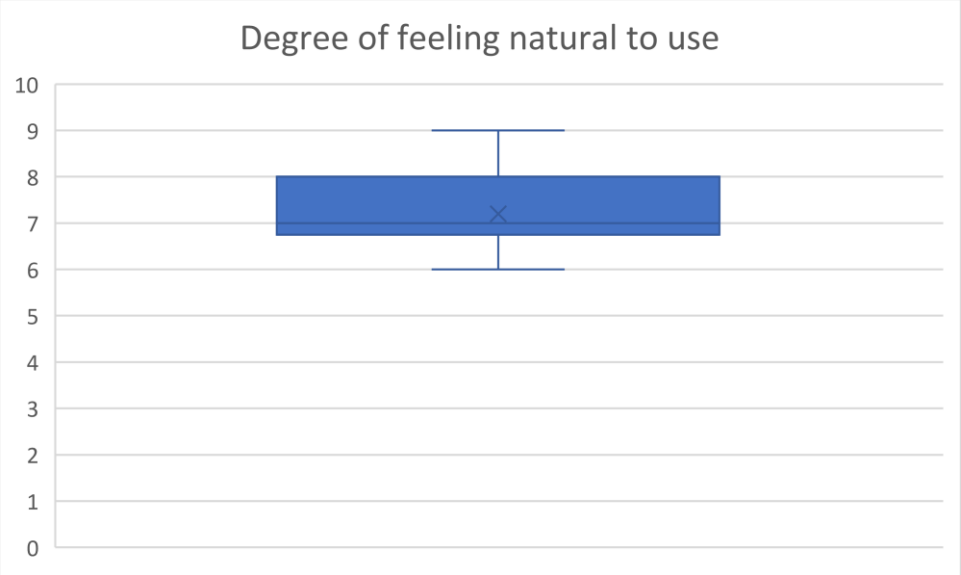



*Figure 23 How natural was using the tool?*

*Likeliness using the tool again for creating quick prototypes*

The questionnaire included asking users to rate from 1 to 5 stars how likely they are to use the tool again when creating quick prototypes for Arduino. Since the goal during the workshop is to create quick prototypes and not necessarily design full products, the question was specifically aimed at quick prototypes, as for creating a fully featured product the tool might not have all the functions (yet). The results are shown in the following graph.

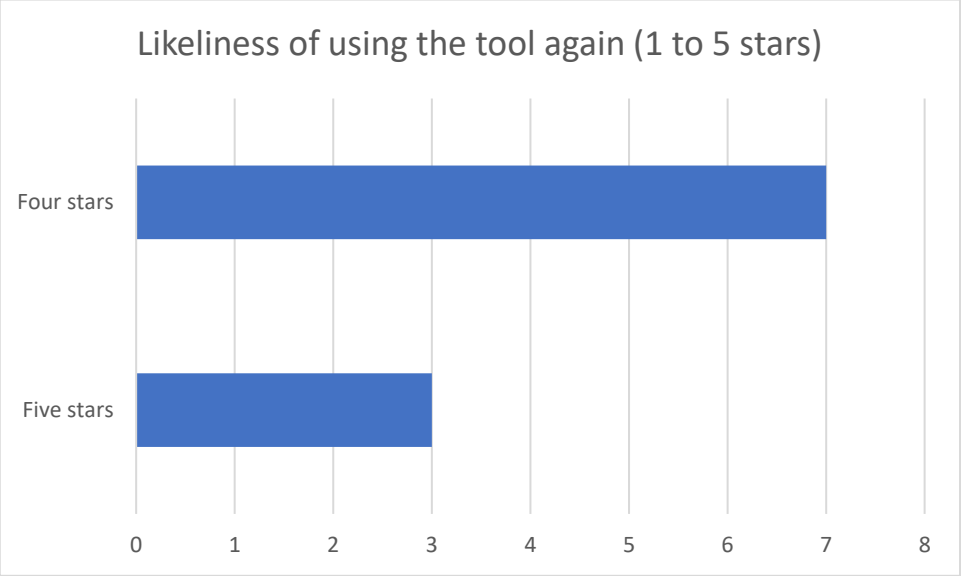



*Figure 24 How likely are users to use the tool again?*

### 8.6.6 Observations during the workshop

In addition to collecting data from the questionnaire, the researcher made observations during the workshop. These observations are briefly discussed in this section.

*Functionality*

Something that users missed when making the assignments was a reset button, a way to quickly clear the entire canvas when starting on a new assignment. What three users tried but was not implemented was trying to connect multiple inputs to one output, this was their preference over having a separate split block.

*UI*

All workshop participants tried to double-click the value input box, which is not supported by the library. Some participants even tried this multiple times. This finding shows that while this was not in the tool's requirements, it probably should have been implemented. In addition, it was not clear for some users that if there was something connected to the node that that means that the corresponding sliders does not work anymore.

*Hardware*

A usb extension cable would have been a welcome addition to the toolkit, now it often was difficult to upload code whilst keeping all the hardware connected to the microcontroller.

*Computational thinking*

Users who did not write down or explain the desired behaviour often got interesting results that did not match their intentions. While this is interesting, it does mean that participants can get frustrated with themselves, because they end up searching for a block that does one very specific thing, rather than first elaborating their idea and then going to a program one step at a time.

*Motivating to experiment*

Some participants got sidetracked by experimenting with everything they could instead of following the given assignments. While this is a great result in terms of tool immersion, it does mean that some users had a different experience with the assignments than others. This did result in much more of the tool being tested and in addition to using blocks as they were originally intended, they were also tested in more surprising ways, for example setting the colour of the LED strip to values above 255, the value rolled back to 0 again.

# 9 Conclusion

The goal of this graduation project was to find a solution that makes programming accessible for designers during a short workshop where there is not time to teach programming to every participant. This research has shown that when using the toolkit, the prototypes that can be created use a lot of different combinations of code, to support this the user must write code in some form. For minimizing the difficulties that come with writing code without prior knowledge visual programming can be an effective solution. Another finding was that there is not a lot of research into flow-based programming whilst it is used often in settings were learning to code is not important. For this reason, this style of programming was used during this project.

Overall, the created tool provides a solution for less experienced programmers (designers) to program more easily and quickly in the wearable technology workshop. Most users experienced that the flow-based tool significantly reduced the threshold for creating code. This enabled the less experienced users to write code independently, which allows them to participate in a workshop without excessive guidance. The programming tool was very effective at stimulating the user to experiment with the provided hardware and enabled them to change parts of their program, so they could be creative, but unfortunately not everything was possible. The users who had more experience with programming had to get used to the thought process of creating code using a flow-based language. In addition, frustrations occurred when the tool lacked a function a programmer was used to when programming. To improve on this the tool would need to be developed further focussing on making sure that it has a high ceiling.

During this research, it became clear that flow-based visual programming is something we can expect to see more of in the future, as hardware is becoming more readily available to end-users, there is a need for a simplified way of programming. Therefore, more research is needed into what makes a visual programming environment effective.

# 10 Discussion

### *The prototype*

During this research, a prototype visual programming tool was created, and while this tool was usable, it had a couple of issues. One of the bigger flaws that may have significantly reduced the usability of the tool is that when the user sets a value by connecting a node the slider for this value is still visible and adjustable. This was confusing for many users as they did not know exactly how the generated code would respond to the values set in this menu (the values are ignored if something is connected). During the evaluation it became clear that some users indicated the inspector as unclear, whether this was the main cause will have to be investigated further. In addition, there were some other issues with the user interface, but they will not all be discussed here, as many of these are minor issues but they do add up. A full list of the currently identified improvements can be found in Appendix H – Improvements to be made.

### *The evaluation of the prototype*

The evaluation of the prototype could be improved upon. The current sample size of ten participants is small, especially when you consider that of those ten users, only one had zero programming experience. To increase the validity of the results, the test should be repeated with a recommended twenty programmers and twenty non-programmers. This larger sample size would allow a more quantitative approach to the data, for example by having participants complete a questionnaire that uses a seven-point Likert scale. This would allow for a more accurate analysis, as the degree to which they agree with a statement can be measured more precisely than whether they agree, disagree, or neither.

### *Analysis of differences between visual programming types*

In addition to a more thorough test of a flow-based tool, it would be interesting to compare it to a block-based tool, because the decision to create a flow-based language is now based on the small amount of existing research on this subject.

# 11 References


[1] *Lymberis, A. (2003). "Smart wearables for remote health monitoring, from prevention to rehabilitation: current R&D, future challenges," 4th International IEEE EMBS Special Topic Conference on Information Technology Applications in Biomedicine, Birming.*

[2] A. Mader and W. Eggink, "A design process for creative technology.," in *Proceedings of the 16th International conference on engineering and product design education.*, Enschede, 2014.

[3] *http://wiki.edwindertien.nl/doku.php?id=workshops:wearables.*

[4] *Durling, David. (2004). "Horse or cart? Designer creativity and personality." http://durling.org/papers_files/EAD.pdf.*

[5] *https://opendata.cbs.nl/statline/#/CBS/nl/dataset/70962NED/table?fromstatweb.*

[6] *Niedlinger, Kristin & Dertien, Edwin. (2015). TEI 2015 Studio Interactive Inflatables: Amplifying Human Behaviors. 489-491. 10.1145/2677199.2683588. https://dl.acm.org/doi/pdf/10.1145/2677199.2683588.*

[7] *https://www.kobakant.at/DIY/?p=7394.*

[8] *Kazemitabaar, Majeed & McPeak, Jason & Jiao, Alexander & He, Liang & Outing, Thomas & Froehlich, Jon. (2017). MakerWear: A Tangible Approach to Interactive Wearable Creation for Children. 133-145. 10.1145/3025453.3025887. https://www.researchgate.net/publ.*

[9] *https://www.byor.nl/.*

[10] *Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. (2009). Scratch: programming for all. Communications of the ACM, 52(11), p. 60-67. DOI:10.1145/1592761.1.*

[11] *Weintrop, D. (2015). Minding the Gap Between Blocks-Based and Text-Based Programming (Abstract Only). In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15). Association for Computing Machinery, New York, NY, USA, 72.*

[12] *Booth, T., Stumpf, S. (2013). End-User Experiences of Visual and Textual Programming Environments for Arduino. In: Dittrich Y., Burnett M., Mørch A., Redmiles D. (eds) End-User Development. IS-EUD 2013. Lecture Notes in Computer Science, vol 7897. Spring.*

[13] *Sousa, T. B. (2012, November). Dataflow programming concept, languages and applications. In Doctoral Symposium on Informatics Engineering (Vol. 130)..*

[14] *Schaefer, R. (2011). On the limits of visual programming languages. SIGSOFT Softw. Eng. Notes 36, 2 (March 2011), p.7–8. DOI: 10.1145/1943371.1943373.*

[15] *"Scratch for Arduino", Accessed on: Apr. 10, 2020. [Online] Available: http://s4a.cat/.*

[16] *"Creating a program in TinkerCAD", Accessed on Apr. 16, 2020. [Online] Available: www.tinkercad.com.*

[17] *A. B. Pratomo and R. S. Perdana, "Arduviz, a visual programming IDE for arduino," 2017 International Conference on Data and Software Engineering (ICoDSE), Palembang, 2017, pp. 1-6..*

[18] *A. Rajalakshmi and H. Shahnasser, "Internet of Things using Node-Red and alexa," 2017 17th International Symposium on Communications and Information Technologies (ISCIT), Cairns, QLD, 2017, pp. 1-4..*

[19] "Unity Visual Scripting," [Online]. Available: https://unity.com/features/unity-visual-scripting. [Accessed 14 June 2023].

[20] Unreal Engine, "Blueprints Visual Scripting," [Online]. Available: https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html. [Accessed 14 June 2023].

[21] Mathworks, "Simulink - Simulation and Model-Based Design - MATLAB," [Online]. Available: https://nl.mathworks.com/products/simulink.html. [Accessed 14 June 2023].

[22] Bar, M., & Neta, M. (2006). Humans Prefer Curved Visual Objects. Psychological Science, 17(8), 645-648. https://doi.org/10.1111/j.1467-9280.2006.01759.x.

[23] *Moscow. [Online]. Available: https://www.projectsmart.co.uk/moscow-method.php.*

[24] *'Arduino CLI', Accessed on: Dec. 13, 2020. [Online] Available: https://arduino.github.io/arduino-cli/latest/.*

[25] *https://www.sensoree.com/tei-2015/.*

# Appendices

## Appendix A – Example programs

### A1. The full table with possible prototypes

| Idea | Description | Requirements |
|---|---|---|
| Posture coach | Alert the user with sound or vibration if wrong posture is detected | Timer, conditional statement, threshold on flex sensor |
| Turn signals | Turn signal with led strips on back, activated by touch sensors | Threshold on sensor, timer, light animation, |
| Party shirt | Make your clothes light up with the rhythm of the music or heartrate, or a combination where the brightness depends on heartrate and the rhythm is based on the music. | Microphone filter frequencies, light animations, threshold, frequency analysis, FFT |
| Tail | Wiggle tail when excited. | Sweep the servo, detect change in gsr, average of gsr |
| Accident avoidance | Detect sound and nudge user in a different direction with servo | Threshold, sound recognition, comparison, servo movement |
| Fall detector | When the user falls out of bed, have an alarm turn on | Alarm sound, detect rapid change in acceleration, custom Algorithm to detect if it was a fall or just rolling over in bed |
| Car safety | Use heartrate and speed/acceleration to detect dangerous driving and alert the user/someone else | Speed calculation, detect large change in value, connect to external device |
| Flex shoes | Flex sensor in running shoes to obtain insight in running technique | Pattern detection, store values, use time in calculation |
| Soccer shoes | Accelerometer to measure the power at which a soccer ball is kicked | Accelerometer data, custom calculation (with weight of ball and other parameters), peak detection |
| Christmas sweater | A simple ugly sweater that lights up and plays music, the music will be paired with a LED animation | Switch statement, LED animations, button press, music |
| Face mask with touch alarm | Capacitive sensor in a face mask to detect when a user wants to touch it, with a vibration | Cap sense library, vibration alarm, threshold |

| | | |
|---|---|---|
| Bike gear change suggestion | Use GSR, heartrate and speed to tell the user to switch gears up or down for maximum biking efficiency or a more effective workout | Read values, compare to "perfect" values (array), calculate, output voice |
| Activity watch | Vibrate when a user sits still too long, then use accelerometer to detect exercise | Read values, timer, condition statement, exercise counter, vibration |
| Vented jacket | Use the GSR sensor to detect sweat, if the user is sweating open the shirt using the servo to increase cooling | Read values, threshold, servo movement |
| Automated sunglasses | When a user looks down for 5 seconds put sunglasses in front of eyes with a servo, if a user than clicks a button lift the sunglasses again | Conditional statement, timer, servo control |
| Night light | If it is nighttime and the gsr and heartrate sensor indicate stress, turn on the LEDs to provide more light for the user. | Conditional statement, threshold on gsr and heartrate, timer, array with night times for each time of the year |
| High five counter | Count the number of high fives received and use the LEDs to show this to the user. | Counter, filtering, LED states, detection algorithm |
| Keep me awake | Keep someone awake by vibrating if they fall asleep, if that does not wake the user make an increasingly annoying sound. Detectable with heartrate sensor and accelerometer (if a user does not move and has a low heartrate for a certain amount of time, he/she probably fell asleep) | Timer (s), tone control, vibration, conditional statement |
| Musical glove | Using flex devices on the back of the fingers, allow the user to make music | Tone control, loudness, read values |
| Equalizer shirt | Put LED strips in a shirt that react to sounds/music | FFT, lighting control |
| Starry night dress | A dress with LEDs embedded throughout making some nice twinkle effects, press a button to switch modes | Switch statement, LED animations, button press |
| Fitness monitor | Using the heart rate, gsr and accelerometer detect activity and calculate calories burned | Read values, store values, calculate |
| Measure fluid in knees | Use a capacitive sensor to detect excess liquid in the knees and alert the user | Capacitive sensing, compare to reference, threshold, alarm |

| | | |
|---|---|---|
| Lifting technique | With a flex sensor and accelerometer detect whether someone lifts with his back or with his knees, alert the user when wrong lifting technique is sensed | Read values, conditional statement, combine data of the sensors, alarm |
| Arthrosis sensor | Detect arthrosis with a flex sensor on the finger, if the value keeps fluctuating the user might have arthrosis | Read values, measure over time, detect rapid changes, |
| Epilepsy detection | Epilepsy detection with an accelerometer, measuring rapid movement during an epilepsy attack while sleeping, when detected set of an alarm or connect to an app | Read values, detect sudden changes, data over time, arrays of data, alarm, connect to external device |
| Animal repellent | Use the buzzer to create high frequency sounds, keeps animals away from you | Tone output, sweep tones |
| Pickpocket sensor | When a hand is detected in the pocket vibrate | Conditional statement |

## A2. Programming keywords and their occurrence in the possible prototypes

| **Keyword** | **Count** |
|---|---|
| Read values | 8* |
| Threshold | 8 |
| Timer | 6 |
| Conditional statement | 6 |
| Alarm | 5 |
| Custom calculation | 5 |
| Animation | 4 |
| Servo control | 4 |
| Vibration | 3 |
| Array | 3 |
| Tone generation | 3 |
| Switch statement | 2 |
| FFT | 2 |
| Filter | 2 |
| Counter | 2 |
| Algorithm | 2 |
| Average | 1 |

# Appendix B – Current language types

| **Block-based** | **Flow-based** | **Other** |
|---|---|---|
| Arduviz | Visuino | Flowcode |
| ArduBlock | XOD | |
| Scratch4Arduino | | |
| Tinkercad circuits | | |
| mBlock | | |
| Minibloq | | |
| ArduBlockly | | |
| Modkit | | |
| BlocklyDuino | | |

# Appendix C – The workshop

## C1. The assignment

# Wearable technology workshop

## Introduction

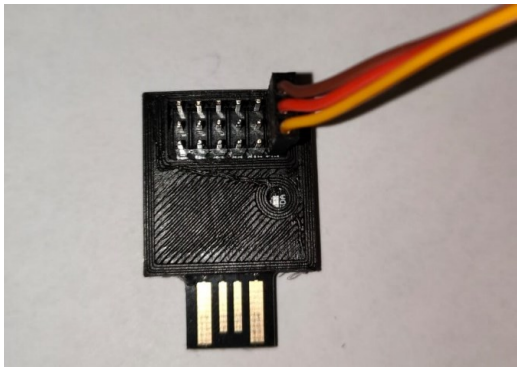
In this brief workshop, you will be completing three minor tasks using the programming tool. You will also receive a cheat sheet that illustrates how to link the sensors to the arduino and the data they measure/the values they output.

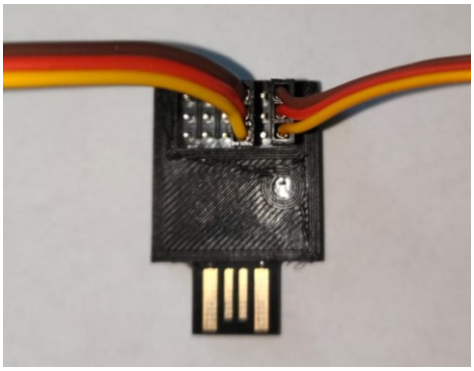
## Part 1- Getting started

As the first challenge in any Arduino tutorial usually involves making a LED blink, we are going to make something similar: a heartrate sensor that uses the LED strip to tell you your current heartrate.

### Connections:

First, we connect the LED strip to pin 11 (the far-right side) exactly as shown in the picture on the left, make sure the color of the wires matches:





Next, connect the heartrate sensor to pin 9 (the 4th pin from the left) as shown in the picture on the right.

Now that the sensor and the LED strip are connected it is time to explain what exactly we want the code to do. When the heartrate is bellow 40bpm we turn on 0 LEDs and after that for every 20bpm we turn one more LED on:

| Heartrate | LEDs that are on |
|---|---|
| 0 to 39 | 0 |
| 40 to 59 | 1 |
| 60 to 79 | 2 |
| 80 to 99 | 3 |
| 100 to 119 | 4 |
| 120 to 139 | 5 |
| 140 to 159 | 6 |
| 160 to 179 | 7 |
| 180 to 199 | 8 |

### Coding

To start drag the heartrate block from the menu on the left onto the canvas, as you now still have the block selected make sure to set the pin to 9 in the settings menu on the right. Now do the same for the LED block and set its pin to 11 and slide Brightness up to somewhere above 50.

Make sure there is some space between the two blocks by dragging them to opposing sides of the canvas. Now add the Map block in between the two blocks, and set the values as follows: Minimum input value to 40, Maximum input value to 200, Minimum output value to 1 and Maximum output value to 9. Now connect the blocks together by clicking on the white dot next to Current heartrate and dragging the line to Input of the Map block. Do the same for Result to Amount of LEDS.

### Exporting

Click on the upload button in the top left, after this create a folder on the desktop and click open. When this is done tell me and I will show you how to upload it to the Arduino.

## Part 2- Time to tinker

### Playing with the code

Now that you know how to connect and configure the blocks it is time to give it your own twist. First tweak something small. Some suggestions of things you might be able to do:

- Change the LEDs color to red if the heart rate is high.
- Lower the brightness if your average heart rate is low so you can relax without bright lights.

Now that you have changed something it is time to test, upload the code and see if it works.

### Adding some new hardware

To make it more unique add any of the other sensors or actuators to your existing build. Make sure to connect them to the proper pins (as stated in the reference sheet) and mind the colors. If you do not know what you want to add you can draw a card from the sensors or actuators stack and somehow work whatever you got your wearable. When you have done this make sure to upload your code and see the results.

### Swapping hardware

Now that you have explored some of the options with the heartrate sensor and the LED strip it is time to change something. For inspiration we will use cards, draw one, if it is a sensor replace the heart rate sensor, if it is an actuator replace the LED strip with it. First write down what you think you can do with the new hardware combination before you start making it.

Now adjust the code to match with your hardware swap, use the reference sheet to see what some of the blocks can do. If that does not work, you can ask for help if you need it. Time to upload and test your code again.

## Part 3- All the freedom

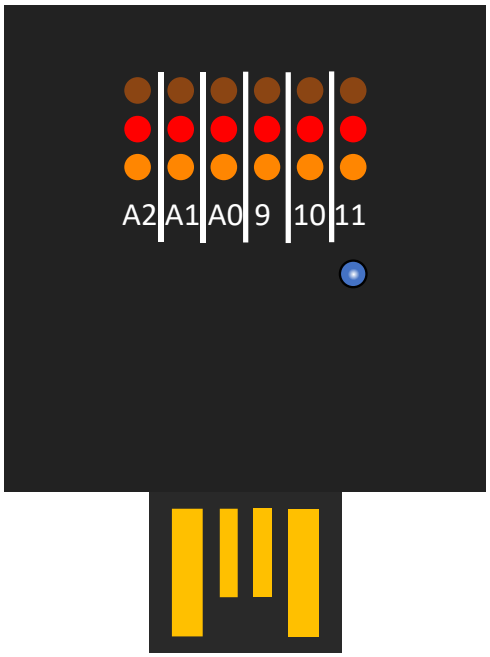
Make anything you want using items from the kit, hint if you do not have any inspiration feel free to draw a card or two. First write down or explain to someone what it is you thought of then build and code it.

# Info sheet

## Connecting sensors or actuators

All sensors should be connected with the brown wire facing away from the usb plug of the microcontroller. In the overview of the blocks that is included you can see which sensors/actuators can be connected were. If you are unsure also refer to the image of the controller below.

### Overview of the controller




## Programming

### Using the blocks

You can create blocks by dragging them from the menu on the left into the lighter grey working area. You can delete blocks by dragging them back into the menu on the left or by selecting them and pressing delete.

### Connecting blocks together

To connect the blocks together, drag from one of the white circles to another. Note you cannot connect an output to an output or an input to an input. To delete a connection, you can click on either of the end points.

### Changing settings of a block

To change the settings of a block simply click on the block and use the menu that appears on the right.

# Actuators

| Block | Function | Input | Compatible pins |
|---|---|---|---|
| Buzzer | Outputs a sound | Tone (hz) sets the frequency in hertz. Duration (ms) sets how long the tone should be played for in ms. | A2, A1, A0, 9, 10, 11 |
| LedStrip | Set the ledstrip | Brightness sets the brightness where 0 is off and 255 is the max. Red, green and blue, set the amount of certain color components where 0 is none and 255 is the maximum amount of this color. Amount sets the number of LEDs from 0 to 8. | A2, A1, A0, 9, 10, 11 |
| Servo | Moves from 0 to 180 degrees | Angle, input any number from 0 to 180 and watch the servo moves to that angle | A2, A1, A0, 9, 10, 11 |
| VibrationMotor | Vibrates | Vibrates at an intensity from 0 to 255 | 9,10,11 |

# Utility

| Block | Function | Input | Output |
|---|---|---|---|
| Comparison | Compare Input to the threshold | Any value and optionally a Threshold value | "Output value" if the statement is true |
| Split | Splits the signal | Any value | The same value as the Input twice |
| Math | Simple math including: +, -, *, /, % | One or two values | The result of value one and two with the operator used. |
| Map | Maps a value from one range to another: Meaning a value of "Minimum input value" would get mapped to "Minimum output value", a value of "Maximum input value" to "Maximum output value". Values between the minimum and maximum value get scaled to the new output range. | Any value | The mapped value |
| Timer | Creates a pulse with a set length every set amount of time. | Length of the signal in milliseconds, time the signal is off in milliseconds, the value when the signal is on, and the value when the signal is off. | Either the value when on or off based on the time |

# Sensors

| Block | Function | Output | Compatible pins |
|---|---|---|---|
| Accelerometer | Can be used to measure acceleration or orientation | Outputs a value of 0 to 1023 based on how fast it is moving and the angle of the sensor | A2, A1, A0 |
| DistanceSensor | Measure the distance to an object | Outputs the distance in cm, accurate from 10cm to 130cm | A2, A1, A0 |
| FlexSensor | Measure the amount it is bend | When it is not bending the value is roughly 512. Decreases when bend in one direction and increases when bend in the other direction. | A2, A1, A0 |
| GSRSensor | Measures the resistance of your skin | Without wearing the sensor upload some code for it and make sure it outputs 512 by adjusting the little potentiometer using a screwdriver. Then when wearing it changes based on your skins resistance. | A2, A1, A0 |
| HeartrateSensor | Measures your heartrate | The heartrate in beats per minute calculated over the last x (Number of measurements) beats, the average hearthrate in beats per minute over the last (Number of measurements times Number of averages) beats. | 9,10,11 |
| SoundSensor | Detect sound | Returns 1023 if the sound is over the thresshold, you can set the thresshold using the potentiometer on the sensor. Otherwise, it returns 0 | A2, A1, A0, 9, 10, 11 |
| TouchSensor | Detect touch | Returns 1023 if touched or 0 if it's not being touched | A2, A1, A0, 9, 10, 11 |

# Appendix D – Questionnaire

Questionnaire Wearable technology workshop

Knowledge about programming

1. Have you programmed before?
2. If you did program before, what language(s) have you used?
3. How comfortable are you with programming on a scale of 1 to 10? (Where 1 is not at all and 10 is extremely comfortable)
4. Are you familiar with visual programming? Visual programming is any way of creating code without typing it.
5. If you are familiar with visual programming which programs/languages, do you know that use it
6. If you ever used visual programming yourself, which languages/programs did you use?

The software used during the workshop

7. Did using the tool feel natural to you? (Where 1 is not at all and 10 is extremely natural)
8. Could you have used this tool without any guidance?
9. Did it take you long to get started with programming in the software?
10. In your experience was the programming in Part 1 - Getting started difficult?
11. In your experience was the programming in Part 2 - Time to tinker difficult?
12. In your experience was the programming in Part 3 - All the freedom difficult?
13. Did the tool help you with structuring your idea into code?
14. How much effort did it take to convert your idea into code?
15. Did the software allow you to create what you had in mind?
16. Does this software motivate you to experiment with the different components?
17. Do you have any ideas for things you could make with the components, that you have not made yet during this workshop?
18. Do you think that you can create other/new things using the tool you used today?
19. Are you confident that you can also prototype more complex projects using this tool?
20. How likely are you to use the tool again for making quick prototypes? (1 to 5 stars)

Questions for people that programmed before

21. Did the tool save you time compared to normal coding?
22. Did the tool make it more difficult to create something that you would already know how to do in code?

Optional remarks

23. If you have any optional remarks or improvements that you would like to make you can put them here.

# Appendix E – Information document

## Smart wearable technology software test information

Researcher: Kevin Smid

Date of writing: 17-1-2023

### The research:

My research is in ways to make programming more accessible for anyone that wants to participate in a workshop. To determine if the tool that was developed is a possible solution it will be compared against a different commonly used tool.

### What is asked of me?

- Before starting the user test, you will be asked to fill in a questionnaire regarding your coding proficiency, this so that we can determine which part of the target group you are in.
- After this you will get a brief overview followed by a assignment using one of the two tools that are being compared. It does not matter if you can not complete the assignment, and don't be afraid to write down any remarks regarding the tools. Depending on the test this will be followed by a secondary assignment in the other tool.
- Lastly, we will finish with a questionnaire asking your opinion on the tool(s)
- All in all, this will take between 20-40min.

### Data usage

All data collected will be filled in by you and stored without any personal data. In the report this data may be used as a quote combined with an indication of your coding level. The data will only be accessible in full (anonymised) to me (Kevin Smid) and will be stored until the end of the research.

If at any point you have any questions, feel free to ask them. If at any point for any reason you wish to withdraw/leave this study, you are able to without the need for explanation.

# Appendix F – Consent form

## Consent Form for Smart wearable technology tool

**YOU WILL BE GIVEN A COPY OF THIS INFORMED CONSENT FORM**

| *Please tick the appropriate boxes* | Yes | No |
|---|---|---|
| **Taking part in the study** | | |
| I have read and understood the study information dated 17/01/2023 or it has been read to me. I have been able to ask questions about the study and my questions have been answered to my satisfaction. | ○ | ○ |
| I consent voluntarily to be a participant in this study and understand that I can refuse to answer questions and I can withdraw from the study at any time, without having to give a reason. | ○ | ○ |
| I understand that taking part in the study involves the participant to fill in a questionnaire after using the prototypes. | ○ | ○ |
| **Use of the information in the study** | | |
| I understand that information I provide will be used for a bachelor thesis for the study Creative Technology. | ○ | ○ |
| I understand that personal information collected about me that can identify me, such as my name will not be shared beyond the study team. | ○ | ○ |
| I agree that my information can be quoted anonymised in research outputs. | ○ | ○ |

**Signatures**

| _____ | _____ | _____ |
|---|---|---|
| Name of participant | Signature | Date |

I have accurately read out the information sheet to the potential participant and, to the best of my ability, ensured that the participant understands to what they are freely consenting.

| Kevin Smid | _____ | _____ |
|---|---|---|
| Researcher name | Signature | Date |

**Study contact details for further information:** ***Kevin Smid, k.smid-1@student.utwente.nl***

**Contact Information for Questions about Your Rights as a Research Participant**

If you have questions about your rights as a research participant, or wish to obtain information, ask questions, or discuss any concerns about this study with someone other than the researcher(s), please contact the Secretary of the Ethics Committee Information & Computer Science: ethicscommittee-CIS@utwente.nl

UNIVERSITY OF TWENTE.

# Appendix G – Results

| ID | Programming experience | Used programming languages | Programming comfortability | Familiarity visual programming | Visual programming known | Visual programming used |
|---|---|---|---|---|---|---|
| A | Yes | Arduino, C, Processing, C#, Python, R | 7 | Yes | Blockly | blockly, lego mindstorms |
| B | Yes | Matlab | 3 | Yes | Simulink | Simulink |
| C | Yes | Processing, Arduino, Python, C#, C++, Dart, Html, Css, Sql | 8 | Yes | Unity Visual Scripting, Unity ShaderGraph, Unity VFX Graph, Blender Nodes, blockly | Unity Visual Scripting, Unity ShaderGraph, Unity VFX Graph, Blender Nodes, blockly |
| D | Yes | Python, php, JavaScript, arduino, processing, html/css | 7 | Yes | blockly, lego mindstorms | blockly, lego mindstorms |
| E | Yes | Processing, arduino, c++, c#, python | 8 | Yes | Blockly, unity visual scripting | Blockly |
| F | Yes | c++, python, Arduino | 4 | Yes | Blender, Rhino/Grasshopper, Unity | Blender, Rhino/Grasshopper |
| G | Yes | Arduino, Processing (Java based), Python | 6 | Yes | | |
| H | No (skip to question 4 | | 0 | No | | |
| I | Yes | Python, R, Java | 6 | Yes | SPSS (not recommended) | |
| J | Yes | Processing, Arduino, Python, Java | 6 | No | | |

| ID | Degree feeling natural to use | Usage without any guidance 1 | Usage without any guidance 2 |
|---|---|---|---|
| A | 7 | Yes | |
| B | 6 | Yes | but then it would have taken a little longer and I would have made mistakes |
| C | 7 | Yes | |
| D | 8 | Yes | |
| E | 7 | Probably | With enough time, probably. The inspector window was the least obvious |
| F | 7 | Yes | |
| G | 9 | Yes | but sometimes for people with 0 knowledge, the values of 0-1023 and 0-255 can be confusing, especially when mixing them up with the operators (blue blocks) |
| H | 7 | Yes | would just have taken a lot of time, trial, and error on what would do what. |
| I | 6 | No | because I was over-complicating my approach. Though I may have figured it out in more time and the manual without personal guidance. |
| J | 8 | Yes | but some tips were useful especially in understanding it a little faster |

| ID | Long time required getting started 1 | Long time required getting started 2 |
|---|---|---|
| A | No | null |
| B | A while | It always takes me a while before I understand how a program works. That was also the case with this program. |
| C | No | null |
| D | No | null |
| E | No | the logic is simple, but some functions like the split are less logical. |
| F | No | it was similar to other visual programming languages I have used |
| G | No | null |
| H | I think so | the start was a lot of reading and then checking if I read it correctly. |
| I | No | |
| J | No | |

| ID | Experienced difficulty part 1 getting started 1 | Experienced difficulty part 1 getting started 2 |
|---|---|---|
| A | No | null |
| B | No, not at all. | null |
| C | just the math | null |
| D | No | null |
| E | It was harder than the others | as I had to get familiar with the software and the Arduino. |
| F | No | null |
| G | No | null |
| H | No | just a lot of double checking. |
| I | No | and I was not familiar with any hardware |
| J | No | |

| ID | Experienced difficulty part 2 time to tinker 1 | Experienced difficulty part 2 time to tinker 2 |
|---|---|---|
| A | No | null |
| B | Somewhat difficult. | Understanding 'map' took a while. |
| C | No | null |
| D | Medium | null |
| E | Much more straightforward | although I did forget to set the inspector settings. |
| F | No | null |
| G | the very first moment a bit nervous | like oh no I have to think now but then it was all very easy and intuitive. |
| H | More trial and error and a case of experimenting rather than having it correct immediately. | null |
| I | No | I did notice that my approach was complicated (where I blocked or transferred a signal). So rather than the software being complicated, I was not familiar with the most optimal way of thinking through what I wanted to implement. |
| J | A little more | but only at the beginning, then I better understood how it all works |

| ID | Experienced difficulty part 3 all the freedom 1 | Experienced difficulty part 3 all the freedom 2 |
|---|---|---|
| A | No | null |
| B | I noticed that after part 2 I already figured it out a bit, | which makes you feel more comfortable when adding new things. |
| C | Not really | null |
| D | Challenging | null |
| E | Even easier | only hard to think of an idea |
| F | Yes | I gave myself a significant challenge and certain quality of life tools were missing, such as an \|absolute\| function and logic statements |
| G | No | especially not after learning the system in part2 |
| H | Nope | because I made a plan fitting to my (not so) high skill level. |
| I | No | I did not make a very effective intruder alarm though :D |
| J | No | because most of it already became clear in parts 1 and 2 |

| ID | Helpfulness structuring idea into code 1 | Helpfulness structuring idea into code 2 |
|---|---|---|
| A | No | null |
| B | Yes | null |
| C | Yes | null |
| D | Yes it did | null |
| E | Yes | as the tools were limited, that makes brainstorming easier. Also, the tool is straightforward and required less thinking than if I were to write the code myself. |
| F | Yes | the visual programming approach was much more intuitive than writing code. |
| G | Yes! | it is very handy to keep the overview |
| H | Yes | if I had to do this in writing I would graduate sooner than finishing my idea. |
| I | Yes and no. | For the hardware part (for which I have no experience) the tool helped a lot. For the comparisons elements (especially) I was sometimes too used to thinking in the way I already know from my prior programming experiences. |
| J | Yes | |

| ID | Required effort converting idea into code |
|---|---|
| A | it is limiting my style |
| B | Not really very much. It is easy that you see the steps happening in the boxes |
| C | Not that much, if I wrote it out on paper first |
| D | Yes but I chose a challenging challenge |
| E | A little bit but not much, the same logical problem solving is required as coding, just less of it. |
| F | several clicks. No effort at all |
| G | on a scale from 1-10 I would say 2. very easy but sometimes have to think twice for the blue blocks and their values. |
| H | Not that much, as it did not really require any knowledge as to how coding works. |
| I | Not too much with a little guidance |
| J | Surprisingly little, I usually struggle with coding in Arduino |

| ID | Enabling converting idea into code 1 | Enabling converting idea into code 2 |
|---|---|---|
| A | No | null |
| B | Yes | null |
| C | Yes | null |
| D | Almost | null |
| E | Yes | although at first it seemed like there was no function for what I wanted, but that could be solved by using other functions. |
| F | Yes | the only limiting factor was a lack of clear vision from myself |
| G | Yes | null |
| H | Close to | but that was also because I did not finetune it a few times. |
| I | Yes | but for the intruder alarm I noticed when I finished my idea that it is not a good alarm, but that's a design error |
| J | Yes | |

| ID | Motivating to experiment with the components 1 | Motivating to experiment with the components 2 |
|---|---|---|
| A | Yes | saves time with collecting al libraries and figuring them out |
| B | Yes | It is nice to see that with a number of components you can make many different things by combining the components. |
| C | Yes | because easily hotswappable |
| D | Yes | it seems very easy to add and remove sensors/actuators |
| E | Yes I think so | its much easier to test out crazy ideas and just play around with the 'code' |
| F | yes | it was very easy to swap out and made me curious to try other sensors and actuators |
| G | Yes! | since it was so easy to use it was very rewarding fast to make a new component work, so I wanted to keep adding new things. |
| H | Yes | the buzzer was very fun. |
| I | It does! | As I said, I do not have prior experience with the hardware components, but I feel that I could work with them nevertheless |
| J | Yes | it was fun that there were so many different components |

| ID | Ideas for using the components 1 | Ideas for using the components 2 |
|---|---|---|
| A | Led strip white with neutral heartbeat and blue with low read with high | null |
| B | No | null |
| C | Yes pretty much everything | null |
| D | Not right now | but it would be easy to come up with new ideas |
| E | The ones that I thought of I made. | But I can image you can do most stuff with the provided tools |
| F | The ultimate rave suit | a shirt with embedded led strip audio visualisers that change colour depending on the acceleration and bending of your limbs |
| G | Projects for the uni modules etc. It would be very useful. | null |
| H | A sensor to see how far you are from something that you can not see. (Something like a parking sensor) | null |
| I | A working intruder alarm | null |
| J | Not by heart but there are many possibilities | null |

| ID | Tool enabling to create other things 1 | Tool enabling to create other things 2 | Confidence in tool prototyping more complex projects 1 | Confidence in tool prototyping more complex projects 2 |
|---|---|---|---|---|
| A | No | then I would use actual code. But good to quickly prototype | No | null |
| B | Yes | null | I think so | It will take a little longer, but it will eventually work. |
| C | definitely | null | Yes | null |
| D | Yes | null | Sometimes the blocks are limiting | as I would have more control coding stuff myself. |
| E | Yes | I think plenty is possible. | Yes | although it might get very messy and for larger projects it might become more cluttered than code would. |
| F | Yes | null | Yes | null |
| G | Yes! | The programming language would not be a restriction to let my creativity flow | Yes | this would definitely help with me making concepts of things faster and easier since it is ALWAYS a hassle to get my new Arduino components working (at least for me). |
| H | Yes | I think that would be possible to make my previously mentioned idea. | Yes | but that is also because I did not make that difficult things as I am not that experienced in the area of coding and wanted to start of easy. |
| I | Probably! | | Yes | but some guidance (tutorials etc.) would be appreciated |
| J | Yes definitely | | Yes | but I am not sure how you can create dependencies and loops between parts |

| ID | Likeliness using tool again for quick prototyping | Saving time compared to normal coding 1 | Saving time compared to normal coding 2 |
|---|---|---|---|
| A | 4 | Yes | null |
| B | 5 | Yes | null |
| C | 4 | Yes | because I have not used arduino in a while |
| D | 4 | For the easy stuff | yes. |
| E | 4 | For simple solutions | yes. |
| F | 5 | Yes | it prevented the many syntax errors and little mistakes I usually make when coding. It also made it easier to change orders of operations, change parts and keep a clear overview of the program |
| G | 5 | yes! a lot! I would say that this might be 10x faster. | null |
| H | 4 | null | null |
| I | 4 | Can't compare | because I haven't coded comparable projects in normal coding |
| J | 4 | Yes! | |

| ID | Something was more difficult than coding 1 | Something was more difficult than coding 2 | Any optional remarks |
|---|---|---|---|
| A | Yes | null | use wifi and ota updates. no cables |
| B | No | null | null |
| C | Yes, basic math | but in general not | It was not always clear which direction I had to put in the wires. Small usability improvements would have been nice. (disabled input fields, quick snap lines etc) |
| D | Yes | there was something that I knew how to do in code but did not manage in the tool. | null |
| E | I think it is equal. | null | Split block needs more outputs. I would like multiple lines from one output. Curves :). And more logical UX in the inspector. |
| F | Yes | as mentioned before, some useful tools such as logic gates and more advanced math functions like absolute or sine waves were missing. Also, the ability to make classes and functions was missing. | <3 |
| G | No | null | more components to play with ;) |
| H | null | null | The buzzer is great :) |
| I | No | haven't coded comparable projects before (involving hardware) | None |
| J | No | | Only the addition of making dependencies between parts or possible loops. |

# Appendix H – Improvements to be made

## H1. Functionality

- Automatic uploading
- Saving and loading the workspace
- Serial monitor
- Serial plotter
- Custom block creation
- More math options
- Control logic for example AND and OR
- Custom methods where the user can type
- Calibration
- Split block multiple outputs
- Help button that shows information

## H2. UI

- Hide/lock sliders when the value is not used
- Overview of the pins that are in use
- Double click on text box should select all text
- Open a drop-down menu when clicked instead of having to use the button
- Save last file location
- Grouping blocks into a single block
- Zoom on the canvas
- Make blocks autofit based on text, leading to smaller blocks
- Quick snap lines
- Bendable lines/curves

## H3. Hardware

- A microphone instead of the current sound sensor
- Pin labels printed on the Beetle
- Wires that fit in one direction
- USB extension cable